

Tutorial: Normal location model

EB, PJ, MG, CR

March 21, 2017

Setting

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

The model specifies $Y \sim \mathcal{N}(\mu, \sigma^2)$, the parameters are (μ, σ) , the prior is standard Normal on μ , and Gamma(2, 1) on σ . The data are generated from a Gamma(10, 5) distribution, with $n = 1000$ observations.

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 generate some data.

```
# number of observations
nobservations <- 1000
# observations from a Gamma model, with mean 2
obs <- rgamma(nobservations, shape = 10, rate = 5)
```

We define a model.

```
# function to generate from prior distribution first argument is number of
# desired samples second argument contains hyper-parameters
rprior <- function(N, parameters) {
  particles <- matrix(nrow = N, ncol = 2)
  particles[, 1] <- rnorm(N, mean = parameters$mu_0, sd = 1/sqrt(parameters$nu))
  particles[, 2] <- rgamma(N, shape = parameters$alpha, rate = parameters$beta)
  return(particles)
}
# function to evaluate prior log-density first argument is a matrix of
# parameters (one per row) second argument contains hyper-parameters
dprior <- function(thetas, parameters) {
  logdensities <- dnorm(thetas[, 1], mean = parameters$mu_0, sd = 1/sqrt(parameters$nu),
    log = TRUE)
  logdensities <- logdensities + dgamma(thetas[, 2], shape = parameters$alpha,
    rate = parameters$beta, log = TRUE)
  return(logdensities)
}
```

```

}
# we specify a data-generating mechanism, given a parameter theta
simulate <- function(theta) {
  observations <- theta[1] + rnorm(nobservations) * theta[2]
  return(observations)
}
# we collect everything in a list; the 'parameters' list contains the
# hyper-parameters
target <- list(simulate = simulate, rprior = rprior, dprior = dprior, parameter_names = c("mu",
  "sigma"), parameters = list(mu_0 = 0, nu = 1, alpha = 2, beta = 1), thetadim = 2,
  ydim = 1)

```

Distance calculation and Monte Carlo algorithm

We define a way of calculating a distance between fake data and the observed data. Here we use the 1-Wasserstein distance, which is the distance between sorted samples.

```

y_obs_sorted <- sort(obs)
# function to compute 1-Wasserstein distance between observed data and fake
# data given as argument
compute_distance <- function(y_fake) {
  y_fake <- sort(y_fake)
  return(mean(abs(y_obs_sorted - y_fake)))
}

```

We now specify algorithmic parameters in a list.

```

# 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.

```

# now run the algorithm
wsmcresults <- wsmc(compute_distance, target, param_algo, maxtime = 10)

## step 1 completed, in 0.611 seconds, 1024 distances calculated
## step 2... running until time >= 10 seconds
##   acceptance rates: 79.98047 %, threshold = 2.333193 , min. dist. = 0.09226229
##   total # distances calculated: 4267 (for this step: 3243)
##   total time spent: 1.733 seconds (for this step: 1.122 seconds)
## step 3... running until time >= 10 seconds
##   acceptance rates: 78.41797 %, threshold = 1.684126 , min. dist. = 0.06826041
##   total # distances calculated: 7553 (for this step: 3286)
##   total time spent: 2.49 seconds (for this step: 0.757 seconds)
## step 4... running until time >= 10 seconds
##   acceptance rates: 77.63672 %, threshold = 1.275666 , min. dist. = 0.06422625
##   total # distances calculated: 10854 (for this step: 3301)
##   total time spent: 3.311 seconds (for this step: 0.821 seconds)
## step 5... running until time >= 10 seconds
##   acceptance rates: 78.02734 %, threshold = 0.9583558 , min. dist. = 0.04986393

```

