EP4
Markov Chain Monte Carlo

Definitions, tuning, parameters and precision

André Vinícius Rocha Pires
10737290

May 2019

1 Introduction

In our sequence of experiences applying Monte Carlos Methods in numerical integration, we get to Markov Chain Monte Carlo(MCMC). Now, besides the challenge to calculate the integral with a good precision with the minimum number of samples, we have some concepts to think about. Other problem to solve is that there is a algorithm parameter which needs to be tuned for each function we will integrate. With these questions to be answered, this report is organized with a section for MCMC details and a code section for documentation.

2 Particularities on Markov Chain Monte Carlo

Applying MCMC, one can sample from a function without knowing its *normalizing constant*. This is possible thanks to using either *Metropolis-Hastings Algorithm* or the *Metropolis-Barker Algorithm*. Although this is not the code section, we will explain here how this is implemented. The mathematical proof of its correctness is beyond the scope of this report.

Remembering the definitions of a Markov Chain, we have a *stationary* distribution S, and a kernel. The first particularity is that here we are doing continuous Markov Chain and our kernel will be one of that *algorithms* together with a *proposal* distribution P. Then:

the Metropolis-Hastings algorithm consists in the following:

1. Pick a initial state x_0;
2. Set $t = 0$;
3. Iterate:
   (a) randomly generate a candidate x' according to $P(x'|x_t)$;

   (b) calculate the acceptance probability $A(x', x_t) = min(1, \frac{S(x')P(x'|x_t)}{S(x_t)P(x_t|x')})$

   (c) Accept or Reject x'

      i. generate a random number $u \sim U(0,1)$

      ii. if $u \leq A(x', x_t)$ accept $x_{t+1} = x'$

      iii. else $x_{t+1} = x_t$

   (d) set $t = t + 1$

the Metropolis-Barker algorithm consists in the following:

1. Pick a initial state x_0;

2. Set $t = 0$;

3. Iterate:

   (a) randomly generate a candidate x' according to $P(x'|x_t)$;

   (b) calculate the acceptance probability $A(x', x_t) = min(1, \frac{S(x')P(x'|x_t)}{S(x_t)P(x_t|x')+S(x')P(x'|x_t)})$

   (c) Accept or Reject x'

      i. generate a random number $u \sim U(0,1)$

      ii. if $u \leq A(x', x_t)$ accept $x_{t+1} = x'$

      iii. else $x_{t+1} = x_t$

   (d) set $t = t + 1$

The second particularity refers to the calculation of our acceptance probability. It's the proof that we don't need to normalize our function:

If z is a normalizing constant, then:

$$S(x) = \frac{s(x)}{z} \Rightarrow \frac{S(x')}{S(x_t)} = \frac{s(x')}{s(x_t)}$$

Besides this, if P is symmetric:

$$P(x'|x_t) = P(x_t|x')$$

Therefore:
on Hastings

$$\frac{S(x')P(x'|x_t)}{S(x_t)P(x_t|x')} = \frac{s(x')P(x'|x_t)/z}{s(x_t)P(x'|x_t)/z} = \frac{s(x')}{s(x_t)}$$

on Barker

$$\frac{S(x')P(x'|x_t)}{S(x_t)P(x_t|x') + S(x')P(x'|x_t)} = \frac{s(x')P(x'|x_t)/z}{(s(x_t) + s(x'))P(x'|x_t)/z} = \frac{s(x')}{s(x_t) + s(x')}$$

Knowing what is the kernel for our continuous Markov Chain, comes the third particularity: we define our *proposal* distribution as $P = N(x_t, \sigma)$. But then we have to discuss what value to give to σ so our algorithm has its best performance. This will be done on the code section.

The last particularity refers to a known *praxis* in MCMC, the *burn-in*. It consists on not including a quantity of the first samples in the integral calculation. Here, we will not use *burn-in* on our MCMC. Apart from being a waste of efficiency, our reason is statiscal: if we are making inference from the random numbers generated, and all these events are actually inside the sample space(domain of S), then they contribute fairly to our estimate. There is also, in our opinian, a misunderstanding of this concept. In this link, at the bottom of the page, there is the best explanation we have seen on this subject, which reveals that what people do ins't really *burn-in*.

3 Code Documentation

Markov Chain Monte Carlo(MCMC) consists on a group of nested functions. We will start from the inner functions and then go to the outer ones.

3.1 Function *step*

- **Description**
  This is the function which implements the *Metropolis Algorithm* for one step, i.e. $x_{i+1}|x_i$. To do this, the function receives x_i as input and returns x_{i+1}. The implementation of the algorithm was explained in the previous section.

