# Fair-Coverage Monte Carlo Generation of Monotonic Functions within Envelopes

**T. K. Satish Kumar**
Knowledge Systems Laboratory
Stanford University
tksk@ksl.stanford.edu

**Abstract**

The Monte Carlo technique is an alternative to semi-quantitative simulation of incompletely known differential equations. Here, a large number of ODEs matching the given QDE are randomly generated making use of the available semi-quantitative information. Each ODE is then numerically integrated and conclusions are drawn from the family of results produced. Maintaining fair-coverage (or true randomness) is important for producing unbiased conclusions when we randomly generate monotonic functions matching the original incomplete specifications in the numerical integration phase of these techniques. Earlier attempts did not ensure fair-coverage in a theoretical sense. We provide a hierarchy of algorithms (exponential in number) to do this when envelopes for the monotonic functions are specified. These algorithms can be ranked according to increasing degrees of nearness to fair-coverage and decreasing computational tractability. The appropriate algorithm can then be selected from this hierarchy to suit the requirements of the problem.

## Introduction

The Monte Carlo technique is an alternative to semi-quantitative simulation of incompletely known differential equations, finding its use in a number of real life applications. Here, a large number of ODEs matching the given QDE are randomly generated making use of the available semi-quantitative information. Each ODE is then numerically integrated and conclusions are drawn from the family of results produced. The greater the number of ODEs generated, the more confidence one has in not missing a rare type of behavior.

One other important factor on which the usefulness of the Monte Carlo technique depends is whether the underlying space is covered fairly or not. Maintaining fair-coverage (or true randomness) is important for producing unbiased conclusions when we randomly generate monotonic functions matching the original incomplete specifications in the numerical integration phase of these techniques.

In this paper, we provide a hierarchy of algorithms (exponential in number) to do this when envelopes for the monotonic functions are specified. The algorithms can be made to work under two different frameworks - the *grid* representation and the *point-list* representation. These algorithms can also be ranked according to increasing degrees of nearness to fair-coverage and decreasing computational tractability. The appropriate algorithm can then be selected from this hierarchy to suit the requirements of the problem. This paper also shows how some clever mechanisms can be used to achieve a polynomial space complexity for all these algorithms as opposed to an exponential amount of space that would be otherwise required in their naive implementation.

## Previous Approaches

In (Gazi et. al. 1997), the following algorithm was used for randomly generating a monotonic increasing function y=f(x) such that $f_l(x) \leq f(x) \leq f_u(x)$, where $f_l(x)$ and $f_u(x)$ are the given quantitative envelope functions.

Algorithm: GENERATE-FUNCTION
1. Let G (the number of Grid points to be selected in x) be a sufficiently large integer.
2. Let $x_1$ and $x_G$ be the smallest and greatest possible $x$ values to be considered respectively. The interval $[x_1, x_G]$ will be divided by the grid points to G-1 intervals of equal length.
3. Choose $y_1$ randomly in $[f_l(x_1), f_u(x_1)]$ from some distribution.
4. Choose $y_G$ randomly in $[max(f_l(x_G), y_1), f_u(x_G)]$ from some distribution.
5. Do the following for all available $(x_i, y_i)(x_j, y_j)$ pairs with remaining untouched grid points between them:
   5.1 Select a midpoint $x_k$ in this interval
   5.2 Randomly choose $y_k$ in
   $[max(f_l(x_k), y_i), min(f_u(x_k), y_j)]$
6. Connect the points $(x_k, y_k)$, k = 1 ... G with straight lines.
(Decreasing functions are treated analogously.)

### Problems

The probability distribution to be used in step 5.2 of the above algorithm to choose a $y_k$ for a given $x_k$ is of crucial importance from the point of view of fair coverage. For the case of a uniform distribution, the functions generated would be unfairly attracted towards the upper envelope. Trying to favor points near the lower envelope depending upon a tuned parameter (Gazi et. al. 1997) poses many problems apart from the fact that this is not the theoretically right thing

to do. Other problems related to this approach are identified in (Say 2000). Among them is the observation that there is an unjustified shape restriction on the generated function when trying to impose differentiability of the function for QSIM. The method may also waste resources to generate parts of the function that will turn out to be unnecessary in the numerical integration phase. This is because of the *grid* representation that is used. The *point-list* representation described in the next section solves this particular problem.

