Fast Generation of Nested Space-filling Latin Hypercube Sample Designs

Keith Dalbey, PhD
Sandia National Labs, Dept 1441
Optimization & Uncertainty Quantification

George N. Karystinos, PhD
Technical University of Crete, Dept of Electronic & Computer Engineering

Feb. 28 – Mar. 4, 2011

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy’s National Nuclear Security Administration under Contract DE-AC04-94AL85000.
Outline

- Sampling: Why & What’s Good?
- Sample Design Quality Metrics
- “Binning Optimality,” a New Space-filling Metric
- Latin Hypercube Sampling (LHS)
- Jittered Sampling
- Binning Optimal Symmetric Latin Hypercube Sampling (BOSLHS)
- Conclusions
- Current Work
Problem: generate a $M$ dimensional sample design with $N$ points at which to evaluate a simulator

Why sample simulator input?
- To calculate statistics of outputs with uncertain inputs
- To optimize e.g., guess several times and pick best guess
- To construct meta-models (fast surrogates for slow simulators)

What qualities do we want in a sample design?
- Design should be space-filling
- Low-dimensional projections of points should be well spaced
- Sample point locations should be uncorrelated with each other
- Regularity is bad, leads to biased results
- Nesting: want a SEQUENCE of designs that inherit all points from earlier members in the sequence
Sample Design Quality Metrics

• Lots of metrics; fortunately one of them is almost always the most important

• “Discrepancy” (some norm of difference between points per sub-volume and uniform density): lower is better
  – “Koksma-Hlawka-like inequality” bounds error in a computed mean in terms of discrepancy
  – Centered L2 Discrepancy (usually most important metric)
  – Wrap-Around L2 Discrepancy (important for periodic variables)

• Unfortunately, discrepancy is expensive (O(M N^2) ops) to calculate for designs with large numbers of points, N, so...

• Can’t guess a large number of designs & pick the best

• WARNING: Regularity is easy way to get low discrepancy
Sample Design Quality Metrics

Other “partial” metrics

• “Coverage” (fraction of hypercube's volume filled by convex hull of points, VERY expensive for even moderately high dimensions): higher coverage is better

• Condition number of sample design's correlation matrix (can be evaluated in $O(M^2N)$ ops): lower is better

• “t” quality metric when design is considered to be a tms-net (quasi-Monte Carlo; metric moderately expensive $O((m-t+1+s)Cs s b^m)$ ops where $s=M$, $b^m=N$): lower “t” is better

• NEW! degree of Binning Non-Optimality (can be evaluated in $O(N \log(N))$ time): lower is better
“Binning Optimality”
A New Space-filling Metric

A sample design is “Binning Optimal” (in base 2) if

Short answer:
Every sub-bin that should contain a point does

Long answer:
• When you recursively subdivide M-dimensional hypercube into $2^M$ disjoint congruent sub-cube bins, all bins of same generation contain same number of points
• The above must hold true until bins are so small that they each contain either 0 or 1 points

![Graph showing the distribution of points in bins. The graph has labeled axes X and Y, with points indicating bin occupancy. The graph illustrates that the points are distributed optimally, aligning with the concept of “Binning Optimality.”]
“Binning Optimality”
Can Be Evaluated in $O(N \log(N))$ Ops

- Generate bin ids as indices into a Morton space-filling curve, also known as a “Z-curve” $O(N \log(N)) + O(NM)$ work
- Quicksort bin ids $O(N \log(N))$ work
- Tally bins ids: $O(N)$ work
- A FFT of difference of sequential sorted Z-curve bin ids reveals regularity (cyclic patterns)
Latin Hypercube Sampling (LHS)

- Form of stratified random sampling that converges with fewer points than Monte Carlo Sampling
- Each column contains 1 point
- Each row contains 1 point
- Quality of design depends on pairing of dimensions used to form points (tough problem)
- Cell-centered LHS with randomly paired dimensions
  - gets 1D projections “perfect”
  - is NOT space-filling
Jittered Sampling

- Jittered Sampling = Tensor product sampling + random offset
- Better 1D projections than Tensor Product sampling
- **Worse 1D projections than LHS**
- Each cell contains a point $\Rightarrow$ **space-filling** as cell size $\rightarrow 0$

These are Binning Optimal
Binning Optimal Symmetric Latin Hypercube Sampling (BOSLHS)

