Partially observed Markov processes with spatial structure via the R package spatPomp

Kidus Asfaw
Univ. of Michigan

Joonha Park
Univ. of Kansas

Aaron A. King
Univ. of Michigan

Edward Ionides
Univ. of Michigan

Abstract

We introduce a computational framework for modeling and statistical inference on high-dimensional dynamic systems. Our primary motivation is the investigation of metapopulation dynamics arising from a collection of spatially distributed, interacting biological populations. To make progress on this goal, we embed it in a more general problem: inference for a collection of interacting partially observed nonlinear non-Gaussian stochastic processes. Each process has a unit label, and for spatiotemporal models the units correspond to spatial locations. The dynamic state for each unit may be discrete or continuous, scalar or vector valued. In metapopulation applications, the state can represent a structured population or the abundances of a collection of species at a single location. We consider models where the collection of states has a Markov property. A sequence of noisy measurements is made on each unit, resulting in a collection of time series. A model of this form is called a spatiotemporal partially observed Markov process (SpatPOMP). The R package **spatPomp** provides an environment for implementing SpatPOMP models, analyzing data using existing methods, and developing new inference approaches. Our presentation of **spatPomp** therefore provides a review of various methodologies in a unifying notational framework. We demonstrate the package on a simple Gaussian system and on a nontrivial epidemiological model for measles transmission within and between cities. We show how to construct user-specified SpatPOMP models within **spatPomp**.

Keywords: Markov processes, hidden Markov model, state space model, stochastic dynamical system, maximum likelihood, plug-and-play, spatiotemporal data, mechanistic model, sequential Monte Carlo, R.

1. Introduction

A spatiotemporal partially observed Markov process (SpatPOMP) model consists of incomplete and noisy measurements of a latent Markov process having spatial as well as temporal structure. A SpatPOMP model is a special case of a vector-valued partially observed Markov process (POMP) where the latent states and the measurements are indexed by a collection of spatial locations known as units. Many biological, social and physical systems have the spatiotemporal structure, dynamic stochasticity and imperfect observability that characterize SpatPOMP models. The spatial structure of SpatPOMPs adds complexity to the problems of likelihood estimation, parameter inference and model selection for nonlinear and non-Gaussian systems. The objective of the **spatPomp** package is to facilitate model development and data analysis in the context of the general class of SpatPOMP models, enabling scientists to separate the scientific task of model development from the statistical task of providing inference tools.

Modeling and inference for spatiotemporal dynamics has long been considered a central challenge in ecology and epidemiology. Bjørnstad and Grenfell (2001) identified six challenges of data analysis for ecological and epidemiological dynamics: (i) combining measurement noise and process noise; (ii) including covariates in mechanistically plausible ways; (iii) continuous time models; (iv) modeling and estimating interactions in coupled systems; (v) dealing with unobserved variables; (vi) spatiotemporal models. Challenges (i) through (v) require nonlinear time series analysis methodology, and this has been successfully addressed over the past two decades via the framework of POMP models. Software packages such as **pomp** (King *et al.* 2016), **nimble** (Michaud *et al.* 2021), **LiBBi** (Murray 2015) and **mcstate** (FitzJohn *et al.* 2020) nowadays provide routine access to widely applicable modern inference algorithms for POMP models, as well as platforms for sharing models and data analysis workflows. Monte Carlo methods widely used for fitting nonlinear stochastic models to time series do not scale well for high-dimensional systems and so are not practically applicable to SpatPOMP models. Thus, challenge (vi) requires state-of-the-art algorithms which have favorable scalability properties.

The **spatPomp** package brings together general purpose methods for carrying out Monte Carlo statistical inference that meet all the requirements (i) through (vi). For this purpose, **spatPomp** provides an abstract representation for specifying SpatPOMP models. This ensures that SpatPOMP models formulated with the package can be investigated using a range of methods, and that new methods can be readily tested on a range of models. In its current form, **spatPomp** is appropriate for data analysis with a moderate number of spatial units (say, 100) having nonlinear and non-Gaussian dynamics. In particular, **spatPomp** is not targeted at very large spatiotemporal systems such as those that arise in geophysical data assimilation (Anderson *et al.* 2009) though some of the tools developed in that context can be applied to smaller models and are provided by the package. Spatiotemporal systems with Gaussian dynamics can be investigated with **spatPomp**, but a variety of alternative methods and software are available in this case (Wikle *et al.* 2019; Sigrist *et al.* 2015; Cappello *et al.* 2020).

The **spatPomp** package builds on the **pomp** package described by King *et al.* (2016). Mathematically, a SpatPOMP model is also a POMP model, and this property is reflected in the object-oriented design of **spatPomp**. The package is implemented using S4 classes (Chambers 1998; Genolini 2008; Wickham 2019) and the basic class '`spatPomp`' extends the class '`pomp`' provided by **pomp**. This allows new methods to be checked against extensively tested methods in low-dimensional settings for which POMP algorithms are applicable. However, standard Monte Carlo statistical inference methods for nonlinear POMP models suffer from a *curse of dimensionality* (Bengtsson *et al.* 2008). Extensions of these methods for situations with more than a few units must, therefore, take advantage of the special structure of SpatPOMP models. Figure 1 illustrates the use case of the **spatPomp** package relative to the **pomp** package and methods that use Gaussian approximations to target models with massive dimensionality. Highly scalable methods, such as the Kalman filter and ensemble Kalman filter, entail approximations that may be inappropriate for nonlinear, non-Gaussian, count-valued models arising in metapopulation analysis. Using **spatPomp**, one can assess whether alternative methods are superior for the task at hand.

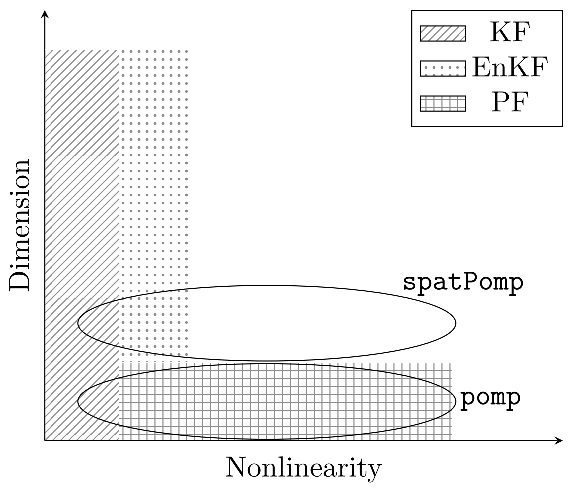

Figure 1 – The use case for the **spatPomp** package. For statistical inference of models that are approximately linear and Gaussian, the Kalman Filter (KF) is an appropriate method. If the nonlinearity in the problem increases moderately but the dimension of the problem is very large (e.g. geophysical models), the ensemble Kalman Filter (EnKF) is useful. In low-dimensional but very nonlinear settings, the particle filter (PF) is widely applicable and the **pomp** package targets such problems. The **spatPomp** package and the methods implemented in it are intended for statistical inference for nonlinear models that are of moderate dimension. The nonlinearity in these models (e.g. epidemiological models) is problematic for Gaussian approximations and the dimensionality is large enough to make the particle filter unstable.

A SpatPOMP model is characterized by the transition density for the latent Markov process and unit-specific measurement densities. Once these elements are specified, calculating and simulating from all joint and conditional densities are well defined operations. However, different statistical methods vary in the operations they require. Some methods require only simulation from the transition density whereas others require evaluation of this density. Some methods avoid working with the model directly, replacing it by an approximation, such as a linearization. For a given model, some operations may be considerably easier to implement and so it is useful to classify inference methods according to the operations on which they depend. In particular, an algorithm is said to be *plug-and-play* if it utilizes simulation of the latent process but not evaluation of transition densities (Bretó *et al.* 2009; He *et al.* 2010). The arguments for and against plug-and-play methodology for SpatPOMP models are essentially the same as for POMP models (He *et al.* 2010; King *et al.* 2016). Simulators are relatively easy to implement for many SpatPOMP models; plug-and-play methodology facilitates the investigation of a variety of models that may be scientifically interesting but mathematically inconvenient. Modern plug-and-play algorithms can provide statistically efficient, likelihood-based or Bayesian inference. The computational cost of plug-and-play methods may be considerable, due to the large number of simulations involved. Nevertheless, the utility of plug-and-play methods for POMP models has been amply demonstrated in scientific applications. In particular, plug-and-play methods implemented using **pomp** have proved capable for various scientific investigations (e.g., King *et al.* 2008; Bhadra *et al.* 2011; Shrestha *et al.* 2011, 2013; Earn *et al.* 2012; Roy *et al.* 2013; Blackwood *et al.* 2013a,b; He *et al.* 2013; Bretó 2014; Blake *et al.* 2014; Martinez-Bakker *et al.* 2015; Bakker *et al.* 2016;

Becker *et al.* 2016; Buhnerkempe *et al.* 2017; Ranjeva *et al.* 2017; Marino *et al.* 2019; Pons-Salort and Grassly 2018; Becker *et al.* 2019; Kain *et al.* 2021; Stocks *et al.* 2020). Although the **spatPomp** package framework permits implementation of methods with and without the plug-and-play property, development of the package to date has emphasized plug-and-play methods.

The remainder of this paper is organized as follows. Section 2 defines mathematical notation for SpatPOMP models and relates this to their representation as objects of class '`spatPomp`' in the **spatPomp** package. Section 3 introduces likelihood evaluation via several spatiotemporal filtering methods. Section 4 describes parameter estimation algorithms which build upon these filtering techniques. Section 5 constructs a simple linear Gaussian SpatPOMP model and uses this example to illustrate statistical inference. Section 6 presents the construction of spatially structured compartment models for population dynamics, in the context of coupled measles dynamics in UK cities; this demonstrates the kind of nonlinear stochastic system primarily motivating the development of **spatPomp**. Finally, Section 7 discusses extensions and applications of **spatPomp**.

2. SpatPOMP models and their representation in spatPomp

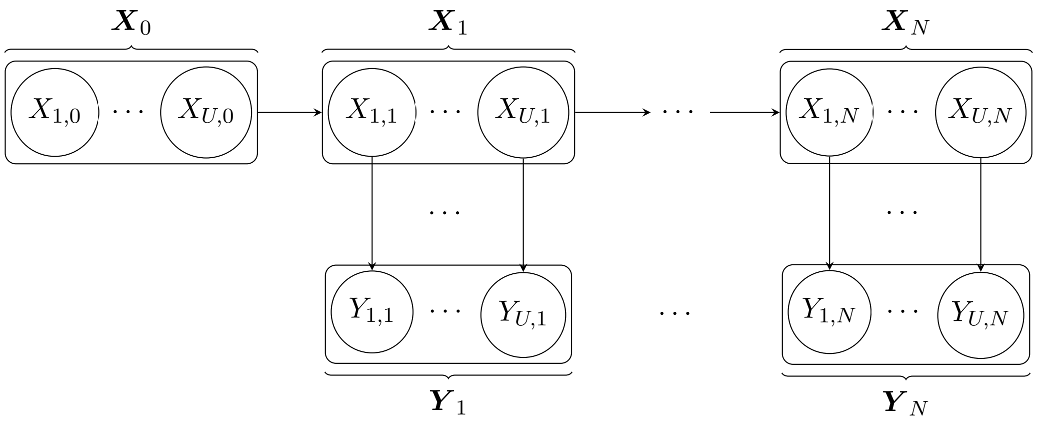

Figure 2 – The structure of a spatiotemporal partially observed Markov process (SpatPOMP) model. The latent dynamic process is $\{\boldsymbol{X}(t), t_0 \leq t \leq t_N\}$. At observation times t_n, the value of the latent process is denoted by $\boldsymbol{X}_n = (X_{1,n}, \ldots, X_{U,n})$. The partial and noisy observations at this times are modeled by $\boldsymbol{Y}_n = (Y_{1,n}, \ldots, Y_{U,n})$.

We set up notation for SpatPOMP models extending the POMP notation of King *et al.* (2016). A diagrammatic representation is given in Figure 2. Suppose there are U units labeled $\{1, 2, \ldots, U\}$, which we write as $1\!:\!U$. Let $t_1 < t_2 < \cdots < t_N$ be a collection of times at which measurements are recorded on one or more units, and let t_0 be some time preceding t_1 at which we initialize our model. We observe a measurement $y^*_{u,n}$ on unit u at time t_n, where $y^*_{u,n}$ could take the value `NA` if no measurement was recorded. We postulate a latent stochastic process taking value $\boldsymbol{X}_n = (X_{1,n}, \ldots, X_{U,n})$ at time t_n, with boldface denoting a collection of random variables across units. The observation $y^*_{u,n}$ is modeled as a realization of an observable random variable $Y_{u,n}$, and we suppose that the collection of observable random variables are conditionally independent given the collection of latent random variables. The process $\boldsymbol{X}_{0:N}$ is required to have the Markov property, i.e., $\boldsymbol{X}_{0:n-1}$ and $\boldsymbol{X}_{n+1:N}$ are conditionally independent given $\boldsymbol{X}_n$. Optionally, there may be a continuous time process $\boldsymbol{X}(t)$ defined for $t_0 \leq t \leq t_N$ such that $\boldsymbol{X}_n = \boldsymbol{X}(t_n)$.

Let $f_{\boldsymbol{X}_{0:N},\boldsymbol{Y}_{1:N}}(\boldsymbol{x}_{0:N}, \boldsymbol{y}_{1:N}; \theta)$ be the joint density of $X_{1:U,0:N}$ and $Y_{1:U,1:N}$ evaluated at $x_{1:U,0:N}$ and $y_{1:U,1:N}$, depending on an unknown parameter vector, θ. We do not distinguish between continuous and discrete spaces for the latent and observation processes, so the term *density* encompasses probability mass functions. The SpatPOMP structure permits a factorization of the joint density in terms of the initial density, $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0; \theta)$, the transition density, $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \,|\, \boldsymbol{x}_{n-1}\,; \theta)$, and the unit measurement density, $f_{Y_{u,n}|X_{u,n}}(y_{u,n}|x_{u,n}\,; \theta)$, given by

$$f_{\boldsymbol{X}_{0:N},\boldsymbol{Y}_{1:N}}(\boldsymbol{x}_{0:N}, \boldsymbol{y}_{1:N}; \theta) = f_{\boldsymbol{X}_0}(\boldsymbol{x}_0; \theta) \prod_{n=1}^{N} f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}; \theta) \prod_{u=1}^{U} f_{Y_{u,n}|X_{u,n}}(y_{u,n}|x_{u,n}\,; \theta).$$

This notation allows $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}$ and $f_{Y_{u,n}|X_{u,n}}$ to depend on n and u, thereby permitting models for temporally and spatially inhomogeneous systems.

2.1. Implementation of SpatPOMP models

A SpatPOMP model is represented in **spatPomp** by an object of class ‘`spatPomp`’. Slots in this object encode the components of the SpatPOMP model, and can be filled or changed using the constructor function `spatPomp()` and various other convenience functions. Methods for the class ‘`spatPomp`’ (i.e., functions defined in the package which take a class ‘`spatPomp`’ object as their first argument) use these components to carry out computations on the model. Table 1 lists elementary methods for a class ‘`spatPomp`’ object, and their translations into mathematical notation.

Class ‘`spatPomp`’ inherits from the class ‘`pomp`’ defined by the **pomp** package. In particular, **spatPomp** extends **pomp** by the addition of unit-level specification of the measurement model. This reflects the modeling assumption that measurements are carried out independently in both space and time, conditional on the current value of the latent process. There are five unit-level functionalities of class ‘`spatPomp`’ objects: `dunit_measure`, `runit_measure`, `eunit_measure`, `vunit_measure` and `munit_measure`. These model components are specified by the user via an argument to the `spatPomp()` constructor function of the same name.

All the model components of a class ‘`spatPomp`’ object are listed in Table Table 1. It is not necessary to supply every component—only those that are required to run an algorithm of interest. For example, the functions `eunit_measure` and `vunit_measure`, calculating the expectation and variance of the measurement model, are used by the ensemble Kalman filter (EnKF, Section 3.2) and iterated EnKF (Section 4.2). The function `munit_measure` returns a parameter vector corresponding to given mean and variance, used by one of the options for a guided intermediate resampling filter (GIRF, Section 3.1) and iterated GIRF (Section 4.1).