### The Point-List Representation

One proposed solution (Say 2000) to some of the problems described in the previous subsection is based on generating the monotonic functions point by point as the need arises during integration, and representing them as lists of these points, without committing to any shape about the intervals in between. Essentially, we want to do the following whenever f(x) is supposed to be evaluated for x=r:

IF the point (r,y) is a member of f's point-list
THEN RETURN y
ELSE IF r > rightmost(f) OR r < leftmost(f)
THEN EXTRAPOLATE ELSE INTERPOLATE

While the *point-list* representation solves the problem of differentiability and not wasting resources for x-coordinates that are not used, it still does not say anything about producing monotonic functions from a fair distribution. In (Say 2000), *angles* were being used to favor points closer to the lower envelope to offset the unfair attraction to the upper envelope.

### Problems with Using Angles

The following problems are still associated with using angles to produce random monotonic functions between the specified envelopes (Say 2000). (1) Although using angles in the EXTRAPOLATE function works against the unfair attraction to the upper envelope by biasing points closer to the lower envelope, this is still just an ad-hoc solution and does not ensure *true randomness*. (2) The INTERPOLATE function (Gazi et. al. 1997) by itself, is not justified to be theoretically correct because the choice for the middle point can be biased either to the upper or the lower envelope depending upon the particular case. (3) In the case where $y$ intervals for two consecutive x-values do not overlap, a uniform distribution (over each of them independently) provides *true randomness*; but the usage of angles incorrectly favors the points closer to the lower envelope.

## A Proposed Solution

Suppose that we want to find a random $x$ and $y$ under the constraints that: $x_1 < x < x_2$, $y_1 < y < y_2$ and $x < y$. Quite naturally, first randomly generating $x$ between $x_1$ and $x_2$ and then generating $y$ according to the additional constraints imposed, does not serve the purpose (the result is attracted more towards $y_2$). This is because not all constraints are being considered at a single time. Figure 1 shows that region *r* is more dense than region *s* if the x-coordinate is chosen independently first. The mistake committed by all

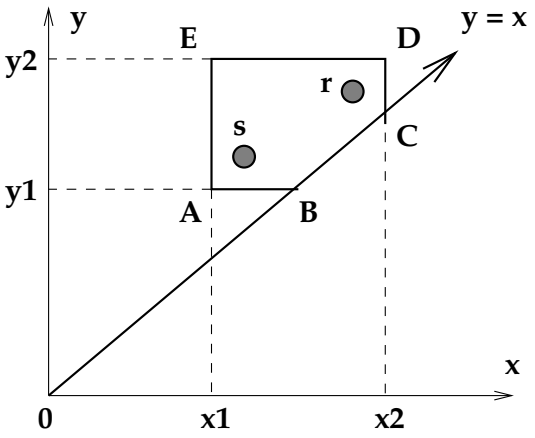



Figure 1: Illustrates a convex solution space. Note that region $r$ is more dense than region $s$ if the x-coordinate is chosen *independently* first.

earlier attempts was that constraints were being procrastinated with each new dimension, thereby leading to an unfair distribution in the final outcome. The only way to get a *truly random* generation is by considering all the constraints at once.

First, consider the solution space from which the random sample has to be drawn (see Figure 1). An important property of this space is its *convexity*. We know that any point inside the convex area can be expressed as a linear combination of its vertices. Therefore, to generate a random $x$ and $y$ satisfying all the constraints, we randomly generate 5 numbers (between 0 and 1) ($\lambda_1$, $\lambda_2$, $\lambda_3$, $\lambda_4$, $\lambda_5$). Now, the random $(x, y) = \frac{\lambda_1 A + \lambda_2 B + \lambda_3 C + \lambda_4 D + \lambda_5 E}{\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5}$. The trick here is that although in reality there was another constraint that the random weighting factors for A, B, C, D and E (the $\lambda' s$) must sum to one, we satisfied it by *normalization*; still maintaining *uniform randomness* [1].

## A Generalized Algorithm

We introduce a generic algorithm that works under both the frameworks - the *grid* representation and *the point-list* representation. In both cases, complexities are in terms of the number of x-coordinate lines (denoted by the variable $n$). A further discussion and comparison is made later in the paper. Let us now formally define some notions (see Figure 2) related to the geometry of the envelopes and the coordinate lines to facilitate explanation of the ideas in the algorithm.

