ConvolutionalFixedSum: Uniformly Generating Random Values with a Fixed Sum Subject to Arbitrary Constraints

David Griffin
University of York, UK
david.griffin@york.ac.uk

Robert I. Davis
University of York, UK
rob.davis@york.ac.uk

***Abstract*—This paper addresses the problem of uniform random generation of vectors of values with a fixed sum, subject to upper and lower constraints on the individual component values. Solutions to this problem are used extensively in the generation of tasksets, specifically task utilization values, in support of the performance assessment of schedulability tests for real-time systems. This paper introduces a general-purpose solution in the form of an Inverse Volume Ratio Sampling method that is applicable provided that it is possible to determine the ratio of the volume below a given hyperplane to the total volume of the valid region in n-dimensional space, as demarcated by the constraints and the fixed sum. An efficient approach is derived for volume calculation using numerical convolution, thus instantiating the *ConvolutionalFixedSum* algorithm, which provides a user-specified level of precision, while scaling at $O(n^3 \log(n))$. A stringent uniformity test is developed, called the slices test, which is able to fully explore the extent of the valid region in each of the n dimensions. The slices test reveals that while the outputs of *UUnifast* and *ConvolutionalFixedSum* form uniform distributions, in some cases the outputs of prior state-of-the-art algorithms do not.**

I. Introduction

Assessing the performance of schedulability tests in real-time systems requires the use of tasksets with uniformly generated random utilization values that sum to a fixed value, i.e., the total utilization. Without this property of uniformity, bias can occur in the testing process [3], meaning that the results can become invalid and potentially misleading. However, the problem of creating such uniformly generated random utilization values is non-trivial, and several intuitive but naïve approaches can lead to non-uniform distributions [3], [4]. The first commonly used algorithm to address these issues in real-time systems was *UUnifast* [3], [4], derived by Bini and Buttazzo in 2004. *UUnifast* solves the problem for single processor systems, where the task utilization values generated are between 0 and 1, and the total utilization is no greater than 1. Hence, there are no constraints on the individual utilization values, except for the fixed sum itself.

The problem of creating uniformly generated random values with a fixed sum becomes harder as more constraints are added. Generating such values to support assessment of schedulability analyses for multiprocessor systems requires that the utilization of each task is constrained, between 0 and 1, so that the task fits on a single processor; however, unlike in the single processor case, the total utilization can greatly exceed this value. The *UUnifast-Discard* algorithm [9], [10], developed by Davis and Burns in 2009, addressed this problem via a simple extension to earlier work. *UUnifast-Discard* has the drawback that it is not fully scalable, in particular it is not viable when the average number of tasks per processor is low. This issue was subsequently addressed in 2009, when Emberson et al. identified the *RandFixedSum* algorithm [11], originally developed in 2006 as a MATLAB routine by Stafford [35]. Nevertheless, *RandFixedSum* is unable to deal with individual constraints applied to the utilization of each task, such as those needed to support performance assessment of schedulability analyses for mixed-criticality systems [5].

The more general problem, with both upper and lower constraints on individual task utilizations, was first addressed by the Dirichlet Rescale Algorithm (DRS) [14], derived by Griffin et al. in 2020. Unfortunately, as subsequently shown in the evaluation in Section VI, the outputs of the *DRS* algorithm can form a non-uniform distribution in the case where one constraint is sufficiently small. The *DRS* algorithm also has high complexity and is thus expensive to compute, compared to *RandFixedSum* and *UUnifast*.

In this paper, we introduce the Inverse Volume Ratio Sampling (IVoRS) method for uniform random sampling from a constrained region of n-dimensional space. This general-purpose method is applicable provided that it is possible to determine the ratio of the volume below a given hyperplane to the total volume of the valid region in n-dimensional space, as demarcated by the constraints. Root-finding algorithms such as Interpolate-Truncate-Project (ITP) [29] can invert this calculation, providing a way to sample via inverse transform sampling To illustrate IVoRS, we show that the *UUnifast* algorithm is a specific case of the IVoRS method.

To provide uniform sampling from an n-dimensional valid region demarcated by upper and lower constraints, we introduce the *ConvolutionalFixedSum* algorithm, which utilizes the IVoRS method in conjunction with a convolution-based volume calculation and ITP. Two implementations of *ConvolutionalFixedSum* are provided: an analytical method, which scales at $O(2^n)$, and a numerical approximation, which provides uniformity to a user-specified level of precision, and scales at $O(n^3 \log(n))$.

Alternative approaches that were briefly considered include Gibbs sampling [13], which is an application of the Metropolis-Hastings algorithm [17]. Gibbs sampling is effective if the conditional probability distribution is fast to sample from. However, for the problem considered, this distribution takes exponential time to calculate, and the valid region is a complex shape; an n-dimensional hull of up to 2^n points. Further, Gibbs sampling uses a 'warm-up' period to converge to the target distribution. These properties render Gibbs sampling inefficient in this case. In addition, the output from these methods is auto-correlated and therefore not necessarily appropriate in this context.

To verify that the outputs from the *ConvolutionalFixedSum* algorithm form a uniform distribution, we developed a highly sensitive uniformity test, called the slices test. The slices test divides the valid region into a number of slices of equal volume, and compares the number of points generated in each of these slices to the expected number via a χ^2 test [34]. As the slices span the full volume of the valid region, the slices test is sensitive to non-uniformity wherever it occurs. The slices test was used to evaluate both analytical and numerical *ConvolutionalFixedSum* algorithms, with both passing a stringent array of tests. These tests were also applied to the *UUnifast*, *RandFixedSum*, and *DRS* algorithms. The outputs of *UUnifast* and *RandFixedSum* were verified as uniform; however, the outputs of *DRS* were shown to be non-uniform in some cases.

Runtime performance assessment of the numerical *ConvolutionalFixedSum* algorithm shows that it improves upon the performance of the *DRS* algorithm by several orders of magnitude. Further, with optimizations to select when convolution-based volume calculation is appropriate, *ConvolutionalFixedSum* can employ a *UUnifast*-like mode to provide comparable performance to *UUnifast* in cases with no constraints.

ConvolutionalFixedSum supersedes the *DRS* algorithm. It provides an effective solution that supports appropriate taskset generation, when constraints are applied per task. As shown in Section V of [14] this can improve understanding of scheduling policies that were previously evaluated using less sophisticated methods. *ConvolutionalFixedSum* has a wide range of potential uses. It is applicable to problems where modeling of an unbiased partitioning of a fixed resource is required, subject to individual constraints. Diverse examples include packet and message scheduling in telecommunications, the division of an overall budget or investment into different categories in economics, and bed allocation between departments in a hospital setting [21].

A. Organization

The remainder of the paper is organized as follows: Section II reviews the literature on the problem of uniformly generating vectors of random values with a fixed sum, from the perspective of real-time systems. Section III derives the IVoRS method for uniform sampling from a valid region demarcated by upper and lower constraints in n-dimensional space. Section IV illustrates how the IVoRS method works by using it to derive the *UUnifast* algorithm. Section V provides a method based on convolution and Fast-Fourier Transforms (FFT) to calculate the volume ratio required, thus instantiating the *ConvolutionalFixedSum* algorithm. Section VI evaluates the uniformity of the outputs from *ConvolutionalFixedSum* and the existing state-of-the-art algorithms. The run-time performance of *ConvolutionalFixedSum* is also compared to that of the *DRS* algorithm. Section VII concludes with a summary and directions for future work. The Appendix gives an account of the implementation issues that cause the outputs from the *DRS* algorithm to form a non-uniform distribution in some cases.

II. Related Work

When assessing the performance of schedulability tests [8] for real-time systems, it is necessary to ensure that the input data used is free from bias. However, this problem has a non-trivial constraint: that the total utilization, U, is divided between n tasks. This results in the distribution of utilization values to each task being non-uniform. Therefore, it is not sufficient to simply allocate random numbers from a uniform distribution, as would be the case when randomly selecting a point on a square, cube, or hypercube. Indeed, no linear transformation is capable of generating a uniform distribution, as demonstrated by the *UScale* algorithm, the outputs of which were shown to be non-uniform and a cause of bias [3], [4].

In 2004, Bini and Buttazzo derived the *UUnifast* algorithm [3], [4]. This was the first algorithm within the real-time systems community that solved the uniformity problem. *UUnifast* works by recognizing that the probability of selecting any given value grows polynomially with the number of tasks, and so uses an inverse-polynomial transformation to achieve a uniform distribution. This approach was later identified by Griffin et al. [14] as being equivalent to sampling from the flat Dirichlet distribution [30] via the marginal Beta distribution method [12], which also provided a formal mathematical proof of uniformity for *UUnifast*.

While *UUnifast* solved the problem of sampling utilization values to support traditional schedulability analyses of single processor systems, advances in scheduling theory for real-time systems provided new challenges. Scheduling multiprocessor systems provided an additional constraint: while the total utilization U could potentially be as large as $U = m$ for m processors, no single threaded task could use more than one processor, hence the utilization of each individual task was constrained to be no more than 1. Initially, this problem was addressed by Davis and Burns [9], [10] using a simple extension to *UUnifast* called *UUnifast-Discard*, which allows for $U > 1$ and discards outputs when any of the n components exceeds 1. *UUnifast-Discard* is effective when $n \gg U$; however, when the number of tasks is low, it becomes impractical due to the large number of discards required.

