SIMPLE EXAMPLES OF THE USE OF MIF WITHIN THE POMP PACKAGE

AARON A. KING

In this vignette, we show the use of the POMPpackage, and the MIFalgorithm within it by means of simple examples. The first is a trivial example: much simpler methods could be used for this model. Nevertheless, it will serve to illustrate the functionality provided by POMPand to give some hints as to its use. The second example is nontrivial and involves a model for which alternate fitting procedures are available. One can therefore check the consistency of the results MIFgives.

1. Getting started with pomp

2. A trivial example: IID noise

A trivial example of a state-space model is afforded by the case of an i.i.d. random variable. The example is useful insofar as it allows us to demonstrate the use of the `pomp` package.

2.1. **The model.** We will denote the unobserved (state) process by X_t and the observed process by Y_t. To be specific, assume that the unobserved process is just a constant process: $X_t = \mu$. The observation process is just measurement with normal error:

$$Y_t \sim N(X_t, \sigma^2).$$

So $Y_t \sim N(\mu, \sigma^2)$. Although it isn't natural in this problem, many other problems will have a additional parameters, namely, the initial values of the state variables. Let's assume that we have an initial condition in this problem as well: $X_0 = x_0$. We'll have to estimate μ, σ, and x_0.

2.2. **Inference using `pomp`.** The basic element of the `pomp` package is the `pomp`, or "partially-observed Markov process" object. Let's set up such an object for the IID model above using simulated data. First, we'll need to load the package:

```
> require(pomp)
```

We'll use the following parameters for generating the simulated data.

Date: May 22, 2007.

```
> true.params <- c(m = 5, s = 3, x0 = 10)
```

Now we come to the setup of the pomp object itself. First we'll generate some simulated data:

```
> x <- with(as.list(true.params), rbind(time = seq(1, 100), obs = rnorm(100,
+     mean = m, sd = s)))
```

Next, we'll need four basic functions, which define the model we're interested in. These are

- rprocess: This function simulates one step of the unobserved process, i.e., it is a random draw from the conditional distribution $f(X_{t+1} \,|\, X_t)$.
- rmeasure: This simulates the measurement process, given the state. That is, it is a random draw from the conditional distribution $f(Y_t \,|\, X_t)$.
- dmeasure: This furnishes the probability density or mass of the measurement process, i.e., it gives $f(Y_t = y \,|\, X_t)$.
- particles:

```
> rprocess <- function(X, t1, t2, ...) {
+     X["x", ] <- X["m", ]
+     X
+ }
> rmeasure <- function(X, time, ...) {
+     rnorm(ncol(X), mean = X["x", ], sd = X["s", ])
+ }
> dmeasure <- function(X, Y, ...) {
+     dnorm(Y["obs"], mean = X["x", ], sd = X["s", ])
+ }
> particles <- function(Np, center, sd, ...) {
+     X <- matrix(0, nrow = length(center) + 1, ncol = Np)
+     rownames(X) <- c("x", names(center))
+     X[names(center), ] <- rnorm(Np * length(center), mean = center,
+         sd = sd)
+     X["x", ] <- X["x0", ]
+     X
+ }
```

```
> iid <- pomp(data = x, t0 = 0, rprocess = rprocess, rmeasure = rmeasure,
+     dmeasure = dmeasure, particles = particles)
```

We can simulate from the pomp object:

```
> y <- simulate(iid, coef = c(m = 9, s = 1, x0 = 100))
```