**Definition** A *coordinate line* is a vertical line drawn at a particular x-coordinate.

**Definition** The *lower* and *upper caps* on a *coordinate line* are the points where the coordinate line cuts the lower and upper envelopes respectively.

[1] Another observation to make here is that the number of random numbers required is more than the number of coordinates viz $x$ and $y$. This corresponds to the Lagrange multipliers for optimization under constraints (which is equivalent to optimization in a higher dimension)

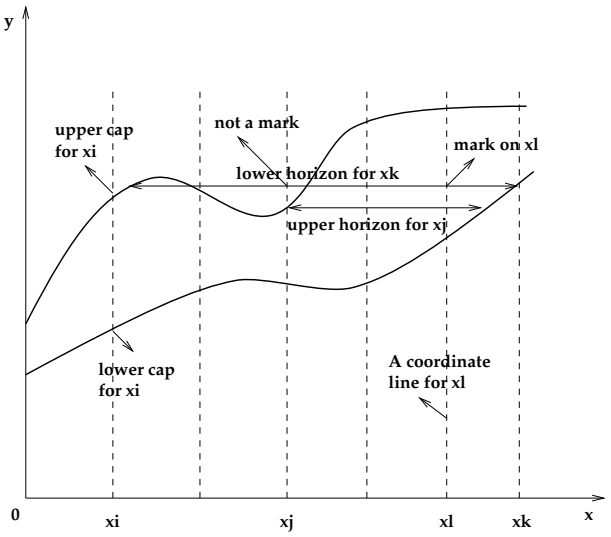



Figure 2: Diagrammatically illustrates the definitions of various related notions about the geometry of the envelopes.

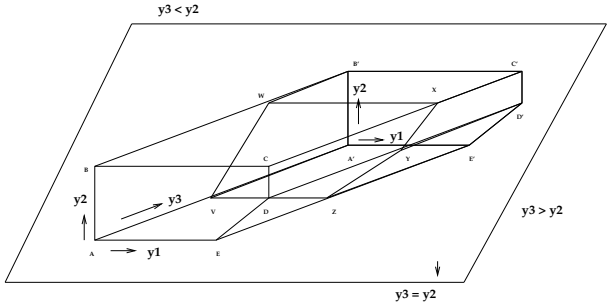



Figure 3: Roughly illustrates how the extension of the solution space is done for each new dimension.

**Definition** The *upper horizon* at a coordinate line is a maximal horizontal line-segment with one end point on the lower envelope and the other (left-side) end point as the upper cap on the coordinate line.
**Definition** The *lower horizon* at a coordinate line is a maximal horizontal line-segment with one end point on the upper envelope and the other (right-side) end point as the lower cap on the coordinate line.
**Definition** A *mark* on a coordinate line is an intersection point of that coordinate line with any of the horizons such that the intersection point itself lies within the caps on the coordinate line. (Note that this definition includes the caps on the coordinate line themselves, since they are the end points of the lower and upper horizons for that coordinate line.)

## Conceptualization of the Solution Space

First of all, suppose that we want to compute the vertices of the n-dimensional convex solution hyperspace. Conceptually, the computation can be done in two phases (see Figure 3):

**Duplication Phase** Consider the x-values in increasing order. For each new $x_k$, duplicate the current (k-1)-dimensional region at $(y_k = lb(y_k))$ and $(y_k = ub(y_k))$. Here, $lb$ and $ub$ are the bounding lower and upper envelope curves given. In the new k-dimensional space thus obtained, there is a line connecting corresponding points in the two copies of the (k-1)-dimensional solution space that was existing before $x_k$ was considered (one at each of the bounding values for $x_k$).

**Intersection Phase** We now need to consider the extra constraint that the random value generated against $x_k$ (i.e. $y_k$) must be greater than that generated against $x_{k-1}$ (i.e. $y_{k-1}$) to maintain *monotonicity*. The required solution space is therefore that part of the k-dimensional convex space (constructed from the *duplication phase*) which lies above the hyperplane $y_k = y_{k-1}$. The intersections of this hyperplane can be computed easily by using the notion of a *consistent* and *inconsistent point*.

