Multidimensional Integration Techniques

Jessica Garrett
University of Colorado
Department of Mathematical & Statistical Sciences

December 3, 2014

Abstract

Multi-dimensional integration arises in a variety of disciples including particle physics, PDEs with random coefficients, statistical mechanics, and notably mathematical finance. In this report, we use a variety of methods to estimate a multivariate integral, namely classic Monte Carlo, quasi-Monte Carlo, and lattice techniques. We consider a simple two-dimensional function to study the aspects of each method. Numerical results are provided for each technique as well as comparisons in terms of error convergence between methods. From a two-dimensional domain, we discuss difficulties and limitations in the extension into higher dimensions among the various approaches.

Contents

List of Figures

List of Tables

1 Introduction

Applications of numerical integration are hidden within many economic and physical disciplines including particle physics, statistical mechanics and PDEs with random coefficients [4]. Its most notable appearance is, however, in mathematical finance. Consider the famous Collateralized Mortgage Obligation (CMO) problem involving the estimation of current cash flows from a pool of mortgages [8]. It is typical for a prospective homeowner to sign a 30-year mortgage, with the option of repaying the loan every month. Of course the likelihood of repayment is dependent on interest rate at the given time. As a result, the problem at hand becomes a 360 dimensional (or variable) expected value, as there are 360 possible payment occasions. These types of problems fueled the need for new approaches not yet familiar to the computational field of numerical integration.

There are many techniques for numerical differentiation, or *quadrature*, which produce convincing approximation to the actual integral value. Although simplistic, such a technique was perhaps first explored in an introductory calculus course; the "Trapezoid Rule." Recall a continuous function $f(x)$ over the interval (a, b) whose integral can be approximated using the mere knowledge of

$$\int_{x_0}^{x_1} f(x)\,dx \approx \frac{w}{2}\,(y_0 + y_1) \quad \text{where } w = x_1 - x_0$$

Similarly, Simpson's Rule interpolates three data points yielding an even better approximation using a degree two polynomial. (In fact, MATLAB's built in quad function uses an adaptive Simpson's rule to numerically compute an integral value.) Of course, this begs the question of why must we limit ourselves to just two or three data points when we can partisan our interval into two or more panels? Such composite rules consider the subdivision of our overall interval (a, b) and simply compile the sum over all panel approximations.

These well-known numerical methods are certainly appropriate and practical for functions whose domains are of limited dimensions and are easily separable. However, as R increases, conventional methods quickly become inconvenient, even for a computer. The phrase *the curse of dimensionality*, first stated by Richard Bellman in the mid 1950's, illustrates the difficulties of high dimensional problems [1]. We will explore a popular family of techniques for numerical integration, the methods of Monte Carlo.

There are many variants of Monte Carlo integration, and even within each variant exists multiple ways of definition. It is essential to first understand that the umbrella term can correlate to several unique procedures. The underlying theme, however, considers a point set P_n to approximate finite integrals spanning multiple dimensions. The way in which P_n is defined, gives life to various classes of Monte Carlo estimation. Like most numerical methods, Monte Carlo aims to estimate integral values by preforming a shift from a continuous setting to a discrete.

1.1 Defining Random

Linguistically, the term random may be used in conversation with ease. However, we will see that intuition and mathematical rigor are truly at battle. Random number generation (RNG) has been a subject of debate among mathematicians for decades. There are two main divisions of mathematical randomness, *pseudo* and *quasi*[1] Under *pseudo*-random generation, "samples" or outputs are independent and identically distributed (IID). That is, a given output x is independent of all other x_j, for $i \neq j$ and belongs to the same probability distribution. For example, MATLABs `rand` function considers a uniform distribution, while `randn` uses the normal, or Gaussian distribution. Here, we are in a sense at least "pretending" to be random. The first portion of this paper will address simple and classic Monte Carlo techniques which use strictly (pseudo) random number generation. We will often refer to this process as CMC.

In contrast, under *quasi*-random generation, mutual independence among outputs is sacrificed, and our sample space is no longer IID. Simply put, the "choosing" of one output is directly based on a previous output. This fact alone implies the usage of deterministic sequences for quasi-random generation. Such common sequences are the Halton, Niederreiter, Sobol and Faure sequences (all named after their creator), as they share the same characteristic of *low discrepancy*. We will touch upon some of these sequences and

[1]The prefix pseudo comes from the Greek word meaning "falsehood", implying a misleading appearance of something genuine. Quasi is Latin for "almost".

their characteristics later. Any Monte Carlo technique under quasi-random simulation will often be refereed to as QMC for classic Monte Carlo.

2 Classic Monte Carlo Techniques

2.1 Method 1: Simply Random

Suppose we wish to estimate $\int_\Omega f(x)\,dx$ over some nth-dimensional domain Ω. As a trivial example considering calculating the area of a quarter circle with radius $r = 0.5$.

$$I = \int_0^r \sqrt{r^2 - x^2}\,dx$$

Now consider some easy to compute area, Γ, such that $\Omega \subseteq \Gamma$. For our example, let Γ be defined by the space $[0,r] \times [0,r]$. Generate n random points that lie within Γ and define $\hat{n}$ to be the number of points generated that also lie in Ω. In this procedure, we are preforming a sequence of Bernoulli trials, as each output has exactly two results; either $x_i \in \Omega$ or $x_i \notin \Omega$. The ratio of $\hat{n}$ points that live within Ω verses n total points multiplied by our known area Γ gives us an approximation to I. We will refer to this process as "Method 1".

$$I \approx \Gamma\frac{\hat{n}}{n}.$$

2.2 Method 2: The Formal Definition

Another approach incorporates the function's average value into the process. Recall that the average value of a (continuous) function is defined by

$$f_{average} = \frac{1}{b-a}\int_a^b f(x)\,dx$$

Extending this idea to numerical estimation, consider the set $\{x_i\}$ defined by n random points in Ω, then the average value of f, denoted as μ_n, would be approximated by

$$f_{average} = \mu_n \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i).$$

Then, the estimated integral value, I, would simply be $\mu_n \int_\Omega dx$. Continuing with our example, $\int_\Omega dx$ would just be length $r = 0.5$. Note that when specifically using the phrase "classic Monte Carlo", most texts consider the above expression, yet over the unit hypercube, $\Omega \in (0,1)^d$. Figure 1 below demonstrates the difference between each method.

