Generation of Points within an n-dimensional Hypercube Subject to Constraints

Emre Sevgen

April 2019

1 Goals

Given a formatted list of constraints formatted as g(x) >= 0.0, we wish to generate points in n-dimensions within a unit hypercube that obey these constraints. The principal goals are:

1. Points are spread as uniformly as possible within the constrained space
2. Execution taking less than 5 minutes.

2 Approach

A few approaches were initially considered and abandoned:

1. Starting with analytic solutions - first loop through constraints that only have one dimension, then loop through constraints that only have two dimensions, then switching to stochastic sampling.
2. Randomly generating points and checking.
3. Starting with a uniform grid and refining.

First approach was abandoned on the basis of making too many assumptions about the nature of the problem, and the difficulty of implementing a quick, clean solution. The second and third were abandoned on that basis that some constraints have minimal volume (e.g. alloy.txt) and thus random sampling is extremely inefficient and uniform grid would require an astronomical amount of points in high dimensions to land within the constraint volume.

Finally, we decided to take advantage of the fact that there's always a single point provided that obey the constraints. Thus, we turned to a very simplified Markov chain Monte Carlo style approach. Thankfully, there are no target or underlying probability distributions that complicate the problem. However, we also wanted to avoid problems that could arise from potentially non-ergodic constraints and did not want to assume that all problems would have connected manifolds or volumes. Therefore, we decided that a certain amount of completely random sampling (from the entire hypercube) should be included to allow escape from an initial volume to an unconnected region of phase space.

The final formula starts from a seed point, and starts proposing moves that could land uniformly anywhere within the hypercube. For each rejected move, the step size is reduced according to:

$$new\_step = old\_step * decay$$

When a move is accepted, the point is added to a running list, and the step-size is either reset back to spanning the entire hypercube, or simply increased (user choice, default = reset). Subsequent moves are proposed from the new point. After a certain number of total proposals, a new point is selected randomly from the pool of all available points and a new series of moves are proposed.

The decay ensures that some progress will always be made eventually, as a string of failed moves will result in very small proposed steps that are increasingly likely to be within the constraint volume. At the same time, resetting the proposal length provides chances for the random sampling to find distant regions in the hypercube that obey constraints, but are hard or impossible to get to via short step random walks.

3 Results, Limitations and Improvements

3.1 Results

The method performs fairly well, providing a decent spread of 1000 points by visual inspection on the alloy.txt problem in appx. 9 seconds with the default parameters (decay = 0.99). Increasing the decay to 0.9995 in alloy.txt allows better sampling, but takes 138 seconds to execute.

3.2 Limitations

The method generates correlated samples unless steps parameter is reduced and decay is very close to 1. At decay 0.99 (the default) some correlation is still observable.

3.3 Improvements

Picking of new points can be improved considerably, also increasing performance. A good idea would be to pick points similar to in k-means++, where a point is more likely to be chosen as seed the farther it is from the geometric center of all points.