The problem of an extra constraint on each task was addressed by the *RandFixedSum* algorithm [11] identified in 2009 by Emberson at al., and originally developed as a MATLAB routine by Stafford [35] in 2006. *RandFixedSum* exploits the symmetry of the problem to sample from the valid region demarcated by the constraints; however, it has higher

complexity $O(n^2)$, in both execution time and memory space than *UUnifast*, and lacks a formal proof.

While *RandFixedSum* only supports a single constraint on task utilizations, individual constraints restricting the utilization values of individual tasks are required to support assessment of schedulability analyses when task utilizations are multi-valued or can be decomposed into multiple constituent parts. This occurs with mixed-criticality systems [5], multi-core systems [23], with typical and worst-case execution times [31], [1], self-suspensions [6], and resource locking. As an example, in mixed-criticality systems each task designated as high-criticality typically has both high-criticality and low-criticality execution time budgets. These correspond to a high-criticality utilization and a smaller low-criticality utilization for the task. When uniformly generating random values for the tasks' low-criticality utilizations, these values must not exceed the corresponding high-criticality utilizations already chosen. Thus, the latter values effectively form individual per-task upper constraints. Alternatively, generating low-criticality utilization values first, results in a set of individual per-task lower constraints on the high-criticality utilization values.

Naïve approaches could be employed to the above problem, such as the *UAdd* algorithm [14]. Using *UAdd*, high-criticality tasks can be considered as having two uniformly sampled components, with the first of these components designated as the low-criticality utilization, and the sum of the two components designated as the high-criticality utilization. While this ensures that the model of mixed-criticality systems is met, the Central Limit Theorem [2] shows that the sum of two or more uniformly distributed variables does not itself form a uniform distribution. Alternatively, the *UUnifast-Discard* approach [9], [10] could be used. In this context, *UUnifast-Discard* would generate sets of values using *UUnifast*, discarding any that break the constraints given to it. This is guaranteed to result in a uniform distribution of points that meet the constraints; however, this comes at the expense of discarding a large number of points, to the point where the algorithm is impractical. For example, if the constraints are $[1, 10^{-3}, 10^{-3}]$, then only about 1 in 5000 of all generated points will be accepted.

The Dirichlet Rescale Algorithm (*DRS*) [14] was designed to address these issues, thus supporting individual constraints on task utilization values. The *DRS* algorithm works by observing that affine transformations preserve the uniformity of points sampled across the transformed space. Therefore, it is possible to sample a point using the flat Dirichlet distribution and then apply linear transformations (rescaling) until that point lies within the valid region, provided that those linear transformations always include the entire valid region. While the *DRS* algorithm is computationally expensive, as it always moves towards a solution this was deemed by the authors to be an acceptable trade-off. Unfortunately, the *DRS* algorithm has multiple issues. The issue of complexity was acknowledged by the authors. Each rescale operation has $O(n^2)$ complexity, and many rescales may be required. While the *DRS* algorithm does not have a formal proof of complexity, it was observed [14] to have exponential scaling with respect to n.

A more insidious problem is that the *DRS* algorithm does not always manage to generate a uniform distribution. The uniformity test used by Griffin et al. [14] only provides evidence of uniformity around the center of the region. The slices test, defined in this paper, is a far more effective test of uniformity close to the edges of the valid region. Using the slices test, the outputs of the *DRS* algorithm are shown, in Section VI, to be non-uniform. The reasons for this non-uniformity are discussed in the Appendix.

III. IVoRS: Sampling from a Uniform Multivariate Distribution of finite arbitrary shape

In order to sample from a distribution, two things are required: firstly, a definition of the distribution, and secondly an algorithm to sample from it. To accomplish the latter, inverse transform sampling [34] is typically used for univariate distributions: a random number is generated in the range $[0, 1]$, and then this value is transformed by the distribution's Inverse Cumulative Distribution Function (ICDF) [34].

The logic of inverse transform sampling is as follows: for a given distribution and $x \in [0, 1]$, the ICDF at point x gives a value y such that the proportion of the distribution below y is x. Therefore, it is possible to transform a point x from one distribution A to a point from another distribution B by transforming x by the Cumulative Distribution Function (CDF) of A and the ICDF of B. Using the notation F_A to denote the CDF of the random variable A, The resulting point, $y = F_B^{-1}(F_A(x))$ has the same properties with respect to distribution B as x has with respect to distribution A. As the CDF of uniform random numbers in the range $[0, 1]$ is the identity function, and uniform univariate random values can be generated by a number of means, inverse transform sampling can be trivially applied to generate univariate random values that follow a given distribution, provided that the distribution's ICDF is known.

This intuition can also be applied to multivariate distributions, by applying it to the inverse marginal CDF, as can be seen in the marginal-beta distribution method of sampling from the Dirichlet distribution [12, p.585], or the *UUnifast* algorithm [3], [4], [14]. A marginal distribution is the distribution of a single variate, regardless of the values taken by other variates. Traditionally, marginal distributions were computed for discrete valued data by summing the frequencies of each value a variate took in the margins of a table, hence the name [34].

When using inverse transform sampling and the inverse marginal CDF to sample from a multivariate distribution, the process can be thought of as splitting the problem into a sequence of 1-dimensional sampling problems. Given the random variable to sample $\mathbf{X} = \langle X_1 ... X_n \rangle$, and a function $rand()$, which returns a random value in the range $[0, 1]$, the first variate can be sampled from the inverse marginal CDF as follows:

$$x_1 = inverse(marginal(X_1, F_{\mathbf{X}}))(rand()) \qquad (1)$$

For subsequent variates, the CDF becomes conditional on the variates that have already been sampled, leading to sampling from an inverse marginal conditional CDF as follows:

$$x_i = inverse(marginal(X_i, F_{\mathbf{X} \mid X_j = x_j \ \forall j<i})(rand()) \tag{2}$$

Therefore, to sample from a uniform distribution with a finite arbitrary shape, it only remains to show how the inverse marginal conditional CDF can be calculated.

To begin with, we first define the CDF by exploiting the Probability Density Function (PDF) of the distribution. A PDF defines the density of the probability distribution at a given point, and can be used to calculate the relative likelihood of a point being chosen. The PDF can be defined in relation to the CDF, since the CDF is the integral of the PDF. However, in the case of the uniform distribution, there is an extra piece of information: since every point is equally likely, the PDF is a constant for any point that is part of the distribution, and zero otherwise. Denoting the PDF of the distribution of $\mathbf{X}$ as $f_{\mathbf{X}}$, we can define the PDF and CDF of a univariate distribution as follows:

$$\begin{aligned} f_{\mathbf{X}}(x) &= \begin{cases} c \text{ if } x \text{ is a valid value of } \mathbf{X} \\ 0 \text{ otherwise} \end{cases} \\ F_{\mathbf{X}}(x) &= \int_{-\infty}^{x} f_{\mathbf{X}}(u)du \end{aligned} \tag{3}$$

Where the value of c depends on the distribution $\mathbf{X}$, and in particular, the finite shape that defines the valid values of $\mathbf{X}$. This can the be generalized to a multivariate distribution by integrating over each variate:

$$\begin{aligned} f_{\mathbf{X}}(\mathbf{x}) &= \begin{cases} c \text{ if } \mathbf{x} \text{ is a valid value of } \mathbf{X} \\ 0 \text{ otherwise} \end{cases} \\ F_{\mathbf{X}}(\mathbf{x}) &= \int_{-\infty}^{x_1} ... \int_{-\infty}^{x_n} f_{\mathbf{X}}(\mathbf{U})du_1...du_n \end{aligned} \tag{4}$$

However, as $f_{\mathbf{X}}$ is either c or 0, we can simplify (4) using the validity function $V_{\mathbf{X}}$, which is 1 if $\mathbf{x}$ is a valid value of $\mathbf{X}$, and zero otherwise. As $f_{\mathbf{X}}(\mathbf{x}) = cV_{\mathbf{X}}(\mathbf{x})$, $F_{\mathbf{X}}$ can be rewritten as follows:

$$F_{\mathbf{X}}(\mathbf{x}) = c \int_{-\infty}^{x_1} ... \int_{-\infty}^{x_n} V_{\mathbf{X}}(\mathbf{U})du_1...du_n \tag{5}$$

As $F_{\mathbf{X}}$ is a CDF, the domain of $F_{\mathbf{X}}$ is, by definition, $[0, 1]$. As the function $V_{\mathbf{X}}(\mathbf{x}) \geq 0 \ \forall \mathbf{x}$, the integrals of each variate between $-\infty$ and an upper limit are non-negative and monotonically increasing as that upper limit increases. As $V_{\mathbf{X}}$ is non-zero only on a finite region, it follows that the integral of $V_{\mathbf{X}}$ must have an upper limit. Therefore, c is the inverse of the maximum value that this integral can take. Substituting for c in (5), it follows that:

$$F_{\mathbf{X}}(\mathbf{x}) = \frac{\int_{-\infty}^{x_1} ... \int_{-\infty}^{x_n} V_{\mathbf{X}}(\mathbf{U})du_1...du_n}{\int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} V_{\mathbf{X}}(\mathbf{U})du_1...du_n} \tag{6}$$

Intuitively, the integral over the function $V_{\mathbf{X}}$ can be viewed as a way of calculating the volume of the valid region below the point specified by the upper limits of the integral. Therefore, using $vol(\mathbf{X})$ to denote the volume of the space of valid values of $\mathbf{X}$, the CDF can also be specified as:

$$F_{\mathbf{X}}(\mathbf{x}) = \frac{vol(\mathbf{X}|X_i \leq x_i)}{vol(\mathbf{X})} \tag{7}$$

Thus $F_{\mathbf{X}}$ can be determined solely in relation to the validity function $V_{\mathbf{X}}$, with no dependence on the PDF.

