Tutorial: quantile g-and-k

EB, PJ, MG, CR

June 10, 2017

```
# 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)
```

Setting

This script illustrates the calculation of the likelihood for the quantile g-and-k distribution. Then it runs a Metropolis algorithm to approximate the posterior distribution, and finally the proposed Wasserstein ABC approach. The g-and-k distribution is defined through its quantile function (the inverse of the cumulative distribution function or cdf):

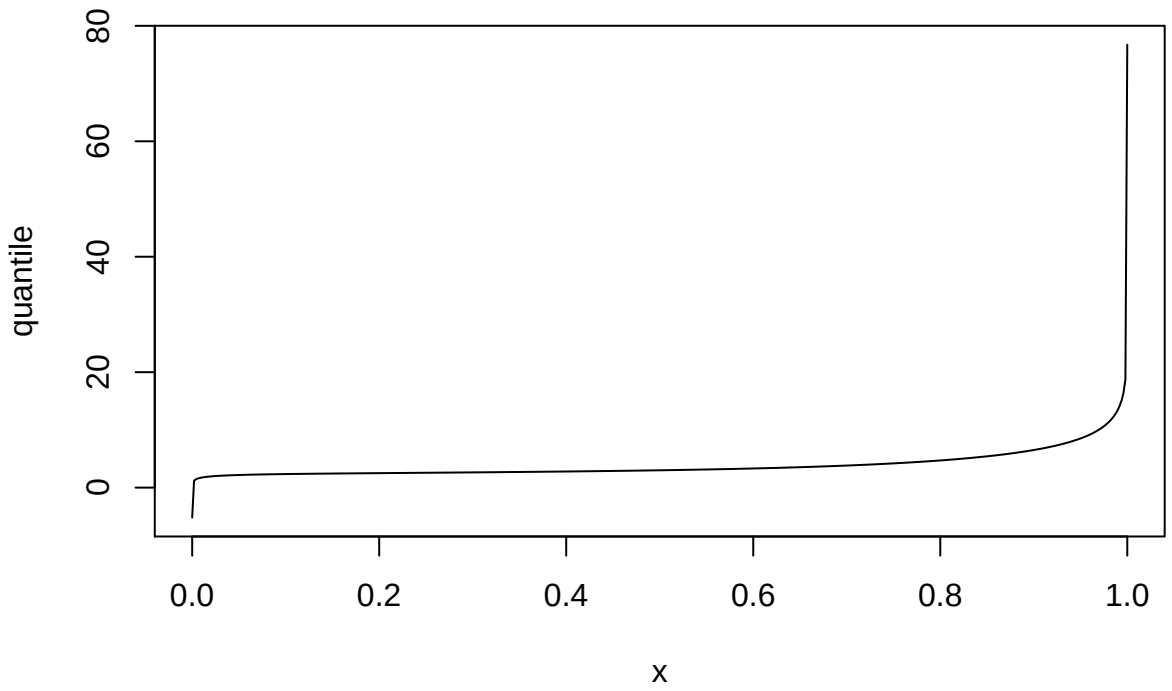
$$r \in (0,1) \mapsto a + b\left(1 + 0.8\frac{1-\exp(-gz(r))}{1+\exp(-gz(r))}\right)\left(1+z(r)^2\right)^k z(r),$$

with parameters (a, b, g, k). We define the data-generating parameter $(a_\star, b_\star, g_\star, k_\star) = (3, 1, 2, 0.5)$.

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector gandk_quantile(NumericVector x, NumericVector theta){
  NumericVector z = qnorm(x);
  double A = theta(0);
  double B = theta(1);
  double c = 0.8;
  double g = theta(2);
  double k = theta(3);
  return A + B * (1 + c * (1 - exp(- g * z)) / (1 + exp(- g * z))) * pow((1 + pow(z, 2.0)), k) * z;
}
```

We can plot the quantile function associated with that parameter.

