Youssef Achari Berrada
Promotion X2013

RAPPORT DE STAGE DE RECHERCHE

Polynomial Lattice Rules: Construction and Selection.

NON CONFIDENTIEL

Département
Mathématiques Appliquées - MAP595
Directeur de stage
Emmanuel Gobet - Nizar Touzi
Maître de stage
Pierre L'Ecuyer
Date du stage
18 avril 2016 - 31 août 2016
Organisme d'accueil
Département d'Informatique et de Recherche Opérationnelle
Adresse
Université de Montréal in Montréal (Québec), Canada, H3C 3J7.

I, the undersigned, do hereby certify:

1. That the results presented in this report are the result of my own work;
2. that I am the author of this report;
3. that I did not use any third-party source or result without explicitly mentioning it, using the standard bibliography rules.

Date: *31 August 2016.*

Signature :
"I declare that this work could not be suspected of any plagiarism"

Achari Berrada Youssef.

Résumé

Les méthodes de quasi-Monte Carlo sont une alternative deterministique aux méthodes de Monte-Carlo traditionnelles et qui utilise l'arithméthique sur des ensembles finis pour générer une suite de points de taille quelconque sur le cube unité en dimension quelconque. Ces points sont évalués par des figures de mérites qui mesurent l'uniformité des points ainsi générés sur le cube unité.
Mon stage s'est focalisé sur l'étude de deux types de méthodes de quasi-Monte Carlo. Le premier étant celui des *lattices rules* et qui reposent sur l'arithmétique dans des anneaux de congruences de taille n : $\mathbb{Z}_n$. Le second est celui des *polynomial lattice rules* qui utilise l'arithmétique sur l'espace des polynomes de degré inférieur à m et à coefficients dans $\mathbb{F}_b$: $G_{b,m}$.
Le principal but de ce stage est de trouver un moyen pour construire et évaluer les *polynomial lattice rules.* Le langage de programmation étant le C++ pour que le code soit ensuite intégrable dans *Lattice Builder*, see [6, 5], qui est un logiciel complet pour la construction des *lattice rules* normaux et qui a été conçu par David Munger et Pierre L'Ecuyer.

Abstract

Quasi-Monte Carlo methods is a deterministic alternative to traditional Monte Carlo methods, and they use arithmetics on a finite set to generate a set of points in the unit-hypercube of dimension s. This points are evaluated using figures of merit to measure the uniformity of the set of points.
During my internship, I studied two types of quasi-Monte Carlo methods. The first is ordinary lattice rules and the second is polynomial lattice rules.
The main goal of my research internship was to build a software that construct and evaluate the polynomial lattice rules. The programming language is C++ in order be able to integrate the code for PLR into Lattice Builder which is a complete software that builds ordinary lattice rules and that was conceived by David Munger with Pierre L'Ecuyer.

Acknowledgements

First of all, I would like to express my gratitude to Pierre L'Ecuyer, my supervisor, for providing me with this internship opportunity, his time and and guidance. And I also would like to express my gratitude to Emmanuel Gobet, Nizar Touzi and Stefano De Marco, who encourage me doing this interesting research internship.
I want to acknowledge David Munger, who helped me with understanding the topic and the Lattice Builder software that he created with Pierre L'Ecuyer.
Finally, I would like to thank everyone working in the Pierre L'Ecuyer's laboratory of Simulation and Optimization, and who I have the chance to meet and know during my internship, Mamadou, Weyan, Ahn and her husband, Shohre, Nazim and Chaker.

Preface

In the first part of my research internship, I started with exploring the theory behind lattice rules using the reference book *Lattice Methods for Multiple Integration* [10].
It helps me to acquaint the basic knowledge behind the quasi-monte carlo methods.
Then I used the Lattice Builder software to consctruct ordinary lattice rules. Then, I looked in the code of Lattice Builder to understand the structure and the tools used in order to be able to implement the polynomial lattice rules inside Lattice Builder.
Lattice Builder is a complex and complete software that uses recent technics in C++, in terms of templates and compilation. It is why this period of my internship was the longest one.
Then I started learning the idea behind polynomial lattice rules to know how we construct and evaluate them. Thus, I understand that the process of constructing polynomial lattice rules is completly independent of the process behind ordinary lattice rules. But the evaluation part is similar, the only thing that changes, is the function $w : [0, 1) \to \mathbb{R}$ we apply to the set of points. I then choose the NTL library to do the construction.