The MATLAB file `area_circle.m` (see Appendix 7.1) computes Monte Carlo integration using both methods described above for estimating a quarter of a circle with radius 0.5. The simulation is carried out for n =10, 100, 1,000, 10,000 and 100,000 randomly generated points. Note that the script calls the function `rand`, so each output has the same probability of "being hit" Table 1 and Table 2 provides numerical results.

n	Estimate	Absolute Error	Relative Error
10^1	0.225000	2.865046e-02	1.459156e-01
10^2	0.195000	1.349541e-03	6.873155e-03
10^3	0.192500	3.849541e-03	1.960555e-02
10^4	0.198650	2.300459e-03	1.171614e-02
10^5	0.196398	4.795915e-05	2.442539e-04

Table 1: Monte Carlo Method 1

Both estimation techniques produce strikingly precise results. Even with just $n = 100$ randomly generated points, Method 1 yields an output of 0.195, with an absolute error within four decimal places of

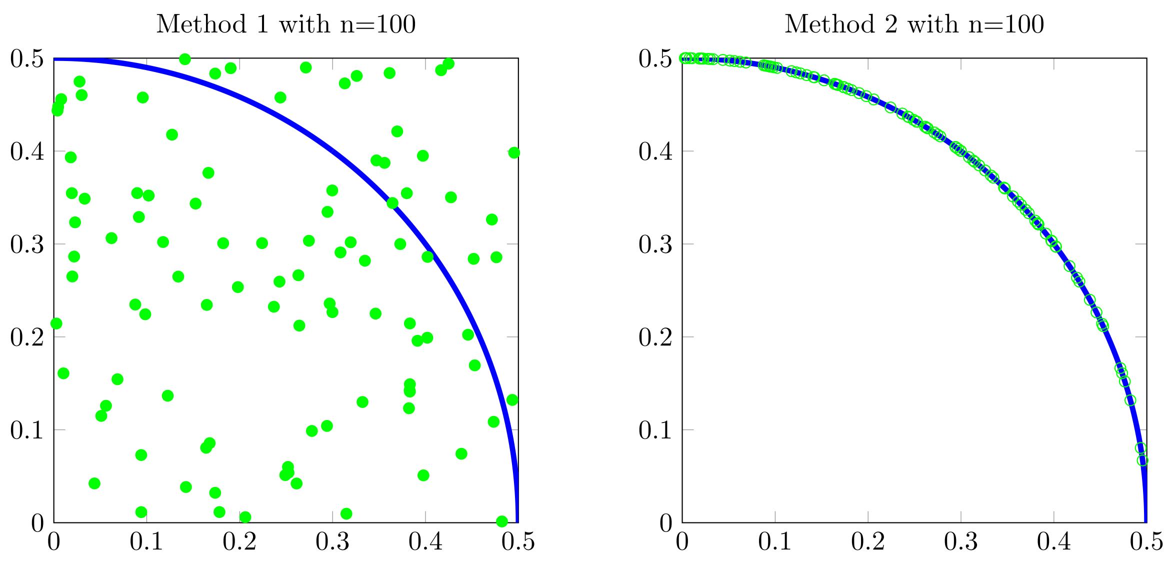


Figure 1: Comparison of Method 1 and Method 2

n	Estimate	Absolute Error	Relative Error
10^1	0.189667	6.682937e-03	3.403592e-02
10^2	0.200077	3.727526e-03	1.898414e-02
10^3	0.194347	2.002913e-03	1.020075e-02
10^4	0.196992	6.424138e-04	3.271787e-03
10^5	0.196511	1.617766e-04	8.239213e-04

Table 2: Monte Carlo Method 2

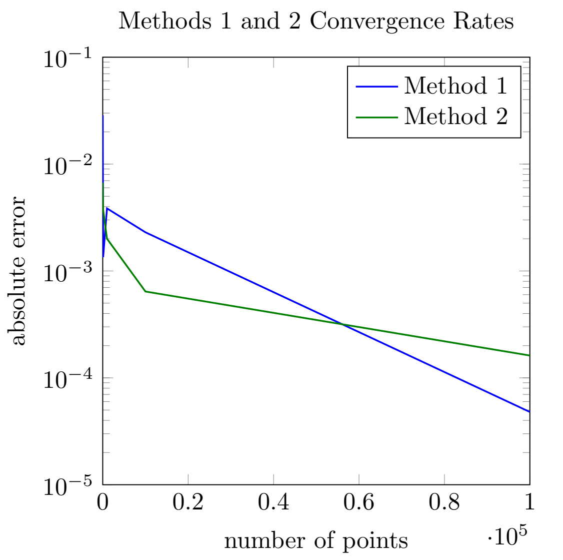


Figure 2: Convergence Rates of Method 1 and Method 2

accuracy. Note that without the presence of a seed in line 17 of `area_circle.m`, the estimated outputs would fluctuate slightly, producing a different error for each simulation. With this in mind, as errors are so close in value between the methods, it would be a fallacy to favor one method over the other based on the above evidence alone. At a glance, we cannot conclude which method is more desirable as Figure 2 displays near identical convergent rates.

2.3 Method 3: The Importance of Sampling

Depending on the integrand, simple Monte Carlo techniques can exhibit poor efficiency when randomly selected points and their associated function values contribute very little to the overall integral. We can improve our estimation by assessing weights to certain regions in Ω with a little knowledge of the behavior of the function at hand. The implementation of a probability density function (PDF) $p(x)$ will allows us to grant precedence of certain regions over others in Ω. In other words, the density of selected points will be greater where function values are greater. Determining an appropriate density function can be challenging but will save us computation time/storage. Consider the following characteristics of a selected $p(x)$.

(i) Like all PDFs, $p(x) \geq 0$ (as probability is never be negative).

(ii) Also, $\int_{-\infty}^{\infty} p(x)\,dx = 1$

(iii) Regions where $f(x)$ is small, $p(x)$ is also small; meaning that there is a low probability that points in this region will be selected.

(iv) Similarly, regions where $f(x)$ is large, $p(x)$ is also large. This insures that we are likely to sample from regions that contribute significantly to the integral.

