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 uniform on $[0, 0.1]$ for ω, uniform on $[0, 2\pi]$ for ϕ, and $\mathcal{N}(0, 1)$ for $\log(\sigma)$. The data are generated from $\omega_\star = 1/80$, $\phi_\star = \pi/4$ and $\sigma_\star = 1$.

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

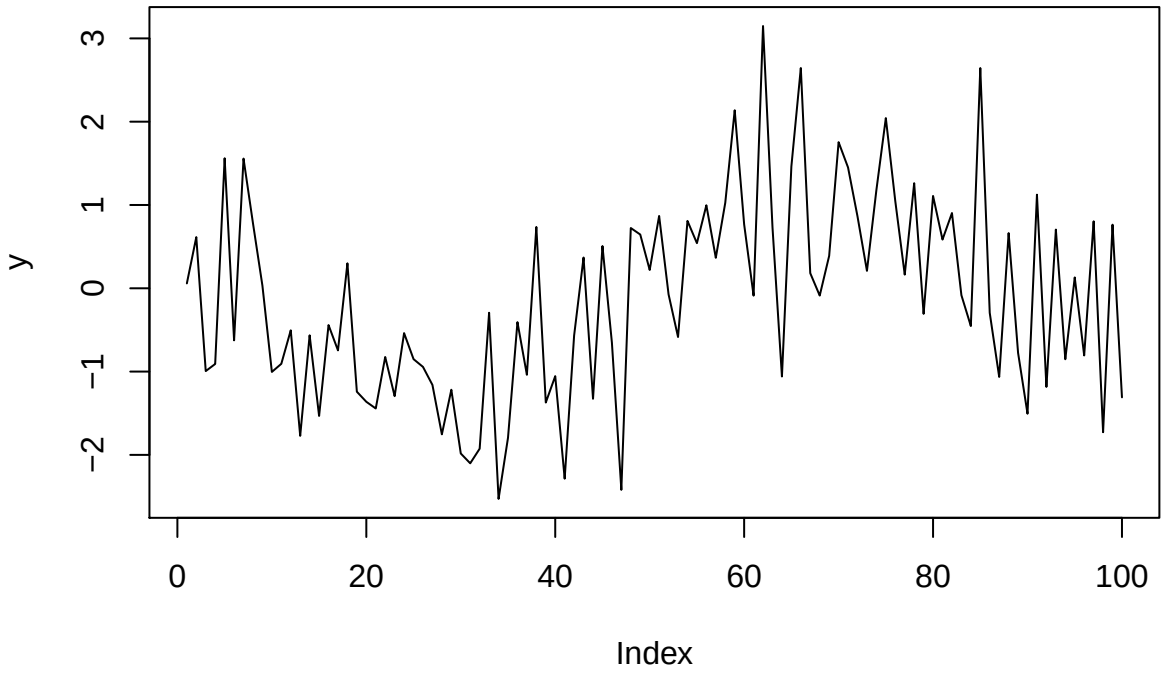
We define the model and generate some data from it.

```
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)
}
# model in a list
target <- list(rprior = rprior, dprior = dprior, simulate = simulate, parameter_names = c("omega",
    "phi", "logsigma"), thetadim = 3, ydim = 1, parameters = list())
# data-generating parameter
theta_star <- c(1/80, pi/4, 0)
# generate observations
obs <- target$simulate(theta_star)
# plot observations
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). We are also replacing the Wasserstein distance by the Swapping distance.

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

# use the swapping distance instead of the exact Wasserstein
compute_distance <- function(y_fake) {
    augmented_y_fake <- augment(y_fake)
    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: 1024 particles, one move per juvenation step, a mixture of

Gaussian as a proposal distribution, etc.