Next, we define the marginal conditional CDF required by (2). This is achieved by observing that as variates are sampled and become fixed, the dimensionality of the valid region for the unsampled variates decreases. For example, in a 3-dimensional problem, if one value is fixed, then the plane on which the remaining variates lie is 2-dimensional. Hence, the marginal conditional CDF of the $i'th$ variate, F_{X_i} can be written as follows:

$$\begin{aligned} &F_{X_i}(x_i) = \\ &\frac{\int_{-\infty}^{x_i} \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} V_{\mathbf{X}}(\mathbf{U}|U_j = u_j \forall j \leq i)du_i du_{i+1}...du_n}{\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} V_{\mathbf{X}}(\mathbf{U}|U_j = u_j \forall j \leq i)du_i du_{i+1}...du_n} \end{aligned} \tag{8}$$

or as a volume ratio:

$$F_{X_i}(x_i) = \frac{vol(\mathbf{X}|X_j = x_j \forall j \leq i, X_i \leq x_i)}{vol(\mathbf{X}|X_j = x_j \forall j \leq i)} \tag{9}$$

Finally, we come to inverting the marginal conditional CDF for use in inverse transform sampling. Unfortunately, analytically calculating the inverse of an arbitrary function is not possible; however, numerically finding the inverse of a function is a well-studied field. Several fundamental statistical distributions, such as the beta and gamma distributions have no exact distribution for their inverse, and yet iterative methods can be used to approximate them [18], [26], [22]. Therefore, for this problem we employ a root finding algorithm to calculate the inverse of the marginal conditional CDF at the point specified by inverse transform sampling. Any root finding method could be used, for example Newton's method [19] or Binary Search [20]. We use the ITP algorithm [29], as it combines the efficiency of Newton's method with the guaranteed convergence and complexity bound of Binary Search.

Putting all of this together gives the Inverse Volume Ratio Sampling (IVoRS) method, listed in Algorithm 1. The IVoRS method can sample from a uniform distribution defined over a given valid region of n-dimensional space, provided *only* that it is possible to calculate the ratio of the volume of a subsection of the region to the volume of the entire region.

As an example, we demonstrate the generation of a uniformly selected point within a solid 3-dimensional shape consisting of ten unit cubes shown in Figure 1. Here, the valid values for each variate depend on the values for the other variates, meaning that it is not possible to calculate the variates independently. However, as the shape is made up of cubes, it is possible to calculate a hyperplane such that a given ratio of the volume lies beneath that hyperplane. Applying the

Algorithm 1 IVoRS Algorithm

Input: ρ, the initial valid region, and $vr(P, x)$, a function that calculates the volume below x of a given region P.

Output: $\mathbf{p}$, a uniformly sampled point from the region described by ρ and vr.

1: $\mathbf{p} \leftarrow []$
2: **for** $k \in [0, n]$ **do**
3: $r \rightarrow$ a random number in $[0, 1]$
4: Set λ such that $vr(\rho, \lambda) = r$ ▷ Use root finding if necessary
5: $\rho \rightarrow \rho$ intersected with the hyperplane $x_i = \lambda$
6: append λ to $\mathbf{p}$
7: **end for**
8: **return** $\mathbf{p}$

Algorithm 2 *UUnifast* Algorithm

Input: U, the total utilization to allocate, and N, the number of tasks

Output: $\mathbf{p}$ a list of length N containing a uniformly sampled point that divides U into N components

1: $\mathbf{p} \leftarrow []$
2: $k \leftarrow 0$
3: **while** $k < (N - 1)$ **do**
4: $r \leftarrow U \cdot \text{random}()^{\frac{1}{N-k}}$
5: append $U - r$ to $\mathbf{p}$
6: $U \leftarrow r$
7: **end while**
8: append U to $\mathbf{p}$
9: **return** $\mathbf{p}$

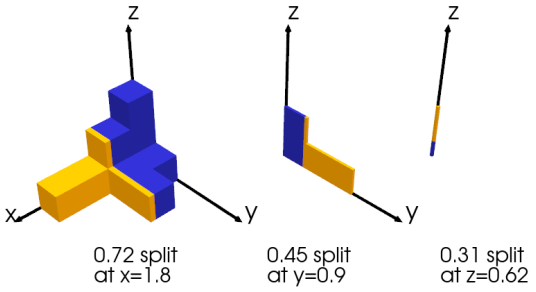

Fig. 1: Demonstrating Uniform CDFs from Volume Ratios

IVoRS method, we can calculate the inverse marginal CDF of the uniform distribution.

We begin by using the inverse marginal CDF to apply inverse transform sampling to the marginal distribution of the x axis. In Algorithm 1, Line 3, we generate the random number 0.72, and then on Line 4 calculate that 0.72 of the volume of the shape lies below the hyperplane $x = 1.8$. Next, on Line 5, the valid region is intersected with the hyperplane $x = 1.8$, reducing the problem to a 2-dimensional problem along the y and z axes. Considering the y axis, we follow the same steps to apply inverse transform sampling to the marginal distribution of y, conditional on $x = 1.8$. We generate 0.45 as our random number, and calculate that 0.45 of the area of this shape lies below $y = 0.9$. Finally, considering the z axis, we intersect the shape with $x = 1.8, y = 0.9$. The resulting line represents the marginal distribution of z, conditional on $x = 1.8, y = 0.9$. We generate our final random number of 0.31, and calculate that 0.31 of the length of the line lies below $z = 0.62$. The uniformly sampled random point is therefore $[1.8, 0.9, 0.62]$.

IV. IVoRS Implementation of UUniFast

To illustrate how the IVoRS method works, we now use it to derive the *UUnifast* algorithm. A pseudocode implementation of *UUnifast* is given below in Algorithm 2.

The *UUnifast* algorithm selects a uniform random point such that each component is greater than zero and the components sum to a given value U. Geometrically, for a vector of length n, this can be represented by a simplex in $n + 1$ dimensional space where the vertices of the simplex are at the intersection of the axes at the value U. This simplex lies on the hyperplane $\Sigma_{i=0}^{n} x_i = U$.

The volume of the valid region can be found by observing that when $U = 1$, the valid region is the standard simplex and therefore for an arbitrary U, the volume is a scaled form of the volume of the standard simplex, which is given by:

$$vol(n) = \frac{\sqrt{n+1}}{n!} \tag{10}$$

Scaling the standard simplex in each dimension by the same factor gives the *UUnifast* simplex for total utilization U. Such scaling also gives a formula for calculating the volume of a standard simplex by it's "height" h i.e., the difference between two parallel hyperplanes that enclose the simplex.

$$vol_h(h, n) = h^n \frac{\sqrt{n+1}}{n!} \tag{11}$$

To obtain the volume ratio, it is necessary to find the volume below a hyperplane of the form $x_i = c$. Cutting off the top of a simplex in this manner results in a more complex shape; however, the portion that is cut off is guaranteed to be a simplex of height $U - x_i$. Therefore, it is simpler to calculate the volume below a hyperplane by subtracting the volume above it from the entire volume. The volume ratio used for the CDF in the IVoRS method can thus be written as:

$$\begin{aligned} VR_{UUniFast}(x, U, n) &= \frac{vol_h(U, n) - vol_h(U - x, n)}{vol_h(U, n)} \\ &= \frac{U^n - (U - x)^n}{U^n} \\ &= 1 - \left(\frac{U - x}{U}\right)^n \end{aligned} \tag{12}$$

Note, that when calculating the volume ratio, the term $\frac{\sqrt{n+1}}{n!}$ cancels out, since every component is a simplex volume calculated via its height, eliminating the need to calculate $n!$

To complete the derivation of the *UUnifast* algorithm using the IVoRS method, it remains to calculate the inverse CDF. In general, the IVoRS method uses root finding algorithms to accomplish this; however, in this case the function can be

rearranged to achieve an analytical solution, where x is the returned value of the inverse CDF and y is a uniform random number in the range $[0, 1]$:

$$\begin{aligned} y &= 1 - \left(\frac{U-x}{U}\right)^n \\ \left(\frac{U-x}{U}\right) &= (1-y)^{\frac{1}{n}} \\ U - x &= U(1-y)^{\frac{1}{n}} \\ x &= U - U(1-y)^{\frac{1}{n}} \\ x &= U - Uy^{\frac{1}{n}} \end{aligned} \tag{13}$$

With the final step holding since y is a uniform random number in the range $[0, 1]$, and therefore the distribution of y and $(1 - y)$ are identical, given that the function $k : [0, 1] \rightarrow [0, 1], k(y) = 1 - y$ is a continuous bijective function.

