

Chapter 10. Statistical methodology for nonlinear partially observed Markov process models

Objectives

- ① To introduce students to the pomp package
- ② To explain how the components of a POMP model are encoded in this package
- ③ To give some experience in the use and manipulation of pomp objects.

Time series analysis via nonlinear partially observed Markov process (NL-POMP) models

Six problems of Bjornstad and Grenfell (Science, 2001): obstacles for ecological modeling and inference via nonlinear mechanistic models:

- ① Combining measurement noise and process noise.
- ② Including covariates in mechanistically plausible ways.
- ③ Continuous time models.
- ④ Modeling and estimating interactions in coupled systems.
- ⑤ Dealing with unobserved variables.
- ⑥ Modeling spatial-temporal dynamics.

Applications of NL-POMP models

- Modeling and inference via nonlinear mechanistic models arises throughout engineering, the sciences (social, biological and physical) and business.
- In finance, we considered a stochastic volatility example in Chapter 1.
- Infectious disease transmission dynamics can be highly nonlinear. Transmission arises when an infected individual contacts a susceptible individual, thus the rate of infections gets a nonlinear term:

Fraction of individuals infected

× Fraction of individuals susceptible to infection

- Each new infection depletes the pool of susceptible individuals. Without depletion of susceptibles, the fraction of individuals susceptible to infection is constant and the epidemic grows exponentially.
- Data on infectious diseases are generally limited to diagnosed cases. Much of the transmission dynamics cannot be directly observed.

Ecology and epidemiology as historical motivation for NL-POMP methods

- Infectious disease epidemiology has motivated developments in statistical methodology and software for NL-POMP models.
- Many other biological populations have similar nonlinearities: the population grows exponentially until limited by some constraint, such as a food resource or a predator. When the resource is used up, or the predator becomes abundant, the population crashes. Then a new cycle begins.
- Ecological systems and epidemiological systems have much in common. Disease systems can be relatively simple ecosystems with just two species: host and pathogen.

An algorithmic approach to inference for POMP models

Recall our notation for partially observed Markov process models. The latent process is $X_n = X(t_n)$ and we write $X_{0:N} = (X_0, \dots, X_N)$. The observation at time t_n is modeled by a random variable Y_n . The one-step transition density, $f_{X_n|X_{n-1}}(x_n|x_{n-1}; \theta)$, together with the measurement density, $f_{Y_n|X_n}(y_n|x_n; \theta)$ and the initial density, $f_{X_0}(x_0; \theta)$, specify the entire joint density via

$$f_{X_{0:N}, Y_{1:N}}(x_{0:N}, y_{1:N}; \theta) = f_{X_0}(x_0; \theta) \prod_{n=1}^N f_{X_n|X_{n-1}}(x_n|x_{n-1}; \theta) f_{Y_n|X_n}(y_n|x_n; \theta)$$

The marginal density for $Y_{1:N}$, evaluated at the data, $y_{1:N}$, is

$$f_{Y_{1:N}}(y_{1:N}; \theta) = \int f_{X_{0:N}, Y_{1:N}}(x_{0:N}, y_{1:N}; \theta) dx_{0:N}.$$

Algorithmic notation matching the mathematical notation for POMP models

To think algorithmically, we define some function calls that provide **basic elements** specifying a POMP model.

- `rprocess()`: a draw from the one-step transition distribution, with density $f_{X_n|X_{n-1}}(x_n|x_{n-1}; \theta)$.
- `dprocess()`: evaluation of the one-step transition density, $f_{X_n|X_{n-1}}(x_n|x_{n-1}; \theta)$.
- `rmeasure()`: a draw from the measurement distribution with density $f_{Y_n|X_n}(y_n|x_n; \theta)$.
- `dmeasure()`: evaluation of the measurement density, $f_{Y_n|X_n}(y_n|x_n; \theta)$.
- This follows the standard R notation, for example we expect `rnorm` to draw from the normal distribution, and `dnorm` to evaluate the normal density.
- A general POMP model is fully specified by defining these basic elements.

Specifying our own POMP model

- The user will have to say what the basic elements are for their chosen POMP model.
- Algorithms can then use these basic elements to carry out inference for the POMP model.
- We will see that there are algorithms that can carry out likelihood-based inference for this general POMP model specification.

What does it mean for statistical methodology to be simulation-based?