```
## total # distances calculated: 14168 (for this step: 3314)
## total time spent: 4.007 seconds (for this step: 0.696 seconds)
## step 6... running until time >= 10 seconds
## acceptance rates: 76.5625 %, threshold = 0.6968365 , min. dist. = 0.04986393
## total # distances calculated: 17491 (for this step: 3323)
## total time spent: 4.848 seconds (for this step: 0.841 seconds)
## step 7... running until time >= 10 seconds
## acceptance rates: 75.09766 %, threshold = 0.4796462 , min. dist. = 0.05039149
## total # distances calculated: 20871 (for this step: 3380)
## total time spent: 5.562 seconds (for this step: 0.714 seconds)
## step 8... running until time >= 10 seconds
## acceptance rates: 75.09766 %, threshold = 0.3468625 , min. dist. = 0.05641825
## total # distances calculated: 24304 (for this step: 3433)
## total time spent: 6.608 seconds (for this step: 1.046 seconds)
## step 9... running until time >= 10 seconds
## acceptance rates: 71.875 %, threshold = 0.2451915 , min. dist. = 0.05192055
## total # distances calculated: 27770 (for this step: 3466)
## total time spent: 7.298 seconds (for this step: 0.69 seconds)
## step 10... running until time >= 10 seconds
## acceptance rates: 71.67969 %, threshold = 0.1817934 , min. dist. = 0.05093453
## total # distances calculated: 31301 (for this step: 3531)
## total time spent: 8.399 seconds (for this step: 1.101 seconds)
## step 11... running until time >= 10 seconds
## acceptance rates: 69.62891 %, threshold = 0.1364398 , min. dist. = 0.04472092
## total # distances calculated: 34955 (for this step: 3654)
## total time spent: 9.165 seconds (for this step: 0.766 seconds)
## step 12... running until time >= 10 seconds
## acceptance rates: 69.33594 %, threshold = 0.1055347 , min. dist. = 0.03240912
## total # distances calculated: 38868 (for this step: 3913)
## total time spent: 10.02 seconds (for this step: 0.855 seconds)
```

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

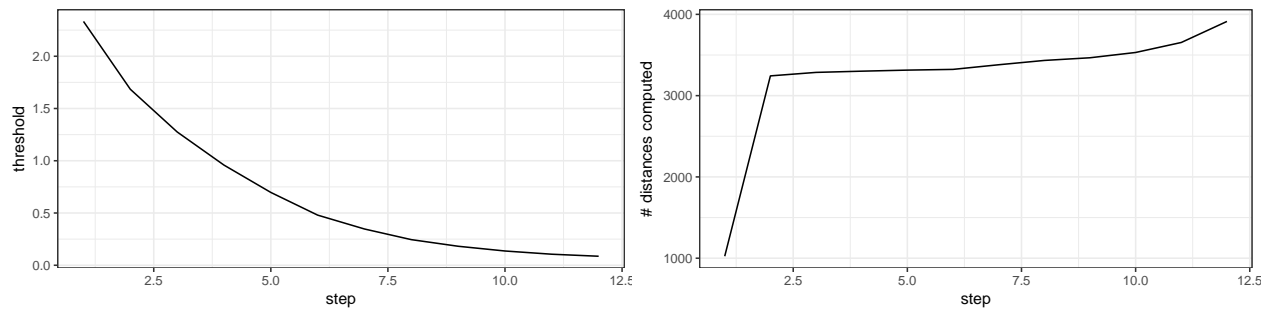
```
# we have access to all the generated particles, distances, and thresholds
names(wsmcresults)

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

# latest_y contains the latest generated data and distances_history stores
# all the calculated distances, so
tail(wsmcresults$distances_history, n = 1)[[1]][1] == compute_distance(wsmcresults$latest_y[[1]])

