

Fitting stochastic COVID model: the inference trio

Andrew Tredennick

4/02/2020

Overview

Given the importance of the COVID-19 pandemic and the the various difficulties inherent in fitting the transmission model, we will use three different methods of inference. First, we will use maximum likelihood by iterated filtering (MIF) to quickly hone in on the likelihood surface. Second, we will use particle MCMC to estimate uncertainty around parameter estimates, using the MIF MLEs as starting points for the MCMC. We will also employ informative priors to make parameters identifiable. Third, and in parallel to the pMCMC, we will use Approximate Bayesian Computation (ABC-MCMC) to also estimate uncertainty around parameters. In concert, the three methods of model fitting should allow us to make robust inference on model parameters and, in turn, generate meaningful forecasts with clearly propagated uncertainty. All model simulation and inference will use `pomp` functions.

Here, I outline the three methods and general approach. Note that none of the analyses are “production runs.” So lack of convergence, and lack of confidence, in paramete estimates is to be expected.

Parameters to estimate

There are 11 parameters in the stochastic model (Table 1). We need to decide on which to fix and which to assign informative or vague priors. In MIF, we can achieve priors by constraining the standard deviation of the random walk for some parameters (I have not actually seen this done, but see no reason why not...).

Table 1: Parameters in the stochastic model.

Parameter	Definition	Fixed	Prior
β_d	Transmission rate of detected infectious individuals		
β_u	Transmission rate of undetected infectious individuals		
β_e	Transmission rate of presymptomatic infectious individuals		
ξ	Factor by which β_d is reduced due to preventative actions		
γ_u	Progression rate from I to C, pre-intervention		
γ_d	Progression rate from I to C, post-intervention		
σ	Progression rate from E to I		
f_0	Fraction of detected E to I, pre-intervention		
f_1	Fraction of detected E to I, post-intervention		
ρ	Case reporting fraction		
θ	Over-dispersion parameter for Negative Binomial observation process		

MIF

We will use MIF to rapidly hone in on the likelihood surface. First, we load the `pomp` object

```
pomp_object <- readRDS("../output/covid-ga-pomp-object.RDS")
```

Then we set the parameters we want to estimate. At the same time, we can set the standard deviation of the random walk for exploring parameter space. Here, I've set `sd = 0` for all the β parameters, treating them as known/fixed. Recall that the β parameters are on the log scale. They are exponentiated within the process model itself.

```
params_to_estimate <- c("beta_d", "beta_u", "beta_e", "beta_red_factor",  
                        "gamma_u", "gamma_d", "detect_frac_0")  
  
params_perts <- rw.sd(beta_d = 0, # change to let it vary  
                      beta_u = 0, # change to let it vary  
                      beta_e = 0, # change to let it vary  
                      beta_red_factor = 0.02,  
                      gamma_u = 0.02,  
                      gamma_d = 0.02,  
                      detect_frac_0 = 0.02)  
  
curr_theta <- coef(pomp_object)  
print(round(curr_theta, 2))
```

##	beta_d	beta_u	beta_e	beta_red_factor
##	-1.548e+01	-1.756e+01	-1.848e+01	5.000e-01
##	t_int1	t_int2	t_int3	gamma_u
##	1.200e+01	1.200e+01	1.200e+01	4.000e-01
##	gamma_d	detect_frac_0	detect_frac_1	sigma
##	2.000e+00	1.000e-01	5.000e-01	1.080e+00
##	rho	theta	S_0	E1_0
##	5.000e-01	1.000e+02	1.060e+07	3.500e+01
##	E2_0	E3_0	E4_0	E5_0
##	3.500e+01	3.500e+01	3.500e+01	3.500e+01
##	E6_0	I1_0	I2_0	I3_0
##	3.500e+01	1.400e+01	1.400e+01	1.400e+01
##	I4_0	Iu1_0	Iu2_0	Iu3_0
##	1.400e+01	1.110e+02	1.110e+02	1.110e+02
##	Iu4_0	C_0	Ru_0	
##	1.110e+02	1.000e+00	1.000e+00	