We can plot the results (Fig. 1).

```
> plot(y)
```

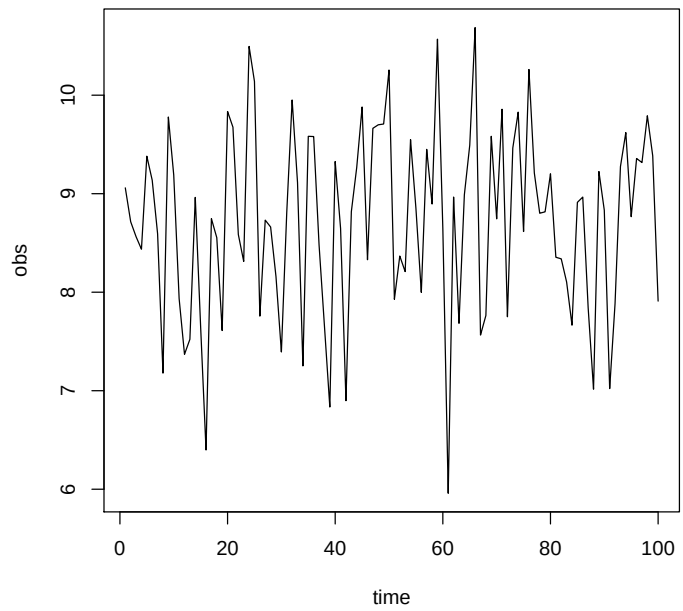


FIGURE 1. Simulating from an `pomp` object. The command used to generate the plot is shown above.

Now we can use the MIF algorithm to find maximum likelihood estimates of the parameters.

```
> z <- mif(iid, start = c(m = 3, s = 5, x0 = 10), rw.sd = c(m = 0.1,
+     s = 0.1, x0 = 1), Nmif = 1, ivps = "x0", pars = c("m", "s"),
+     stvs = "x", alg.pars = list(cooling.factor = 0.99, CC = 2,
+         Np = 50, T0 = 3))
```

```
> z <- continue(z, Nmif = 50, alg.pars = list(cooling.factor = 0.99,
+     CC = 5, Np = 50, T0 = 3))
```

To perform additional MIF iterations on a `mif` object, one uses the `continue` function.

```
> z <- continue(z, Nmif = 500)
```

```
> truth <- c(true.params, loglik = sum(dnorm(iid@data[2, ], mean = 5,
+     sd = 3, log = T)))
> mle.params <- c(m = mean(iid@data[2, ]), s = sd(iid@data[2, ]),
+     x0 = NA)
```

```
> predvarplot(z)
```

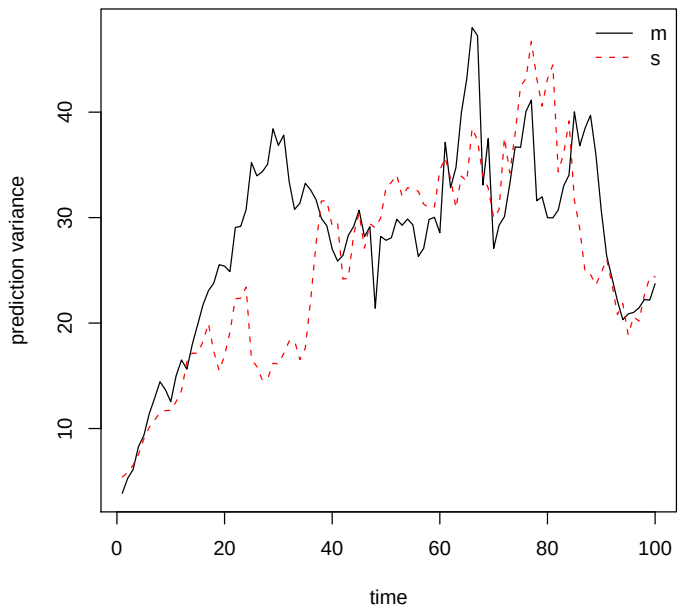


FIGURE 2. Using `predvarplot` to choose a good value of `CC`. The command used to generate the plot is shown above. A value of 5 looks good.

```
> mle <- c(mle.params, loglik = with(as.list(mle.params), sum(dnorm(iid@data[2
+     ], mean = m, sd = s, log = T))))
> best <- pfilter(z)
> best.estimate <- c(coef(z), best$loglik)
> require(xtable)
> xtable(cbind(truth = truth, mle = mle, "MIF estimate" = best.estimate),
+     caption = "Comparison of true, MLE, and best MIF parameters",
+     digits = c(0, 3, 3, 3))
```

```
> par(mfrow = c(2, 4))
> plot(z)
> par(mfrow = c(1, 1))
```

