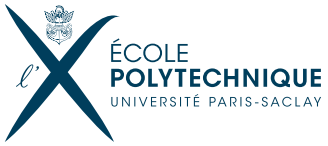

# RESEARCH INTERNSHIP REPORT

# CONSTRUCTION OF EQUIDISTRIBUTED POINT SETS

## Rapport non confidentiel

Pierre Marion (X2015)

| | |
|---|---|
| Option: | Department of Applied Mathematics |
| Field: | Probabilistic and Statistical Modeling |
| Referent: | Vincent Bansaye |
| Supervisor: | Pierre L'Ecuyer |
| Date: | 25 mars 2018 - 25 août 2018 |
| Organism: | University of Montreal |
| | Department of Computer Science and Operations Research |
| | Aisenstadt Pavilion, QC H3T 1N8 Montreal, Canada |




# Déclaration d'intégrité relative au plagiat

Je soussigné Marion, Pierre certifie sur l'honneur:

1. Que les résultats décrits dans ce rapport sont l'aboutissement de mon travail.
2. Que je suis l'auteur de ce rapport.
3. Que je n'ai pas utilisé des sources ou résultats tiers sans clairement les citer et les référencer selon les règles bibliographiques préconisées.

*Je déclare que ce travail ne peut être suspecté de plagiat.*

Le 17 août 2018, Pierre Marion

# Abstract

L'utilisation de méthodes quasi-Monte Carlo (QMC) pour l'intégration numérique ou la simulation stochastique permet d'obtenir des réductions de variance conséquents par rapport à une estimation Monte-Carlo simple. Ces méthodes QMC sont basées sur des constructions algébriques de points avec de bonnes propriétés d'équi-distribution dans $[0,1]^s$, comme les réseaux digitaux à $2^k$ points en base 2. On s'intéressera dans ce rapport à la preuve d'un résultat de convergence de variance pour cette construction. En utilisant des points de Sobol, nous obtenons un taux de convergence de la variance en $\mathcal{O}(k^{2(s-1)}/4^k)$, à comparer avec le taux usuel en $\mathcal{O}(1/2^k)$ pour l'estimation Monte-Carlo.

Concrètement, la plupart des utilisateurs se servent d'ensembles de points QMC pré-calculés et stockés dans un logiciel. Cependant, dans certains cas, ces points pré-calculés ne sont pas adaptés au problème de l'utilisateur, et il est alors intéressant de calculer des nouveaux points QMC plus adaptés à ce problème. Lors de ce stage, nous avons grandement étendu un logiciel pour la construction à la volée d'ensembles de points QMC paramétrés par l'utilisateur. Ce logiciel libre, qui s'appelle désormais *LatNet Builder*, est à notre connaissance le seul qui répond à cette problématique avec ce degrée de généralité et de configurabilité. Notre principal ajout au logiciel est l'implémentation efficace de la construction de réseaux digitaux en base 2. L'efficacité numérique de ces constructions est testée et discutée dans ce rapport.

Cette implémentation suppose de concevoir des algorithmes efficaces pour l'évaluation de la qualité des réseaux digitaux. En particulier, nous avons conçu un nouvel algorithme, qui devrait faire l'objet d'une publication, pour le calcul d'un critère appelé la $t$-valeur : il est plus performant que ceux connus dans la littérature dans le cas où $k \geq s+1$.

The use of quasi-Monte Carlo (QMC) methods for numerical integration or stochastic simulation can bring consequent variance reduction compared to a simple Monte Carlo sampling. These QMC methods are based on algebraic constructions of points in $[0,1]^s$ with good properties of equidistribution, such as $2^k$-points digital nets in base 2. The proof of a result of variance convergence for this construction will be of interest in this report. By using Sobol points, a variance convergence rate in $\mathcal{O}(k^{2(s-1)}/4^k)$ is obtained, to be compared with the usual rate of $\mathcal{O}(1/2^k)$ for Monte-Carlo sampling.

Concretly, most practitioners use pre-computed QMC point sets, which are stored in a software. Nevertheless, in some cases, these pre-computed point sets are not adapted to the practitioner's problem. Then it can be beneficial to compute new bespoke QMC points. During this internship, we greatly extended a software for the on-the-fly construction of QMC point sets parametrized by the user. This open-source software, now called *LatNet Builder*, is to our knowledge the only one to answer this issue to such a degree of generality and configurability. Our main contribution to the software is the efficient implementation of the construction of digital nets in base 2. The numerical efficiency of these constructions is tested and discussed in this report.

This implementation requires the conception of efficient algorithms for the evaluation of the quality of digital nets. In particular, we conceived a new algorithm for the computation of a criterion called the $t$-value, which should be the subject of a paper to be published. This new algorithm is more performant than those known in the literature in the case when $k \geq s+1$.

# Acknowledgment

This work would not have been possible without the help of Maxime Godin, who contributed heavily and enthusiastically to this project. If proof was needed, we proved that sometimes a group of two can achieve more than the sum of two separate contributions.

I would like to thank warm-heartedly Professor Pierre l'Ecuyer who offered the position at University of Montreal, and guided us throughout the project. His vision on the field of QMC was most enlightening.

My thanks also go to Florian Puchhammer for his help and presence throughout the summer, as well as to all the interns, PhD students and post-docs at the Stochastic Simulation and Optimization lab.

Finally, I would like to thank Professor Emmanuel Gobet for helping me to find this position.

# Contents

# 1 Introduction

To perform numerical integration and stochastic simulation, *Monte-Carlo (MC) sampling* is a foremost method. Typically for a function $f$ of finite variance, one can estimate

$$\mu = \int_{[0,1]^s} f(\mathbf{u}) d\mathbf{u}$$

by the Monte-Carlo estimator $\hat{\mu}_n = \frac{1}{n}\sum_{i=0}^{n-1} f(\mathbf{U}_i)$, where the $\mathbf{U}_i$ are independent uniformly distributed variables over $[0,1]^s$. The central limit theorem then garantees that the error will decrease as $\mathcal{O}(n^{-1/2})$. By using variance reduction techniques, one can improve this convergence rate.

*Quasi-Monte Carlo (QMC) sampling* is a way of reducing the variance, which consists in taking evenly distributed deterministic point sets as evaluation points. The estimator $\hat{\mu}_n$ is replaced by the deterministic approximation $\bar{\mu}_n = \frac{1}{n}\sum_{i=0}^{n-1} f(\mathbf{u}_i)$ where the points $\mathcal{P}_n = \{\mathbf{u}_0, \cdots, \mathbf{u}_{n-1}\}$ cover the hypercube evenly.

Generally speaking, QMC point sets are designed to be *low-discrepancy point sets* [12]: there exists a bound of the general form

$$|\bar{\mu}_n - \mu| \leq D(\mathcal{P}_n) V(f) \tag{1}$$

holding for functions $f$ belonging to some function space (most often an Hilbert space) $\mathcal{H}$. Here $V(f)$ is the norm of $f$ in this space, while $D(\mathcal{P}_n)$ is a measure of the *discrepancy* of the point set in this space. The goal is that $D(\mathcal{P}_n)$ converges at a faster rate than the Monte-Carlo estimator: typically, with the techniques explored below, we expect

$$D(\mathcal{P}_n) = \mathcal{O}(n^{-1}(\log n)^{s-1}) = o(n^{-1/2}). \tag{2}$$

There are a great number of parametrized constructions that allow to obtain a bound of the form (1). An exhaustive survey is by far out of the scope of this report, so we chose to focus on one particular construction, namely *digital nets*. This method constructs points parametrized by matrices called *generating matrices*, one per dimension. The quality of a digital net depends on the choice of its generating matrices.

The optimization of these parameters is a difficult problem that is left to specialists: usually, practitioners use pre-computed QMC point sets, which are tabulated and stored in a software made by researchers, such as [2,6]. These point sets were found by optimizing some sort of quality criterion. Unfortunately, this 'one-size-fits-all' approach does not always work, as these point sets will be more or less adapted depending on the application. The goal of this internship was to design, develop and distribute a software to compute on the fly new digital nets, according to the parameters of the problem: dimension $s$, number of points $n$, regularity and structure of the function $f$, etc. The hope is that these bespoke point sets will yield better convergence rates than pre-computed sets.

Finding a good digital net to solve a given numerical integration problem is a two-step optimization process:

1. find some pertinent quality criterion, chosen in accordance with the parameters of the problem. In the QMC literature [13,19], this criterion is called a *figure of merit*.

2. find generating matrices such that the obtained digital net optimizes the quality criterion.

The first point boils down to finding the appropriate space of functions $\mathcal{H}$, then deriving a bound of the form (1). The figure of merit is obtained as a majoration of the discrepancy $D(P_n)$. There is often a tradeoff between the tightness of the majoration and the computation time of the figure of merit. To be able to compute it efficiently, some ingenious algorithmic techniques are needed, to spare and reuse computations.

The second point deals with the exploration of the space of generating matrices where efficiency issues arise. For instance, to generate $2^{10}$ points in dimension 5, one needs to find five $10 \times 10$ binary matrices, thus explore a space of cardinality $2^{500} \simeq 10^{50}$. An exhaustive search is unfeasible, hence the need to find more economical exploration methods, and also restrict the search space to special classes of matrices, found using algebraic methods.

Thus, the problem of finding highly-uniform point sets lies at the fruitful interface of three broad domains: algebra and linear algebra, analysis, and algorithmic.

Once a good point set has been generated, it can be tested to evaluate the gain in convergence rates in comparison with standard Monte Carlo sampling. These numerical experiments are done using test functions that mimick real-world problems. The real-world applications are numerous and include for instance the simulation of Partial Differential Equations, of the activity in a network, of the evolution of the price of a financial option, etc.

A software has been developed at the Stochastic Simulation and Optimization laboratory, to construct highly-uniform point sets for quasi-Monte Carlo. Our internship consisted in extending the software to include new construction methods, and especially digital nets. Previously called Lattice Builder as it constructed lattice rules [14], it is now called *LatNet Builder* for 'Lattices and Nets Builder'. It is written in C++ using state-of-the-art programming techniques to reach outstanding numerical efficiency. To our knowledge, it is currently the most advanced open-source software in the world for construction of quasi-Monte Carlo point sets.

This report is organized as follows: in section 2, we delve into the theoretical background, to show how bounds of the form (1) can be obtained. The goal is also to present notions that will be necessary to understand the work done in the next sections. The theoretical work in this section is not new, although the figure of merit we introduce was not presented under this form in the literature. In section 3, we explain a new method that we designed during the internship to compute efficiently some figures of merit. In section 4, we present some numerical experiments that use our point sets. The work presented in both these sections is entirely new in our knowledge.

# 2 Digital nets: constructions and figures of merit

## 2.1 First definitions: digital nets, $t$-value

The definitions in this subsection are taken from [4], adapted for base-2, as we will work only in this base, for numerical efficiency purposes: the use of base-2 for number representation in computers allow for drastic computation speed-up compared to other bases, as we will see later.

Nets are a type of point set whose quality is measured according to a property of uniformity of the distribution of points: for some interval $J$, we ask that the 'right' number of points falls into $J$, i.e. the empirical measure of the interval coïncides with its

true measure. Of course, such a property will not be verified by all possible intervals, so we restrict ourselves to a reasonable class of intervals.

**Definition 2.1.** For a given dimension $s \geq 1$, a positive integer $k$ and an integer $t$ with $0 \leq t \leq k$, a point set $\mathcal{P}$ of $2^k$ points in $[0,1)^s$ is called a $(t,k,s)$-net if every interval of the form

$$J = \prod_{l=1}^{s} \left[ \frac{A_l}{2^{d_l}}, \frac{A_l + 1}{2^{d_l}} \right) \text{ where } d_1, \ldots, d_s \in \mathbb{N}_0 \text{ with } d_1 + \cdots + d_s = k - t \text{ and } A_l \in [0, 2^{d_l})$$

contains exactly $2^t$ points of $\mathcal{P}$.

The (strict) *t-value* of the net is the smallest $t$ verifying this property, and we denote it $t(\mathcal{P})$.

**Remark 2.2.** As every point set in $[0,1)^s$ of cardinality $2^k$ is a $(k,k,s)$ net, being a net is not an interesting property in itself. We are in fact looking for nets with the smallest possible $t$-value. As we will see later, the $t$-value will be the key ingredient to construct an interesting figure of merit, as it has two very nice properties: it is related to the discrepancy of the point set, and it is easy to compute under some assumptions on the net.

Although the concept of net is very general, we will be solely interested in a particular type of net construction, which is the most widely used [4]: digital nets. This construction allows to reduce the problem of finding a good net to a linear algebra problem.

**Definition 2.3.** Let $\mathbb{F}_2$ be the finite field with 2 elements (denoted 0 and 1). For a given dimension $s \geq 1$, positive integers $r$ and $k$, let $M_1, \ldots, M_s$ be $r \times k$ matrices over $\mathbb{F}_2$.