We formulate an appropriate $p(x)$ based on the behavior of our original function $f(x) = \sqrt{r^2 - x^2}$. One way to approach such a formulation is to first divide the region of interest into n panels. Of course, the more divisions, the better $p(x)$ will fit with $f(x)$. For the sake of simplicity, we divide our region $[0, 0.5]$ into 10 equal panels. This leads to the creation of a discrete piecewise function, $p(x)$. Define w_i to be the midpoint of each panel. So $W = \{w_i : 1 \leq i \leq n\} = [0.025 : 0.05 : 0.475]$. Then, let $p(x)$ is the following step function

$$p(w_j) = \frac{f(w_j)}{\sum_{i=1}^{10} f(w_i)} \qquad \text{for } i, j = 1, 2, \ldots, 10$$

on the interval $[0, 0.5]$ and 0 elsewhere. Note that our $p(x)$ is more precisely a probability mass function (PMF) by definition. Also, notice that characteristic (ii)is satisfied as

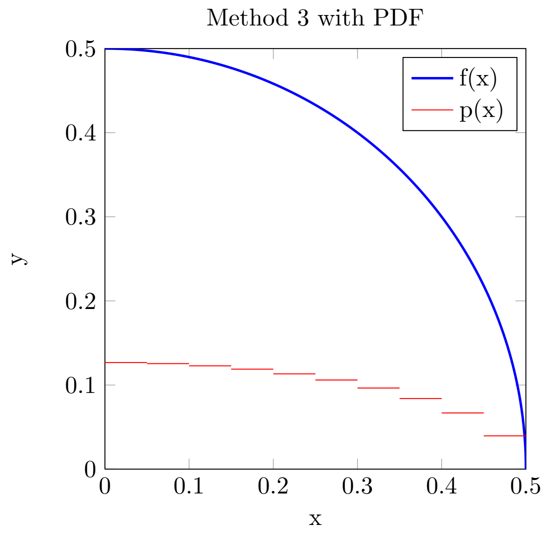
$$\int_{-\infty}^{\infty} p(x)\,dx = \sum_{j=1}^{10} p(x_j) = \frac{f(w_1) + f(w_2) + \cdots + f(w_{10})}{\sum_{i=1}^{10} f(w_i)} = 1$$

Loosely speaking, $p(x)$ tells use what percentage of points should be sampled from the associated jth panel. Let's again carry out simulations with n = 10, 100, 1,000, 10,000, 100,000. (See Appendix 7.2 for script details.)

n	Estimate	Absolute Error	Relative Error
10^1	0.192034	4.316001e-03	2.198121e-02
10^2	0.196620	2.700119e-04	1.375159e-03
10^3	0.196124	2.253721e-04	1.147811e-03
10^4	0.196273	7.697253e-05	3.920179e-04
10^5	0.196352	2.222748e-06	1.132036e-05

Table 3: Monte Carlo Method 3

Notice that (ii) is satisfied as P = [0.1267, 0.1255, 0.1229, 0.1189, 0.1133, 0.1060 , 0.0964, 0.0839, 0.0668, 0.0396] and `sum(P*0.05) = 1`. Figure 3 clearly validates the remaining three conditions, (i),(iii) and (iv). With n = 100,000 we have a computed integral value of 0.196352 with an absolute error of 2.2×10^{-6}. Comparing convergence graphs we see a slightly faster convergence rate with Method 3, specifically when



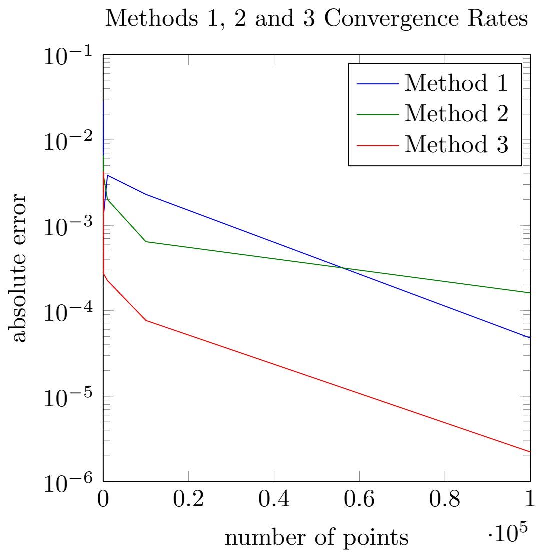


Figure 3: Convergence rates of Method 3

compared to Method 2. Yet these differences are not extremely noteworthy for the same reason mentioned before. Even with added complexities, Method 1 (the simplest in design) yields relatively compatible results to the more elaborate Method 3. This observation is surely a consequence of the chosen function; particularly it's symmetric behavior. Are there better, *faster*, techniques of numerical estimation that we can consider?

3 Quasi-Random and Discrepancy

Recall that under quasi-random generation, independence among outputs is no longer considered. Classic Monte Carlo methods, especially with small values of n, can be unfavorable due to the occurrence of clustering or just the opposite, having pockets of empty space. Quasi-Monte Carlo techniques strive to fix this inefficiency by generating points that better represent the overall domain. As mentioned before, such techniques exploit behaviors of low-discrepancy sequences. Discrepancy speaks to the uniformity of distribution among points in space. In simple terms, a sequence carries low-discrepancy if its numbers are equi-distributed within a given volume or hyper-volume [3]. Formally, discrepancy is defined as,

$$D(J, P_n) = |(A(J)/n) - V(J)|$$

where the region J is a subset of Ω, our region of interest, $A(J)$ is the number of points in J, $V(j)$ is the "volume" of J, n is the total number points sampled and P_n is the point set. More specifically for our example, $\Omega = [0, 0.5] \times [0, 0.5]$ and J is the quarter circle located in Quadrant I so that $V(J) = \pi/16$. In other words, discrepancy is the difference between the proportion of points living in a subspace and the true volume of the space [3]. Of course, if this difference is insignificant in value, then our randomly generated points accurately represent the overall space and the method, or sequence, of generation is of low-discrepancy. The idea of discrepancy can be applied to point sets living in any dimension d. If $D(J, P_n) \leq \mathcal{O}(n^{-1} \ln n^d)$ then the point set is said to have low discrepancy, yet there is no official numerical cut-off.

