evolMC: a package for Monte-Carlo simulation

W Burchett E Roualdes G Weyenberg

November 22, 2013

1 Introduction

2 Methods

3 Results

3.1 Software

We present the R package evolMC, which is a framework for doing Markov-Chain Monte Carlo simulations. It includes explicit support for several MCMC algorithms, including Metropolis-Hastings, Gibbs, parallel tempering, and the evolutionary updating methods.

The package aims to provide a general framework for constructing a chain, leaving it up to the user to provide the specific pieces specific to the problem. The user must first define a *state object* and then implement a number of functions which act on this state object. In many traditional cases this state object can simply be a vector of reals, but any R object (e.g. a phylogenetic tree object defined by a 3rd-party package) can be used. The functions provided by evolMC will eventually produce a *chain* object, which is simply an R list where each element is an instance of the state object.

4 Discussion

A evolMC

The evolMC package includes a vignette demonstrating a few basic use cases, with example code. Here we briefly discuss the architectural vision underlying the package and a few of the most important functions.

evolMC was designed to be as flexible as possible, and hence it operates on a quite abstract level. It is written in a very functional style of programming; most of the important functions in evolMC accept functions as arguments and return new functions. An important goal was for the user to be free to use whatever object they wish to represent a realization of the random element in

question. This departs from many existing R MCMC packages, which require vector- or array-type structures for the random elements in question.

The main job of the user of evolMC is to define a few basic functions which can interact in specified ways with whatever object is selected to represent states of the chain. For example, the user will often need to implement a function which evaluates a log-densitiy for a given state. Using the routines provided by evolMC, one will then manipulate these functions, eventually producing a function which implements some desired MCMC algorithm.

For example, the core function in evolMC is `iterate(n,f,x)`. This function is also among the least interesting, because it simply creates a list where the k-th element is $f^{k-1}(x)$. (Function exponentiation means iterated application here.) If the provided function f is "carefully selected" then the resulting chain might have "desirable properties". Of course, we are likely interested in the case where the "desirable property" in question is that the resulting list represents a sample from some target distribution. Other functions in the package help one to construct "carefully selected" functions `f`.

Perhaps the most basic MCMC algorithm is the Gibbs sampler. Suppose we want to sample from $[x, y]$. If we can sample from $[x|y]$ and $[y|x]$, then we can implement a Gibbs sampler. To do this one need only implement functions `rxt` and `ryx`, which do the respective sampling. These functions should accept the current state as their first argument, and return a new state object with the relavant changes made.

Once one has these functions, the gibbs function can be used to produce a new updating function which will implement the Gibbs algorithm. The basic signature is `gibbs(...)`. The ... argument allows one to provide any number of updating functions; in our case we have two, `rxt` and `ryx`. The result of `gibbs(rxy,ryx)` is a new "carefully selected" function with signature `function(state)` which acts as a Gibbs updater. If called repeatedly (e.g. by iterate) it will alternate between using `rxy` and `ryx` to update the state.[1]

A second important algorithm is the Metropolis algorithm. The function `metropolis` helps one to construct an updating function which implements Metropolis-Hastings. To use this function one must first implement: the log-density function for the target distribution (`ln.d`); a proposal generator function (`r.prop`); and (if required) a log-density function for the proposal generator (`ln.d.prop`). A call `metropolis(ln.d, r.prop, ln.d.prop)` will return a "carefully selected" function (with signature `function(state)`), which implements the Metropolis-Hastings scheme you have defined.

These three basic building blocks can be used to construct a suprising number of more advanced MCMC algorithms, and indeed many of the more advanced agorithms provided by evolMC are constructed using only these pieces.

[1] The sequential choice of updating functions is the default mode of operation for the function returned by `gibbs`. It can also stochastically select an updating function according to provided weights.