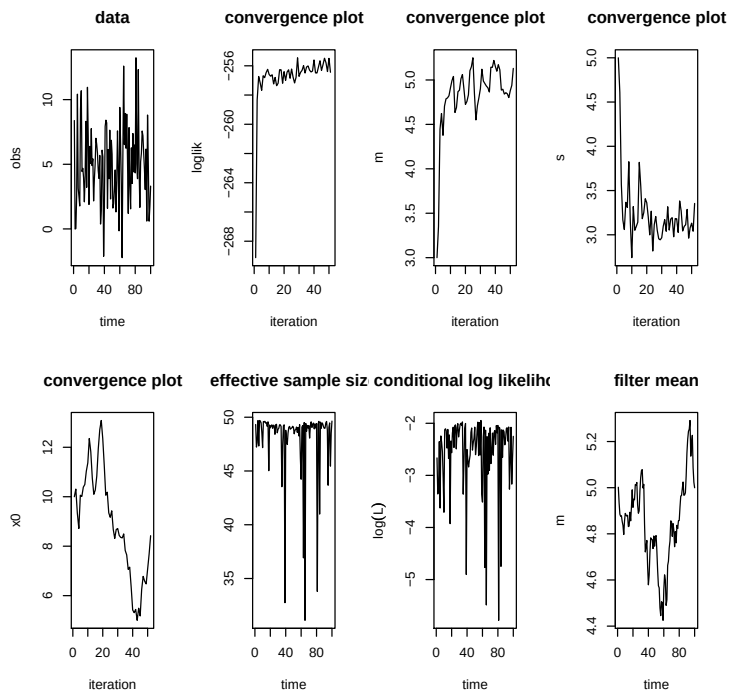


FIGURE 3. Plotting a `mif` object.

	truth	mle	MIF estimate
m	5.000	4.978	5.106
s	3.000	3.085	3.033
x0	10.000		5.498
loglik	−254.119	−254.065	−254.168

TABLE 1. Comparison of true, MLE, and best MIF parameters

3. A nontrivial example: a 2-D Ornstein-Uhlenbeck process

We will want to maximize the likelihood on a transformed parameter space. It is useful to define functions that will perform the forward and inverse parameter transformations.

```
> par(mfrow = c(2, 4))
> plot(z)
> par(mfrow = c(1, 1))
```