3.1 Halton Sequence

One commonly used low-discrepancy sequence for quasi-random generation is the Halton sequence, first introduced to numerical integration in the 1950's [9]. The idea is simple. First, let p be a prime number. If we are interested in a total of n quasi-random points over $[0, 1]$, then we would convert each integer 1,2,3...n in base p notation. If the kth integer is represented by $b_i b_{i-1} ... b_2 b_1$, then the kth random number would simply be $0.b_1b_2...b_{i-1}b_i$. In most applications the number generated in base p would then be converted back to decimal form.

For example, let's consider the first prime number, $p = 2$ to be our base and suppose we are interested in generating five random numbers. The set of integers $1, 2, 3, 4, 5$ in base $p = 2$ arithmetic would of course be $1, 10, 11, 100, 101$. Reversing the digits and moving the decimal to the left side, our set of randomly generated numbers is $0.1, 0.01, 0.11, 0.001, 0.101$. Lastly, converting this binary sequence back to decimal form, our final set of the first five numbers of the Halton sequence in base 2 is $0.5, 0.25, 0.75, 0.125, 0.625$. It is important to note that the Halton sequence simply extends the idea of the Van der Corput sequence (1935) into higher dimensions, as the Van der Corput sequence only considers the first dimension and uses base $p = 2$ by definition [9].

The difference between pseudo-random number generation and quasi is best understood visually. Let's compare 500 pseudo-random points in R^2 and 500 quasi-random points in R^6. To be clear, we are generating a Halton sequence in base 2, 3, 5, 7, 11, and 13 as these are the first six prime numbers. For the sake of display, let's only plot two consecutive dimensions together.

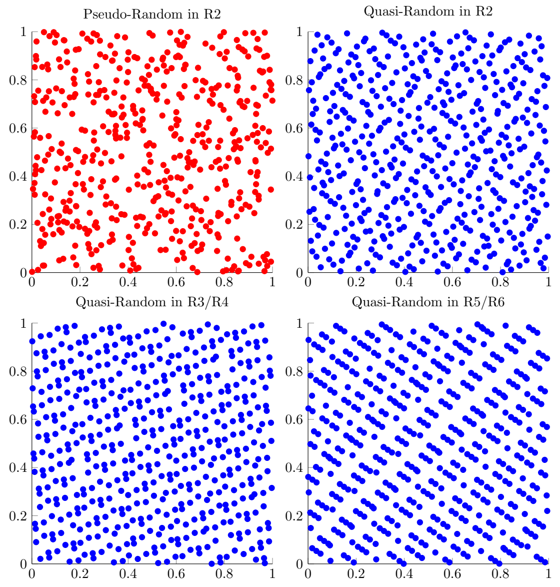


Figure 4: Quasi-Random Generation

We see that under quasi-random generation, points sampled are self-avoiding and cover a space much more uniformly. The three quasi-random plots, R^2 vs R^1, R^3 vs R^4 and R^5 vs R^6 are clearly better for numerical integration purposes in comparison to pseudo-random generation in R^2. However, we do see a considerable amount of "white space" in our last graph, implying that numerical integration aided by quasi-random number generation (defined by the Halton sequence specifically) may lose its appeal for problems living in higher dimensions. We will touch upon this idea later.

3.2 Quasi-Monte Carlo Techniques

Let's consider Methods 1, 2 and 3 now using quasi-random generation. As our problem lives only in R^2, numbers generated in the x direction will be defined by the Halton sequence in base 2 and in base 3 for the y direction.

Referring to Figure 5 we see a comparison for all six Monte Carlo techniques simulated thus far. As with most numerical methods, we are concerned with efficiency; often defined by computation time or number of iterations. For this reason, our region of focus will be of lower values of n. Graphically, our results are interesting about $n = 1000$. Comparing the type of number generation among methods, quasi-Monte Carlo out preforms classic (pseudo) Monte Carlo by a whole order in Methods 1 and 2. In CMC-Method

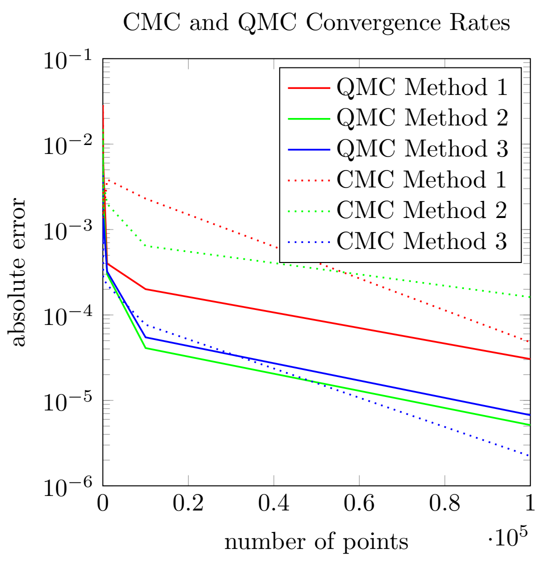


Figure 5: Classic and Quasi MC Convergence Rates

1, absolute error was approximately 2.3×10^{-3} while the QMC absolute error was roughly 2×10^{-4} and 6.4×10^{-4} versus 4.1×10^{-5} respectively for Method 2. However CMC-Method 3 and QMC-Method 3 had similar absolute errors at $n \approx 10{,}000$, yet QMC was still slightly better. Overall CMC and QMC Method 1 did not perform as well as the other four techniques. This result is not surprising as recall that Method 1 simply generated points randomly in both directions while Methods 2 and 3 considered function values in their simulations (although in different ways). In general, QMC Methods produced better estimates to the integral I than its pseudo CMC counter parts. Table 4 summarizes numerical results for QMC Methods 1, 2, and 3. See script `QMCmethods123.m` under Appendix 7.4 for details.

