Example codes for “Backward importance sampling for online estimation of state space models”

Anonymous authors

Contents

1 About the codes

- *All the codes presented below are `R` codes that should work for a `R` version ≥ 3.4.*
- The proposed algorithm is mainly embedded in two `R` packages, one for the first section (that also contains code for the state of the art *Grand PaRIS* algorithm) and one for the second. To read the raw code of the proposed algorithm, the reader should navigate in the `src` directory of the packages after having downloaded them.

2 Comparison for the Sine Model

2.1 Installation of the required package

The required codes are packages into the `GrandParisPackage` package for the `R` software, available on github.

To install this package in `R`, simply run:

```
devtools::install_github("papayoun/GrandParisPackage")
```

2.2 Code for the comparison

Then, the full comparison is done in the following code:

```r
# Cleaning -----------------------------------------------------------------

rm(list = ls())

# Packages -----------------------------------------------------------------

library(GrandParisPackage)
library(parallel) # For parallel computing (do not work on windows)
library(tidyverse) # For data processing
library(tictoc) # For time comparisons

# Simulating data ----------------------------------------------------------

my_seed <- 333 # For all experiments, the random seed

set.seed(my_seed) # For reproducibility
trueTheta <- pi / 4; trueSigma2 <- 1;
n <- 11; times <- seq(from = 0, to = 5, length = n);
SINEprocess <- SINE_simulate(theta = trueTheta, sigma2 = trueSigma2,
                             x0 = 10, times = times)
observations <- SINEprocess[, "observations"]


# Experiment_function ------------------------------------------------------

# A function that computes the approximated estimate of E[X_0 | Y_{0:n}]
# for different tilde(N)
# It does it 400 times

compare_on_ntilde <- function(n_particle = 100, n_tilde = NULL){
  if(is.null(n_tilde)){
    stop("You must provide ntilde!")
  }
  n_rep <- 400
  AR <- do.call(rbind,
                mclapply(1:n_rep, function(i){
                  tic()
                  res <- GrandParisPackage:::E_track(observations = observations,
                                                     ind_tracked = 0,
                                                     observationTimes = times,
                                                     thetaStart = trueTheta,
                                                     particleSize = n_particle,
                                                     backwardSampleSize = n_tilde,
                                                     sigma2Start = trueSigma2,
                                                     nIterations = 1)
                  my_tictoc <- toc()
                  dur <- my_tictoc$toc - my_tictoc$tic
                  names(dur) <- NULL
                  data.frame(N = n_particle,
                             N_tilde = n_tilde, Xhat = res, Time = dur,
                             Method = factor("AR",
                                             levels = c("AR", "IS")))
                }, mc.cores = 10))
  IS <- do.call(rbind,
                mclapply(1:n_rep, function(i){
                  tic()
                  res <- GrandParisPackage:::E_track_IS(observations = observations,
                                                        ind_tracked = 0,
                                                        observationTimes = times,
                                                        thetaStart = trueTheta,
                                                        particleSize = n_particle,
                                                        backwardSampleSize = n_tilde,
                                                        sigma2Start = trueSigma2,
                                                        nIterations = 1)
                  my_tictoc <- toc()
                  dur <- my_tictoc$toc - my_tictoc$tic
                  names(dur) <- NULL
                  data.frame(N = n_particle,
                             N_tilde = n_tilde, Xhat = res, Time = dur,
                             Method = factor("IS",
                                             levels = c("AR", "IS")))
                }, mc.cores = 10))
  return(rbind.data.frame(AR, IS))
}


# Obtaining estimates -----------------------------------------------------

set.seed(my_seed) # For reproducibility
my_ntildes <- c(2, 5, 10, 20, 30)


# Run only once (experiment) ----------------------------------------------

res  <- lapply(my_ntildes, function(n_t)
  compare_on_ntilde(n_particle = 100, n_tilde = n_t))
res_df <- do.call(rbind.data.frame, res)

# Plotting results --------------------------------------------------------

main_plot <- ggplot(res_df, aes(x = factor(N_tilde), fill = Method)) +