2.2. Examples included in the package

Examples of class ‘`spatPomp`’ objects are constructed by the functions `bm()`, `lorenz()` and `measles()`. These create class ‘`spatPomp`’ models with user-specified dimensions for a correlated Brownian motion model, the Lorenz-96 atmospheric model (Lorenz 1996), and a spatiotemporal measles SEIR model, respectively. Under the hood, all these examples are built by setting up a call to `spatPomp()`, and the source code for these functions serves as a

Method	Argument to `spatPomp()`	Mathematical terminology
`dunit_measure`	`dunit_measure`	Evaluate $f_{Y_{u,n}\vert X_{u,n}}(y_{u,n} \mid x_{u,n}\,;\theta)$
`runit_measure`	`runit_measure`	Simulate from $f_{Y_{u,n}\vert X_{u,n}}(y_{u,n} \mid x_{u,n}\,;\theta)$
`eunit_measure`	`eunit_measure`	Evaluate $\mathrm{e}_{u,n}(x,\theta) = \mathsf{E}[Y_{u,n} \mid X_{u,n} = x\,;\theta]$
`vunit_measure`	`vunit_measure`	Evaluate $\mathrm{v}_{u,n}(x,\theta) = \mathrm{Var}[Y_{u,n} \mid X_{u,n} = x\,;\theta]$
`munit_measure`	`munit_measure`	$\mathrm{m}_{u,n}(x,V,\theta) = \boldsymbol{\psi}$ if $\mathrm{v}_{u,n}(x,\boldsymbol{\psi}) = V$, $\mathrm{e}_{u,n}(x,\boldsymbol{\psi}) = \mathrm{e}_{u,n}(x,\theta)$
`rprocess`	`rprocess`	Simulate from $f_{\boldsymbol{X}_n\vert\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}\,;\theta)$
`dprocess`	`dprocess`	Evaluate $f_{\boldsymbol{X}_n\vert\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}\,;\theta)$
`rmeasure`	`rmeasure`	Simulate from $f_{\boldsymbol{Y}_n\vert\boldsymbol{X}_n}(\boldsymbol{y}_n \mid \boldsymbol{x}_n\,;\theta)$
`dmeasure`	`dmeasure`	Evaluate $f_{\boldsymbol{Y}_n\vert\boldsymbol{X}_n}(\boldsymbol{y}_n \mid \boldsymbol{x}_n\,;\theta)$
`rprior`	`rprior`	Simulate from the prior distribution $\pi(\theta)$
`dprior`	`dprior`	Evaluate the prior density $\pi(\theta)$
`rinit`	`rinit`	Simulate from $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)$
`timezero`	`t0`	t_0
`time`	`times`	$t_{1:N}$
`obs`	`data`	$\boldsymbol{y}^*_{1:N}$
`states`	—	$\boldsymbol{x}_{0:N}$
`coef`	`params`	θ

Table 1 – Elementary methods for class ‘`spatPomp`’ objects, the argument used to assign them via the `spatPomp` constructor function, and their definition in mathematical notation.

template for writing user-specified models. In Section 6, we work through the construction of a scientifically motivated class ‘`spatPomp`’ object.

Our first `spatPomp` model is built by `bm10 <- bm(U = 10, N = 20)`, creating a simulation of $U = 10$ correlated Brownian motions each with $N = 20$ measurements. The correlation structure and other model details are discussed in Section 5. We can view the data using `plot(bm10)`, shown in Fig. 3. For customized plots using the many plotting options in R for class ‘`data.frame`’ objects, the data in `bm10` can be extracted using `as.data.frame(bm10)`. The accessor functions in Table 1 extract various components of `bm10` via `timezero(bm10)`, `time(bm10)`, `obs(bm10)`, `states(bm10)`, `coef(bm10)`. The internal representation of all the components of the object can be inspected via `spy(bm10)`.

2.3. Data and observation times

The only mandatory arguments to the `spatPomp()` constructor are `data`, `times`, `units` and `t0`. The `data` argument requests a class ‘`data.frame`’ object containing observations for each spatial unit at each observation time. Missing data for some or all units at each observation time can be coded as `NA`. When there is missing data, it is the user’s responsibility to assign a measurement model that describes a reasonable mechanism for missingness. The name of the `data` column containing observation times is supplied to the `times` argument; the name of the column containing the unit names is supplied to the `units` argument. The `t0` argument supplies the initial time from which the dynamic system is modeled, which should be no greater than the first observation time.

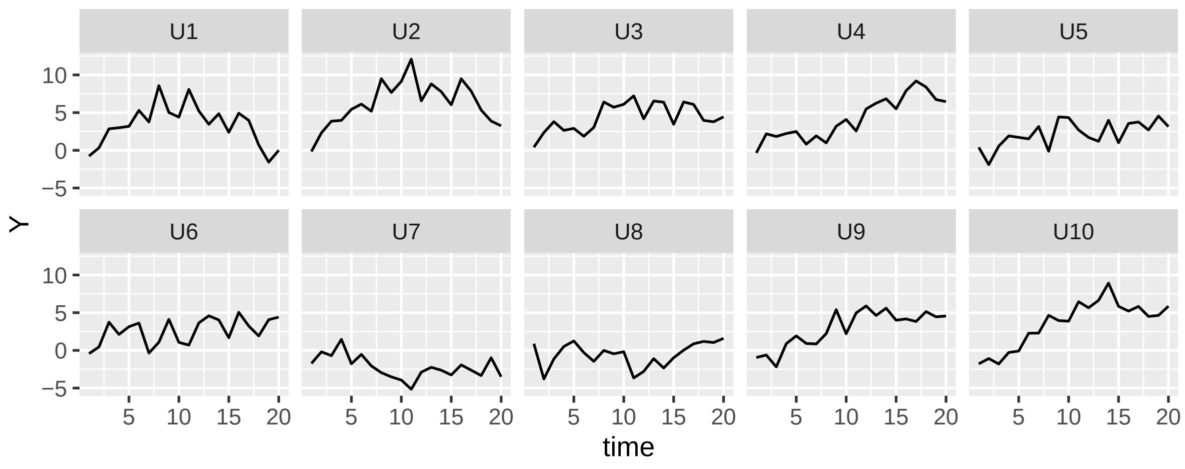

Figure 3 – Output of the `plot()` method on a class '`spatPomp`' object representing a simulation from a 10-dimensional correlated Brownian motions model with 20 observations that are one unit time apart.

We may also wish to add parameter values, latent state values, and some or all of the model components from Table 1. We need to define only those components necessary for operations we wish to carry out. In particular, plug-and-play methodology by definition never uses `dprocess`. An empty `dprocess` slot in a class '`spatPomp`' object is therefore acceptable unless a non-plug-and-play algorithm is attempted.

2.4. Initial conditions

The initial state of the latent process, $\boldsymbol{X}_0 = \boldsymbol{X}(t_0;\theta)$, is a draw from the initial distribution, $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)$. If the initial conditions are known, there is no dependence on θ. Alternatively, there may be components of the θ having the sole function of specifying $\boldsymbol{X}_0$. These components are called *initial value parameters* (IVPs). By contrast, parameters involved in the transition density or measurement density are called *regular parameters* (RPs). This gives rise to a decomposition of the parameter vector, $\theta = (\theta_{\mathrm{RP}}, \theta_{\mathrm{IVP}})$. We may specify $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta_{\mathrm{RP}}, \theta_{\mathrm{IVP}})$ to be a point mass at θ_{IVP}, in which case θ_{IVP} exactly corresponds to $\boldsymbol{X}_0$. The `bm10` model has this structure, and the initialization can be tested by `rinit(bm10)`. The measles model of Section 6.3 specifies $\boldsymbol{X}_0$ as a deterministic function of θ_{IVP} since it is convenient to describe latent states as counts and the corresponding IVPs as proportions.

2.5. Parameters

Many **spatPomp** methods require a named numeric vector to represent a parameter, θ. In addition to the initial value parameters introduced in Section 2.4, a parameter can be *unit-specific* or *shared*. A unit-specific parameter has a distinct value defined for each unit, and a shared parameter is one without that structure, so we can write $\theta = (\phi, \psi_{1:U})$. The unit methods in Table 1 require only ϕ and ψ_u when evaluated on a single unit, u. A shared/unit-specific structure can be combined with an RP/IVP decomposition to give

$$\theta = (\phi_{\mathrm{RP}}, \phi_{\mathrm{IVP}}, \psi_{\mathrm{RP},1:U}, \psi_{\mathrm{IVP},1:U}).$$

The `bm10` and measles examples are coded with unit-specific IVPs and shared RPs. The dimension of the parameter space can increase quickly with the number of unit-specific parameters. Shared parameters provide a more parsimonious description of the system, which is desirable when it is consistent with the data.

2.6. Covariates

Scientifically, one may be interested in the impact of a vector-valued covariate process, $\mathbf{Z}(t)$, on the latent dynamic system. Our modeling framework allows the transition density, $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}$, and the measurement density, $f_{\boldsymbol{Y}_n|\boldsymbol{X}_n}$, to depend arbitrarily on time, and this includes the possibility of dependence on one or more covariates. A covariate process is called *shared* if, at each time, it takes single value which influences all the units. A *unit-specific* covariate process, $\mathbf{Z}(t) = Z_{1:U}(t)$, has a value, $Z_u(t)$, for each unit, u. In **spatPomp**, covariate processes can be supplied as a class '`data.frame`' object to the `covar` argument of the `spatPomp()` constructor function. This `data.frame` requires a column for time, spatial unit, and each of the covariates. If any of the variables in the covariates `data.frame` is common among all units the user must supply the variable names as class '`character`' vectors to the `shared_covarnames` argument of the `spatPomp()` constructor function. All covariates not declared as shared are assumed to be unit-specific. **spatPomp** manages the task of presenting interpolated values of the covariates to the elementary model functions at the time they are called. An example implementing a SpatPOMP model with covariates is presented in Section 6.

2.7. Specifying model components using C snippets

The **spatPomp** function `spatPomp_Csnippet` extends the Csnippet facility in **pomp** which allows users to specify the model components in Table 1 via fragments of C code. The use of Csnippets permits computationally expensive calculations to take advantage of the performance of C. The Csnippets are compiled in a suitable environment by a call to `spatPomp()`, however, `spatPomp()` needs some help to determine which variables should be defined. In behavior inherited from **pomp**, the names of the parameters and latent variables must be supplied to `spatPomp` using the `paramnames` and `unit_statenames` arguments, and the names of observed variables and covariates are extracted from the supplied data. In **spatPomp**, unit-specific variable names can be supplied as needed via arguments to `spatPomp_Csnippet`. These can be used to specify the five `unit_measure` model components in Table 1 which specify properties of the spatially structured measurement model characteristic of a SpatPOMP. For a `unit_measure` Csnippet, automatically defined variables also include the number of units, `U`, and an integer `u` corresponding to a numeric unit from `0` to `U-1`.

A Csnippet can look similar to a domain-specific language. For example, the unit measurement density for the `bm10` example is simply

```
R> spatPomp_Csnippet("lik = dnorm(Y,X,tau,give_log);")
```

Here, `spatPomp` makes all the required variables available to the Csnippet: the unit state name variable, `X`; the unit measurement variable, `Y`; the parameter, `tau`; and a logical flag `give_log` indicating whether the desired output is on log scale, following a standard convention for the C interface to R distribution functions (R Core Team 1999). For models of increasing complexity the full potential of the `C` language is available. In particular, additional `C` variables can be defined when needed, as demonstrated in Section 6.

Unlike the strict unit structure required for the measurement process, the latent process for a SpatPOMP model can have arbitrary spatial dependence between units. We cannot in general define the full coupled dynamics by a collection of `runit_process` functions defined separately for each unit. Therefore, **spatPomp** relies on a `rprocess` function defined exactly as for **pomp**. A **spatPomp** Csnippet for `rprocess` will typically involve a computation looping through the units, which requires access to location data used to specify the interaction between units. The location data can be accessed in the Csnippet using the `globals` argument. Further details on this are postponed to Section 6.

2.8. Simulation

A first step to explore a SpatPOMP model is to simulate stochastic realizations of the latent process and the resulting measurements. This is carried out by `simulate()` which requires specification of `rprocess` and `rmeasure`. For example, `simulate(bm10)` produces a new object of class '`spatPomp`' for which the original data have been replaced with a simulation from the specified model. Unless a `params` argument is supplied, the simulation will be carried out using the parameter vector in `coef(bm10)`. Optionally, `simulate` can be made to return a class '`data.frame`' object by supplying the argument `format='data.frame'` in the call to `simulate`.

3. Likelihood evaluation

Likelihood evaluation for SpatPOMP models is a by-product of a filtering calculation. The curse of dimensionality associated with spatiotemporal models can make filtering for SpatPOMP models computationally challenging, even though a single likelihood evaluation cannot be more than a small step toward a complete likelihood-based inference workflow. A widely used time-series filtering technique is the basic particle filter (PF) available as `pfilter` in the **pomp** package. However, PF and many of its variations scale poorly with dimension (Bengtsson *et al.* 2008; Snyder *et al.* 2015). Thus, in the spatiotemporal context, successful particle filtering requires state-of-the-art algorithms. Below, we introduce four such algorithms implemented in the **spatPomp** package: a guided intermediate resampling filter (GIRF) implemented as `girf`, an adapted bagged filter (ABF) implemented as `abf`, an ensemble Kalman filter (EnKF) implemented as `enkf`, and a block particle filter (BPF) implemented as `bpfilter`.

The filtering problem can be decomposed into two steps, prediction and filtering. For all the filters we consider here, the prediction step involves simulating from the latent process model. The algorithms differ primarily in their approaches to the filtering step, also known as the data assimilation step or the analysis step. For PF, the filtering step is a weighted resampling from the prediction particles, and the instability of these weights in high dimensions is the fundamental scalability issue with the algorithm. GIRF carries out this resampling at many intermediate timepoints with the goal of breaking an intractable resampling problem into a sequence of tractable ones. EnKF estimates variances and covariances of the prediction simulations, and carries out an update rule that would be exact for a Gaussian system. BPF carries out the resampling independently over a partition of the units, aiming for an inexact but numerically tractable approximation. ABF combines together many high-variance filters using local weights to beat the curse of dimensionality. We proceed to describe these algorithms in more detail.

3.1. The guided intermediate resampling filter (GIRF)

The guided intermediate resampling filter (GIRF, Park and Ionides 2020) is an extension of the auxiliary particle filter (APF, Pitt and Shepard 1999). GIRF is appropriate for moderately high-dimensional SpatPOMP models with a continuous-time latent process. All particle filters compute importance weights for proposed particles and carry out resampling to focus computational effort on particles consistent with the data (see reviews by Arulampalam *et al.* 2002; Doucet and Johansen 2011; Kantas *et al.* 2015). In the context of **pomp**, the `pfilter` function is discussed by King *et al.* (2016). GIRF combines two techniques for improved scaling of particle filters: the use of a guide function and intermediate resampling.

The guide function steers particles using importance weights that anticipate upcoming observations. Future measurements are considered up to a lookahead horizon, L. APF corresponds to a lookahead horizon $L = 2$, and a basic particle filter has $L = 1$. Values $L \leq 3$ are typical for GIRF.

Intermediate resampling breaks each observation interval into S sub-intervals, and carries out reweighting and resampling on each sub-interval. Perhaps surprisingly, intermediate resampling can facilitate some otherwise intractable importance sampling problems (Del Moral and Murray 2015). APF and the basic particle filter correspond to $S = 1$, whereas choosing $S = U$ gives favorable scaling properties (Park and Ionides 2020).

In Algorithm 1 the F, G and P superscripts indicate filtered, guide and proposal particles, respectively. The goal for the pseudocode in Algorithm 1, and subsequent algorithms in this paper, is a succinct description of the logic of the procedure rather than a complete recipe for efficient coding. Therefore, the pseudocode does not focus on opportunities for memory overwriting and vectorization, though these may be implemented in **spatPomp** code.

We call the guide in Algorithm 1 a *bootstrap guide function* since it is based on resampling the Monte Carlo residuals calculated in step 5. Another option of a guide function in `girf` is the simulated moment guide function developed by Park and Ionides (2020) which uses the `eunit_measure`, `vunit_measure` and `munit_measure` model components together with simulations to calculate the guide. The expectation of Monte Carlo likelihood estimates does not depend on the guide function, so an inexact guide approximation may lead to loss of numerical efficiency but does not affect the consistency of the procedure.