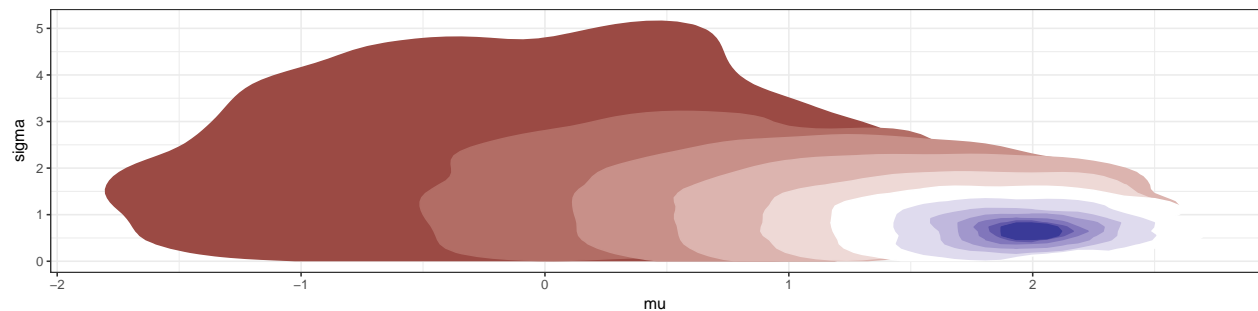
## [1] TRUE

# let's plot some of the output, for instance the evolution of the distance
# thresholds and the number of simulations per step
grid.arrange(plot_threshold(wsmcresults), plot_ncomputed(wsmcresults), ncol = 2)
```

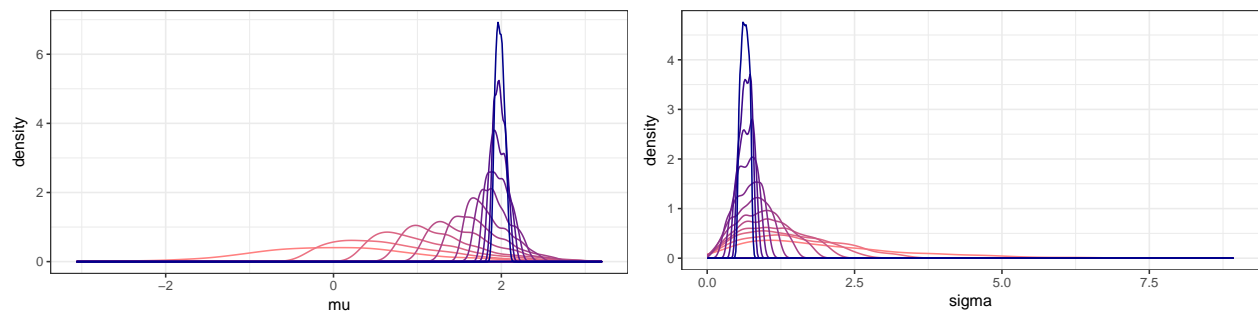


We can look at the distributions of parameters.

```
# and let's look at the parameters themselves
plot_bivariate_polygon(wsmcresults, i1 = 1, i2 = 2)
```



```
grid.arrange(plot_marginal(wsmcresults, i = 1), plot_marginal(wsmcresults, i = 2),
  ncol = 2)
```



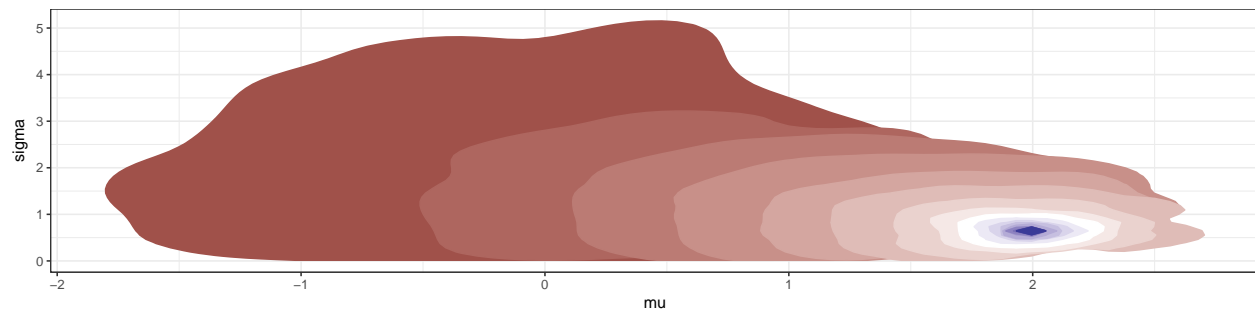
Finally, we can proceed to more steps of the algorithm, and plot the resulting output.