The *digital net* generated by these matrices is the $s$-dimensional point set $\{\boldsymbol{x_0}, \ldots, \boldsymbol{x_{2^k-1}}\}$ of cardinality $2^k$ defined as follows: for each $0 \leq i \leq 2^k - 1$, let $\boldsymbol{i} = (a_0, \ldots, a_{k-1})^T$ be the digits (in $\mathbb{F}_2$) of the 2-adic expansion $i = \sum_{j=0}^{k-1} a_j 2^j$. For each coordinate $l$, let $\boldsymbol{y_l} = M_l \times \boldsymbol{i} = (y_{i,l,1}, \ldots, y_{i,l,r})^T$. Write $\boldsymbol{x_i} = (x_{i,1}, \ldots, x_{i,s})^T$. Then:

$$x_{i,l} = \sum_{j=1}^{r} y_{i,l,j} 2^{-j}.$$

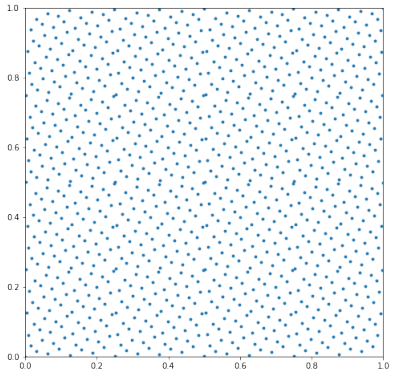
**Example 2.4.** Take as an example the two $k \times k$ matrices

$$M_1 = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix} \qquad \text{and} \qquad M_2 = \begin{pmatrix} 0 & \ldots & 0 & 1 \\ 0 & \ldots & 1 & 0 \\ \vdots & \iddots & \vdots & \vdots \\ 1 & \ldots & 0 & 0 \end{pmatrix}$$

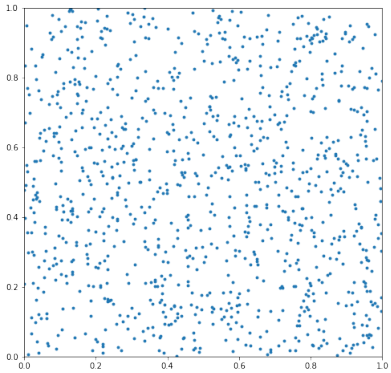
The identity matrix $M_1$ maps an integer $i = \sum_{j=0}^{k-1} a_j 2^j$ to $\sum_{j=0}^{k-1} a_j 2^{-j}$.

The reversed identity matrix $M_2$ maps an integer $i = \sum_{j=0}^{k-1} a_j 2^j$ to $i/2^k$, since $\boldsymbol{y_2} := M_2 \times \boldsymbol{i} = (a_{k-1}, \ldots, a_0)^T$, thus $\sum_{j=1}^{k} y_{i,2,j} 2^{-j} = \sum_{j=1}^{k} a_{k-j} 2^{-j} = \frac{i}{2^k}$.

The resulting point set is called the $2^k$-point Hammersley point set in base 2, and is represented in Figure 1 for $k = 10$.



(a) Hammersley point set in base 2



(b) Random point set

Figure 1: Comparison of two point sets with 1024 points in dimension 2

**Remark 2.5.** A digital net is uniquely defined by the ordered set of its generating matrices, hence we will as of now indifferently speak of the $t$-value of a digital net or of the $t$-value of an (ordered) set of matrices. The number of matrices $s$ equals the dimensionality of the net, their number of columns $k$ relates to the cardinality of the net, while their number of rows $r$ relates to the number of significant digits for each point.

In the following, we consider only *projection-regular* digital nets, which correspond to full-rank generating matrices. This condition ensure that the projection of the digital net on any coordinate is $\{0, 1/n, 2/n, \ldots, (n-1)/n\}$.

The following proposition directly unfolds from the definition of a digital net.

**Proposition 2.6.** *For any integers $0 < m \leq k$, the restriction of a digital net $\mathcal{P}$ of cardinality $2^k$ to its first $2^m$ points yields a digital net. The generating matrices of the smaller net are the submatrices consisting of the first $m$ columns of the original net's generating matrices. The integer $m$ is called the* level *of the restriction.*

For many applications, the required number of points is not known in advance, thus QMC estimators need to be refined by increasing the number of points [14]. For this reason, we want to find digital nets whose restrictions at different levels are all highly-uniform. A figure of merit investigating the quality of a point set at different levels is called a multi-level figure of merit, and we present such a figure in the next section.

## 2.2 Multi-level $t$-value-based figure of merit

**Definition 2.7.** Given a set $S$ of cardinality $s$, we denote by $\mathfrak{u}$ a subset of $\{1, \ldots, s\}$ and by $S_\mathfrak{u}$ the corresponding subset of $S$.

**Definition 2.8.** For a matrix $M$, $M^{(r,m)}$ is the upper-left submatrix of $M$ containing the $r$ first rows and $m$ first columns.

**Definition 2.9.** For a set of $r \times k$ matrices $S = (M_1, \ldots, M_s)$, for a non-empty projection $\mathfrak{u} \subseteq \{1, \ldots, s\}$[1], for a level $m \in \{1, \ldots, k\}$, we denote by $t_m(S_\mathfrak{u})$ the $t$-value of the set of matrices $(M_l^{(r,m)})_{l \in \mathfrak{u}}$ at level $m$. This quantity depends on the level $m$ and coïncides with the usual $t$-value for $m = k$.

[1] In this report, $\subseteq$ denotes inclusion and $\subset$ denotes strict inclusion.

We are interested in the following class of figures of merit, parametrized by the positive norm type $q$, the function $h : \mathbb{N}^3 \mapsto \mathbb{R}$, the general positive weights [5] $(\gamma_{\mathfrak{u}})_{\emptyset \neq \mathfrak{u} \subseteq \{1,\dots,s\}}$ , and the integers $m_{min}, m_{max} \in [1, k]$:

$$D^{(q)}_{h,\gamma,m}(S) = \left( \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1,\dots,s\}} (\gamma_{\mathfrak{u}} h(|\mathfrak{u}|, m, t_m(S_{\mathfrak{u}})))^q \right)^{1/q} \tag{3}$$

$$D^{(q)}_{h,\gamma,m_{min},m_{max}}(S) = \max_{m_{min} \leq m \leq m_{max}} D^{(q)}_{h,\gamma,m}(S) \tag{4}$$

**Remark 2.10.** The class of figures of merit (4) is very general, and in particular encompasses the $t$-value as defined in 2.1, if one chooses $m_{min} = m_{max} = k$, $\gamma_{\mathfrak{u}} = \mathbf{1}_{\mathfrak{u}=\{1,\dots,s\}}$ and $h(|\mathfrak{u}|, m, t) = t$. Besides, this class of figures of merit is called *projection-dependent*, because the merit of each subprojection of $\{1, \dots, s\}$ is summed to obtain the total figure.

Special cases of this kind of figures of merit have already been investigated in [6] which proposed:

$$h^{(p)}(|\mathfrak{u}|, m, t) = \frac{t^p}{m - t + 1} \text{ and } \gamma_{\mathfrak{u}} = \begin{cases} 0.9999^{\min(\mathfrak{u})-1} & \text{if } |\mathfrak{u}| = 2, \\ 0 & \text{otherwise.} \end{cases}$$

where $\min(\mathfrak{u})$ denotes the value of the minimum coordinate in $\mathfrak{u}$.

Besides, (4) can be generalized by replacing the max operator by any accumulator, and in particular a sum. Moreover $q$ can be taken infinite, which means that the inner sum is replaced by a maximum.

In subsection 2.3, we explain intuitively why such a figure should work, while in subsection 2.4, we derive a bound of the form (1) related to this figure of merit.

## 2.3 Rationale behind a projection-dependent figure of merit

Looking at the convergence rate (2), one expects QMC methods to outperform Monte Carlo only in low dimension. Indeed, ensuring that

$$\frac{(\log n)^{s-1}}{n} \leq \frac{1}{\sqrt{n}}$$

requires at least that $n \geq 2^{s-1}$, which becomes infeasible in high dimension (more than 30).

Actually the reality is more nuanced, as in some applications, most of the variance of the problem is captured by its low-dimensional projections. More precisely, if the function $f(\boldsymbol{u}) = f(u_1, \dots, u_s)$ to be integrated is of finite variance, its so-called *ANOVA decomposition* is

$$f(\boldsymbol{u}) = \sum_{\mathfrak{u} \subseteq \{1,\dots,s\}} f_{\mathfrak{u}}(\boldsymbol{u}) \tag{5}$$

where each $f_{\mathfrak{u}}$ depends only on $\{u_i, i \in \mathfrak{u}\}$, and the $f_{\mathfrak{u}}$ functions are orthogonal and centered [15]. The Monte Carlo variance decomposes accordingly as

$$\sigma^2 := \operatorname{Var} f(\boldsymbol{U}) = \sum_{\mathfrak{u} \subseteq \{1,\dots,s\}} \sigma^2_{\mathfrak{u}}, \text{ where } \sigma^2_{\mathfrak{u}} = \operatorname{Var} f_{\mathfrak{u}}(\boldsymbol{U})$$

**Definition 2.11.** If we have, for some $\rho$ close to 1, and some $d \in \{1, \dots, s\}$,

$$\sum_{\substack{\mathfrak{u} \subseteq \{1,\dots,s\} \\ |\mathfrak{u}| \leq d}} \sigma_{\mathfrak{u}}^2 \geq \rho \sigma^2,$$

we say that that $f$ has effective dimension $d$ in the superposition sense.

This suggests that:

- the 'right' notion of dimension to characterize QMC efficiency is not the true dimension but the effective dimension: if the effective dimension is low enough, even if the true dimension is very high, QMC may efficiently reduce the variance (see sections 4.3 and 4.4 for examples).
- the figure of merit should weight each subprojection of $[0,1]^s$, accordingly to the part of the variance of $f$ explained by this projection, giving more importance to the projections which capture more variance of $f$. Indeed, we want to be more precise in our numerical integration of these projections.

This explains the interest of a projection-dependent figure of merit. Moreover, introducing weights gives latitude to tune more precisely the figure of merit depending on the structure of the function $f$, and hopefully obtain in the end a point set better adapted to the application.

## 2.4 Bound for the worst-case QMC integration error

For the figures of merit given by (3) and (4), we want to obtain an error bound for the QMC integration error. To do so, we follow [9] for definitions 2.12 and 2.13.

Here, for any projection $\mathfrak{u} \subseteq \{1, \dots, s\}$ and any vector $\boldsymbol{z} \in [0,1)^s$, let $\boldsymbol{z}_{\mathfrak{u}}$ denote the projection of $\boldsymbol{z}$ on $\mathfrak{u}$, let $\mathcal{P}_{\mathfrak{u}}$ denote the projection on $\mathfrak{u}$ of a set of points $\mathcal{P}$. By $(\boldsymbol{z}_{\mathfrak{u}}, 1)$, we mean the vector $\boldsymbol{z}$, with all components whose indices are not in $\mathfrak{u}$ replaced by 1.

**Definition 2.12.** By $W$ we denote the Sobolev space of functions on $[0,1)^s$ and let $f$ be in $W$. For positive weights $\gamma = (\gamma_{\mathfrak{u}})_{\mathfrak{u} \subseteq \{1,\dots,s\}}$, $W$ is equipped with the norm:

$$||f||_\gamma := \sum_{\mathfrak{u} \subseteq \{1,\dots,s\}} \gamma_{\mathfrak{u}}^{-1} \int_{[0,1]^{|\mathfrak{u}|}} \left| \frac{\partial^{|\mathfrak{u}|}}{\partial z_{\mathfrak{u}}} f(z_{\mathfrak{u}}, 1) \right| dz_{\mathfrak{u}}. \tag{6}$$

**Definition 2.13.** The weighted star-discrepancy $D^*_{n,\gamma}$ of a point set $\mathcal{P} = (x_0, \dots, x_{n-1}) \subseteq [0,1)^s$ is defined by

$$D^*_{n,\gamma}(\mathcal{P}) = \sup_{z \in [0,1)^s} \max_{\emptyset \neq \mathfrak{u} \subseteq \{1,\dots,s\}} \gamma_{\mathfrak{u}} |\operatorname{disc}(z_{\mathfrak{u}}, 1)|,$$

where for $z = (z_1, \dots, z_s)$,

$$\operatorname{disc}(z_{\mathfrak{u}}) = \prod_{i \in \mathfrak{u}} z_i - \frac{1}{n} \left| \left\{ i : x_{i,\mathfrak{u}} \in \prod_{i \in \mathfrak{u}} [0, z_i) \right\} \right|.$$

This discrepancy measures the weighted maximal difference between the proportion of points in each projection of a hypercube $[0, z)$ and the volume of this projection. It is a generalization of the (non-weighted) star-discrepancy where all the weights are taken equal to 1, and which we denote by $D^*_n(\mathcal{P})$.

Now we can give an upper bound for the integration error, namely the generalized Koksma-Hlawka inequality [24]:

**Proposition 2.14.** *For every $f \in W$, for every point set $\mathcal{P} = (x_0, \dots, x_{n-1}) \subseteq [0,1)^s$,*

$$\left| \int_{[0,1]^s} f(u)du - \frac{1}{n} \sum_{i=0}^{n-1} f(x_i) \right| \leq ||f||_\gamma D^*_{n,\gamma}(\mathcal{P})$$

This Hölder-like inequality allows to separate the integration error into two terms: the variation $||f||_\gamma$ which only depends on the integrand $f$, and the weighted star-discrepancy $D^*_{n,\gamma}(\mathcal{P})$ which only depends on the point set $\mathcal{P}$.