Contents

1 Introduction

Monte Carlo and quasi-Monte Carlo methods are often used for estimating integrals over the s-dimensional unit cube of the form

$$I(f) := \int_{[0,1)^s} f(u)\,du \tag{1}$$

The basic idea is to estimate $I(f)$ by the equal-weight cubature rule of the form

$$Q_N(f; P_n) := \frac{1}{N}\sum_{i=0}^{N-1} f(x_k) \tag{2}$$

with $P_n := \{x_k\}_{k=0}^{N-1}$.
We know that for all Riemann integrable functions f, $Q_N(f) \longrightarrow I(f)$ when $N \to \infty$ if and only if the sequence P_n is uniformly distributed.
For the Monte Carlo method, the points of P_n are independent random vectors uniformly distributed over $[0,1)^s$. Then, by the central-limit theorem, we know that the size of the confidence interval on $I(f)$ converges as $O(\sigma/\sqrt{N})$, where $\sigma^2 = \mathrm{Var}\,[f(U)]$ and $U \sim U([0,1)^s)$ see [3].
For the quasi-Monte Carlo method, the points are generated deterministicly using arithmetics over finite sets, Z_n with $n \in \mathbb{Z}_0$ or F_p with p prime for lattice rules and $G_{b,m}$ for polynomial lattice rules with b prime and $m \in \mathbb{Z}_0$. Then the discrepancy allows us to measure the uniformity of the points that we generate. One of the formulation of the discrepancy is called the star disrepancy or the discrepancy at the origin for a set of points P of size N in the unit hypercube of s-dimension and it is expressed as follows

$$D_N^*(P) = \sup_{B\in J}\left|\frac{A(B,P)}{N} - \lambda_s(B)\right| \tag{3}$$

where

- J is all the intervals of the form

$$\prod_{i=0}^{s}[0, u_i)$$

  where u_i is in the semi-open interval $[0,1)$

- $A(B,P)$ is the number of points of P in B

- λ_s is the Lebesgue measure.

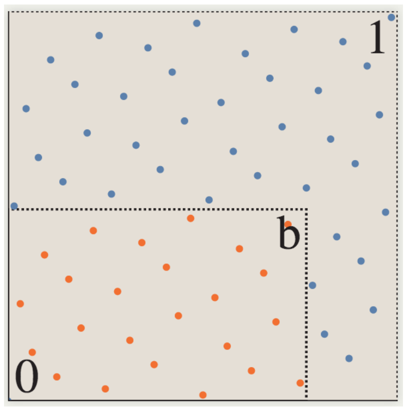


Figure 1: Star Discrepancy

The Koksma-Hlawka inequality gives us a worst-case error bound of any function f in a given Hilbert space, and its expression is

$$|I(f) - Q_N(f)| \leq D_N^*(P_N).V(f) \tag{4}$$

with $V(f)$ is the variation of f see [4].
What we can notice here is that the inequality 4 provides us with a bound that is a product of two terms : one depends only on f and the other depends only on P_N. Thus, if the space of functions we want to integrate has a bounded variation, the integration error depends only on the point set P. Unlike Monte Carlo, we have a worst case bound instead of just a confidence interval [4].
Later, we will consider the walsh series in base b that span $L_2([0,1)^s)$ to express the error bound in a way we can compute it in the case of polynomial lattice rules.

2 Construction for Rank-1 lattices

In the construction of quasi-monte carlo point set, we consider only the rank-1 lattices. These are lattices generated by a single generating vector modulo the modulus of the point set. Thus, the generating vector will determine the quality of the lattice when used for numerical integration.

2.1 Lattice Rules

The points of a rank-1 lattice rule with generating vector $z \in \mathbb{Z}_N^s$ are given by

$$\mathbf{x_k} = \frac{\mathbf{z}k \mod N}{N} \tag{5}$$

with $k = 0, ..., N-1$.
The quality of a rank-1 lattice rule, for a fixed N and s is fully determined by the choice of $\mathbf{z}$, see [10, 9].
The software Lattice Builder can generate different type of Lattice Rules.