To show this is the same as the iterative step of *UUnifast*, we note that x is the variate appended to $\mathbf{p}$ in Algorithm 2. Hence, $x = U - r$, and rewriting y as the result from the function $random()$ in Algorithm 2, we can show equivalence as follows:

$$\begin{aligned} x = U - U(y^{\frac{1}{n}}) &= U - r \\ r &= U(\text{random}()^{\frac{1}{n}}) \end{aligned} \tag{14}$$

Letting $n = N - k$, corresponding to the k'th iteration of *UUnifast*, this matches the assignment to r in Algorithm 2. Hence, *UUnifast* performs an equivalent calculation to the IVoRS method using this volume calculation.

V. ConvolutionalFixedSum

In this section, we show how to calculate the volume of the valid region of a simplex demarcated by upper and lower constraints. We assume that the problem has been converted into a *canonical* form, where the lower constraints are all zero, the total utilization is 1, and the upper constraints have been scaled accordingly. Later, we show how the same transformations used by the *DRS* algorithm [14] can be applied to convert any valid problem to and from this form.

To calculate the required volume, we borrow, and decipher, due to somewhat non-standard notation, a trick from Wolpert and Wolf [37], as follows. The integral of a function $H(\mathbf{x}) = \prod_{i=1}^{n} h_i(x_i)$, where x_i are the components of $\mathbf{x}$, over the scaled standard simplex can be written in the form:

$$\int_0^1 \cdots \int_0^{1-\sum_{i=1}^{n-2} x_i} \int_0^{1-\sum_{i=1}^{n-1} x_i} H(\mathbf{x}) dx_{n-1} dx_{n-2} \cdots dx_1 \tag{15}$$

Note that, as expected, this describes the volume of an $n-1$ dimensional simplex as the variable x_n is not integrated over, and $x_n = 1 - \sum_{i=1}^{n-1} x_i$. By defining a shorthand term $\sigma_k = 1 - \sum_{i=1}^{k-1} x_i$, and expanding $H(\mathbf{x})$ to the $h_i(x_i)$ functions, we arrive at the following:

$$\int_0^1 \cdots \int_0^{\sigma_{n-2}} \int_0^{\sigma_{n-1}} \prod_{i=1}^{n} h_i(x_i) dx_{n-1} dx_{n-2} \cdots dx_1 \tag{16}$$

Noting that, by definition, $x_n = \sigma_n = \sigma_{n-1} - x_{n-1}$, and that the variables $x_1 ... x_{n-2}$ are constant multiplicands with respect to an integral over x_{n-1} and therefore can be extracted from the integration, we can rewrite (16) as follows:

$$\begin{aligned} \int_0^1 \cdots \int_0^{\sigma_{n-2}} \prod_{i=1}^{n-2} h_i(x_i) \\ \int_0^{\sigma_{n-1}} h_{n-1}(x_{n-1}) h_n(\sigma_{n-1} - x_{n-1}) dx_{n-1} \\ dx_{n-2} \cdots dx_1 \end{aligned} \tag{17}$$

Convolution, denoted by $\otimes$, is a commutative operator applied to two functions as follows:

$$\begin{aligned} (f \otimes g)(v) &= \int_0^t f(v) g(t-v) dv \\ &= \int_0^t g(v) f(t-v) dv \quad = (g \otimes f)(t) \end{aligned} \tag{18}$$

Recognizing that the middle line of (17) is the convolution of h_{n-1} and h_n over σ_{n-1}, we can perform a substitution and further unraveling of the integral product to arrive at:

$$\begin{aligned} \int_0^1 \cdots \int_0^{\sigma_{n-3}} \prod_{i=1}^{n-3} h_i(x_i) \\ \int_0^{\sigma_{n-2}} h_{n-2}(x_{n-2}) \cdot (h_{n-1} \otimes h_n)(\sigma_{n-2} - x_{n-2}) dx_{n-2} \\ dx_{n-3} \cdots dx_1 \end{aligned} \tag{19}$$

Finally, we recognize that this rule can be applied inductively, as the middle line of (19) is the convolution of h_{n-2} and $(h_{n-1} \otimes h_n)$ over σ_{n-2}. Noting that convolution is commutative, and denoting multiple convolutions by $\bigotimes$, we can rewrite the integral of the function $H(\mathbf{x})$ over the scaled standard simplex as:

$$\left(\bigotimes_{i=1}^{n} h_i \right)(1) \tag{20}$$

where the value 1, comes from the outermost integral of (19), determining the value to convolve over. Therefore, to calculate the volume of the valid region, we need to supply a function of the correct form, which takes the value 1 within the valid region, and 0 everywhere else. We define the function $V_{\mathbf{uc}}(\mathbf{x})$ as follows, where $\mathbf{uc}$ is the vector of upper constraints:

$$V_{\mathbf{uc}}(\mathbf{x}) = \prod v_{uc_i}(x_i) \tag{21}$$

$$v_{uc_i}(x) = \begin{cases} 1 & 0 \leq x \leq uc_i \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

By construction, $V_{\mathbf{uc}}(\mathbf{x}) = 1$ if and only if $\mathbf{x}$ is between the lower constraints (assumed to be zero) and the upper constraints, and therefore lies within the valid region. Note that $V_{\mathbf{uc}}(\mathbf{x})$ is integrable everywhere on $\mathbb{R}$. By construction, $V_{\mathbf{uc}}(\mathbf{x})$ is of the correct form to be used in the above volume calculation, and can therefore be used to calculate volumes via convolution. Substituting into (20) yields the following function for the volume of a valid region, vvr:

In order to calculate volume ratios for the IVoRS method, it suffices to use the same process, but modifying the constraints on the first variate to obtain the volume for a subsection of the valid region. For this, we define a modification function m which constructs the appropriate modified constraint vector as follows:

$$m(\mathbf{uc}, w) = \langle \min(uc_1, w), uc_2, ...uc_n \rangle \tag{23}$$

The modified constraint vector enables the volume of the valid region below w to be calculated. A volume ratio function, vr'_{cfs}, appropriate for use in the IVoRS method, can then be defined as:

$$vr'_{cfs}(\mathbf{uc}, w) = \frac{vvr(m(\mathbf{uc}, w))}{vvr(\mathbf{uc})} \tag{24}$$

Finally, we relax the simplifying assumption that the lower constraints are all 0 and the total utilization is 1. To do this, we use the method employed by the *DRS* algorithm [14]. The lower constraints are subtracted from the corresponding upper constraints and the total utilization, and the upper constraints are scaled so that the total utilization is 1. (The inverse of this transformation is subsequently applied to the generated point to obtain the output from *ConvolutionalFixedSum*). The vr'_{cfs} function can be extended into the vr_{cfs} function, which encompasses the additional parameters required by the initial transformation as follows:

$$vr_{cfs}(U, \mathbf{lc}, \mathbf{uc}, w) = vr'_{cfs}\left(\frac{\langle uc_i - lc_i \in [1, n] \ \forall i \rangle}{U - \sum_{i=1}^{n} lc_i}, w\right) \tag{25}$$

where n is the number of tasks, and equates to the length of the vectors $\mathbf{lc}$ and $\mathbf{uc}$. Finally, we observe that the first three parameters $U, \mathbf{lc}$, and $\mathbf{uc}$ describe the shape of the valid region, and can therefore be expressed as a single parameter P. Utilizing the form $vr_{cfs}(P, w)$, the function can be employed in the IVoRS method, see Algorithm 1, forming the *ConvolutionalFixedSum* algorithm, with ITP [29] used to find the required volume ratio. To compute the convolution in (**??**), two methods are possible: Analytical and Numerical.

A. Analytical Method

Wolpert and Wolf [37] use their technique in conjunction with the Laplace Transform [32], since in the Laplace domain convolution behaves as function multiplication, making the operation trivial. However, while it is possible to recover functions from the Laplace domain, the function may become more complicated. In the case of the *ConvolutionalFixedSum* algorithm, again making the simplifying assumption that $\mathbf{lc} = 0$ and $U = 1$, the functions v_{uc_i} can be implemented with a step function, which is a well-studied function with regards to the Laplace transform [32]. However, once the functions are multiplied within the Laplace domain, factored and then recovered by a package such as SymPy [27], the convolution of n step functions comprises 2^n step functions[1]. This is expected, since each upper constraint can interact with every other upper constraint, giving a total of 2^n combinations of upper constraints to take into account.

[1]If the lower constraints are not normalized to zero, this becomes 2^{2^n} step functions, since for each set of upper constraints, each set of lower constraints must be checked.

Algorithm 3 Analytical Convolution

Input: $\mathbf{uc}$, a vector of length n containing the upper constraints.