Furthermore, we have the following estimate for $D^*_{n,\gamma}(\mathcal{P})$ linking the weighted star-discrepancy to the non-weighted star-discrepancy of every projection [4, eq. 5.18], in the case of a digital net:

**Proposition 2.15.** *Let $\Delta(t,k,s)$ be such that for any $(t,k,s)$-net $\mathcal{P}$, the inequality $2^k D^*_{2^k}(\mathcal{P}) \leq \Delta(t,k,s)$ holds. Then, for the weighted star discrepancy $D^*_{2^k,\gamma}$, we have:*

$$2^k D^*_{2^k,\gamma}(\mathcal{P}) \leq \max_{\emptyset \subset \mathfrak{u} \subseteq \{1,\dots,s\}} \gamma_\mathfrak{u} \Delta(t(\mathcal{P}_\mathfrak{u}), k, |\mathfrak{u}|)$$

The last ingredient is to derive a bound on the star discrepancy of a $(t,k,s)$-net. This is done thanks to the proposition below [4, corollary 5.3]:

**Proposition 2.16.** *The star discrepancy of a $(t,k,s)$-net $\mathcal{P}$ satisfies:*

$$2^k D^*_{2^k}(\mathcal{P}) \leq 2^t \sum_{i=0}^{s-1} \binom{k-t}{i}$$

Recall that the goal is to obtain a bound on the integration error by some figure of merit easy to compute. These figures will be used to select the best performing net in *LatNet Builder*, as we hope that a low figure of merit translates into a low integration error. Propositions 2.14 to 2.16, along with equation (3), give the following bounds on the integration error:

**Theorem 2.17.** *For every $f \in W$, for every set $\gamma$ of positive weights, for every set of $k \times k$ matrices $S = (M_1, \dots, M_s)$, for every levels $m_{min}, m_{max} \in [1,k]$, let $\mathcal{P}_m$ be the digital net (of cardinality $2^m$) defined by the matrices $(M_1^{(m,m)}, \dots, M_s^{(m,m)})$. Then*

$$\left| \int_{[0,1]^s} f(u)du - \frac{1}{2^m} \sum_{x \in \mathcal{P}_m} f(x) \right| \leq ||f||_\gamma D^{(+\infty)}_{h_*,\gamma,m}(S) \tag{7}$$

$$\leq ||f||_\gamma \left( \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1,\dots,s\}} (\gamma_\mathfrak{u} h(|\mathfrak{u}|, m, t_m(S_\mathfrak{u})))^q \right)^{1/q} \tag{8}$$

$$\max_{m_{min} \leq m \leq m_{max}} \left| \int_{[0,1]^s} f(u)du - \frac{1}{2^m} \sum_{x \in \mathcal{P}_m} f(x) \right| \leq ||f||_\gamma D^{(+\infty)}_{h_*,\gamma,m_{min},m_{max}}(S) \tag{9}$$

*where* $D^{(+\infty)}_{h,\gamma,m}$ *and* $D^{(+\infty)}_{h,\gamma,m_{min},m_{max}}$ *are defined in (3) and (4) and* $h_*$ *is the function defined by*

$$h_*(|\mathfrak{u}|, m, t) = 2^{t-m} \sum_{i=0}^{|\mathfrak{u}|-1} \binom{m-t}{i}.$$

**Remark 2.18.** In the next part, to speed up the computations, we will want to take some weights equal to zero. In this case, the previous theorem is still valid at the expense of restricting the integrand to a smaller space of functions: in (6), we see that for each projection $\mathfrak{u}$, $\gamma_\mathfrak{u}$ can be taken zero if and only if $\int_{[0,1]^{|\mathfrak{u}|}} \left| \frac{\partial^{|\mathfrak{u}|}}{\partial z_\mathfrak{u}} f(z_\mathfrak{u}, 1) \right| dz_\mathfrak{u} = 0$.

## 2.5 Exploring the space of digital nets

Now that we have defined a figure of merit, the goal is to optimize it over the space of digital nets, that is over all possible generating matrices in $GL_k(\mathbb{F}_2)^s$. We call this space of generating matrices the *search space* or the *exploration space*. To perform this task, we resort to a computer search.

As explained in the introduction, this space is mind-boggingly enormous, so an exhaustive exploration is impossible. To tackle this problem, two complementary solutions are implemented in *LatNet Builder*: restricting the search space to include only some particular constructions of digital nets, and finding more efficient exploration methods than an exhaustive search. In what follows, we briefly investigate some of the constructions and exploration methods implemented in *LatNet Builder*.

### 2.5.1 Construction methods

Two main construction methods are implemented in *LatNet Builder*:

*Polynomial lattice rules* [18] are digital nets where the generating matrices $M_1, \dots, M_s$ are so-called 'Hankel' matrices, that is of the form:

$$M_j = \begin{bmatrix} u_1^{(j)} & u_2^{(j)} & \cdots & u_k^{(j)} \\ u_2^{(j)} & u_3^{(j)} & \iddots & u_{k+1}^{(j)} \\ \vdots & \iddots & \iddots & \vdots \\ u_k^{(j)} & u_{k+1}^{(j)} & \cdots & u_{2k-1}^{(j)} \end{bmatrix}.$$

A digital net with Hankel generating matrices coïncides with a lattice in the space of rational functions over $\mathbb{F}_2$, hence the name of 'polynomial lattice rule'.

To generate $2^{10}$ points in dimension 5, the polynomial lattice rule search space is of cardinality $2^{19\times 5} = 2^{95} \simeq 10^{29}$, which is already much smaller than the cardinality of the whole search space.

This construction is mentioned here for the sake of completness, and because we use it in our numerical experiments in section 4. For details on the use of polynomial lattice rules in *LatNet Builder*, we refer to the report of Maxime Godin.

*Sobol digital nets* [6, 25] are defined as follow: to generate the $j$-th component of the points, we need to choose a primitive polynomial of degree $s_j$ over $\mathbb{Z}_2$, which we write

$$x^{s_j} + a_{1,j}x^{s_j-1} + a_{2,j}x^{s_j-2} + \cdots + a_{s_j-1,j}x + 1$$

Define the sequence of positive integers $\{m_{1,j}, m_{2,j}, \dots\}$ by the recurrence relation:

$$m_{r,j} = 2a_{1,j}m_{r-1,j} \oplus 2^2 a_{2,j}m_{r-2,j} \oplus \cdots \oplus 2^{e_j-1}a_{e_j-1,j}m_{r-e_j+1,j} \oplus 2^{e_j}m_{r-e_j,j} \oplus m_{r-e_j,j}, \quad (10)$$

where $\oplus$ is the bitwise exclusive-or operator.

The $\{m_{1,j}, m_{2,j}, \dots\}$ are called the direction numbers for coordinate $j$. The initial values $m_{1,j}, \dots, m_{e_j,j}$ can be chosen freely provided that each $m_{r,j}$, $1 \leq r \leq e_j$ is odd and less than $2^r$. As a consequence, all the direction numbers $m_{r,j}$ are odd and less than $2^r$. We can write $m_{r,j} = \sum_{l=1}^{r} m_{r,j,l}2^{r-l}$. Then, for any positive integer $k$, the $j$-th generating matrix $M_j$ of size $k$ can be written:

$$M_j = \begin{bmatrix} 1 & m_{2,j,1} & m_{3,j,1} & \cdots & m_{k,j,1} \\ 0 & 1 & m_{3,j,2} & \cdots & m_{k,j,2} \\ 0 & 0 & 1 & \cdots & m_{k,j,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}. \quad (11)$$

LatNet Builder implements the same choice of primitive polynomials as [6], that is to sort the polynomials by increasing degree (see also section 2.6).

Searching for the best Sobol digital net means searching the best direction numbers $m_{1,j}, \dots, m_{e_j,j}$, for each $j \in \{1, \cdots, s\}$. This drastically reduces the size of the search space, especially in low dimension. For instance, it turns out that the size of the Sobol search space to generate an arbitrary number of points in dimension 5 is only $2^7$.

### 2.5.2 Exploration methods

The *LatNet Builder* software supports, among others, the following exploration methods:

- Random search of the exploration space.
- Component-By-Component (CBC) greedy search. This method was introduced in [23], and is massively used in the QMC literature. Starting from an empty net, one coordinate is added at a time, by minimizing the figure of merit over all possible generating matrices for this new coordinate. Once the matrix for this coordinate is chosen, we greedily move to the next coordinate, without ever going back.
- Random Component-By-Component search is very similar to the CBC search, except that a random subset of all matrices is sampled at each step instead of the whole set of matrices.
- Mixed Component-By-Component search is an intermediate where CBC is used for the first coordinates then random CBC is used for the last coordinates.

One may wonder why primitive optimization algorithms (such as random seach) are used instead of more efficient algorithms. The latter was seldom attempted in the QMC literature. For instance, [21] uses evolutionary algorithms to find low-discrepancy point sets. However, these results were not convincing insofar as their point set was not directly benchmarked with a point set found with a simple random seach and a similar budget. Thus it remains unclear if their evolutionary algorithm performs really better than a random search. We believe the lack of good optimization algorithms is explained by the lack of local structure of this optimization problem (of the figure of merit over the space of digital nets).

## 2.6 Sequence of points and convergence rate

We announced in the introduction a convergence rate for QMC methods of $\mathcal{O}(n^{-1}(\log n)^{s-1})$. To obtain this convergence rate, we would like to make $k$ and $m_{max}$ go to infinity. To formalize this idea, we introduce the notion of digital sequence.

**Definition 2.19.** A *digital sequence* is the (infinite) point set obtained by taking $r = k = +\infty$ in definition 2.3, that is by having infinite generating matrices.

In particular, Sobol digital nets (defined in section 2.5.1) are in fact digital sequences, because the definition (11) is valid for any $k$, thanks to the recurrence relation (10). We now focus on this particular construction, to obtain the announced convergence rate. Sobol [25] proved the following proposition that gives a garantee on the quality of his digital nets:

**Proposition 2.20.** *For a given dimension $s$, choose $s$ primitive polynomials of degree $s_j$ over $\mathbb{Z}_2$. Then for any initial direction numbers, and for any number of points, the $t$-value of the resulting Sobol digital net is bounded as follows*

$$t \leq \sum_{j=1}^{s} (s_j - 1) \tag{12}$$

This explains why the primitive polynomials should be chosen of the smallest possible degree, which is the choice implemented in *LatNet Builder*.

**Lemma 2.21.** *For fixed integers $t \geq 0$, $s > 0$, when $m$ goes to infinity,*

$$2^{t-m} \sum_{i=0}^{s-1} \binom{m-t}{i} = \mathcal{O}\left(\frac{m^{s-1}}{2^m}\right)$$

*Proof.* When $m \to +\infty$, the sum is majorated by $s$ times its last term, i.e.

$$2^{t-m} \sum_{i=0}^{s-1} \binom{m-t}{i} = \mathcal{O}\left(2^{t-m} s \binom{m-t}{s-1}\right) = \mathcal{O}\left(2^{-m} \binom{m}{s-1}\right) = \mathcal{O}\left(2^{-m} m^{s-1}\right)$$

□

The two previous results along with Theorem 2.17 above give the following result:

**Proposition 2.22.** *For every choice of initial direction numbers, for every $m > 0$, let $\mathcal{P}_m$ be the resulting Sobol digital net (of cardinality $2^m$) and let $S_m$ be the resulting set of matrices. Then for every $f \in W$, for every set $\gamma$ of positive weights, when $m$ goes to infinity,*

$$\left| \int_{[0,1]^s} f(u)du - \frac{1}{2^m} \sum_{x \in \mathcal{P}_m} f(x) \right| = \mathcal{O}\left(\frac{m^{s-1}}{2^m}\right)$$

*Proof.* (7) directly yields

$$\left| \int_{[0,1]^s} f(u)du - \frac{1}{2^m} \sum_{x \in \mathcal{P}_m} f(x) \right| = \mathcal{O}(D^{(+\infty)}_{h,\gamma,m}(S))$$
$$= \mathcal{O}\left( \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1,\dots,s\}} \gamma_{\mathfrak{u}} 2^{t_m(S_{\mathfrak{u}})-m} \sum_{i=0}^{|\mathfrak{u}|-1} \binom{m - t_m(S_{\mathfrak{u}})}{i} \right)$$

In each term of the sum, $t_m(S_{\mathfrak{u}})$ can be majored by the constant defined in (12). Then an application of lemma 2.21 to each term yields the result by bounding $|\mathfrak{u}|$ by $s$. □

**Remark 2.23.** In the figure of merit (4), a maximum is taken over a range of values of $m$. But the result above shows that the order of magnitude of the quantity $D^{(+\infty)}_{h,\gamma,m}(S)$ decreases with $m$. Thus it actually makes more sense to *renormalize* the figure of merit and to consider, instead of (4) the figure

$$\tilde{D}^{(q)}_{h,\gamma,m_{min},m_{max}}(S) = \max_{m_{min} \leq m \leq m_{max}} \frac{2^m}{m^{s-1}} D^{(q)}_{h,\gamma,m}(S).$$

The hope is that every term in the maximum has the same order of magnitude, so it actually makes sense to compare them and take their maximum.

## 2.7 Randomized quasi-Monte Carlo

The bounds obtained previously on the integration error, such as theorem 2.17 or proposition 2.22, apply to deterministic point sets. The framework of random sampling with variance estimations and confidence intervals is lost. *Randomized quasi-Monte Carlo (RQMC) sampling* allows to regain it by re-randomizing QMC point sets in a way that preserves their equidistribution properties. The underlying QMC point set does not change, but is randomized according to some randomization rule.

Several randomizations methods are studied in the RQMC literature. Here we focus on one particular method, *digital shift* [4, p. 159], that works well with digital nets.

**Definition 2.24.** For $x = \sum_{i=1}^{\infty} \xi_i 2^{-i} \in [0,1)$ and $\sigma = \sum_{i=1}^{\infty} \zeta_i 2^{-i} \in [0,1)$, we define the *digitally shifted point* $y$ by

$$y = x \oplus \sigma := \sum_{i=1}^{\infty} (\xi_i + \zeta_i) 2^{-i}$$

where the '+' is addition in $\mathbb{F}_2$.

For a point set $\mathcal{P} = (\boldsymbol{x}_0, \dots, \boldsymbol{x}_{n-1}) \subset [0,1)^s$, and a $\boldsymbol{\sigma} \in [0,1)^s$, the point set $\mathcal{P}_\sigma = (\boldsymbol{x}_0 \oplus \boldsymbol{\sigma}, \dots, \boldsymbol{x}_{n-1} \oplus \boldsymbol{\sigma}) \subset [0,1)^s$ is called the (2-adic) *digitally shifted version of* $\mathcal{P}$. The vector $\boldsymbol{\sigma} \in [0,1)^s$ is called a (2-adic) *digital shift*.