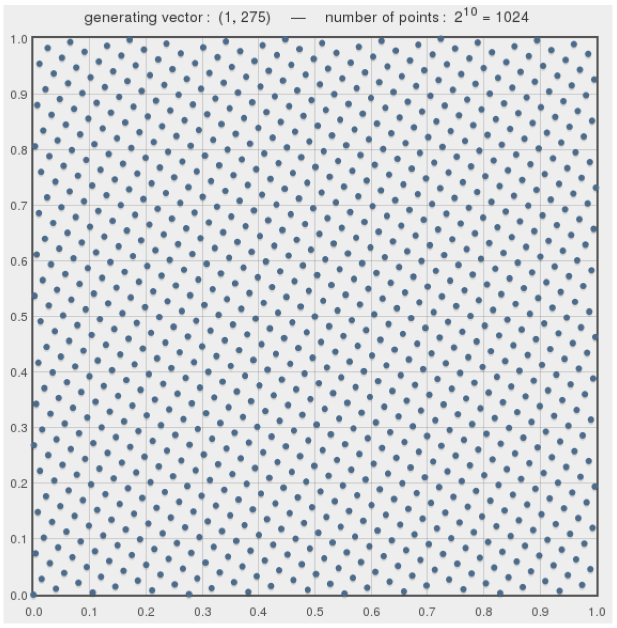


Figure 2: Lattice rule generated by Lattice Builder

2.2 Polynomial Lattice Rules

Niederreiter introduced a similar point set, from an algebraic point of view, see [8], where all the scalars in the equation 5 for a lattice rule are replaced by polynomials over a finite field $\mathbb{F}_b[X]$. The lattice points are given by

$$\mathbf{x}_k(X) = \frac{\mathbf{z}(X)k(X) \mod P(X)}{P(X)} \in \left(\mathbb{F}_b\left(X^{-1}\right)\right)^s \tag{6}$$

with $k(X) \in G_{b,m}$.
Where for all $j = 1, .., s$

$$x_{k,j}(X) = \sum_{i \geq 1} x_{k,j,i} X^{-i} \in \mathbb{F}_b\left(X^{-1}\right) \tag{7}$$

The number of points is $N = b^m$ and $m = \deg(P)$. Unlike ordinary lattice rules where n can be any non negative integer, $P(X)$ is mostly chosen irreducible over $\mathbb{F}_b[X]$.
We obtain the points from the polynomial lattice 6 by evaluating the polynomial point up to some precision w.

$$y_{k,j} = [x_{k,j}(b)]_w = \sum_{i \geq 1}^{w} x_{k,j,i} b^{-i} \tag{8}$$

The rule using $\{\mathbf{y}_k\}_{k=0}^{b^m-1}$, where $\mathbf{y}_k = (y_{k,1}, ..., y_{k,s})$, is called a rank-1 polynomial lattice rule.
In order to obtain the coefficient of Laurent series in 7, we can use the proposition 10.4 of the book *Digital Nets and Sequences* [2] that allows us to obtain any precision we want.

Proposition 1 *Let b be a prime power. For $P(X) \in \mathbb{F}_b[X]$ such as P is a monic polynomial and* $\deg(P) = m$*, and $Q(X) \in G_{b,m}$ with the notations as follow*
$P(X) = X^m + a_1 X^{m-1} + ... + a_{m-1}X + a_m$
$Q(X) = q_1 X^{m-1} + ... + q_{m-1}X + q_m$
In the Laurent series expansion of

$$\frac{Q(X)}{P(X)} = \sum_{l=1}^{\infty} x_l X^{-l}$$

The first m coefficients $x_1, .., x_m$ are obtained by solving the linear system

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ a_1 & 1 & \ddots & & \vdots \\ \vdots & a_1 & \ddots & \ddots & \vdots \\ a_{m-2} & & \ddots & \ddots & 0 \\ a_{m-1} & a_{m-2} & \cdots & a_1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ddots \\ x_m \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{pmatrix}$$

and for $l > m$, w_l is obtained from the linear recursion in $\mathbb{F}_b$,

$$x_l + x_{l-1}a_1 + x_{l-2}a_2 + \cdots + x_{l-m}a_m = 0$$

3 Figure of Merit for polynomial lattice rules

3.1 Walsh space

We consider in this section the functions that can be expressed as an absolutely convergent series by a basis $\{\mathrm{wal}_{b,\mathbf{h}}(\mathbf{x})\}_{\mathbf{h}\in\mathbb{N}_0^s}$.

$$f(\mathbf{x}) = \sum_{\mathbf{h}\in\mathbb{N}_0^s} f_{\mathbf{h}}\,\mathrm{wal}_{b,\mathbf{h}}(\mathbf{x}) \tag{9}$$