  labs(x = expression(tilde(N))) +
  theme(text = element_text(family = "Symbola", size = 24))

p1 <- main_plot +
  geom_boxplot(aes(y = Time)) +
  scale_y_continuous(trans = "log10") +
  labs(y = "Comput. time (seconds)")
p2 <-  main_plot +
  geom_boxplot(aes(y = Xhat)) +
  labs(y = expression(hat("\U1d53c")~"["~X[0]~"|"~Y[0:n]~"]"))
my_plot <- gridExtra::grid.arrange(p1, p2)

# Controlling N -----------------------------------------------------------

compare_on_n <- function(n_particle = 100, # Number of particles
```

```r
                          alphas = NULL, # Power of N to obtain Ntilde
                          # 0.5 and 0.6 in the article
                          frac = NULL){ # Fraction of N to obtain Ntilde (0.1 in the article)
  n_tilde_AR <-  2
  levels_IS <- paste0("IS_a_", alphas, "_p_", frac)
  n_rep <- 50
  AR <- do.call(rbind,
                mclapply(1:n_rep, function(i){
                  tic()
                  res <- GrandParisPackage:::E_track(observations = observations,
                                                     ind_tracked = 0,
                                                     observationTimes = times,
                                                     thetaStart = trueTheta,
                                                     particleSize = n_particle,
                                                     backwardSampleSize = n_tilde_AR,
                                                     sigma2Start = trueSigma2,
                                                     nIterations = 1)
                  my_tictoc <- toc()
                  dur <- my_tictoc$toc - my_tictoc$tic
                  data.frame(N = n_particle,
                             N_tilde = n_tilde_AR, Xhat = res, Time = dur,
                             Method = factor("AR",
                                             levels = c("AR", levels_IS)))
                }, mc.cores = 10))
  IS <- do.call(rbind.data.frame,
                mapply(function(my_alpha, my_prop){
                  n_tilde_IS = ceiling(my_prop * n_particle^my_alpha)
                  do.call(rbind.data.frame,
                          mclapply(1:n_rep, function(i){
                    tic()
                    res <- GrandParisPackage:::E_track_IS(observations = observations,
                                                          ind_tracked = 0,
                                                          observationTimes = times,
                                                          thetaStart = trueTheta,
                                                          particleSize = n_particle,
                                                          backwardSampleSize = n_tilde_IS,
                                                          sigma2Start = trueSigma2,
                                                          nIterations = 1)
                    my_tictoc <- toc()
                    dur <- my_tictoc$toc - my_tictoc$tic
                    names(dur) = NULL
                    data.frame(N = n_particle,
                               N_tilde = n_tilde_IS, Xhat = res, Time = dur,
                               Method = factor(paste0("IS_a_", my_alpha, "_p_", my_prop),
                                               levels = c("AR", levels_IS)))
                  }, mc.cores = 10))
                }, alphas, frac, SIMPLIFY = F))
  return(rbind.data.frame(AR, IS))
}

my_n_particles <- c(50, 100, 200, 500, 1000, 2000)
set.seed(my_seed)
res_npart  <- lapply(my_n_particles,
                 function(my_n) compare_on_n(n_particle = my_n,
                                             alphas = c(0.5, 0.6, 1),
                                             frac = c(1, 1, 0.1)))
res_df_npart <- do.call(rbind.data.frame, res_npart)

# Plotting results

my_labels <- expression("AR,"~tilde(N)==2, "IS,"~tilde(N)== N^0.5,
                        "IS,"~tilde(N)== N^0.6, "IS,"~tilde(N)== N/10)
main_plot <- res_df_npart %>%
  mutate(Method = factor(Method, labels = my_labels)) %>%
  ggplot(aes(x = factor(N), fill = Method)) +
  labs(x = expression(N)) +
  theme(text = element_text(family = "Symbola", size = 24)) +
  scale_fill_discrete(labels = my_labels) +
  theme(legend.text.align = 0)

p1 <- main_plot +
  geom_boxplot(aes(y = Time)) +
  scale_y_continuous(trans = "log10") +
  labs(y = "Comput. time (seconds)")
p2 <-  main_plot +
  geom_boxplot(aes(y = Xhat)) +
  labs(y = expression(hat("\U1d53c")~"["~X[0]~"|"~Y[0:n]~"]"))
my_plot <- gridExtra::grid.arrange(p1, p2)

ggplot(res_df_npart, aes(x = N, y = Time)) +
  geom_point() + geom_smooth() +
  facet_wrap(~Method, scales = "free_y") +
  labs(x = expression(tilde(N)),
       y = "Comput. time (seconds)")
```

3 Online estimation in the Sine model

In this section, the online estimation is performed

```r
rm(list = ls())
library(GrandParisPackage)
library(tidyverse)
library(parallel)
```

