Tutorial: cosine trend model

EB, PJ, MG, CR

June 8, 2017

Setting

This script applies the proposed ABC with Wasserstein distance, to approximate the posterior distribution in a cosine trend model.

The model specifies $Y_t \sim \mathcal{N}(\cos(2\pi\omega t + \phi), \sigma^2)$, the parameters are (ω, ϕ, σ^2), the prior is ... The data are generated from ...

We begin by loading the package, registering multiple cores, setting the random number generator, etc.

```
# load package
library(winference)
# register parallel cores
registerDoParallel(cores = detectCores())
# remove all
rm(list = ls())
# apply preferences for ggplotting
require(gridExtra)
theme_set(theme_bw())
# set RNG seed
set.seed(11)
```

Data and model

We define the model and generate some data

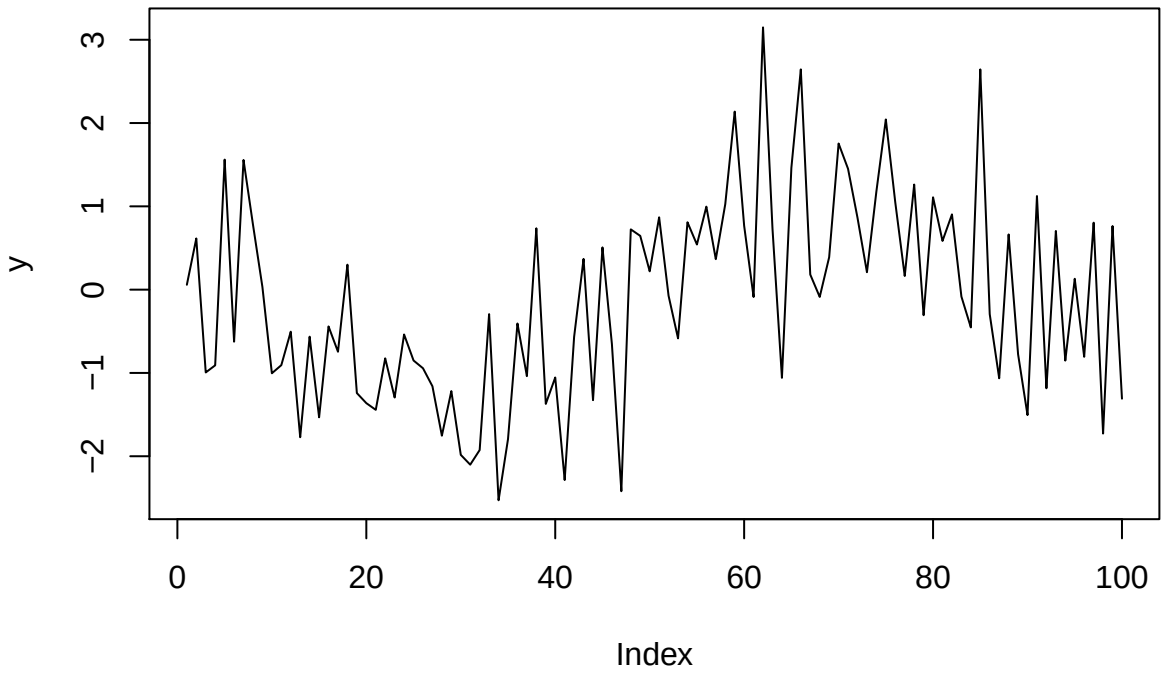
```
nobservations <- 100

rprior <- function(nparticles, ...) {
    omegas <- runif(nparticles, min = 0, max = 1/10)
    phis <- runif(nparticles, min = 0, max = 2 * pi)
    logsigma <- rnorm(nparticles)
    return(cbind(omegas, phis, logsigma))
}

# evaluate the log-density of the prior, for each particle
dprior <- function(thetas, ...) {
    logdensities <- dnorm(thetas[, 3], 0, 1, log = TRUE)
    logdensities <- logdensities + dunif(thetas[, 1], min = 0, max = 1/10, log = TRUE)
    logdensities <- logdensities + dunif(thetas[, 2], min = 0, max = 2 * pi,
        log = TRUE)
    return(logdensities)
}
# function to generate a dataset for each theta value
simulate <- function(theta) {
    observations <- cos(2 * pi * theta[1] * (1:nobservations) + theta[2]) +
        rnorm(nobservations, mean = 0, sd = exp(theta[3]))
    return(observations)
```

```
}
#
target <- list(rprior = rprior, dprior = dprior, simulate = simulate, parameter_names = c("omega",
    "phi", "logsigma"), thetadim = 3, ydim = 1, parameters = list())

theta_star <- c(1/80, pi/4, 0)
obs <- target$simulate(theta_star)
plot(obs, type = "l", ylab = "y")
```



Distance calculation and Monte Carlo algorithm

We define a way of calculating a distance between fake data and the observed data. Here we use curve matching: we augment each observation y_t with the time index t, and then define a ground metric on the space of (t, y_t).

```
lambda <- 1
multiplier <- lambda * (max(obs) - min(obs))
augment <- function(series) rbind(series, multiplier * (1:length(series))/length(series))
augmented_obs <- augment(obs)

# we're using the Hilbert distance
compute_distance <- function(y_fake) {
    augmented_y_fake <- augment(y_fake)
    # return(hilbert_distance(augmented_obs, augmented_y_fake, p = 1, ground_p =
    # 2))
    return(swap_distance(augmented_obs, augmented_y_fake, p = 1, ground_p = 2,
        tolerance = 1e-05)$distance)
}
# test: compute_d(target$simulate(target$rprior(1)))
```