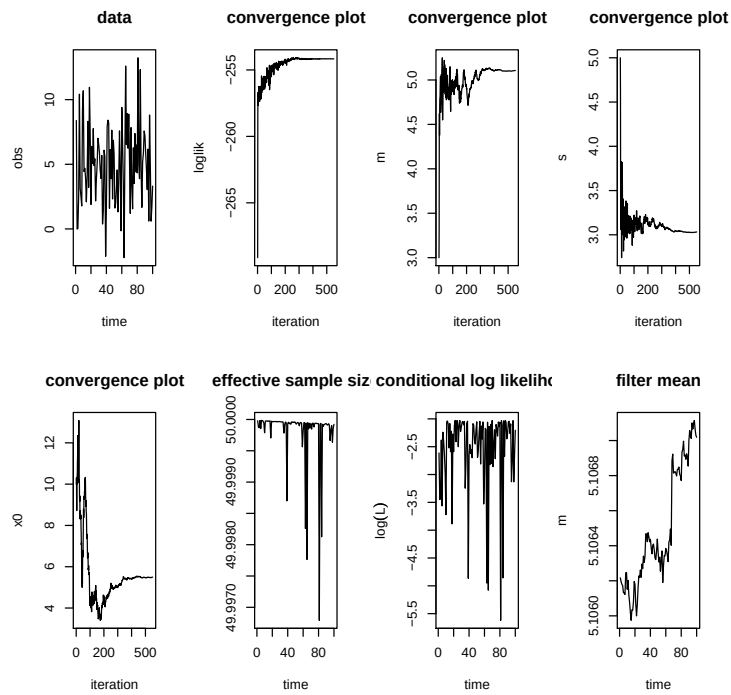


FIGURE 4. Results of fitting the model to the data using MIF.

```
> par.trans <- function(params) {
+     x <- params
+     r <- length(dim(x))
+     if (r > 1) {
+         x <- apply(x, 2:r, par.trans)
+     }
+     else {
+         x["alpha.1"] <- log(-params["alpha.1"])
+         x["alpha.2"] <- log(params["alpha.2"])
+         x["alpha.3"] <- log(-params["alpha.3"])
+         x["alpha.4"] <- log(-params["alpha.4"])
+         x["sigma.1"] <- log(params["sigma.1"])
+         x["sigma.3"] <- log(params["sigma.3"])
+         x["tau"] <- log(params["tau"])
+     }
```

```
+     x
+ }
> par.untrans <- function(params) {
+     x <- params
+     r <- length(dim(x))
+     if (r > 1) {
+         x <- apply(x, 2:r, par.trans)
+     }
+     else {
+         x["alpha.1"] <- -exp(params["alpha.1"])
+         x["alpha.2"] <- exp(params["alpha.2"])
+         x["alpha.3"] <- -exp(params["alpha.3"])
+         x["alpha.4"] <- -exp(params["alpha.4"])
+         x["sigma.1"] <- exp(params["sigma.1"])
+         x["sigma.3"] <- exp(params["sigma.3"])
+         x["tau"] <- exp(params["tau"])
+     }
+     x
+ }
```

```
> rw.sd <- c(alpha.1 = 0.1, alpha.2 = 0.1, alpha.3 = 0.1, alpha.4 = 0.1,
+     sigma.1 = 0.1, sigma.2 = 0.1, sigma.3 = 0.1, tau = 0.1, x1.0 = 0.2,
+     x2.0 = 0.2)
> start <- c(alpha.1 = -0.1, alpha.2 = 0.1, alpha.3 = -0.2, alpha.4 = -0.5,
+     sigma.1 = 0.1, sigma.2 = 0, sigma.3 = 2, tau = 2, x1.0 = 5,
+     x2.0 = 3)
> truth <- c(alpha.1 = -0.5, alpha.2 = 0.1, alpha.3 = -0.1, alpha.4 = -0.5,
+     sigma.1 = 1, sigma.2 = 0, sigma.3 = 1, tau = 0.5, x1.0 = 5,
+     x2.0 = 3)
> estnames <- c("alpha.1", "alpha.3", "sigma.1", "sigma.3", "tau")
> fixnames <- c("alpha.2", "alpha.4", "sigma.2")
> ivpnames <- c("x1.0", "x2.0")
```