**Definition** A point is said to be *consistent* if its coordinates are in a monotonically increasing order (i.e. $y_{1_p} < y_{2_p} < ..y_{k_p}$). Otherwise, it is said to be *inconsistent*.

The procedure, conceptually, is therefore simply to cut lines joining consistent and inconsistent points by the hyper plane $y_k = y_{k-1}$ to find other consistent vertices of the solution space.

Algorithm (conceptual): SOLUTION-SPACE-VERTICES
1. Set SSV = $\phi$. (consider $x_1..x_n$ in increasing order)
2. FOR $i$ = 1:$n$ DO
   2.1 [Duplication Phase]
   SSV = SSV.lb($y_i$) $\cup$ SSV.ub($y_i$)
   2.2 [Intersection Phase]
   FOR every *inconsistent* P in SSV:
      2.2.1 Let EQ be the set of all Q s.t. Q is a consistent neighbor of P.
      2.2.2 Let ER be the set of all R such that R = point on line segment PQ lying on plane $y_i = y_{i-1}$. (R is consistent).
      2.2.3 SSV = SSV - P $\cup$ ER
3. SSV is the required set of points of the convex n-dimensional solution hyperspace.

In the above (conceptual) algorithm, we observe that the number of points in the solution space is unwieldy [2]. We would need an exponential amount of space to store all the vertices and an exponential amount of time to generate all the $\lambda$s and compute the random linear combination of these vertices. We solve each of these two problems (i.e. achieve tractable space and time complexities) by relating the solution space to geometric properties of the envelopes explained earlier.

## Line Segment Encoding of Solution Space

We now prove a series of properties and culminate by proving that the solution vertices can be encoded in a polynomial

[2] Of the order of $2^n$. Proof omitted in this paper

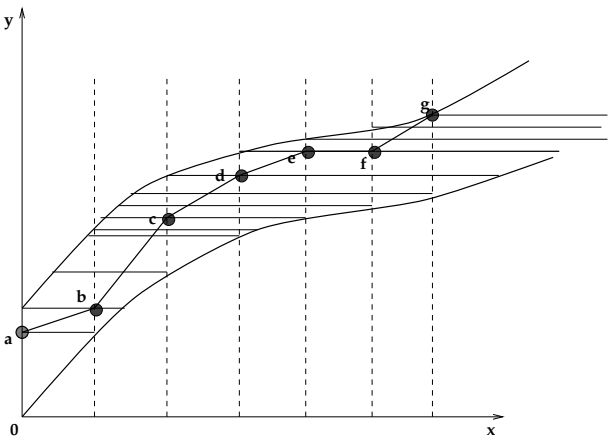



Figure 4: Shows the *line-segment encoding* of the solution space vertices.

space of $O(n^2)$ by just maintaining the *marks* on each coordinate line.

**Definition** The *line-segment representation* of a solution vertex is a continuous set of line segments (joined end to end) (*abcdefg* in Figure 4) where the $i^{th}$ coordinate value of the solution vertex corresponds to the y-value of the mark (selected to represent the vertex) on the $i^{th}$ coordinate line.

**Lemma** A vertex is *inconsistent* if and only if its line-segment representation has a piece with a negative slope.

**Lemma** The *duplication phase* corresponds to joining all marks on coordinate line $x_{k-1}$ with the caps on the coordinate line for $x_k$.

**Lemma** The *intersection phase* corresponds to replacing the inconsistent line-segments by a pair of horizontal line-segments starting from each of the end-points as shown in Figure 5.

**Lemma** The set of solution vertices produced by the conceptual algorithm is exactly the same as those represented by a consistent sequence of marks (i.e. consistent line-segment representations).

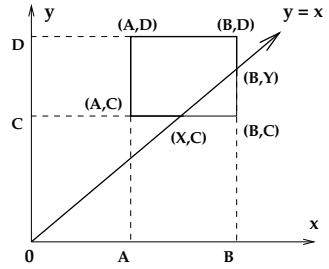
This is because the horizontal lines drawn in the intersection phase are exactly the horizons drawn to compute the marks.