3.2 Weighted figures of merit

From this point, we will consider b = 2.
Like in the case of lattice builder, see [5, 6], we compute the figure of merit by choosing the type of the weights, and by using the two following formulas :
For a polynomial lattice rules of size N :

$$P_N = \{\mathbf{y}_k\}_{k=0}^{b^m-1}$$

And for u a subset of $\{1, \cdots, s\}$,
We have

$$D_u(P_N) = \frac{1}{N}\sum_{k=0}^{N-1}\prod_{j\in u} w(y_{k,j}) \tag{10}$$

$$D(P_N) = \sum_{\mathbf{0}\neq u\subset\{1:s\}} \gamma_u\,|D_u(P_N)| \tag{11}$$

where w is a function defined by

$$w(x) = 12.\left(\frac{1}{6} - 2^{\lfloor log_2(x)\rfloor - 1}\right) \tag{12}$$

see [9] for more details on how we obtain this function.
And $\gamma = \{\gamma_u\}_{u\subset\{1:s\}}$ is the a set of non-negative weights which determine the influence of the different dimensions.
The weights can be of two types or their combination :

- **A product weights** where each dimension has a fixed weight and the weight of a subset u is the product of the weights of each composant in u
- **An order-dependent weights** where the weight of u depends only on the size of u.

4 Code

I used the NTL library to construct and evaluate the figure of merite of polynomial lattice rules, because it has two essential properties :

- It allows modular arithmetic on polynomials.
- It provides an arbitrary-precision on floating point numbers. The default precision is 150 bits and the minimum precision is 53 bits.

In the following, the file **tools.h** describes the function used to build PLRs :

```
#include <iostream>

// NTL libraries used :
#include <NTL/GF2X.h>
#include <NTL/mat_GF2.h>
#include <NTL/vec_GF2.h>
#include <NTL/vec_RR.h>
#include <NTL/ZZ.h>
#include <NTL/RR.h>
#include <NTL/vector.h>

using namespace NTL;
using namespace std;

class tools {
public:

/**      Algebra Calculus for Construction of PLR    **/
//Extract coefficient from polynomial p and put them in a matrix.
    static mat_GF2      coeffMatrix( const GF2X& p);
//Extract coefficient from polynomial q and put them in a
//vector of size n.
    static vec_GF2      coeffVector( const GF2X& q, long n);
//Generate the 2^m polynomials over the finite field F_2
    static vec_GF2X     genk( long m);
//  X = V * q mod f
    static void         productMod(vec_GF2X& x, const GF2X& q,
                const vec_GF2X& v, const GF2X& f);
//  Compute the Laurent coefficient of q/p with a precision
//of w in a vector (1 dimension)
    static vec_GF2      laurentVec(const GF2X& q, const GF2X& p,
                long w );
//  Compute the Laurent coefficient of k/p with a precision
//of w in a matrix (1 dimension)
    static mat_GF2      laurentMat(const vec_GF2X& k, const GF2X& p,
                long w );
// given a Laurent coefficient matrix, compute the coordinate
// in a vector of R for one given dimension
    static vec_RR       coord(const mat_GF2& C);
// Extract a n-long vector from vector v.
    static vec_GF2X     extract(const vec_GF2X& v , long n );

 /**     Function to compute the figure of merit     **/
 // Compute the walsh space error.
    static RR           wal( const RR& x);
    static RR           walVec(const vec_RR& v);
//Compute the Order Dependent weight figure of merit of size s.
    static RR           ODWs( const Vec<vec_RR>& plr);
//Compute the Order Dependent weight figure of merit of size 1
// given a vector of weight w.
    static RR           ODW1( const Vec<vec_RR>& plr, const vec_RR& w);

/**      Generation of a PLR   **/
// generate a PLR of generation vector z and irreducible polynomial P
// with precision w.
    static Vec<vec_RR> genplr( const vec_GF2X& z, const vec_GF2X& k,
                const GF2X& p, long w);

// Generate the best PLR of 2^m points with precision w, in s-dimenion
// and with comparing iter times different PLR over a random search
    static Vec<vec_RR> genRandom( long w, long m, long s, long iter );

// Generate the best PLR of 2^m points with precision w, in s-dimenion
// such as the PLR is a Korobov lattice rules
    static Vec<vec_RR> genKorobov( long w, long m, long s, long iter );

// Generate the best PLR of 2^m points with precision w, in s-dimenion
// such as the PLR is constructed component by component
    static Vec<vec_RR> tools::genCBC(long w,long m, long s ,long iter);

/**       Output plr in format to plot       **/
    static void outputPLR( const Vec<vec_RR>& plr, long w);
};
```