- Simulating random processes can be easier than evaluating their transition probabilities.
- Thus, we may be able to write `rprocess()` but not `dprocess()`.
- **Simulation-based** methods require the user to specify `rprocess()` but not `dprocess()`.
- **Plug-and-play**, **likelihood-free** and **equation-free** are alternative terms for simulation-based.
- Much development of simulation-based statistical methodology has occurred in the past decade.

The **pomp** R package for POMP models

- `pomp` is an R package for data analysis using partially observed Markov process (POMP) models.
- Note the distinction: lower case `pomp` is a software package; upper case POMP is a class of models.
- `pomp` builds methodology for POMP models in terms of arbitrary user-specified `rprocess()`, `dprocess()`, `rmeasure()`, and `dmeasure()` functions.
- Following modern practice, most methodology in `pomp` is simulation-based, so does not require specification of `dprocess()`.
- `pomp` has facilities to help construct `rprocess()`, `rmeasure()`, and `dmeasure()` functions for model classes of scientific interest.
- `pomp` provides a forum for development, modification and sharing of models, methodology and data analysis workflows.
- `pomp` is available from CRAN or github.

Example: the Ricker model

- The Ricker model is a basic model in population biology. We start with a deterministic version and then add process noise and measurement error.
- The **Ricker equation** is a deterministic differential equation modeling the dynamics of a simple population, including population growth and resource depletion.

$$[R1] \quad P_{n+1} = r P_n \exp(-P_n).$$

- Here, P_n is the population density at time $t_n = n$ and r is a fixed value (a parameter), related to the population's intrinsic capacity to increase.
- $P_n = \log(r)$ is an **equilibrium**, meaning that if $P_n = \log(r)$ then $P_{n+1} = P_{n+2} = \dots = P_n$. Another equilibrium is $P_n = 0$.
- P is a **state variable**, r is a **parameter**.
- If we know r and the **initial condition** P_0 , this deterministic Ricker equation predicts the future population density at all times.
- The initial condition, P_0 is a special kind of parameter, an **initial-value parameter**.

Adding stochasticity to the Ricker equation

- We can model process noise by making the growth rate r into a random variable.
- For example, if we assume that the intrinsic growth rate is log-normally distributed, P becomes a stochastic process governed by

$$[R2] \quad P_{n+1} = r P_n \exp(-P_n + \varepsilon_n), \quad \varepsilon_n \sim \text{Normal}(0, \sigma^2),$$

- Here, the new parameter σ is the standard deviation of the noise process ε .

Question 10.1. Does adding Gaussian noise mean we have a Gaussian latent process model? What does it mean to say that the model for $P_{0:N}$ described by equation [R2] is Gaussian?

Adding measurement error to the Ricker model

- Let's suppose that the Ricker model is our model for the dynamics of a real population.
- For most populations, outside of controlled experiments, we cannot know the exact population density at any time, but only estimate it through sampling.
- Let's model measurement error by treating the measurement y_n , conditional on P_n , as a draw from a Poisson distribution with mean ϕP_n . This corresponds to the model

$$[R3] \quad Y_n | P_n \sim \text{Poisson}(\phi P_n).$$

- The parameter ϕ is proportional to the sampling effort.

Writing the Ricker model as a POMP model

- For our standard definition of a POMP model $(X_{0:N}, Y_{0:N})$, we can check that equations [R2] and [R3] together with the parameter P_0 define a POMP model with

$$X_n = P_n \tag{1}$$

$$Y_n = Y_n \tag{2}$$

- Following the usual POMP paradigm, P_n is a true but unknown population density at time t_n .

Working with the Ricker model in pomp

- The R package `pomp` provides facilities for modeling POMP models, a toolbox of statistical inference methods for analyzing data using POMP models, and a development platform for implementing new POMP inference methods.
- The basic data-structure provided by `pomp` is the object of class `pomp`, alternatively known as a `pomp` object.
- A `pomp` object is a container that holds real or simulated data and a POMP model, possibly together with other information such as model parameters, that may be needed to do things with the model and data.
- Let's see what can be done with a `pomp` object.
- First, if we haven't already, we must install `pomp`. The package needs access to code compilers to operate properly so you should check the installation instructions at <https://kingaa.github.io/pomp/install.html>

- For the following, we also load some other packages.