Output: c, the value of the convolution at 1, giving the volume of the valid region described by $\mathbf{uc}$.

1: $c \leftarrow 0$
2: $A \leftarrow \{1...n\}$ ▷ Set of values from 1 to n
3: **for** $a \subset A$ **do**
4: **if** $\sum_{i \in a} uc_i \leq U$ **then**
5: $c' = (1 - \sum_{i \in a} uc_i)^{(n-1)}$
6: $c \leftarrow c + -1^{|a|} c'$
7: **end if**
8: **end for**
9: **return** c

A generic version of the analytical convolution algorithm is given in Algorithm 3. As can be seen on Line 3, every subset of the constraints must be checked individually. Including the empty set, there are 2^n subsets of A (the set of values from 1 to n). Line 4 checks to see if a given set of constraints is applicable, i.e. the constraints intersect with the valid region. If so, Line 5 calculates the volume of this intersection and Line 6 updates the current volume. Note that if the number of constraints that make up a region is odd, its volume is subtracted from the valid region, otherwise it is added. This can be explained by analogy to the volume calculation, considering what each region represents:

1) For 0 constraints, the volume is the entire upper constraint simplex, and should therefore be added.
2) For 1 constraint, the volume is a corner of the upper constraints simplex that is greater than an upper constraint, and should therefore be subtracted.
3) For 2 constraints, the volume is the intersection of two of the previous constraints, and should therefore be added to correct for the double counting that occurred in the previous step for 1 constraint.

and so on.

As the complexity of Algorithm 3 is $O(2^n)$, it is unfortunately intractable for large n, especially as the use of the ITP algorithm for calculating the inverse CDF results in multiple calls to the volume calculation. This leads us to the numerical approximation for convolution.

B. Numerical Approximation

Convolution is frequently used in signal processing [33], and so numerical methods for convolution are well studied. It is possible to approximate the convolution of functions by transforming the functions into signals. These signals can then be sampled, numerically convolved, and then an approximation of the convolution returned. To accomplish this, the first step is to define the functions $v^s_{uc}(x)$, which encode the functions $v_{uc}(x)$ as signals with s samples.

$$v^s_{uc}(x) = \begin{cases} 1 & 0 \leq x \leq \lceil min(uc, 1) \cdot s + \frac{1}{2} \rceil \\ 0 & \text{otherwise} \end{cases} \quad (26)$$

We can then construct signals by sampling the functions v^s_{uc} over the range $[0, s]$. As expected, the number of samples s controls the accuracy of the numerical approximation. While this calculation uses a canonical form where $U = 1$, the actual value of U is however important in terms of precision. For example, $s = 1000$ and $U = 1$ yields 3 decimal places of precision, whereas $s = 1000$ and $U = 10$ only yields 2 decimal places of precision. As experiments assessing the performance of scheduling algorithms may require a fixed degree of accuracy, the sample length s can be specified in terms of a required granularity ϵ and U. This results in setting $s = \lceil U\epsilon \rceil$. With this formulation, the total utilization U affects the runtime of the numerical approximation, since larger values of U lead to commensurately larger sample sizes.

To avoid excluding any part of the valid region, the signals used represent a volume slightly larger than that of the valid region. This can result in the *ConvolutionalFixedSum* algorithm initially generating a point that lies outside of the valid region. If this happens, then the point is discarded, and the algorithm retries, generating a new point. In practice, retries are a rare event that occurs with a probability that is inversely proportional to the signal size. For example, in our experiments, the retry rate was approximately 0.1%, with a negligible effect on performance.

Once the sampled signals are obtained, than they can be convolved. As with the analytical method, a transformation exists that reduces convolution to pairwise multiplication: the Fourier Transform [32]. The Fast Fourier Transform (FFT) [7] is the most efficient method for this, with complexity of $O(s\,log(s))$, with optimizations available for real valued data. Once all of the signals are transformed by FFT, then they can be multiplied, the inverse FFT applied, and the result of the convolution extracted at the end of the sampled interval. An outline of this approach is given in Algorithm 4.

Algorithm 4 Numerical Convolution

Input: $\mathbf{uc}$, a vector of length n containing the upper constraints, s, the number of samples to use

Output: c, the value of the convolution, giving the volume of the valid region described by $\mathbf{uc}$.

1: $conv \leftarrow \text{fft}([v^s_{uc_1}(x) \mid x \in [0, s]])$
2: **for** $y \in [2...n]$ **do**
3: $\quad conv \leftarrow conv \odot \text{fft}([v^s_{uc_y}(x) \mid x \in [0, s]])$
4: **end for**
5: $c \leftarrow \text{ifft}(conv)[s]$
6: **return** c

The complexity of the numerical method of volume calculation is $O(n^2\, s \log(sn))$, which makes it far more appropriate than the analytical method for use with large n. When used with the IVoRS method, this results in a total complexity of $O(n^3\, s \log(sn))$ for the numerical *ConvolutionalFixedSum* algorithm[2].

There are some practical concerns when implementing the numerical method of convolution that relate to floating point precision. Firstly, it is important to sort the constraints from largest to smallest, as precision is expressed relative to the largest constraint under consideration. By processing the largest constraints first, for which smaller constraints have less impact, error from the numerical approximation can be minimized. Similarly, normalizing the convolved signal after each convolution reduces the effect of large values causing error. Finally, as FFT libraries implement circular convolution [25], it is necessary to pad the signal for each convolution.

To improve performance, best practice techniques were employed, taken from state of the art implementations of convolution [16], as well as the academic literature [24]. This included ensuring that all arrays were trimmed of leading and trailing zeros, and no unnecessary convolutions were carried out. We also implemented a caching strategy, exploiting the fact that when searching for a root to calculate the inverse CDF only one constraint is changed. Hence, caching the convolution of the remaining constraints reduces the number of convolutions required significantly. Without this optimization, the algorithm would have had a complexity of $O(n^4 log(n))$ rather than $O(n^3 log(n))$.

C. Example of applying ConvolutionalFixedSum

Figure 2 shows an illustrative example of how the *ConvolutionalFixedSum* algorithm solves a 3-dimensional problem. In this example, we assume a total utilization of 1.0, represented by the green simplex. There are three upper constraints, $x < 0.5, y < 0.7, z < 0.8$, and implicit lower constraints of $x, y, z > 0$, represented by the red simplex. The valid region is the intersection of the two simplicies. The algorithm begins by solving for the largest constraint in the z axis. To perform inverse transform sampling, a random number is generated in the range $[0, 1]$; in this case 0.59. ITP is then used with either the analytical or numerical volume ratio calculation to find the value of z such that 0.59 of the volume of the valid region lies below z. To a precision of three decimal places, ITP makes an initial guess of 0.250, before refining this value to 0.324, then 0.289, before finally reaching an answer of 0.290. Hence, $z = 0.290$ is selected. The x and y coordinates are simpler to solve for, as the line defined by $z = 0.290$ only need be split at a randomly selected point. Solving for y, the value of y must lie between 0.210 and 0.7. (The lower bound occurs because allocating $y < 0.210$ would cause $x = 1.0 - z - y > 0.5$, breaking the constraint on x). Once again, a random number is generated for inverse-transform sampling, this time 0.12. The line segment is therefore split such that 0.12 of the length of the line is before the point, thus obtaining $y = 0.269$. Finally, solving $x+y+z = 1$ gives $x = 0.441$. The uniformly sampled random point is therefore $[0.441, 0.269, 0.290]$.

[2]An optimization to $O(n^2 \log(n)\, s\log(sn) \log(s))$ is theoretically possible, but has not yet been implemented.

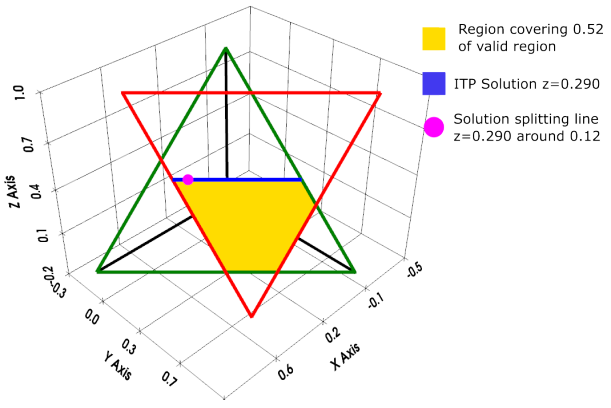

Fig. 2: Application of *ConvolutionalFixedSum* to the constraints $[x < 0.4, y < 0.7, z < 0.8]$

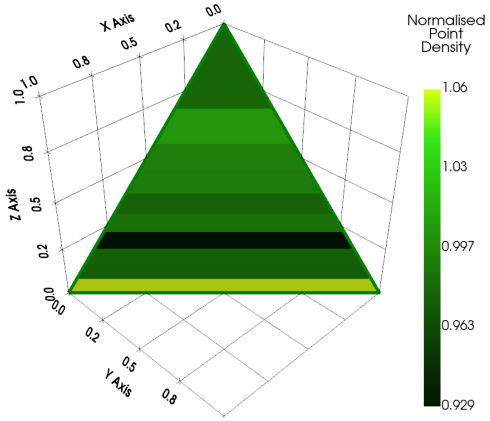

Fig. 3: Slices uniformity test in 3 dimensions for *UUnifast*. Range of Normalized Point Density Values $[0.929, 1.06]$.