The intermediate resampling is represented in Algorithm 1 by the loop of $s = 1, \ldots, S$ in step 6. The intermediate times are defined by $t_{n,s} = t_n + (t_{n+1} - t_n) \cdot s/S$ and we write $\boldsymbol{X}_{n,s} = \boldsymbol{X}(t_{n,s})$. The resampling weights (step 12) are defined in terms of guide function evaluations $g^{P,j}_{n,s}$. The only requirement for the guide function to achieve unbiased estimates is that it satisfies $g^{F,j}_{0,0} = 1$ and $g^{P,j}_{N-1,S} = f_{\boldsymbol{Y}_N|\boldsymbol{X}_N}(\boldsymbol{y}^*_N \,|\, \boldsymbol{X}^{F,j}_{N-1,S}\,;\theta)$, which is the case in Algorithm 1. The particular guide function calculated in step 11 evaluates particles using a prediction centered on a function

$$\boldsymbol{\mu}(\boldsymbol{x}, s, t\,;\theta) \approx \mathsf{E}[\boldsymbol{X}(t) \,|\, \boldsymbol{X}(s) = \boldsymbol{x}\,;\theta].$$

We call $\boldsymbol{\mu}(\boldsymbol{x}, s, t\,;\theta)$ a *deterministic trajectory* associated with $\boldsymbol{X}(t)$. For a continuous time SpatPOMP model, this trajectory is typically the solution to a system of differential equations

Algorithm 1: `girf(P,Np=`J`, Ninter=`S`, Nguide=`K`, Lookahead=`L`)`, using notation from Table 1 where `P` is a '`spatPomp`' object equipped with `rprocess`, `dunit_measure`, `rinit`, `skeleton`, `obs`, `coef`.

```
input: simulator for $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1};\theta)$ and $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0;\theta)$; evaluator for
    $f_{Y_{u,n}|X_{u,n}}(y_{u,n} \mid x_{u,n};\theta)$, and $\boldsymbol{\mu}(\boldsymbol{x},s,t;\theta)$; data, $\boldsymbol{y}^*_{1:N}$; parameter, $\theta$; number of
    particles, $J$; number of guide simulations, $K$; number of intermediate timesteps,
    $S$; number of lookahead lags, $L$.
1  initialize: simulate $\boldsymbol{X}^{F,j}_{0,0} \sim f_{\boldsymbol{X}_0}(\,\cdot\,;\theta)$ and set $g^{F,j}_{0,0}=1$ for $j$ in $1\!:\!J$
2  for $n$ in $0\!:\!N-1$ do
3      sequence of guide forecast times, $\mathbb{L}=(n+1)\!:\!\min(n+L,N)$
4      guide simulations, $\boldsymbol{X}^{G,j,k}_{\mathbb{L}} \sim f_{\boldsymbol{X}_{\mathbb{L}}|\boldsymbol{X}_n}(\,\cdot\,|\boldsymbol{X}^{F,j}_{n,0};\theta)$ for $j$ in $1\!:\!J$, $k$ in $1\!:\!K$
5      guide residuals, $\boldsymbol{\epsilon}^{j,k}_{0,\ell}=\boldsymbol{X}^{G,j,k}_{\ell}-\boldsymbol{\mu}(\boldsymbol{X}^{F,j}_{n},t_n,t_\ell;\theta)$ for $j$ in $1\!:\!J$, $k$ in $1\!:\!K$, $\ell$ in $\mathbb{L}$
6      for $s$ in $1\!:\!S$ do
7          prediction simulations, $\boldsymbol{X}^{P,j}_{n,s} \sim f_{\boldsymbol{X}_{n,s}|\boldsymbol{X}_{n,s-1}}(\,\cdot\,|\boldsymbol{X}^{F,j}_{n,s-1};\theta)$ for $j$ in $1\!:\!J$
8          deterministic trajectory, $\boldsymbol{\mu}^{P,j}_{n,s,\ell}=\boldsymbol{\mu}(\boldsymbol{X}^{P,j}_{n,s},t_{n,s},t_\ell;\theta)$ for $j$ in $1\!:\!J$, $\ell$ in $\mathbb{L}$
9          pseudo guide simulations, $\hat{\boldsymbol{X}}^{j,k}_{n,s,\ell}=\boldsymbol{\mu}^{P,j}_{n,s,\ell}+\boldsymbol{\epsilon}^{j,k}_{s-1,\ell}-\boldsymbol{\epsilon}^{j,k}_{s-1,n+1}+\sqrt{\frac{t_{n+1}-t_{n,s}}{t_{n+1}-t_{n,0}}}\,\boldsymbol{\epsilon}^{j,k}_{s-1,n+1}$
            for $j$ in $1\!:\!J$, $k$ in $1\!:\!K$, $\ell$ in $\mathbb{L}$
10         discount factor, $\eta_{n,s,\ell}=1-(t_{n+\ell}-t_{n,s})/\{(t_{n+\ell}-t_{\max(n+\ell-L,0)})\cdot(1+\mathbb{1}_{L=1})\}$
11         $g^{P,j}_{n,s}=\prod_{\ell\,\mathrm{in}\,\mathbb{L}}\prod_{u=1}^{U}\left[\frac{1}{K}\sum_{k=1}^{K} f_{Y_{u,\ell}|X_{u,\ell}}\Big(y^*_{u,\ell}\,\big|\,\hat{X}^{j,k}_{u,n,s,\ell};\theta\Big)\right]^{\eta_{n,s,\ell}}$ for $j$ in $1\!:\!J$
12         for $j$ in $1\!:\!J$, $w^j_{n,s}=\begin{cases} f_{\boldsymbol{Y}_n|\boldsymbol{X}_n}(\boldsymbol{y}_n \mid \boldsymbol{X}^{F,j}_{n,s-1};\theta)\; g^{P,j}_{n,s}\big/g^{F,j}_{n,s-1} & \text{if } s=1 \text{ and } n\neq 0\\ g^{P,j}_{n,s}\big/g^{F,j}_{n,s-1} & \text{else}\end{cases}$
13         log likelihood component, $c_{n,s}=\log\Big(J^{-1}\sum_{q=1}^{J} w^q_{n,s}\Big)$
14         normalized weights, $\tilde{w}^j_{n,s}=w^j_{n,s}\Big/\sum_{q=1}^{J} w^q_{n,s}$ for $j$ in $1\!:\!J$
15         select resample indices, $r_{1:J}$ with $\mathbb{P}[r_j=q]=\tilde{w}^q_{n,s}$ for $j$ in $1\!:\!J$
16         $\boldsymbol{X}^{F,j}_{n,s}=\boldsymbol{X}^{P,r_j}_{n,s}$ , $g^{F,j}_{n,s}=g^{P,r_j}_{n,s}$ , $\boldsymbol{\epsilon}^{j,k}_{s,\ell}=\boldsymbol{\epsilon}^{r_j,k}_{s-1,\ell}$ for $j$ in $1\!:\!J$, $k$ in $1\!:\!K$, $\ell$ in $\mathbb{L}$
17     end
18     set $\boldsymbol{X}^{F,j}_{n+1,0}=\boldsymbol{X}^{F,j}_{n,S}$ and $g^{F}_{n+1,0,j}=g^{F}_{n,S,j}$ for $j$ in $1\!:\!J$
19 end
output: log likelihood, $\lambda^{\mathrm{GIRF}}=\sum_{n=0}^{N-1}\sum_{s=1}^{S} c_{n,s}$, and filter particles, $\boldsymbol{X}^{F,1:J}_{N,0}$
complexity: $\mathcal{O}\big(JLUN(K+S)\big)$
```

Algorithm 2: `enkf(P,Np=`J`)`, using notation from Table 1 where `P` is a '`spatPomp`' object equipped with `rprocess`, `eunit_measure`, `vunit_measure`, `rinit`, `coef`, `obs`.

```
input: simulator for $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}\,;\theta)$ and $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)$; evaluator for $\mathrm{e}_u(X_{u,n},\theta)$
       and $\mathrm{v}_u(X_{u,n},\theta)$; parameter, $\theta$; data, $\boldsymbol{y}^*_{1:N}$; number of particles, $J$.
1  initialize filter particles, $\boldsymbol{X}_0^{F,j} \sim f_{\boldsymbol{X}_0}(\,\cdot\,;\theta)$ for $j$ in $1\!:\!J$
2  for $n$ in $1\!:\!N$ do
3      prediction ensemble, $\boldsymbol{X}_n^{P,j} \sim f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\,\cdot\,|\boldsymbol{X}_{n-1}^{F,j};\theta)$ for $j$ in $1\!:\!J$
4      centered prediction ensemble, $\tilde{\boldsymbol{X}}_n^{P,j} = \boldsymbol{X}_n^{P,j} - \frac{1}{J}\sum_{q=1}^{J}\boldsymbol{X}_n^{P,q}$ for $j$ in $1\!:\!J$
5      forecast ensemble, $\hat{\boldsymbol{Y}}_n^{j} = \mathrm{e}_u(X_{u,n}^{P,j},\theta)$ for $j$ in $1\!:\!J$
6      forecast mean, $\overline{\boldsymbol{Y}}_n = \frac{1}{J}\sum_{j=1}^{J}\hat{\boldsymbol{Y}}_n^{j}$
7      centered forecast ensemble, $\tilde{\boldsymbol{Y}}_n^{j} = \hat{\boldsymbol{Y}}_n^{j} - \overline{\boldsymbol{Y}}_n$ for $j$ in $1\!:\!J$
8      forecast measurement variance, $R_{u,\tilde{u}} = \mathbb{1}_{u,\tilde{u}}\,\frac{1}{J}\sum_{j=1}^{J}\mathrm{v}_u(\boldsymbol{X}_{u,n}^{P,j},\theta)$ for $u,\tilde{u}$ in $1\!:\!U$
9      forecast estimated covariance, $\Sigma_Y = \frac{1}{J-1}\sum_{j=1}^{J}(\tilde{\boldsymbol{Y}}_n^{j})(\tilde{\boldsymbol{Y}}_n^{j})^T + R$
10     prediction and forecast sample covariance, $\Sigma_{XY} = \frac{1}{J-1}\sum_{j=1}^{J}(\tilde{\boldsymbol{X}}_n^{P,j})(\tilde{\boldsymbol{Y}}_n^{j})^T$
11     Kalman gain, $K = \Sigma_{XY}\Sigma_Y^{-1}$
12     artificial measurement noise, $\boldsymbol{\epsilon}_n^{j} \sim \mathrm{Normal}(\boldsymbol{0},R)$ for $j$ in $1\!:\!J$
13     errors, $\boldsymbol{r}_n^{j} = \hat{\boldsymbol{Y}}_n^{j} - \boldsymbol{y}_n^*$ for $j$ in $1\!:\!J$
14     filter update, $\boldsymbol{X}_n^{F,j} = \boldsymbol{X}_n^{P,j} + K(\boldsymbol{r}_n^{j} + \boldsymbol{\epsilon}_n^{j})$ for $j$ in $1\!:\!J$
15     $\lambda_n = \log\left[\phi(\boldsymbol{y}_n^*\,;\overline{\boldsymbol{Y}}_n,\Sigma_Y)\right]$ where $\phi(\cdot\,;\boldsymbol{\mu},\Sigma)$ is the Normal$(\boldsymbol{\mu},\Sigma)$ density.
16 end
output: filter sample, $\boldsymbol{X}_n^{F,1:J}$, for $n$ in $1\!:\!N$; log likelihood estimate, $\lambda^{\mathrm{EnKF}} = \sum_{n=1}^{N}\lambda_n$
complexity: $\mathcal{O}(JUN)$
```

that define a vector field called the *skeleton* (Tong 1990). The skeleton is specified by a Csnippet filling the `skeleton` argument to `spatPomp()`. The forecast spread around this deterministic prediction is given by the simulated bootstrap residuals constructed in step 5.

3.2. The ensemble Kalman filter (EnKF)

Ensemble Kalman filter (EnKF) algorithms use observations to update simulations from the latent Markov model via an update rule based on a Gaussian conditional density (Evensen 1994; Evensen and van Leeuwen 1996). The prediction step advances the Monte Carlo ensemble to the next observation time by using simulations from the postulated model In the filtering step, the sample estimate of the state covariance matrix and the measurement variance are combined to update each ensemble member, using a rule that approximates the conditional distribution were the variables jointly Gaussian.

The **spatPomp** implementation of EnKF is described in Algorithm 2. In step 8, the conditional variance of the measurement at the current time step is approximated by constructing a diagonal covariance matrix whose diagonal elements are the sample average of the theoretical unit measurement variances at each unit. This is written using an indicator function $\mathbb{1}_{u,\tilde{u}}$ which takes value 1 if $u = \tilde{u}$ and 0 otherwise. The `vunit_measure` model component aids in this step whereas `eunit_measure` specifies how we can construct forecast data (step 5) that can be used to later update our prediction particles in step 14. In step 12 we add artificial

Algorithm 3: `bpfilter(P,Np=`J`,block_list=`$\mathcal{B}$`)` using notation from Table 1 where `P` is a '`spatPomp`' object equipped with `rprocess`, `dunit_measure`, `rinit`, `obs`, `coef`.

```
input: simulator for $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \,|\, \boldsymbol{x}_{n-1}\,;\theta)$ and $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)$; number of particles, $J$;
       evaluator for $f_{Y_{u,n}|X_{u,n}}(y_{u,n} \,|\, x_{u,n}\,;\theta)$; data, $\boldsymbol{y}^*_{1:N}$; parameter, $\theta$; blocks, $\mathcal{B}_{1:K}$;
1 initialization, $\boldsymbol{X}_0^{F,j} \sim f_{\boldsymbol{X}_0}(\,\cdot\,;\theta)$ for $j$ in $1\!:\!J$
2 for $n$ in $1\!:\!N$ do
3     prediction, $\boldsymbol{X}_n^{P,j} \sim f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\,\cdot\,|\boldsymbol{X}_{n-1}^{F,j};\theta)$ for $j$ in $1\!:\!J$
4     block weights, $w_{k,n}^j = \prod_{u\in\mathcal{B}_k} f_{Y_{u,n}|X_{u,n}}(y^*_{u,n} \,|\, X_{u,n}^{P,j}\,;\theta)$ for $j$ in $1\!:\!J$, $k$ in $1\!:\!K$
5     resampling indices, $r_{k,n}^j$ with $\mathbb{P}\left[r_{k,n}^j = i\right] = \tilde{w}_{k,n}^i \big/ \sum_{q=1}^J w_{k,n}^q$ for $j$ in $1\!:\!J$, $k$ in $1\!:\!K$
6     resample, $X_{\mathcal{B}_k,n}^{F,j} = X_{\mathcal{B}_k,n}^{P,r_{j,k}}$ for $j$ in $1\!:\!J$, $k$ in $1\!:\!K$
7 end
output: log likelihood, $\lambda^{\text{BPF}}(\theta) = \sum_{n=1}^N \sum_{k=1}^K \log\left(\frac{1}{J}\sum_{j=1}^J w_{k,n}^j\right)$, filter particles $\boldsymbol{X}_{1:N}^{F,1:J}$
complexity: $\mathcal{O}(JUN)$
```

measurement error to arrive at a consistent sample covariance for the filtering step (Evensen 1994; Evensen and van Leeuwen 1996), writing Normal($\boldsymbol{\mu}, \Sigma$) for independent draws from a multivariate normal random variable with mean $\boldsymbol{\mu}$ and variance matrix Σ.

EnKF achieves good dimensional scaling relative to PF by replacing the resampling step with an update rule inspired by a Gaussian approximation. Since we envisage **spatPomp** primarily for situations with relatively low dimension, this implementation does not engage in regularization issues required when the dimension of the observation space exceeds the number of particles in the ensemble.

Our EnKF implementation supposes we have access to the measurement mean function, $\mathrm{e}_{u,n}(x,\theta)$, and the measurement variance, $\mathrm{v}_u(x,\theta)$, defined in Table 1. For common choices of measurement model, such as Gaussian or negative binomial, $\mathrm{e}_{u,n}$ and $\mathrm{v}_{u,n}$ are readily available. Section 6 provides an example of simple Csnippets for `eunit_measure` and `vunit_measure`. In general, the functional forms of $\mathrm{e}_{u,n}$ and $\mathrm{v}_{u,n}$ may depend on u and n, or on covariate time series.

3.3. Block particle filter

Algorithm 3 is an implementation of the block particle filter (BPF Rebeschini and van Handel 2015), also called the factored particle filter (Ng *et al.* 2002). BPF partitions the units into a collection of blocks, $\mathcal{B}_1, \ldots, \mathcal{B}_K$. Each unit is placed in exactly one block. BPF generates proposal particles by simulating from the joint latent process across all blocks, exactly as the particle filter does. However, the resampling in the filtering step is carried out independently for each block, using weights corresponding only to the measurements in the block. Different proposal particles may be successful for different blocks, and the block resampling allows the filter particles to paste together these successful proposals. This avoids the curse of dimensionality, while introducing an approximation error that may be large or small depending on the model under consideration.

The user has a choice of specifying the blocks using either the `block_list` argument or `block_size`, but not both. `block_list` takes a class '`list`' object where each entry is a vector representing the units in a block. `block_size` takes an integer and evenly partitions $1\,{:}\,U$ into blocks of size approximately `block_size`. For example, if there are 4 units, executing `bpfilter` with `block_size=2` is equivalent to setting `block_list=list(c(1,2),c(3,4))`.

3.4. Adapted bagged filter (ABF)

The adapted bagged filter (Ionides *et al.* 2021) combines many independent particle filters. This is called *bagging*, (*b*ootstrap *agg*regat*ing*), since a basic particle filter is also called a bootstrap filter. The adapted distribution is the conditional distribution of the latent process given its current value and the subsequent observation (Johansen and Doucet 2008). In the adapted bagged filter, each bootstrap replicate makes a Monte Carlo approximation to a draw from the adapted distribution. Thus, in the pseudocode of Algorithm 4, $\boldsymbol{X}^{A,i}_{0:N}$ is a Monte Carlo sample targeting the adapted sampling distribution,

$$f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)\prod_{n=1}^{N} f_{\boldsymbol{X}_n|\boldsymbol{Y}_n,\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n\,|\,\boldsymbol{y}^*_n,\boldsymbol{x}_{n-1}\,;\theta). \qquad (1)$$

Each adapted simulation replicate is constructed by importance sampling using proposal particles $\{\boldsymbol{X}^{P,i,j}_n\}$. The ensemble of adapted simulation replicates are then weighted using data in a spatiotemporal neighborhood of each observation to obtain a locally combined Monte Carlo sample targeting the filter distribution, with some approximation error due to the finite spatiotemporal neighborhood used. This local aggregation of the bootstrap replicates also provides an evaluation of the likelihood function.