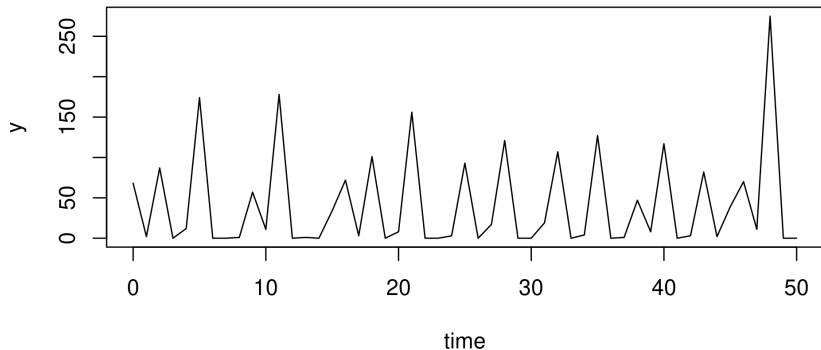
```
set.seed(594709947L)
require(ggplot2)
require(plyr)
require(reshape2)
require(pomp)
```


- A pre-built pomp object encoding the Ricker model comes included with the package. Load it by

```
ricker <- ricker()
```

We can plot the data:

```
plot(ricker)
```



- We can simulate from the model:

```
simulated_ricker <- simulate(ricker)
```

- What kind of object have we created?

```
class(simulated_ricker)
```

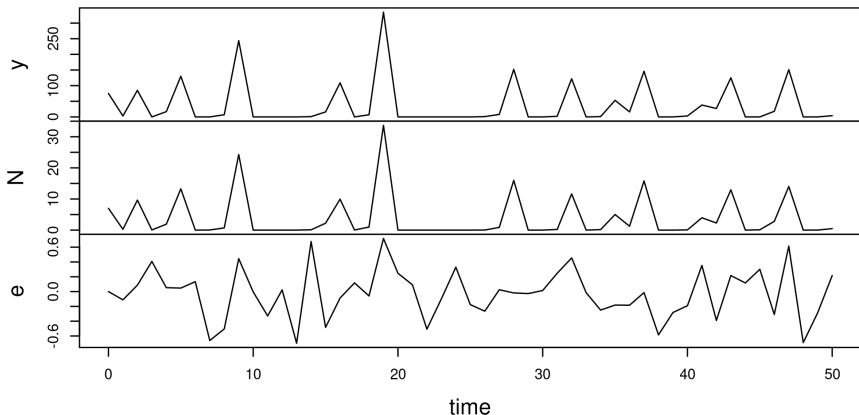
```
## [1] "pomp"  
## attr(,"package")  
## [1] "pomp"
```

What is a generic function?

- How does the concept of a **generic function** fit in with the following related concepts,
- **object-oriented programming**
- assigning a **class** to an object.
- **overloading** of functions or operators.
- **inheritance** between classes, when one class extends another.
- How does object-oriented programming work in R? How is this similar or different from any other environment in which you have seen object-oriented programming?
- For current purposes, we don't need to be experts in object-oriented programming in R. However, we should be familiar with some R object-orientated basics.

- **S3 classes** (<http://adv-r.had.co.nz/00-essentials.html#s3>)
- **S4 classes** (<http://adv-r.had.co.nz/S4.htm>)
- We should be able to recognize when code we are using employs S3 and S4 classes.
- We should know where to turn to for help if we find ourselves needing to know more details about how these work.
- `pomp` uses the S4 class system, so that system is of more immediate relevance. Many older R packages use S3 classes.

```
plot(simulated_ricker)
```



- This pomp representation uses N for our variable P_n

Question 10.2. Why do we see more time series in the simulated pomp object?

Different formats for simulation

- We can turn a pomp object into a data frame:

```
y <- as.data.frame(ricker)
head(y,3)
```