VI. Evaluation

This section evaluates the uniformity of the outputs of the *ConvolutionalFixedSum*, *DRS*, *RandFixedSum* and *UUnifast* algorithms. To reduce the number of variables, our evaluation only considers problems in their *canonical* form, i.e. $U = 1$ and $\mathbf{lc} = 0$, since all problems can be transformed to this form. The validity of the numerical approximation used in *ConvolutionalFixedSum* is also investigated. Finally, the runtime performance of the numerical *ConvolutionalFixedSum* algorithm is determined, considering different sample sizes.

A. Uniformity Testing: the Slices Test

To verify that the outputs from a given algorithm form a uniform distribution, we developed a highly sensitive uniformity test, called the *slices test*, which utilizes volume calculations. In each of the n dimensions, the slices test divides the valid region into k slices of equal volume with the slice boundaries defined by hyperplanes parallel to the axes. N points are then generated by the algorithm, and allocated to the appropriate containing slice. Since each slice has the same volume, the expected number of points in each slice is N/k, subject to statistical variation. A Chi-squared (χ^2) test [34] is then used to determine how likely the observed distribution is to occur, assuming the null hypothesis of a uniform distribution. The slices test has the advantage that it explores the whole of the valid region in each dimension and is sensitive to any non-uniformity at the edges, as well as to any gradient effects.

Figures 3 and 4 illustrate the slices test showing a uniform distribution for *UUnifast* and *ConvolutionalFixedSum* respectively. Note, the normalized density is given by the number of points contained in each slice divided by the expected number N/k. In each figure the density is close to 1, subject to statistical variation. The distribution of points to slices shown in these figures pass the χ^2 test, indicating uniformity.

To calculate the volume of the slices, the inverse CDF from *ConvolutionalFixedSum* was used. This allowed each slice to be expressed as being bounded by $ICDF(\frac{k}{10})$ and $ICDF(\frac{k+1}{10})$ for $k \in [0, 10)$. To calculate the slices, the analytical method was used. To verify that these slices were correct, the slices test was conducted on the provably uniform *UUnifast* algorithm; if the slices were incorrect, then the slices test would fail *UUnifast*, creating a contradiction.

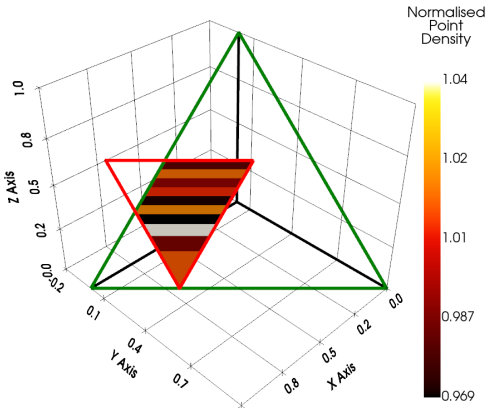

Fig. 4: Slices uniformity test in 3 dimensions for *ConvolutionalFixedSum*. Range of Normalized Point Density Values $[0.970, 1.04]$.

We used the slices test to evaluate the uniformity of the outputs from the *ConvolutionalFixedSum*, *DRS*, *RandFixedSum* and *UUnifast* algorithms. In each case $N = 10{,}000$ points were generated across $k = 10$ slices in each of n dimensions. Further, the number of dimensions (tasks) was varied in the range $[3, 15]$, with the total utilization set to $U = 1$. In addition, the following parameters were used:

- *UUnifast*: No other parameters
- *RandFixedSum*: Single upper constraint drawn from a uniform distribution between $[\frac{1.01}{n}, 1]$. The lower bound of this distribution is chosen to ensure sufficient volume in the valid region for the slices test to be effective.
- *DRS* and *ConvolutionalFixedSum*: Upper constraints generated by *UUnifast* with a fixed sum of 1.5.
- Numerical *ConvolutionalFixedSum*: Signal size $s = 10{,}000$

Each test was repeated 1000 times, thus 117,000 χ^2 tests

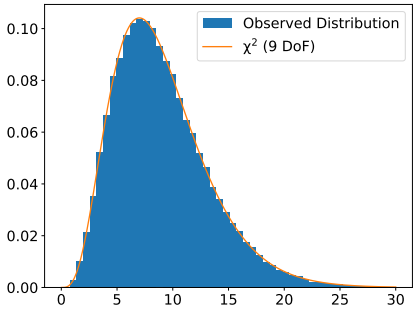

Fig. 5: Distribution of χ^2 values for 117,000 *UUnifast* experiments

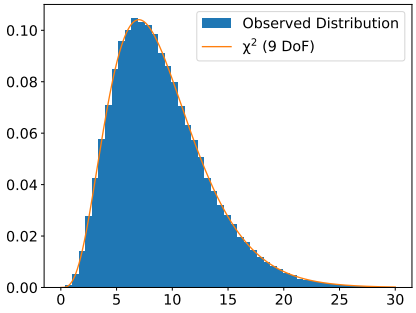

Fig. 6: Distribution of χ^2 values for 117,000 analytical *Convolutional-FixedSum* experiments

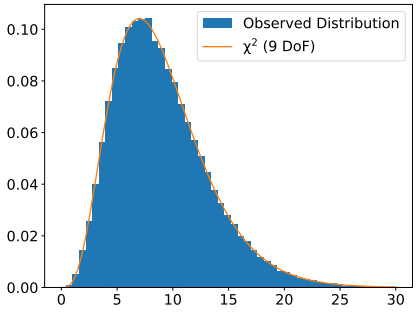

Fig. 7: Distribution of χ^2 values for 117,000 numerical *Convolutional-FixedSum* experiments

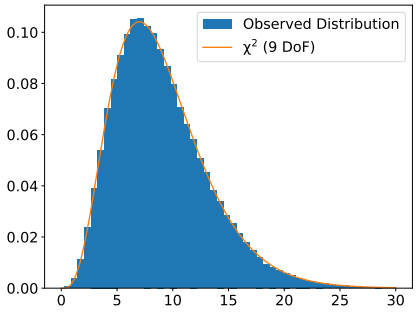

Fig. 8: Distribution of χ^2 values for 117,000 *RandFixedSum* experiments

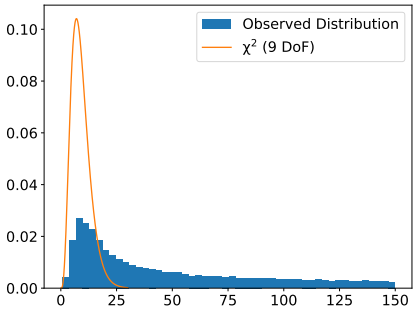

Fig. 9: Distribution of χ^2 values distribution for 117,000 *DRS* experiments

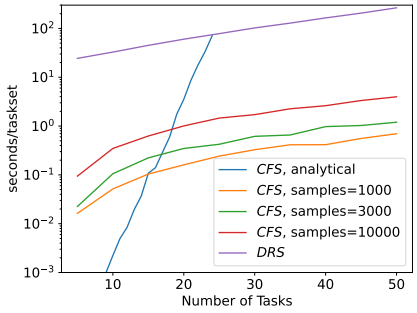

Fig. 10: Performance of *ConvolutionalFixedSum* and *DRS* algorithms

were run on each algorithm. To run these tests 117,000,000 points were generated by each algorithm (10,000 for each of the 1,000 experiments for each $n \in [3, 15]$).

To evaluate the 117,000 χ^2 tests, the distribution of the observed χ^2 test statistic was plotted against the χ^2 distribution with 9 Degrees of Freedom (9-DoF). These distributions were then compared using the Kolmogorov-Smirnov (KS) test [34] to determine if the observed cumulative distribution differed from the χ^2 distribution with a significance of 0.05.

Figure 5 shows the distribution of the observed χ^2 statistic vs. the χ^2 (9-DoF) distribution for *UUnifast*. The results from *UUnifast* produce values matching the χ^2 (9-DoF) distribution and pass the KS-test with a p-value of 0.45. Figure 6 similarly shows that the results from the analytical *ConvolutionalFixed-Sum* algorithm also pass the KS-test with a p-value of 0.70.

The χ^2 results for the numerical *ConvolutionalFixedSum* algorithm, shown in Figure 7 show a similar result, and again passes the KS-test with a p-value of 0.66. However, note that the numerical *ConvolutionalFixedSum* algorithm is an approximation. For this experiment, a signal size of 10,000 was used; however, with a signal size of 1,000, the approximation is insufficient and fails the KS-test with a p-value of $\approx 10^{-83}$. (See Section VI-D for a discussion of best practice in using the numerical *ConvolutionalFixedSum* algorithm).

Figure 8 shows the distribution of the observed χ^2 statistic vs. the χ^2 (9-DoF) distribution for *RandFixedSum*, again showing the expected distribution and passing the KS-test with a p-value of 0.23.