## Data Structure

We use a 2D array $A$ as the underlying data structure in our algorithm (see Figure 6). $A_{ij}$ gives the y-value of the $j^{th}$ mark on coordinate line $i$. Marks on each coordinate line are ranked (in an increasing order) according to their y-values. $A_{ij}$ also stores two other fields *count* and *p-value* along with some indexing information [3]. The use of these fields will be described later in the paper. We also ensure that all cells are initialized with a special symbol in them so that any unused cells (arising by virtue of coordinate lines having fewer marks on them than the maximum possible - viz $2n$) can be identified.

**Space Complexity** There are $n$ row indices each corresponding to a coordinate line and against them there are a

[3]But all cells are of the same size, facilitating pointer arithmetic



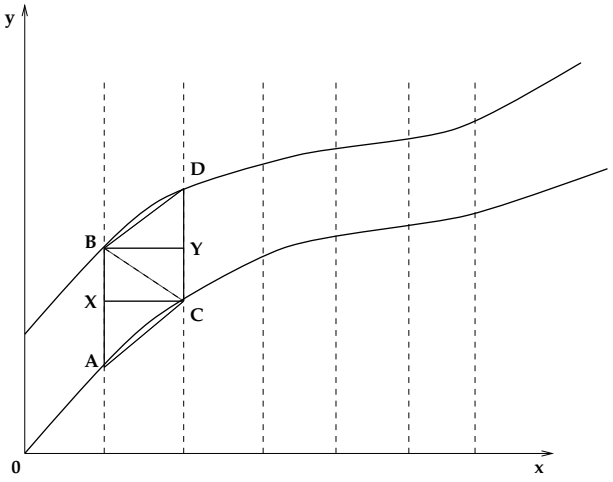




Figure 5: Illustrates how negative slopes are replaced with *horizons*. BC being inconsistent, is replaced with XC and BY.

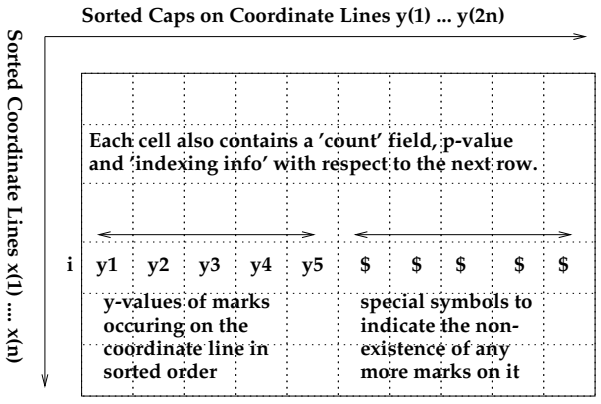



Figure 6: The underlying data structure for the algorithm.

maximum of $2n$ y-values stored. This is because all marks have the same y-value as one of the caps and there are 2n caps all together. The space complexity is therefore $O(n^2)$.

**Time Complexity** The time complexity involved in building and maintaining the data structure depends on the representation we use. We discuss the time complexities under the *grid* and *point-list* representations separately below.

In the *grid* approach, we need to consider the caps in sorted order and add them to each partially built row if it lies within the caps for that coordinate line. Note that this comparison is not done against all coordinate lines. The *lower caps* are compared to coordinate lines occurring *before* the coordinate line to which the cap belongs and the *upper caps* are compared to coordinate lines occurring *after* the coordinate line to which the cap belongs. This is in accordance with how the horizons are built. This procedure clearly takes $O(n^2)$ time and the other complexities related to sorting the caps and initialization of the data structure are subsumed by it.

In the *point-list* approach however, coordinate lines can occur in a random order and it clearly takes $O(n^2)$ time to update the data structure consistently for each new line added into consideration (even if one rebuilt the whole data structure upon every new addition). All together, the time complexity comes to $O(n^3)$ where $n$ is the number of coordinate lines present so far in the point-list representation. Again, this subsumes other complexities related to sorting and initialization.

It can be observed therefore, that the *point-list* representation has the advantage of using fewer coordinate lines but requires $O(n^3)$ time to maintain a consistent data structure. The *grid* approach has the advantage of using an $O(n^2)$ time algorithm to build the data structure once and for all, but may be using many more unwarranted coordinate lines to work with.