On a given bootstrap replicate i at a given time n, all the adapted proposal particles $\boldsymbol{X}^{P,i,1:J}_n$ in step 3 are necessarily close to each other in state space because they share the parent particle $\boldsymbol{X}^{A,i}_{n-1}$. This reduces imbalance in the adapted weights in step 5, which helps to battle the curse of dimensionality that afflicts importance sampling. The combination of the replicates for the filter estimate in step 11 is carried out using only weights in a spatiotemporal neighborhood, thus avoiding the curse of dimensionality. For any point (u,n), the neighborhood $B_{u,n}$ should be specified as a subset of $A_{u,n}=\{(\tilde{u},\tilde{n}):\tilde{n}<n \text{ or } (\tilde{u}<u \text{ and } \tilde{n}=n)\}$. If the model has a mixing property, meaning that conditioning on the observations in the neighborhood $B_{u,n}$ is negligibly different from conditioning on the full set $A_{u,n}$, then the approximation involved in this localization is adequate.

Steps 1 through 7 do not involve interaction between replicates and therefore iteration over i can be carried out in parallel. If a parallel back-end has been set up by the user, the `abf` method will parallelize computations over the replicates using multiple cores. The user can register a parallel back-end using the `doParallel` package (Wallig and Weston 2020, 2019) prior to calling `abf`.

```
R> library("doParallel")
R> registerDoParallel(detectCores())
```

The neighborhood is supplied via the `nbhd` argument to `abf` as a function which takes a point in space-time, (u,n), and returns a list of points in space-time which correspond to $B_{u,n}$. An example with $B_{u,n}=\{(u-1,n),(u,n-1)\}$ follows.

Algorithm 4: `abf(P,replicates=`$\mathcal{I}$`,Np=`J`,nbhd=`$B_{u,n}$`)`, using notation from Table 1 where `P` is a ‘`spatPomp`’ object equipped with `rprocess`, `dunit_measure`, `rinit`, `obs`, `coef`.

```
input: simulator for $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \,|\, \boldsymbol{x}_{n-1}\,;\theta)$ and $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)$; evaluator for
       $f_{Y_{u,n}|X_{u,n}}(y_{u,n} \,|\, x_{u,n}\,;\theta)$; data, $\boldsymbol{y}^*_{1:N}$; parameter, $\theta$; number of particles per
       replicate, $J$; number of replicates, $\mathcal{I}$; neighborhood structure, $B_{u,n}$
1  initialize adapted simulation, $\boldsymbol{X}^{A,i}_0 \sim f_{\boldsymbol{X}_0}(\cdot\,;\theta)$ for $i$ in $1\!:\!\mathcal{I}$
2  for $n$ in $1\!:\!N$ do
3      proposals, $\boldsymbol{X}^{P,i,j}_n \sim f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\,\cdot \,|\, \boldsymbol{X}^{A,i}_{n-1}\,;\theta)$ for $i$ in $1\!:\!\mathcal{I}$, $j$ in $1\!:\!J$
4      $w^{i,j}_{u,n} = f_{Y_{u,n}|X_{u,n}}(y^*_{u,n} \,|\, X^{P,i,j}_{u,n}\,;\theta)$ for $u$ in $1\!:\!U$, $i$ in $1\!:\!\mathcal{I}$, $j$ in $1\!:\!J$
5      adapted resampling weights, $w^{A,i,j}_n = \prod_{u=1}^{U} w^{i,j}_{u,n}$ for $u$ in $1\!:\!U$, $i$ in $1\!:\!\mathcal{I}$, $j$ in $1\!:\!J$
6      set $\boldsymbol{X}^{A,i}_n = \boldsymbol{X}^{P,i,j}_n$ with probability $w^{A,i,j}_n \left(\sum_{q=1}^{J} w^{A,i,q}_n\right)^{-1}$ for $i$ in $1\!:\!\mathcal{I}$
7      $w^{P,i,j}_{u,n} = \prod_{\tilde{n}=1}^{n-1}\left[\frac{1}{J}\sum_{q=1}^{J}\prod_{(\tilde{u},\tilde{n})\in B_{u,n}} w^{i,q}_{\tilde{u},\tilde{n}}\right]\prod_{(\tilde{u},n)\in B_{u,n}} w^{i,j}_{\tilde{u},\tilde{n}}$ for $u$ in $1\!:\!U$, $i$ in $1\!:\!\mathcal{I}$, $j$ in $1\!:\!J$
8  end
9  filter weights, $w^{F,i,j}_{u,n} = \dfrac{w^{i,j}_{u,n}\; w^{P,i,j}_{u,n}}{\sum_{p=1}^{\mathcal{I}}\sum_{q=1}^{J} w^{P,p,q}_{u,n}}$ for $u$ in $1\!:\!U$, $n$ in $1\!:\!N$, $i$ in $1\!:\!\mathcal{I}$, $j$ in $1\!:\!J$
10 conditional log likelihood, $\lambda_{u,n} = \log\left(\sum_{i=1}^{\mathcal{I}}\sum_{j=1}^{J} w^{F,i,j}_{u,n}\right)$ for $u$ in $1\!:\!U$, $n$ in $1\!:\!N$
11 set $X^{F,j}_{u,n} = X^{P,i,k}_{u,n}$ with probability $w^{F,i,k}_{u,n}\, e^{-\lambda_{u,n}}$ for $u$ in $1\!:\!U$, $n$ in $1\!:\!N$, $j$ in $1\!:\!J$
output: filter particles, $\boldsymbol{X}^{F,1:J}_n$, for $n$ in $1\!:\!N$; log likelihood, $\lambda^{\mathrm{ABF}} = \sum_{n=1}^{N}\sum_{u=1}^{U}\lambda_{u,n}$
complexity: $\mathcal{O}(\mathcal{I}JUN)$
```

```
R> example_nbhd <- function(object, unit, time){
+    nbhd_list = list()
+    if(time>1) nbhd_list <- c(nbhd_list, list(c(unit, time-1)))
+    if(unit>1) nbhd_list <- c(nbhd_list, list(c(unit-1, time)))
+    return(nbhd_list)
+  }
```

ABF can be combined with the guided intermediate resampling technique used by GIRF to give an algorithm called ABF-IR (Ionides *et al.* 2021) implemented as `abfir`.

3.5. Considerations for choosing a filter

Of the four filters described above, only GIRF provides an unbiased estimate of the likelihood. However, GIRF has a relatively weak theoretical scaling support, beating the curse of dimensionality only in the impractical situation of an ideal guide function (Park and Ionides 2020). EnKF, ABF and BPF gain scalability by making different approximations that may or may not be appropriate for a given situation. The choice of filter in a particular application is primarily an empirical question, and **spatPomp** facilitates a responsible approach of trying multiple options. Nevertheless, we offer some broad guidance. EnKF has low variance but is relatively sensitive to deviations from normality; in examples where this is not a concern, such as the example in Sec. **??**, EnKF can be expected to perform well. BPF can break conservation laws satisfied by the latent process, such as a constraint on the total population in all units; ABF satisfies such constraints but has been found to have higher variance than BPF on some benchmark problems (Ionides *et al.* 2021). For the measles model built by `measles()`, BPF and ABF have been found to perform better than EnKF and GIRF (Ionides *et al.* 2021). For the Lorenz-96 example built by `lorenz()`, GIRF and BPF perform well (Ionides *et al.* 2021).

4. Inference for SpatPOMP models

We focus on iterated filtering methods (Ionides *et al.* 2015) which provide a relatively simple way to coerce filtering algorithms to carry out parameter inference, applicable to the general class of SpatPOMP models considered by **spatPomp**. The main idea of iterated filtering is to extend a POMP model to include dynamic parameter perturbations. Repeated filtering, with parameter perturbations of decreasing magnitude, approaches the maximum likelihood estimate. Here, we present iterated versions of GIRF, EnKF and the unadapted bagged filter (UBF), a version of ABF with $J = 1$. These algorithms are known as IGIRF (Park and Ionides 2020), IEnKF (Li *et al.* 2020) and IUBF (Ionides *et al.* 2021) respectively. SpatPOMP model estimation is an active area for research (for example, Katzfuss *et al.* 2020) and **spatPomp** provides a platform for developing and testing new methods, in addition to new models and data analysis.

4.1. Iterated GIRF for parameter estimation

Algorithm 5 describes `igirf()`, the **spatPomp** implementation of IGIRF. This algorithm carries out the IF2 algorithm of Ionides *et al.* (2015) with filtering carried out by GIRF, therefore its implementation combines the `mif2` function in **pomp** with `girf` (Algorithm 1). For Algorithm 5, we unclutter the pseudocode by using a subscript and superscript notation for free indices, meaning subscripts and superscripts for which a value is not explicitly specified in the code. We use the convention that a free subscript or superscript is evaluated for all values in its range, leading to an implicit 'for' loop. This does not hold for capitalized subscripts and superscripts, which describe the purpose of a Monte Carlo particle, matching usage in Algorithm 1.

The quantity $\Theta_{n,s}^{P,m,j}$ gives a perturbed parameter vector for θ corresponding to particle j on iteration m at the s^{th} intermediate time between n and $n+1$. The perturbations in Algorithm 5 are taken to follow a multivariate normal distribution, with a diagonal covariance matrix scaled by σ_{n,d_θ}. Normal perturbations are not theoretically required, but this is a common choice in practice. The `igirf` function permits perturbations to be carried out on a transformed scale, specified using the `partrans` argument, to accommodate situations where normally distributed perturbations are more natural on the log or logistic scale, or any other user-specified scale. For regular parameters, i.e. parameters that are not related to the initial conditions of the dynamics, it may be appropriate to set the perturbation scale independent of n. If parameters are transformed so that a unit scale is relevant, for example using a logarithmic transform for non-negative parameters, an appropriate default value is $\sigma_{n,d_\theta} = 0.02$. Initial value parameters (IVPs) are those that determine only the latent state at time t_0, and these should be perturbed only at the beginning of each iteration m. The matrix $\sigma_{0:N,1:D_\theta}$ can be constructed using the `rw.sd` function, which simplifies the

Algorithm 5: `igirf(P,params=`θ_0`,Ngirf=`M`,Np=`J`,Ninter=`S`,Nguide=`K`,Lookahead=`L`,rw.sd=`$\sigma_{0:N,1:D_\theta}$`,cooling.fraction.50=`a`)` using notation from Table 1 where `P` is a '`spatPomp`' object equipped with `rprocess`, `dunit_measure`, `skeleton`, `rinit`, `obs`.

```
input: simulator for $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \,|\, \boldsymbol{x}_{n-1}\,;\theta)$ and $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)$; evaluator for
       $f_{Y_{u,n}|X_{u,n}}(y_{u,n}\,|\,x_{u,n}\,;\theta)$, and $\boldsymbol{\mu}(\boldsymbol{x},s,t\,;\theta)$; data, $\boldsymbol{y}^*_{1:N}$; starting parameter, $\theta_0$;
       iterations, $M$; particles, $J$; lookahead lags, $L$; intermediate timesteps, $S$; random
       walk intensities, $\sigma_{0:N,1:D_\theta}$; cooling fraction in 50 iterations, $a$.
note: free indices are implicit 'for' loops, calculated for $j$ in $1\!:\!J$, $k$ in $1\!:\!K$,
      $\ell$ in $(n+1)\!:\!\min(n+L,N)$, $u$ in $1\!:\!U$, $d_\theta, d'_\theta$ in $1\!:\!D_\theta$.
1  initialize parameters, $\Theta^{F,0,j}_{N-1,S} = \theta_0$
2  for $m$ in $1\!:\!M$ do
3      initialize parameters, $\Theta^{F,m,j}_{0,0} \sim \mathrm{Normal}\big(\Theta^{F,m-1,j}_{N-1,S}\,, a^{2m/50}\,\Sigma_{\mathrm{ivp}}\big)$ for
         $\left[\Sigma_{\mathrm{ivp}}\right]_{d_\theta,d'_\theta} = \sigma^2_{\mathrm{ivp},d_\theta}\mathbb{1}_{d_\theta=d'_\theta}$
4      initialize filter particles, simulate $\boldsymbol{X}^{F,j}_{0,0} \sim f_{\boldsymbol{X}_0}\left(\,\cdot\,;\Theta^{F,m,j}_{0,0}\right)$ and set $g^{F,j}_{n,0}=1$
5      for $n$ in $0\!:\!N-1$ do
6          guide simulations, $\boldsymbol{X}^{G,j,k}_{\ell} \sim f_{\boldsymbol{X}_\ell|\boldsymbol{X}_n}\big(\,\cdot\,|\boldsymbol{X}^{F,j}_{n,0}\,;\Theta^{F,m,j}_{n,0}\big)$
7          guide residuals, $\boldsymbol{\epsilon}^{j,k}_{0,\ell} = \boldsymbol{X}^{G,j,k}_{\ell} - \boldsymbol{\mu}\big(\boldsymbol{X}^{F,j}_{n,0}, t_n, t_\ell\,;\Theta^{F,m,j}_{n,0}\big)$
8          for $s$ in $1\!:\!S$ do
9              perturb parameters, $\Theta^{P,m,j}_{n,s} \sim \mathrm{Normal}\big(\Theta^{F,m,j}_{n,s-1}\,, a^{2m/50}\,\Sigma_n\big)$ for
                 $\left[\Sigma_n\right]_{d_\theta,d'_\theta} = \sigma^2_{n,d_\theta}\mathbb{1}_{d_\theta=d'_\theta}/S$
10             prediction simulations, $\boldsymbol{X}^{P,j}_{n,s} \sim f_{\boldsymbol{X}_{n,s}|\boldsymbol{X}_{n,s-1}}\big(\,\cdot\,|\boldsymbol{X}^{F,j}_{n,s-1}\,;\Theta^{P,m,j}_{n,s}\big)$
11             deterministic trajectory, $\boldsymbol{\mu}^{P,j}_{n,s,\ell} = \boldsymbol{\mu}\big(\boldsymbol{X}^{P,j}_{n,s}, t_{n,s}, t_\ell\,;\Theta^{P,m,j}_{n,s}\big)$
12             pseudo guide simulations,
                 $\hat{\boldsymbol{X}}^{j,k}_{n,s,\ell} = \boldsymbol{\mu}^{P,j}_{n,s,\ell} + \boldsymbol{\epsilon}^{j,k}_{s-1,\ell} - \boldsymbol{\epsilon}^{j,k}_{s-1,n+1} + \sqrt{\frac{t_{n+1}-t_{n,s}}{t_{n+1}-t_{n,0}}}\,\boldsymbol{\epsilon}^{j,k}_{s-1,n+1}$
13             discount factor, $\eta_{n,s,\ell} = 1-(t_{n+\ell}-t_{n,s})/\{(t_{n+\ell}-t_{\max(n+\ell-L,0)})\cdot(1+\mathbb{1}_{L=1})\}$
14             $g^{P,j}_{n,s} = \prod_{\ell=n+1}^{\min(n+L,N)}\prod_{u=1}^{U}\left[\frac{1}{K}\sum_{k=1}^{K} f_{Y_{u,\ell}|X_{u,\ell}}\Big(y^*_{u,\ell}\,|\,\hat{X}^{j,k}_{u,n,s,\ell}\,;\Theta^{P,m,j}_{n,s}\Big)\right]^{\eta_{n,s,\ell}}$
15             $w^j_{n,s} = \begin{cases} f_{\boldsymbol{Y}_n|\boldsymbol{X}_n}\big(\boldsymbol{y}_n\,|\,\boldsymbol{X}^{F,j}_{n,s-1}\,;\Theta^{F,m,j}_{n,s-1}\big)\;g^{P,j}_{n,s}\Big/g^{F,j}_{n,s-1} & \text{if } s=1,\ n\neq 0 \\ g^{P,j}_{n,s}\Big/g^{F,j}_{n,s-1} & \text{else}\end{cases}$
16             normalized weights, $\tilde{w}^j_{n,s} = w^j_{n,s}\Big/\sum_{q=1}^{J} w^q_{n,s}$
17             resampling indices, $r_{1:J}$ with $\mathbb{P}\left[r_j=q\right] = \tilde{w}^q_{n,s}$
18             set $\boldsymbol{X}^{F,j}_{n,s} = \boldsymbol{X}^{P,r_j}_{n,s}$ , $g^{F,j}_{n,s} = g^{P,r_j}_{n,s}$ , $\boldsymbol{\epsilon}^{j,k}_{s,\ell} = \boldsymbol{\epsilon}^{r_j,k}_{s-1,\ell}$, $\Theta^{F,m,j}_{n,s} = \Theta^{P,m,r_j}_{n,s}$
19         end
20     end
21 end
output: Iterated GIRF parameter swarm, $\Theta^{F,M,1:J}_{N-1,S}$
Monte Carlo maximum likelihood estimate: $\frac{1}{J}\sum_{j=1}^{J}\Theta^{F,M,j}_{N-1,S}$
complexity: $\mathcal{O}(MJLUN(K+S))$
```

construction for regular parameters and IVPs. The `cooling.fraction.50` argument takes the fraction of `rw.sd` by which to perturb the parameters after 50 iterations of `igirf`. If using the default geometric cooling schedule, a value of `cooling.fraction.50=0.5` means that the perturbation standard deviation decreases roughly 1% per iteration.