- **Math**
  The basis for this Algorithm are the Markov Chains and the Ergodic Theory. The details are beyond the scope of this report.

- **Usage**
  step(x, s, p, mode = "barker")

- **Arguments**
  - x: the actual x of the Markov Chain
  - s: the potential stationary distribution
  - p: the proposal distribution
  - mode: the algorithm mode. Default "barker"

- **Details**
  It's important to note that the way the *Metropolis Algorithms* are implemented prevents that a x is generated out of the stationary function domain. This is guaranteed because in these regions the probability is *zero*. But also because of this, its necessary to protect the calculation

of the acceptance probability from division by zero. Both the restriction on the stationary function domain and the protection to division by zero were implemented using a if.

3.2 Function *run*

- **Description**
  This function produces a Markov Chain. It receives a x_0 as input and generates the sequence applying iteractively the *step* function.
- **Math**
  The basis for this Algorithm are the Markov Chains and the Ergodic Theory. The details are beyond the scope of this report.
- **Usage**
  run(x, s, p, nsteps)
- **Arguments**
  - x: the x_0 of the Markov Chain
  - s: the potential stationary distribution
  - p: the proposal distribution
  - nsteps: the number of steps (samples)

3.3 Function *mcmc*

- **Description**
  This function produces one estimate for the density of a given interval in our potential stationary distribution.
- **Math**
  The estimation of the density to the interval is given by the proportion of samples in the interval by the total of samples. The basis for this is:

$$P(X_n = x | X_0 = x_0) \xrightarrow[n \to \infty]{} S(x)$$

- **Usage**
  mcmc(s, p, lower, upper, nsteps = 1000)
- **Arguments**
  - s: the potential stationary distribution
  - p: the proposal distribution
  - lower: lower bound of integration
  - upper: upper bound of integration
  - nsteps: the number of steps (samples). Defaut 1000.

3.4 Function *tuning*

- **Description**
  This function estimates the best σ for using in the proposal distribution.
- **Math**
  The basis for this function is the autocorrelation between the poins in the Markov Chain. Knowing that small correlation implies independence, we sum the total autocorrelation of a given sequence. This is made for each tried σ and put in a vector. Then, we can plot this vector and choose the smaller sum to be our optimal σ.
- **Usage**
  tuning(s, maxsigma, nsteps)
- **Arguments**
  - s: the potential stationary distribution
  - maxsigma: integer. The maximum σ to be experimented
  - nsteps: the number of steps(samples) for the *run* function nested in this function
- **Details**
  The most interesting thing about this method of finding the best σ is to observe the evolution of the autocorrelation on the plot. Also, there is a connetion between this and the real meaning of *burn-in*.

3.5 Function *mcmc_integrate*

- **Description**
  This function returns the mean and variance of a series of estimates obtained through the function *mcmc*. The number of estimates is passed to this function as an argument.
- **Math**
  The basis to this function is the Central Limit Theorem.
- **Usage**
  mcmc_integrate(s, p, lower, upper, nruns = 10000)
- **Arguments**
  - s: the potential stationary distribution
  - p: the proposal distribution
  - lower: lower bound of integration
  - upper: upper bound of integration
  - nruns: number of estimates. Default = 8000

3.6 Function *mcmc_precision*

- **Description**
  The problem of defining the the best number of iterations was solved with a R function that recursively executes the *mcmc_integrate* function, until it reaches a mathematical precision criteria given by the arguments. To do this, is implemented a loop with a stopping criteria that depends on an error tax. This error tax is calculated empirically, with the data produced by the function itself. The returned data, besides the resulting integral, shows the efficiency of the applied method through the number of iterations, the variance reduction and the estimated error tax:

  - n : the minimum necessary number of iterations for a given error tolerance
  - variance : the variance achieved
  - expected_value: the integral desired
  - error : the estimated error tax

- **Math**
  The basis to this function is the Central Limit Theorem.

- **Usage**
  mcmc_precision(s,p,lower, upper,CI = 0.95, errtolerance = 0.01)

- **Arguments**

  - s: the potential stationary distribution
  - p: the proposal distribution
  - lower: lower bound of integration
  - upper: upper bound of integration
  - CI: the confidence interval used to define the error. Default = 0.95
  - errtolerance: the wanted precision. Default = 0.01

- **Details**
  There is an implementation of a progress bar which shows the evolution of the precision, and the n is incremented in a thousand on each iteration.