```
# parameter of data-generating process
theta_star <- c(3, 1, 2, 0.5)
# define associated quantile function
quantile_star <- function(r) {
    gandk_quantile(r, theta_star)
}
# plot function
curve(sapply(x, FUN = function(v) quantile_star(v)), from = 1e-10, to = 1 -
    1e-10, n = 500, ylab = "quantile")
```



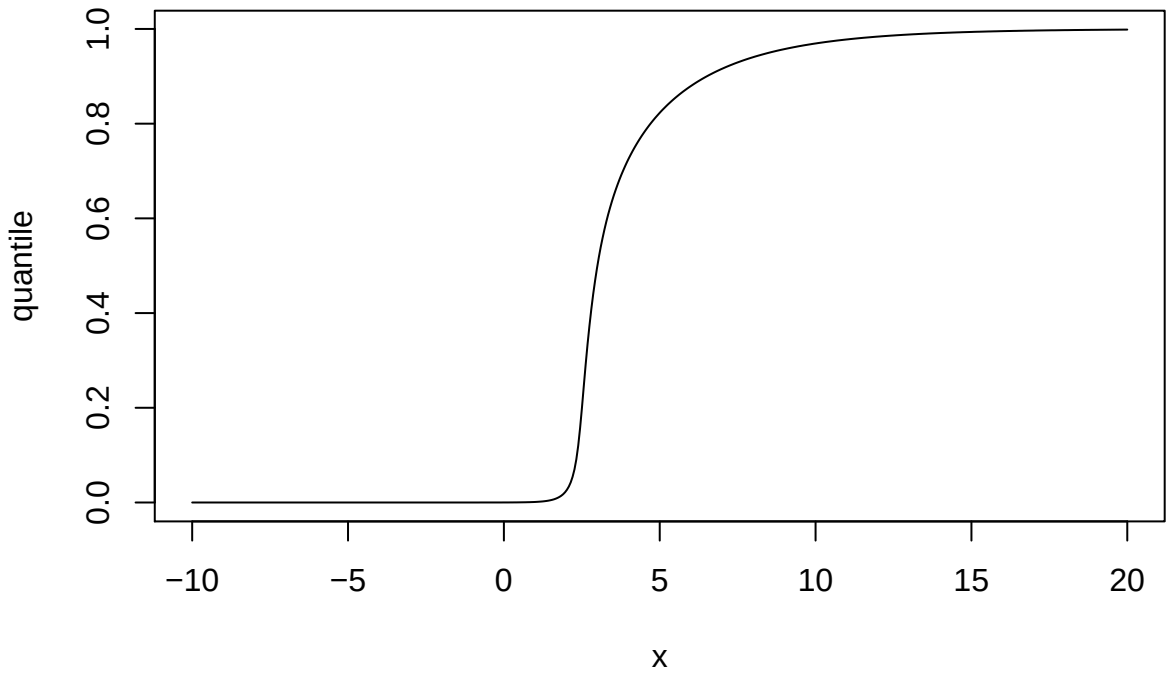
To compute the likelihood, we need to get the probability density function (or pdf) of the g-and-k distribution. The pdf is the derivative of the cdf, which is the inverse of the quantile function. We first get the cdf, by numerical inversion of the quantile function. We use a simple binary search scheme.

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
double gandk_cdf(double y, NumericVector theta, int maxsteps = 1000,
                 double tolerance = 1e-10, double lower = 1e-20, double upper = 1-1e-20){
  double A = theta(0);
  double B = theta(1);
  double c = 0.8;
  double g = theta(2);
  double k = theta(3);

  int istep = 0;
  double current_try = (upper + lower) / 2;
  double current_size = (upper - lower) / 4;
  NumericVector dd(1);
  dd(0) = current_try;
  NumericVector z = qnorm(dd);
  double fattempt = A + B * (1 + c * (1 - exp(- g * z(0))) / (1 + exp(- g * z(0)))) *
    pow((1 + pow(z(0), 2.0)), k) * z(0);
  while (!(fattempt > y-tolerance && fattempt < y+tolerance) && (istep < maxsteps)){
    istep++;
    if (fattempt > y-tolerance){
      current_try = current_try - current_size;
      dd(0) = current_try;
      NumericVector z = qnorm(dd);
      fattempt = A + B * (1 + c * (1 - exp(- g * z(0))) / (1 + exp(- g * z(0)))) *
        pow((1 + pow(z(0), 2.0)), k) * z(0);
      current_size = current_size / 2;
```

```
    } else {
      current_try = current_try + current_size;
      dd(0) = current_try;
      NumericVector z = qnorm(dd);
      fattempt = A + B * (1 + c * (1 - exp(- g * z(0))) / (1 + exp(- g * z(0)))) *
        pow((1 + pow(z(0), 2.0)), k) * z(0);
      current_size = current_size / 2;
    }
  }
  return current_try;
}
```