We specify algorithmic parameters in a list.

```r
# algorithmic parameters: number of particles, number of moves per
# rejuvenation step, proposal distribution, the number of steps to perform
# in total, the diversity parameter used in the threshold adaptation, the
# number of hits to use in the r-hit kernel, and the maximum number of
# trials to use in the r-hit kernel before rejecting.
param_algo <- list(nthetas = 1024, nmoves = 1, proposal = mixture_rmixmod(),
    minimum_diversity = 0.5, R = 2, maxtrials = 1e+05)
```

We now run the algorithm.

```r
# now run the algorithm until 5e5 model simulations have been performed
wsmcresults <- wsmc(compute_distance, target, param_algo, maxsimulation = 5e5)
```

```
## step 1 completed, in 1.042 seconds, 1024 distances calculated
## step 2... running until # distances calculated >= 5e+05
##   acceptance rates: 60.25391 %, threshold = 0.6805984 , min. dist. = 0.4022497
##   total # distances calculated: 4613 (for this step: 3589)
##   total time spent: 3.555 seconds (for this step: 2.513 seconds)
## step 3... running until # distances calculated >= 5e+05
##   acceptance rates: 55.76172 %, threshold = 0.6162271 , min. dist. = 0.4022497
##   total # distances calculated: 9727 (for this step: 5114)
##   total time spent: 6.792 seconds (for this step: 3.237 seconds)
## step 4... running until # distances calculated >= 5e+05
##   acceptance rates: 51.46484 %, threshold = 0.5882843 , min. dist. = 0.3853963
##   total # distances calculated: 16981 (for this step: 7254)
##   total time spent: 10.943 seconds (for this step: 4.151 seconds)
## step 5... running until # distances calculated >= 5e+05
##   acceptance rates: 47.94922 %, threshold = 0.5684011 , min. dist. = 0.3581171
##   total # distances calculated: 27955 (for this step: 10974)
##   total time spent: 16.729 seconds (for this step: 5.786 seconds)
## step 6... running until # distances calculated >= 5e+05
##   acceptance rates: 42.96875 %, threshold = 0.5534788 , min. dist. = 0.3768071
##   total # distances calculated: 44049 (for this step: 16094)
##   total time spent: 25.804 seconds (for this step: 9.075 seconds)
## step 7... running until # distances calculated >= 5e+05
##   acceptance rates: 39.55078 %, threshold = 0.5410348 , min. dist. = 0.347245
##   total # distances calculated: 64239 (for this step: 20190)
##   total time spent: 37.335 seconds (for this step: 11.531 seconds)
## step 8... running until # distances calculated >= 5e+05
##   acceptance rates: 36.42578 %, threshold = 0.5297448 , min. dist. = 0.347245
##   total # distances calculated: 89371 (for this step: 25132)
##   total time spent: 50.759 seconds (for this step: 13.424 seconds)
## step 9... running until # distances calculated >= 5e+05
##   acceptance rates: 28.22266 %, threshold = 0.5186506 , min. dist. = 0.3453301
##   total # distances calculated: 109005 (for this step: 19634)
##   total time spent: 61.345 seconds (for this step: 10.586 seconds)
## step 10... running until # distances calculated >= 5e+05
##   acceptance rates: 25.97656 %, threshold = 0.5096611 , min. dist. = 0.3453301
##   total # distances calculated: 123739 (for this step: 14734)
##   total time spent: 69.535 seconds (for this step: 8.19 seconds)
## step 11... running until # distances calculated >= 5e+05
##   acceptance rates: 30.17578 %, threshold = 0.4997761 , min. dist. = 0.3453301
##   total # distances calculated: 134463 (for this step: 10724)
##   total time spent: 75.686 seconds (for this step: 6.151 seconds)
## step 12... running until # distances calculated >= 5e+05
```

```
##   acceptance rates: 33.98438 %, threshold = 0.488573 , min. dist. = 0.3490015
##   total # distances calculated: 144063 (for this step: 9600)
##   total time spent: 80.966 seconds (for this step: 5.28 seconds)
## step 13... running until # distances calculated >= 5e+05
##   acceptance rates: 44.92188 %, threshold = 0.4754115 , min. dist. = 0.3242744
##   total # distances calculated: 151090 (for this step: 7027)
##   total time spent: 84.946 seconds (for this step: 3.98 seconds)
## step 14... running until # distances calculated >= 5e+05
##   acceptance rates: 46.67969 %, threshold = 0.454917 , min. dist. = 0.3242744
##   total # distances calculated: 157507 (for this step: 6417)
##   total time spent: 88.599 seconds (for this step: 3.653 seconds)
## step 15... running until # distances calculated >= 5e+05
##   acceptance rates: 50.58594 %, threshold = 0.434209 , min. dist. = 0.3254691
##   total # distances calculated: 164737 (for this step: 7230)
##   total time spent: 92.706 seconds (for this step: 4.107 seconds)
## step 16... running until # distances calculated >= 5e+05
##   acceptance rates: 44.43359 %, threshold = 0.4167614 , min. dist. = 0.3254691
##   total # distances calculated: 173433 (for this step: 8696)
##   total time spent: 97.541 seconds (for this step: 4.835 seconds)
## step 17... running until # distances calculated >= 5e+05
##   acceptance rates: 43.84766 %, threshold = 0.4026009 , min. dist. = 0.3167099
##   total # distances calculated: 184869 (for this step: 11436)
##   total time spent: 103.744 seconds (for this step: 6.203 seconds)
## step 18... running until # distances calculated >= 5e+05
##   acceptance rates: 42.38281 %, threshold = 0.3914741 , min. dist. = 0.2918619
##   total # distances calculated: 201323 (for this step: 16454)
##   total time spent: 112.401 seconds (for this step: 8.657 seconds)
## step 19... running until # distances calculated >= 5e+05
##   acceptance rates: 43.35938 %, threshold = 0.3825034 , min. dist. = 0.2918619
##   total # distances calculated: 222061 (for this step: 20738)
##   total time spent: 124.015 seconds (for this step: 11.614 seconds)
## step 20... running until # distances calculated >= 5e+05
##   acceptance rates: 41.60156 %, threshold = 0.3741942 , min. dist. = 0.2918619
##   total # distances calculated: 249904 (for this step: 27843)
##   total time spent: 140.309 seconds (for this step: 16.294 seconds)
## step 21... running until # distances calculated >= 5e+05
##   acceptance rates: 40.42969 %, threshold = 0.3657035 , min. dist. = 0.2918619
##   total # distances calculated: 293120 (for this step: 43216)
##   total time spent: 163.66 seconds (for this step: 23.351 seconds)
## step 22... running until # distances calculated >= 5e+05
##   acceptance rates: 42.28516 %, threshold = 0.359652 , min. dist. = 0.2918619
##   total # distances calculated: 352572 (for this step: 59452)
##   total time spent: 196.495 seconds (for this step: 32.835 seconds)
## step 23... running until # distances calculated >= 5e+05

## Warning in mixmodCluster(data = data.frame(thetas_check), nbCluster =
## nclust, : All models got errors!

## [1] "error in fitting MCMC proposal, trying Rmixmod 5 times.."
##   acceptance rates: 43.65234 %, threshold = 0.3535059 , min. dist. = 0.3005248
##   total # distances calculated: 433561 (for this step: 80989)
##   total time spent: 239.416 seconds (for this step: 42.921 seconds)
## step 24... running until # distances calculated >= 5e+05
##   acceptance rates: 39.55078 %, threshold = 0.347923 , min. dist. = 0.2915512
##   total # distances calculated: 549601 (for this step: 116040)
```