5 Some first results

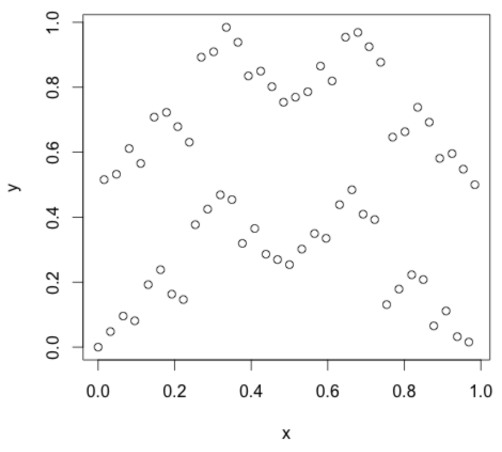


Figure 3: 2^6 PLR points

The following graph shows that if the polynomial P is irreducible and is coprime with each component of the generating vector $\mathbf{z}(X)$, it implies that our lattice is fully projection regular, which means that in each dimesion we have exactly N different point evaluations, see [2].

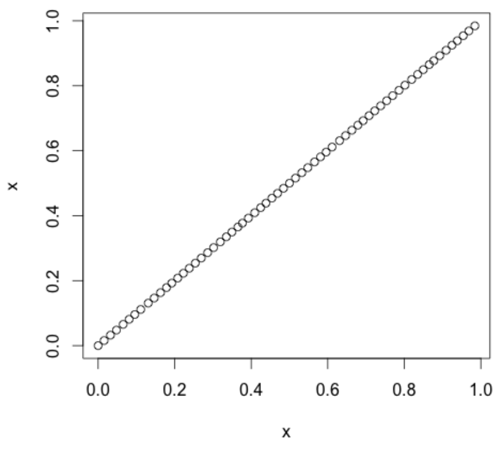


Figure 4: 2^6 fully projection regular rank-1 rule

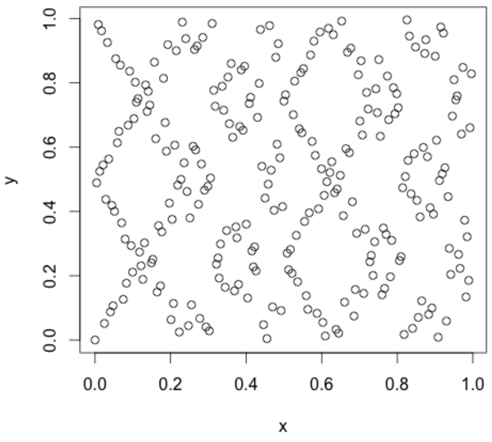


Figure 5: 2^8 PLR points

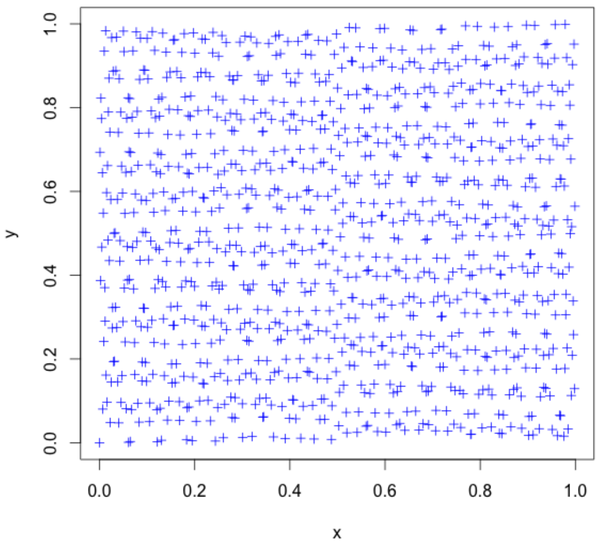
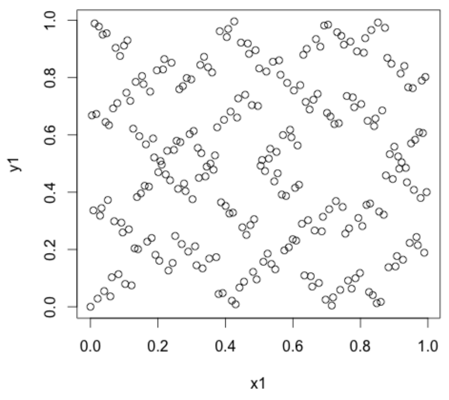