4.2. Iterated EnKF for parameter estimation

Algorithm 6 describes `enkf`, an implementation of the iterated ensemble Kalman filter (IEnKF) which extends the IF2 approach for parameter estimation by replacing a particle filter with an ensemble Kalman filter. As described in Section 4.1, we employ a free index notation whereby superscripts and subscripts that are not otherwise specified have an implicit 'for' loop. An IEnKF algorithm was demonstrated by Li *et al.* (2020). Alternative inference approaches via EnKF include using the EnKF likelihood within Markov chain Monte Carlo (Katzfuss *et al.* 2020).

IEnKF can successfully estimate only those parameters that influence the forecast mean, $\hat{\boldsymbol{Y}}$. In particular, it can fail to estimate variance parameters. For example, in a correlated Gaussian random walk model such as `bm10`, the forecast $\hat{\boldsymbol{Y}}$ is independent of the measurement variance parameter τ, and so IEnKF is ineffective in estimating τ. By contrast, for geometric Brownian motion, which is obtained by exponentiating Brownian motion, IEnKF can estimate τ because higher values of τ lead to higher values of $\hat{\boldsymbol{Y}}$ on average. In this case, if the average forecast is different from the observed data, the τ parameter gets updated accordingly to reduce the error. Therefore, IEnKF may need to be coupled with other parameter estimation methods (such as IGIRF) to estimate parameters that do not affect the forecast mean.

4.3. Iterated block particle filter for parameter estimation

The success of `bpfilter` on a variety of spatiotemporal models (Ionides *et al.* 2021) raises the question of how to extend a block particle filter for parameter estimation. An iterated filtering algorithm accommodating the structure of the block particle filter has recently proposed by Ionides *et al.* (2022). A previous adaptation by Ning and Ionides (2021) addresses the special case where all parameters are unit-specific. These algorithms are implemented by `ibpf`, which requires a `spatPomp` model with the property that estimated parameters can be perturbed across units as well as through time. Thus, any estimated parameter (whether shared or unit-specific) must be coded as a unit-specific parameter in order to apply this method. The spatiotemporal perturbations are used only as an optimization tool for model parameters which are fixed though time and space (for shared parameters) or just through time (for unit-specific parameters). The algorithm uses decreasing perturbation magnitudes so that the perturbed model approaches the fixed parameter model as the optimization proceeds.

An example model compatible with `ibpf` is constructed by the `he10()` function. This builds a measles model similar to the `measles()` example discussed in Section 6, with the difference that the user can select which parameters are unit-specific. For further discussion of `ibpf`, and related questions about selecting shared versus unit-specific parameters, we refer the reader to Ionides *et al.* (2022).

4.4. Iterated UBF for parameter estimation

Algorithm 7 describes the `iubf` function which carries out parameter estimation by iterating

Algorithm 6: `ienkf(P,params =` θ_0`,Nenkf =` M`,cooling.fraction.50 =` a`,rw.sd =` $\sigma_{0:N,1:D_\theta}$`, Np =` J`)`, using notation from Table 1 where `P` is a '`spatPomp`' object equipped with `rprocess`, `eunit_measure`, `vunit_measure`, `rinit`, and `obs`.

```
input: simulator for $f_{X_n|X_{n-1}}(x_n \mid x_{n-1}\,;\theta)$ and $f_{X_0}(x_0\,;\theta)$; evaluator for $e_{u,n}(x,\theta)$ and
       $v_{u,n}(x,\theta)$; data, $y^*_{1:N}$; number of particles, $J$; number of iterations, $M$; starting
       parameter, $\theta_0$; random walk intensities, $\sigma_{0:N,1:D_\theta}$; cooling fraction in 50
       iterations, $a$.
note: free indices are implicit 'for' loops, calculated for $j$ in $1\!:\!J$, $u$ and $\tilde{u}$ in $1\!:\!U$, $d_\theta$ and
      $d'_\theta$ in $1\!:\!D_\theta$.
1  initialize parameters, $\Theta^{F,0,j}_N = \theta_0$
2  for $m$ in $1\!:\!M$ do
3      initialize parameters, $\Theta^{F,m,j}_0 \sim \text{Normal}\big(\Theta^{F,m-1,j}_N\,,\, a^{2m/50}\,\Sigma_0\big)$ for
         $\big[\Sigma_n\big]_{d_\theta,d'_\theta} = \sigma^2_{n,d_\theta}\mathbb{1}_{d_\theta=d'_\theta}$
4      initialize filter particles, simulate $X^{F,j}_0 \sim f_{X_0}\Big(\,\cdot\,;\Theta^{F,m,j}_0\Big)$.
5      for $n$ in $1\!:\!N$ do
6          perturb parameters, $\Theta^{P,m,j}_n \sim \text{Normal}\big(\Theta^{F,m,j}_{n-1}\,,\, a^{2m/50}\,\Sigma_n\big)$
7          prediction ensemble, $X^{P,j}_n \sim f_{X_n|X_{n-1}}\big(\,\cdot\,|X^{F,j}_{n-1};\Theta^{P,m,j}_n\big)$
8          process and parameter ensemble, $Z^{P,j}_n = \begin{pmatrix} X^{P,j}_n \\ \Theta^{P,m,j}_n \end{pmatrix}$
9          centered process and parameter ensemble, $\tilde{Z}^{P,j}_n = Z^{P,j}_n - \frac{1}{J}\sum_{q=1}^J Z^{P,q}_n$
10         forecast ensemble, $\hat{Y}^j_{u,n} = e_u(X^{P,j}_{u,n}, \Theta^{P,m,j}_n)$
11         centered forecast ensemble, $\tilde{Y}^j_n = \hat{Y}^j_n - \frac{1}{J}\sum_{q=1}^J \hat{Y}^q_n$
12         forecast measurement variance, $R_{u,\tilde{u}} = \mathbb{1}_{u,\tilde{u}}\,\frac{1}{J}\sum_{j=1}^J v_u(X^{P,j}_{u,n}, \Theta^{P,m,j}_n)$
13         forecast sample covariance, $\Sigma_Y = \frac{1}{J-1}\sum_{j=1}^J(\tilde{Y}^j_n)(\tilde{Y}^j_n)^T + R$
14         prediction and forecast sample covariance, $\Sigma_{ZY} = \frac{1}{J-1}\sum_{j=1}^J(\tilde{Z}^{P,j}_n)(\tilde{Y}^j_n)^T$
15         Kalman gain: $K = \Sigma_{ZY}\Sigma_Y^{-1}$
16         artificial measurement noise, $\epsilon^j_n \sim \text{Normal}(\mathbf{0}, R)$
17         errors, $r^j_n = \hat{Y}^j_n - y^*_n$
18         filter update: $Z^{F,j}_n = \begin{pmatrix} X^{F,j}_n \\ \Theta^{F,m,j}_n \end{pmatrix} = Z^{P,j}_n + K\big(r^j_n + \epsilon^j_n\big)$
19     end
20 end
21 set $\theta_M = \frac{1}{J}\sum_{j=1}^J \Theta^{F,M,j}_N$
output: Monte Carlo maximum likelihood estimate, $\theta_M$.
complexity: $\mathcal{O}(MJUN)$
```

Algorithm 7: `iubf(P,params =` θ_0`,Nubf =` M`,Nparam =` K`,Nrep_per_param =` $\mathcal{I}$`,nbhd=`$B_{u,n}$`,` `prop=`p`,cooling.fraction.50 =` a`,rw.sd =` $\sigma_{0:N,1:D_\theta}$`)`, using notation from Table 1 where `P` is a '`spatPomp`' object equipped with `rprocess`, `dunit_measure`, `rinit`, `obs` and `coef`.

input: simulator for $f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}(\boldsymbol{x}_n \,|\, \boldsymbol{x}_{n-1}\,;\theta)$ and $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)$; evaluator for $f_{Y_{u,n}|X_{u,n}}(y_{u,n} \,|\, x_{u,n}\,;\theta)$; data, $\boldsymbol{y}^*_{1:N}$; starting parameter, θ_0; number of parameter vectors, K; number of replicates per parameter, $\mathcal{I}$; neighborhood structure, $B_{u,n}$; number of iterations, M; resampling proportion, p; random walk intensities, $\sigma_{0:N,1:D_\theta}$; cooling fraction in 50 iterations, a.

note: free indices are implicit 'for' loops, calculated for i in $1\!:\!\mathcal{I}$, k in $1\!:\!K$, u and $\tilde{u}$ in $1\!:\!U$, d_θ and d'_θ in $1\!:\!D_\theta$.

```
1  initialize parameters, $\Theta^{F,0,k}_N = \theta_0$
2  for $m$ in $1:M$ do
3      initialize parameters, $\Theta^{F,m,k}_0 = \Theta^{F,m-1,k}_N$
4      initialize filter particles, $\boldsymbol{X}^{F,m,k,i}_0 \sim f_{\boldsymbol{X}_0}\left(\,\cdot\,;\Theta^{F,m,k}_0\right)$
5      for $n$ in $1:N$ do
6          perturb parameters, $\Theta^{P,m,k,i}_n \sim \text{Normal}\big(\Theta^{F,m,k}_{n-1}\,,\, a^{2m/50}\,\Sigma_n\big)$, where
           $\big[\Sigma_n\big]_{d_\theta,d'_\theta} = \sigma^2_{n,d_\theta}\,\mathbb{1}_{d_\theta=d'_\theta}$
7          proposals, $\boldsymbol{X}^{P,m,k,i}_n \sim f_{\boldsymbol{X}_n|\boldsymbol{X}_{n-1}}\big(\,\cdot \mid \boldsymbol{X}^{F,m,k,i}_{n-1}\,;\Theta^{P,m,k,i}_n\big)$
8          unit measurement density, $w^{k,i}_{u,n} = f_{Y_{u,n}|X_{u,n}}\big(y^*_{u,n} \mid X^{P,m,k,i}_{u,n}\,;\Theta^{P,m,k,i}_n\big)$
9          local prediction weights, $w^{P,k,i}_{u,n} = \prod_{(\tilde{u},\tilde{n})\in B_{u,n}} w^{k,i}_{\tilde{u},\tilde{n}}$
10         parameter log likelihoods, $r^k_n = \sum_{u=1}^{U} \log\left(\frac{\sum_{i=1}^{\mathcal{I}} w^{k,i}_{u,n}\, w^{P,k,i}_{u,n}}{\sum_{\tilde{i}=1}^{\mathcal{I}} w^{P,k,\tilde{i}}_{u,n}}\right)$ for $k$ in $1:K$,
11         Select the highest $pK$ weights: find $s$ with
           $\{s(1),\dots,s(pK)\} = \{k : \sum_{\tilde{k}=1}^{K} \mathbf{1}\{r^{\tilde{k}} > r^k\} < pK\}$
12         Make $1/p$ copies of successful parameters, $\Theta^{F,m,k}_n = \Theta^{F,m,s(\lceil pk\rceil)}_n$ for $k$ in $1:K$
13         Set $\boldsymbol{X}^{F,m,k,i}_n = \boldsymbol{X}^{P,m,s(\lceil pk\rceil),i}_n$
14     end
15 end
```

output: Iterated UBF parameter swarm: $\Theta^{F,M,1:K}_N$

Monte Carlo maximum likelihood estimate: $\frac{1}{K}\sum_{k=1}^{K}\Theta^{F,M,1:K}_N$.

complexity: $\mathcal{O}(MK\mathcal{I}UN)$

an unadapted bagged filter (UBF) with perturbed parameters. Note that UBF is the special case of ABF with $J = 1$. UBF and IUBF were found to be effective on the measles model (Ionides *et al.* 2021) and `iubf` was developed with this application in mind. This algorithm makes with K copies of the parameter set, and iteratively perturbs the parameter set while evaluate a conditional likelihood at each observation time using UBF. In each observation time, IUBF selects perturbed parameter sets yielding the top p quantile of the likelihoods.

4.5. Inference algorithms inherited from pomp

Objects of class '`spatPomp`' inherit methods for inference from class '`pomp`' objects implemented in the **pomp** package. As discussed earlier, the IF2 algorithm (Ionides *et al.* 2015) has been used for maximum likelihood parameter estimation in numerous applications. IF2 can be used to check the capabilities of newer and more scalable inference methods on smaller examples for which it is known to be effective. Extensions for Bayesian inference of the currently implemented high-dimensional particle filter methods (GIRF, ABF, EnKF, BPF) are not yet available. Bayesian inference is available in **spatPomp** using the approximate Bayesian computing (ABC) method inherited from **pomp**, `abc()`. ABC has previously been used for spatiotemporal inference (Brown *et al.* 2018) and can also serve as a baseline method. However, ABC is a feature-based method that may lose substantial information compared to full-information methods that work with the likelihood function rather than a summary statistic.

5. Demonstrating data analysis tools on a toy model

We illustrate key capabilities of **spatPomp** using the `bm10` model for correlated Brownian motion introduced in Sec.2.2. This allows us to demonstrate a data analysis in a simple context where we can compare results with a standard particle filter as well as validate all methods against the exact solutions which are analytically available. To define the model mathematically, consider spatial units $1, \dots, U$ located evenly around a circle, where $\text{dist}(u, \tilde{u})$ is the circle distance,

$$\text{dist}(u, \tilde{u}) = \min\left(|u - \tilde{u}|, |u - \tilde{u} + U|, |u - \tilde{u} - U|\right).$$

The latent process is a U-dimensional Brownian motion $\boldsymbol{X}(t)$ having correlation that decays with distance. Specifically,

$$dX_u(t) = \sum_{\tilde{u}=1}^{U} \rho^{\text{dist}(u,\tilde{u})}\, dW_{\tilde{u}}(t),$$

where $W_1(t), \dots, W_U(t)$ are independent Brownian motions with infinitesimal variance σ^2, and $|\rho| < 1$. Using the notation in Section 2, we suppose our measurement model for discrete-time observations of the latent process is

$$Y_{u,n} = X_{u,n} + \eta_{u,n}$$

where $\eta_{u,n} \overset{\text{iid}}{\sim} \text{Normal}(0, \tau^2)$. The model is completed by providing the initial conditions, $\{X_u(0), u \in 1 : U\}$, which are specified as parameters. The parameters for `bm10` are

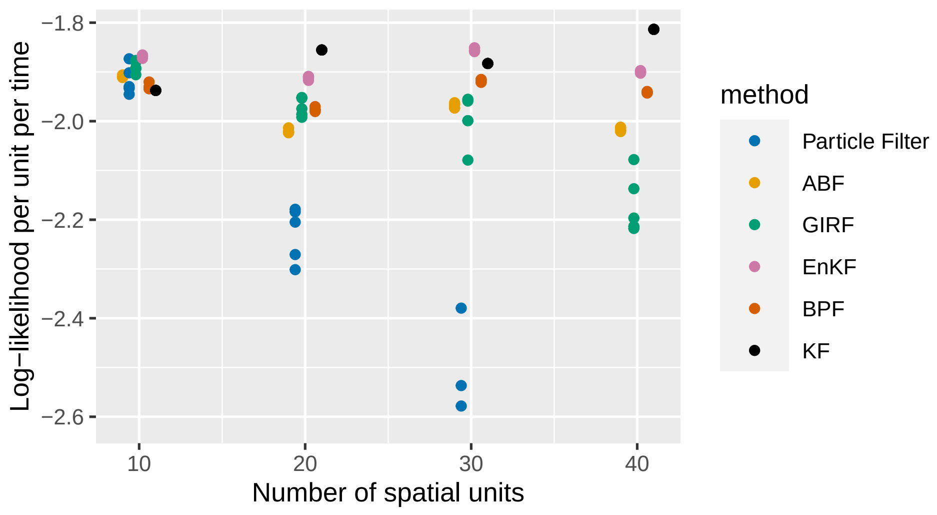

Figure 4 – Log-likelihood estimates for 5 replications of ABF, BPF, EnKF, GIRF and particle filter on correlated Brownian motions of various dimensions. The Kalman filter (KF) provides the exact likelihood in this case.

```
R> coef(bm10)
  rho sigma   tau  X1_0  X2_0  X3_0  X4_0  X5_0  X6_0  X7_0  X8_0
  0.4   1.0   1.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
 X9_0 X10_0
  0.0   0.0
```

5.1. Computing the likelihood

The `bm10` object contains all necessary model components for likelihood evaluation using the four algorithms described in Section 3. For example,

```
R> girf(bm10,Np=1000,Nguide=50,Ninter=5,lookahead=1)
R> bpfilter(bm10, Np=1000, block_size=2)
R> enkf(bm10, Np=1000)
```

generates objects of class '`girfd_spatPomp`', '`bpfiltered_spatPomp`' and '`enkfd_spatPomp`' respectively. A `plot` method provides diagnostics, and the resulting log-likelihood estimate is extracted by `logLik`.