- Gets 1D projections right
- Is space-filling
- Combines most of best features of LHS and Jittered sampling
- Design quality is better than regular LHS or Jittered sampling
- Is very fast: generated Nested BOSLHS $M=8$ dim, $N=2^{16}=65536$ points design in 8.21 seconds
- Currently limited to $M=2^p \leq 16$ dimensions (low degree of binning non-optimality for non integer $p$, working on extending to $M > 16$)
Nested BOSLHS Algorithm

1. Start with (lower Z half of) small BOSLHS design
2. Choose new set of bins that are maximally spaced from old bins
3. Generate a new BOSLHS by randomly filling new bins
4. Combine old & new designs, split each row/column/etc. in half, & randomly send each half 1 of duplicate coordinates
5. Repeat steps 2 through 4 as many times as desired.
Higher Dimensions Are More Complex

• Need $\log_2(N)/1$ bits to uniquely identify each 1D bin

• Binning Optimality in M-D sets first $\log_2(N)/M$ bits per dimension (BPD)

• BOSLHS matches first $\log_2(N)/M$ bits of 1D designs to M-D design; “random” matching of remaining bits (step 3 of previous slide)

• But can use Binning Optimality in subsets of dimensions to match bits $\log_2(N)/M+1 \rightarrow \log_2(N)/2$

• First cut: randomly match first $\log_2(N)/M$ BPD of M/2 2D BOSLHS designs to M-D design
Higher Dimensions Are More Complex

• Bit # \( \text{ceil}(\log_2(N)/M) \) is “tricky” when \( \log_2(N)/M \) is not integer

• Bins/Octants for that bit must be max spaced in M-D; solution is endpoints of max spaced rotated orthogonal axes (see next slide), but getting max spaced subsets of dimensions is “trickier”

• Nesting makes bit # \( \text{ceil}(\log_2(N)/M) \) “trickier”

• Ensuring symmetry makes things “trickier”
Higher Dimensions Need a Maximally Spaced List of Octants

• Generating list is simple for up to $M = 8$ dimensions. It’s difficult beyond that BUT…

• It’s similar to digital communication problems

• Collaborator, Professor George N. Karystinos of Technical University of Crete (Department of Electronic & Computer Engineering), found a group theory solution for arbitrarily large dimensions

• But… memory requirements prevent even listing the octants for $M > 32$

• Working on generating maximally spaced partial list
Results: Eyeball Metric $M = 4D$

- Plotted all 6 combinations of 2 out of $M = 4$ dimensions
- BOSLHS is visibly space-filling!
Results: Centered L2 Discrepancy (Lower is Better)

Plots are for average of 40 random designs
Results: Sobol Sequence Has Lower Discrepancy But Is Regular

Sobol Sequence $M=4 \ N=256 \ CD_2(X)=0.00997676$

Regularity in sample designs results in biased statistics
Results: What Complete Irregularity (Monte Carlo Sampling) Looks Like

Monte Carlo Sampling $M=4$ $N=256$ $CD_2(X)=0.0447803$
Results: Nested BOSLHS Is Not Regular

2D-Subset Nested BOSLHS $M=4$ $N=256/4096$ $CD_2(X)=0.0159709$
Results

• BOSLHS has low discrepancy without being regular

• BOSLHS also scores well in other metrics: it has high “coverage,” low correlations between dimensions, and a low (t,m,s)-net rating

• **VERY fast**: MATLAB generated a N=2^{16} point M=8 dimensional space-filling nested BOSLHS design in ~8.21 seconds on an Intel 2.53 GHz processor (algorithms reported in literature take “minutes” for non-nested space-filling N = 100 point designs)

• By comparison, it took ~298.2 seconds (O(N^2M) ops) to evaluate discrepancy for same design
Conclusions

• Defined new space-filling metric “Binning Optimality” that evaluates in $O(N \log(N))$ time

• Found related way to detect regularity in sample designs

• Developed fast algorithm for **Nested** Binning Optimal Symmetric Latin Hypercube Sampling (BOSLHS) that
  
  – is also Binning Optimal in some 2D subsets
  
  – combines best features of LHS & Jittered Sampling
Ongoing Work

- Current BOSLHS algorithm is space-filling in full M dimensional space, 1 dimensional projections, and some 2D subsets. Want to be space-filling in more 2D and other larger subsets of dimensions.

- Extension to larger (> 16) and arbitrary (non power of 2) numbers of dimensions.

- How well does BOSLHS do in other design quality metrics?

