Guide Line for the Application of Approximative Bayesian Computing Algorithms

Sampling of the Posterior Distribution without Likelihoods

Andreas Scheidegger

March 1, 2010

It is often impossible to derive the likelihood function of complex stochastic models. Nevertheless, the posteriori distribution of the parameter can be computed with approximative Bayesian computing algorithms—without using a likelihood function. However, there are some pitfalls in the practical application. This guide line provides a generic setup for the use of efficient approximative Bayesian computing methods.

Contents

1 Introduction

The classical Bayesian methods to compute the posterior distribution of the parameters are difficult to apply for stochastic models. The traditional methods require a likelihood function which is often impossible to derive for stochastic models. However, approximative Bayesian computing (ABC) methods allows sampling from the—at least approximative—posterior distribution without using a likelihood function.

Although some ABC methods are quite simpel, there are several pitfalls in the application. Efficient sampling algorithms like Markov Chain Monte Carlo (MCMC) or Partial Rejection Control (PRC) adapted for ABC require five things:

1. A stochastic model $M(\theta)$ which can simulate data given the parameter vector θ
2. A set of summary statistics S_i which *sufficiently* described the (simulated or real) data
3. A distance function $\rho(\cdot, \cdot)$ which quantifies the (dis)similarity of data

4. A tolerance ϵ which is the maximum distance to accept a proposed parameter vector
5. A transition kernel $q(\cdot)$

Especially the points 2 and 4 are ticklish. The choice of the summary statistics is important, because they should reflect all characteristics of the data. A summary statistic is *sufficient* "when no other statistic which can be calculated from the same sample provides additional information as to the value of the parameter [of the distribution]" (Fisher, 1922; see also Wikipedia, 2010).

A Preliminaries

A1 Presampling

Samples of a presampling are necessary to find the right summary statistics and for choosing the tolerance.

1. sample N_{pre} parameter vectors θ_n, $n = 1, \ldots, N_{pre}$ from the prior density $\pi(\theta)$
2. simulate data D_n^{sim} with model $M(\theta_n)$ for each parameter vector θ_n, $n = 1, \ldots, N_{pre}$
3. compute all I summary statistics for each D_n^{sim}

B Sampling

After the preliminaries, the rest of the sampling can be done with more efficient sampling methods. Possible algorithms are Markov chain Monte Carlo without likelihoods and Partial Rejection Control without likelihoods (see below). 9).

2 Literature

Fisher, M.A. (1922). On the Mathematical Foundations of Theoretical Statistics. *Philosophical Transactions of the Royal Society.* A: 222: 309–368. http://digital.library.adelaide.edu.au/dspace/bitstream/2440/15172/2/18pt1.pdf, accessed January 28, 2010.

Gilks, W., Richardson, S. & Spiegelhalter, D. (1998). *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics.* 2nd Edition. Chapman & Hall.

A Miscellaneous

A.1 Variance of the sample mean

Variance of the sample mean:

$$Var(\hat{\mu}) = \frac{\sigma^2}{n} \tag{1}$$

$\hat{\mu}$: the sample mean
σ: standard deviation
n: number of *independed* samples

A.2 Variance of the p-th sample quantile

Approximative variance of the p-th sample quantile:

$$Var(\hat{x}_p) = \frac{p(1-p)}{n \cdot f(\hat{x}_p)^2} \tag{2}$$

$\hat{x}_p$: the p-th quantile of x
$f()$: density function n: number of *independed* samples

B R-code

B.1 Generic Markov Chain Monte Carlo sampler without likelihoods

```
# ============================================================================
# Approximative Bayesian Computing (ABC)
# Generic Markov Chain Monte Carlo sampler without likelihoods
#
# Andreas Scheidegger
# 10.11.2009
# ============================================================================

# See Marjoram et al. (2003), Markov Chain Monte Carlo without likelihoods, PNAS
    100(26), pp 15324-15328.
# Implementation of algorithm F

# -- Arguments --
# f.dist: function which simualtes data and returns the distance between the real
    and the simulated data.
#         The first argument must be the parameter vector.
# d.priori: function which returns the density of the prior distribution of a
    parameter vector.
# n.sample: number of samples
# eps: accepted tolerance between data and model output
# init: initial values
# sigma: vector of standard dev. of the gaussian proposal distribution
# verbose: number of samples between printed outputs
# ...: arguments for f.dist
```

```
#
# -- Value --
# matrix containing in each row a (autocorrelated) sample of all parameters and
    the corresponding delta value

ABC.MCMC <- function (f.dist, d.priori, n.sample, eps, init, sigma, verbose=100,
    ...) {

  # Number of parameter
  n.para <- length(init)

  # Matrix to store samples
  sample <- matrix(NA, ncol=n.para, nrow=n.sample)

  # Vector to store  distances
  delta <- rep(NA, n.sample)

  # Initional values
  sample[1,] <- as.matrix(init)
  delta[1] <- f.dist(sample[1,], ...)

  # repeat for each sample
  for (k in 2:n.sample)  {

    # print
    if((k %% verbose)==0) cat("k =", k, "\n")

    # F1: generate candidate point
    x.prob <- sample[k-1,]+rnorm(n.para,sd=sigma)    # gaussian proposal density (
        rounded to integer)

    # F2, F3: simulate data and calculate distance to the real data
    dist <- f.dist(x.prob, ...)

    # if dist <= eps go to F4
    if(dist <= eps) {

      # F4: calculate prob. h (due to the symmetrie of the proposal density, it is
          not necessary in F4)
      h <- min(1, d.priori(x.prob)/d.priori(sample[k-1,]) )

      # F5: accept x.prob mit prob. h
      if(runif(1)<h) {
        sample[k,] <- x.prob
        delta[k] <- dist
      # else stay on the same point
      } else {
        sample[k,] <- sample[k-1,]
        delta[k] <- delta[k-1]
      }

    # else stay on the same point
    } else {
      sample[k,] <- sample[k-1,]
      delta[k] <- delta[k-1]
    }

  }

  # combine samples and delta
  sample <- cbind(delta, sample)
```

```
  if(length(names(init))!=n.para) colnames(sample) <- c("delta", paste("para.",1:n
      .para, sep="") )
  if(length(names(init))==n.para) colnames(sample) <- c("delta", names(init) )

  #return samples and distances
  return(sample)
}
```