Extensions to the pomp package and framework

Spencer J. Fox   Qianying (Ruby) Lin   Jesse Wheeler

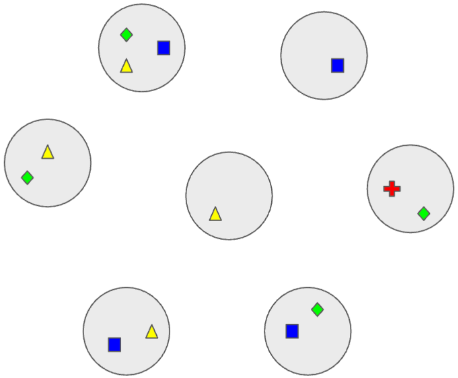
Package Extensions

Mathematically, POMP models are very versatile, and many of the ideas useful to the models described so far are also applicable in more general settings. This gives rise to a few useful extensions of the `pomp` package, which we try to describe here.

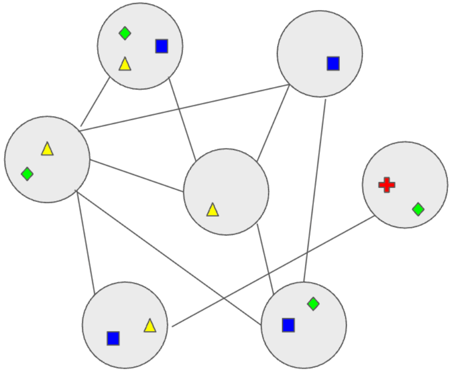
- Meta-population data and models
  - `panelPomp`
  - `spatPomp`
- Genomic / phylogenetic data
  - `phylopomp`

Each of these packages extend the `pomp` package to other useful cases.

Meta-population: Data and Models



panelPomp: data from related, independent systems.



spatPomp: data from a single interacting system.

Meta-population packages: Examples

Both packages require `pomp` to be installed, and the internal workings of both packages actually use many of the `pomp` functions. Thus, familiarity with `pomp` is a prerequisite.

- `panelPomp`:
  - We have *panel* or *longitudinal* data.
  - Several independent yet related systems. Information from all systems is useful, not just one location.
  - Simultaneously allows for features shared by each system as well as system unique features.
  - Example: endemic COVID-19 measured in 3 locations: New York, London, Atlanta.
- `spatPomp`:
  - The name comes from the idea that we have spatially explicit POMP models.
  - The data at multiple locations are thought of as location-specific measurements from the same system. That is, the dynamic systems underlying the data are connected.
  - Example: Modeling early stages of COVID-19 in China. Here, we might have measurements for many cities, but at first cases were only found in Wuhan.

PanelPomp: a collection of POMP models

A panelPomp model is really just a collection of pomp models. This is also how they are built:

```r
library(panelPomp)

mod1 <- pomp(..., params = c('p1' = 0.1, 'p2' = 1.2, 'p3' = 0.9))
mod2 <- pomp(..., params = c('p1' = 0.1, 'p2' = 1.1, 'p3' = 0.6))
mod3 <- pomp(..., params = c('p1' = 0.1, 'p2' = 1.5, 'p3' = 0.75))

ppomp <- panelPomp(
  object = list(mod1, mod2, mod3),
  shared = c("p1" = 0.1), # Shared-value parameters
  specific = c("p2", "p3") # Unit-Specific parameters
)
```

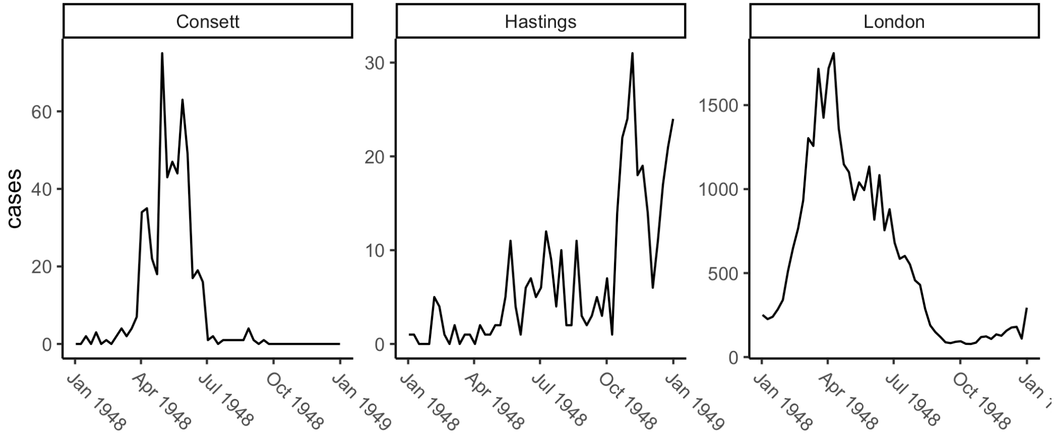
Measles Example

We can build off of the measles example by looking at UK measles from multiple cities, and building an SEIR model for the data. There is built in models and data in `panelPomp`

```r
measSIR <- panelMeasles(
  units = c("Consett", "London", "Hastings"),
  first_year = 1948,
  last_year = 1948
  )
```