3.1 Simulating data

3.1.1 Simulation parameters

```r
# True parameters
trueTheta <- pi/4; trueSigma2 <- 1;
n <- 5000; times <- seq(0, by = 1, length = n)
```

3.1.2 Simulation

```
# The following code creates 500 trajectories in a directory "simulated_data"
# that can be created with the following code
# dir.create("simulated_data")
n_traj <- 500
# For windows user, replace by lapply and remove the mc.cores argument below
mclapply(1:n_traj,
         function(i){
           seed <- 100 + i
           set.seed(seed)
           simulated_POD <- SINE_simulate(theta = trueTheta,
                                          sigma = trueSigma2, x0 = 0,
                                          times = times)
           write.table(simulated_POD, paste0("simulated_data/simul_data_seed", seed, ".txt"),
                       col.names = T, row.names = F, sep = ";")
         },
         mc.cores= detectCores())
```

3.2 Estimation

Code for estimation is only shown for one trajectory. This gves the result of the first part of section 5.3 (the Figure 4)

```
# Get the appropriate trajectory
seed <- 122
set.seed(seed) # For reproducibility
simulated_POD <- read.table(paste0("simulated_data/simul_data_seed", seed, ".txt"),
                            sep = ";", header = T)
observations <- simulated_POD[, "observations"]


# Estimation parameters ---------------------------------------------------

get_estimation_one_obs_several_start <- function(){
  gradientSteps <- get_grad_steps(0.6, cst = 8)
  n_start_points <- 50
  n_particles <- 100
  N_tilde <- n_particles / 10
  allRes <- mclapply(1:n_start_points, function(seed){
    set.seed(seed)
    thetaStart <- runif(1, 0, 2 * pi)
    fastTangOR(observations, times, particleSize = n_particles,
               thetaModel = thetaStart, sigma2 = trueSigma2,
               updateOrders = rep(TRUE, n),
               gradientSteps = gradientSteps,
               all = FALSE, estimateTheta = TRUE, estimateSigma2 = FALSE,
               randomWalkParam = 1, backwardSampleSize = N_tilde, IS = TRUE)
  },
  mc.cores = detectCores() - 1)
  thetaEst <- sapply(allRes, function(x) x$Estimates[,1]) %% (2*pi)
  out_path <- paste0("simulation_results/est_oneObs_severalStarts_",
                     ifelse(IS, "IS", "AR"), ".txt")
```

```r
  write.table(thetaEst, file = out_path, col.names = F,
              row.names = F, sep = ";")
  return(NULL)
}

get_estimation_one_obs_several_start(TRUE)
```

4 Experiments for theLotka Volterra Model

4.1 Installation of the required package

The required codes are packages into the `LotkaVoltR` package for the `R` software, available on github.

To install this package in `R`, simply run:

```r
devtools::install_github("papayoun/LotkaVoltR")
```

4.2 Smoothing on synthetic data

4.2.1 Simulating data

```r
# Cleaning

rm(list = ls()) # Cleaning environment

# Librairies

library(LotkaVoltR) # Dedicated library
library(tidyverse) # For data processing

# Dynamics parameters

a1 <- c(12, 0.05, 1) # Prey parameters
a2 <- c(2, 0.2, 0.1) # Predator parameters
Gamma <- matrix(c(0.5, 0.1, 0.1, 0.2), nrow = 2) # Diffusion parameters
mu0 <- c(50,20) # Mean of the inital distribution
Sigma0 <- diag(1, 2) # Variance of the initial distribution

# Observation process parameters

Sigma_obs <- matrix(c(0.01, 0.005, 0.005, 0.01), ncol = 2) # Observation noise
q_values <- c(0.2, 0.3) # Known q values

# Model creation

# Creation of a Partially Observed Lotka Volterra model
POLV_model <- POLV_create(a1 = a1, a2 = a2, gam = Gamma, mu0 = mu0,
                          sigma0 = Sigma0, cov = Sigma_obs, qs = q_values)

# Synthetic data simulation
# Simulation times for the Euler scheme

simulation_times <- seq(from = 0, by = 1e-6, # Simulation time step, small!!
                        length.out = 3*10^6 + 1)

# If the simulation must be done at small time steps, the output can be thinned
# Here we keep 301 points

selection <- seq(1, length(simulation_times), length.out = 301)
set.seed(333)
simulated_process <- POLV_simulate(POLV_model,
                                   times = simulation_times,
                                   selection = selection)

# Extracting observations for smoothing

observation_times <- simulation_times[selection]
observations <- simulated_process[, c("Y1", "Y2")]
```