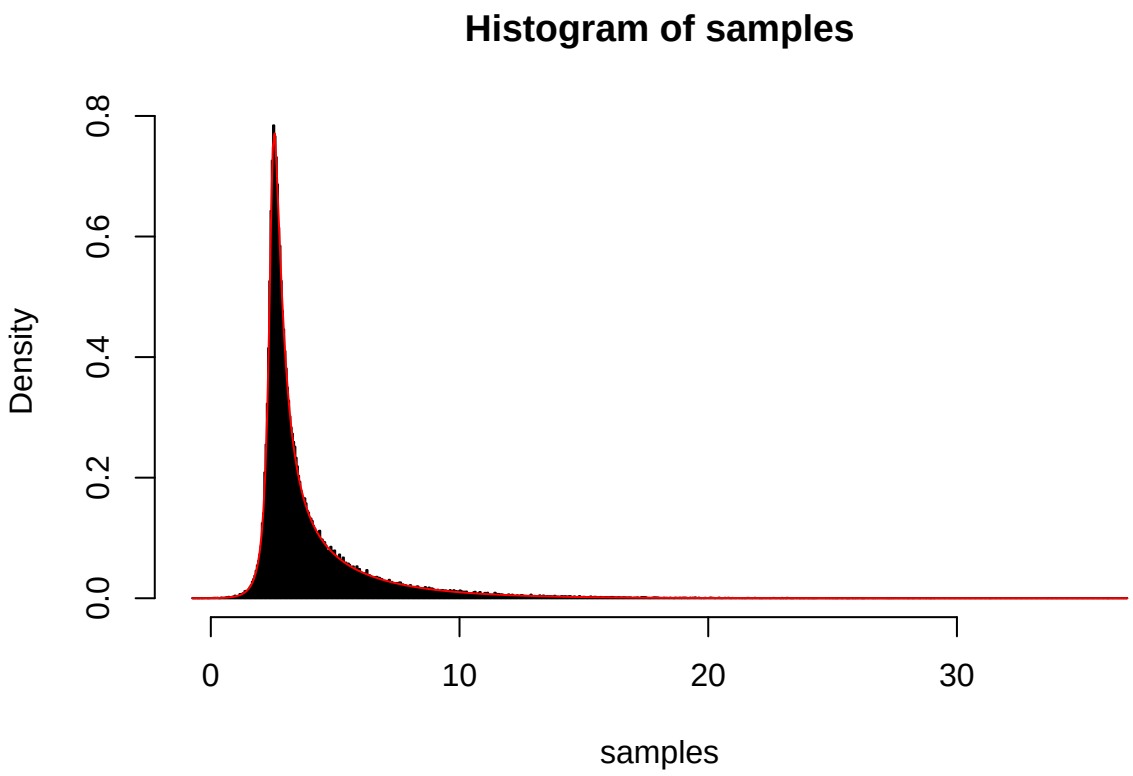
We can now plot the cdf of this distribution.

```
# plot cdf
curve(sapply(x, FUN = function(v) gandk_cdf(v, theta_star)), from = -10, to = 20,
    n = 500, ylab = "quantile")
```



Finally we can get the pdf by numerical differentiation, e.g. using the *numDeriv* package.

```
library(numDeriv)
cdf_ <- function(z) sapply(z, FUN = function(v) gandk_cdf(v, theta_star))
pdf <- function(x) {
    return(grad(cdf_, x))
}
# test: sample from the distribution
samples <- gandk_quantile(runif(1e+05), theta_star)
# plot the histogram and overlay pdf
hist(samples, prob = TRUE, nclass = 1000)
curve(pdf(x), add = TRUE, col = "red", n = 500)
```