Method	n	Estimate	Absolute Error	Relative Error
Method 1	10^1	0.225000	2.865046e-02	1.459156e-01
	10^2	0.202500	6.150459e-03	3.132403e-02
	10^3	0.196750	4.004592e-04	2.039522e-03
	10^4	0.196550	2.004592e-04	1.020930e-03
	10^5	0.196380	3.045915e-05	1.551272e-04
Method 2	10^1	0.211360	1.501042e-02	7.644742e-02
	10^2	0.198840	2.490747e-03	1.268527e-02
	10^3	0.196650	3.001461e-04	1.528632e-03
	10^4	0.196391	4.098532e-05	2.087365e-04
	10^5	0.196355	5.139678e-06	2.617617e-05
Method 3	10^1	0.197026	6.761737e-04	3.443725e-03
	10^2	0.197756	1.406438e-03	7.162928e-03
	10^3	0.196673	3.235214e-04	1.647681e-03
	10^4	0.196404	5.472564e-05	2.787154e-04
	10^5	0.196356	6.721894e-06	3.423432e-05

Table 4: Quasi-Monte Carlo Methods 1, 2 and 3

4 Lattices Rules

A slightly different quasi-Monte Carlo approach implements the idea of a *lattice*, used almost exclusively in numerical integration. Lattice concepts parallel QMC tactics seen previously as it magnifies the beauty of low-discrepancy. The process of quasi-random number generation in d dimensions is produced simply by defining each coordinate direction as some low discrepancy sequence in d consecutive primes, base p. Here, each dimension was considered separately. In contrast, lattices, in it's most general form, project all necessary dimensions simultaneously through the use of a *generating vector.* The formation of lattice points are so systematic that they form a group under the operation of addition modulo the integers [5]. Sequences used previously for QMC do not reflect this quality however.

In the late 1950's, number theorists (notably Hlawka and Korobov) developed *lattice rules* which served as a guide to which styles of lattices should be used in the context numerical integration [5]. The oldest and most simplistic in design is known as a *rank-1* lattice rule. *Rank* refers to the number of summations used in the expression (in canonical form) [10]. A rank-1 lattice rule is defined as

$$I \approx \frac{1}{N}\sum_{k=1}^{N} f\left(\left\{k\frac{z}{N}\right\}\right)$$

where n is the number of points and z is the generation vector, $z \in Z^d$. The braces indicate that each component to is be replaced by it's fractional part in [0,1). This is why initially, only periodic functions considered the use of a lattice rule as they are cyclic as well [10]. Each z_i in the generating vector is restricted to the set $\{1, 2, ...n-1\}$ and gcd(z_i, n)=1; greatly restricting the defining vector. For rank 1 lattice rules, the point setP_n belongs to a subgroup of the finite additive Abelian, where P_n satisfies the following five axioms of the (additive) Abelian group found in abstract algebra [6]. $\forall z, x, z \in P_n$

(i) Closure: $x + y \in P_n$

(ii) Associative: $(xy) + z = x + (y + z)$

(iii) Identitiy: $\exists a \in P_n$ such that $x + a = a + x = x$

(iv) Inverse: $\exists(-x) \in P_n$ such that $x + (-x) = 0$

(v) Communative: $x + y = y + x$

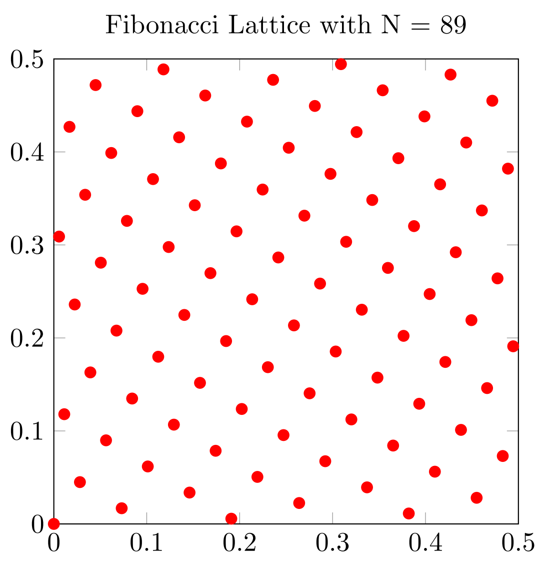
4.1 Fibonacci Lattice Rule

A common rank-1 lattice rule is defined by Fibonacci numbers. With $n = F_k$ points, we defined our generating vector v as $(1, F_{k-1})$, where F_k is the kth number in the Fibonacci sequence. Figure 6 shows both a Fibonacci lattice rule with n=89 points and $z = (1, 55)$ and for comparison, an elementary (grid) lattice, defined simply by the product rule.

4.2 Monte Carlo via Lattice Techniques

4.3 Method 1

Let's now consider how lattice techniques compare to previous CMC and QMC methods. The grid lattice is limited to values of n number of points that satisfy $\sqrt{n} \equiv 0 \pmod 1$ (i.e., n must be a perfect square.). The Fibonacci lattice has even more restrictive on the number of points generated as n must be a Fibonacci number. We will carry out 24 simulations starting with the 2nd Fibonacci number, $n = 1$, followed by the 3rd, $n = 2$, then $n = 3$, $n = 5$, etc., until we reach the 25th Fibonacci number, $n =$ 75,025. For the sake of variety, we will also consider another low-discrepancy sequence for quasi-random number generation the Sobol sequence. Note that this sequence generates identical samples in the first dimension (i.e., the x-direction) as the Halton base-2 sequence. Consequently, results would be identical for Methods 2 and 3, which is why we will not discuss the use of this particular sequence in detail. Figure 7 and Tables 5, 6 and 7 summarizes our results for Method 1 via grid and Fibonacci lattices. (See Appendix 7.7 for script details.)