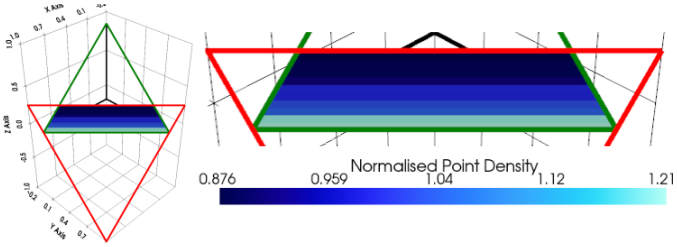

Fig. 11: Slices test illustrating non-uniformity of *DRS*. Range of Normalized Point Density Values $[0.878, 1.21]$.

Finally, Figure 9 illustrates that the outputs from the *DRS* algorithm lack uniformity. In this case, the observed χ^2 distribution fails the KS-test (p-value 0). Examining the data in detail reveals that as the number of constraints increases, so the non-uniformity also increases. This is expected as the number of constraints increases, so the probability of smaller constraints increases since the constraints sum to 1.5. The presence of very small constraints appears to cause issues for the *DRS* algorithm. This is forced in Figure 11, which shows a 3-dimensional view of a 4-dimensional problem with constraints $[1.0, 1.0, 0.25, 10^{-4}]$. As the dimension not shown has a constraint with a tiny magnitude, it should not impact uniformity with respect to the remaining dimensions; however, Figure 11 shows a clear gradient, and a significantly greater variation in normalized point density than observed in Figures

3 and 4. The distribution of points to slices shown in this figure fails the χ^2 test, indicating non-uniformity.

B. Validity Testing

The analytical *ConvolutionalFixedSum* algorithm was stress tested by conducting checks as part of the slices test to ensure that the slices returned were of equal volume. As each slice is constructed by the difference between two sets of constraints, this validity test is able to determine if analytical *ConvolutionalFixedSum* is behaving as expected. For some sets of constraints, the validity test did not pass due to floating point precision issues, which are largely unavoidable. Algorithm 3 (Lines 5 and 6) sums values across a potentially very wide range. Our implementation uses Kahan-Babušhka-Neumaier summation [28] to mitigate error, which is detectable by monitoring the compensation term in the Kahan-Babušhka-Neumaier sum. This is performed automatically within the *ConvolutionalFixedSum* algorithm. Using this method, approximately 0.3% of the generated sets of points have observable floating point error. However, such non-uniformity is confined to the dimensions with the smallest upper constraints, which were observed to have an upper constraint $< 3.3 \times 10^{-5}$, with dimensions that have larger constraints unaffected. Hence, the absolute deviation from uniformity across the whole of the valid region is negligible.

To stress test the validity of the numerical *ConvolutionalFixedSum* algorithm, we conducted tests by comparing the numerical volume calculation to the analytical convolution with a single constraint set, which is equivalent to the derivation of *UUnifast* in Section IV. This method is used for testing the maximum value of n as the numerical convolution based slices test may be subject to similar degradation with large n, invalidating the test. From these tests, we observed that the volume calculation used in the numerical *ConvolutionalFixedSum* algorithm with $10{,}000$ samples provides a good approximation of the *UUnifast* volume ratio calculation until $n = 50$. Given that the worst case for numerical convolution is maximizing the signal, this provides confidence that the numerical *ConvolutionalFixedSum* algorithm provides good accuracy up to at least $n = 50$. Beyond $n = 50$, numerical *ConvolutionalFixedSum* begins to degrade due to floating point accuracy limitations.

C. Performance Testing

Figure 10 shows the runtime performance of the *ConvolutionalFixedSum* and *DRS* algorithms when run on a Raspberry Pi 4. The Pi 4 was chosen as a readily available computer for replicability of results, while providing sufficient computational power. Observe that increasing the sample size s, while necessary for accuracy, is relatively expensive leading to a polynomial increase in runtime, $O(s \log(s))$. The numerical *ConvolutionalFixedSum* algorithm, however, still significantly outperforms the *DRS* algorithm, and is approximately two orders of magnitude faster with the default sample size of $s = 10{,}000$. The runtime of the analytical *ConvolutionalFixedSum* algorithm grows exponentially, with a slight deviation around $n = 17$, likely due to the memory architecture of the Raspberry Pi 4, indicating that it is ill-suited to problems where n is substantially greater than 20.

The *UUnifast* and *RandFixedSum* algorithms have complexities of $O(n)$ and $O(n^2)$ respectively, both are substantially faster than *ConvolutionalFixedSum* and should be used if the flexibility to cater for individual constraints is not required.

D. Best Practice when using ConvolutionalFixedSum

The *ConvolutionalFixedSum* algorithm can exhibit limited non-uniformity arising from accuracy issues, due to the inherent approximation of numerical convolution, e.g., too small a sample size, and due to how floating point precision affects the implementation of both analytical and numerical convolution. This section discusses best practice in mitigating these issues. In the implementation of analytical convolution, a check has been implemented that detects if floating point error may be of concern. This check can sometimes give false positives, indicating potential floating point errors when they are none, but *not* false negatives, which would indicate that accuracy is fine, when it is not. Floating point error typically occurs when the canonical form of an upper constraint is very small. This was encountered in our experiments when a single constraint was $< 3.3 \times 10^{-5}$. It can also occur when there are multiple somewhat larger constraints. Any dimension affected by floating point error will have values that may not be completely uniform, but other dimensions will not be affected. This issue can therefore be mitigated by removing dimensions with a very small range of potential values from the problem description, and instead treating those values as fixed. Alternatively, users may choose to accept the non-uniformity in very small dimensions, given that the impact on overall uniformity across the whole of the valid region is negligible.

For numerical *ConvolutionalFixedSum*, a signal size that is too small to represent the constraints accurately can compromise uniformity. This can be completely avoided by ensuring that the signal size is at least $\lceil \frac{10^x}{U} \frac{min(U, max(uc_i - lc_i)}{min(uc_j - lc_j)} \rceil \mid i, j \in [1, n]$, where x is the number of decimal places of precision required. For example, with $U = 1$, a minimum range $min(uc_j - lc_j) = 0.01$, and $x = 2$ decimal places of accuracy, equates to a minimum recommended signal size of $s = 10{,}000$. Alternatively, the slices test can be used to check for accuracy. In the event that it is not possible to use analytical convolution to calculate the volume of the slices, e.g., for large n, numerical convolution can be used, with a sample size that is 10 times larger than that used by the numerical *ConvolutionalFixedSum* algorithm, albeit with a higher probability of false negatives occurring.

VII. Conclusions

Work on this paper originally started with an assessment of the DRS algorithm that revealed an issue with the uniformity of its outputs. On further investigation, this non-uniformity appeared to be due to insurmountable implementation issues. Having identified the need for an alternative algorithm, we

designed the IVoRS method. This method employs standard techniques for drawing from a multivariate distribution with volume calculation and root finding algorithms to allow sampling from any region whose volume can be calculated.

Having established the IVoRS method, we showed how the *UUnifast* algorithm can be derived via the IVoRS method, providing an alternative proof of correctness for *UUnifast*. We then derived a method for calculating the volume of the valid region and sections of it using convolution, thus instantiating the *ConvolutionalFixedSum* algorithm. As the analytical form of *ConvolutionalFixedSum* has a complexity of $O(2^n)$, we also developed a numerical approximation that uses Fast Fourier Transforms to achieve a complexity of $O(n^3 \log(n))$.

To demonstrate the correctness of *ConvolutionalFixedSum*, we developed the highly sensitive slices test for uniformity that provides an effective test over the entire valid region. To verify that this test was correct, we applied it to the *UUnifast* algorithm, which is proven to produce outputs that follow a uniform distribution. We used the slices test to verify that the outputs from the *RandFixedSum* algorithm also follow a uniform distribution, while those from the *DRS* algorithm do not. Both the analytical and numerical *ConvolutionalFixedSum* algorithms were shown to provide outputs that follow a uniform distribution. The accuracy of both methods under extreme conditions was also investigated, and best practices for avoiding floating point error in the analytical method and approximation error in the numerical method devised. Performance testing showed that the analytical *ConvolutionalFixedSum* algorithm is tractable for small values of $n \leq 20$. For larger values of n, the numerical *ConvolutionalFixedSum* algorithm provides substantially better scaling, and is therefore the practical choice.

In summary, *ConvolutionalFixedSum* presents a precise, but $O(2^n)$ analytical method and a more tractable $O(n^3 \log(n))$ numerical approximation, with the required statistical tests provided to show that neither method suffers from the substantial non-uniformity issues of *DRS*. We recommend that the use of the *DRS* algorithm is deprecated, and replaced by *ConvolutionalFixedSum*, which provides both superior performance and uniformity. All source code for the analytical and numerical *ConvolutionalFixedSum* algorithms, and the evaluation methods is available online [15].

As further work, we intend to produce a version of *ConvolutionalFixedSum* that takes advantage of GPUs for performing convolution, and examine extensions to support task sets of size greater than 50. We also intend to produce a version of the algorithm that directly supports a discrete version of the problem. This requires that a vector of n values is generated that sum to within some small error ϵ of a fixed value, and that the values produced are from a discrete lattice.

