

ABC inference for Normal mean and variance

Paul Blischak

Regular ABC algorithm

```
normal_start_time <- proc.time()
# Read in the data and sort it
dat <- as.vector(read.table("data.txt",header=F))
dat <- sort(dat[,1])

# Set up number of simulations and number of parameters to save
reps <- 50000
save <- 100
dist_vals <- rep(NA,reps)

# Draw values from the prior distributions
tmp_mean <- runif(reps, -20, 20)
tmp_variance <- runif(reps, 0, 50)

# Loop through the prior values, simulate data and compare to observed data
for(i in 1:reps){
  tmp_dat <- rnorm(length(dat), tmp_mean[i], sqrt(tmp_variance[i]))
  tmp_dat <- sort(tmp_dat)
  dist_vals[i] <- dist(rbind(dat,tmp_dat))
}

# Sort distance values (summary stat)
dist_indexes <- sort(dist_vals, index.return=T)

# Get the top values based on the number of parameter values you want to save
save_indexes <- dist_indexes$ix[1:save]

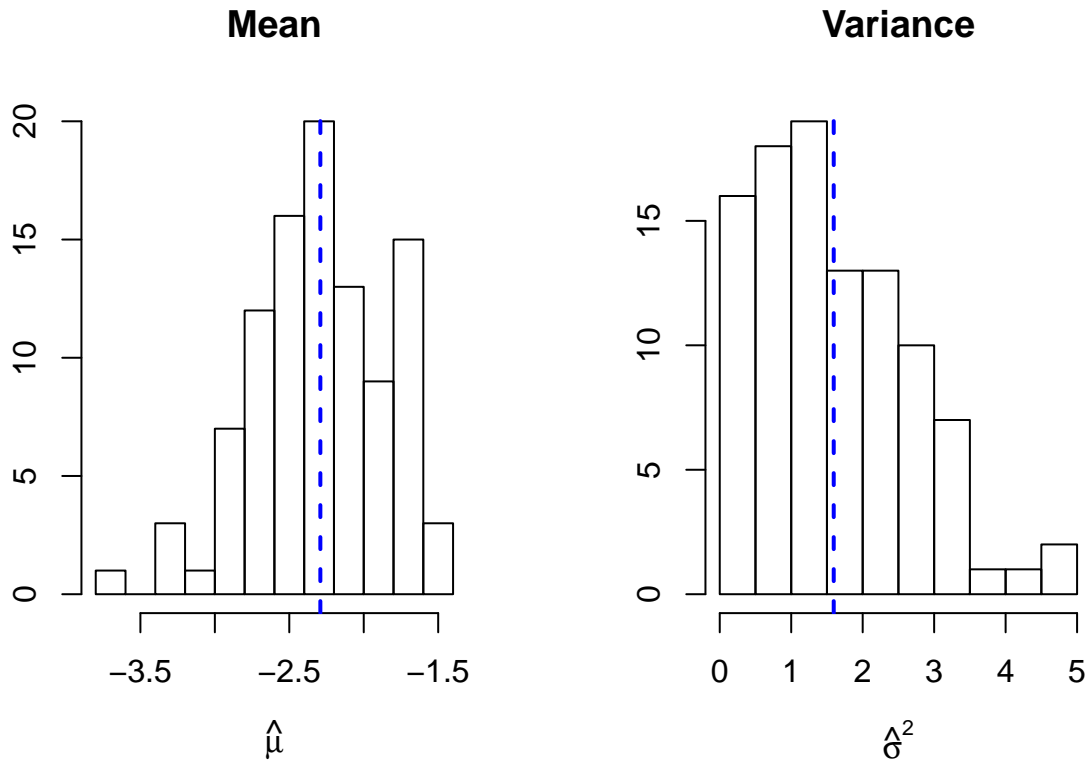
# Get the corresponding parameter values with the lowest distances from the observed data
saved_means <- tmp_mean[save_indexes]
saved_variances <- tmp_variance[save_indexes]
normal_total_time <- proc.time() - normal_start_time

# Store in a data frame and print as a knitr table
res <- data.frame(Parameter=c("Normal Mean","Normal Variance"),
                  Mean=c(mean(saved_means), mean(saved_variances)),
                  SD=c(sd(saved_means),sd(saved_variances)))

knitr::kable(res, digits=3)
```

