ADVANCED TOPICS IN POMP

AARON A. KING

Contents

This document serves to introduce the low-level interface to pomp objects and to give some examples of the use of native (C or FORTRAN) codes in pomp.

1. The low-level interface

There is a low-level interface to pomp objects, primarily designed for package developers. Ordinary users should have little reason to use this interface. In this section, each of the methods that make up this interface will be introduced.

The init.state method is called to initialize the state (unobserved) process. It takes a vector or matrix of parameters and returns a matrix of initial states.

```
data(ou2)
true.p <- coef(ou2)
x0 <- init.state(ou2)
x0
```

```
   [,1]
x1   -3
x2    4
```

```
new.p <- cbind(true.p,true.p,true.p)
new.p["x1.0",] <- 1:3
init.state(ou2,params=new.p)
```

```
   [,1] [,2] [,3]
x1    1    2    3
x2    4    4    4
```

The rprocess method gives access to the process model simulator. It takes initial conditions (which need not correspond to the zero-time t0 specified when the pomp object was constructed), a set of times, and a set of parameters. The initial states and parameters must be matrices, and they are checked for commensurability. The method returns a rank-3 array containing simulated state trajectories, sampled at the times specified.

```
x <- rprocess(ou2,xstart=x0,times=time(ou2,t0=T),params=as.matrix(true.p))
dim(x)
```

```
[1]   2   1 101
```

```
x[,,1:5]
```

```
   [,1]      [,2]       [,3]      [,4]     [,5]
x1   -3 -3.283357 -0.6275075  6.035261 9.479388
x2    4  7.630037  7.6131727 10.854834 2.597988
```

Note that the dimensions of `x` are `nvars x nreps x ntimes`, where `nvars` is the number of state variables, `nreps` is the number of simulated trajectories (which is the number of columns in the `params` and `xstart` matrices), and `ntimes` is the length of the `times` argument. Note also that `x[,,1]` is identical to `xstart`.

The `rmeasure` method gives access to the measurement model simulator:

```
x <- x[,,-1,drop=F]
y <- rmeasure(ou2,x=x,times=time(ou2),params=as.matrix(true.p))
dim(y)
```

```
[1]   2   1 100
```

```
y[,,1:5]
```

```
        [,1]       [,2]      [,3]     [,4]      [,5]
y1 -1.548630 -0.5511519  6.020389 9.617936  9.241980
y2  8.308526  9.2890896 12.274503 1.584327 -1.991055
```

The `dmeasure` and `dprocess` methods give access to the measurement and process model densities, respectively.

```
fp <- dprocess(ou2,x=x,times=time(ou2),params=as.matrix(true.p))
dim(fp)
```

```
[1]  1 99
```

```
fp[,36:40]
```

```
[1] 0.006356768 0.008230949 0.011165882 0.018466369
[5] 0.015810859
```

```
fm <- dmeasure(ou2,y=y[,1,],x=x,times=time(ou2),params=as.matrix(true.p))
dim(fm)
```

```
[1]   1 100
```

```
fm[,36:40]
```

```
[1] 0.118826706 0.065905743 0.016613495 0.118275252
[5] 0.008519114
```

All of these are to be preferred to direct access to the slots of the `pomp` object, because they do sanity checks on the inputs and outputs.

2. Acceleration using native codes.

Since many of the methods we will use require us to simulate the process and/or measurement models many times, it is a good idea to use native (compiled) codes for the computational heavy lifting. This can result in many-fold speedup. The `pomp` package includes some examples that use C codes. Here, we'll have a look at how the discrete-time 2-D Ornstein-Uhlenbeck process with normal measurement error is implemented.

Recall that the unobserved Ornstein-Uhlenbeck (OU) process $X_t \in \mathbb{R}^2$ satisfies

$$X_t = A\,X_{t-1} + \xi_t.$$

The observation process is

$$Y_t = B\,X_t + \varepsilon_t.$$

In these equations, A and and B are 2×2 constant matrices; ξ_t and ε_t are mutually-independent families of i.i.d. bivariate normal random variables. We let $\sigma\sigma^T$ be the variance-covariance matrix of ξ_t, where σ is lower-triangular; likewise, we let $\tau\tau^T$ be that of ε_t.

You can load a `pomp` object for this model with the command

```
data(ou2)
```

Here we'll examine how this object is put together.