We also need a “proposal” function that prerturbs the initial values by a little bit for each MIF run.

```
prop_func <- function(theta) {  
  betas <- theta[c("beta_d", "beta_u", "beta_e")]  
  one <- rnorm(n = length(betas), mean = betas, sd = 0) # update sd if desired  
  others <- theta[-(which(names(theta) %in% names(betas)))]  
  two <- rlnorm(n = (length(others)),  
               meanlog = log(others),  
               sdlog = 1)  
  out <- c(one, two)  
  names(out) <- names(theta)  
  return(out)  
}
```

Now we can run the `mif2` function in parallel from several different parameter starting points.

```

num_particles <- 2000
num_mif_iterations <- 5
num_cores <- parallel::detectCores() - 1 # alter as needed
foreach (i = 1:num_cores,
         .combine = c,
         .export = c("params_perts",
                     "prop_func",
                     "curr_theta")) %dopar% {
  theta_guess <- curr_theta
  theta_guess[params_to_estimate] <- prop_func(curr_theta[params_to_estimate])
  mif2(pomp_object, Nmif = num_mif_iterations, params = theta_guess,
       Np = num_particles, cooling.fraction = 0.5, rw.sd = params_perts)
} -> mifs

```

```

## Warning: executing %dopar% sequentially: no parallel backend registered
## Warning: filtering failure at last filter iteration:
## Warning: in 'mif2': 5 filtering failures occurred.
## Warning: filtering failure at last filter iteration:
## Warning: in 'mif2': 7 filtering failures occurred.
## Warning: filtering failure at last filter iteration:
## Warning: in 'mif2': 7 filtering failures occurred.
## Warning: filtering failure at last filter iteration:
## Warning: in 'mif2': 4 filtering failures occurred.
## Warning: filtering failure at last filter iteration:
## Warning: in 'mif2': 3 filtering failures occurred.

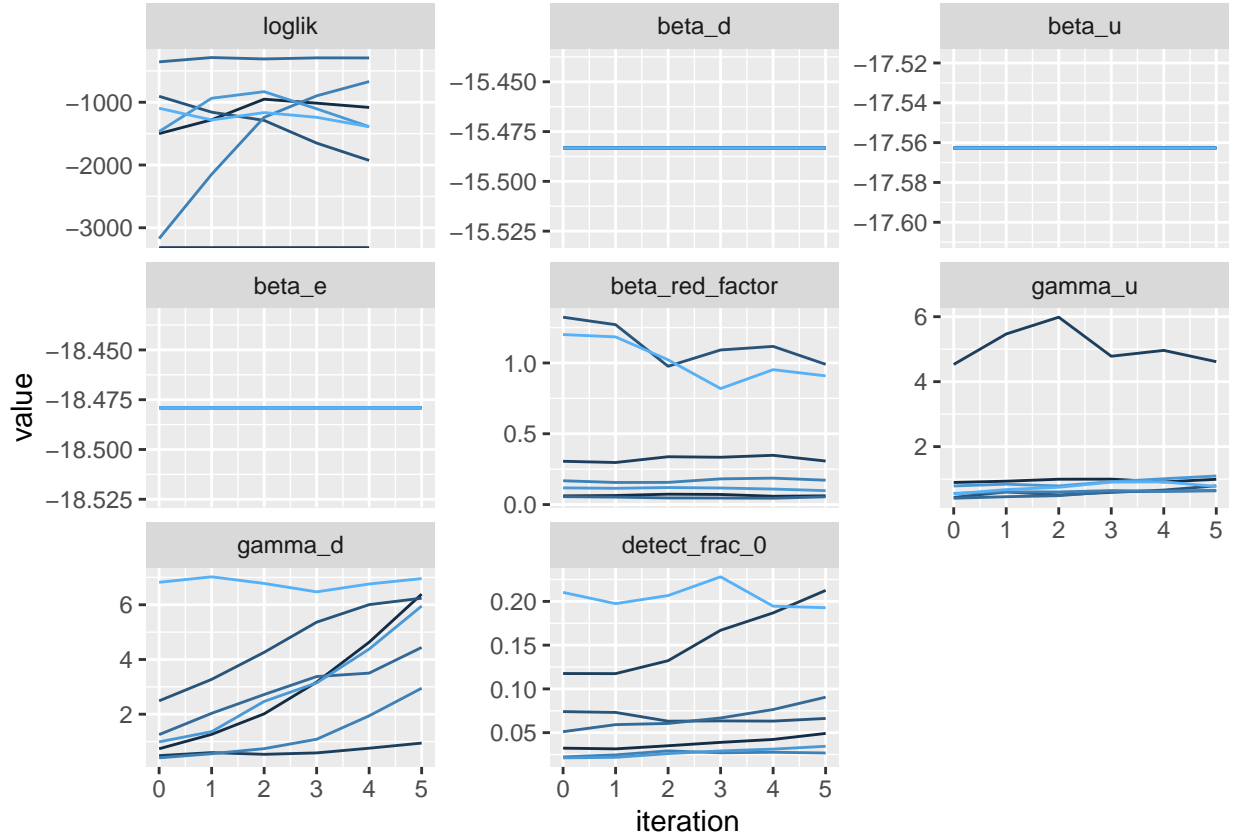
```