Parameter	Mean	SD
Normal Mean	-2.291	0.443
Normal Variance	1.595	1.068

```
# Plot the posterior distributions for the mean and variance
par(mfrow=c(1,2))
hist(saved_means, main="Mean", xlab=expression(hat(mu)), ylab="")
abline(v=mean(saved_means),col="blue",lty="dashed",lwd=2)
hist(saved_variances, main="Variance", xlab=expression(hat(sigma)^2), ylab="")
abline(v=mean(saved_variances),col="blue",lty="dashed",lwd=2)
```



ABC–Sequential Monte Carlo algorithm

```
sequential_start_time <- proc.time()
# Read in data
dat <- as.vector(read.table("data.txt",header=F))
dat <- sort(dat[,1])

# Number of SMC rounds
rounds <- 5

# Number of iterations per round
N <- 10000

# Fraction of samples to save per round
tau <- c(0.2,rep(0.25,rounds-1))

# Number of samples to save from first round
Q <- N*tau[1]

# Threshold value for simulated vectors to be saved
```

```

epsilon <- rep(NA,rounds)

# Storage vector for dist measure from simulated data
dist_vals <- rep(NA,N)

# Draw initial mean and variance samples for first round from prior
tmp_mean <- runif(N, -20, 20)
tmp_variance <- runif(N, 0, 50)

## ----- 1st round ----- ##

# Draw N data sets and compare
for(i in 1:N){
  tmp_dat <- rnorm(length(dat), tmp_mean[i], sqrt(tmp_variance[i]))
  tmp_dat <- sort(tmp_dat)
  dist_vals[i] <- dist(rbind(dat,tmp_dat))
}

# Sort dist values and save the top Q means and variances
dist_indexes <- sort(dist_vals, index.return=T)
save_indexes <- dist_indexes$ix[1:Q]
epsilon[1] <- dist_indexes$x[Q] # first epsilon is the max dist value
saved_means <- tmp_mean[save_indexes]
saved_variances <- tmp_variance[save_indexes]

## ----- 2nd through rth rounds ----- ##

for(r in 2:rounds){

  curr_num_saved <- 0
  dist_vals <- rep(NA,Q)
  tmp_saved_means <- rep(NA,Q)
  tmp_saved_variances <- rep(NA,Q)

  while(curr_num_saved < Q){
    curr_mean <- sample(saved_means,1)
    curr_mean <- curr_mean + runif(1,-0.05,0.05)
    curr_variance <- sample(saved_variances,1)
    curr_variance <- curr_variance + runif(1,0,0.05)
    curr_dat <- rnorm(30,curr_mean,sqrt(curr_variance))
    curr_dat <- sort(curr_dat)
    curr_dist <- dist(rbind(dat,curr_dat))

    # Only save value if it is better than the previously saved max
    if(curr_dist < epsilon[r-1]){

      curr_num_saved <- curr_num_saved + 1
      dist_vals[curr_num_saved] <- curr_dist
      tmp_saved_means[curr_num_saved] <- curr_mean
      tmp_saved_variances[curr_num_saved] <- curr_variance
    }
  }
}