Finally, while this paper was under review, we were contacted by the authors of [36]. They had independently shown, both analytically and empirically, that the *DRS* algorithm can produce outputs that are not uniformly distributed. They proposed a revision of the algorithm called *DRSC* that resolves this issue. However, the current version of *DRSC* achieves this at substantial computational cost, with complexity that lies between that of the *DRS* and *UUniFast-Discard* algorithms, reducing its viability for large n and for small normalized constraints. The *DRSC* algorithm [36] also adds support for multivariate constraints, such as $x_1 + x_2 < 1$.

Appendix: Uniformity issues with DRS

When investigating the observed lack of uniformity in the *DRS* algorithm, we focused on the use of floating point as this was noted as a potential weakness [14] in the *DRS* implementation. The issues we found included:

- Floating Point Error: The mitigation used in the *DRS* algorithm checks if the sum of the generated values is within a parameter ϵ of the total required. However, this assumes that all floating point errors are in the same direction, which is not necessarily the case.
- Finite Entropy: The double precision floating points used by the *DRS* algorithm encode 53 bits of entropy, since they are within the range [0, 1]. This places a hard limit on the number of rescales that can be performed before the randomness of the initial point is exhausted.

From our experiments, we concluded that the issues with the *DRS* algorithm can potentially be mitigated by: (i) ensuring that the ϵ parameter is substantially smaller than the smallest constraint, and (ii) limiting the maximum magnitude of rescales applied to counter concerns about floating point error and finite entropy. Both mitigations increase the probability of retrying. As *DRS* already appears to scale exponentially, as shown in Figure 10, increasing the probability of retrying will increase the runtime further. An alternative mitigation using arbitrary precision floating points was considered, which does not increase the probability of retrying, but this was deemed not to be computationally tractable. Therefore, while the *DRS* algorithm can theoretically be fixed, the fixes are not practical and hence we cannot recommend its use.

Acknowledgments

This research was funded in part by the MARCH Project (EP/V006029/1), Innovate UK SCHEME project (10065634) and the CHEDDAR Communications hub (EP/Y037421/1, EP/Y036514/1, EP/X040518/1). EPSRC Research Data Management: No new primary data was created during this study.

References

[1] Sanjoy K. Baruah. Rapid routing with guaranteed delay bounds. In *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*, pages 13–22. IEEE Computer Society, 2018. doi:10.1109/RTSS.2018.00012.

[2] Patrick Billingsley. *Convergence of Probability Measures*, volume 23. Wiley Series in Probability and Statistics, 1999. doi:10.1002/9780470316962.

[3] Enrico Bini and Giorgio C. Buttazzo. Biasing effects in schedulability measures. In *16th Euromicro Conference on Real-Time Systems (ECRTS 2004), 30 June - 2 July 1004, Catania, Italy, Proceedings*, pages 196–203. IEEE Computer Society, 2004. URL: https://doi.ieeecomputersociety.org/10.1109/ECRTS.2004.7, doi:10.1109/ECRTS.2004.7.

[4] Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real Time Syst.*, 30(1-2):129–154, 2005. URL: https://doi.org/10.1007/s11241-005-0507-9, doi:10.1007/S11241-005-0507-9.

[5] Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, 2018. doi:10.1145/3131347.

[6] Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn B. Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil C. Audsley, Raj Rajkumar, Dionisio de Niz, and Georg von der Brüggen. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real Time Syst.*, 55(1):144–207, 2019. URL: https://doi.org/10.1007/s11241-018-9316-9, doi:10.1007/S11241-018-9316-9.

[7] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. URL: http://www.jstor.org/stable/2003354.

[8] Robert I. Davis. On the evaluation of schedulability tests for real-time scheduling algorithms. In *Proceedings International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2016. URL: https://waters2016.inria.fr/files/2017/02/WATERS16-proceedings-final.pdf#page=4.

[9] Robert I. Davis and Alan Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In Theodore P. Baker, editor, *Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1-4 December 2009*, pages 398–409. IEEE Computer Society, 2009. doi:10.1109/RTSS.2009.31.

[10] Robert I. Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real Time Syst.*, 47(1):1–40, 2011. URL: https://doi.org/10.1007/s11241-010-9106-5, doi:10.1007/S11241-010-9106-5.

[11] Paul Emberson, Roger Stafford, and Robert I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *Proceedings International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, July 2010. URL: https://retis.sssup.it/waters2010/waters2010.pdf#page=6.

[12] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian data analysis (3rd edn).*, volume 23. Chapman & Hall/CRC, 2021.

[13] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(6):721–741, 1984. doi:10.1109/TPAMI.1984.4767596.

[14] David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 76–88. IEEE, 2020. doi:10.1109/RTSS49844.2020.00018.

[15] David Griffin and Robert I. Davis. ConvolutionalFixedSum Software, March 2025. URL: https://github.com/dgdguk/convolutionalfixedsum/, doi:10.5281/zenodo.15107012.

[16] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi:10.1038/s41586-020-2649-2.

[17] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 04 1970. arXiv:https://academic.oup.com/biomet/article-pdf/57/1/97/23940249/57-1-97.pdf, doi:10.1093/biomet/57.1.97.

[18] Dhivya Prabhu K, Sanjeev Singh, and V. Antony Vijesh. A third-order iterative algorithm for inversion of cumulative central beta distribution. *Numer. Algorithms*, 94(3):1331–1353, 2023. URL: https://doi.org/10.1007/s11075-023-01537-6, doi:10.1007/S11075-023-01537-6.

[19] Carl T Kelley. *Solving nonlinear equations with Newton's method*. SIAM, 2003.

[20] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[21] Jie Li, Sichen Li, Jun Luo, and Haihui Shen. Simulation optimization for inpatient bed allocation with sharing. *Journal of Systems Science and Systems Engineering*, 2024. doi:10.1007/s11518-024-5625-9.

[22] Alberto Llera and Christian Beckmann. Estimating an inverse gamma distribution. 05 2016. doi:10.48550/arXiv.1605.01019.

[23] Claire Maiza, Hamza Rihani, Juan Maria Rivas, Joël Goossens, Sebastian Altmeyer, and Robert I. Davis. A survey of timing verification techniques for multi-core real-time systems. *ACM Comput. Surv.*, 52(3):56:1–56:38, 2019. doi:10.1145/3323212.

[24] Filip Markovic, Alessandro Vittorio Papadopoulos, and Thomas Nolte. On the convolution efficiency for probabilistic analysis of real-time systems. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, July 5-9, 2021, Virtual Conference*, volume 196 of *LIPIcs*, pages 16:1–16:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.ECRTS.2021.16, doi:10.4230/LIPICS.ECRTS.2021.16.

[25] Clare D McGillem and George R Cooper. *Continuous and discrete signal and system analysis*. Saunders College Publishing, 1991.

[26] M. E. Mead. Generalized inverse gamma distribution and its application in reliability. *Communications in Statistics - Theory and Methods*, 44(7):1426–1435, 2015. arXiv:https://doi.org/10.1080/03610926.2013.768667, doi:10.1080/03610926.2013.768667.

[27] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. doi:10.7717/peerj-cs.103.

[28] A. Neumaier. Rundungsfehleranalyse einiger verfahren zur summation endlicher summen. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 54(1):39–51, 1974. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/zamm.19740540106, doi:10.1002/zamm.19740540106.

[29] Ivo F. D. Oliveira and Ricardo H. C. Takahashi. An enhancement of the bisection method average performance preserving minmax optimality. *ACM Trans. Math. Softw.*, 47(1):5:1–5:24, 2021. doi:10.1145/3423597.

[30] Ingram Olkin and Herman Rubin. Multivariate beta distributions and independence properties of the wishart distribution. *Annals of Mathematical Statistics*, 35(1):261–269, March 1964. doi:10.1214/aoms/1177703748.

[31] Sophie Quinton, Matthias Hanke, and Rolf Ernst. Formal analysis of sporadic overload in real-time systems. In Wolfgang Rosenstiel and Lothar Thiele, editors, *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 515–520. IEEE, 2012. doi:10.1109/DATE.2012.6176523.

[32] Laurent Schwartz. *Mathematics for the Physical Sciences*. Addison-Wesley Publishing Company, 1966.

[33] Steven W. Smith. *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, USA, 1997.

[34] Murray R Spiegel and Larry J Stephens. *Schaum's outline of statistics*. McGraw Hill Professional, 2017.

[35] Roger Stafford. Random vectors with fixed sum. Technical Report Available at https://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum, MathWorks, 2006.

[36] Rick S. H. Willemsen, Wilco van den Heuvel, and Michel van de Velden. Generating random vectors satisfying linear and nonlinear constraints, 2025. URL: https://arxiv.org/abs/2501.16936, arXiv:2501.16936.

[37] David H. Wolpert and David R. Wolf. Estimating functions of probability distributions from a finite set of samples. *Phys. Rev. E*, 52:6841–6854, Dec 1995. URL: https://link.aps.org/doi/10.1103/PhysRevE.52.6841, doi:10.1103/PhysRevE.52.6841.