Locally-informed proposals in Metropolis-Hastings algorithm with applications

1 Markov chains

Let us start with a foundation of this thesis, Markov chains and briefly talk about its properties.

A Markov chain is a stochastic process describing a sequence on some countable state space S in which the probability of each event depends only on the state of previous event.

Such a chain is uniquely defined by a stochastic transition matrix P and a initial distribution

1.1 Properties

A Markov chain is irreducible if there is a possibility to reach every state from every state.

A state is aperiodic if there is a possibility of coming back to it after each step. A Markov chain is aperiodic when every state is irreducible and a state is aperiodic.

1.2 Ergodic chains

A probability distribution pi is called stationary if it satisfies balance equation.

A Markov chain is ergodic when it is irreducible and aperiodic.

Let Xk be a ergodic Markov chain with a transition matrix P, then. This theorem gives us a long-term behavior of a MC and is a basis for approximating distributions. Generating a MC is often easy, so if we simulate it long enough we will obtain observations from a given distribution pi.

2 Markov chain Monte Carlo methods

Now for the question, how to construct a MC such that it has our distribution of interest pi. We can use MH algorithm for that.

Construct a MC, which has a stationary distribution pi. Assume that Q is stochastic matrix which corresponds to an irreducible and aperiodic Markov chain and (...). This constraint is slightly relaxed symmetry. Let us consider a matrix defined as:

A matrix defined in 2 is stochastic, irreducible, aperiodic and has a stationary distribution pi.

2.1 Algorithm

We do not need to construct such a matrix, we can just use M-H algorithm. This algorithm creates a MC with stationary distribution pi.

3 Traveling salesman problem

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

This problem is known to be a NP-hard one, so deterministic algorithms are not enough for it.

Given an undirected weighted graph G = (V, E), |V | = n find a permutation sigma min of vertices such that

where Sn is a set of all permutations of vertices and wi,j is distance (weight) between cities i and j.

4 Approximate solutions

4.1 Softmax

Okay, so how do we combine all of this? To do that we need a softmax function. For a given vector x a softmax is defined just as an exponent of its value divided by a normalizing constant, so that we have a probability.

So now instead of looking for permutation with a minimal weight, we can equivalently look for permutation that maximizes probability. This is the main idea behind **stochastic optimization via MCMC**. We are finding solutions with higher probabilities, so also smaller weights, by looking at most frequently appearing permutations.

For that we need to approximate sampling from this softmax over weights.

4.2 M-H algorithm

Let define distribution pi as a softmax over vector of weights. So when we work with pi, the steps of M-H algorithm will look like this. As we can see working with softmax and logarithms are simplifying equations.

The next step is to define a matrix Q.

4.3 Candidates

But before that let us define a candidate.

A neighbour sigma of a permutation sigma is a permutation, that for some k, l it satisfies (k) = (l), (l) = (k) and (i) = (i) for the rest of indices.

These neighbours are the original tour with two swaped indices. This let us consider a smaller space – there are n(n-1)= n2 neighbours if the number of vertices is n.

4.4 Random candidates

Sample neighbours uniformly. It is **equivalent to choosing random indices** to swap. It is easy to implement. where S is a set of all possible neighbours of sigma.

4.5 Example

Now that we have distribution pi and matrix Q, we need to come up with an efficient way of computing weight for a next candidate in MH step, let us look at example.

When given a tour sigma and its neighbour sigma they differ only on those edges where swap is happening, let us say k, l. So for this situation we have tours:

Assuming we know weight of sigma, to compute difference we need to remove weights and add.

4.6 Locally-informed proposals

The idea is to balance the increase in the probability of neighbour with decrease of reverse probability, such that it will be easy to compute.

The distribution is chosen in such a way, so that we can easily group up the terms in acceptance criterion:

There is a tau parameter, which is called temperature. It was proven that tau = 2 is optimal and it simplifies the equation further.

Unfortunately this time we need to keep a vector of differences between weights to be able to sample from Q.

Again like before we need to come up with an efficient way of computing weight differences for a new candidate.

4.7 Example

Let us set k = 2 and l = 7, then the set of indices to consider is 1, 2, 3, 6, 7, 8 and the permutations:

Assuming we know the difference between those, we know that most of their neighbours will have the same difference.

4.8 Simulated annealing

We can use one more parameter: The idea is to describe a probability of state using a cooling parameter tk such that it reminds the cooling of a metal and may change with each step:

where C is normalizing constant.The quotient of probabilities then is:

Setting tk = 1 will end up with a HMC and tk =/= 1 a non homogeneous MC.

5 Results

Okay, so now for results. Every plot will present distance obtained via algorithms over iterations.

5.1 Initial condition

These plots show us two methods with the same initial conditions. Different colors are for different seeds, so initial conditions. We can see that behaviour of the algorithms is the same. For better initial conditions they can get lower.

5.2 Simulated annealing

These plots show how random neighbours are influenced with cooling parameter 1 and 3 over logarithm of number of steps. This was done only for RN because too see any difference we had to iterate to 1 million, which is infeasible with LIP. We can see no big difference between cooling parameters.

5.3 Temperature

These plots show us how LIP is influenced with temperature parameter. The results were not significant, so we defaulted to tau = 2.

5.4 Algorithms comparison

The blue line here is RN and orange LIP. Red dotted line is the actual optimum for a given dataset. We can see how quickly they diverge. First steps of LIP algorithm is taking the best neighbours it can and stays there or makes small improvements.

5.5 Tables

The tables present concrete results obtained for those methods when running 20k iterations. We can see that LIP has always better results but takes much more time. The last dataset took around 1,5h.

The second table presents the same properties, but ends LIP algorithm when it reaches the same results as RN for 20k iterations. We can see that LIP is getting to the same results in reasonable time.

6 Conclusions

LIP algorithm is decreasing the distance quicker than RN, LIP converges to some value, which is always smaller than the RN, LIP reaches better results in feasible time, Only first hundreds iterations matter, LIP algorithm is considerably slower for a longer run,

7 Improvements

Use of interning in Python3, use of concurrency, sampling from Q using MCMC methods, better tuning of temperature and cooling parameters.