Even for methods designed to be scalable, Monte Carlo variance can be expected to grow with the size of the dataset, and approximations used to enhance scalability may result in bias. Figure 4 investigates how the accuracy of the likelihood estimate scales with U for the `bm` model. Since class '`spatPomp`' inherits from class '`pomp`', we can compare **spatPomp** methods against the `pfilter` algorithm from **pomp**. We see that the performance of `pfilter` rapidly degrades as dimension increases, whereas the **spatPomp** methods scale better. On this Gaussian problem, the exact likelihood is available via the Kalman filter, and EnKF is almost exact since the Gaussian approximation used to construct its update rule is correct.

Table 2 – Comparison of computational resources of the filtering algorithms

Method	Resources (core-minutes)	Particles (per replicate)	Replicates	Guide particles	Lookahead
Particle Filter	0.014	2000	-	-	-
ABF	0.46	100	500	-	-
GIRF	0.20	500	-	50	1
EnKF	0.016	2000	-	-	-
BPF	0.022	2000	-	-	-

Computing resources used by each algorithm for Figure 4 are given in Table 2. Each algorithm was allowed to use 10 central processing unit (CPU) cores to evaluate all the likelihoods and the algorithmic settings were fixed as shown in the table. CPU time is not necessarily the only relevant consideration, for example, when applying `pfilter` with a large number of units and a complex model, memory constraints rather than CPU requirements may limit the practical number of particles. By contrast, ABF has a high CPU requirement but it parallelizes easily and so it can take advantage of distributed resources.

The time-complexity of GIRF is quadratic in U, due to the intermediate time step loop shown in the pseudocode in Section 3.1, whereas the other algorithms scale linearly with U for a fixed algorithmic setting. However, a positive feature of GIRF is that it shares with PF the property that it targets the exact likelihood, i.e., it is consistent for the exact log likelihood as the number of particles grows and the Monte Carlo variance approaches zero. GIRF may be a practical algorithm when the number of units prohibits PF but permits effective use of GIRF. EnKF and BPF generally run the quickest and require the least memory. However, the Gaussian and independent blocks assumptions, respectively, of the two algorithms must be reasonable to obtain likelihood estimates with low bias. On a new problem, it is advantageous to compare various algorithms to reveal unexpected limitations of the different approximations inherent in each algorithm.

5.2. Parameter inference

The correlated Brownian motions example also serves to illustrate parameter inference using IGIRF. Suppose we have data from the correlated 10-dimensional Brownian motions model discussed above. We consider estimation of the parameters σ, τ and ρ when that the initial conditions, $\{X_u(0), u \in 1\!:\!U\}$, are known to be zero. We demonstrate a search started at

```
R> start_params <- c(rho = 0.8, sigma = 0.4, tau = 0.2,
+    X1_0 = 0, X2_0 = 0, X3_0 = 0, X4_0 = 0, X5_0 = 0,
+    X6_0 = 0, X7_0 = 0, X8_0 = 0, X9_0 = 0, X10_0 = 0)
```

We start with a quick test of `igirf`. Note that the perturbation standard deviation to zero for the initial value parameters, so that we only estimate the parameters of interest.

```
R> i <- 2
R> ig1 <- igirf(
+    bm10,
+    params=start_params,
+    Ngirf=switch(i,2,50),
+    Np=switch(i,10,1000),
+    Ninter=switch(i,2,5),
+    lookahead=1,
+    Nguide=switch(i,5,50),
+    rw.sd=rw.sd(rho=0.02,sigma=0.02,tau=0.02),
+    cooling.type = "geometric",
+    cooling.fraction.50=0.5
+  )
```

Here, we use a computational intensity variable, `i`, to switch between algorithmic parameter settings. For debugging, testing and code development we use `i=1`. For a final version of the manuscript, we use `i=2`.

`ig1` is an object of class '`igirfd_spatpomp`' which inherits from class '`girfd_spatpomp`'. A useful diagnostic of the parameter search is a plot of the change of the parameter estimates during the course of an `igirf()` run. Each iteration within an `igirf` run provides a parameter estimate and a likelihood evaluation at that estimate. The `plot` method for a class '`igirfd_spatPomp`' object shows the convergence record of parameter estimates and their likelihood evaluations. As shown in Figure 5, this `igirf` search has allowed us to explore the parameter space and climb significantly up the likelihood surface to within a small neighborhood of the maximum likelihood. The run took 13.2 minutes on one CPU core for this example with 10 spatial units. For larger models, one may require starting multiple searches of the parameter space at various starting points by using parallel runs of `igirf()` on a larger machine with multiple cores.

The log-likelihood plotted in Figure 5, and that computed by `logLik(ig1)`, correspond to the perturbed model. These should be recomputed to obtain a better estimate for the unperturbed model. Different likelihood evaluation methods can be applied, as shown in Section 5.1, to investigate their comparative strengths and weaknesses. For `bm10`. the model is linear and Gaussian and so the maximum likelihood estimate of our model and the likelihood at this estimate can be found numerically using the Kalman filter. The maximum log likelihood is -373.0, whereas the likelihood obtained by `ig1` is -374.2. This shortfall is a reminder that Monte Carlo optimization algorithms should usually be replicated, and should be used with inference methodology that accommodates Monte Carlo error, as discussed in Section 5.3.

5.3. Monte Carlo profiles

Proper interpretation of a parameter estimate requires uncertainty estimates. For instance, we may be interested in estimating confidence intervals for the coupling parameters of spatiotemporal models. These are parameters that influence the strength of the dependence between the latent dynamics in different spatial units. In our correlated Brownian motions example, ρ plays this role. The dependence between any two units is moderated by the distance between the units and the value of ρ.

We can often estimate confidence intervals for parameters like τ and σ which drive the dynamics of each spatial unit. However, coupling parameters can be hard to detect because any signal can be overwhelmed by the inevitably high variance estimates of high-dimensional

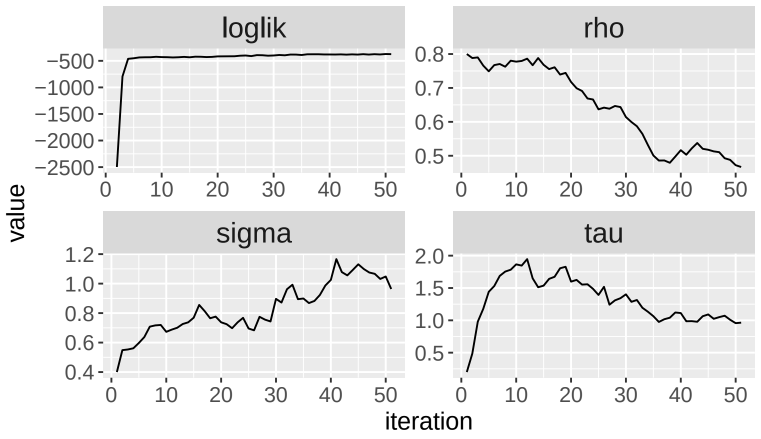

Figure 5 – The output of the `plot()` method on the object of class '`igirfd_spatPomp`' that encodes our model for correlated Brownian motions produces convergence traces for ρ, σ and τ, and the corresponding log likelihoods. Over 50 iterations `igirf()` has allowed us to get within a neighborhood of the maximum likelihood.

models. Full-information inference methods like IGIRF which are able to mitigate high variance issues in the filtering step can allow us to extract what limited information is available on coupling parameters like ρ. Here we will construct a profile likelihood for ρ with a 95% confidence interval that adjusts for Monte Carlo error.

A profile over a model parameter is a collection of maximized likelihood evaluations at a range of values of the profiled parameter. For each fixed value of this parameter, we maximize the likelihood over all the other parameters. We often use multiple different starting points for each fixed value of the profiled parameter.

Let us first design our profile over ρ by setting the bounds over all other model parameters from which we will draw starting values for likelihood maximization. We use the **pomp** function `profile_design` to construct a data frame with rows being starting values at random from a "hyperbox" subject to `rho` fixed at a sequence of values. We take advantage of the `partrans` method defined by the `partrans` argument to `spatPomp`, defined here as

```
R> partrans = parameter_trans(log = c("sigma", "tau"), logit = c("rho"))
```

This enables us to build the box on a transformed scale designed to make sense for additive parameter perturbations.

```
R> theta_lo_trans <- partrans(bm10,coef(bm10),dir="toEst") - log(2)
R> theta_hi_trans <- partrans(bm10,coef(bm10),dir="toEst") + log(2)
R> profile_design(
+    rho=seq(from=0.2,to=0.6,length=10),
+    lower=partrans(bm10,theta_lo_trans,dir="fromEst"),
+    upper=partrans(bm10,theta_hi_trans,dir="fromEst"),
+    nprof=switch(i,2,10)
+  ) -> pd
```

The argument `nprof` sets the number of searches, each started at a random starting point, for each value of the profiled parameter, `rho`. We can now run `igirf` for each search. We run these jobs in parallel using `foreach` and `%dopar%` from the `foreach` package (Wallig and Weston 2020) and collecting all the results together using `bind_rows` from `dplyr` (Wickham *et al.* 2020). Once we get a final parameter estimate from each `igirf` run, we can estimate the likelihood at this point by running, say, `enkf()`, 10 times and appropriately averaging the resulting log likelihoods.

```
R> foreach (p=iter(pd,"row"),.combine=dplyr::bind_rows) %dopar% {
+    library(spatPomp)
+    ig2 <- igirf(ig1,params=p,rw.sd=rw.sd(sigma=0.02,tau=0.02))
+    ef <- replicate(switch(i,2,10),enkf(ig2,Np=switch(i,50,2000)))
+    ll <- sapply(ef,logLik)
+    ll <- logmeanexp(ll,se=TRUE)
+    data.frame(as.list(coef(ig2)),loglik=ll[1],loglik.se=ll[2])
+  } -> rho_prof
```

Here, calling `igirf` on `igirf_out` imports all the previous algorithmic settings except for those that we explicitly modify. Each row of `rho_prof` now contains a parameter estimate and corresponding log likelihood from running `igirf` on a starting parameter in `pd`. We now construct a a 95% Monte Carlo adjusted profile confidence interval (Ionides *et al.* 2017) for ρ.

```
R> rho_mcap <- mcap(rho_prof[,"loglik"],parameter=rho_prof[,"rho"])
R> rho_mcap$ci
[1] 0.2524525 0.4830831
```

Note that the data in `bm10` are generated from a model with $\rho = 0.4$.

6. A spatiotemporal model of measles transmission

A **spatPomp** data analysis may consist of the following major steps. First, we obtain data, postulate a class of models that could have generated the data and bring these two pieces together via a call to `spatPomp()`. Second, we employ the tools of likelihood-based inference: evaluating the likelihood at specific parameter sets, maximizing likelihoods under the postulated class of models, constructing Monte Carlo adjusted confidence intervals, or performing hypothesis tests by comparing maximized likelihoods in a constrained region of parameter space with maximized likelihoods in the unconstrained parameter space. Third, we criticize our model by comparing simulations to data, or by considering rival models. In this section, we focus on the first step, showing how to bring data and models together via a metapopulation compartment model for measles dynamics in the 6 largest cities in England in the pre-vaccine era. Tools for the second step have been covered in Sections 3 and 4. The third step benefits from the flexibility of the large model class supported by **spatPomp**.

Compartment models for population dynamics divide up the population into categories (called *compartments*) which are modeled as homogeneous. The rate of flow of individuals between a pair of compartments may depend on the count of individuals in other compartments.

Compartment models have widespread scientific applications, especially in the biological and health sciences (Bretó *et al.* 2009). Spatiotemporal compartment models can be called patch models or metapopulation models in an ecological context, since the full population is divided into a "population" of sub-populations. We develop a spatiotemporal model for disease transmission dynamics of measles within and between multiple cities, based on the model of Park and Ionides (2020) which adds spatial interaction to the compartment model presented by He *et al.* (2010). We use this example to demonstrate how to construct spatiotemporal compartment models in **spatPomp**. The `measles()` function in **spatPomp** constructs such an object, and here we describe key steps in this construction.

6.1. Mathematical model for the latent process

We first define the model mathematically, starting with a description of the coupling, corresponding here to travel between cities. Let $v_{u\tilde{u}}$ denote the number of travelers from city u to $\tilde{u}$. Here, $v_{u\tilde{u}}$ is constructed using the gravity model of Xia *et al.* (2004):

$$v_{u\tilde{u}} = g \cdot \frac{\overline{\text{dist}}}{\bar{p}^2} \cdot \frac{p_u \cdot p_{\tilde{u}}}{\text{dist}(u,\tilde{u})},$$

where $\text{dist}(u,\tilde{u})$ denotes the distance between city u and city $\tilde{u}$, p_u is the average across time of the census population $P_u(t)$ for city u, $\bar{p}$ is the average of p_u across cities, and $\overline{\text{dist}}$ is the average of $\text{dist}(u,\tilde{u})$ across pairs of cities. The gravitation constant g is scaled with respect to $\bar{p}$ and $\overline{\text{dist}}$.

The measles model divides the population of each city into susceptible, S, exposed, E, infectious, I, and recovered/removed, R, compartments. The number of individuals in each compartment for city u at time t are denoted by $S_u(t)$, $E_u(t)$, $I_u(t)$, and $R_u(t)$. The latent state is $\boldsymbol{X}(t) = \big(X_1(t),\dots,X_U(t)\big)$ with $X_u(t) = \big(S_u(t),E_u(t),I_u(t),R_u(t)\big)$. The dynamics of the latent state can be written in terms of flows between compartments, together with flows into and out of the system, as follows:

$$\left.\begin{array}{rclclcl} dS_u(t) &=& dN_{BS,u}(t) &-& dN_{SE,u}(t) &-& dN_{SD,u}(t) \\ dE_u(t) &=& dN_{SE,u}(t) &-& dN_{EI,u}(t) &-& dN_{ED,u}(t) \\ dI_u(t) &=& dN_{EI,u}(t) &-& dN_{IR,u}(t) &-& dN_{ID,u}(t) \end{array}\right\} \quad \text{for } u=1,\dots,U.$$

Here, $N_{SE,u}(t)$, $N_{EI,u}(t)$, and $N_{IR,u}(t)$ are counting process corresponding to the cumulative number of individuals transitioning between the compartments identified by the subscripts. The recruitment of susceptible individuals into city u is denoted by the counting process $N_{BS,u}(t)$, primarily modeling births. Each compartment also has an outflow, written as a transition to D, primarily representing death, which occurs at a constant per-capita rate μ. The number of recovered individuals $R_u(t)$ in city u is defined implicitly from $P_u(t) = S_u(t)+E_u(t)+I_u(t)+R_u(t)$. $R_u(t)$ plays no direct role in the dynamics, beyond accounting for individuals not in any of the other classes.

To define the Markov model, we specify a rate for each counting process. Thus, μ_{EI} is the rate at which an individual in E progresses to I, and $1/\mu_{EI}$ is called the mean disease latency. Similarly, $1/\mu_{IR}$ is the mean infectious period. These two quantities are parameters of the model, and are considered to be constant across time and space. The mortality rates are fixed at $\mu_{SD} = \mu_{ED} = \mu_{ID} = \mu_{RD} = \mu_D$ with life expectancy $1/\mu_D = 50$yr. The birth rate, $\mu_{BS,u}$, is treated as a covariate determined by public records for each city, u. We treat $\mu_{BS,u}$ as the known population rate of births, whereas all other rates are defined per capita. The disease transmission rate, $\mu_{SE,u}$, is parameterized as

$$\mu_{SE,u}(t) = \overline{\beta}\,\mathrm{seas}(t)\left[\frac{I_u(t)}{P_u(t)} + \sum_{\tilde{u}\neq u}\frac{v_{u\tilde{u}}}{P_u(t)}\left\{\frac{I_{\tilde{u}}(t)}{P_{\tilde{u}}(t)} - \frac{I_u(t)}{P_u(t)}\right\}\right]\frac{d\Gamma_{SE,u}}{dt},$$

where the mean transmission rate, $\overline{\beta}$, is parameterized as $\overline{\beta} = \mathcal{R}_0(\mu_{IR} + \mu_D)$ where $\mathcal{R}_0$ is the basic reproduction rate; seas(t) is a periodic step function taking value $1 - A$ during school vacations and $1 + 0.381\,A$ during school terms, defined so that the average value of seas(t) is 1; the multiplicative white noise $d\Gamma_{SE,u}/dt$ is a derivative of a gamma process $\Gamma_{SE,u}(t)$ having independent gamma distributed increments with $\mathsf{E}[\Gamma_{SE,u}(t)] = t$ and $\mathrm{Var}[\Gamma_{SE,u}(t)] = \sigma^2_{SE}t$. We call σ^2_{SE} the infinitesimal variance of the noise. The formal meaning of $d\Gamma_{SE,u}/dt$ as white noise on the rate of a Markov chain was developed by Bretó *et al.* (2009) and Bretó and Ionides (2011). In brief, an Euler numerical solution depends on the rate function integrated over a small time interval of length Δt. The integrated noise process is an increment of the gamma process, and these increments are independent gamma random variables. When the continuous time Markov chain is equal to the limit of the Euler solutions, as $\Delta t \to 0$, then these solutions for provide a practical approach to working with the model. An Euler step for such a solution is defined via the Csnippet for `rprocess`, below. This code involves use of the `reulermultinom` function, which is the C interface to the R function `reulermultinom` provided by **pomp**. It keeps track of all the rates for possible departures from a compartment. The gamma white noise in these rates is added using the `rgammawn` function, which is also defined by **pomp** in both C and R.