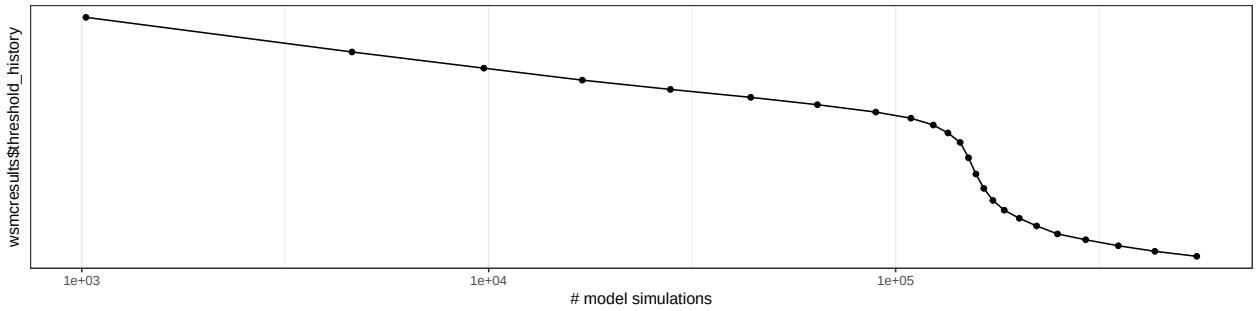
```
##   total time spent: 308.095 seconds (for this step: 68.679 seconds)
```

Now we can look at the ouput, with various plots.

```
# we have access to all the generated particles, distances, and thresholds
names(wsmcresults)
```