**Consistent Solution Space Vertices** Let us consider the task of generating consistent sequences of marks from the line-segment encoding of solution space vertices. Suppose we imagine a graph in which all the marks are nodes and that there is an edge between mark $i$ and mark $j$ if they occur on consecutive coordinate lines and $j$ has a higher y-value than $i$. Suppose we need to pick consistent line-segment representations randomly; then we need to do the following to ensure uniformity. We make use of the *p-value* at each mark. The *p-value* is calculated as the sum of the *p-values* of its neighbors at the next level, with the *p-value* for the last level nodes being set to 1. The algorithm would now simply be to choose at each level a node amongst a candidate set of nodes at the same level with a probability proportional to its *p-value* and then proceed to its neighbors at the next level as being candidates for the next selection and so forth (see Figure 7).

Algorithm: RANDOM-PATH
1. *CandSet* = all Level-1 nodes
   (marks on the first coordinate line).
2. *Random-Path* = $\{\}$.
3. *rand* = random number in [0,$\sum_{CandSet}$ *p-values*].

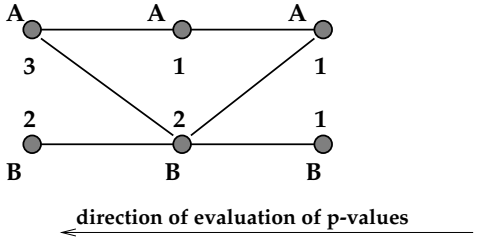



Figure 7: Choosing paths *randomly* and *fairly*. Note that the probability of choosing any path = 1/5. For example, P(ABB) = 3/5.2/3.1/2 = 1/5.

4. *Current* = binary search for *rand* in
   *count field*s of *CandSet*.
5. Add *Current* as the next node in the *Random-Path*.
6. FOR $k$ = 2 to $n$ DO
   5.1 *CandSet* = neighbors of *Current* in level $k$.
   5.2 *rand* = random number in [0, *p-value* of *Current*].
   5.3 *Current* = binary search for *rand* in
   *count field*s of *CandSet*.
   5.4 Add *Current* as the next node in the *Random-Path*.
7. Return *Random-Path*.

**Computing Count, P-value and Indexing Information** We also need to address the issue of jumping to a random neighbor in $O(logn)$ time if we want to have $O(nlogn)$ complexity for the generation of a single random solution space vertex. This can be achieved if we can perform a binary search on the neighbors of a point having generated a random number in the interval [0, *p-value*]. Being able to perform binary search requires that we maintain in each cell a *count field* equal to the sum of the *p-values* of all marks occurring above it on the same coordinate line. Notice that since the sorted order of marks is already present in the way we build our data structure, we only need to maintain indexing information of how each cell maps into the following row in which it is present. Indexing information can be computed by walking down a pair of consecutive rows simultaneously and reusing computations made. This takes $O(n)$ time for a pair of rows and needs to be done for (n-1) pairs. The overall time complexity remains $O(n^2)$ in the case of the *grid* representation and $O(n^3)$ in the case of the *point-list* representation. It is easy to observe that the *count* and *p-values* can also be computed while building up the indexing information.

## Degrees of Uniform Randomness: Exploiting Convexity of the Solution Space

We now present two theorems [4] which apply to *convex hyper-spaces* and see how we can import them into our mainstream ideas.

**Theorem 1** The set of points which can be chosen by a linear combination of the vertices of a n-dimensional convex solution space is the same as that which can be chosen by

[4] Formal proofs are omitted in this paper

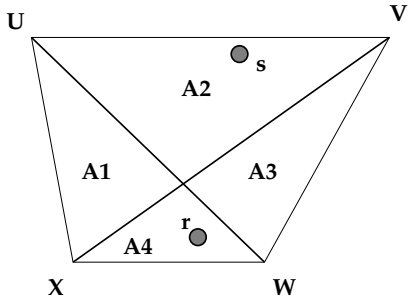



Figure 8: A simple example illustrating the truth of the two theorems. Region $r$ is more dense than region $s$ when $k$=3; but they are equally dense when k=4.

first selecting $k$ ($n < k$) of these vertices and then taking their linear combination.

**Theorem 2** For random choices of the $k$ vertices, the distribution in the probabilities with which different points can be selected increasingly tends towards *uniformity* with increasing $k$ (i.e. *entropy* increases with increasing $k$).