We can look at the MIF traces to diagnose convergence (not expected here as these are demonstration runs).

```

mifs %>%
  traces() %>%
  melt() %>%
  filter(variable %in% c("loglik", params_to_estimate)) %>%
  ggplot(aes(x=iteration, y=value, group=L1, color=L1))+
  geom_line()+
  facet_wrap(~variable, scales="free_y")+
  guides(color=FALSE)

```



Lastly, we use a replicate particle filter to compute the likelihood of the parameters at the last MIF iteration. The parameter set with the highest likelihood are the MLEs (again, not in this demonstration phase). We save the MLEs as initial conditions for particle MCMC. (Don't mind the particle failure warnings. There are very few and I expect they will disappear as we run more MIF iterations.)

```
pf1 <- foreach(mf = mifs, .combine = c) %dopar% {
  pf <- replicate(n = 5, logLik(pfilter(mf, Np = 2000)))
  logmeanexp(pf)
}
```

```
## Warning: in 'pfilter': 2 filtering failures occurred.
```

```
## Warning: in 'pfilter': 3 filtering failures occurred.
```

```
## Warning: in 'pfilter': 3 filtering failures occurred.
```

```
## Warning: in 'pfilter': 3 filtering failures occurred.
```

```
## Warning: in 'pfilter': 3 filtering failures occurred.
```

```
mf1 <- mifs[[which.max(pf1)]]
theta_mif <- coef(mf1)
print(round(theta_mif, 2))
```

```
##      beta_d      beta_u      beta_e beta_red_factor
##    -1.548e+01    -1.756e+01    -1.848e+01      5.000e-02
##      t_int1      t_int2      t_int3      gamma_u
##      1.200e+01      1.200e+01      1.200e+01      6.400e-01
```

##	gamma_d	detect_frac_0	detect_frac_1	sigma
##	4.440e+00	9.000e-02	5.000e-01	1.080e+00
##	rho	theta	S_0	E1_0
##	5.000e-01	1.000e+02	1.060e+07	3.500e+01
##	E2_0	E3_0	E4_0	E5_0
##	3.500e+01	3.500e+01	3.500e+01	3.500e+01
##	E6_0	I1_0	I2_0	I3_0
##	3.500e+01	1.400e+01	1.400e+01	1.400e+01
##	I4_0	Iu1_0	Iu2_0	Iu3_0
##	1.400e+01	1.110e+02	1.110e+02	1.110e+02
##	Iu4_0	C_0	Ru_0	
##	1.110e+02	1.000e+00	1.000e+00	

pMCMC

Using the lightly edited `pomp` model generated from `./code/make-pomp-object.R`, I used the `pomp::pmcmc` function to estimate parameters. The initial conditions for the MCMC are the MIF MLEs.