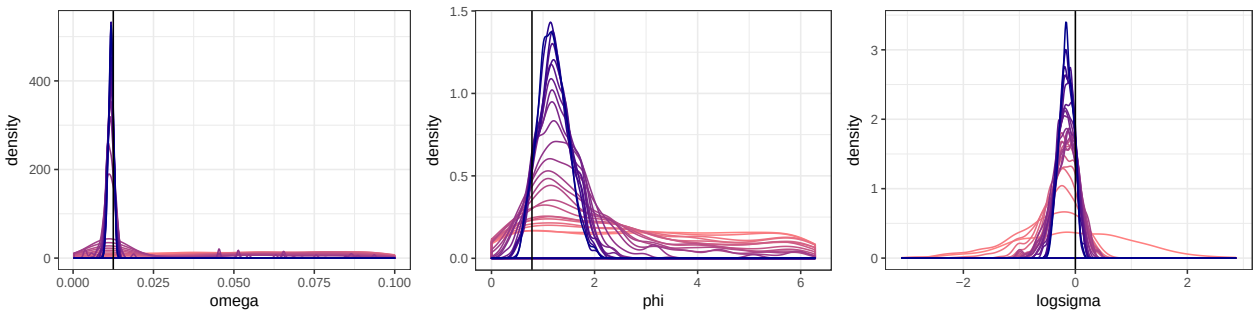
```
##  [1] "thetas_history"    "distances_history" "threshold_history"
##  [4] "param_algo"        "latest_y"          "ncomputed"
##  [7] "compute_d"         "target"            "normcst"
## [10] "compute_times"
```

```
# let's plot some of the output, for instance the thresholds against the
# cumulative number of model simulations (which is equal to the number of
# distances calculated)
qplot(x = cumsum(wsmcresults$ncomputed), y = wsmcresults$threshold_history,
    geom = "line") + scale_y_log10() + scale_x_log10() + xlab("# model simulations") +
    geom_point()
```



We can look at the distributions of parameters.

```
# and let's look at the parameters themselves
grid.arrange(plot_marginal(wsmcresults, i = 1) + geom_vline(xintercept = theta_star[1]),
    plot_marginal(wsmcresults, i = 2) + geom_vline(xintercept = theta_star[2]),
    plot_marginal(wsmcresults, i = 3) + geom_vline(xintercept = theta_star[3]),
    ncol = 3)
```



Posterior samples via MCMC

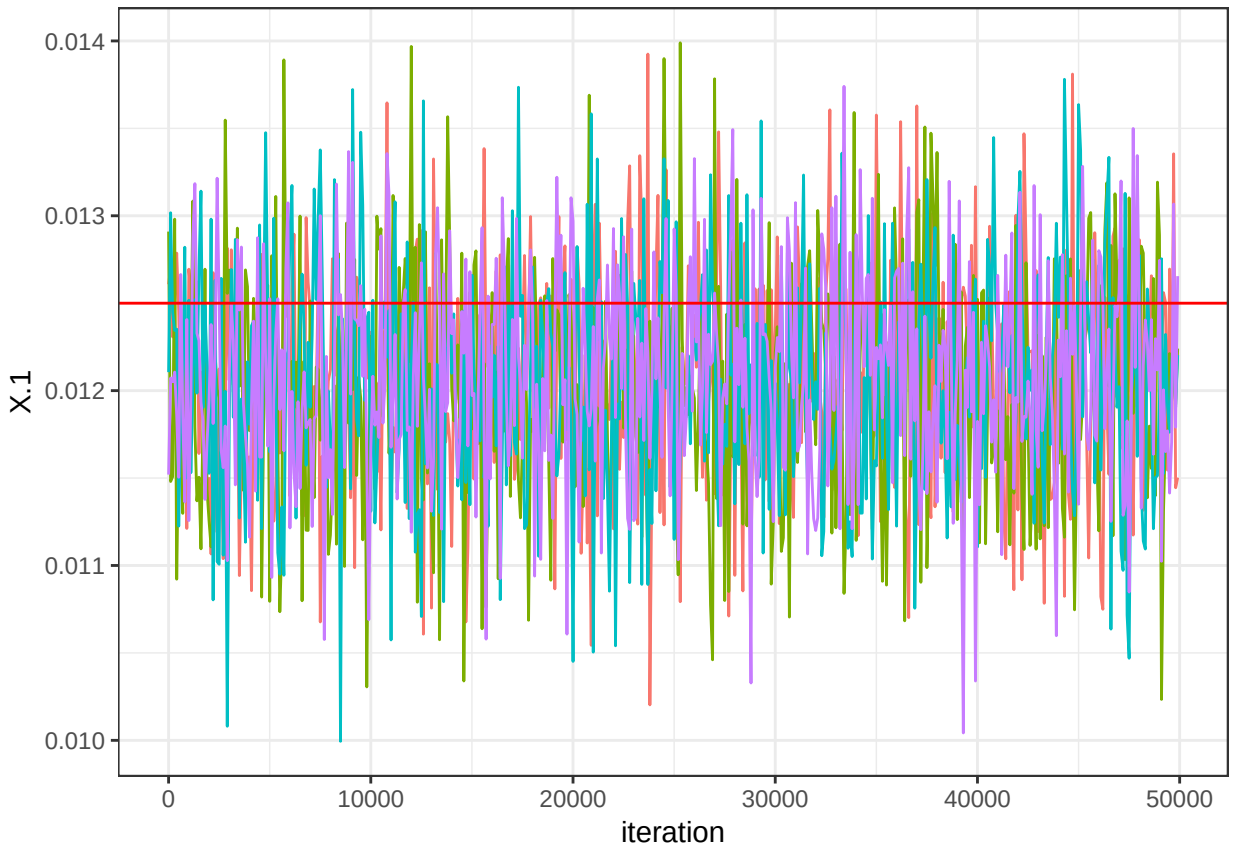
We use the WABC output to initialize a Metropolis-Hastings algorithm.

```
# define log-likelihood function
target$loglikelihood <- function(thetas, ys, ...) {
    evals <- rep(0, nrow(thetas))
    for (itheta in 1:nrow(thetas)) {
```

```
        backbone <- cos(2 * pi * thetas[itheta, 1] * (1:nobservations) + thetas[itheta,
            2])
        evals[itheta] <- sum(dnorm(ys, mean = backbone, sd = exp(thetas[itheta,
            3]), log = TRUE))
    }
    return(evals)
}
thetas <- wsmcresults$thetas_history[[length(wsmcresults$thetas_history)]]
thetas_cov <- cov(thetas)
# initial states of the Markov chain
theta_init <- thetas[sample(x = 1:nrow(thetas), 4), ]
# tuning parameters
tuning_parameters <- list(niterations = 50000, nchains = nrow(theta_init), cov_proposal = thetas_cov,
    adaptation = 0, init_chains = theta_init)
# run adaptive MH scheme
mh <- metropolishastings(obs, target, tuning_parameters)
```