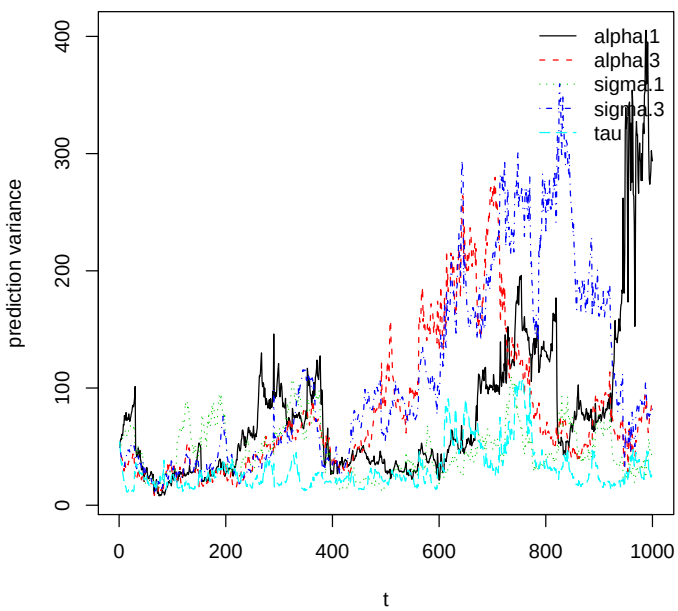
```
> rprocess <- function(X, t1, t2, ...) {
+     nvar <- nrow(X)
+     np <- ncol(X)
+     stateindex <- match(c("x1"), rownames(X)) - 1
+     parindex <- match(c("alpha.1", "alpha.2", "alpha.3", "alpha.4",
+         "sigma.1", "sigma.2", "sigma.3"), rownames(X)) - 1
+     result <- .C("ou2_adv", X = as.double(X), nvar = as.integer(nvar),
+         np = as.integer(np), stateindex = as.integer(stateindex),
+         parindex = as.integer(parindex), DUP = FALSE, NAOK = TRUE,
+         PACKAGE = "pomp")$X
```

```
+     array(result, dim = dim(X), dimnames = dimnames(X))
+ }
> dmeasure <- function(X, Y, ...) {
+     n <- dim(X)
+     measindex <- match(c("x1", "tau"), rownames(X)) - 1
+     .C("normal_dmeasure", n = as.integer(n), index = as.integer(measindex),
+         X = as.double(X), y = as.double(Y[2]), f = double(n[2]),
+         DUP = FALSE, NAOK = TRUE, PACKAGE = "pomp")$f
+ }
> rmeasure <- function(X, time, ...) {
+     n <- dim(X)
+     nsim <- n[2]
+     measindex <- match(c("x1", "tau"), rownames(X)) - 1
+     matrix(.C("normal_rmeasure", n = as.integer(n), index = as.integer(measi
+         X = as.double(X), cases = double(nsim), DUP = FALSE,
+         NAOK = TRUE, PACKAGE = "pomp")$cases, 1, nsim)
+ }
> particles <- function(Np, center, sd, statenames, parnames, fixnames,
+     ivpnames, ...) {
+     X <- matrix(data = 0, nrow = length(statenames) + length(parnames) +
+         length(fixnames) + length(ivpnames), ncol = Np, dimnames = list(c(st
+         parnames, fixnames, ivpnames), NULL))
+     X[parnames, ] <- rnorm(n = Np * length(parnames), mean = center[parnames
+         sd = sd[parnames])
+     X[fixnames, ] <- center[fixnames]
+     X[ivpnames, ] <- rnorm(n = Np * length(ivpnames), mean = center[ivpnames
+         sd = sd[ivpnames])
+     X[statenames, ] <- X[ivpnames, ]
+     X
+ }
> ex <- pomp(data = rbind(t = 1:1000, z = rep(0, 1000)), t0 = 0,
+     rprocess = rprocess, dmeasure = dmeasure, rmeasure = rmeasure,
+     particles = particles, parnames = estnames, fixnames = fixnames,
+     ivpnames = c("x1.0", "x2.0"), statenames = c("x1", "x2"))
> ex <- simulate(ex, coef = par.trans(truth))
> print(arima.fit <- arima(ex@data[2, ], order = c(2, 0, 2)))
Call:
arima(x = ex@data[2, ], order = c(2, 0, 2))

Coefficients:
          ar1     ar2      ma1     ma2  intercept
```

```
      -0.0871  0.1424  -0.3373  0.0650    -0.0221
s.e.   0.3767  0.2091   0.3765  0.0836     0.0273

sigma^2 estimated as 1.258:  log likelihood = -1533.84,  aic = 3079.68
> ff <- pfilter(ex, Np = 1000, coef = par.trans(truth))
> print(c(loglik = ff$loglik, nfail = ff$nfail))
   loglik     nfail
-1535.221     0.000
> x <- mif(ex, start = par.trans(start), rw.sd = rw.sd, Nmif = 1,
+     pars = estnames, ivps = ivpnames, stvs = c("x1", "x2"), alg.pars = list(
+         CC = 7, T0 = 4, cooling.factor = 0.95), max.fail = 1000,
+     weighted = F)
> predvarplot(x)
```