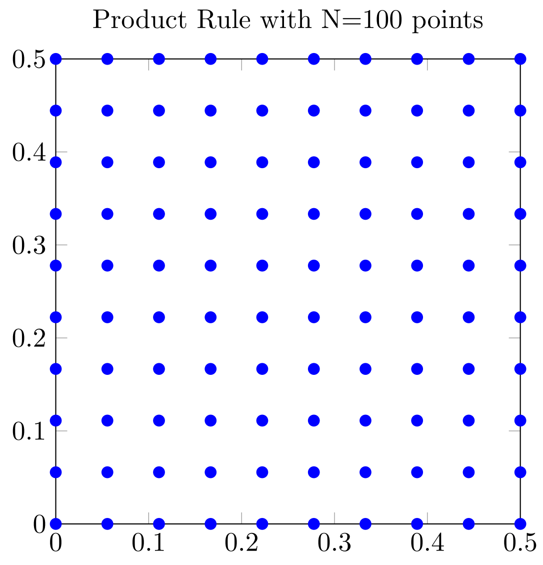


Figure 6: Lattices

n	Estimate	Absolute Error	Relative Error
8	0.218750	2.240046e-02	1.140846e-01
89	0.196629	2.796726e-04	1.424361e-03
987	0.196809	4.589698e-04	2.337514e-03
6765	0.196268	8.198726e-05	4.175577e-04
75025	0.196358	8.339857e-06	4.247454e-05

Table 5: Fibonacci Lattice

n	Estimate	Absolute Error	Relative Error
9	0.222222	2.587268e-02	1.317685e-01
100	0.182500	1.384954e-02	7.053513e-02
961	0.197711	1.361177e-03	6.932418e-03
10000	0.194725	1.624541e-03	8.273719e-03
99856	0.195890	4.599599e-04	2.342556e-03

Table 6: Product Rule Lattice

n	Estimate	Absolute Error	Relative Error
10^1	0.225000	2.865046e-02	1.459156e-01
10^2	0.202500	6.150459e-03	3.132403e-02
10^3	0.196500	1.504592e-04	7.662822e-04
10^4	0.196375	2.545915e-05	1.296624e-04
10^5	0.196348	2.040849e-06	1.039396e-05

Table 7: QMC via Sobol-Method 1

Figure 7 shows convergence rates expressed with exact numerical values, as well as with a linear trend. It is no surprise that the grid lattice converged the slowest as it the most inefficient in design. Notably, it had a similar trend to CMC-Method 1. The Fibonacci lattice, in contrast, performed quite well. What is most alarming is its error at extremely small values of n. For example, at n=89 the estimated integral I is within 4 decimal places of accuracy. At this value of n the Fibonacci lattice produces better results than CMC-Methods 1 and 3, QMC-Methods 1,2 and 3 and the grid lattice for n=100 points. Figure 7 also shows the importance of sequence choice under quasi-Monte Carlo as QMC-Halton and QMC-Sobol display dissimilar convergence rates.

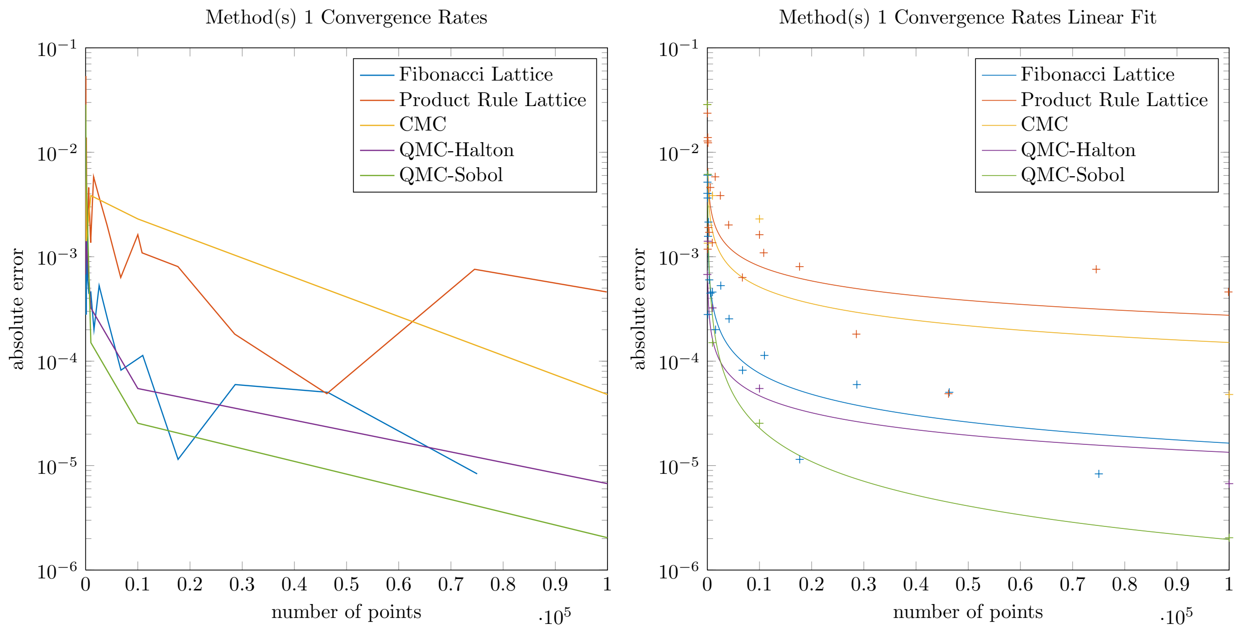


Figure 7: Lattice, QMC and CMC Convergence Rates via Method 1

4.4 Method 2

Recall that Method 2 uses the function's average value, estimated numerical in a discrete setting, to assess it's approximated integral value. Sampled values spanned only in the x direction. In the context of a d= 1 lattice, the generating vector z would simply be 1, clearly not a very interesting lattice. This resulting processes is identical to the well known "left Riemann sum rule" where the domain is divided into n equal rectangles or panels. For comparison, we will consider a general rank-1, 1-dimensional "lattice" for $n = 10^k$ for k = 1,2,...,5. Refer to Figure 8 for a comparison of all variants of Method 2 and Table 8 for numerical results of Method 2 via lattice.

n	Estimate	Absolute Error	Relative Error
10^1	0.206532	1.018285e-02	5.186085e-02
10^2	0.197526	1.176524e-03	5.991986e-03
10^3	0.196472	1.226758e-04	6.247829e-04
10^4	0.196362	1.242650e-05	6.328765e-05
10^5	0.196351	1.247676e-06	6.354360e-06

Table 8: Lattice Method 2

4.5 Method 3

Here, we will use the same "lattice" as seen in Method 2. Under Method 3, we assigned weights to appropriate regions in Ω, through the use of a PDF, $p(x)$. In the context of our problem, $p(x)$ indicated what percentage of overall n points would fall in each of the 10 panels. We will need to scale and shift our lattice appropriately