4.2.2 Performing smoothing

```r
parameters_list <- list(a1 = a1, a2 = a2, mu0 = mu0,
                        sigma0 = Sigma0,
                        RWC = diag(0.005, 2), # Parameter for the particle filter
                        qs = q_values, cov = Sigma_obs,
                        wD = 1, w0 = 1,
                        gam = Gamma)
particle_filter <- PF_create(parameters_list, t(observations),
                             observation_times, 2e2,
                             n_euler_skel = 30)

# Performing the smoothing
# This creates the particles and their (filtering) weights, together
# with computing the wanted E[X_k | Y_{0:n}] for all k

smoothing_exp <- particle_filter$runSmoothing()
```

4.2.3 Plotting results

```r
# Creating labels for the plot
my_levels <- c(obs = 'Y[t]',
               est = 'hat("\U1d53c")~"["~X[t]~"|"~Y[0:n]~"]"',
               true = 'X[t]')

smoothing_mean <- smoothing_exp %>%
  as_tibble() %>%
  rename(X1 = V1, X2 = V2) %>%
  mutate(method = my_levels["est"])
observed_values <- simulated_process %>%
  as.data.frame() %>%
  select(Y1, Y2) %>%
  mutate(Y1 = Y1 / q_values[1], # Putting back in the actual state space
         Y2 = Y2 / q_values[2]) %>%
  rename(X1 = Y1, X2 = Y2) %>%
  mutate(method = my_levels["obs"])
true_values <- simulated_process %>%
  as.data.frame() %>%
  select(X1, X2) %>%
  mutate(method = my_levels["true"])

concatened_results <- bind_rows(observed_values,
                                true_values,
                                smoothing_mean) %>%
  mutate(method = factor(method, levels = my_levels)) # Pour l'ordre
ggplot(concatened_results) +
  aes(x = X1, y = X2, color = method, linetype = method) +
  geom_point() +
  geom_path() +
  theme(legend.position = "none") +
  facet_wrap(~method, labeller = label_parsed) +
  labs(x = "Number of preys", y = "Number of predators") +
  theme(strip.text = element_text(size = 24))
```

4.3 Experiments on Lynx data

4.3.1 Data set

```r
## Data sourced from
## http://www-rohan.sdsu.edu/~jmahaffy/courses/f00/math122/labs/labj/q3v1.htm
## Originally published in E. P. Odum (1953), Fundamentals of Ecology,
## Philadelphia, W. B. Saunders.
## Initial values inspired from https://gist.github.com/mages/1f0f0d5bbe50af81cc19

rm(list = ls()) # Cleaning environment
raw_data <-  "Year   Hares   Lynx
1900    30  4
1901    47.2    6.1
1902    70.2    9.8
1903    77.4    35.2
1904    36.3    59.4
1905    20.6    41.7
1906    18.1    19
1907    21.4    13
1908    22  8.3
1909    25.4    9.1
1910    27.1    7.4
1911    40.3    8
1912    57  12.3
1913    76.6    19.5
1914    52.3    45.7
1915    19.5    51.1
1916    11.2    29.7
1917    7.6 15.8
```

```
1918    14.6    9.7
1919    16.2    10.1
1920    24.7    8.6"

hares_lynx_data <- read.table(text = raw_data, header = TRUE)
```

4.3.2 Performing EM for estimation

** Initial parameters **

```
# Dynamics parameters

a1 <- c(0.695617134, 0.001147425, 0.030333635) # Prey parameters
a2 <- c(0.630905980, 0.021756719, 0.001352425) # Predator parameters
Gamma <- matrix(c(0.22, 0, 0, 0.2), nrow = 2) # Diffusion parameters
mu0 <- c(30, 4) # Mean of the inital distribution
Sigma0 <- diag(1, 2) # Variance of the initial distribution

# Observation process parameters

Sigma_obs <- matrix(c(0.01, 0.00, 0.00, 0.01), ncol = 2) # Observation noise
q_values <- c(1, 1) # Known q values
initial_param <- list(a1 = a1, a2 = a2, mu0 = mu0,
                      sigma0 = Sigma0, RWC = diag(0.005, 2),
                      qs = q_values, cov = Sigma_obs,
                      wD = 1, wO = 1,
                      gam = Gamma)

# Model creation
```

EM functions: A generalized EM is performed. At each step, candidates are generated using a home made (not optimized!) evolution strategy, by sampling around the current parameter through a Gaussian distribution. Each parameter lives in a constrained space. Each new offspring is generated in the unconstrained $\mathbb{R}$ space before being put back to the constrained space. This is mainly done with the logistic function.