The process model simulator and density functions are as follows:

```
ou2.rprocess <- function (xstart, times, params, paramnames, ...) {
  nvar <- nrow(xstart)
  npar <- nrow(params)
  nrep <- ncol(xstart)
  ntimes <- length(times)
  ## get indices of the various parameters in the 'params' matrix
  ## C uses zero-based indexing!
  parindex <- match(paramnames,rownames(params))-1
  array(
        .C("ou2_adv",
           X = double(nvar*nrep*ntimes),
           xstart = as.double(xstart),
           par = as.double(params),
           times = as.double(times),
           n = as.integer(c(nvar,npar,nrep,ntimes)),
           parindex = as.integer(parindex),
           DUP = FALSE,
           NAOK = TRUE,
           PACKAGE = "pomp"
           )$X,
        dim=c(nvar,nrep,ntimes),
        dimnames=list(rownames(xstart),NULL,NULL)
        )
}
ou2.dprocess <- function (x, times, params, log, paramnames, ...) {
  nvar <- nrow(x)
  npar <- nrow(params)
  nrep <- ncol(x)
  ntimes <- length(times)
```

```
    parindex <- match(paramnames,rownames(params))-1
    array(
          .C("ou2_pdf",
             d = double(nrep*(ntimes-1)),
             X = as.double(x),
             par = as.double(params),
             times = as.double(times),
             n = as.integer(c(nvar,npar,nrep,ntimes)),
             parindex = as.integer(parindex),
             give_log=as.integer(log),
             DUP = FALSE,
             NAOK = TRUE,
             PACKAGE = "pomp"
             )$d,
          dim=c(nrep,ntimes-1)
          )
  }
```

The call that constructs the `pomp` object is:

```
ou2 <- pomp(
            times=seq(1,100),
            data=rbind(
              y1=rep(0,100),
              y2=rep(0,100)
              ),
            t0=0,
            rprocess = ou2.rprocess,
            dprocess = ou2.dprocess,
            dmeasure = "normal_dmeasure",
            rmeasure = "normal_rmeasure",
            paramnames=c(
              "alpha.1","alpha.2","alpha.3","alpha.4",
              "sigma.1","sigma.2","sigma.3",
              "tau"
              ),
            statenames = c("x1","x2"),
            PACKAGE="pomp"
            )
```

Notice that the process model is implemented using using `.C`, while the measurement model is specified by giving the names of native C routines. Read the source (file 'ou2.c') to see the definitions of these functions.

We'll specify some parameters:

```
p <- c(
       alpha.1=0.9,alpha.2=0,alpha.3=0,alpha.4=0.99,
       sigma.1=1,sigma.2=0,sigma.3=2,
       tau=1,x1.0=50,x2.0=-50
       )

tic <- Sys.time()
x <- simulate(ou2,params=p,nsim=500,seed=800733088)
```

```
toc <- Sys.time()
print(toc-tic)
```

```
Time difference of 0.928875 secs
```

In this example, we've written our simulators and density functions "from scratch". `pomp` provides "plug-in" facilities to make it easier to define certain kinds of models. These plug-ins can be used with native codes as well, as we'll see in the next example.

3. A more complex example: a seasonal SIR model

The SIR model is a mainstay of theoretical epidemiology. It has the deterministic skeleton

$$\begin{aligned}\frac{dS}{dt} &= \mu\,(N-S) + \beta(t)\,\frac{I}{N}\,S\\ \frac{dI}{dt} &= \beta(t)\,\frac{I}{N}\,S - \gamma\,I - \mu\,I\\ \frac{dR}{dt} &= \gamma\,I - \mu\,R\end{aligned}$$

Here $N = S + I + R$ is the (constant) population size and β is a time-dependent contact rate. We'll assume that the contact rate is periodic and implement it as a covariate. We'll implement a stochastic version of this model using an Euler-multinomial approximation to the continuous-time Markov process. As an additonal wrinkle, we'll assume that the rate of the infection process $\beta\,I/N$ is perturbed by white noise.

```
pomp(
     times=seq(1/52,4,by=1/52),
     data=rbind(measles=numeric(52*4)),
     t0=0,
     statenames=c("S","I","R","cases","W"),
     paramnames=c("gamma","mu","iota","beta1","beta.sd","pop","rho","nbasis","degree","period"),
     zeronames=c("cases"),
     comp.names=c("S","I","R"),
     rprocess=euler.sim(
       step.fun="sir_euler_simulator",
       delta.t=1/52/20
       ),
     skeleton.vectorfield="sir_ODE",
     rmeasure="binom_rmeasure",
     dmeasure="binom_dmeasure",
     PACKAGE="pomp",
     initializer=function(params, t0, comp.names, ...){
       p <- exp(params)
       snames <- c("S","I","R","cases","W")
       fracs <- p[paste(comp.names,"0",sep=".")]
       x0 <- numeric(length(snames))
       names(x0) <- snames
       x0[comp.names] <- round(p['pop']*fracs/sum(fracs))
       x0
     }
     ) -> euler.sir
```

```
coef(euler.sir) <- c(
                    gamma=log(26),mu=log(0.02),iota=log(0.01),
                    beta1=log(1200),beta2=log(1800),beta3=log(600),
                    beta.sd=log(1e-3),
                    pop=log(2.1e6),
                    rho=log(0.6),
                    S.0=log(26/1200),I.0=log(0.001),R.0=log(1-0.001-26/1200),
                    nbasis=3,degree=3,period=1
                    )

euler.sir <- simulate(euler.sir,nsim=1,seed=329348545L)
```

This example can be loaded via

```
data(euler.sir)
```

A. A. King, Departments of Ecology & Evolutionary Biology and Mathematics, University of Michigan, Ann Arbor, Michigan 48109-1048 USA

E-mail address: kingaa at umich dot edu

URL: http://www.umich.edu/~kingaa