Once we get the pdf, we can simply compute the likelihood by multiplying the pdf evaluated at each observation. However, we can be more efficient by sorting the observations, and inverting the quantile function for each observation in increasing order. Indeed, we can then successively reduce the range of the binary search, using the fact that the quantile function is an increasing function. Such an implementation of the g-and-k likelihood is provided below.

```
# this function computes the likelihood of ys, given each theta each theta
# being a row in the matrix 'thetas'
loglikelihood <- function(thetas, ys, ...) {
    n <- length(ys)
    evals <- rep(0, nrow(thetas))
    for (itheta in 1:nrow(thetas)) {
        ll <- function(ys, h = 1e-05, tolerance = 1e-10) {
            all_ys <- c(ys - h, ys + h)  # for finite difference differentiation
            o <- order(all_ys)
            x <- rep(0, length(all_ys))
            x[o[1]] <- gandk_cdf(y = all_ys[o[1]], theta = thetas[itheta, ],
                tolerance = tolerance)
            for (i in 2:length(all_ys)) {
                x[o[i]] <- gandk_cdf(y = all_ys[o[i]], theta = thetas[itheta,
                  ], tolerance = tolerance, lower = x[o[i - 1]])
            }
            return(sum(log((x[(n + 1):(2 * n)] - x[1:n])/(2 * h))))
        }
        evals[itheta] <- ll(ys)
    }
    return(evals)
}
```

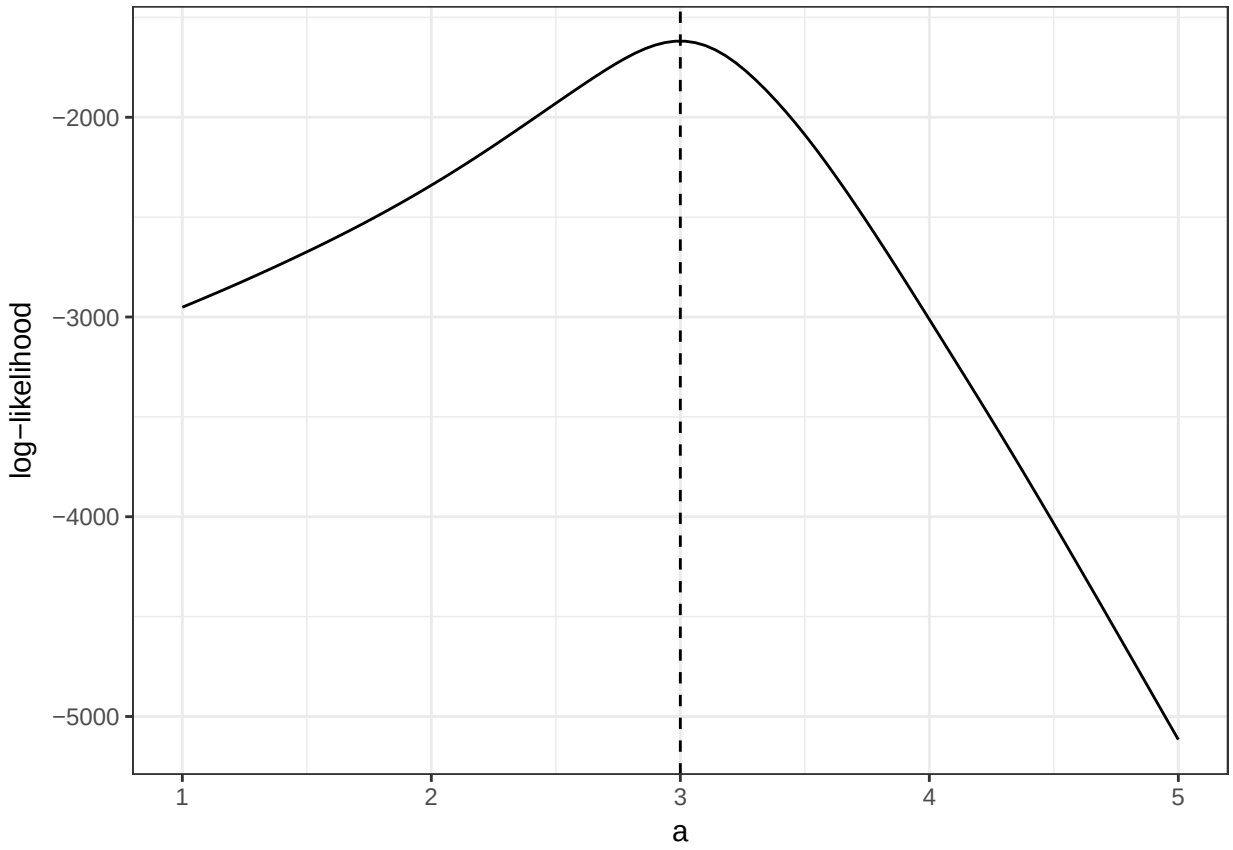
We now consider parameter inference for the g-and-k distribution. The code below defines a model, and