Measles Example: Figure

```
plot(measSIR)
```



panelPomp: parameters

Parameter names in `panelPomp` have the following conventions. If the parameter is shared, it just is called by the name. If it is unit-specific, the name of the unit follows the name of the parameter: `<param>[<unit>]`.

```r
coef(measSIR)
```

```
                mu       sigma[Consett]       gamma[Consett]
          2.00e-02             4.26e+01             1.72e+02
      rho[Consett]            R0[Consett]   amplitude[Consett]
          6.50e-01             3.59e+01             2.00e-01
    alpha[Consett]          iota[Consett]      cohort[Consett]
          1.01e+00             7.31e-02             3.10e-01
      psi[Consett]           S_0[Consett]         E_0[Consett]
          4.06e-01             3.22e-02             1.83e-05
      I_0[Consett]           R_0[Consett]     sigmaSE[Consett]
          1.97e-05             9.68e-01             7.12e-02
   sigma[Hastings]        gamma[Hastings]        rho[Hastings]
```

panelPomp: shared vs unit-specific

A key feature of `panelPomp` objects is which parameters are shared, which are unit-specific. If parameters are shared, that means they have the same value for all units. We can inspect and modify which parameters are which using the functions `shared` and `specific`:

```r
shared(measSIR)
```

```
  mu
0.02
```

```r
shared(measSIR) <- c(shared(measSIR), 'alpha' = 1)
shared(measSIR)
```

```
   mu alpha
 0.02  1.00
```

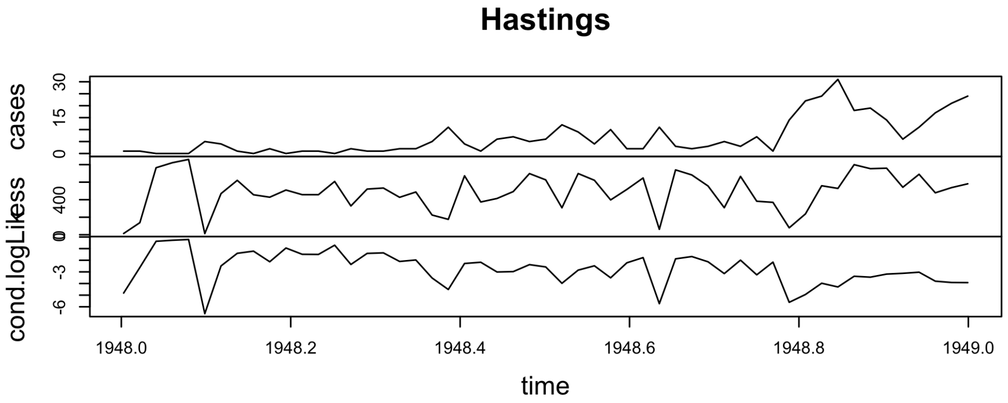
```r
specific(measSIR)
```

```
           Consett Hastings   London
```

panelPomp Functions

`panelPomp` is the easiest extension: If you can build one `pomp`, you can build multiple into a single `panelPomp` object. Existing functions and algorithms are similar as well!

```
pfilter(measSIR, Np = 1000) |> plot(unit = 'Hastings')
```