First, define the prior density. The priors are based on the initial values chosen by eye and the variance around the means was chosen by looking at the resulting distributions. Note that `beta_d`, `beta_u`, and `beta_e` are all exponentiated within the process model itself, meaning that estimation occurs on the log scale. I did this because `pomp::pmcmc` *does not* adhere to the parameter transformations imposed on the `pomp` object (<https://github.com/kingaa/pomp/issues/22#issuecomment-227276797>).

```
prior_dens <- Csnippet(
  "
  lik = dnorm(beta_d, log(2e-7), 0.8, 1) +
    dnorm(beta_u, log(5e-8), 0.4, 1) +
    dunif(beta_red_factor, 0.01, 1, 1) +
    dunif(detect_frac_0, 0.01, 0.6, 1) +
    dnorm(beta_e, log(5e-8), 0.4, 1) +
    dlnorm(gamma_u, log(0.5), 1, 1) +
    dlnorm(gamma_d, log(0.5), 1, 1);
  if (!give_log) lik = exp(lik);
  "
)
```

Now we can run the particle MCMC.

```
n <- 1 # number of mcmc chains

# Parameters to estimate
estpars <- c("beta_d", "beta_u", "beta_e", "beta_red_factor",
            "gamma_u", "gamma_d", "detect_frac_0", "theta")

# Set noise level for parameter random walk for proposals
rw.sd <- c(beta_d = 0.05, beta_u = 0.05, beta_e = 0.05,
           beta_red_factor = 0.005, gamma_u = 0.1, gamma_d = 0.1,
           detect_frac_0 = 0.005, theta = 0.5)

out_mcmc <- pmcmc(
  pomp(
    covid_ga_pomp,
    params = theta_mif, # set to MLEs
    dprior = prior_dens,
    paramnames = c("beta_d", "beta_u", "beta_e", "beta_red_factor",
```

```

        "gamma_u", "gamma_d", "detect_frac_0", "theta")
    ),
    Nmcmc = 2000,
    Np = 2000,
    proposal = mvn.diag.rw(rw.sd),
    verbose = TRUE
)

```

Note that the above results in no errors, no warnings, and no particle failures. The acceptance rates hovers around 0.2 (starting high around 0.5, and ending low around 0.1, by the final MCMC iterations).