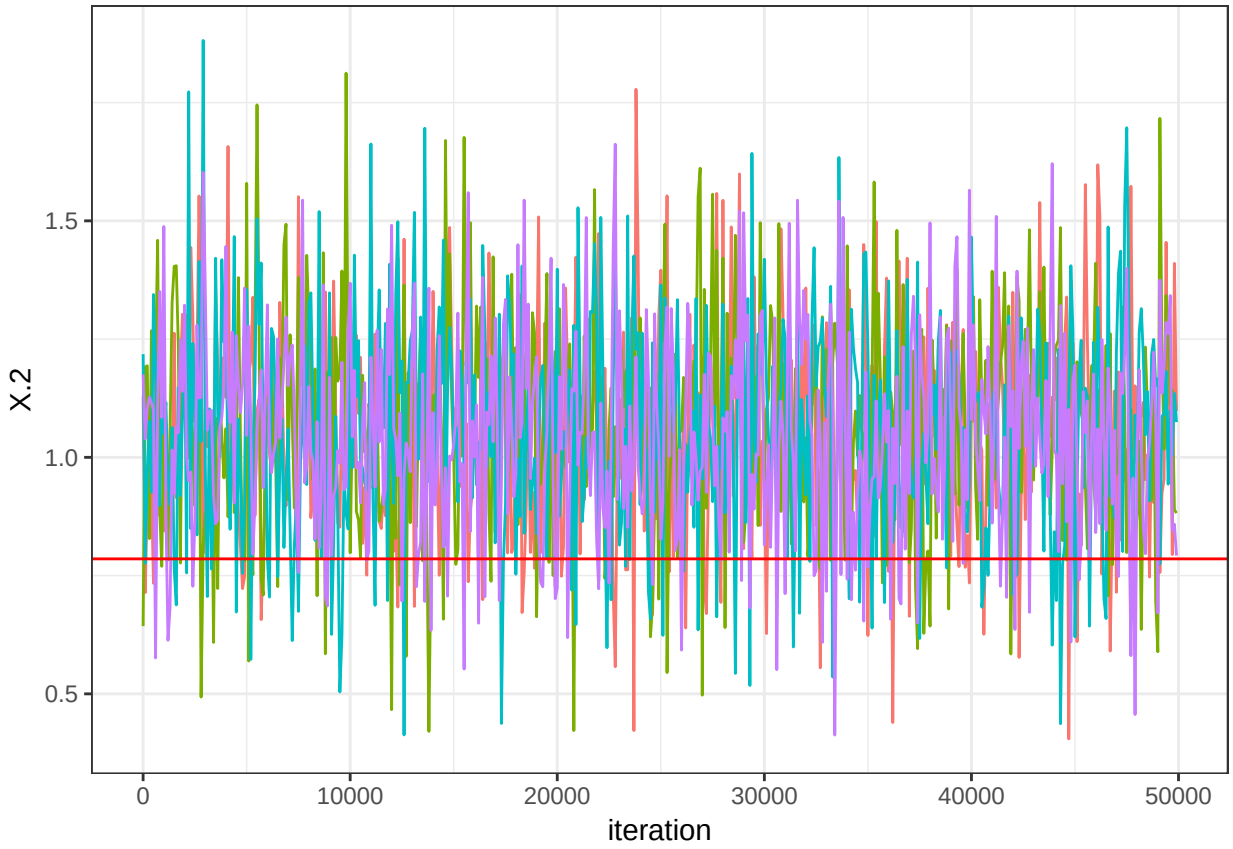
```
## average acceptance: 31.797 %
```

```
burnin <- 0
chain.df <- mhchainlist_to_dataframe(mh$chains)
g1 <- ggplot(chain.df %>% filter(iteration > burnin, iteration%%100 == 1), aes(x = iteration,
    y = X.1, group = ichain, colour = factor(ichain))) + geom_line() + theme(legend.position = "none")
g1 <- g1 + geom_hline(yintercept = theta_star[1], col = "red")
g1
```