The definition 2.1 of $(t,k,s)$-nets yields the following nice property:

**Lemma 2.25.** *For any digital shift $\boldsymbol{\sigma} \in [0,1)^s$, and any $(t,k,s)$-net $\mathcal{P}$, the digitally shifted version of $\mathcal{P}$ remains a $(t,k,s)$-net. In other words, the $t$-value is digital-shift invariant.*

This entails in particular than our figure of merit (3) is also digital-shift invariant. This means the figure of merit also furnishes a bound for the variance:

**Proposition 2.26.** *For every $f \in W$, for every set $\gamma$ of positive weights, for every choice of initial direction numbers, for every $m > 0$, let $\mathcal{P}_m$ be the resulting Sobol digital net (of cardinality $2^m$). Let*

$$\mathrm{Var}\left(\hat{\mu}_{2^m,\mathrm{rqmc}}\right) = \mathbb{E}_\sigma\left[\left|\int_{[0,1]^s} f(u)du - \frac{1}{2^m}\sum_{x\in\mathcal{P}_{\sigma,m}} f(x)\right|^2\right]$$

*be the variance of the RQMC estimator over all possible digital shifts. Then the variance converges with the following rate*

$$\mathrm{Var}\left(\hat{\mu}_{2^m,\mathrm{rqmc}}\right) = \mathcal{O}\left(D^{(+\infty)}_{h,\gamma,m,m}(\mathcal{P}_m)^2\right) = \mathcal{O}\left(\frac{m^{2(s-1)}}{4^m}\right)$$

*Proof.* This is a simple consequence of theorem 2.17, proposition 2.22, and lemma 2.25. □

This rate of convergence is to be compared with the usual $\mathcal{O}(1/2^m)$ for MC sampling.

**Remark 2.27.** In sections 2.4 to 2.7, we chose to focus on one specific figure of merit, and how it could be used to bound the star discrepancy and thus the error of integration. Nevertheless, many other discrepancies and bounds can be defined, leading to other figures of merit that are also of the general form (3). For instance [4, p. 557], if one takes $q = 1$ and

$$h_2(|\mathfrak{u}|, m, t) = \frac{1}{6} + 4^t(m-t)^{|\mathfrak{u}|-1},$$

the resulting figure of merit $D^{(1)}_{h_2,\gamma,m}$ also bounds the variance of a discrepancy function (the weighted L2-discrepancy) and thus of the integration error:

$$\mathrm{Var}\left(\hat{\mu}_{2^m,\mathrm{rqmc}}\right) = \mathcal{O}\left(D^{(1)}_{h_2,\gamma,m}(\mathcal{P}_m)^2\right) = \mathcal{O}\left(\frac{m^{2(s-1)}}{4^m}\right)$$

# 3 Efficient computation of the figure of merit

In this section, a new method to compute efficiently the figure of merit (4) is presented. In many use cases, this method turns out to be much quicker than the previously known techniques. For instance, our implementation allows to compute (4) for a digital net with $2^{21}$ points in dimension 10 in approximately 2 seconds.

This novel algorithm will probably be the subject of a paper to be published.

## 3.1 A linear algebra expression of the $t$-value

The $t$-value is defined for general nets but in the case of digital nets, it can be reformulated as a constructive linear algebra problem. Let $s$ be the dimension of the net, and $M_1, ...M_s$ the $k \times k$ generating matrices.

**Definition 3.1.** A (strong) *composition of $r$ in $s$ parts* is a tuple of strictly positive integers $d_1, ..., d_s$ such that $\sum_{l=1}^s d_l = r$. There are $\binom{r-1}{s-1}$ such compositions [7]. A weak composition of $r$ in $s$ parts is a tuple of non-negative integers $d_1, ..., d_s$ such that $\sum_{l=1}^s d_l = r$.

**Definition 3.2.** For a set $S = \{M_1, ..., M_s\}$, for $m \in [1, k]$, the *linear independence parameter* $\rho_m(S)$ is the maximum integer $r$ such that for every *weak* composition $(d_1, ..., d_s)$ of $r$ in $s$ parts, the $d_1$ first rows of $M_1^{(m,m)}$, the $d_2$ first rows of $M_2^{(m,m)}$, ..., up to the $d_s$ first rows of $M_s^{(m,m)}$ are all linearly independent. This linear independence can be checked by computing the rank of the block matrix

$$C = \begin{bmatrix} M_1^{(d_1,m)} \\ M_2^{(d_2,m)} \\ \cdots \\ M_s^{(d_s,m)} \end{bmatrix}.$$

In what follows, we call this block matrix the *composition matrix*. If the composition matrix is full-rank, the composition is called *admissible* with respect to $S$ at level $m$ [4] (p. 200).

The following proposition, demonstrated in Theorem 4.52 (page 150) of [4], boils down the computation of the $t$-value to a linear algebra problem:

**Theorem 3.3.** *For a set $S = \{M_1, ..., M_s\}$ and for $0 < m \leq k$, the $t$-value $t_m(S)$ of $S$ at level $m$ equals $m - \rho_m(S)$.*

Recall that we want to compute the figure of merit given by (4). For the weights $\gamma_{\mathfrak{u}}$, we choose *finite-order weights* [4, p. 95] of fixed order $d$, which are weights equal to zero for every subset $\mathfrak{u}$ of cardinality strictly greater than $d$ (for instance, [6] takes $d = 2$). **Thus the computation of the figure of merit boils down to computing $\rho_m(S_{\mathfrak{u}})$, for every $\mathfrak{u}$ such that $|\mathfrak{u}| \leq d$, and every $m \in [m_{min}, m_{max}]$.**

## 3.2 A dynamic programming equation for the linear independence parameter

We now want to derive an efficient dynamic programming method to compute $\rho_m(S_{\mathfrak{u}})$, for every $\mathfrak{u}$ such that $|\mathfrak{u}| \leq d$, and every $m \in [m_{min}, m_{max}]$.

We propose a dynamic programming algorithm to compute the linear independence parameter of a set of matrices $S$, given the linear independence parameter of every strict subset of $S$.

**Definition 3.4.** For a set $S = \{M_1, ..., M_s\}$, for $0 < m \leq k$, the partial linear independence parameter $\tilde{\rho}_m(S)$ is the maximum integer $r$ such that for every *strong* composition $d_1, ..., d_s$ of $r$ in $s$ parts, the $d_1$ first rows of $M_1^{(m,m)}$, the $d_2$ first rows of $M_2^{(m,m)}$, ..., up to the $d_s$ first rows of $M_s^{(m,m)}$ are all linearly independent.

**Remark 3.5.** Note that $\rho_m(S) \leq \tilde{\rho}_m(S) \leq m$, and $\rho_m$ and $\tilde{\rho}_m$ both increase with $m$ (for a fixed $S$).

This notion of partial linear independence allows us to formulate the computation of $\rho_m$ as a dynamic programming problem:

**Proposition 3.6.** *For each non-empty subset $\mathfrak{u} \subseteq \{1, \ldots, s\}$, for $0 < m \leq k$, the following dynamic programming equations hold:*

$$\rho_m(S_\mathfrak{u}) = \min\left(\min_{\mathfrak{v}\subset\mathfrak{u}}\{\rho_m(S_\mathfrak{v})\}, \tilde{\rho}_m(S_\mathfrak{u})\right) \tag{13}$$

$$= \min\left(\min_{\substack{\mathfrak{v}\subset\mathfrak{u}\\|\mathfrak{v}|=|\mathfrak{u}|-1}}\{\rho_m(S_\mathfrak{v})\}, \tilde{\rho}_m(S_\mathfrak{u})\right) \tag{14}$$

*with the convention* $\rho_m(\emptyset) = 0$.

*Proof.* The first equality is well-known in the litterature, for instance it is the basis of the algorithm to compute the $t$-value in [22]. The second equality unfolds from the first by noting that $S \to \rho_m(S)$ is non-increasing for the inclusion partial order. □

To compute $\rho_m(S_\mathfrak{u})$, for every $\mathfrak{u}$ such that $|\mathfrak{u}| \leq d$, and every $m \in [m_{min}, m_{max}]$, we first compute $\rho_m(\{M\})$ for all $M \in S$ and all $m$. Then, we use (14) in the following manner: traverse the subsets of $\{1, \ldots, s\}$ by increasing cardinality until reaching cardinality $d$. For each subset $\mathfrak{u}$, we compute $r_{max} = \min_{\mathfrak{v}\subset\mathfrak{u},|\mathfrak{v}|=|\mathfrak{u}|-1}\{\rho_{m_{max}}(S_\mathfrak{v})\}$, then compute $\rho_m(S_\mathfrak{u})$ for all $m$ by testing, by decreasing values of $r$ starting from $r_{max}$, if $\tilde{\rho}_m(S_\mathfrak{u}) \geq r$.

**Definition 3.7.** Let $m_{adm}(S, \mathfrak{u}, r)$ be the minimal level such that every composition of $r$ in $|\mathfrak{u}|$ parts is admissible with respect to $S$ at that level, or $+\infty$ if this condition is not verified for any level. In what follows, we call this quantity the *all-admissible level*. When there is no ambiguity, we will denote it $m_{adm}(\mathfrak{u}, r)$ or $m_{adm}(r)$.

More precisely, our algorithm consists in computing $m_{adm}(\mathfrak{u}, r)$ for decreasing values of $r$. Using the fact that for a given subset $S$, $\tilde{\rho}_m$ increases with $m$, we obtain that $\forall m \geq m_{adm}(\mathfrak{u}, r)$, $\tilde{\rho}_m(S_\mathfrak{u}) \geq r$, thus:

$$\forall r \leq r_{max},\ \forall m \in [m_{adm}(\mathfrak{u}, r), m_{adm}(\mathfrak{u}, r+1)[,\ \rho_m(S_\mathfrak{u}) = r$$

with the convention $m_{adm}(r_{max}+1) = +\infty$. Algorithm 1 states the procedure explained above, where AllAdmissibleLevel($S$, $\mathfrak{u}$, $r$) computes $m_{adm}(S, \mathfrak{u}, r)$.

The difficult step is the computation of $m_{adm}(r)$, and our main contribution is a procedure to do so efficiently, which we describe next.

## 3.3 Computation of the all-admissible level

**Definition 3.8.** For a given composition matrix $C$, the smallest level such that the composition is admissible at that level is noted $m(C)$. If $C$ is not full rank, $m(C)$ is defined as $+\infty$.

A naive way to compute $m_{adm}(S, \mathfrak{u}, r)$ would be to compute all composition matrices, then compute $m(C)$ for each composition matrix $C$. $m_{adm}(r)$ is the maximum over all compositions of these numbers $m(C)$. Given $C$, to compute $m(C)$, two strategies were envisioned in the literature [20, 22]:

- Strategy S (for Schmid): enumerate all the possible combinations of the rows of $C$. There are $2^r$ such combinations in the $\mathbb{F}_2$ vector space of rows.
- Strategy PS (for Pirsic & Schmid): use the Gauss reduction algorithm on $C$, with the constraint that the columns should remain unpermuted.

**Algorithm 1** Computation of $\rho_m(S_{\mathfrak{u}})$ for all $m \in [m_{min}, m_{max}]$ and all $\mathfrak{u}$ s.t. $|\mathfrak{u}| \leq d$

**Input**: A set of matrices $S$, a maximal cardinality $d$, numbers of columns $m_{min}$ and $m_{max}$
**Output**: The independence parameters $\rho_m(S_{\mathfrak{u}})$ for all $m \in [m_{min}, m_{max}]$ and all $\mathfrak{u}$ s.t. $|\mathfrak{u}| \leq d$

**function** ComputeRho($S$, $d$, $m_{min}$, $m_{max}$)

% case of subsets of cardinality 1
**for** $l \in \{1, \ldots, s\}$ **do**
Let $M_l$ be the $l$-th matrix of $S$
**for** $m \in \{m_{min}, ..., m_{max}\}$ **do**
$\rho_m(\{M_l\}) = \text{rank}(M_l^{(m,m)})$ % can be computed efficiently (see below)
**end for**
**end for**

% case of subsets of cardinality 2 to $d$
**for** $l \in \{2, \ldots, d\}$ **do**
**for** $\mathfrak{u} \subseteq \{1, \ldots, s\}$, with $|\mathfrak{u}| = l$ **do**
Let $r_{max} = \min_{\mathfrak{v} \subset \mathfrak{u}, |\mathfrak{v}| = l-1} \{\rho_{m_{max}}(S_{\mathfrak{v}})\}$
Let $m_{adm}(r_{max} + 1) = +\infty$
**for** $r \in \{r_{max}, ..., l\}$ **do**
Let $m_{adm}(r)$ = AllAdmissibleLevel($S$, $\mathfrak{u}$, $r$)
**if** $m_{adm}(r) < m_{adm}(r+1)$ **then**
**for** $m \in \{m_{adm}(r), ..., m_{adm}(r+1) - 1\}$ **do**
$\rho_m(S_{\mathfrak{u}}) = \min\left(\min_{\mathfrak{v} \subset \mathfrak{u}, |\mathfrak{v}| = |\mathfrak{u}|-1} \{\rho_m(S_{\mathfrak{v}})\}, r\right)$ % equation (14)
**end for**
**end if**
**if** $m_{adm}(r) \leq m_{min}$ **then** % the output at step $l$ has been fully computed
**break**
**end if**
**end for**
**end for**
**end for**

**return** $\rho_m(S_{\mathfrak{u}})$ for all $m \in [m_{min}, m_{max}]$ and all $\mathfrak{u}$ s.t. $|\mathfrak{u}| \leq d$
**end function**

As the number of compositions grows very fast with $r$ and $|\mathfrak{u}|$, these simple approaches do not scale well. In the following, we develop a new approach (strategy MG for Marion & Godin) to spare and reuse computations, which is an improvement of strategy PS.

Let's start by remarking that some composition matrices will be very similar, for instance some only differ by one row. It turns out that an intelligent traversal of the compositions allow to visit all the composition matrices in a way such that two consecutive composition matrices differ by exactly one row [7]. Therefore, it is possible to reuse some computations from a composition matrix to the next one. More precisely, we use an adaptation from the classical Gauss reduction algorithm [1] that allows to efficiently change a row in a reduced matrix.