```
# let's do 10 more steps
wsmcresults_continued <- wsmc_continue(wsmcresults, maxtime = 10)

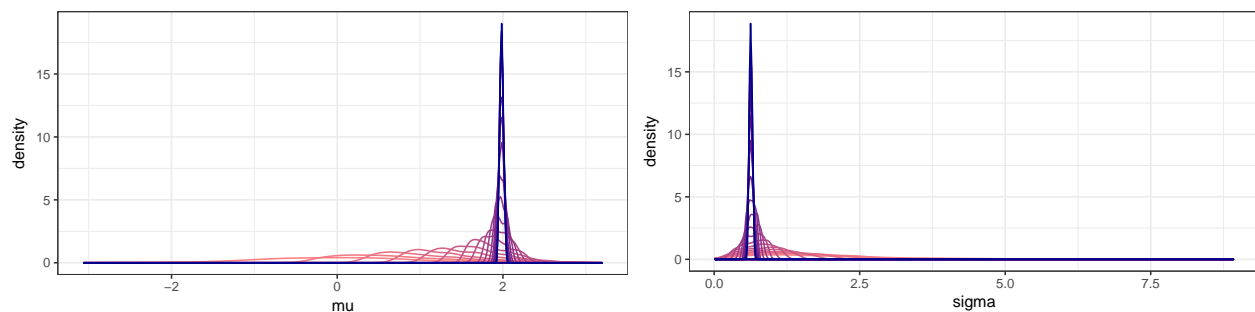
## result file contains the result of 12 steps
## step 13... running until time >= 10 seconds (since continue)
##   acceptance rates: 62.89062 %, threshold = 0.08615183 , min. dist. = 0.0343736
##   total # distances calculated: 43367 (for this step: 4499)
##   total time spent: 11.254 seconds (since continue: 1.234 s; this step: 1.234 s)
## step 14... running until time >= 10 seconds (since continue)
##   acceptance rates: 57.32422 %, threshold = 0.07398617 , min. dist. = 0.0343736
##   total # distances calculated: 48641 (for this step: 5274)
##   total time spent: 12.356 seconds (since continue: 2.336 s; this step: 1.102 s)
## step 15... running until time >= 10 seconds (since continue)
##   acceptance rates: 49.51172 %, threshold = 0.06637303 , min. dist. = 0.0343736
##   total # distances calculated: 55923 (for this step: 7282)
##   total time spent: 13.529 seconds (since continue: 3.509 s; this step: 1.173 s)
```

```
## step 16... running until time >= 10 seconds (since continue)
## 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: 49.60938 %, threshold = 0.0613975 , min. dist. = 0.0343736
## total # distances calculated: 66192 (for this step: 10269)
## total time spent: 15.094 seconds (since continue: 5.074 s; this step: 1.565 s)
## step 17... running until time >= 10 seconds (since continue)
## acceptance rates: 49.12109 %, threshold = 0.05790521 , min. dist. = 0.0303372
## total # distances calculated: 81956 (for this step: 15764)
## total time spent: 16.891 seconds (since continue: 6.871 s; this step: 1.797 s)
## step 18... running until time >= 10 seconds (since continue)
## acceptance rates: 46.28906 %, threshold = 0.05509206 , min. dist. = 0.03461449
## total # distances calculated: 106142 (for this step: 24186)
## total time spent: 19.338 seconds (since continue: 9.318 s; this step: 2.447 s)
## step 19... running until time >= 10 seconds (since continue)
## acceptance rates: 43.65234 %, threshold = 0.05260114 , min. dist. = 0.03353037
## total # distances calculated: 141711 (for this step: 35569)
## total time spent: 22.423 seconds (since continue: 12.403 s; this step: 3.085 s)
```

```
#
plot_bivariate_polygon(wsmcresults_continued, i1 = 1, i2 = 2)
```



```
grid.arrange(plot_marginal(wsmcresults_continued, i = 1), plot_marginal(wsmcresults_continued,
i = 2), ncol = 2)
```



The Wasserstein ABC posterior has concentrated around a specific region of the parameter space, around the parameter θ that minimizes the Wasserstein distance between the empirical distribution of the data and the model's distribution.