```
# 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, for a certain budget of model simulations. This might take a few minutes.

```
# now run the algorithm until 3e5 model simulations have been performed
wsmcresults <- wsmc(compute_distance, target, param_algo, maxsimulation = 3e5)
```

```
## step 1 completed, in 0.938 seconds, 1024 distances calculated
## step 2... running until # distances calculated >= 3e+05
##   acceptance rates: 63.47656 %, threshold = 0.6849645 , min. dist. = 0.426255
##   total # distances calculated: 4531 (for this step: 3507)
##   total time spent: 3.838 seconds (for this step: 2.9 seconds)
## step 3... running until # distances calculated >= 3e+05
##   acceptance rates: 56.73828 %, threshold = 0.6185983 , min. dist. = 0.4063413
##   total # distances calculated: 9401 (for this step: 4870)
##   total time spent: 6.747 seconds (for this step: 2.909 seconds)
## step 4... running until # distances calculated >= 3e+05
##   acceptance rates: 48.24219 %, threshold = 0.5894059 , min. dist. = 0.3994483
##   total # distances calculated: 16603 (for this step: 7202)
##   total time spent: 11.253 seconds (for this step: 4.506 seconds)
## step 5... running until # distances calculated >= 3e+05
##   acceptance rates: 47.65625 %, threshold = 0.5695419 , min. dist. = 0.3799856
##   total # distances calculated: 27034 (for this step: 10431)
##   total time spent: 17.046 seconds (for this step: 5.793 seconds)
## step 6... running until # distances calculated >= 3e+05
##   acceptance rates: 45.60547 %, threshold = 0.5543208 , min. dist. = 0.3557828
##   total # distances calculated: 42012 (for this step: 14978)
##   total time spent: 25.142 seconds (for this step: 8.096 seconds)
## step 7... running until # distances calculated >= 3e+05
##   acceptance rates: 43.94531 %, threshold = 0.5415454 , min. dist. = 0.3557828
##   total # distances calculated: 63154 (for this step: 21142)
##   total time spent: 36.929 seconds (for this step: 11.787 seconds)
## step 8... running until # distances calculated >= 3e+05
##   acceptance rates: 34.96094 %, threshold = 0.5292628 , min. dist. = 0.3468094
##   total # distances calculated: 85868 (for this step: 22714)
##   total time spent: 49.458 seconds (for this step: 12.529 seconds)
## step 9... running until # distances calculated >= 3e+05
##   acceptance rates: 26.75781 %, threshold = 0.5192233 , min. dist. = 0.3468094
##   total # distances calculated: 105901 (for this step: 20033)
##   total time spent: 60.276 seconds (for this step: 10.818 seconds)
## step 10... running until # distances calculated >= 3e+05
##   acceptance rates: 27.24609 %, threshold = 0.5093771 , min. dist. = 0.3468094
##   total # distances calculated: 117214 (for this step: 11313)
##   total time spent: 66.799 seconds (for this step: 6.523 seconds)
## step 11... running until # distances calculated >= 3e+05
##   acceptance rates: 30.07812 %, threshold = 0.499519 , min. dist. = 0.3066876
##   total # distances calculated: 126289 (for this step: 9075)
##   total time spent: 71.94 seconds (for this step: 5.141 seconds)
```

```
## step 12... running until # distances calculated >= 3e+05
##   acceptance rates: 37.20703 %, threshold = 0.4889268 , min. dist. = 0.3242378
##   total # distances calculated: 133572 (for this step: 7283)
##   total time spent: 76.263 seconds (for this step: 4.323 seconds)
## step 13... running until # distances calculated >= 3e+05
##   acceptance rates: 48.04688 %, threshold = 0.4721686 , min. dist. = 0.3260388
##   total # distances calculated: 139621 (for this step: 6049)
##   total time spent: 79.957 seconds (for this step: 3.694 seconds)
## step 14... running until # distances calculated >= 3e+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: 51.75781 %, threshold = 0.4497214 , min. dist. = 0.3303123
##   total # distances calculated: 146215 (for this step: 6594)
##   total time spent: 84.062 seconds (for this step: 4.105 seconds)
## step 15... running until # distances calculated >= 3e+05
##   acceptance rates: 48.4375 %, threshold = 0.4298037 , min. dist. = 0.3182463
##   total # distances calculated: 153786 (for this step: 7571)
##   total time spent: 88.616 seconds (for this step: 4.554 seconds)
## step 16... running until # distances calculated >= 3e+05
##   acceptance rates: 46.67969 %, threshold = 0.4128622 , min. dist. = 0.3228464
##   total # distances calculated: 162909 (for this step: 9123)
##   total time spent: 94.246 seconds (for this step: 5.63 seconds)
## step 17... running until # distances calculated >= 3e+05
##   acceptance rates: 43.75 %, threshold = 0.3987623 , min. dist. = 0.3167819
##   total # distances calculated: 175703 (for this step: 12794)
##   total time spent: 101.695 seconds (for this step: 7.449 seconds)
## step 18... running until # distances calculated >= 3e+05
##   acceptance rates: 43.65234 %, threshold = 0.3894656 , min. dist. = 0.3139326
##   total # distances calculated: 192004 (for this step: 16301)
##   total time spent: 110.883 seconds (for this step: 9.188 seconds)
## step 19... running until # distances calculated >= 3e+05
##   acceptance rates: 40.52734 %, threshold = 0.3802066 , min. dist. = 0.3092048
##   total # distances calculated: 212787 (for this step: 20783)
##   total time spent: 121.846 seconds (for this step: 10.963 seconds)
## step 20... running until # distances calculated >= 3e+05
##   acceptance rates: 43.65234 %, threshold = 0.3720511 , min. dist. = 0.3103245
##   total # distances calculated: 242112 (for this step: 29325)
##   total time spent: 137.242 seconds (for this step: 15.396 seconds)
## step 21... running until # distances calculated >= 3e+05
##   acceptance rates: 40.23438 %, threshold = 0.3643131 , min. dist. = 0.2980217
##   total # distances calculated: 284204 (for this step: 42092)
##   total time spent: 159.145 seconds (for this step: 21.903 seconds)
## step 22... running until # distances calculated >= 3e+05
```