In the remainder of this subsection, we describe the general idea of our Gauss reduction algorithm, then present the procedure AllAdmissibleLevel.

For a matrix $C$, our Gauss reduction algorithm uses row operations (row transvections and permutations) to find a row operation matrix $L$ and a reduced matrix $T$ such that:

- $L \times C = T$
- $L$ is an invertible matrix
- $T$ is in reduced row echelon form [1]: each pivot (the first nonzero number from the left in each row) is the only non-zero entry in its column.

To find $L$ and $T$, we successively 'pivot' on each row, from the first to the last row. To pivot on a row means (1) use transvections to zero all coefficients of the row which lay on a column with a pivot, (2) find the first non-zero coefficient on the row, which becomes a pivot, and (3) use transvections again to zero all coefficients on the column of this new pivot. The method Pivot($L$, $C$, $T$, $i$) does the operation of pivoting on the $i$-th row, and returns the matrices $L$ and $T$ modified by the pivoting operation.

An easy consequence of the algorithm is that the rank of $C$ equals the number of pivots in $T$. Even better, the rank of the submatrix formed by the $m$ first columns of $C$ equals the number of pivots in the $m$ first columns of $T$, due to the fact, that at step (2) above, we define the new pivot as the *first* non-zero coefficient.

These properties thus allow to compute $m(C)$ very quickly, once $T$ is known: $m(C)$ is $+\infty$ if there are less pivots than rows, else the column index of the right-most pivot. AdmissibleLevel($T$) does this computation, and returns $m(C)$.

Iterating through the composition matrices, from a composition matrix $C$, the next composition matrix is a matrix $C'$ such that $C$ and $C'$ only differ by a row: that is, we need to remove a row from $C$ and add a new row $R$ to get $C'$. Let's remark that the order of the rows in the composition matrices does not matter for our computations, so we can switch the order as needed. In particular, it is licit to add the new row in $C$ at any position; thus we will always add the new row at the place where the old row was removed, for ease of explanation and computation. At the same time that we modify $C$ to $C'$, we also need to modify $L$ and $T$ to $L'$ and $T'$ to preserve the three properties listed above, thus allowing us to compute $m(C')$ and continue iterating through the compositions.

This modification procedure consists in three steps, which are described in Figure 2, where we note $i$ the index of the row in $C$ we want to exchange; the procedure ExchangeRow($L$, $C$, $T$, $i$, $R$) does this operation and returns the new triplet ($L$, $C$, $T$). *The*

*main interest of this algorithm is that it calls* PIVOT *only once (at the end of Step 3), whereas a full recomputation of $T$ would require $r$ calls to* PIVOT. This is the profound reason that justifies the efficiency of this procedure.

We are now able to present procedure ALLADMISSIBLELEVEL($S$, $\mathfrak{u}$, $r$), which computes $m_{adm}(S, \mathfrak{u}, r)$. We start from a given fixed composition of $r$ into $|\mathfrak{u}|$ parts[2]. We compute the composition matrix $C$ and the reduced matrix $T$ for this composition. Then we cycle through all the possible compositions, updating the reduction as described above for each composition, and computing at each step $m(C)$. The traversal of all compositions is done using a Gray code for compositions, described in [7]. Finally we return $m_{adm}(S, \mathfrak{u}, r) = \max m(C)$. Algorithm 2 presents this procedure.

**Algorithm 2** Computation of the all-admissible level $m_{adm}(S, \mathfrak{u}, r)$

**Inputs**: A set of matrices $S$, a projection $\mathfrak{u}$, a number of rows $r$
**Output**: The all-full-rank width $m_{adm}(S, \mathfrak{u}, r)$

```
function AllAdmissibleLevel(S, r, m_min)
    Initialize C to a given composition matrix, T to C, L to identity
    for i ∈ {1, ..., r} do
        (L, T) = Pivot(L, C, T, i)
    end for        % T is now in reduced row echelon form
    Let m(C) = AdmissibleLevel(T, r)
    if m(C) = +∞ then        % m_adm(S, u, r) ≥ m(C) = +∞
        return +∞
    end if
    while there is an unvisited composition do
        Visit this composition, i.e. find the index i of the row to remove in T, and the
    value of the new row R
        (L, C, T) = ExchangeRow(L, C, T, i, R)
        Let m(C) = AdmissibleLevel(T, r)
        if m(C) = +∞ then        % m_adm(S, u, r) ≥ m(C) = +∞
            return +∞
        end if
    end while
    return max_C m(C)
end function
```

**Remark 3.9.** For concision, we do not present here pseudo-code for the procedures PIVOT, ADMISSIBLELEVEL, and EXCHANGEROW, as they were thoroughly described previously.

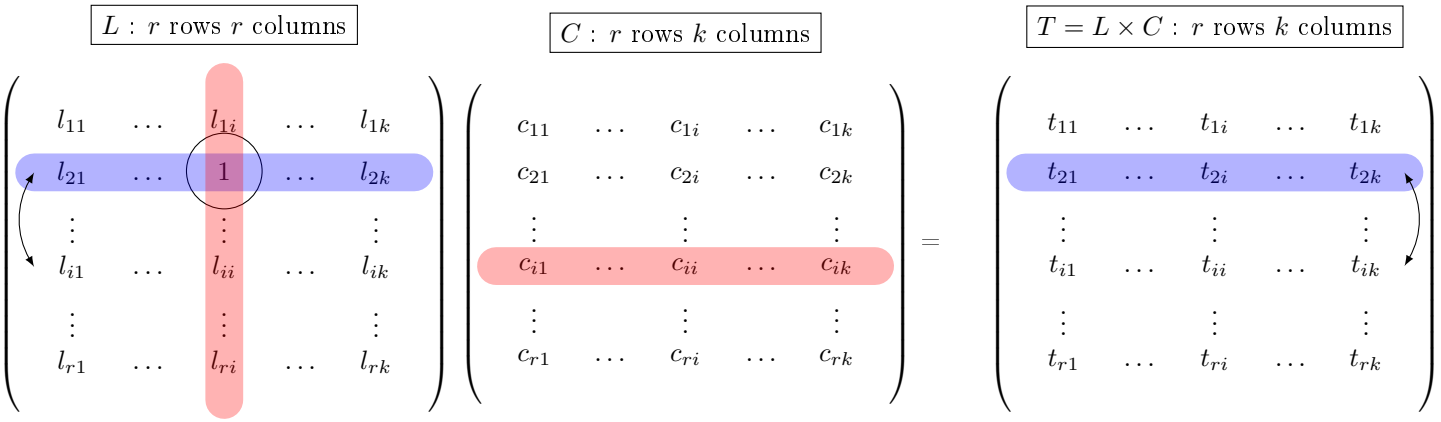
## 3.4 Complexity of the computation of the multi-level $t$-value

For the Gauss reduction procedures, an efficient implementation using adapted data structures to save computations lead to the following complexities:

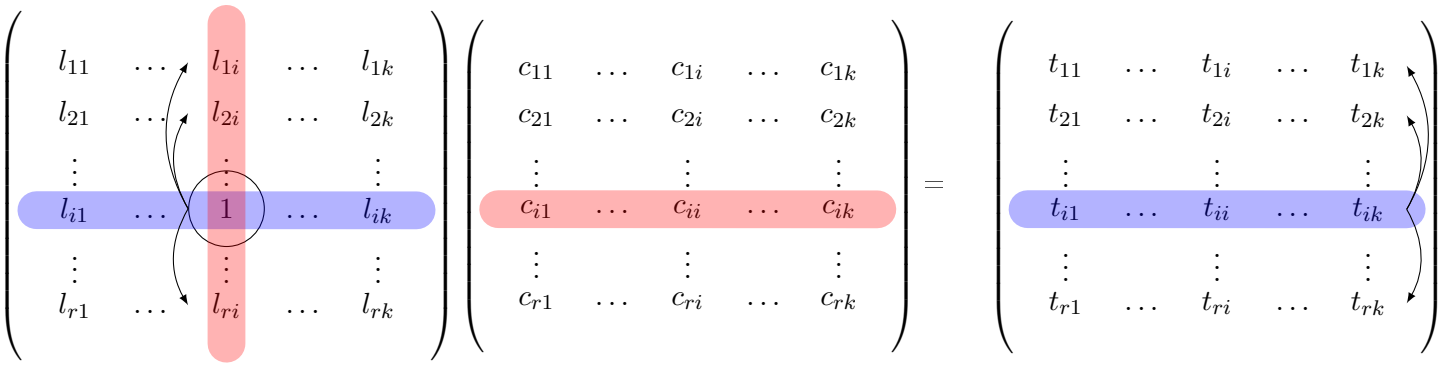
- for PIVOT and EXCHANGEROW: $\mathcal{O}(r + k)$

[2] In the implementation, this composition is $(r - |\mathfrak{u}| + 1, 1, \ldots, 1)$.

Step 1: Find a row with a now-zero coefficient in column $i$ of $L$, and exchange this row (blue row) and row $i$, in $L$ and in $T$.
The red row is the row we want to remove in $C$, and the red column is the associated column of $L$ in the $L \times C$ matrix multiplication: the elements of the red row are multiplied exactly by the elements of the red column in the matrix multiplication.




Step 2: Use the 1 at position $(i, i)$ in $L$ to zero all coefficients on column $i$ of $L$, using transvections.



Step 3: Replace row $i$ of $C$ and $T$ with the new row $c'_i$ (row in green). Set all coefficients of the row $i$ of $L$ to 0, except the diagonal coefficient.

The previous steps ensure that $T' = L' \times C'$ and $L'$ is an invertible matrix. The last substep (not shown here) is to pivot on row $i$ of $T'$ to put $T'$ back to reduced row echelon form.

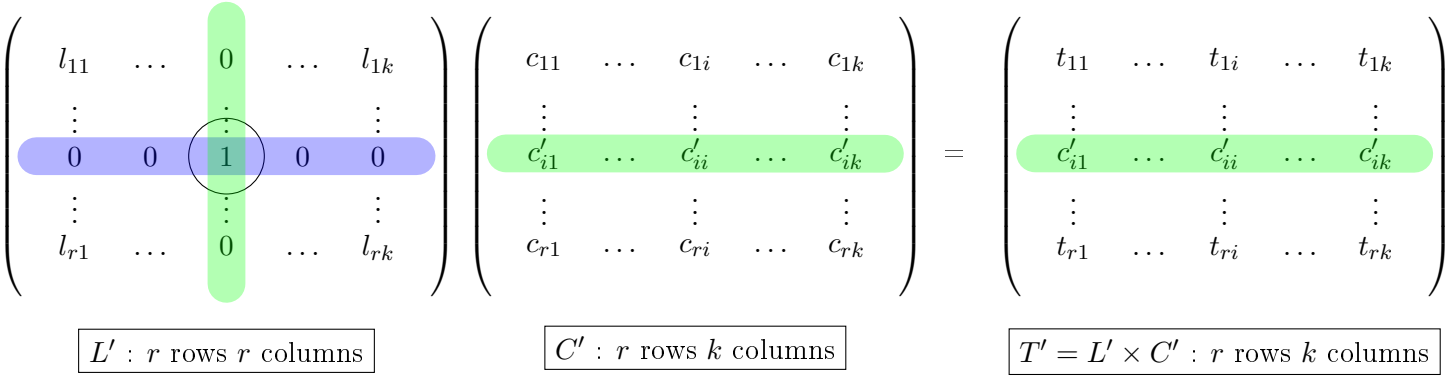



Figure 2: Procedure to exchange two rows in the Gauss reduction algorithm

- for AdmissibleLevel: $\mathcal{O}(1)$

The key advantage of this method is that the exchange of one row is much more efficient than the recomputation of the whole Gauss reduction, which has a cost $\mathcal{O}(r(r+k))$[3]. There is nearly no spatial complexity overhead in this method compared to strategy PS; the most significant overhead is to store $L$, which approximately doubles the memory usage compared to storing a single matrix.

The average complexity of the determination of the next composition in line 11 of Algorithm 2 can very loosely be majored by $\mathcal{O}(r)$, since it is proportional to the average number of leading ones of a strong partition of $r$ into $|\mathfrak{u}|$ parts. In fact, although it is not necessary here, it can be shown that this complexity is $\frac{r}{r-|\mathfrak{u}|+1}$, thus much smaller than $\mathcal{O}(r)$ when $r >> |\mathfrak{u}|$. Then the worst-case complexity for Algorithm 2 (attained when all the compositions are visited in the while loop) is

$$\mathcal{O}\left(\left[r+\binom{r-1}{|\mathfrak{u}|-1}\right](r+k)\right).$$

For Algorithm 1, the complexity can be majored by

$$\mathcal{O}\left(sk^2+\sum_{l=2}^{d}\binom{s}{l}\sum_{r=l}^{k}\left(\left[r+\binom{r-1}{l-1}\right](r+k)\right)\right)=\mathcal{O}\left(\sum_{l=1}^{d}\binom{s}{l}\sum_{r=l}^{k}\left(\left[r+\binom{r-1}{l-1}\right](r+k)\right)\right).$$

Using the identity on binomial coefficient $\sum_{r=l}^{k}\binom{r-1}{l-1}=\sum_{r=l}^{k}\left[\binom{r}{l}-\binom{r-1}{l}\right]=\binom{k}{l}$, our total complexity can be re-written

$$\mathcal{O}\left(\sum_{l=1}^{d}\binom{s}{l}\left[2k\binom{k}{l}-\binom{k}{l+1}+k^3\right]\right).$$

As soon as $k \geq d+2$ (which is always the case in our applications), the $k^3$ term is negligible, thus we proved the following theorem:

**Theorem 3.10.** *For a set of $k \times k$ matrices $S=(M_1,\ldots,M_s)$, for finite-order weights of order $d \leq k-2$, the algorithm computes the multi-level t-value based figure of merit (4) with complexity:*

$$\mathcal{O}\left(k\sum_{l=1}^{d}\binom{s}{l}\binom{k}{l}\right). \tag{15}$$

*When $d=k-1$ or $d=k$, the algorithm has the complexity*

$$\mathcal{O}\left(k\sum_{l=1}^{d}\binom{s}{l}\left[\binom{k}{l}+k^2\right]\right). \tag{16}$$

Two cases are of special interest:

- if $d=2$ (setting of [6], sparsest weights), we obtain $\mathcal{O}(s^2k^3)$.