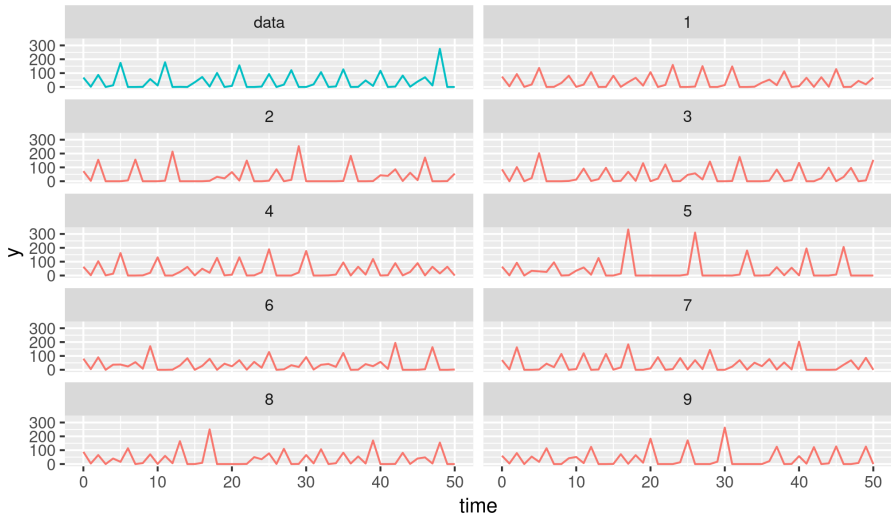
```
##    time  y
## 1     0 68
## 2     1  2
## 3     2 87
```

```
head(simulate(ricker,format="data.frame"))
```

```
##    time .id          N          e    y
## 1     0   1 7.000000e+00 0.0000000  85
## 2     1   1 4.326721e-01 0.4163146   1
## 3     2   1 1.406521e+01 0.1141514 147
## 4     3   1 7.566248e-04 0.4348591   0
## 5     4   1 4.084348e-02 0.1893917   0
## 6     5   1 1.794869e+00 0.0237836  21
```

- We can also run multiple simulations simultaneously:

```
x <- simulate(ricker,nsim=9,format="data.frame",include.data=TRUE)
ggplot(data=x,aes(x=time,y=y,group=.id,color=(.id=="data")))+
  geom_line()+guides(color=FALSE)+
  facet_wrap(~.id,ncol=2)
```



The deterministic skeleton

- The **deterministic skeleton** is a version of the POMP model without process noise.
- It is generated by `trajectory()`.

```
y <- trajectory(ricker)
dim(y)

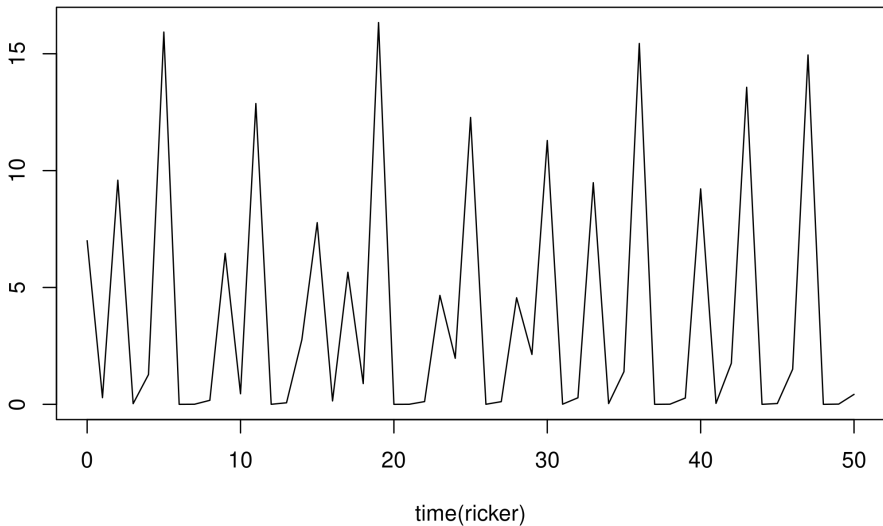
## [1]  2  1 51

dimnames(y)

## $variable
## [1] "N" "e"
##
## $rep
## NULL
##
## $time
## NULL
```



```
plot(time(ricker),y["N",1,],type="l")
```



Working with model parameters I

- `ricker` has parameters associated with it:

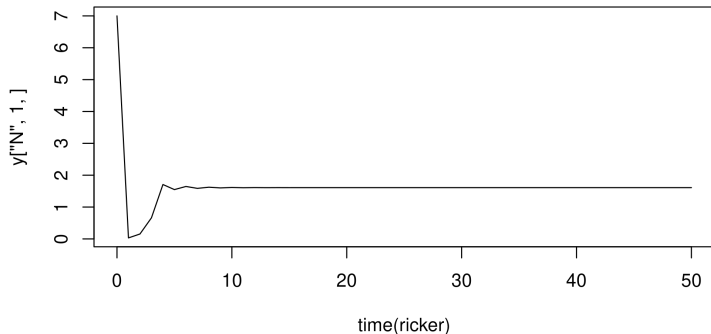
```
coef(ricker)

##           r      sigma      phi           c      N_0      e_0
## 44.70118  0.30000 10.00000  1.00000  7.00000  0.00000
```