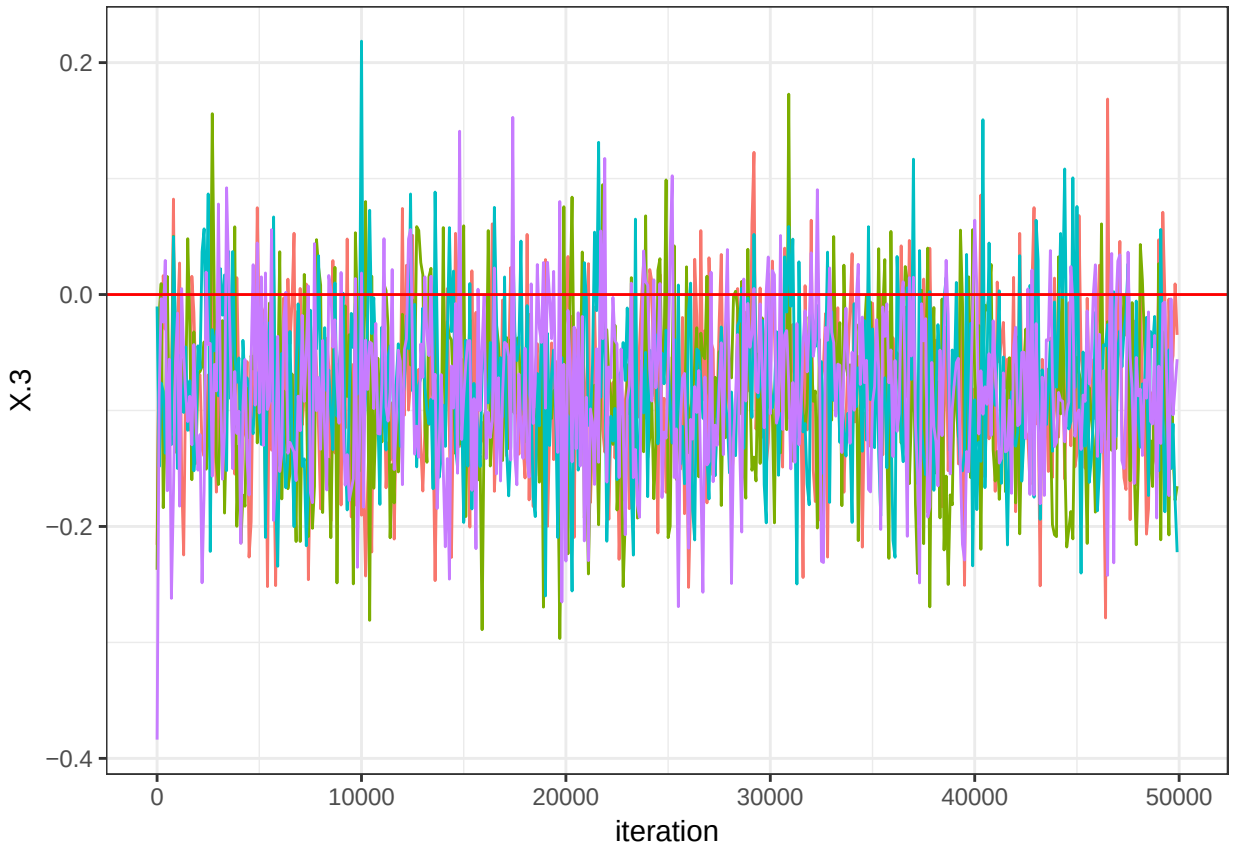



```
g2 <- ggplot(chain.df %>% filter(iteration > burnin, iteration%%100 == 1), aes(x = iteration,
```

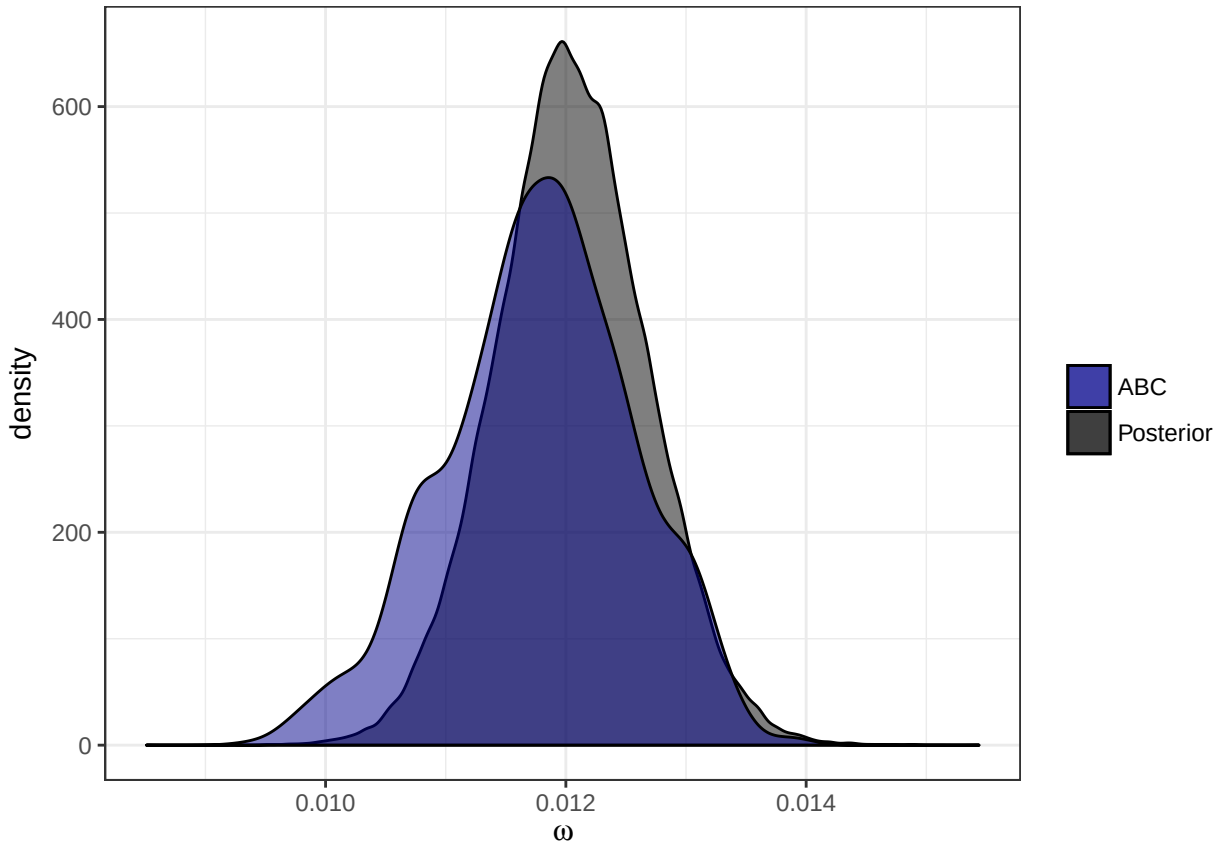
```
    y = X.2, group = ichain, colour = factor(ichain))) + geom_line() + theme(legend.position = "none")
g2 <- g2 + geom_hline(yintercept = theta_star[2], col = "red")
g2
```