[3]Because we are working in $\mathbb{F}_2$, we lose one order of complexity compared to the usual $\mathcal{O}(r^2(r+k))$ for the Gauss reduction. Indeed, in $\mathbb{F}_2$, a row operation such as a transvection can be implemented as an integer operation in $\mathcal{O}(1)$, whereas such an operation costs $\mathcal{O}(r)$ in other fields.

- if $d = \max(s, k)$ (most general weights), we can simplify the complexity using the Vandermonde equality to obtain $\mathcal{O}\left(k\binom{s+k}{s}\right)$ if $s \leq k-2$, $\mathcal{O}\left(k\binom{s+k}{s} + k^3 2^s\right)$ else.

Here is a couple of additional remarks on the complexity:

- Early stopping criterions both in Algorithms 1 and 2 can further reduce the complexity (and do reduce it in practice). In particular, in Algorithm 2, we stop as soon as we encounter a non-admissible composition. Thus an interesting question to improve these algorithms would be to find a heuristic for a good starting point in the exploration of the different compositions, for which we hope to run quickly into this stopping criterion.
- The algorithm is highly parallelizable because the most expensive step is the while-loop in Algorithm 2, when visiting the different compositions; the exploration of the space of compositions can be parallelized, and each explorer can compute the while-loop of Algorithm 2 on its own, sending a termination signal if it encounters a composition such that AdmissibleLevel equals $+\infty$.

## 3.5 Comparison with other proposed methods

The algorithm proposed in [22] allows to compute a multi-level $t$-value based figure of merit (4) using strategy S. They obtain the complexity

$$\sum_{r=1}^{k} \sum_{l=1}^{\max(d,r)} \binom{s}{l}\binom{r-1}{l-1} 2^{r-l} = \sum_{l=1}^{d} \binom{s}{l} \sum_{r=l}^{m} \binom{r-1}{l-1} 2^{r-l}.$$

We can majorate $2^{r-l}$ by $2^{m-l}$ (which is not too bad when $m \gg s$), thus majorate their complexity by

$$\sum_{l=1}^{d} 2^{k-l} \binom{s}{l}\binom{k}{l},$$

which is under a similar form than (15), replacing the $k$ factor with $2^{k-l}$, which is clearly a worse complexity.

Other algorithms have been proposed to compute only the $t$-value of a net (definition 2.1), without computing the $t$-values of all subprojections, which are needed to compute the figure of merit (4). The complexity of these algorithms is to be compared with the case $d = \max(s, k)$ in theorem 3.10. In this case, our algorithm computes the $t$-value of the whole net, and also computes the $t$-value of all the subprojections as a bonus.

[20] implements strategy PS to compute the $t$-value, with complexity:

$$\sum_{r=1}^{k} \binom{r+s-1}{r} \frac{r(r-1)}{2}$$

Our strategy is a refinement of this strategy, as it allows to remember the results of computations between two composition matrices. When $s \leq k$, we observe that the last term of this sum is of the same order of magnitude than our total complexity, i.e. $\mathcal{O}\left(k^2\binom{k+s-1}{s}\right)$, thus their total complexity is much superior to ours.

[3] proposes strategy DM (for Dick & Matsumoto), based on a duality-based error analysis, with complexity $\mathcal{O}(ks2^k)$. The comparison with our complexity really depends on the relative value of $k$ and $s$: our algorithm is polynomial in $k$ and exponential in $s$ whereas theirs is exponential in $k$ and polynomial in $s$. We refer to section 4.2 for more details on the comparison of these complexities. We will observe that our algorithm always performs better when the number of points is greater than $2^{s+1}$.

As a summary, *our algorithm performs better than all previously known algorithms in the case $k \geq s+1$, for all dimensions less than 20.*

# 4 Numerical implementation and experiments

We begin by presenting our work on the *LatNet Builder* software during the internship. Afterwards, in subsections 2 to 4, results of numerical experiments and practical examples are presented. In subsection 5, a summary of the main ideas found in this part is proposed.

## 4.1 *LatNet Builder* software

*LatNet Builder* is a software to search for good lattices and base-2 digital nets. It has been developed at the Stochastic Simulation and Optimization laboratory of University of Montreal since 2012. It was first named *Lattice Builder* as ordinary lattice rules were the only set type implemented [14]. In 2017, Ayman Jemel from École polytechnique implemented polynomial lattice rules. During our internship, with Maxime Godin, we implemented digital nets construction, and thus renamed the software *LatNet Builder*.

To our knowledge, there is no other open-source software at the level of generality of *LatNet Builder*. Dirk Nuyens provides the QMC4PDE (quasi-Monte Carlo for Partial Differential Equations) software [8]. It constructs ordinary and polynomial lattice rules depending on user-defined parameters. Some weights implemented in QMC4PDE are specific to the application to PDEs. Apart from these specific weights, all the functionalities of QMC4PDE are included in *LatNet Builder*.

To experiment and use the point sets in an RQMC setting, the *Stochastic Simulation in Java* (SSJ) library [11] has also been developed at the Stochastic Simulation and Optimization laboratory. This library was used for the experiments presented in sections 4.3 and 4.4.

Our main contribution to *LatNet Builder* is to include the search for digital nets, with modern programing (C++ standard 14). We implemented the following main features[4]:

- Digital Net constructions (section 2.5.1): explicit (directly from explicit generating matrices), Sobol, polynomial lattice rules.
- Exploration methods (section 2.5.2): exhaustive, random, CBC.
- Figures of merit: $t$-value (definition 2.1), $t$-value based projection-dependent figures of merit (3), coordinate-uniform figures of merit and interlaced figures of merit. For concision, we do not explain the two last classes of figures in this report, but refer to the report of Maxime Godin for details.
- Normalizations of these figures of merit: see remark 2.23 for more details.

[4]This list is incomplete: more features were implemented but seem too esoteric to be mentioned here.

| Card. \ Dimension | **5** | **20** | **100** |
|---|---|---|---|
| $\mathbf{2^{10}}$ | $2.0 \cdot 10^{-3}$ | $7.3 \cdot 10^{-3}$ | $9.0 \cdot 10^{-2}$ |
| $\mathbf{2^{20}}$ | $3.4 \cdot 10^{-3}$ | $3.6 \cdot 10^{-2}$ | 0.63 |
| $\mathbf{2^{30}}$ | $9.0 \cdot 10^{-3}$ | 0.11 | 2.7 |

(a) Weights of order 2

| Card. \ Dimension | **5** | **20** | **100** |
|---|---|---|---|
| $\mathbf{2^{10}}$ | $2.9 \cdot 10^{-3}$ | $1.8 \cdot 10^{-2}$ | 0.60 |
| $\mathbf{2^{20}}$ | $1.0 \cdot 10^{-2}$ | 0.20 | 43.8 |
| $\mathbf{2^{30}}$ | $2.7 \cdot 10^{-2}$ | 0.68 | 96.5 |

(b) Weights of order 3

Table 1: Execution time in seconds for strategy MG for the computation of the $t$-value based figure of merit (3), with finite-order weights of orders 2 and 3. Every experiment consisted in the evaluation of the $t$-value of a random (projection-regular) net, and was repeated on 10 independent uniformly-distributed samples. The experiments were led on an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz processor.

- multilevel digital nets: multilevel figures of merit, such as (4), are supported by *LatNet Builder* and their computation yields no overhead compared with unilevel figures of merit.

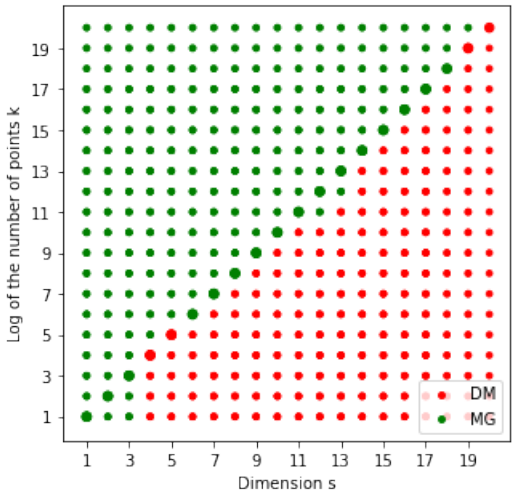
A secondary contribution to the software is the packaging and distribution. The software is currently available for the users, which was not the case last year after the internship of Ayman Jemel. The open-source software is distributed as a C++ executable and library, and also as a Python package. In addition, we developed a Graphical User Interface (GUI) for the software. The GUI was developed in Python in the Jupyter ecosystem, and is available online on the Binder platform (which graciously runs Jupyter notebooks). We invite the curious reader to test the software online (no download needed) by following the link
`https://mybinder.org/v2/gh/umontreal-simul/latnetbuilder/master?urlpath=/apps%2Fpython-wrapper%2Fnotebooks%2FInterface.ipynb`,
or to have a look at the GitHub page of the project:
`https://github.com/umontreal-simul/latnetbuilder`

## 4.2 Benchmarking of the implementations of the strategies to compute the $t$-value

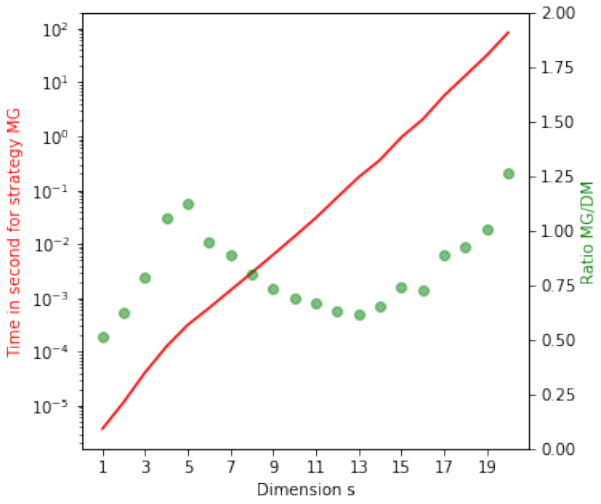
For the computation of the $t$-value, strategies MG and DM explained above are efficiently implemented in LatNet Builder. In Figure 3, we compare the runtime for random (projection-regular) nets. As expected from Theorem 3.10 and formulas in Section 3.5, our strategy performs better for high number of points and low dimension, while DM performs better for high dimension, and low number of points.

However, one of the main interest of our method is the possibility to compute not only the $t$-value but also $t$-value based projection-dependent figures of merit (4). By taking advantage of the special forms of weights, the computation time is greatly reduced. In Table 1, we present some benchmarking results for finite-order weights. This case often arises in the literature; for instance, [6], which is a main reference for the computation of high-quality digital nets, uses finite-order weights of order 2.

Even in very high dimension with a large number of points, the execution time remains reasonable (except for weights of order 3 in dimension 100 with more than $2^{20}$ points).




(a) For each dimension and each log of the number of points, the color of the dot corresponds to the quickest implementation. The bigger dots represent the diagonal $s = k$.




(b) For each diagonal point of Figure 3a ($s = k$), we plot the execution time of strategy MG and the ratio between the execution times of the two strategies. The execution time seems to increase exponentially with the dimension. For moderate dimensions (6-18), MG performs better.

Figure 3: Comparison of the implementations of strategies MG and DM for the computation of the $t$-value. Every experiment consisted in the evaluation of the $t$-value of a random (projection-regular) net, and was repeated on 20 independent uniformly-distributed samples. The experiments were led on an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz processor.

This configuration is much more favorable than the previous case where the execution time was already impractical for dimension 20 and $2^{20}$ points.

## 4.3 A numerical integration experiment

### 4.3.1 Presentation of the problem

To test our point sets, we use the set of test functions, that depend on one parameter $c \in (0, 1]$, introduced in [26]:

$$f_{c,s}(x) = \prod_{i=1}^{s} \left(1 + c\left(x_i - \frac{1}{2}\right)\right)$$

This set of functions is very interesting as a toy example, because we know its ANOVA decomposition (see (5)), which is equal to its power expansion

$$f_{c,s}(x) = 1 + \sum_{d=1}^{s} c^d \sum_{i_1 < \cdots < i_d} \left(x_{i_1} - \frac{1}{2}\right) \cdots \left(x_{i_d} - \frac{1}{2}\right) \tag{17}$$

This means the contribution of the subprojection $\{i_1, \cdots, i_d\}$ to the variance of $f$ is equal to $\left(\frac{c}{12}\right)^{2d}$, because $\mathrm{Var}(X - \frac{1}{2}) = \mathbb{E}((X - \frac{1}{2})^2) - (\mathbb{E}(X - \frac{1}{2}))^2 = \frac{1}{12}$.

According to [13], this means we know the ideal weights, up to a multiplicative constant: they should be chosen as $(\alpha c)^{2d}$. We arbirtrarly chose $\alpha^2 = 1/2$.