Multiplicative white noise provides a way to model over-dispersion, a phenomenon where data variability is larger than can be explained by binomial or Poisson approximations. Over-dispersion on a multiplicative scale is also called environmental stochasticity, or logarithmic noise, or extra-demographic stochasticity. Over-dispersion is well established for generalized linear models (McCullagh and Nelder 1989) and has become increasingly apparent for compartment models as methods have become available to address it Bjørnstad and Grenfell (2001); He *et al.* (2010); Stocks *et al.* (2020).

6.2. Mathematical model for the measurement process

The observations for city u are bi-weekly reports of new cases. We model the total new cases in an interval by keeping track of transitions from I to R, since we expect that identified cases will typically be isolated from susceptible individuals. Therefore, we introduce a new latent variable, defined at observation times as

$$C_{u,n} = N_{IR,u}(t_n) - N_{IR,u}(t_{n-1}).$$

To work with $C_{u,n}$ in the context of a SpatPOMP model, we note that this variable has Markovian dynamics corresponding to a continuous time variable $C_u(t)$ satisfying $dC_u(t) = dN_{IR,u}(t)$ with the additional property that we set $C_u(t) = 0$ immediately after an observation time. To model the observation process, we define $Y_{u,n}$ as a normal approximation to an over-dispersed binomial sample of $C_{u,n}$ with reporting rate ρ. Specifically, conditional on $C_{u,n} = c_{u,n}$,

$$Y_{u,n} \sim \mathrm{Normal}\big[\rho\, c_{u,n}, \rho\,(1-\rho)\,c_{u,n} + \tau^2\rho^2 c^2_{u_n}\big],$$

where τ is a measurement overdispersion parameter.

6.3. Construction of a measles spatPomp object

The construction of class ‘`spatPomp`’ objects is similar to the construction of class ‘`pomp`’ objects discussed by King *et al.* (2016). Here, we focus on the distinctive features of SpatPOMP models. Suppose for our example below that we have bi-weekly measles case counts from $U = 10$ cities in England as reported by Dalziel *et al.* (2016) in the object `measles_cases` of class ‘`data.frame`’. Each city has about 15 years (391 bi-weeks) of data with no missing data. The first three rows of this data are shown here. We see the column corresponding to time is called `year` and is measured in years (two weeks is equivalent to 0.038 years).

```
 year       city cases
 1950     LONDON    96
 1950 BIRMINGHAM   179
 1950  LIVERPOOL   533
```

We can construct a `spatPomp` object by supplying three minimal requirements in addition to our data above: the column names corresponding to the units labels (`‘city’`) and observation times (`‘year’`) and the time at which the latent dynamics are supposed to begin. Here we set this to two weeks before the first recorded observations.

```
R> measles6 <- spatPomp(
+    data=measles_cases,
+    units='city',
+    times='year',
+    t0=min(measles_cases$year)-1/26
+  )
```

Internally, unit names are mapped to an index $1, \ldots, U$. The number assigned to each unit can be checked by inspecting their position in `unit_names(measles)`.

We proceed to collect together further model component, which we will add to `measles6` by another call to `spatPomp()`. First, we suppose that we have covariate time series for each city in a class ‘`data.frame`’ object called `measles_covar`. In this case, we have census population and lagged birthrate data. We consider lagged birthrate because we assume children enter the susceptible pool when they are old enough to go to school. The required format is similar to the `data` argument, though the times do not have to correspond to observation times since **spatPomp** will interpolate the covariates as needed.

```
 year       city       pop lag_birthrate
 1950     LONDON 3389306.0      70571.23
 1950 BIRMINGHAM 1117892.5      24117.23
 1950  LIVERPOOL  802064.9      19662.96
```

We now move on to specifying our model components as Csnippets. To get started, we define the movement matrix $(v_{u,\tilde{u}})_{u,\tilde{u} \in 1:U}$ as a global variable in `C` that will be accessible to all model components, via the `globals` argument to `spatPomp()`.

```
R> measles_globals <- spatPomp_Csnippet("
+    const double v_by_g[6][6] = {
+    {0,2.42,0.950,0.919,0.659,0.786},
+    {2.42,0,0.731,0.722,0.412,0.590},
+    {0.950,0.731,0,1.229,0.415,0.432},
+    {0.919,0.722,1.229,0,0.638,0.708},
+    {0.659,0.412,0.415,0.638,0,0.593},
+    {0.786,0.590,0.432,0.708,0.593,0}
+    };
+  ")
```

We now construct a Csnippet for initializing the latent process at time t_0, that is, drawing from $f_{\boldsymbol{X}_0}(\boldsymbol{x}_0\,;\theta)$. This is done using unit-specific IVPs, as discussed in Sections 2.4 and 2.5. Here, the IVPs are called `S1_0`,…,`S6_0`, `E1_0`,…,`E6_0`, `I1_0`,…,`I6_0` and `R1_0`,…,`R6_0`. These code for the initial value of the corresponding states, `S1`,…,`S6`, `E1`,…,`E6`, `I1`,…,`I6` and `R1`,…,`R6`. Additional book-keeping states, `C1`,…,`C6`, count accumulated cases during an observation interval and so are initialized to zero. The arguments `unit_ivpnames = c('S','E','I','R')` and `unit_statenames = c('S','E','I','R','C')` enable `spatPomp()` to expect these variables and define then as needed when compiling the Csnippets. Similarly, `unit_covarnames = 'pop'` declares the corresponding unit-specific population covariate. This is demonstrated in the following Csnippet specifying `rinit`.

```
R> measles_rinit <- spatPomp_Csnippet(
+    unit_statenames = c('S','E','I','R','C'),
+    unit_ivpnames = c('S','E','I','R'),
+    unit_covarnames = c('pop'),
+    code = "
+      for (int u=0; u<U; u++) {
+        S[u] = nearbyint(pop[u]*S_0[u]);
+        E[u] = nearbyint(pop[u]*E_0[u]);
+        I[u] = nearbyint(pop[u]*I_0[u]);
+        R[u] = nearbyint(pop[u]*R_0[u]);
+        C[u] = 0;
+      }
+    "
+  )
```

The `rprocess` Csnippet has to encode only a rule for a single Euler increment from the process model. Further, **spatPomp** provides C definitions of all parameters (e.g. `amplitude`) in addition to the state variables and covariates, so the user need only define additional variables used.

```
R> measles_rprocess <- spatPomp_Csnippet(
+    unit_statenames = c('S','E','I','R','C'),
+    unit_covarnames = c('pop','lag_birthrate'),
+    code = "
+      double beta, seas, i, dw, births, rate[6], trans[6];
+      int u,v;
+
+      // transmission rate with school term-time seasonality
+      t = (t-floor(t))*365.25;
+      if ((t>=7&&t<=100)||(t>=115&&t<=199)||(t>=252&&t<=300)||(t>=308&&t<=356))
+        seas = 1.0+amplitude*0.2411/0.7589; else seas = 1.0-amplitude;
+      beta = R0*(muIR+muD)*seas;
+
+      for (u = 0 ; u < U ; u++) {
+        // gravity model contacts within and between cities
+        i = I[u]/pop[u];
+        for (v=0; v < U ; v++) {
+          if(v != u)
+            i += g * v_by_g[u][v] * (I[v]/pop[v] - I[u]/pop[u]) / pop[u];
+        }
+
+        dw = rgammawn(sigmaSE,dt); // gamma white noise
+        rate[0] = beta*i*dw/dt;  // stochastic force of infection
+        rate[1] = muD;  // natural S death
+        rate[2] = muEI; // rate of ending of latent stage
+        rate[3] = muD; // natural E death
+        rate[4] = muIR; // recovery
+        rate[5] = muD; // natural I death
+
+        // transitions between compartments
+        births = rpois(lag_birthrate[u] * dt);
+        reulermultinom(2,S[u],&rate[0],dt,&trans[0]);
+        reulermultinom(2,E[u],&rate[2],dt,&trans[2]);
+        reulermultinom(2,I[u],&rate[4],dt,&trans[4]);
+
+        S[u] += births   - trans[0] - trans[1];
+        E[u] += trans[0] - trans[2] - trans[3];
+        I[u] += trans[2] - trans[4] - trans[5];
+        R[u] = pop[u] - S[u] - E[u] - I[u];
+        C[u] += trans[4];           // incidence, reported & unreported
+      }
+    "
+  )
```

The measurement model is chosen to allow for overdispersion relative to the binomial distribution with success probability ρ. Here, we show the Csnippet defining the unit measurement model. The `lik` variable is pre-defined and is set to the evaluation of the unit measurement density in either the log or natural scale depending on the value of `give_log`.

```
R> measles_dunit_measure <- spatPomp_Csnippet("
+    double m= rho*C;
+    double v = m*(1.0-rho+psi*psi*m);
+    lik = dnorm(cases,m,sqrt(v),give_log);
+  ")
```

The user may also directly supply `dmeasure` that returns the product of unit-specific measurement densities. The latter is needed to apply **pomp** functions which require `dmeasure` rather than `dunit_measure`. We create the corresponding Csnippet in `measles_dmeasure`, but do not display the code here. Next, we construct a Csnippet to code `runit_measure`,

```
R> measles_runit_measure <- spatPomp_Csnippet("
+    double cases;
+    double m= rho*C;
+    double v = m*(1.0-rho+psi*psi*m);
+    cases = rnorm(m,sqrt(v));
+    if (cases > 0.0) cases = nearbyint(cases);
+    else cases = 0.0;
+  ")
```

We also construct, but do not display, a Csnippet `measles_rmeasure` coding the class '`pomp`' version `rmeasure`.

Next, we build Csnippets for `eunit_measure` and `vunit_measure` which are required by EnKF and IEnKF. These have defined variables named `ey` and `vc` respectively, which should return $\mathsf{E}[Y_{u,n} \,|\, X_{u,n}]$ and $\mathrm{Var}[Y_{u,n} \,|\, X_{u,n}]$. For our measles model, we have

```
R> measles_eunit_measure <- spatPomp_Csnippet("ey = rho*C;")
R> measles_vunit_measure <- spatPomp_Csnippet("
+    double m = rho*C;
+    vc = m*(1.0-rho+psi*psi*m);
+  ")
```

It is convenient (but not necessary) to supply a parameter vector of scientific interest for testing the model. Here, we use a parameter vector with duration of infection and latent period both set equal to one week, following Xia *et al.* (2004), and the basic reproduction number set to $\mathcal{R}_0 = 30$. The gravitational constant, $g = 1500$, was picked to produce a qualitative visual match for simulations.

```
R> IVPs <- rep(c(0.032,0.00005,0.00004,0.96791),each=6) # SEIR fractions at t0
R> names(IVPs) <- paste0(rep(c('S','E','I','R'),each=6),1:6,"_0")
R> measles_params <- c(R0=30,amplitude=0.5,muEI=52,muIR=52,muD=0.02,
+    alpha=1,sigmaSE=0.01,rho=0.5,psi=0.1,g=1500,IVPs)
```

Special treatment is afforded to latent states that track accumulations of other latent states between observation times. These accumulator variables should be reset to zero at each observation time. The `unit_accumvars` argument provides a facility to specify the unit-level names of accumulator variables, extending the `accumvars` argument to `pomp()`. Here, there is one accumulator variable, `C`, which is needed since the reported cases in an interval is modeled to depend on the total number of new cases. The pieces of the SpatPOMP are now added to `measles6` via a call to `spatPomp`:

```
R> measles6 <- spatPomp(
+    data = measles6,
+    covar = measles_covar,
+    unit_statenames = c('S','E','I','R','C'),
+    unit_accumvars = c('C'),
+    paramnames = names(measles_params),
+    rinit = measles_rinit,
+    rprocess = euler(measles_rprocess, delta.t=1/365),
+    dunit_measure = measles_dunit_measure,
+    eunit_measure = measles_eunit_measure,
+    vunit_measure = measles_vunit_measure,
+    runit_measure = measles_runit_measure,
+    dmeasure = measles_dmeasure,
+    rmeasure = measles_rmeasure,
+    globals = measles_globals
+  )
```

Here, we have not filled the `skeleton` and `munit_measure` arguments, used by `girf` and `abfir`. These can be found in the **spatPomp** package source code for `measles()`.

6.4. Simulating measles data

In Figure 6, we compare a simulation with the data. to the data using the code below and the `plot()` method on the class '`spatPomp`' objects resulting from the simulation and the `measles6` object (which includes the true case reports) respectively. Epidemiological settings may be clearer when looking on the log scale, and so we use the `log=TRUE` argument to `plot()`. This figure shows some qualitative similarity between the simulations and the data, with opportunity for future work to investigate discrepancies.

7. Conclusion

The **spatPomp** package is both a tool for data analysis based on SpatPOMP models and a principled computational framework for the ongoing development of inference algorithms. To date, **spatPomp** has focused on algorithms with the plug-and-play property and on applications to metapopulation dynamics. By design, the package supports inclusion of new algorithms with and without the plug-and-play property, and models from diverse applications. Spatiotemporal data analysis using mechanistic models is a nascent topic, and future methodological developments are anticipated. Since the mission of **spatPomp** is to be a home for such analyses, the package developers welcome contributions and collaborations to further expand the functionality of **spatPomp**.

Complex models and large datasets can challenge available computational resources. With this in mind, key components of the **spatPomp** package and associated models are written

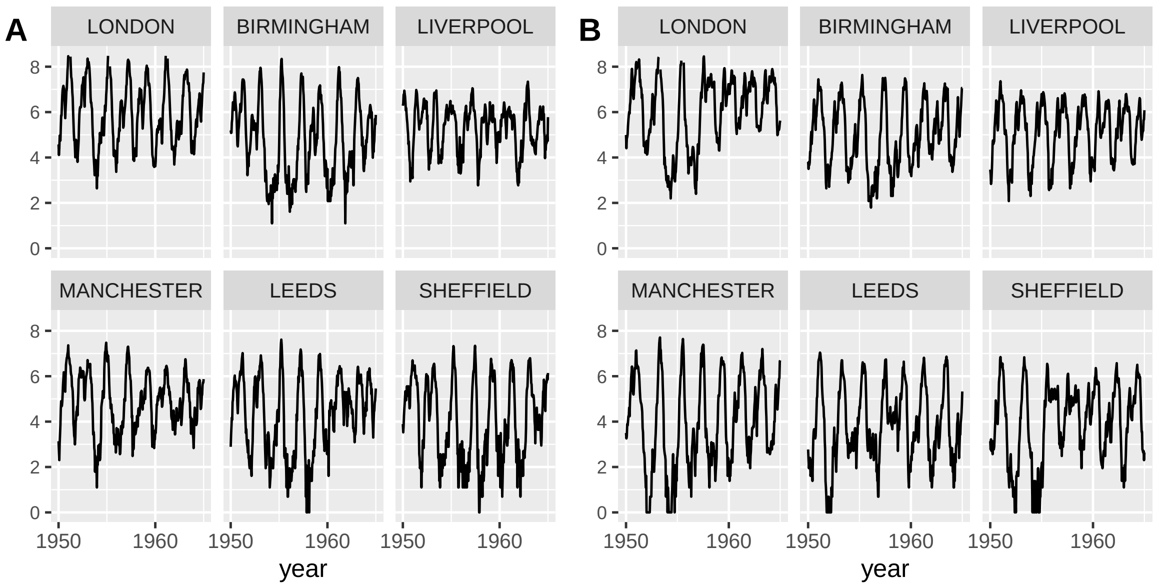

Figure 6 – A: reported measles cases in two week intervals for the six largest cities in England, `plot(measles6,log=TRUE)`. B: simulated data, `plot(simulate(measles6),log=TRUE)`. The vertical scale is `log(cases+1)`.

in C. This permits competitive performance on benchmarks (FitzJohn *et al.* 2020) within an R environment. The use of multi-core computing is helpful for computationally intensive methods. Two common computationally intensive tasks in **spatPomp** are the assessment of Monte Carlo variability and the investigation of the roles of starting values and other algorithmic settings on optimization routines. These tasks require only embarrassingly parallel computations and need no special discussion here.