- These are the parameters at which the simulations and deterministic trajectory computations above were done.
- We can run these at different parameters:

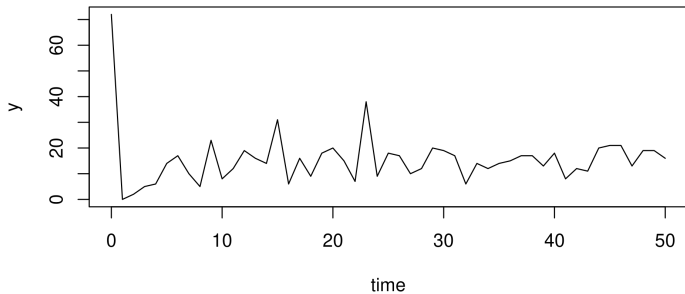
```
theta <- coef(ricker)
theta[c("r", "N.0")] <- c(5, 3)
y <- trajectory(ricker, params=theta)
plot(time(ricker), y["N", 1, ], type="l")
```

Working with model parameters II



```
x <- simulate(ricker, params=theta)
plot(x, var="y")
```

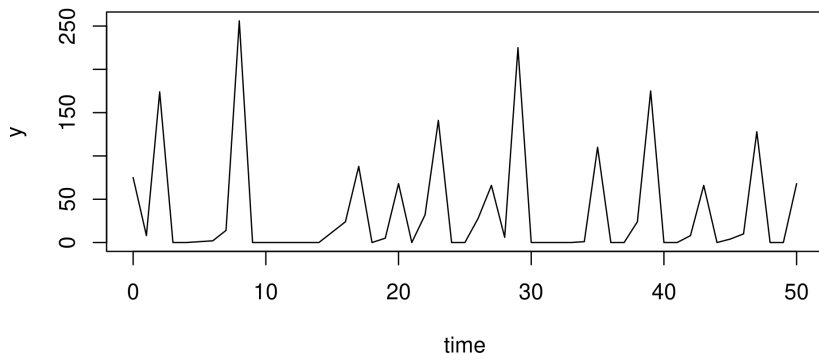
Working with model parameters III



- We can also change the parameters stored inside of ricker:

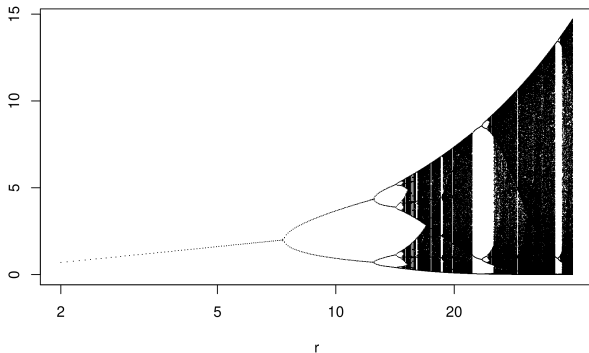
```
coef(ricker,c("r","N.0","sigma")) <- c(39,0.5,1)
coef(ricker)
plot(simulate(ricker),var="y")
```

##	r	sigma	phi	c	N_0	e_0	N.0
##	39.0	1.0	10.0	1.0	7.0	0.0	0.5



It is possible to work with more than one set of parameters at a time:

```
p <- parmat(coef(ricker),500)
p["r",] <- seq(from=2,to=40,length=500)
y <- trajectory(ricker,params=p,times=200:1000)
matplot(p["r",],y["N",,],pch=".",col='black',xlab='r',ylab='N',log=
```



This is a bifurcation diagram
(https://en.wikipedia.org/wiki/Bifurcation_diagram) for the
Ricker equation.

Question 10.3. How do you interpret this bifurcation diagram?

- What does it mean when the single line for small values of r splits into a double line, around $r = 0.8$?
- What does it mean when solid vertical lines appear, around $r = 18$?
- A bifurcation diagram like this can only be computed for a deterministic map. However, the bifurcation diagram for the deterministic skeleton can be useful to help understand a stochastic process. We'll see an example later in this chapter.

- Look at the R code for the bifurcation diagram. Notice that the first 200 iterations of the Ricker map are discarded, by setting `times=200:1000`. This is a technique called **burn-in**.
- This is used when aiming to simulate the steady state of a dynamic system, ignoring **transient behavior** from initial conditions.

