MCMC-sampling Report

Qiu Wei

May 18, 2017

Contents

1 Introduction

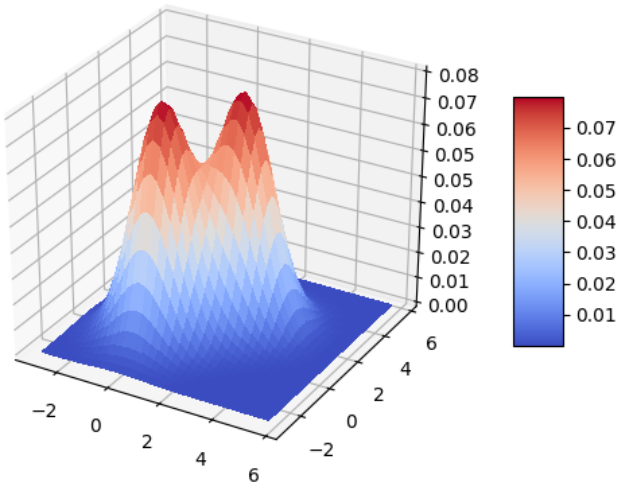
This experiment realized a sampling algorithm using MCMC sampling. The density function is the average of two Gauss function in the plane.

To evaluate the performance of this algorithm, I use two methods. The first is comparing the density of a specific point calculated by Monto Carlo method and its accurate value calculated by density function. The second one is KL divergence, which is commom in calculating the distance between two distributions.

The core functions of sampling algorithm is written in Clojure, and the evaluation part,including calculate the KL divergence and draw the graphs, is written in Python3.5.

2 MH sampling

The density function is the average of two Gauss function, the first is standard Gauss function and the second is shifting the center of standard Gauss function to point [3 3]. Here is the distribution of it:



I use MH sampling algorithm to get a sequence of samples. The core function is mh-get-next-sample.

```
(defn- mh-get-next-sample
  "Get next sample in MH algorithm"
  [previous-sample]
  (let [new-sample (util/gauss-sample-2d)
        threshold (/ (* (util/standard-gauss-function-2d previous-sample)
                        (density-function new-sample))
                     (* (util/standard-gauss-function-2d new-sample)
                        (density-function previous-sample)))
        alpha (min threshold 1)
        u (util/uniform-sample)]
    (if (<= u alpha)
      new-sample
      previous-sample)))
```

The principle of this algorithm can be found in many websites so I will not restate it.

3 Evaluation

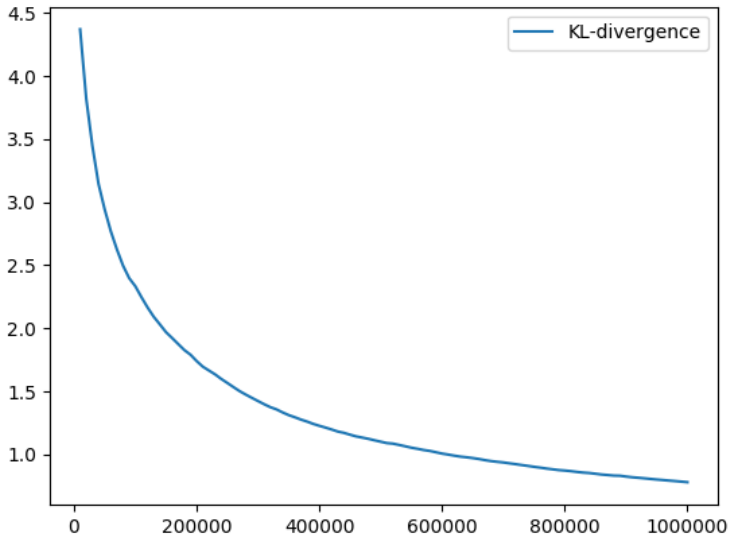
3.1 Comparing Density Function

I choose 5 points to do this test and here is the result:

```
mcmc-sampling.core> (mc-calc-probability-density [0 0])
0.084224795884231
mcmc-sampling.core> (density-function [0 0])
0.08103498377846177
mcmc-sampling.core> (mc-calc-probability-density [1 1])
0.05681831468380663
mcmc-sampling.core> (density-function [1 1])
0.05854983152431917
mcmc-sampling.core> (mc-calc-probability-density [-1 -1])
0.02342760762312699
mcmc-sampling.core> (density-function [-1 -1])
0.029284736402332784
mcmc-sampling.core> (mc-calc-probability-density [1 2])
0.057454934456174216
mcmc-sampling.core> (density-function [1 2])
0.05479829295736942
mcmc-sampling.core> (mc-calc-probability-density [2 2])
0.07696733047924058
mcmc-sampling.core> (density-function [2 2])
0.08103498377846177
```

Obviously,the density function calculated via samples have 1 digit of significant figure. When I tried farther points, such as [-2 -2], it prints zero. So even with 1000000 iteration, the final product still have little difference with origin function,especially in farther points.

3.2 KL Divergence



The value of KL divergence decreases as the number of iterations increases and the final value is 0.780989858100924.

4 Problems

From different kinds of evaluations, we can point out that MH algorithm converges slowly in this condition.

In fact, the sampling distribution is sparse in most areas, so it is rare to get a sample far away the centroid. Maybe it will converge fastly using Gibbs sampling algorithm.