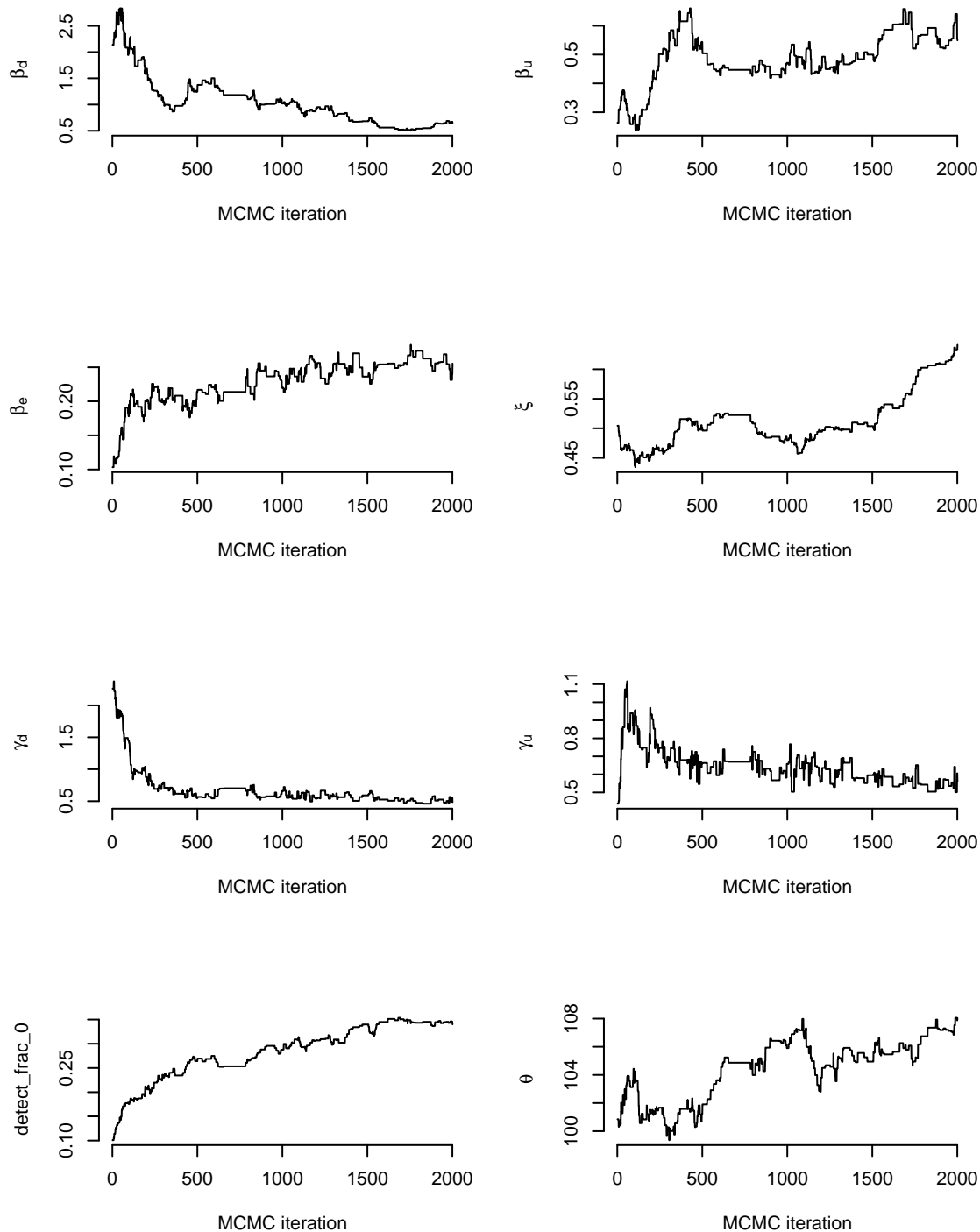
Let's take a look at the chains.

```

chain <- as.data.frame(out_mcmc@traces)

par(mfrow = c(4,2))
plot(exp(chain$beta_d)*10600000, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = expression(beta[d]))
plot(exp(chain$beta_u)*10600000, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = expression(beta[u]))
plot(exp(chain$beta_e)*10600000, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = expression(beta[e]))
plot(chain$beta_red_factor, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = expression(xi))
plot(chain$gamma_d, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = expression(gamma[d]))
plot(chain$gamma_u, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = expression(gamma[u]))
plot(chain$detect_frac_0, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = "detect_frac_0")
plot(chain$theta, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = expression(theta))

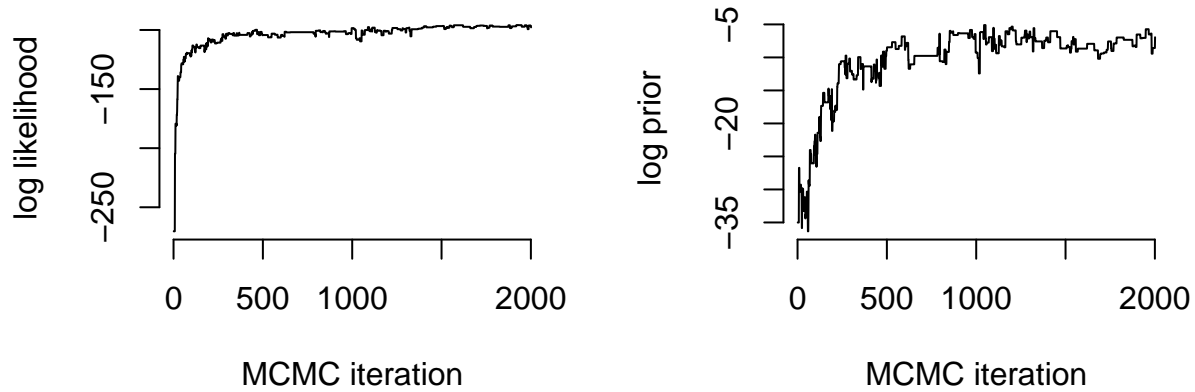
```