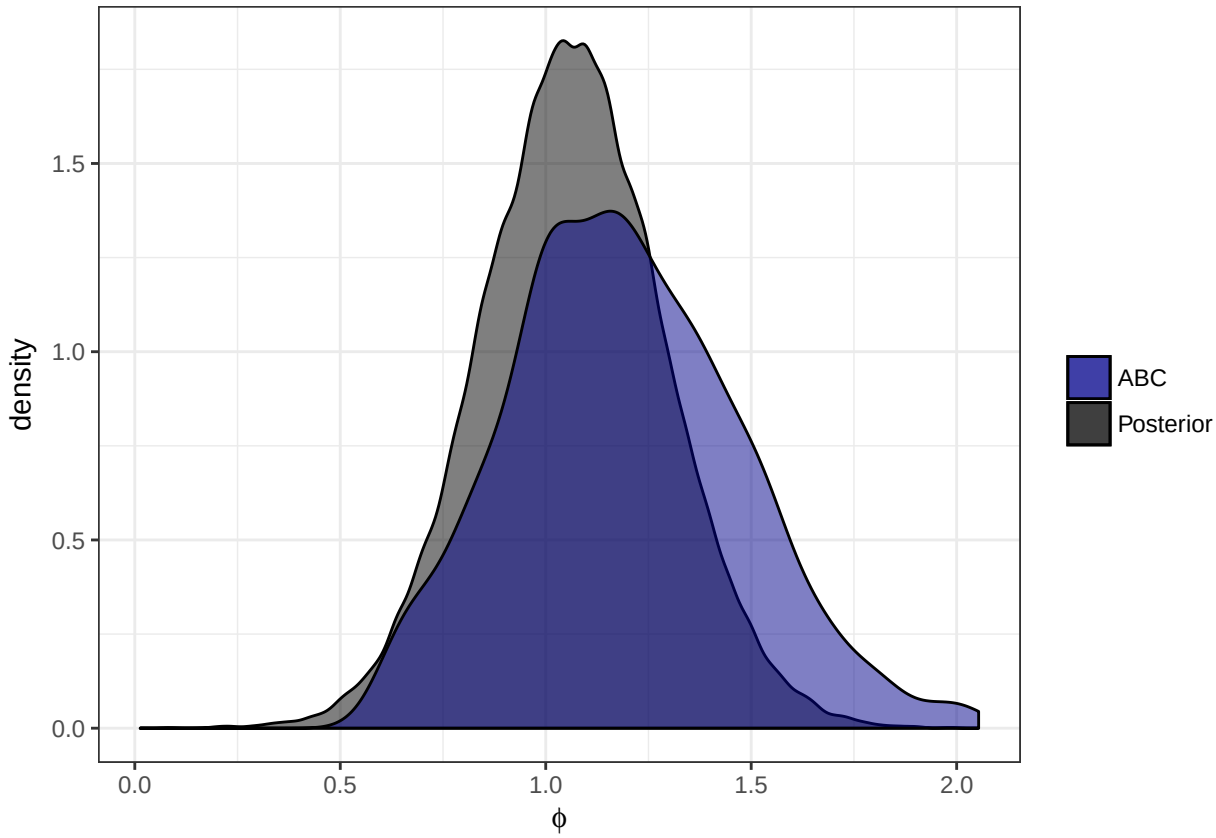

```
g3 <- ggplot(chain.df %>% filter(iteration > burnin, iteration%%100 == 1), aes(x = iteration,
    y = X.3, group = ichain, colour = factor(ichain))) + geom_line() + theme(legend.position = "none")
g3 <- g3 + geom_hline(yintercept = theta_star[3], col = "red")
g3
```