Moreover, there is a notion of effective dimension (or steepness) of the problem: heuristically, we can say that the effective dimension (in the superposition sense) is approximately equal to $sc$. This can be seen from several ways. First,

$$\sup f_{c,s} - \inf f_{c,s} = \left(1 + \frac{c}{2}\right)^s - \left(1 - \frac{c}{2}\right)^s \simeq \exp\left(\frac{1}{2}cs\right)$$

The 'steepness' of the problem is therefore controled by the parameter $cs$. Another way to understand heuristically the importance of this parameter $cs$ is to see that the total variance of $f_{c,s}$ equals

$$\sum_{d=1}^{s} \binom{s}{d} \left(\frac{c}{12}\right)^{2d} = (1 + \frac{c^2}{144})^s$$

The fraction of the variance explained by the first $cs$ dimensions equals

$$\frac{\sum_{d=1}^{cs} \binom{s}{d} (\frac{c}{12})^{2d}}{\sum_{d=1}^{s} \binom{s}{d} (\frac{c}{12})^{2d}}$$

This equals $\mathbb{P}(\bar{X}_s \leq c)$ with $\bar{X}_s = \frac{1}{s}\sum_{i=0}^{s} X_i$ where each $X_i$ follows a $\mathcal{B}((c/12)^2)$ distribution. Since $c > \mathbb{E}(X_i) = (c/12)^2$ we obtain by the Central Limit Theorem that $\mathbb{P}(\bar{X}_s \leq c) \to 1$ when $s \to +\infty$ with fixed $c$. Hence, when $s \to +\infty$ with fixed $c$, the fraction of the variance explained by the first $cs$ dimensions converges to 1. This means that the effective dimension of $f_{c,s}$ in the superposition sense (defined in 2.11) equals $cs$ in the limit.

### 4.3.2 Experimental setting

We numerically integrate $f_{c,s}$ for all $s$ from 5 to 65 by increasing steps of 10, for all $c$ from 0.1 to 1 by increasing steps of 0.1. We use a Monte-Carlo point set with 4000 points and the following QMC point sets, all with 4096 points (i.e $m = 12$). The name at the beginning of each line refer to the name of the point set on the graphs:

- '`Sobol std`': default (pre-computed) Sobol, the primitive polynomials and direction numbers are stored in SSJ, and taken from [16].
- '`random`': a Sobol point set whose direction numbers are taken uniformly at random
- five points sets optimized by *LatNet Builder*:
  - '`t-value`': Sobol digital net optimized for the $t$-value, with a random search of 100 samples.
  - '`PLR`': Polynomial Lattice Rules optimized for the $P_2$ figure of merit [17] with $q = 2$, computed with the fast-CBC algorithm.
  - '`proj-dep t-value`': Sobol digital net optimized for the figure $D^{(+\infty)}_{h_0,\gamma,m}$ with a CBC search (full CBC up to dimension 15, and mixed CBC for higher dimensions, with 100 random samples starting from coordinate 8). Here $h_0(|\mathfrak{u}|, m, t) = t$, which means we simply sum the $t$-values of the subprojections.
  - '`proj-dep t-value *`': Sobol digital net optimized for the figure $D^{(+\infty)}_{h_*,\gamma,m}$ (see theorem 2.17), with the same CBC search as above.

– '`proj-dep t-value L2`': same as above, but with the figure $D^{(1)}_{h_2,\gamma,m}$ (see remark 2.27) instead of $D^{(+\infty)}_{h_*,\gamma,m}$.

For the third point sets, the weights are taken as explained above, i.e. $\gamma_{\mathfrak{u}} = (0.5c^2)^{|\mathfrak{u}|}$. For the last three point sets, it is much more efficient to use finite-order weights (see table 1), so we chose the weights $\gamma_{\mathfrak{u}} = (0.5c^2)^{|\mathfrak{u}|}\mathbf{1}_{|\mathfrak{u}|\leq 3}$. All QMC point sets are randomized by a scrambling [4, p. 396] followed by a digital shift (section 2.7). The variance of the RQMC estimator is estimated by 1000 independents randomizations of the QMC point set.

### 4.3.3 Results

Figure 4 presents the results in terms of variance ratio. This metric is used to compare the performance of RQMC point sets to MC sampling, and also to compare RQMC point sets with each other. We can draw the following conclusions: first, the notion of effective dimension is indeed pertinent, as it controls the effectiveness of RQMC compared to MC. Second, computing on the fly new QMC point sets seems efficient for this example: they often perform better than pre-computed point sets. Third, optimized Sobol nets do not always beat random Sobol nets. Only two out of the four figures of merit yield significantly better nets than random Sobol nets.

All these points suggest that using *LatNet Builder* to compute QMC point set is efficient for this example. Concerning Sobol point sets, the best performing figures of merit are $D^{(+\infty)}_{h_*,\gamma,m}$ and $D^{(1)}_{h_2,\gamma,m}$. They almost always bring a performance gain compared to random Sobol nets. Yet, they are generally beaten by polynomial lattice rules. The reason for that is unclear: it could be because of the change of point set construction (from Sobol to polynomial lattice rules), or because of the change of figure of merit (from $t$-value based figure to $P_2$). To clarify this point, further experiments would be needed, for instance testing Sobol with the $P_2$ figure.

It is worth mentioning that the decomposition (17) greatly helped us in parameterizing *LatNet Builder* by giving us the correct weights. This is an exceptional (toy) behaviour. Usually the weights have to be either guessed or estimated by costly simulations.

We can also study the convergence rates of the different methods: figure 5 represents the gain in variance of RQMC methods compared to a standard MC integration. Polynomial lattice rules present the best variance convergence rate ($1/n^{2.1}$). Sobol nets with the figures $D^{(1)}_{h_2,\gamma,m}$, $D^{(+\infty)}_{h_0,\gamma,m}$ and the simple $t$-value have a convergence rate of $1/n^{2.0}$. Finally, pre-computed Sobol nets, random Sobol nets, and Sobol nets with the figure $D^{(+\infty)}_{h_*,\gamma,m}$ have a convergence rate of $1/n^{1.8}$.

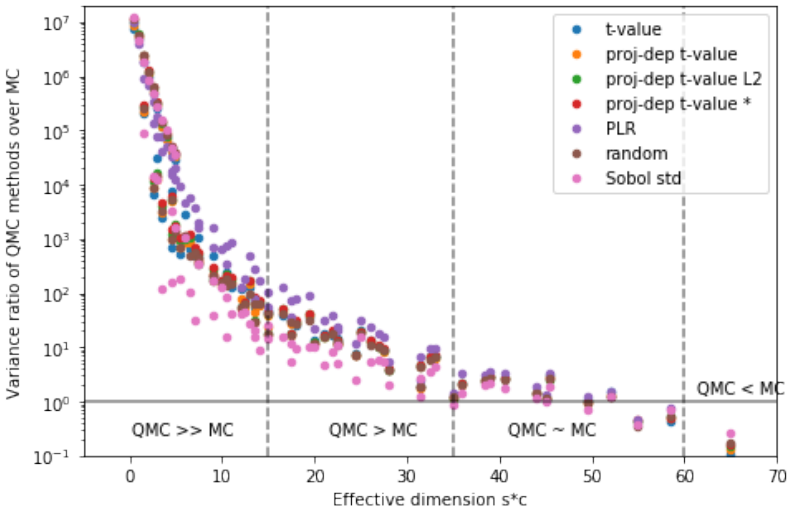
## 4.4 A stochastic simulation experiment

The canonical example for stochastic simulations in the quasi-Monte Carlo community is the pricing of an Asian option. To depart from this example, we study here an Orstein-Uhlenbeck process.

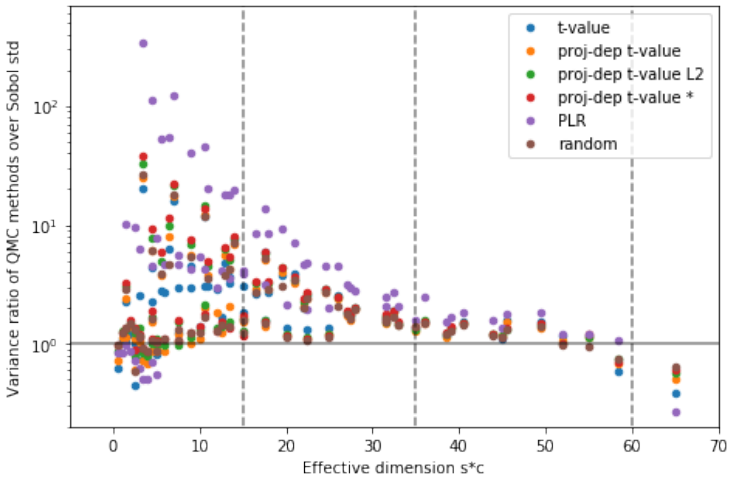
The (centered) Ornstein-Uhlenbeck process $X_t$ verifies the following stochastic differential equation:

$$dX_t = -\theta X_t dt + \sigma dW_t \tag{18}$$

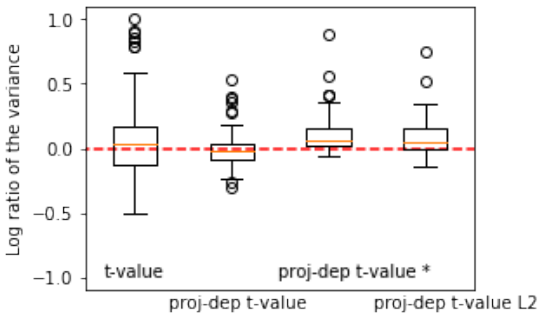
where $\theta$ and $\sigma$ are positive parameters while $W_t$ denotes the Wiener process.




(a) The ratio of the variance of various QMC point sets over the variance of a MC point set is plotted. The four regions delimited by vertical lines correspond to the behaviours identified in [26]: for $sc < 15$, QMC performs much better than MC; for $15 < sc < 35$, QMC performs better than MC; for $35 < sc < 60$, both have similar results, while MC outperforms QMC for $sc > 60$. In very low dimension, the Sobol methods perform better while Polynomial Lattice Rules outperform them starting from effective dimension 6.




(b) The ratio of the variance of various QMC point sets over the variance of a standard Sobol point set is plotted. Except in very low and very high dimensions, we observe an improvement of the variance when using point sets computed by *LatNet Builder* compared to the standard Sobol set. In particular, Polynomial Lattice Rules outperform the standard set by a factor up to 400.




(c) The log of the ratios of the variance of various QMC point sets over the variance of a random Sobol point set is plotted. We observe that the projection dependent $t$-value star-discrepancy and L2-discrepancy merits do almost always better than a random Sobol net. However, it is not the case of the $t$-value and of the projection dependent $t$-value.

Figure 4: Comparative results of QMC point sets depending on the effective dimension of the problem, for a fixed number of points ($2^{12}$ points). In each plot, the metric of performance is the ratio of variance of two estimators. Each point represents an experiment for a given $s$ and a given $c$. The plotted ratio depends on the plot.

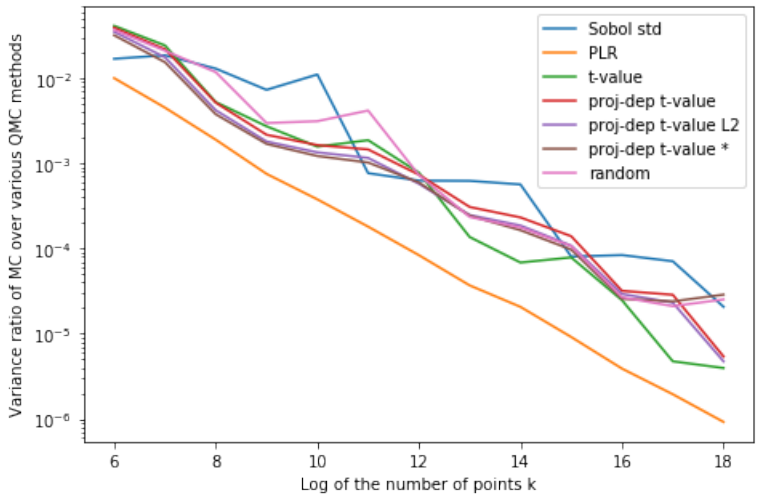



Figure 5: Ratio of the variance of MC sampling over various RQMC methods, as a function of the log in base 2 of the number of points, for $s = 25$ and $c = 0.2$. For each number of points, new point sets are recomputed. To estimate each RQMC variance, 1000 independent randomizations of each QMC point set are computed.

This model intervenes in many physical processes, as it models a Brownian particle in a parabolic potential. For instance, the Brownian motion of a damped harmonic oscillator driven by white noise is given by

$$m\frac{d^2x(t)}{dt^2} + \gamma\frac{dx(t)}{dt} + \omega^2 x(t) = D\eta(t) \tag{19}$$

where $m$ is the mass of the oscillator, $\gamma$ is the damping coefficient, $\omega$ the natural frequency of the oscillator, $D$ the diffusion coefficient (which relates to the temperature) and $\eta(t)$ is the random force due to the density fluctuation of the surrounding medium.

If the viscous damping force $\gamma\frac{dx(t)}{dt}$ is much larger than the inertial term $m\frac{d^2x(t)}{dt^2}$, (19) simplifies to the stochastic differential equation, similar to (18):

$$\frac{dx(t)}{dt} = -\frac{\omega^2}{\gamma}x(t) + \frac{D}{\gamma}\eta(t)$$

The quantity of interest in the following is the expected value of a functional of the process $\mathbb{E}(\int_0^1 f(X_u)du)$. For instance $f = (x \mapsto x^2)$ corresponds to the average potential energy stored in the oscillator.