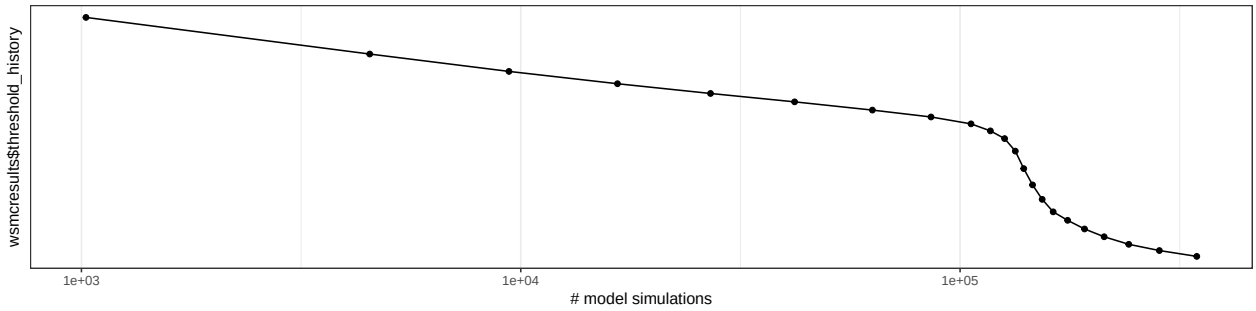
```
## Warning in mixmodCluster(data = data.frame(thetas_check), nbCluster =
## nclust, : All models got errors!
```

```
## [1] "error in fitting MCMC proposal, trying Rmixmod 5 times.."
##   acceptance rates: 43.75 %, threshold = 0.3580466 , min. dist. = 0.2901153
##   total # distances calculated: 345779 (for this step: 61575)
##   total time spent: 191.338 seconds (for this step: 32.193 seconds)
```

Now we can look at the ouput, with various plots, for instance the thresholds against the number of model

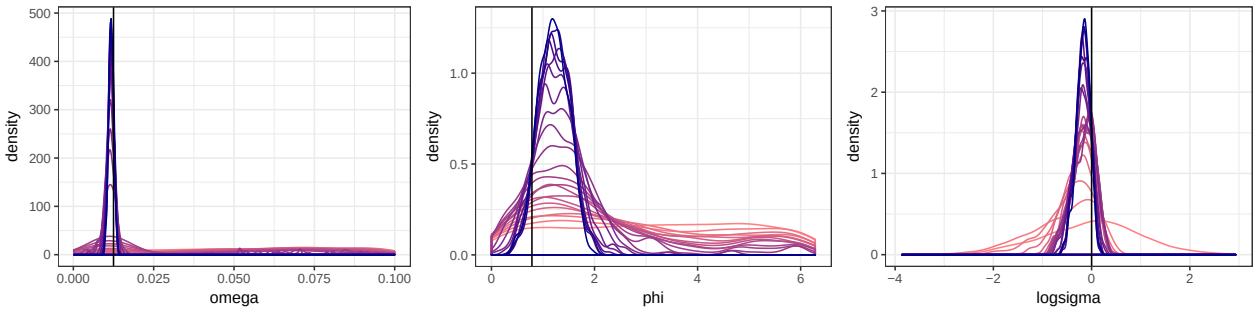
simulations.

```
# names(wsmcresults)
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 marginal distributions of parameters, with the data-generating parameters indicated by vertical lines.