Figure 6: 2^{10} PLR points

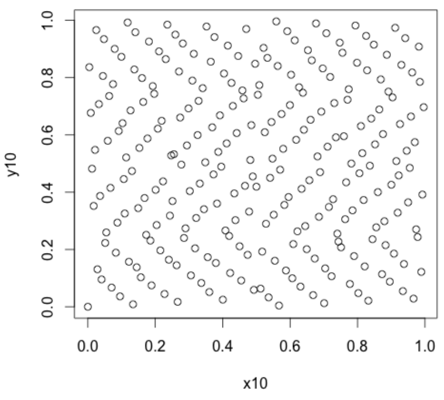
6 Application

6.1 Component by Component construction

In the component by component construction, the generating vector is selected one by one element.

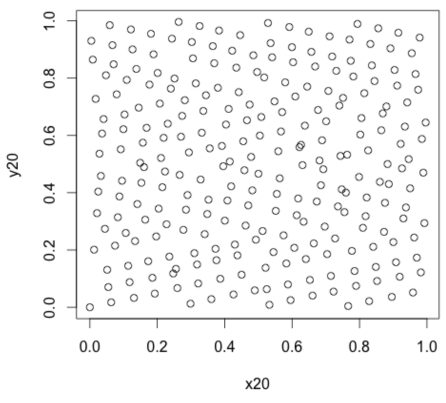


(a) First iteration



//

(b) 10^{th} iteration



(c) 20^{th} iteration

Figure 7: 2^8 CBC consctruction of rule with different iterations in 2 dimension

6.2 Korobov Polynomial Lattice Rules

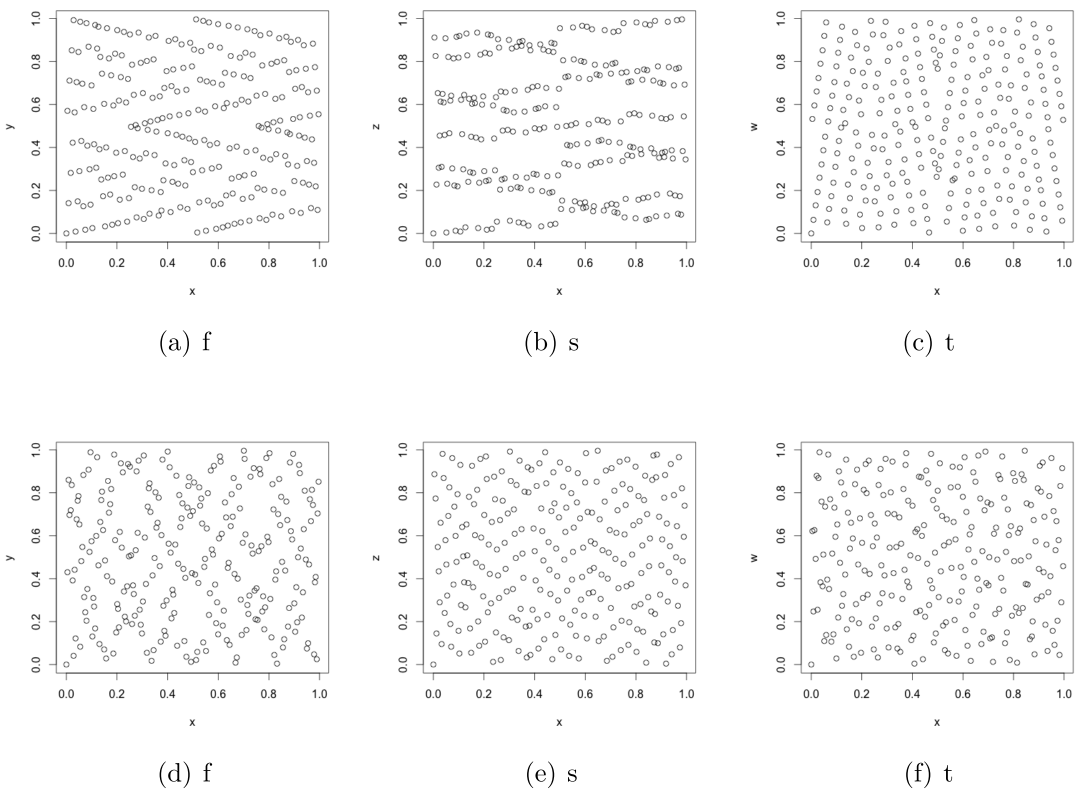
A Korobov polynomial lattice rule is a rank-1 rule whose generating vector, used in equation 6, has the special form