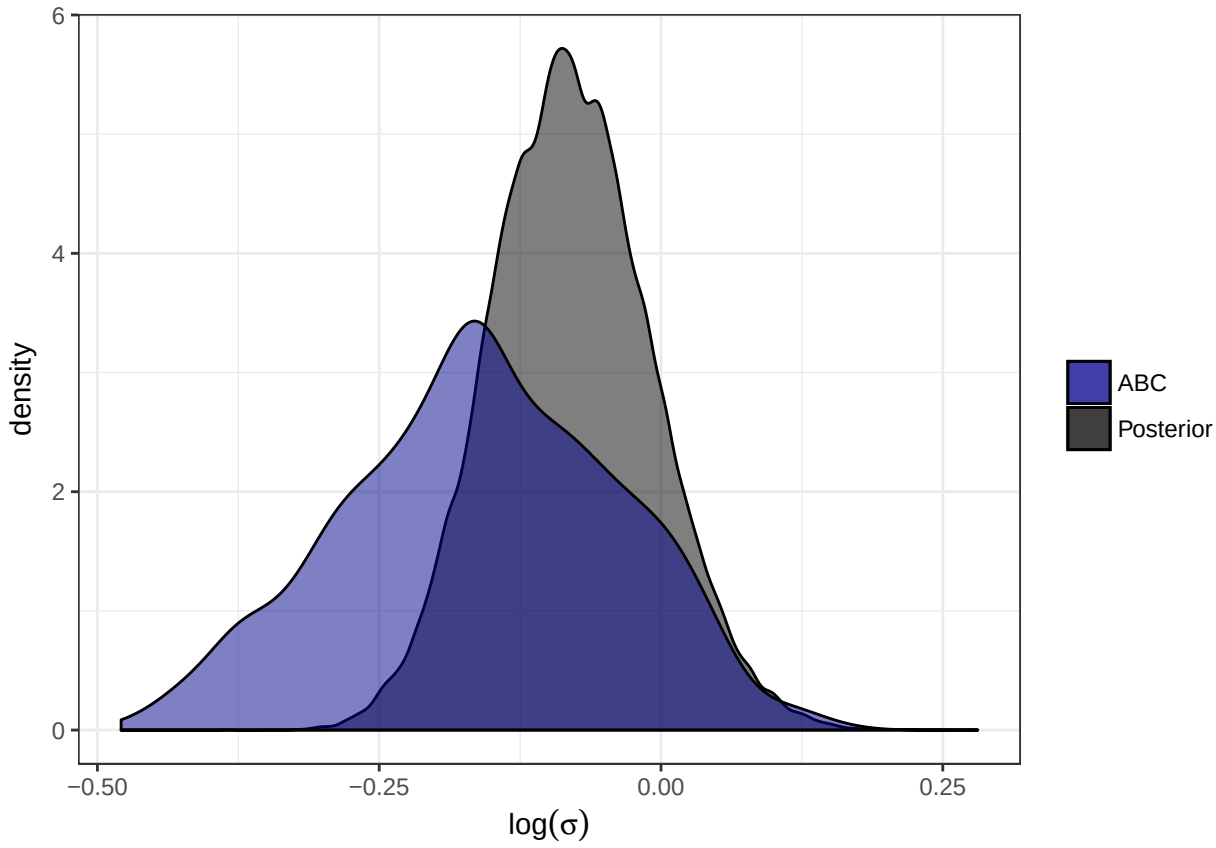

```
wsmc.df <- wsmc_to_dataframe(wsmcresults)
ggplot(chain.df, aes(x = X.1)) + geom_density(aes(y = ..density.., fill = "Posterior"),
    alpha = 0.5) + geom_density(data = wsmc.df %>% filter(step == length(wsmcresults$thetas_history)),
    aes(x = omega, y = ..density.., fill = "ABC"), alpha = 0.5) + scale_fill_manual(name = "",
    values = c(Posterior = "black", ABC = "darkblue")) + xlab(expression(omega))
```




```
ggplot(chain.df, aes(x = X.2)) + geom_density(aes(y = ..density.., fill = "Posterior"),
    alpha = 0.5) + geom_density(data = wsmc.df %>% filter(step == length(wsmcresults$thetas_history)),
    aes(x = phi, y = ..density.., fill = "ABC"), alpha = 0.5) + scale_fill_manual(name = "",
    values = c(Posterior = "black", ABC = "darkblue")) + xlab(expression(phi))
```




```
ggplot(chain.df, aes(x = X.3)) + geom_density(aes(y = ..density.., fill = "Posterior"),
    alpha = 0.5) + geom_density(data = wsmc.df %>% filter(step == length(wsmcresults$thetas_history)),
    aes(x = logsigma, y = ..density.., fill = "ABC"), alpha = 0.5) + scale_fill_manual(name = "",
    values = c(Posterior = "black", ABC = "darkblue")) + xlab(expression(log(sigma)))
```


6
4
2
0
density
−0.50
−0.25
0.00
0.25
log(σ)
ABC
Posterior