CONSTRAINED LATIN HYPERCUBE SAMPLING OF $[0,1]^d$

Subhajit Banerjee
University of California, Davis
June 1, 2018

1 Problem Statement

In $\mathbb{R}^d$ given a unit-hypercube, i.e., $\Omega := [0,1]^d$ and a set of linear or nonlinear inequality constraints $\{g_i(x) \geq 0\}_{i=1}^m$, $m \in \mathbb{Z}^+$, generate `n_results` number of sample points such that:

$$\forall s \in [1, \ldots, \texttt{n\_results}] \text{ and } \forall i \in [1, \ldots m] \quad g_i(\boldsymbol{x}_s) \geq 0$$

where $\boldsymbol{x}_s \in \mathbb{R}^d$; i.e., *all* the sampled points should satisfy *all* the given constraints.

2 Solution

First, Let us define the feasible space $\mathcal{C} \subset \Omega$ as

$$\mathcal{C} := \{\boldsymbol{x} \in \mathbb{R}^d | g_i(\boldsymbol{x}_s) \geq 0 \quad \forall i \in [1, \ldots m]\}.$$

The generated samples would only be useful if they *scope-out* as much of the feasible space $\mathcal{C}$ as possible. Random sampling techniques such as *Monte-Carlo sampling*, *Latin hypercube sampling*, etc. generate random samples of parameter values from a multidimensional probability distribution. In our case, these parameters are the componnets $\{x_j\}_{j=1}^d$ of the vector $\boldsymbol{x} \equiv (x_1, \ldots, x_d)$. For higher-dimensional parameter-space ($d \gg 1$), the Monte-Carlo scheme tends to generate clustered data points. This invalidates the scoping-out of $\mathcal{C}$ with the samples. On the contrary Latin hypercube sampling divides the range of each independent parameters ($x_j \in [0,1]$) equally probable divisions (bins) and the samples are subsequently drawn subjected to the *Latin hypercube requirements*. This leads to more *spread-out* samples $\in \mathcal{C}$. Hence, this is the approach followed to generate the samples for the given problem.

2.1 Algorithm

In 1 we describe the algorithm developed to seek the samples $\boldsymbol{x}_s \in \mathcal{C}$. This is implemented as the method `constraint_latin_hypercube` of the class `Unithypercube` which is contained in the script `sampler.py`.

Algorithm 1 Constrained Latin Hypercube

procedure CONSTRAINTLATINHYPERCUBE($n\_results, \{g_i(\boldsymbol{x})\}_i^m, max\_it$)
  $sample\_size \leftarrow n\_results$ ▷ sample_size as input to LH sampler
  $max\_iter \leftarrow 5$ ▷ Hard-code maximum number of trials as 5
  while $iter \leq max\_iter$ **do**
    $\{\boldsymbol{x}_s\} = LatinHypercubeSampler(sample\_size)$
    for $s \in sample\_size$ **do**
      if $\{g_i(\boldsymbol{x}_s)\}_{i=1}^m \geq 0$ **then**
        $true\_sample\_ind(s) = True$
      else
        $true\_sample\_ind(s) = False$
      end if
    end for
    $count\_pass \leftarrow$ # of True elements in$true_sample_ind$
    if $count\_pass \geq n\_results$ **then**
      Found all the samples satisfying the all the constraints **Break**
    else
      $sample\_size \leftarrow 1.1 * sample\_size$ ▷ Increase the sample size by 10%
      $iter \leftarrow iter + 1$
    end if
  end while
  Return $\{\boldsymbol{x}_s\}$
end procedure

3 A Second Approach

Another approach, which is proposed here but not implemented is described next. To this end, first let us denote $[z]^- = max\{0, -z\}$ and define:

$$h(\boldsymbol{x}) := \sum_{i=1}^{m} [g_i(\boldsymbol{x})]^-.$$

Now, for a given initial guess satisfying $\{g_i(\boldsymbol{x}) \geq 0\}_{i=1}^m$, we seek local minima of $h(\boldsymbol{x})$. Hence, the samples are given as:

$$\boldsymbol{x}_s^* = \arg\min_x \; h(\boldsymbol{x}).$$

One of the trivial ways to do this, using MATLAB say, is to solve an nonlinear optimization problem (solver: `fmincon`) with objective function $f(\boldsymbol{x}) = 0$ subject to the constraints $\{g_i(\boldsymbol{x}) \geq 0\}_{i=1}^m$.