Practical modeling and inference for metapopulation systems, capable of handling scientifically motivated nonlinear, non-stationary stochastic models, is the last open problem of the challenges raised by Bjørnstad and Grenfell (2001). Recent studies have reiterated the need for such methods, and made some progress toward attaining them (Becker *et al.* 2016; Li *et al.* 2020). The **spatPomp** provides a unifying framework, facilitating ongoing model development and methodological advances. Drawing reliable inferences by modeling complex systems require thoughtful data analysis in addition to powerful statistical tools (Saltelli *et al.* 2020), for which purpose **spatPomp** faciliates diagnostic plots and investigation of model variations.

Acknowledgments

This work was supported by National Science Foundation grants DMS-1761603 and DMS-1646108, and National Institutes of Health grants 1-U54-GM111274 and 1-U01-GM110712. We are grateful for constructive feedback from editors and reviewers at *Journal of Statistical Software*.

References

Anderson J, Hoar T, Raeder K, Liu H, Collins N, Torn R, Avellano A (2009). "The data assimilation research testbed: A community facility." *Bulletin of the American Meteorological Society*, **90**(9), 1283–1296. doi:10.1175/2009BAMS2618.1.

Arulampalam MS, Maskell S, Gordon N, Clapp T (2002). "A tutorial on particle filters for online nonlinear, non-Gaussian Bayesian tracking." *IEEE Transactions on Signal Processing*, **50**, 174–188. doi:10.1109/78.978374.

Bakker KM, Martinez-Bakker ME, Helm B, Stevenson TJ (2016). "Digital epidemiology reveals global childhood disease seasonality and the effects of immunization." *Proceedings of the National Academy of Sciences of the USA*, **113**(24), 6689–6694. doi:10.1073/pnas.1523941113.

Becker AD, Birger RB, Teillant A, Gastanaduy PA, Wallace GS, Grenfell BT (2016). "Estimating enhanced prevaccination measles transmission hotspots in the context of cross-scale dynamics." *Proceedings of the National Academy of Sciences*, **113**(51), 14595–14600. doi:10.1073/pnas.1604976113.

Becker AD, Wesolowski A, Bjørnstad ON, Grenfell BT (2019). "Long-term dynamics of measles in London: Titrating the impact of wars, the 1918 pandemic, and vaccination." *PLoS Computational Biology*, **15**(9), e1007305. doi:10.1371/journal.pcbi.1007305.

Bengtsson T, Bickel P, Li B (2008). "Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems." In T Speed, D Nolan (eds.), *Probability and Statistics: Essays in Honor of David A. Freedman*, pp. 316–334. Institute of Mathematical Statistics, Beachwood, OH. doi:10.1214/193940307000000518.

Bhadra A, Ionides EL, Laneri K, Pascual M, Bouma M, Dhiman RC (2011). "Malaria in northwest India: Data analysis via partially observed stochastic differential equation models driven by Lévy noise." *Journal of the American Statistical Association*, **106**, 440–451. doi:10.1198/jasa.2011.ap10323.

Bjørnstad ON, Grenfell BT (2001). "Noisy clockwork: Time series analysis of population fluctuations in animals." *Science*, **293**, 638–643. doi:10.1126/science.1062226.

Blackwood JC, Cummings DAT, Broutin H, Iamsirithaworn S, Rohani P (2013a). "Deciphering the impacts of vaccination and immunity on pertussis epidemiology in Thailand." *Proceedings of the National Academy of Sciences of the USA*, **110**, 9595–9600. doi:10.1073/pnas.1220908110.

Blackwood JC, Streicker DG, Altizer S, Rohani P (2013b). "Resolving the roles of immunity, pathogenesis, and immigration for rabies persistence in vampire bats." *Proceedings of the National Academy of Sciences of the USA*. doi:10.1073/pnas.1308817110.

Blake IM, Martin R, Goel A, Khetsuriani N, Everts J, Wolff C, Wassilak S, Aylward RB, Grassly NC (2014). "The role of older children and adults in wild poliovirus transmission." *Proceedings of the National Academy of Sciences of the USA*, **111**(29), 10604–10609. http://www.pnas.org/content/111/29/10604.full.pdf+html.

Bretó C (2014). "On idiosyncratic stochasticity of financial leverage effects." *Statistics & Probability Letters*, **91**, 20–26. doi:http://dx.xdoi.org/10.1016/j.spl.2014.04.003.

Bretó C, He D, Ionides EL, King AA (2009). "Time series analysis via mechanistic models." *Annals of Applied Statistics*, **3**, 319–348. doi:10.1214/08-AOAS201.

Bretó C, Ionides EL (2011). "Compound Markov counting processes and their applications to modeling infinitesimally over-dispersed systems." *Stochastic Processes and their Applications*, **121**, 2571–2591. doi:10.1016/j.spa.2011.07.005.

Brown GD, Porter AT, Oleson JJ, Hinman JA (2018). "Approximate Bayesian computation for spatial SEIR(S) epidemic models." *Spatial and Spatio-temporal Epidemiology*, **24**, 27–37. doi:10.1016/j.sste.2017.11.001.

Buhnerkempe MG, Prager KC, Strelioff CC, Greig DJ, Laake JL, Melin SR, DeLong RL, Gulland F, Lloyd-Smith JO (2017). "Detecting signals of chronic shedding to explain pathogen persistence: Leptospira interrogans in California sea lions." *Journal of Animal Ecology*, **86**(3), 460–472. doi:10.1111/1365-2656.12656.

Cappello C, De Iaco S, Posa D (2020). "covatest: An R Package for Selecting a Class of Space-Time Covariance Functions." *Journal of Statistical Software*, **94**(1), 1–42. doi:10.18637/jss.v094.i01.

Chambers JM (1998). *Programming with Data: A Guide to the S Language*. Springer Science & Business Media.

Dalziel BD, Bjørnstad ON, van Panhuis WG, Burke DS, Metcalf CJE, Grenfell BT (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:10.1371/journal.pcbi.1004655.

Del Moral P, Murray LM (2015). "Sequential Monte Carlo with highly informative observations." *Journal on Uncertainty Quantification*, **3**, 969–997. doi:10.1137/15M1011214.

Doucet A, Johansen A (2011). "A tutorial on particle filtering and smoothing: Fifteen years later." In D Crisan, B Rozovsky (eds.), *Oxford Handbook of Nonlinear Filtering*. Oxford University Press. URL https://warwick.ac.uk/fac/sci/statistics/staff/academic-research/johansen/publications/dj11.pdf.

Earn DJ, He D, Loeb MB, Fonseca K, Lee BE, Dushoff J (2012). "Effects of school closure on incidence of pandemic influenza in Alberta, Canada." *Annals of Internal Medicine*, **156**, 173–181. doi:10.7326/0003-4819-156-3-201202070-00005.

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**(C5), 10143–10162. doi:10.1029/94JC00572.

Evensen G, van Leeuwen PJ (1996). "Assimilation of geostat altimeter data for the Agulhas Current using the ensemble Kalman filter with a quasigeostrophic model." *Monthly Weather Review*, **124**, 58–96. doi:10.1175/1520-0493(1996)124<0085:AOGADF>2.0.CO;2.

FitzJohn RG, Knock ES, Whittles LK, Perez-Guzman PN, Bhatia S, Guntoro F, Watson OJ, Whittaker C, Ferguson NM, Cori A, Baguelin M, Lees JA (2020). "Reproducible parallel inference and simulation of stochastic state space models using odin, dust, and mcstate." *Wellcome Open Research*, **5**.

Genolini C (2008). "A (not so) short introduction to S4." *Technical report*, The R-Project for Statistical Computing. URL http://christophe.genolini.free.fr/webTutorial/S4tutorialV0-5en.pdf.

He D, Dushoff J, Day T, Ma J, Earn DJD (2013). "Inferring the causes of the three waves of the 1918 influenza pandemic in England and Wales." *Proceedings of the Royal Society of London, Series B*, **280**, 20131345. doi:10.1098/rspb.2013.1345.

He D, Ionides EL, King AA (2010). "Plug-and-play inference for disease dynamics: Measles in large and small towns as a case study." *Journal of the Royal Society Interface*, **7**, 271–283. doi:10.1098/rsif.2009.0151.

Ionides EL, Asfaw K, Park J, King AA (2021). "Bagged filters for partially observed interacting systems." *Journal of the American Statistical Association, pre-published online.* doi:10.1080/01621459.2021.1974867.

Ionides EL, Breto C, Park J, Smith RA, King AA (2017). "Monte Carlo profile confidence intervals for dynamic systems." *Journal of the Royal Society Interface*, **14**, 1–10. doi:10.1098/rsif.2017.0126.

Ionides EL, Nguyen D, Atchadé Y, Stoev S, King AA (2015). "Inference for dynamic and latent variable models via iterated, perturbed Bayes maps." *Proceedings of the National Academy of Sciences of the USA*, **112**(3), 719—-724. doi:10.1073/pnas.1410597112.

Ionides EL, Ning N, Wheeler J (2022). "An Iterated Block Particle Filter for Inference on Coupled Dynamic Systems with Shared and Unit-Specific Parameters." *Statistica Sinica, pre-published online.* doi:10.5705/ss.202022.0188.

Johansen AM, Doucet A (2008). "A note on the auxiliary particle filter." *Statistics & Probability Letters*, **78**, 1498–1504. doi:10.1016/j.spl.2008.01.032.

Kain MP, Childs ML, Becker AD, Mordecai EA (2021). "Chopping the tail: How preventing superspreading can help to maintain COVID-19 control." *Epidemics*, **31**, 100430. doi:10.1016/j.epidem.2020.100430.

Kantas N, Doucet A, Singh SS, Maciejowski J, Chopin N, *et al.* (2015). "On particle methods for parameter estimation in state-space models." *Statistical Science*, **30**(3), 328–351. doi:10.1214/14-STS511.

Katzfuss M, Stroud JR, Wikle CK (2020). "Ensemble Kalman methods for high-dimensional hierarchical dynamic space-time models." *Journal of the American Statistical Association*, **115**(530), 866–885. doi:10.1080/01621459.2019.1592753.

King AA, Ionides EL, Pascual M, Bouma MJ (2008). "Inapparent infections and cholera dynamics." *Nature*, **454**, 877–880. doi:10.1038/nature07084.

King AA, Nguyen D, Ionides EL (2016). "Statistical inference for partially observed Markov processes via the R Package pomp." *Journal of Statistical Software*, **69**, 1–43. doi:10.18637/jss.v069.i12.

Li R, Pei S, Chen B, Song Y, Zhang T, Yang W, Shaman J (2020). "Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (SARS-CoV-2)." *Science*, **368**(6490), 489–493. doi:10.1126/science.abb3221.

Lorenz EN (1996). "Predictability: A problem partly solved." *Proceedings of the Seminar on Predictability*, **1**, 1–18.

Marino JA, Peacor SD, Bunnell D, Vanderploeg HA, Pothoven SA, Elgin AK, Bence JR, Jiao J, Ionides EL (2019). "Evaluating consumptive and nonconsumptive predator effects on prey density using field time-series data." *Ecology*, **100**(3), e02583. doi:10.1002/ecy.2583.

Martinez-Bakker M, King AA, Rohani P (2015). "Unraveling the transmission ecology of polio." *PLoS Biology*, **13**(6), e1002172. doi:10.1371/journal.pbio.1002172.

McCullagh P, Nelder JA (1989). *Generalized Linear Models*. 2nd edition. Chapman and Hall, London.

Michaud N, de Valpine P, Turek D, Paciorek CJ, Nguyen D (2021). "Sequential Monte Carlo Methods in the nimble and nimbleSMC R Packages." *Journal of Statistical Software*, **100**, 1–39. doi:10.18637/jss.v100.i03.

Murray LM (2015). "Bayesian State-Space Modelling on High-Performance Hardware Using LibBi." *Journal of Statistical Software*, **67**(10), 1–36. doi:10.18637/jss.v067.i10.

Ng B, Peshkin L, Pfeffer A (2002). "Factored particles for scalable monitoring." *Proceedings of the 18th Conferenece on Uncertainty and Artificial Intelligence*, pp. 370–377. 1301.0590.

Ning N, Ionides EL (2021). "Iterated Block Particle Filter for High-dimensional Parameter Learning: Beating the Curse of Dimensionality." *arXiv:2110.10745*.

Park J, Ionides EL (2020). "Inference on high-dimensional implicit dynamic models using a guided intermediate resampling filter." *Statistics & Computing*, **30**, 1497–1522. doi:10.1007/s11222-020-09957-3.

Pitt MK, Shepard N (1999). "Filtering via simulation: Auxillary particle filters." *Journal of the American Statistical Association*, **94**, 590–599. doi:10.1080/01621459.1999.10474153.

Pons-Salort M, Grassly NC (2018). "Serotype-specific immunity explains the incidence of diseases caused by human enteroviruses." *Science*, **361**(6404), 800–803. doi:10.1126/science.aat6777.

R Core Team (1999). "Writing R extensions." *R Foundation for Statistical Computing*.

Ranjeva SL, Baskerville EB, Dukic V, Villa LL, Lazcano-Ponce E, Giuliano AR, Dwyer G, Cobey S (2017). "Recurring infection with ecologically distinct HPV types can explain high prevalence and diversity." *Proceedings of the National Academy of Sciences*, p. 201714712. doi:10.1073/pnas.1714712114.

Rebeschini P, van Handel R (2015). "Can local particle filters beat the curse of dimensionality?" *The Annals of Applied Probability*, **25**(5), 2809–2866. doi:10.1214/14-AAP1061.

Roy M, Bouma MJ, Ionides EL, Dhiman RC, Pascual M (2013). "The potential elimination of Plasmodium vivax malaria by relapse treatment: Insights from a transmission model and surveillance data from NW India." *PLoS Neglected Tropical Diseases*, **7**, e1979. doi:10.1371/journal.pntd.0001979.

Saltelli A, Bammer G, Bruno I, Charters E, Di Fiore M, Didier E, Nelson Espeland W, Kay J, Lo Piano S, Mayo D, Pielke R, Portaluri T, Porter TM, Puy A, Rafols I, Ravetz JR, Reinert E, Sarewitz D, Stark PB, Stirling A, van der Sluijs J, Vineis P (2020). "Five Ways to Ensure that Models Serve Society: a Manifesto." *Nature*, **582**, 428–484.

Shrestha S, Foxman B, Weinberger DM, Steiner C, Viboud C, Rohani P (2013). "Identifying the interaction between influenza and pneumococcal pneumonia using incidence data." *Science Translational Medicine*, **5**, 191ra84. doi:10.1126/scitranslmed.3005982.

Shrestha S, King AA, Rohani P (2011). "Statistical inference for multi-pathogen systems." *PLoS Computational Biology*, **7**, e1002135. doi:10.1371/journal.pcbi.1002135.

Sigrist F, Kunsch HR, Stahel WA (2015). "spate: An R package for spatio-temporal modeling with a stochastic advection-diffusion process." *Journal of Statistical Software*, **63**(14), 1–23. doi:10.18637/jss.v063.i14.

Snyder C, Bengtsson T, Morzfeld M (2015). "Performance bounds for particle filters using the optimal proposal." *Monthly Weather Review*, **143**(11), 4750–4761. doi:10.1175/MWR-D-15-0144.1.

Stocks T, Britton T, Höhle M (2020). "Model selection and parameter estimation for dynamic epidemic models via iterated filtering: application to rotavirus in Germany." *Biostatistics*, **21**(3), 400–416. doi:10.1093/biostatistics/kxy057.

Tong H (1990). *Non-linear Time Series: A Dynamical System Approach.* Oxford Science Publ., Oxford.

Wallig M, Weston S (2019). *doParallel: Foreach Parallel Adaptor for the 'parallel' Package.* R package version 1.0.15, URL https://CRAN.R-project.org/package=doParallel.

Wallig M, Weston S (2020). *foreach: Provides Foreach Looping Construct.* R package version 1.5.0, URL https://CRAN.R-project.org/package=foreach.

Wickham H (2019). *Advanced R.* CRC press. URL https://adv-r.hadley.nz/.

Wickham H, François R, Henry L, Müller K (2020). *dplyr: A Grammar of Data Manipulation.* R package version 1.0.0, URL https://CRAN.R-project.org/package=dplyr.

Wikle CK, Zammit-Mangion A, Cressie N (2019). *Spatio-temporal Statistics with R.* CRC Press.

Xia Y, Bjørnstad ON, Grenfell BT (2004). "Measles metapopulation dynamics: A gravity model for epidemiological coupling and dynamics." *American Naturalist*, **164**(2), 267–281. doi:10.1086/422341.

Affiliation:

Kidus Asfaw, Edward Ionides
University
Microsoft of Michigan
48109 Michigan, United States of America
E-mail: `kasfaw@umich.edu`, `ionides@umich.edu`
URL: `https://www.stat.lsa.umich.edu/~ionides/`

Aaron A. King
Department of Ecology & Evolutionary Biology
Center for the Study of Complex Systems
University of Michigan
48109 Michigan, United States of America
E-mail: `kingaa@umich.edu`
URL: `https://kinglab.eeb.lsa.umich.edu/`

Joonha Park
Department of Mathematics
University of Kansas
66045 Kansas, United States of America
E-mail: `j.park@ku.edu`
URL: `https://people.ku.edu/~j139p002`