```
# 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)
```



We see that the marginals start concentrating around the data-generating parameters, as expected from the theory.

Posterior samples via MCMC

We use the WABC output to initialize a Metropolis-Hastings algorithm, targeting the actual posterior distribution.

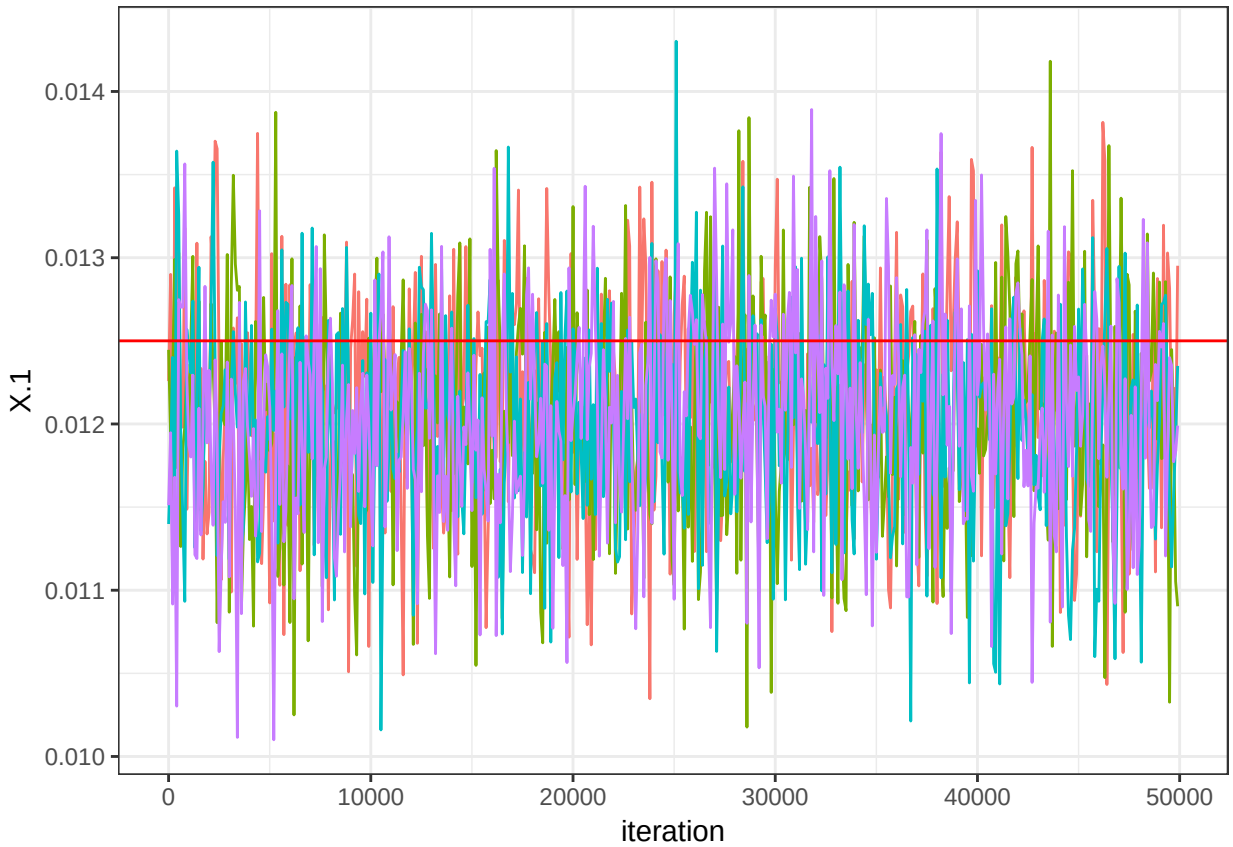
```
# define log-likelihood function
target$loglikelihood <- function(thetas, ys, ...) {
    evals <- rep(0, nrow(thetas))
    for (itheta in 1:nrow(thetas)) {
        backbone <- cos(2 * pi * thetas[itheta, 1] * (1:nobservations) + thetas[itheta,
            2])
        evals[itheta] <- sum(dnorm(ys, 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 = 10000, init_chains = theta_init)
# run adaptive MH scheme
mh <- metropolishastings(obs, target, tuning_parameters)
```