And we can look at the log likelihood and the log prior density over the MCMC iterations.

```
par(mfrow = c(1,2))
plot(chain$loglik, type = "l", bty = "n",
      xlab = "MCMC iteration", ylab = "log likelihood")
```

```
plot(chain$log.prior, type = "l", bty = "n",
     xlab = "MCMC iteration", ylab = "log prior")
```



ABC-MCMC

To bolster the parameter estimates, we will also use ABC-MCMC. Again, we start from the MIF MLEs. But first, we have define a set of summary statistics, S , used to compare model simulations to the data. Note that `pomp` calls these probes.

Each summary statistic needs its own function.

```
get_stat_times <- function(obs_cases) {
  x <- obs_cases
  ma <- function(x, n = 5){stats::filter(x, rep(1 / n, n), sides = 2)}
  stat_times <- ma(diff(log(x)), n = 5)
  d0 <- min(which(stat_times > 0.1))
  suppressWarnings(
    d1 <- d0 + min(which(stat_times[d0:length(stat_times)] < -0.01))
  )
  return(c(d0 = d0, d1 = d1))
}

ds <- get_stat_times(as.numeric(pomp_object@data))
d0 <- ds["d0"]
d1 <- ds["d1"]
if(is.infinite(d1)) d1 <- ncol(pomp_object@data) - 1
d2 <- ncol(pomp_object@data)

max1 <- eval(parse(text = paste0("function(x){ max(log(x[1, 1:(", d0, "-1]] + 10)) }")))
max2 <- eval(parse(text = paste0("function(x){ max(log(x[1, ", d0, ":", d1, "] + 10)) }")))
max3 <- eval(parse(text = paste0("function(x){ max(log(x[1,(", d1, "+1):", d2, "] + 10)) }")))
maxall <- function(x){ max(log(x[1,])) }
maxday <- function(x){ which.max(x[1,]) }

cumi1 <- eval(parse(text = paste0("function(x){ sum(log(x[1, 1:(", d0, "-1]] + 10)) }")))
cumi2 <- eval(parse(text = paste0("function(x){ sum(log(x[1, ", d0, ":", d1, "] + 10)) }")))
```



```
cumi3 <- eval(parse(text = paste0("function(x){ sum(log(x[1, (", d1, "+1):", d2, "] + 10)) }"))))

exp1 <- eval(parse(text = paste0("function(x) { max(x[1, ]) / which.max(x[1, ]) -", d0, "}")"))
regcoef <- function(x) { as.numeric(coef(lm(x[1,] ~ seq_along(x[1,]))) [2]) }
```

I use the same prior density as used for the pMCMC. We also have to define ϵ , which is the threshold distance between the data summary statistics and the simulated summary statistics below which a parameter proposal is accepted. After a bit of trial and error, I settled on $\epsilon = 5$. With the prior density and the random walk settings, this lead to an acceptance ratio of about 2.5-5%. We also need to define the scale of the summary statistics. I used a method that King used in the `pomp` introduction paper. Basically, I run the `pomp::probe` function using the proposed probes and compute the scale as the standard deviation of each summary statistic over the replicate simulations (see `psim <- ...` section in code chunk below).