- Better numerical quantification of “regularity”

- Induce correlations between dimensions?
References


Results: Centered L2 Discrepancy (Lower is Better)

Plots are for average of 40 random designs
Results: Wrap Around L2 Discrepancy (Lower is Better)

Plots are for average of 40 random designs.
Results: Eyeball Metric $M=4D$

- Plotted all 6 combinations of 2 out of $M=4$ dimensions
- BOSLHS is visibly space-filling!
Results: Nested BOSLHS Is Not Regular

Nested BOSLHS $M=4$ $N=256/4096$ $CD_2(X)=0.0190745$
Compared To Original, Nested BOSLHS
Less Regular But Higher Discrepancy

BOSLHS M=4 N=256 $C_{D_2}(X)=0.0153393$
4-D Example

• Difference in 4 dimensions is in choosing maximally spaced bins

• In 2D, only $2^2=4$ sub-bins per level, the $2*2=4$ end points of 1 “orientation” (rotated set of orthogonal axes)
  – If 1 point in bin, new sub-bin is opposite old one
  – If 2 points (1 axis), 2 new sub-bins are other axis
  – Then go 1 bin deeper

• In 4D, $2^4=16$ sub-bins per level, 2 orientations with $2*4=8$ bins each
  – After first axis, randomly select order of other axes in same orientation
  – Then choose other orientation
  – Then go 1 bin deeper

Nested BOSLHS N=64/4096 M=4 CD₂(X)=0.0411594
### Results: Coverage
(higher is better)

"Coverage" for $M = 4$ Dimensions: Average of 40 runs

<table>
<thead>
<tr>
<th>N</th>
<th>Binning Optimal Symmetric LHS</th>
<th>Cell Centered Random LHS</th>
<th>Monte Carlo Sampling</th>
<th>Jittered Sampling</th>
<th>Tensor Product Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.0717773</td>
<td>0.027062</td>
<td>0.0178996</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.135135</td>
<td>0.104126</td>
<td>0.0916156</td>
<td>0.128427</td>
<td>0.0625</td>
</tr>
<tr>
<td>32</td>
<td>0.285717</td>
<td>0.233105</td>
<td>0.219465</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>0.417035</td>
<td>0.372359</td>
<td>0.361626</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>0.56022</td>
<td>0.522201</td>
<td>0.511982</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>0.678416</td>
<td>0.647304</td>
<td>0.645049</td>
<td>0.667668</td>
<td>0.316406</td>
</tr>
<tr>
<td>512</td>
<td>0.773748</td>
<td>0.754804</td>
<td>0.749725</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>0.843177</td>
<td>0.832896</td>
<td>0.831007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>0.896093</td>
<td>0.890245</td>
<td>0.886593</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>0.932229</td>
<td>0.928693</td>
<td>0.927748</td>
<td>0.929509</td>
<td>0.586182</td>
</tr>
<tr>
<td>8192</td>
<td>0.956723</td>
<td>0.954248</td>
<td>0.953466</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>0.97319</td>
<td>0.97129</td>
<td>0.971217</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>0.983415</td>
<td>0.982499</td>
<td>0.982312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65536</td>
<td>0.989815</td>
<td>0.989387</td>
<td>0.98926</td>
<td>0.98965</td>
<td>0.772476</td>
</tr>
</tbody>
</table>
## Results: Condition # of Correlation Matrix (lower is better)

Condition Number of the Correlation Matrix for $M = 4$ Dimensions: Average of 40 runs

<table>
<thead>
<tr>
<th>N</th>
<th>Binning Optimal Symmetric LHS</th>
<th>Cell Centered Random LHS</th>
<th>Monte Carlo Sampling</th>
<th>Jittered Sampling</th>
<th>Tensor Product Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>14.6273</td>
<td>8.23719</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>3.2505</td>
<td>4.14988</td>
<td>3.75258</td>
<td>2.39394</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>1.49974</td>
<td>2.27709</td>
<td>2.15406</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>1.37672</td>
<td>1.76306</td>
<td>1.82367</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>1.2064</td>
<td>1.4508</td>
<td>1.49656</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>1.11022</td>
<td>1.32572</td>
<td>1.33407</td>
<td>1.10916</td>
<td>1</td>
</tr>
<tr>
<td>512</td>
<td>1.05589</td>
<td>1.21341</td>
<td>1.2108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>1.0368</td>
<td>1.1546</td>
<td>1.14725</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>1.02121</td>
<td>1.09974</td>
<td>1.09939</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>1.01246</td>
<td>1.07576</td>
<td>1.07075</td>
<td>1.01254</td>
<td>1</td>
</tr>
<tr>
<td>8192</td>
<td>1.00717</td>
<td>1.04643</td>
<td>1.04922</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>1.00403</td>
<td>1.03608</td>
<td>1.03365</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>1.0027</td>
<td>1.02297</td>
<td>1.02461</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65536</td>
<td>1.00166</td>
<td>1.01872</td>
<td>1.01742</td>
<td>1.00145</td>
<td>1</td>
</tr>
</tbody>
</table>
Results: \((t,m,s)\)-net, “\(t\)” quality metric (lower is better)