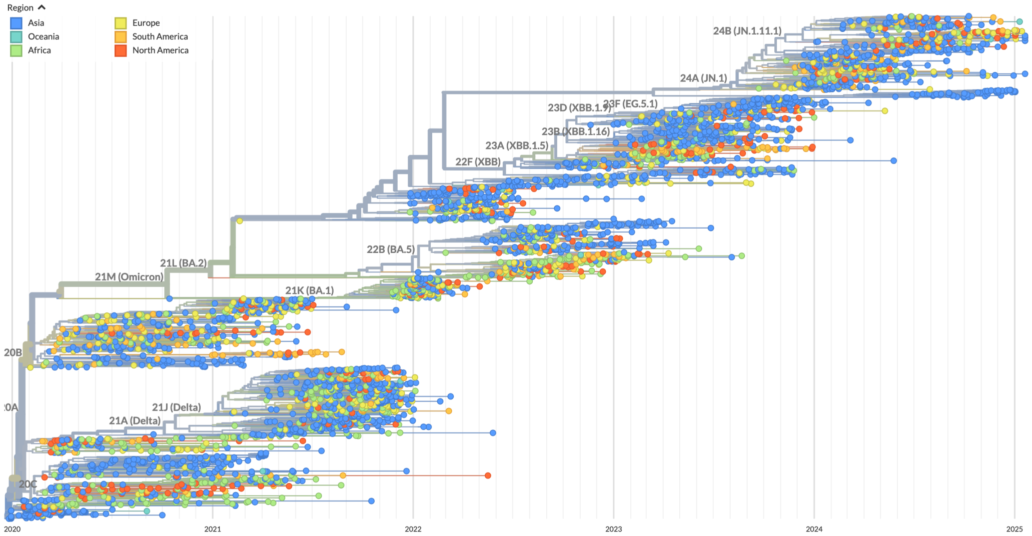


panelPomp: iterated filtering

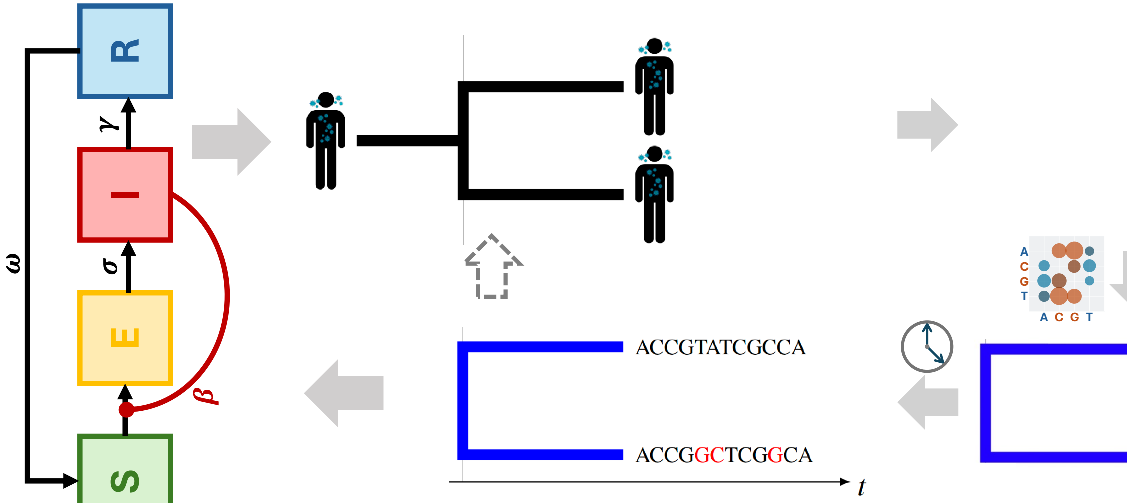
We need a new algorithm for conducting maximum likelihood, either the panel iterated filter (PIF), or marginalized panel iterated filter (MPIF). We don't even need to change the code!

```
mif2Out <- mif2(
  measSIR, rw.sd = rw_sd(rho = 0.02, R_0 = 0.02),
  Nmif = 10, Np = 200, cooling.fraction.50 = 0.5,
  block = TRUE  # block = TRUE does MPIF, usually best + faster.
  )
```

Background: genomic sequences & phylogenetic tree



Background: phylogenetics & phylodynamics



Installation

The github repo for `phylopomp` is on https://github.com/kingaa/phylopomp/.
We can install the latest released version from github using `devtools`.

```
library(devtools)
install_github("kingaa/phylopomp@0.14.8.0")
```

To confirm the package is installed successfully,

```
packageVersion("phylopomp")
```

```
[1] '0.14.9.1'
```

```
library(phylopomp)
```

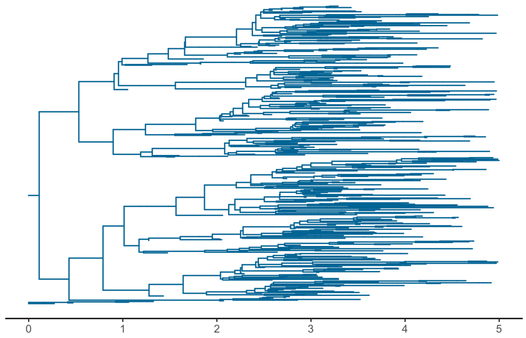
Simulation and inference

- The first useful function of `phylopomp` is to build customized models and simulate genealogies from it. Pre-defined models are included: linear birth-death model (`lbdp`), moran model, SIR, SEIR, two-class SIR model with super-infection (`si2r`), two-strain SIR model (`siir`), etc.
- The second is to infer the models given genealogies using `pomp`. This function is currently under development, while a few models are available: `lbdp`, `moran`, `sir`, and `seir`.