Inference algorithms in pomp

- pomp provides a wide range of inference algorithms. We'll learn about these in detail soon, but for now, let's just look at some of their general features.
- The `pfilter` function runs a simple **particle filter**, which is a Monte Carlo algorithm that can be used to evaluate the likelihood at a particular set of parameters. One uses the `Np` argument to specify the number of particles to use:

```
pf <- pfilter(ricker,Np=1000)
class(pf)

## [1] "pfilterd_pomp"
## attr(,"package")
## [1] "pomp"

plot(pf)
```

Building a custom pomp object

A real pomp data analysis begins with constructing one or more pomp objects to hold the data and the model or models under consideration. We illustrate this process a dataset of the abundance of the great tit (*Parus major* in Wytham Wood, near Oxford (?).

First, we load and plot the data:

```
dat <- read.csv("parus.csv")  
head(dat)
```

```
##   year pop  
## 1 1960 148  
## 2 1961 258  
## 3 1962 185  
## 4 1963 170  
## 5 1964 267  
## 6 1965 239
```

```
plot(pop~year, data=dat, type='o')
```

Let's suppose that we want to fit the stochastic Ricker model discussed above to these data.

- The call to construct a `pomp` object is, naturally enough, `pomp`.
- Documentation on this function can be had by doing `?pomp`.
- Do `class?pomp` to get documentation on the `pomp` class.
- Learn about the various things you can do once you have a `pomp` object by doing `methods?pomp` and following the links therein.
- Read an overview of the package as a whole with links to its main features by doing `package?pomp`.
- A complete index of the functions in `pomp` is returned by the command `library(help=pomp)`.
- The home page for the `pomp` project is (<http://kingaa.github.io/pomp>); there you have access to the complete source code, manuals, mailing lists, etc.

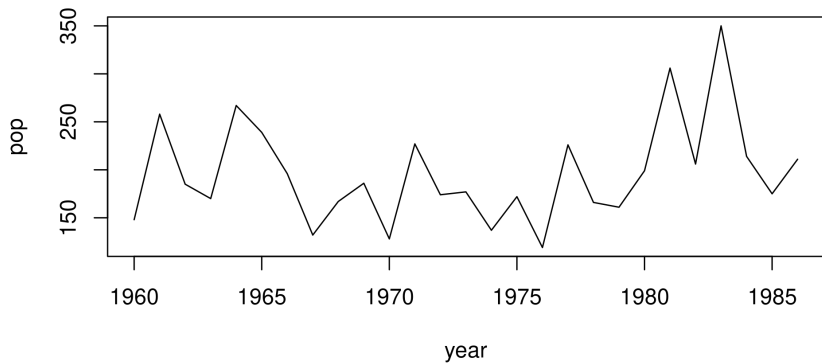
The simplest `pomp` object has only the data:

```
parus <- pomp(dat, times="year", t0=1959)
```

The `times` argument specifies that the column labelled "year" gives the measurement times; `t0` is the "zero-time", the time at which the state process will be initialized. We've set it to one year prior to the beginning of the data.

- We can now plot the data:

```
plot(parus)
```



Adding in the deterministic skeleton

We can add the Ricker model deterministic skeleton to the `parus` `pomp` object. Since the Ricker model is a discrete-time model, its skeleton is a map that takes P_n to P_{n+1} according to the Ricker model equation

$$P_{n+1} = r P_n \exp(-P_n).$$

We provide this to `pomp` in the form of a `Csnippet`, a little snippet of C code that performs the computation.

```
skel <- Csnippet("DN = r*N*exp(-N);")
```

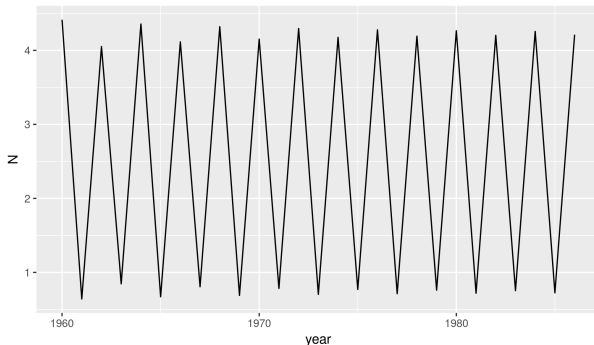
We then add this to the `pomp` object:

```
parus <- pomp(parus, skeleton=map(skel), statenames="N", paramnames="r")
```

Note that we have to inform `pomp` as to which of the variables we've referred to in `skel` is a state variable (`statenames`) and which is a parameter (`paramnames`).