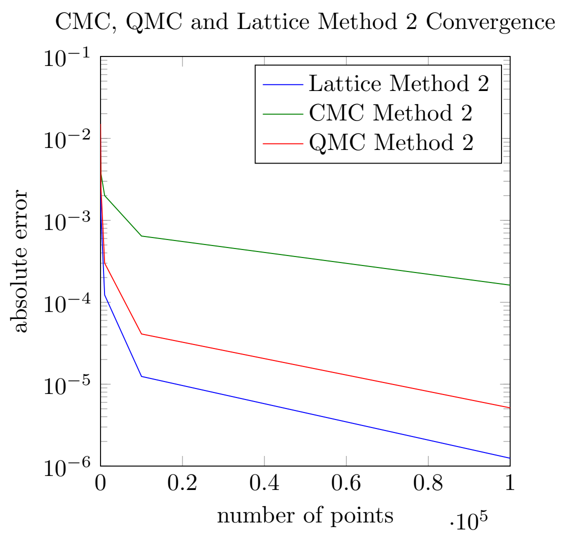


Figure 8: Convergence Rates of Method 2 Techniques

as seen in QMC-Method 3 with the Halton sequence. For example, for n=100, the PDF assigned 12 points to panel 2, spanning $[0.05, 0.1]$. Our lattice with n= 100 is simply defined by the set

$$P_{100} = \{k\tfrac{1}{100}\} \text{ for } k = 1,2,..,100$$

with P_{100} cycling back to its fractional part in $(0, 1]$, 0. Yet, we must scale down the lattice by $r/10 = 0.05$ (as there are 10 panels) and preform a positive shift 0.05, as we are in the second panel. The resulting point set for panel two would be the first 12 points in the above lattice, scaled and shifted as described;

$$P_{12} = 0.05\{k\tfrac{1}{100}\} + 0.05 \text{ for } k = 1,2,..,12, \text{ within the region } (0.05, 0.1].$$

Figure 9 compares convergence rates for all three techniques under Method 3. The lattice technique did not perform well compared to CMC and QMC Methods 2. Looking at the second figure, we can see exactly how x values (and their corresponding y evaluations) were selected for $n = 100$ total points. Notice that points were selected from the left side of each panel, as defined by our rank-1, d=1 lattice. Referring to Table 9, the resulting computed integral value is over-estimated four out of the five simulations, with the exception of $n = 10$. If, in contrast, our function was monotonically increasing, we would then see an under-estimation. Overall, the Fibonacci lattice under Method 1 provided the most precise integral estimations for low values of n in comparison lattice Methods 2 and 3. See Appendix 7.8 for details about lattice Methods 2 and 3.

n	Estimate	Absolute Error	Relative Error
10^1	0.181532	1.481715e-02	7.546310e-02
10^2	0.201422	5.072678e-03	2.583494e-02
10^3	0.201044	4.694347e-03	2.390811e-02
10^4	0.201013	4.663402e-03	2.375051e-02
10^5	0.201011	4.661143e-03	2.373901e-02

Table 9: Lattice Method 3

5 Results

We have considered now three underlying classes of Monte Carlo integration; classic Monte Carlo, quasi-Monte Carlo, and lattice techniques. And within each class, we have defined three unique methods. Looking

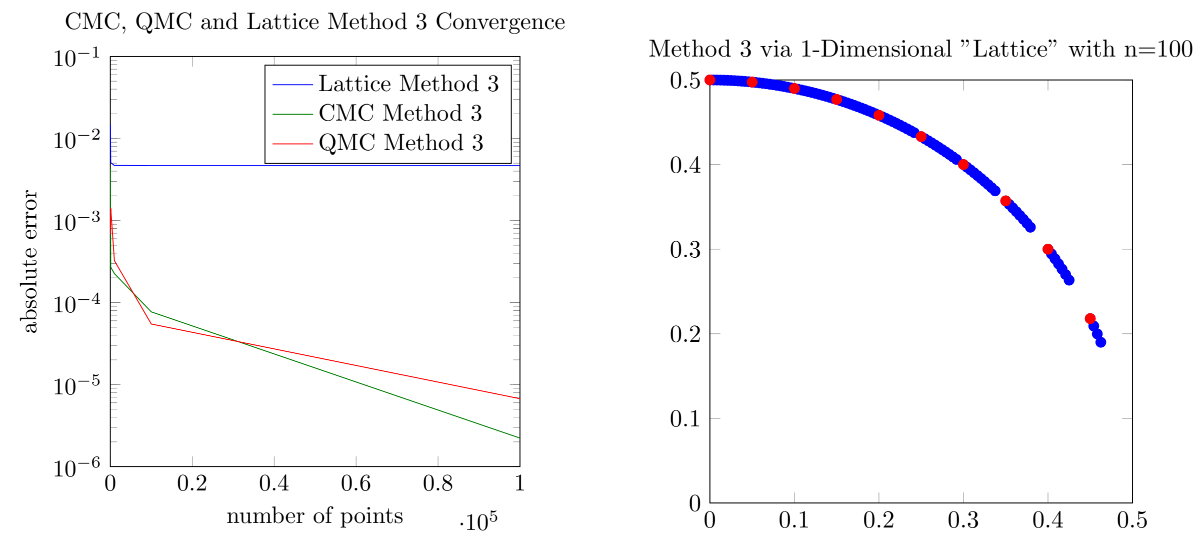


Figure 9: Convergence Rates of Method 3 Techniques

at small values of n, QMC methods out-performs CMC within all three methods. At n=1,000 QMC (via Halton sequence) Methods 1, 2 and 3 produce precise estimations to the integral I with absolute errors of 4×10^{-4} and 3×10^{-4}, 3.24×10^{-4} respectively. QMC-Method 1, define via the Sobol sequence, produced an absolute error of the same order as its brother sequence, Halton. The Fibonacci lattice is a respectable contender as it owns one of the fastest convergence rates among Method 1 techniques. At n=89 the Fibonacci lattice via Method 1 yielded an estimated value of 2.8×10^{-4}. This scope of precision wasn't achieved until n=1000 points for QMC Methods 1, 2 and 3, n=10,000 for CMC-Method 2, and n=100,000 for CMC-Method 1. An extra simulation of n=1,000,000 would be needed in order to produce this type of accuracy using our simple grid lattice.