\((t,m,s)\)-net Rating for \(M = 4\) Dimensions: Average of 40 runs

<table>
<thead>
<tr>
<th>(N)</th>
<th>Binning Optimal Symmetric LHS</th>
<th>Cell Centered Random LHS</th>
<th>Monte Carlo Sampling</th>
<th>Jittered Sampling</th>
<th>Tensor Product Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>(1, 3, 4)</td>
<td>(2, 3, 4)</td>
<td>(3, 3, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>(2, 4, 4)</td>
<td>(3, 4, 4)</td>
<td>(4, 4, 4)</td>
<td>(3, 4, 4)</td>
<td>(3, 4, 4)</td>
</tr>
<tr>
<td>32</td>
<td>(2, 5, 4)</td>
<td>(4, 5, 4)</td>
<td>(5, 5, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>(3, 6, 4)</td>
<td>(5, 6, 4)</td>
<td>(6, 6, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>(4, 7, 4)</td>
<td>(6, 7, 4)</td>
<td>(7, 7, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>(5, 8, 4)</td>
<td>(7, 8, 4)</td>
<td>(8, 8, 4)</td>
<td>(6, 8, 4)</td>
<td>(6, 8, 4)</td>
</tr>
<tr>
<td>512</td>
<td>(5, 9, 4)</td>
<td>(8, 9, 4)</td>
<td>(9, 9, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>(6, 10, 4)</td>
<td>(9, 10, 4)</td>
<td>(10, 10, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>(7, 11, 4)</td>
<td>(10, 11, 4)</td>
<td>(11, 11, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>(8, 12, 4)</td>
<td>(11, 12, 4)</td>
<td>(12, 12, 4)</td>
<td>(9, 12, 4)</td>
<td>(9, 12, 4)</td>
</tr>
<tr>
<td>8192</td>
<td>(8, 13, 4)</td>
<td>(12, 13, 4)</td>
<td>(13, 13, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>(9, 14, 4)</td>
<td>(13, 14, 4)</td>
<td>(14, 14, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>(10, 15, 4)</td>
<td>(14, 15, 4)</td>
<td>(15, 15, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65536</td>
<td>(11, 16, 4)</td>
<td>(15, 16, 4)</td>
<td>(16, 16, 4)</td>
<td>(12, 16, 4)</td>
<td>(12, 16, 4)</td>
</tr>
</tbody>
</table>
O(N log(N)) BOSLHS Algorithm

1. Start with \( n = 2M \) points that are well distributed in \((0, 1)^M\).

2. Select \( n/2 \) of the coordinates in each dimension other than the first to negate in such a way as to obtain \( n \) points that are well distributed in \((0, 1) \otimes (-1, 1)^{M-1}\).

3. Reflect the current \( n \) points through the origin to create \( n \) additional mirror points; this ensures that the design is symmetric.

4. Translate the \( 2n \) points from \((-1, 1)^M\) to \((0, 2)^M\), scale them to \((0, 1)^M\), and then set \( n = 2n \).

5. Repeat steps 2 through 4 until the desired number of points has been obtained, i.e. until \( n = N \).
O(N log(N)) BOSLHS Algorithm
O(N log(N)) BOSLHS Algorithm

The tough part is step 2

Select n/2 of the coordinates in each dimension other than the first to negate in such a way as to obtain n points that are well distributed in \((0, 1) \otimes (-1, 1)^{M-1}\).  

The easy (fast) answer is to recast the problem...
- Don't try change signs of dimensions individually
- Send nearby points to octants that are far apart

The Z-order quicksort will put nearby points in sequential order in \(O(N \log(N))\) ops

We just need a listing of octants in maximally spaced order