generates some data.

```
# number of observations
nobservations <- 1000
# prior is uniform [0,10] on each parameter
rprior <- function(N, parameters) {
    return(matrix(runif(N * 4, min = 0, max = 10), ncol = 4))
}
# evaluate the log-density of the prior, for each particle
dprior <- function(thetaparticles, parameters) {
    densities <- rep(0, nrow(thetaparticles))
    for (i in 1:nrow(thetaparticles)) {
        if (any(thetaparticles[i, ] > 10) || any(thetaparticles[i, ] < 0)) {
            densities[i] <- -Inf
        }
    }
    return(densities)
}
# function to generate a dataset given a parameter
simulate <- function(theta) {
    observations <- gandk_quantile(runif(nobservations), theta)
    return(observations)
}

target <- list(rprior = rprior, dprior = dprior, simulate = simulate, loglikelihood = loglikelihood,
    parameter_names = c("a", "b", "g", "k"), parameters = list(), thetadim = 4,
    ydim = 1)

obs <- target$simulate(theta_star)
```

We can now plot the likelihood of the observations, e.g. as a function of a, the other parameters being fixed.

```
nthetas <- 100
thetas <- matrix(nrow = nthetas, ncol = 4)
as <- seq(from = 1, to = 5, length.out = nthetas)
lls <- rep(0, nthetas)
for (i in 1:nthetas) {
    thetas[i, ] <- theta_star
    thetas[i, 1] <- as[i]
}
lls <- loglikelihood(thetas, obs)

qplot(x = as, y = lls, geom = "line") + geom_vline(xintercept = theta_star[1],
    linetype = 2) + xlab("a") + ylab("log-likelihood")
```