So, now we can run the ABC-MCMC (it's fast!).

```
n <- 1 # number of mcmc chains

estpars <- c("beta_d", "beta_u", "beta_e", "beta_red_factor",
            "gamma_u", "gamma_d", "detect_frac_0", "theta")

# Set noise level for parameter random walk for proposals
rw.sd <- rep(0.1, length(estpars))
names(rw.sd) <- estpars
rw.sd["theta"] <- 1 # larger perturbation for theta

plist <- list(
  max1, max2, max3, maxall,
  cumi1, cumi2, cumi3, exp1, regcoef
)
psim <- probe(pomp_object, probes = plist, nsim = 1000)
# plot(psim)
scale.dat <- apply(psim@simvals, 2, sd)

out_abc <- abc(
  pomp(
    pomp_object,
    params = theta_mif, # set to MLEs
    dprior = prior_dens,
    paramnames = c("beta_d", "beta_u", "beta_e", "beta_red_factor",
                  "gamma_u", "gamma_d", "detect_frac_0", "theta")
  ),
  Nabc = 20000,
  epsilon = 5,
  scale = scale.dat,
  proposal = mvn.diag.rw(rw.sd),
  probes = plist
)
```

And we can plot the chains. Note that with many, many more iterations I was able to get quite good mixing and stationarity.

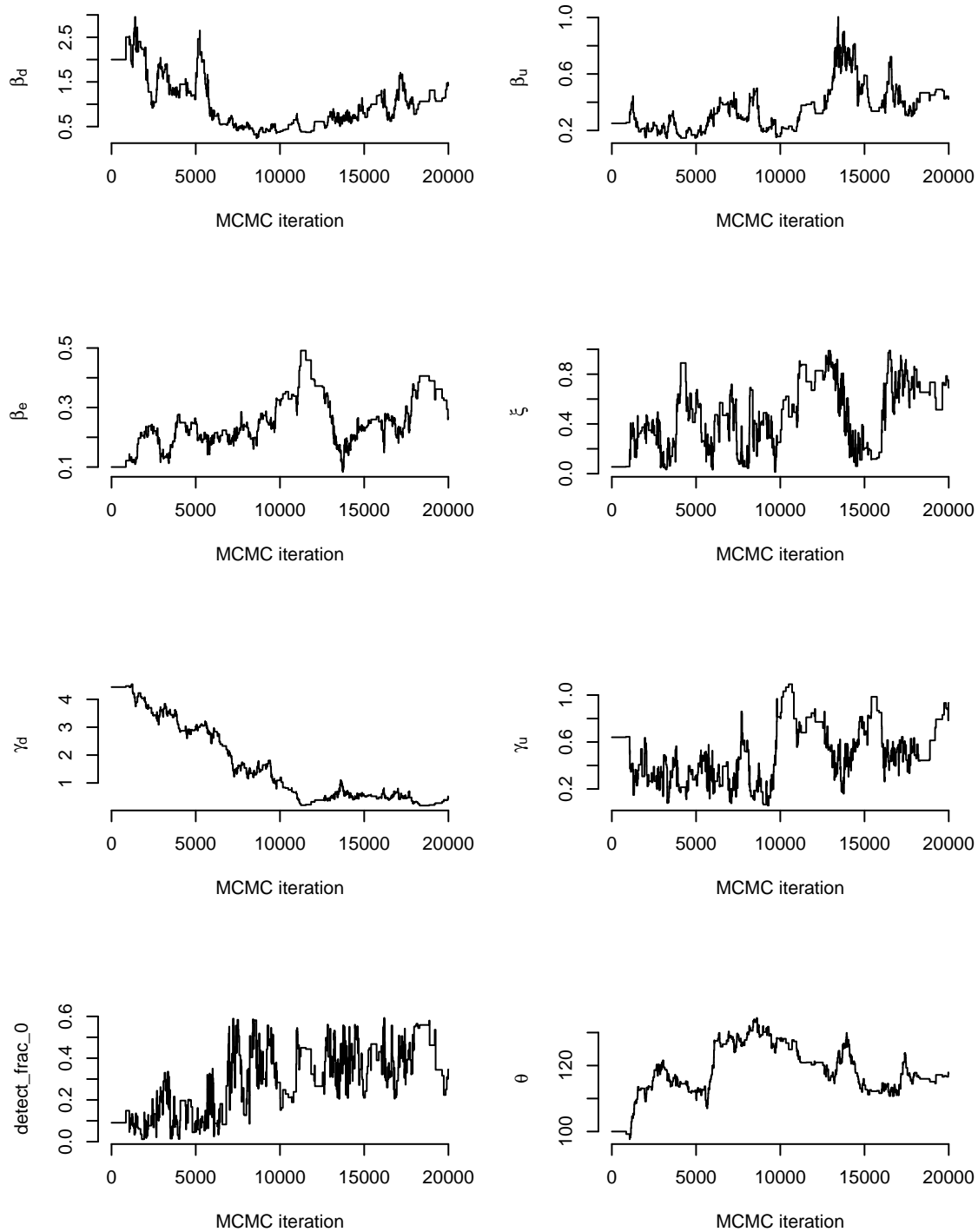
```
chain <- as.data.frame(out_abc@traces)

par(mfrow = c(4,2))
plot(exp(chain$beta_d)*10600000, type = "l", bty = "n",
```

```

      xlab = "MCMC iteration", ylab = expression(beta[d]))
plot(exp(chain$beta_u)*10600000, type = "l", bty = "n",
      xlab = "MCMC iteration", ylab = expression(beta[u]))
plot(exp(chain$beta_e)*10600000, type = "l", bty = "n",
      xlab = "MCMC iteration", ylab = expression(beta[e]))
plot(chain$beta_red_factor, type = "l", bty = "n",
      xlab = "MCMC iteration", ylab = expression(xi))
plot(chain$gamma_d, type = "l", bty = "n",
      xlab = "MCMC iteration", ylab = expression(gamma[d]))
plot(chain$gamma_u, type = "l", bty = "n",
      xlab = "MCMC iteration", ylab = expression(gamma[u]))
plot(chain$detect_frac_0, type = "l", bty = "n",
      xlab = "MCMC iteration", ylab = "detect_frac_0")
plot(chain$theta, type = "l", bty = "n",
      xlab = "MCMC iteration", ylab = expression(theta))