This is because, every $k_2$ selection of nodes can be further subdivided by a $k_1$ selection of nodes among them ($k_2 > k_1$). Figure 8 shows an example for a 2D space. Regions $r$ and $s$ are not equally dense when k=3. Letting k=4 however, makes the whole solution space uniform.

## A Hierarchy of Algorithms for Achieving Increasing Degrees of Uniform Randomness

The random monotonic functions can now be generated by enumerating all possible consistent sequences of marks (representing solution-space vertices) on the coordinate lines, and taking a random normalized linear combination of them. This process however, may be a costly affair since there are an exponential number (denoted by $m$) of such consistent sequences of marks. We present below a hierarchy of algorithms approximating *uniform randomness* to increasing degrees of accuracy.

Algorithm: MONTE-CARLO(k) [k > n]
1. Randomly choose $k$ consistent sequences of marks.
2. Randomly generate $\lambda_1$ .. $\lambda_k$ in the interval [0,1].
3. Use the $\lambda$s generated in (2) to obtain a random normalized linear combination of the sequences chosen in (1).

**Observation** MONTE-CARLO(k) runs in time $(knlogn)$. This is because generating a single solution-space vertex takes $O(nlogn)$ time.

**Observation** Theorem 1 of the previous section ensures that all possible monotonic functions between the envelopes are covered by MONTE-CARLO(k) and, moreover, exactly these are covered.

**Observation** Theorem 2 of the previous section ensures that for all $i, j \in \{n+1...m\}$ and $j > i$, MONTE-CARLO(j) has a better degree of *uniform randomness* than MONTE-CARLO(i).

**Observation** In the sequence of algorithms MONTE-CARLO(k), $k$ ranges from a linear function of $n$ (viz $n+1$) to an exponential function of $n$ (viz $m$). This provides a very fine gradation for trading off time versus accuracy and it is our conjecture that the requirements of most problem domains can be met with some polynomial function of $n$.

**Observation** We have not addressed the possibility of duplicate solution-vertices being chosen in step (1). Although we can try and detect them, we choose not to do so since it happens with an exponentially small probability for small (polynomial) values of k.

**Observation** MONTE-CARLO(m) actually achieves the theoretically correct way of ensuring *uniform randomness*. However, instead of producing $m$ randomly chosen solution-vertices like in step (1), one can actually enumerate all the vertices by performing a DFS on the graph encoded in the underlying data structure.

## Discussion and Summary

In this paper, we addressed the problem of generating random monotonic functions within specified envelopes. Firstly, we proposed the theoretically correct way of ensuring fair-coverage. Secondly, we provided a clever way to represent an exponential number of vertices of the solution hyper-space in $O(n^2)$ space. This was done by relating the geometry of the envelopes to the solution space we had to work with. Thirdly, we provided a hierarchy of algorithms (an exponential number of them) that exhibit a very fine gradation for trading off time versus accuracy (in ensuring fair-coverage). It is our conjecture that the requirements of most problem domains can be met with MONTE-CARLO(k) where $k$ is some polynomial function of $n$. This would also imply a polynomial time complexity for the chosen algorithm. It is important to note that we have exploited the *convexity* of the solution space in many ways: to define what a theoretically correct solution is, to reduce the space requirements, and finally to provide the hierarchy of algorithms.

## References

[1] Brajnik, G. 1997. Statistical Properties of Qualitative Behaviors. In *Proc. Eleventh Intl. Workshop on Qualitative Reasoning*, Cortona, Italy. 233-240.

[2] Gazi, E., Seider W. D., and Ungar, L. H. 1997. A Nonparametric Monte Carlo Technique for Controller Verification. *Automatica* 33: 901-906.

[3] Kalos, M. H., and Whitlock, P. A. 1986. *Monte Carlo Methods* Vol. 1. New York, NY: John Wiley & Sons.

[4] Kuipers, B. J. 1994. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Cambridge, Mass.: The MIT Press.

[5] Ortega, J. A., Gasca R. M., and Toro, M. 1999. A Semi-Quantitative Methodology for Reasoning about Dynamic Systems. In *Proc. Thirteenth Intl. Workshop on Qualitative Reasoning*, Loch Awe, Scotland. 169-177.

[6] Say, A. C. 2000. A New Technique for Monte Carlo Generation of Monotonic Functions. In *Proc. Fourteenth Intl. Workshop on Qualitative Reasoning*, Morelia, Mexico.