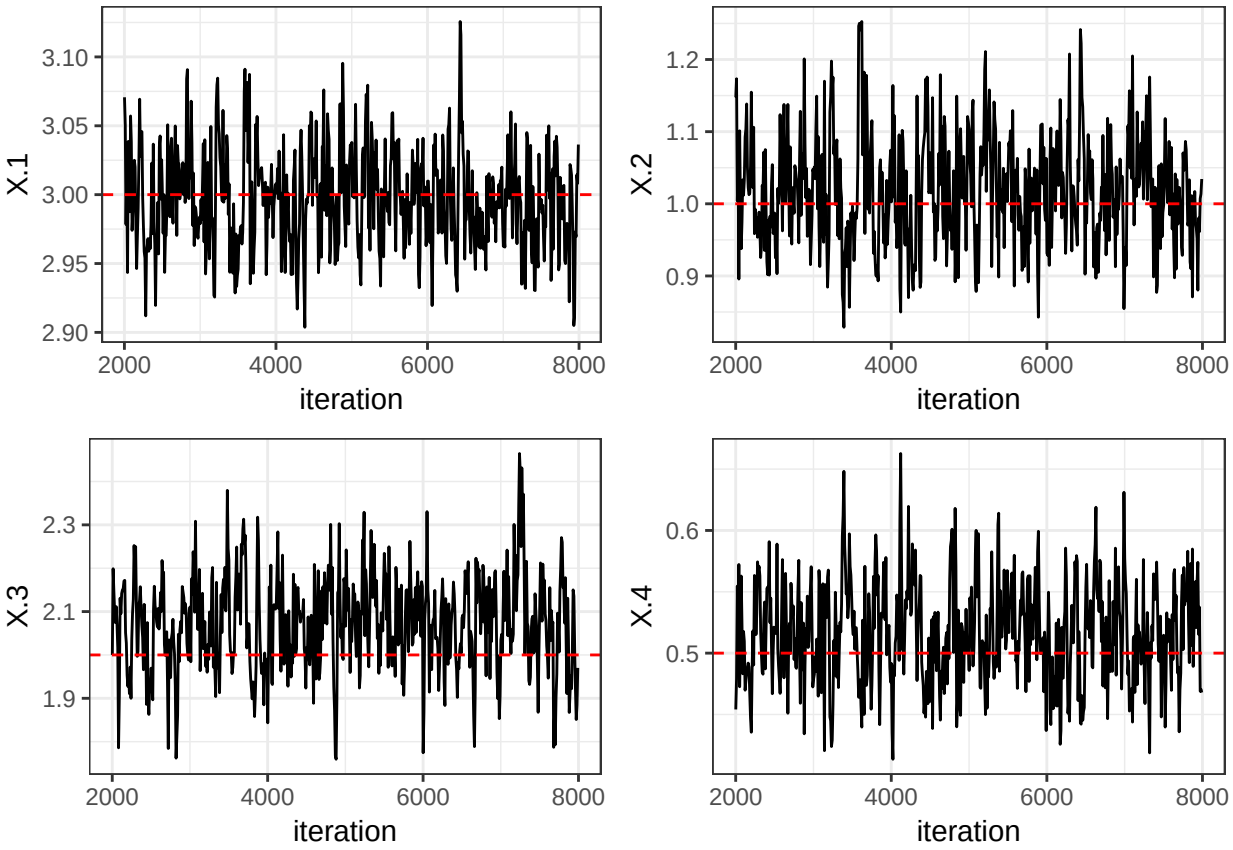


The log-likelihood plot is a good indication that parameter inference is going to be possible. We can perform e.g. MCMC, as follows. This might take a few minutes to run.

```
tuning_parameters <- list(niterations = 8000, nchains = 1, cov_proposal = diag(0.01,
    4, 4), adaptation = 2000, init_chains = matrix(theta_star, nrow = 1))
mh <- metropolishastings(obs, target, tuning_parameters)
```

```
## average acceptance: 48.6125 %
```

```
chain.df <- mhchainlist_to_dataframe(mh$chains)
burnin <- 2000
#
g1 <- ggplot(chain.df %>% filter(iteration > burnin, iteration%%10 == 1), aes(x = iteration,
    y = X.1, group = ichain)) + geom_line() + geom_hline(yintercept = theta_star[1],
    col = "red", linetype = 2)
g2 <- ggplot(chain.df %>% filter(iteration > burnin, iteration%%10 == 1), aes(x = iteration,
    y = X.2, group = ichain)) + geom_line() + geom_hline(yintercept = theta_star[2],
    col = "red", linetype = 2)
g3 <- ggplot(chain.df %>% filter(iteration > burnin, iteration%%10 == 1), aes(x = iteration,
    y = X.3, group = ichain)) + geom_line() + geom_hline(yintercept = theta_star[3],
    col = "red", linetype = 2)
g4 <- ggplot(chain.df %>% filter(iteration > burnin, iteration%%10 == 1), aes(x = iteration,
    y = X.4, group = ichain)) + geom_line() + geom_hline(yintercept = theta_star[4],
    col = "red", linetype = 2)
grid.arrange(g1, g2, g3, g4, ncol = 2)
```



Finally we can run the proposed WABC approach and compare it to the posterior obtained via MCMC. We run the method for 3 minutes, and plot the resulting marginal approximations of the posteriors.

```
# sort observations
obs_sorted <- sort(obs)
# function to compute distance between observed data and data generated
# given theta this corresponds to the 1-Wasserstein distance
compute_d <- function(y_fake, metric = metricL2) {
    y_fake_sorted <- sort(y_fake)
    return(mean((abs(obs_sorted - y_fake_sorted))))
}
# algorithmic parameters
param_algo <- list(nthetas = 1024, nmoves = 1, proposal = mixture_rmixmod(),
    minimum_diversity = 0.5, R = 2, maxtrials = 1e+05)
results <- wsmc(compute_d, target, param_algo, maxtime = 3 * 60)
```

Now, let's look at the marginal distributions obtained via MCMC ("Posterior") and via WABC.

```
wsmc.df <- wsmc_to_dataframe(results)
nsteps <- length(results$thetas_history)
names(chain.df) <- c("ichain", "iteration", target$parameter_names)
plot_marginal <- function(index) {
    g <- ggplot(chain.df %>% filter(iteration > burnin), aes_string(x = target$parameter_names[index]))
    g <- g + geom_density(aes(y = ..density.., fill = "Posterior"), alpha = 0.5)
    g <- g + geom_density(data = wsmc.df %>% filter(step == nsteps), aes(y = ..density..,
        fill = "WABC"), alpha = 0.5)
    g <- g + scale_fill_manual(name = "", values = c(Posterior = "black", WABC = "darkblue"))
    g <- g + xlab(target$parameter_names[index]) + geom_vline(xintercept = theta_star[index])
```

```
    return(g)
}
grid.arrange(plot_marginal(1), plot_marginal(2), plot_marginal(3), plot_marginal(4),
    ncol = 2)
```