Then, a usual EM approach is performed. To ensure good fit, the EM is performed from three different starting points.

```
library(LotkaVoltR) # Dedicated library

# Function generation candidat --------------------------------------------

generate_candidate_list <- function(par_list, # Current parameter
                                    standard_dev_list, # Standard Deviation for
                                    # the future offspring generation
                                    up_lim_list){ # Limits of the constrained space
  out <- par_list
  # Generating a1, in [0, 1] * [0, 0.05] * [0, 0.05]
  sigmoid <- function(x){
    1 / (1 + exp(-x))
  }
  logit <- function(x){
    log(x / (1 - x))
  }
  out$a1 <- (par_list$a1 / up_lim_list$a1) %>% # Back to [0, 1]
    logit() %>% # Back to the real world
    rnorm(n = 3, sd = standard_dev_list$a1) %>% # Moving a bit
    sigmoid() %>% # Retour dans [0,1]
    {. * up_lim_list$a1} # Back to the constrained space
  out$a2 <- (par_list$a2 / up_lim_list$a2) %>% # Back to [0, 1]
    logit() %>% # Back to the real world
    rnorm(n = 3, sd = standard_dev_list$a2) %>% # Moving a bit
    sigmoid() %>% # Retour dans [0,1]
    {. * up_lim_list$a2} # Back to [0, 1] * [0, 0.1] * [0, 0.1]
  diag(out$gam) <- (diag(par_list$gam) / up_lim_list$gam) %>% # Back to [0, 1]
    logit() %>% # Back to the real world
    rnorm(n = 2, sd = standard_dev_list$gam) %>% # Moving a bit
    sigmoid() %>% # Back to [0,1]
    {. * up_lim_list$gam}
  diag(out$cov) <- (diag(par_list$cov) / up_lim_list$cov) %>% # Back to [0, 1]
    logit() %>% # Back to the real world
    rnorm(n = 2, sd = standard_dev_list$cov) %>% # Moving a bit
    sigmoid() %>% # Back to [0,1]
    {. * up_lim_list$cov}
  return(out)
}


EM_function <- function(obs, obs_times, initial_param, initial_sd, up_lims,
                        n_cands, n_iter,
                        seed,
                        name = NULL){
  param_0 <- initial_param
  out <- list(param_0)
  set.seed(seed)
  final_E_steps <- matrix(NA, nrow = n_iter, ncol = n_cands + 1)
  for(i in 1:n_iter){
    print(paste("Iteration", i))
    par_list <- c(purrr::rerun(n_cands,
                               generate_candidate_list(param_0,
                                                       initial_sd,
                                                       up_lims)),
                  list(param_0))
    E_step_evals <- get_E_step(obs_ = t(obs),
                               obsTimes_ = obs_times,
                               myParams = param_0,
                               testedParams = par_list,
                               n_part = 200, n_dens_samp = 100)
    initial_sd <- purrr::map(initial_sd,
                             function(x) 0.9 * x) # Reducing the exploration
    param_0 <- par_list[[which.max(E_step_evals)]]
    out <- c(out, list(param_0))
    final_E_steps[i, ] <- E_step_evals
  }
  return(list(out, final_E_steps))
}
```

```r
observations <- as.matrix(hares_lynx_data[, c("Hares", "Lynx")])
observation_times <- hares_lynx_data$Year

initial_param <- list(a1 = a1, a2 = a2, mu0 = mu0,
                      sigma0 = Sigma0, RWC = diag(0.005, 2),
                      qs = q_values, cov = Sigma_obs,
                      wD = 1, wO = 1,
                      gam = Gamma)
sd_list <- list(a1 = c(1, 1, 1),
                a2 = c(1, 1, 1),
                cov = 0.5,
                gam = 0.5)
upper_lims_list <- list(a1 = c(1, 0.01, 0.05),
                        a2 = c(1, 0.05, 0.01),
                        cov = 0.5,
                        gam = 0.5)
library(parallel) # For parallel computing
results <- mclapply(1:30, function(my_seed){
  EM_function(obs = observations,
              obs_times = observation_times,
              initial_param = initial_param,
              initial_sd = sd_list,
              up_lims = upper_lims_list,
              n_cands = 10,
              n_iter = 30, seed = my_seed,
              name = "lynx")},
  mc.cores = 10)
```

The results are then used to perform the smoothing as in previous section