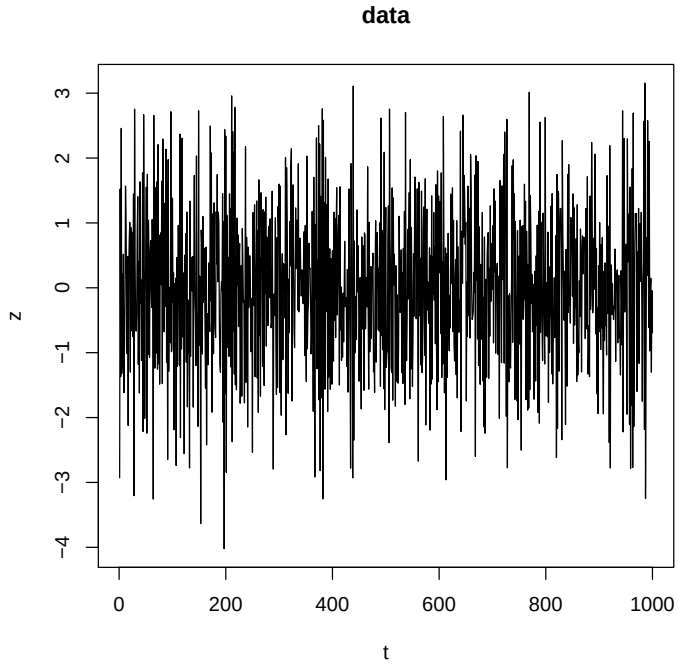

```
> x <- continue(x, Nmif = 20, max.fail = 100, weighted = F)
> x <- continue(x, Nmif = 80, max.fail = 100, weighted = T)
> plot(x)
> print(x)
 1 -variable time series of length: 1000
number of MIF iterations done: 101
```

```
 5  parameter(s) estimated: alpha.1, alpha.3, sigma.1, sigma.3, tau
 2  state variable(s) (STV): x1, x2
 2  initial-value parameter(s) (IVP): x1.0, x2.0
Sigma (parameter scales):
alpha.1 alpha.2 alpha.3 alpha.4 sigma.1 sigma.2 sigma.3     tau    x1.0    x2.0
    0.1     0.1     0.1     0.1     0.1     0.1     0.1     0.1     0.2     0.2

number of particles = 1000
CC = 7 , T0 = 4
cooling factor = 0.95
random walk intensity at last MIF iteration:
 alpha.1  alpha.2  alpha.3  alpha.4  sigma.1  sigma.2  sigma.3      tau
0.000592 0.000592 0.000592 0.000592 0.000592 0.000592 0.000592 0.000592
    x1.0     x2.0
0.001184 0.001184

estimated parameter(s):
alpha.1 alpha.3 sigma.1 sigma.3     tau    x1.0    x2.0
 -1.290  -1.385  -1.109   1.071  -0.365   6.303   5.475

log-likelihood (w/ variable parameters): -1537
number of filtering failures: 0
```




```
> ff.fit <- pfilter(x)
> print(c(loglik = ff.fit$loglik, nfail = ff.fit$nfail))

   loglik     nfail
-1534.519     0.000

> print(c(loglik = ff$loglik, nfail = ff$nfail))

   loglik     nfail
-1535.221     0.000

> print(arima.fit)

Call:
arima(x = ex@data[2, ], order = c(2, 0, 2))

Coefficients:
          ar1     ar2      ma1     ma2  intercept
      -0.0871  0.1424  -0.3373  0.0650    -0.0221
s.e.   0.3767  0.2091   0.3765  0.0836     0.0273

sigma^2 estimated as 1.258:  log likelihood = -1533.84,  aic = 3079.68
```

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

E-mail address: aaron.king@umich.edu

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