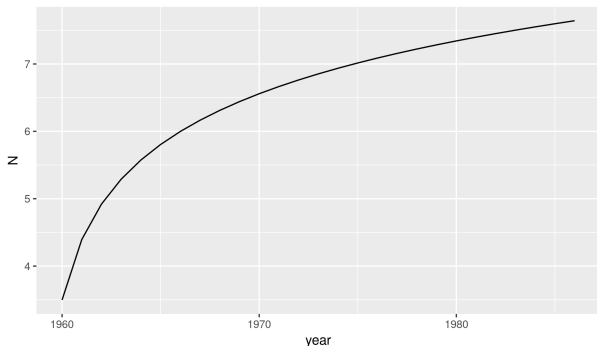
With just the skeleton defined, we are in a position to compute the trajectories of the deterministic skeleton at any point in parameter space. For example,

```
traj <- trajectory(parus, params=c(N.0=1, r=12), format="data.frame")  
ggplot(data=traj, aes(x=year, y=N)) + geom_line()
```



- Note that the dynamics become very different if the skeleton is considered as the derivative of a differential equation rather than as a discrete time map. It is harder to get chaotic dynamics in a continuous time system.

```
parus2 <- pomp(parus,skeleton=vectorfield(skel),statenames="N",param  
traj2 <- trajectory(parus2,params=c(N.0=1,r=12),format="data.frame",  
ggplot(data=traj2,aes(x=year,y=N))+geom_line()
```



A note on terminology

- If we know the state, $x(t_0)$, of the system at time t_0 , it makes sense to speak about the entire trajectory of the system for all $t > t_0$.
- This is true whether we are thinking of the system as deterministic or stochastic.
- Of course, in the former case, the trajectory is uniquely determined by $x(t_0)$, while in the stochastic case, only the probability distribution of $x(t)$, $t > t_0$ is determined.
- In pomp, to avoid confusion, we use the term “trajectory” exclusively to refer to **trajectories of a deterministic process**. Thus, the `trajectory` command iterates or integrates the deterministic skeleton forward in time, returning the unique trajectory determined by the specified parameters. When we want to speak about sample paths of a stochastic process, we use the term **simulation**.
- Accordingly, the `simulate` command always returns individual sample paths from the POMP. In particular, we avoid “simulating a set of differential equations”, preferring instead to speak of “integrating” the equations, or “computing trajectories”.

Adding in the process model simulator

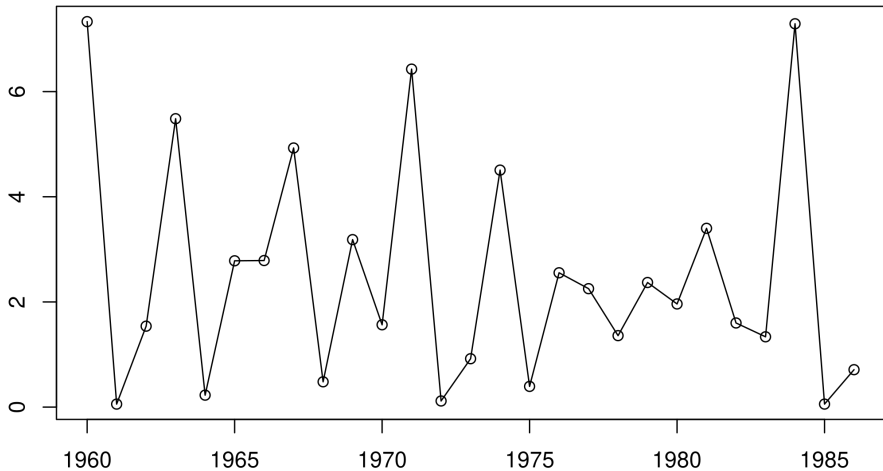
- We can add the stochastic Ricker model to `parus` by writing a Csnippet that simulates one realization of the stochastic process, from an arbitrary time t to $t + 1$, given arbitrary states and parameters:

```
stochStep <- Csnippet("  
  e = rnorm(0,sigma);  
  N = r*N*exp(-N+e);  
")  
pomp(parus,rprocess=discrete.time.sim(step.fun=stochStep,delta.t=1)  
      paramnames=c("r","sigma"),statenames=c("N","e")) -> parus
```

- Note that in the above, we use the `exp` and `rnorm` functions from the C language API for R (<https://cran.r-project.org/doc/manuals/R-exts.html>)
- In general any C function provided by R is available to you. `pomp` also provides a number of C functions that are documented in the header file, `pomp.h`, that is installed with the package.
- See the Csnippet documentation (`?Csnippet`) to read more about how to write them.