An application of the Itô formula yields the solution to (18):

$$d(\exp^{\theta t} X_t) = \theta \exp^{\theta t} X_t dt + \exp^{\theta t}(-\theta X_t dt + \sigma dW_t) = \sigma \exp^{\theta t} dW_t.$$

Taking a null initial condition, we finally obtain

$$X_t = \int_0^t \sigma \exp^{\theta(s-t)} dW_s$$

Following the Riemann method, $\mathbb{E}(\int_0^1 f(X_u)du)$ can be approximated by a $s$-step time discretization of the process $\sum_{j=1}^{s} f(X_{j/s})$. The vector $(X_{j/s})_j$ is a Gaussian vector in dimension $s$ with null mean, and its covariance is given by the Itô isometry:

$$\mathrm{Cov}(X_t, X_u) = \mathbb{E}\left(\int_0^t \sigma \exp^{\theta(v-t)}\, dW_v \int_0^u \sigma \exp^{\theta(v-u)}\, dW_v\right) \tag{20}$$

$$= \mathbb{E}\left(\int_0^1 \mathbf{1}_{v\leq t, v\leq u}\sigma^2 \exp^{2\theta v}\, dv\right) \exp^{-\theta(t+u)} \tag{21}$$

$$= \frac{\sigma^2}{2\theta}\left(-1 + \exp^{2\theta \min(t,u)}\right) \exp^{-\theta(t+u)} \tag{22}$$

The associated variance-covariance matrix is $C$ such that $C_{i,j} = \mathrm{Cov}(X_{i/s}, X_{j/s})$, given by (22). Following (for instance) [13], to simulate this process, we draw $n$ independent random vectors $(\boldsymbol{U}^{(i)})_{i=1\cdots n}$ uniformly distributed on $[0,1]^s$, apply the inverse distribution function $\Phi^{-1}$ of the centered normal Gaussian law to obtain normal Gaussian vectors, then multiply these vectors by $A$ such that $C = AA^T$ to obtain $n$ independent realisations of $(X_{j/s})_j$. Hence our estimator of $\mathbb{E}(\int_0^1 f(X_u)du)$ is:

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{s} f(X_{j/s}^{(i)}) \tag{23}$$

Any matrix $A$ verifying $C = AA^T$ would fit, but a very interesting idea is to do a Principal Component Analysis on $C$. By the Spectral theorem, we can decompose $C = PDP^T$ with $P$ an orthogonal matrix containing the eigenvectors of $C$, and $D$ a diagonal matrix containing the eigenvalues of $C$, sorted by decreasing values. Then we can take $A = PD^{1/2}$. This method allows to drastically reduce the effective dimension of the problem by grouping the variance on the first coordinates on $C$.

As a matter of fact, this method is so efficient that it reduces the effective dimension of the simulation of $\mathbb{E}(\int_0^1 X_u^2 du)$ to 1! Indeed, the QMC estimator (23) of $\mathbb{E}(\int_0^1 X_u^2 du)$ can be rewritten

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{s}(X_{j/s}^{(i)})^2 = \frac{1}{n}\sum_{i=1}^{n}||A\Phi^{-1}(\boldsymbol{U}^{(i)})||_2^2 = \frac{1}{n}\sum_{i=1}^{n}\langle A^T A\Phi^{-1}(\boldsymbol{U}^{(i)}), \Phi^{-1}(\boldsymbol{U}^{(i)})\rangle \tag{24}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{s} D_{j,j}\Phi^{-2}(U_j^{(i)}) \tag{25}$$

$$= \sum_{j=1}^{s} D_{j,j}\frac{1}{n}\sum_{i=1}^{n}\Phi^{-2}(U_j^{(i)}) \tag{26}$$

where the third equality comes from the fact that $A^T A = D$ is a diagonal matrix. (26) means that we have decoupled the coordinates; in other words, we only care about the behaviour of our point set on 1-dimensional projections of $[0,1]^s$. In particular for *any* digital net, the projection of the digital net on any coordinate is $\{0, 1/n, 2/n, \cdots (n-1)/n\}$. Thus, for *any* digital net randomized with a digital shift, the randomized QMC estimator (26) equals

$$X := \frac{1}{n}\sum_{j=1}^{s} D_{j,j}\sum_{i=0}^{n-1}\Phi^{-2}\left(\frac{i+U}{n}\right). \tag{27}$$

where $U$ follows a uniform law on $[0,1]$.

**Proposition 4.1.** *For any digital sequence randomized with a digital shift, the variance of the estimator of* $\mathbb{E}(\int_0^1 X_u^2 du)$ *converges at the rate* $1/n^2$.

*Proof.* The variance of the estimator (27) equals

$$\mathrm{Var}(X) = \frac{c}{n^2}\,\mathrm{Var}\left[\sum_{i=0}^{n-1}\Phi^{-2}\left(\frac{i+U}{n}\right)\right] \tag{28}$$

$$= \frac{c}{n^2}\sum_{i=0}^{n-1}\mathrm{Var}\left[\Phi^{-2}\left(\frac{i+U}{n}\right)\right] + \frac{c}{n^2}\sum_{i=0}^{n-1}\sum_{i'=0}^{n-1}\mathrm{Cov}\left[\Phi^{-2}\left(\frac{i+U}{n}\right),\Phi^{-2}\left(\frac{i'+U}{n}\right)\right] \tag{29}$$

Both sums in (29) can be majorated in the same way; here we only majorate the first term which involves less technicalities. The second term is majorated in a similar fashion and by using the Cauchy-Schwartz inequality.

$$\begin{aligned}\sum_{i=0}^{n-1}\mathrm{Var}\left[\Phi^{-2}\left(\frac{i+U}{n}\right)\right] &\le \sum_{i=0}^{n-1}\mathbb{E}\left[\Phi^{-4}\left(\frac{i+U}{n}\right)\right]\\ &= \sum_{i=0}^{n-1}\int_0^1 \Phi^{-4}\left(\frac{i+x}{n}\right)dx\\ &= \frac{1}{n}\int_0^1 \Phi^{-4}(x)dx\\ &= \mathcal{O}(1)\end{aligned}$$

The last integral equals the fourth moment of the normal distribution, which is finite. □

This shows the pertinence of the notion of effective dimension for quasi-Monte Carlo methods. When the effective dimension is low (here this toy example is extreme since the effective dimension is reduced to 1), quasi-Monte Carlo works extremely well. This example also shows that it is possible to modify the simulation procedure in order to reduce the effective dimension. Here a PCA was used, but another standard method is to use a Brownian bridge.

The obtained convergence rate has been verified numerically.

## 4.5 Summary of the numerical experiments

First, we checked that our algorithm for the computation of the $t$-value is indeed faster than the previously known ones when the logarithm of the number of points exceed the dimension. It also can be drastically speeded-up by considering a projection-dependent $t$-value based figure with finite-order weights.

Second, the right notion to understand the efficiency of QMC and RQMC methods is the effective dimension, that can be defined in several ways [10], and not the dimension by itself.

Third, the example of the Principal Component Analysis shows that the problem formulation can be twisted to make QMC integration work better, by reducing the effective dimension.

Fourth, the performance of bespoke QMC point sets compared to standard ones remains to be tested and understood better.

Fifth, random Sobol nets perform surprinsingly well compared to optimized ones. This can be explained because the Sobol construction is already a 'strong' construction, that gives good garantees about the rate of convergence. Thus random Sobol nets have already good convergence properties, that make them hard to beat with optimized Sobol nets. Said otherwise, there is not much lattitude in the tuning of the performance of Sobol nets.

The last questions show that more numerical experiments are needed to assess the performance of *LatNet Builder*, and in particular of the search for good Sobol nets.

# 5 Conclusion

To conclude, I would like to give a few ideas of where the *LatNet Builder* software stands right now, and where this project shoud be heading.

## 5.1 For researchers and advanced practitioners

*LatNet Builder* is the only open-source software that allows to construct digital nets with a large number of configurable parameters. Thanks to novel algorithms (such as the one presented in section 3) and programming techniques, this software reaches state-of-the-art computing efficiency.

As of now, *LatNet Builder* is a great tool for researchers who would like to numerically study the performance of digital nets constructions, as well as advanced QMC practitioners who know enough of this field to be able to parametrize the software. Time was spent during the internship to reach a greater ease of online testing and local installation of the software (at least for Linux and Mac OS users). This means the software is now ready to be advertised to the QMC community, for testing and usage.

Many questions regarding the performance of the digital nets computed by *LatNet Builder* are open, such as:

- To what extend are the results sensitive to the choice of weights? Is there an efficient algorithm to choose the weights depending on the application?
- For now, the exploration algorithms are very primitive (essentially random search). Can better algorithms be used, that would take advantage of an hypothetical local structure of the exploration space?
- The exploration space can in some cases be extremely large. Is it really worth it to explore such a large space? In some cases, a very economical method (such as choosing the best net among ten random samples) does as good as a full costly exploration. In which case does this happen?

These questions, that we very quickly browsed through during the internship, could be the subject of another internship by themselves.

## 5.2 For the general public

What makes *LatNet Builder* an interesting tool for power users, i.e. its generality and configurability, also makes it a difficult software to use for non-experts. In my opinion, this issue should be adressed, in order to reach a greater user base. To do so, a simplified version of the software should be designed, for instance by

- defining sensible default values for all parameters.
- defining simpler versions of the search procedures, where (much) less parameters are left for the user to tune, for instance like in [8].

Besides, although the relevance of QMC for many integration problems has been theoretically proved and experimentally confirmed for a long time, this method still lacks visibility among the general public: for instance, there is still no QMC generation method in the main Python scientific computing libraries (Numpy, Scipy).

A simplified version of the *LatNet Builder* interface for Python could fill this gap, with the additional benefit of making the work of the QMC community and in particular of the Stochastic Simulation and Optimization Laboratory more well-known.

# References


[1] R. P. Agarwal and E. C. Flaut. *An introduction to linear algebra.* Chapman and Hall, 2017.

[2] P. Bratley and B. L. Fox. Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software*, 14(1):88–100, 1988.

[3] J. Dick and M. Matsumoto. On the fast computation of the weight enumerator polynomial and the $t$-value of digital nets over finite Abelian groups. *SIAM Journal on Discrete Mathematics*, 27(3):1335–1359, 2013.

[4] J. Dick and F. Pillichshammer. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration.* Cambridge University Press, Cambridge, U.K., 2010.

[5] J. Dick, I. H. Sloan, X. Wang, and H. Woźniakowski. Liberating the weights. *Journal of Complexity*, 20(5):593–623, 2004.

[6] S. Joe and F. Y. Kuo. Constructing Sobol sequences with better two-dimensional projections. *SIAM Journal on Scientific Computing*, 30(5):2635–2654, 2008.

[7] P. Klingsberg. A Gray code for compositions. *Journal of Algorithms*, 3(1):41 – 44, 1982.

[8] Frances Y. Kuo and Dirk Nuyens. Application of quasi-Monte Carlo methods to elliptic PDEs with random diffusion coefficients: A survey of analysis and implementation. *Foundations of Computational Mathematics*, 16(6):1631–1696, Dec 2016.

[9] G. Larcher, F. Pillichshammer, and K. Scheicher. Weighted discrepancy and high-dimensional numerical integration. *BIT Numerical Mathematics*, 43(1):123–137, Mar 2003.

[10] P. L'Ecuyer. Quasi-Monte Carlo methods with applications in finance. *Finance and Stochastics*, 13(3):307–349, 2009.

[11] P. L'Ecuyer. SSJ: Stochastic simulation in Java. `http://simul.iro.umontreal.ca/ssj/`, 2016.

[12] P. L'Ecuyer. Randomized quasi-Monte Carlo: An introduction for practitioners. In P. W. Glynn and A. B. Owen, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2016*, Berlin, 2017. Springer-Verlag. to appear.

[13] P. L'Ecuyer and D. Munger. On figures of merit for randomly-shifted lattice rules. In H. Woźniakowski and L. Plaskota, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2010*, pages 133–159, Berlin, 2012. Springer-Verlag.

[14] P. L'Ecuyer and D. Munger. Algorithm 958: Lattice builder: A general software tool for constructing rank-1 lattice rules. *ACM Trans. on Mathematical Software*, 42(2):Article 15, 2016.

[15] P. L'Ecuyer, J.-S. Parent-Chartier, and M. Dion. Simulation of a Lévy process by PCA sampling to reduce the effective dimension. In *Proceedings of the 2008 Winter Simulation Conference*, pages 436–443. IEEE Press, 2008.

[16] C. Lemieux, M. Cieslak, and K. Luttmer. *RandQMC User's Guide: A Package for Randomized Quasi-Monte Carlo Methods in C*, 2004. Software user's guide, available at `http://www.math.uwaterloo.ca/~clemieux/randqmc.html`.

[17] C. Lemieux and P. L'Ecuyer. Randomized polynomial lattice rules for multivariate integration and simulation. *SIAM Journal on Scientific Computing*, 24(5):1768–1789, 2003.

[18] H. Niederreiter. Low-discrepancy point sets obtained by digital constructions over finite fields. *Czechoslovak Math. Journal*, 42:143–166, 1992.

[19] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *SIAM CBMS-NSF Reg. Conf. Series in Applied Mathematics*. SIAM, 1992.

[20] G. Pirsic and W. Ch. Schmid. Calculation of the quality parameter of digital nets and application to their construction. *Journal of Complexity*, 17(4):827–839, 2001.

[21] François-Michel De Rainville, Christian Gagné, Olivier Teytaud, and Denis Laurendeau. Evolutionary Optimization of Low-Discrepancy Sequences. *ACM Transactions on Modeling and Computer Simulation*, 22(2):9:1–9:25, 2012.

[22] W. Ch. Schmid. The exact quality parameter of nets derived from Sobol' and Niederreiter sequences. *Recent Advances in Numerical Methods and Applications*, pages 287–295, 1999.

[23] I. H. Sloan and A. Rezstov. Component-by-component construction of good lattice rules. *Mathematics of Computation*, 71:262–273, 2002.

[24] I. H. Sloan and H. Woźniakowski. When are quasi-Monte Carlo algorithms efficient for high dimensional integrals. *Journal of Complexity*, 14:1–33, 1998.

[25] I. M. Sobol'. The distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R. Comput. Math. and Math. Phys.*, 7(4):86–112, 1967.

[26] I. M. Sobol and D. I. Asotsky. One more experiment on estimating high-dimensional integrals by quasi-monte carlo methods. *Math. Comput. Simul.*, 62(3-6):255–263, March 2003.