```
## average acceptance: 57.3955 %
```

```
burnin <- 0
chain.df <- mhchainlist_to_dataframe(mh$chains)

# trace plot, to check MCMC convergence
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
```

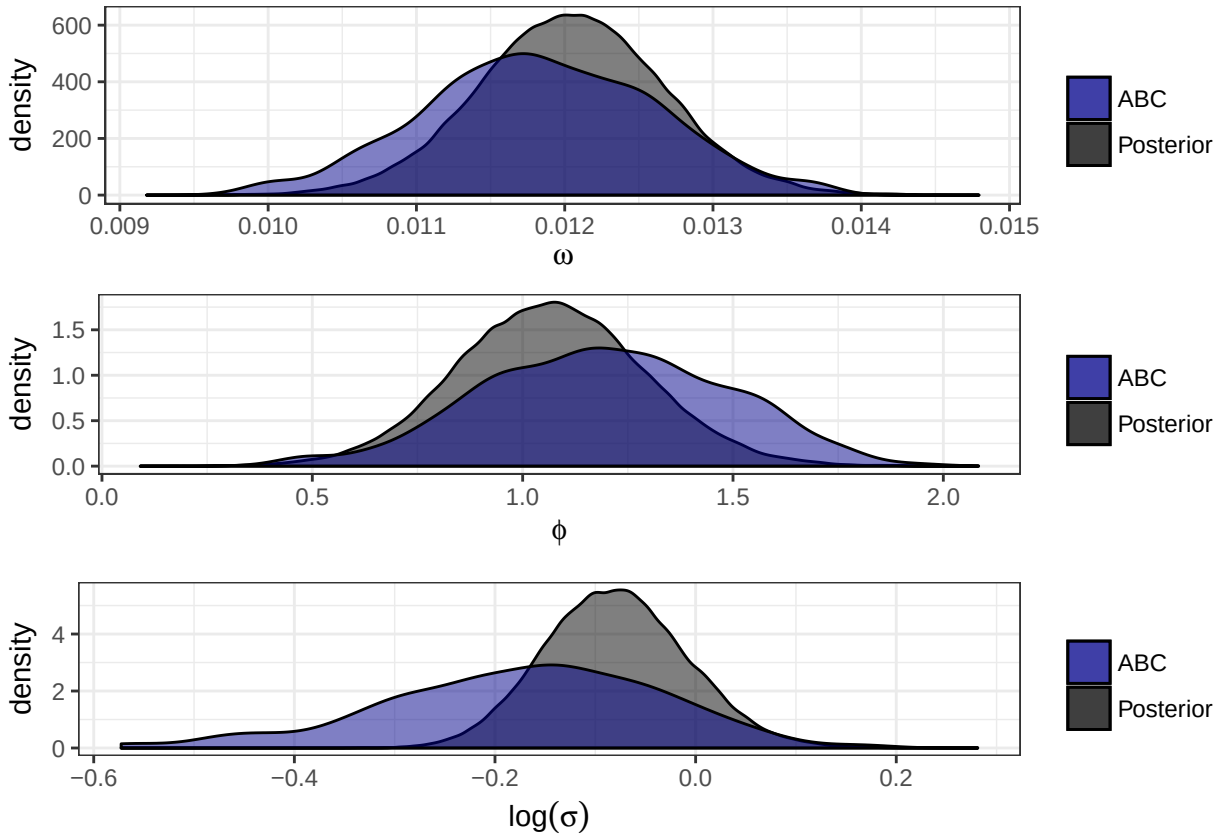



```
# let's look at the marginal distributions
wsmc.df <- wsmc_to_dataframe(wsmcresults)
g1 <- ggplot(chain.df, aes(x = X.1)) + geom_density(aes(y = ..density.., fill = "Posterior"),
    alpha = 0.5) + geom_density(data = wsmc.df %>% filter(step == length(wsmcresults$thetas_history)),
    aes(x = omega, y = ..density.., fill = "ABC"), alpha = 0.5) + scale_fill_manual(name = "",
    values = c(Posterior = "black", ABC = "darkblue")) + xlab(expression(omega))
```

```
g2 <- 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))

g3 <- 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)))

grid.arrange(g1, g2, g3, ncol = 1)
```



We see that the WABC posterior matches the actual posterior. It would be a closer match if we had run it for more steps.