Example: simulation, pre-defined model I

```r
set.seed(1234)
simulate("SIR",Beta=2,gamma=1,psi=2,S0=1000,I0=5,time=2) |>
  simulate(Beta=5,gamma=2,time=5,psi=3) -> model.sir # update params
plot(model.sir)
```

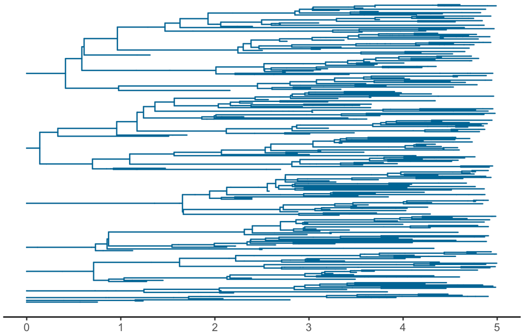
Example: simulation, pre-defined model II



Example: simulation, pre-defined model III

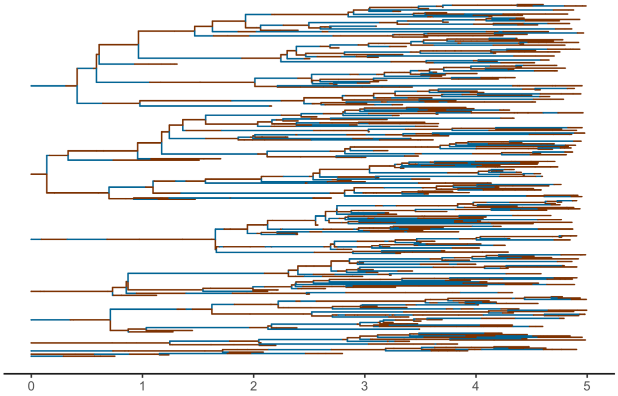
```r
set.seed(1234)
simulate("SEIR",Beta=2,sigma=2,gamma=1,psi=2,S0=1000,I0=5,time=2) |>
  simulate(Beta=5,gamma=2,time=5,psi=3) -> model.seir
plot(model.seir)
```

Example: simulation, pre-defined model IV



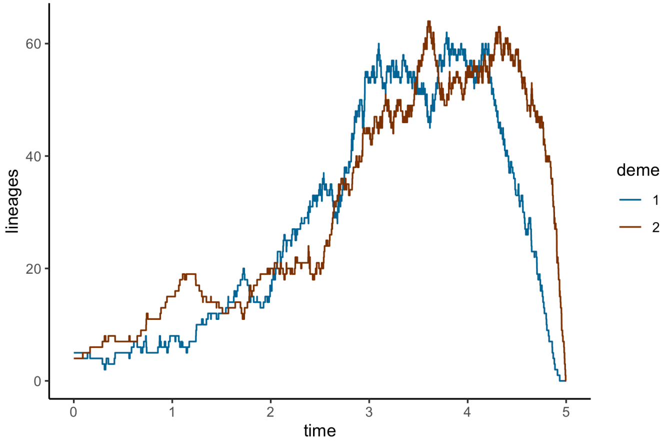
Example: simulation, pre-defined model V

```
plot(model.seir, obscure=FALSE)        # show E and I transitions
```



Example: simulation, pre-defined model VI

```
lineages(model.seir, obscure=FALSE) |> plot()
```



Example: inference, pre-defined model I

```r
library(pomp)
set.seed(1234)
# simulate a geneal object from an SIR model
simulate("SIR",Beta=3,gamma=1,psi=2,omega=1,S0=100,I0=5,time=10) -> x
# build a pomp object
x |>
  sir_pomp(
    Beta=3,gamma=1,psi=2,omega=1,
    S0=100,I0=5,R0=0
  ) -> po
po |> pfilter(Np=5000) -> pf

pf |> logLik()
```

Example: inference, pre-defined model II

```
[1] -326.5652
```

Example: inference, pre-defined model III

```r
set.seed(1234)
simulate("SEIRS",       # simulate a genealogy from an SEIR model
  Beta=4,sigma=1,gamma=1,psi=1,omega=1,
  S0=100,E0=3,I0=5,R0=100, time=5
) -> G
G |>
  seirs_pomp(
    Beta=4,sigma=1,gamma=1,psi=1,omega=1,
    S0=100,E0=3,I0=5,R0=100
  ) |> pfilter(Np=1000) |>
  replicate(n=20) |> concat() -> pf
pf |> logLik() |> logmeanexp(se=TRUE)
```

Example: inference, pre-defined model IV

```
        est          se
-168.856637    1.265223
```

License, acknowledgments, and links

- This lesson is prepared for the Simulation-based Inference for Epidemiological Dynamics module at the Summer Institute in Statistics and Modeling in Infectious Diseases, SISMID.
- The materials build on previous versions of this course and related courses.
- Licensed under the Creative Commons Attribution-NonCommercial license. Please share and remix non-commercially, mentioning its origin.
- Produced with R version 4.5.1 and pomp version 6.3.
- Compiled on 2025-07-23.

Back to Lesson
`R` code for this lesson