- At this point, we have what we need to simulate the stochastic Ricker model.

```
sim <- simulate(parus, params=c(N.0=1, e.0=0, r=12, sigma=0.5),  
               format="data.frame")  
plot(N~year, data=sim, type='o')
```



Adding in the measurement model and parameters

- We complete the specification of the POMP by specifying the measurement model. To obtain the Poisson measurement model described above, we write two Csnippets. The first simulates:

```
rmeas <- Csnippet("pop = rpois(phi*N);")
```

- The second computes the likelihood of observing pop birds given a true density of N:

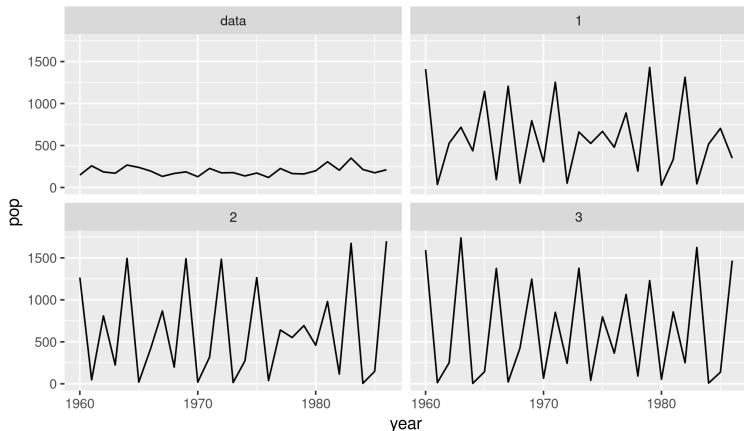
```
dmeas <- Csnippet("lik = dpois(pop,phi*N,give_log);")
```

- Note the give_log argument. When this code is evaluated, give_log will be set to 1 if the log likelihood is desired, and 0 else.
- We add these specifications of rmeasure and dmeasure into the pomp object:

```
pomp(parus,rmeasure=rmeas,dmeasure=dmeas,statenames=c("N"),  
      paramnames=c("phi")) -> parus
```

- To simulate, we must add some parameters to the pomp object:

```
coef(parus) <- c(N.0=1,e.0=0,r=20,sigma=0.1,phi=200)
sims <- simulate(parus,nsim=3,format="data.frame",
  include.data=TRUE)
ggplot(data=sims,mapping=aes(x=year,y=pop))+geom_line()+
  facet_wrap(~.id)
```



Exercise: Ricker model parameters

- Fiddle with the parameters to try and make the simulations look more like the data.
- This will help you build some intuition for what the various parameters do.

Exercise: reformulating the Ricker model

- Reparameterize the Ricker model so that the scaling of P is explicit:

$$P_{n+1} = r P_n \exp \left(-\frac{P_n}{K} \right).$$

- Modify the `pomp` object we created above to reflect this reparameterization.
- Modify the measurement model so that

$$\text{pop}_n \sim \text{Negbin}(\phi P_n, k),$$

i.e., pop_n is negative-binomially distributed with mean ϕP_t and clumping parameter k .

- See `?NegBinomial` for documentation on the negative binomial distribution and the R Extensions Manual section on distribution functions (<https://cran.r-project.org/doc/manuals/R-exts.html>) for information on how to access these in C.

Exercise: The Beverton-Holt model

- Construct a pomp object for the *Parus major* data and the **stochastic Beverton-Holt** model,

$$P_{n+1} = \frac{a P_n}{1 + b P_n} \varepsilon_n,$$

where a and b are parameters and

$$\varepsilon_t \sim \text{Lognormal}(-\tfrac{1}{2}\sigma^2, \sigma^2).$$

- Assume the same measurement model as we used for the Ricker model.

Acknowledgments and License

- These notes build on previous versions at ionides.github.io/531w16 and ionides.github.io/531w18.
- Those notes draw on material developed for a short course on Simulation-based Inference for Epidemiological Dynamics (<http://kingaa.github.io/sbied/>) by Aaron King and Edward Ionides, taught at the University of Washington Summer Institute in Statistics and Modeling in Infectious Diseases, from 2015 through 2018.
- Licensed under the Creative Commons attribution-noncommercial license, <http://creativecommons.org/licenses/by-nc/3.0/>. Please share and remix noncommercially, mentioning its origin.