$$\mathbf{z}(X) = \Big(1, a(X), a^2(X) \mod P(X), \cdots, a^{s-1}(X) \mod P(X)\Big) \tag{13}$$

with $a(X) \in G_{b,m}$.

6.3 Random search

In the random search, we explore different generating vector $\mathbf{z}(X)$, and we choose the best one.



(a) f (b) s (c) t

(d) f (e) s (f) t

Figure 8: 2^8 Random consctruction of rule in 4 dimension in iteration 1 *(a,b,c)* and iteration 20 *(d,e,f)*

7 Conclusion and Further work

Lattice Builder is a complex and a complete software that can be used in command lines as well as with its web graphical user interface. The main difference between lattice rules and polynomial lattice rules is on the consctruction of the points and on the function used on the evaluation of the quality criteria.
So I advice the student who wants to integrate my classes into lattice builder in a further work, to get familiar with Waf project for compiling a C++ project, see [7], as well as the principle behind template classes which can be learned through the reference book on templates by A.Alexandrescu see [1].

References

[1] A. Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

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

[3] P. L'Ecuyer. Polynomial integration lattices. In H. Niederreiter, editor, *Monte Carlo and Quasi-Monte Carlo Methods 2002*, pages 73–98, Berlin, 2004. Springer-Verlag.

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

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

[6] D. Munger and P. L'Ecuyer. Lattice builder: A general software tool for constructing rank-1 lattice rules, 2016. see `https://github.com/umontreal-simul/latbuilder/blob/master/README.md`.

[7] T. Nagy. The waf book, 2010-2016. see `https://waf.io/book/`.

[8] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, PA, 1992.

[9] D. Nuyens. The construction of good lattice rules and polynomial lattice rules. In P. Kritzer, H. Niederreiter, F. Pillichshammer, and A. Winterhof, editors, *Uniform Distribution and Quasi-Monte Carlo Methods: Discrepancy, Integration and Applications*, pages 223–255. De Gruyter, 2014.

[10] I. H. Sloan and S. Joe. *Lattice Methods for Multiple Integration*. Clarendon Press, Oxford, 1994.

A Notations

- $\mathbb{N}_0 := \{0, 1, 2, \cdots\}$
- $\mathbb{N} := \{1, 2, \cdots\}$
- $\mathbb{Z}_n$ Residue class ring modulo n
- $\mathbb{F}_b$ Finite field with b elements with b is a prime power
- $\mathbb{F}_b[X]$ Set of polynomials over $\mathbb{F}_b$
- $\mathbb{F}_b(X^{-1})$ Set of formal Laurent Series over $\mathbb{F}_b$
- $G_{b,m} := \{k(X) \in \mathbb{F}_b[X] : \deg(k) < m\}$

B Walsh series in base b

The one dimensional Walsh functions in base b are defined as

$$\text{wal}_{b,h}(x) := e^{2\Pi i (x_1 h_0 + x_2 h_1 + \cdots + x_n h_{n-1})/b} \tag{14}$$

for $x \in [0, 1)$ such as the unique base b expansion of x is

$$x = \textstyle\sum_{i \geq 1} x_i b^{-i},$$

and for $h \in \mathbb{N}_0$ and $0 \leq h < b^m$ such as

$$b = \textstyle\sum_{i \geq 1}^{m-1} h_i b^i$$

The multivarate walsh functions are defined as the product of the one-dimensional Walsh functions

$$\text{wal}_{b,\mathbf{h}}(\mathbf{x}) := \prod_{j=1}^{s} \text{wal}_{b,h_j}(x_j) \tag{15}$$

We can notice that for $\mathbf{h} = 0$ we have $\text{wal}_{b,\mathbf{0}}(\mathbf{x}) = 1$.