```

```

}

# Save mean and variance samples
dist_indexes <- sort(dist_vals, index.return=T)
save_indexes <- dist_indexes$ix[1:(tau[r]*Q)]
epsilon[r] <- dist_indexes$x[(tau[r]*Q)]
saved_means <- tmp_saved_means[save_indexes]
saved_variances <- tmp_saved_variances[save_indexes]
}

sequential_total_time <- proc.time() - sequential_start_time

# Make a data frame and print as a knitr table
res <- data.frame(Parameter=c("Normal Mean", "Normal Variance"),
                  Mean=c(mean(saved_means), mean(saved_variances)),
                  SD=c(sd(saved_means), sd(saved_variances)))

knitr::kable(res, digits=3)

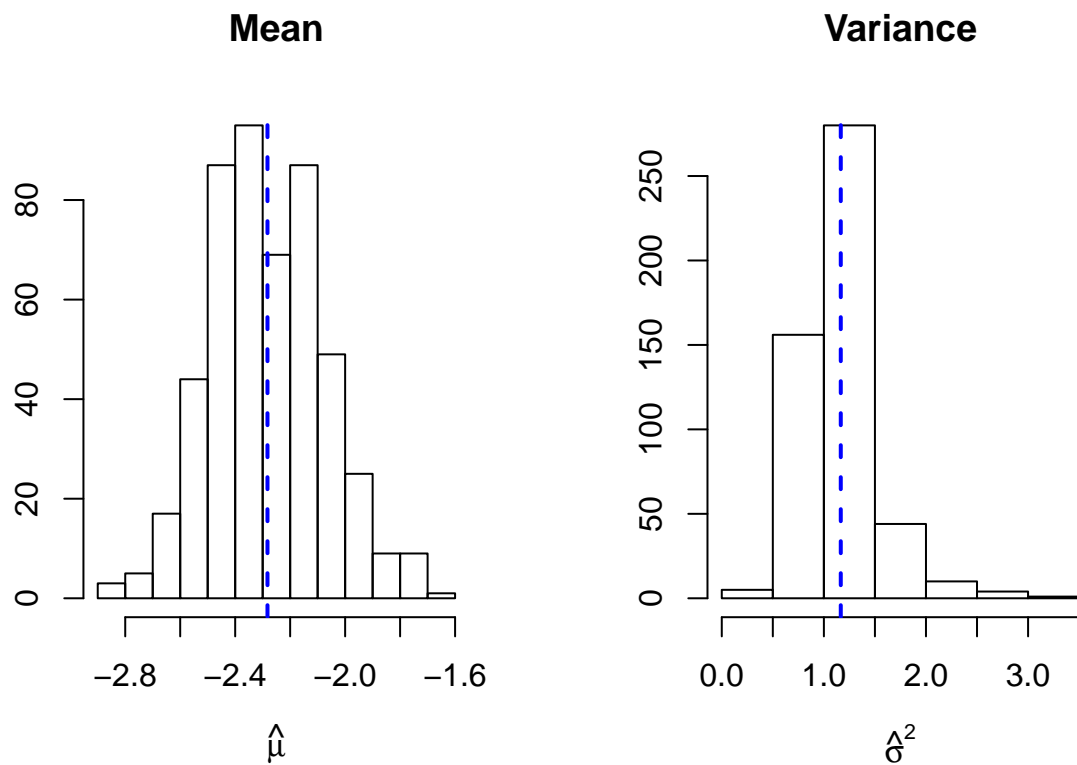
```

Parameter	Mean	SD
Normal Mean	-2.283	0.209
Normal Variance	1.165	0.362

```

# Plot the posterior distribution of the parameters
par(mfrow=c(1,2))
hist(saved_means, main="Mean", xlab=expression(hat(mu)), ylab="")
abline(v=mean(saved_means), col="blue", lty="dashed", lwd=2)
hist(saved_variances, main="Variance", xlab=expression(hat(sigma)^2), ylab="")
abline(v=mean(saved_variances), col="blue", lty="dashed", lwd=2)

```



```
timing <- data.frame(Algorithm=c("ABC","ABC-SMC"),
  Time=c(normal_total_time[3],sequential_total_time[3]),
  Improvement=c(normal_total_time[3]/normal_total_time[3],
    normal_total_time[3]/sequential_total_time[3]))

knitr::kable(timing, digits=3)
```

Algorithm	Time	Improvement
ABC	3.633	1.000
ABC-SMC	1.784	2.036