```



We can also compare the posterior (black) and prior (red) distributions.

```
par(mfrow = c(4, 2))
plot(density(exp(chain$beta_d)*10600000, adjust = 1), bty = "n",
     ylab = "Density", xlab = expression(beta[d]), main = "")
```

```

lines(density(exp(rnorm(100000, log(2e-7), 0.8))*10600000, adjust = 1),
      col = "red", lty = 2)
plot(density(exp(chain$beta_u)*10600000, adjust = 1), bty = "n",
      ylab = "Density", xlab = expression(beta[u]), main = "")
lines(density(exp(rnorm(100000, log(5e-8), 0.4))*10600000, adjust = 1),
      col = "red", lty = 2)
plot(density(exp(chain$beta_e)*10600000, adjust = 1), bty = "n",
      ylab = "Density", xlab = expression(beta[e]), main = "")
lines(density(exp(rnorm(100000, log(5e-8), 0.4))*10600000, adjust = 1),
      col = "red", lty = 2)
plot(density(chain$beta_red_factor, adjust = 1), bty = "n",
      ylab = "Density", xlab = expression(xi), main = "")
lines(x = seq(0, 1, by = 0.01), dunif(x = seq(0, 1, by = 0.01), 0.01, 1),
      col = "red", lty = 2)
plot(density(chain$gamma_u), bty = "n", ylab = "Density",
      xlab = expression(gamma[u]), main = "")
lines(density(rlnorm(100000, log(0.5), 1)),
      col = "red", lty = 2)
plot(density(chain$gamma_d), bty = "n", ylab = "Density",
      xlab = expression(gamma[d]), main = "")
lines(density(rlnorm(100000, log(0.5), 1)),
      col = "red", lty = 2)
plot(density(chain$detect_frac_0, adjust = 1), bty = "n",
      ylab = "Density", xlab = "detect_frac_0", main = "")
lines(x = seq(0, 1, by = 0.01), dunif(x = seq(0, 1, by = 0.01), 0.01, 0.6),
      col = "red", lty = 2)
plot(density(chain$theta), bty = "n", xlab = expression(theta), main = "")

```