Referring to Figure 5, the convergence rates of CMC and QMC between each method differed by roughly an entire order, with the exception of Method 3. Why was this difference between CMC and QMC not as striking for Method 3? Recall the implementation of a probably density function in Method 3. This design assigns weights to particular regions where, the function holds more area or volume. For our example, we divided our region x = [0,0.5] in 10 different panels of equal widths. The resulting $p(x)$ was simply the step function defined in the x direction by these divisions and in the y direct by "function importance" to the overall integral. Note that our $p(x)$ lives in a discrete setting. Consider the n=10,000 iteration where even the smallest number of samples taken was six from panel the last panel, 10. Specifically, we are taking six random numbers from a width of just r/10 = 0.05. One could conclude that the type of generation, quasi or pseudo does not make a difference under such a small domain. Of course this begs the question of setting the number of panels to approach ∞. Surely, if the $p(x)$ used in Method 3 was precisely a probability *density* function, the error difference in quasi-Monte Carlo and classic Monte Carlo simulations would be nonexistent.

Again, in light of Figure 5, QMC Method 2 computed an approximated integral value of 0.196391 compared to the actual value of 0.19635. This was the best estimation at n=10,000 among all techniques. This approach serves as a great example of how certain methods can favor particular integrands due to properties of function at hand. Our function, $f(x) = \sqrt{0.5^2 - x^2}$ is monotonically decreasing within the interval $[0, 0.5]$. Notice that Figure 1 shows a low density of y function evaluations in the region $[0.4, 0.5]$ specifically where the function is changing rapidly. This is no surprise as recall that Method 2 is generating random samples (either pseudo or quasi) across the x direction and not along the function itself. Consequently, the number of expected function evaluations within a region is dependent on the functions variability within that region. Conversely, the region [0,0.1] contains a high density of function evaluation as it's slope is very close to one. This aspect is beneficial to integral estimation via Method 2 as function values are coincidently small where variability are large. Method 2 is deceiving in portraying efficiency, yet the result is a consequence of our integrand, rather than pure cleverness of design. Favoring techniques based on the function at hand can also be said about the Fibonacci lattice rule, yet this rule is often ideal in d=2 regardless of concerned function's characteristics [5].

Lattices, by comparison, showed the benefit of QMC methods defined by a low-discrepancy sequence. In general, the use of the term *sequence* implies that the construction of such a point set is independent of the number of points n in the set. However, a lattice is defined by two variables, one being n and the other being the generating vector, z so it's design is very much a result of n. Lattice-Method 3 demonstrates the hazards of using of point set that is nonextensible in n as it's convergence rate remains relatively stagnate among various values of n. Although we used the same lattice in Method 2, we implemented the entirety of the point set across the region $(0, 0.5]$, so it's convergence rate did not suffer as the value of n did not change.

6 Discussion: A Glimpse into Higher Dimensions

The Monte Carlo techniques (classic, quasi and via lattice) discussed thus far have been in the d=2 dimension space. An extension into higher dimensions causes a change in accuracy among methods as well as a need for a revaluation of the blueprint completely. In other words, convergence rates and error bounds are dimension d dependent for many variations of Monte Carlo, with the exception of the (pseudo) CMC. This fact alone has fueled the need for methods which are deterministic in structure yet with an added element of randomness.

Classic Monte Carlo methods using pseudo-random number generation are desirable in the since that we know much more about its precision. As we are pulling IID samples from a known, uniform distribution, we can compute an expected error. The variance of a random variable Y is $E[(Y - E(Y))^2]$ where the notation $E[Y]$ is the expected value of Y. If we define Y to be the sum of all randomly selected function values divided by n, then

$$E[\frac{Y_1 + Y_2 + ... + Y_n}{n}] = \frac{nA}{n} = A$$

where A is the function's average value defined in section 2.2. The variance of Y would then be

$$E[(\frac{Y_1 + Y_2 + ... + Y_n}{n} - A)^2] = \frac{1}{n^2} \sum E[(Y_i - A)^2] = \frac{1}{n^2} n\sigma^2 = \frac{\sigma^2}{n}$$

where σ^2 is the variance of the function f (the variance of each Y_i). The resulting expected error is $\frac{\sigma}{\sqrt{n}}$ Consequently, the rate of convergence is $\mathcal{O}(n^{-1/2})$ [5]. Although this is a relatively slow rate, it is independent of dimension, the most appealing aspect of this approach. Note that these statistics refer to CMC-Method 1 and 2 under 3.1 and 3.2 exclusively.

The benefit of quasi-random number generation is that samples are more uniformly distributed, resulting in a better representation of the overall space as seen in Figure 4. Although QMC methods generally are more accurate with small values of n than CMC techniques in the context of numerical integration, they come with little know knowledge of an error estimate. At best, a broad representation of accuracy is often given in the form of a Koksma-Hlawka inequality [2].

$$|I - Q_{n,d}(f)| \leq |D^*(P_n)V(f)|$$

where I is the actual integral value and $Q_{n,d}(f)$ is specifically QMC-method 2 with n number of points generated in d dimensions. $V(f)$ is the variance of f and $D^*(P_n)$ is the *star discrepancy* of the point set $P_n = \{x_i : i = 1..n\}$ defined simply as the worst case discrepancy of P_n; $D^*(P_n) = sup|D(J, P_n)|$ where $D(J, P_n)$ is the (*local*) discrepancy described in section 4. *Star* refers to the fact that all vertices's of J must be anchored at the origin. *Extreme* dependency does not have this requirement. In this setting, estimated error depends solely on star discrepancy. For this reason, many texts consider the convergence of QMC to be $\mathcal{O}(n^{-1}(\ln n)^d)$. QMC-Method 1 has a slightly different error rate; $\mathcal{O}(n^{-\frac{1}{2}-\frac{1}{2d}})$ [9].

Even with ideal low-discrepancy sequences, like Halton or Sobol, star discrepancy increases with dimension. Not only is the error estimation for QMC techniques nonextensible in d, it's use isn't even practical for *large* values of d; many consider the cut off to be a conservative $d = 14$. See Figure 10 for plots of quasi random generation defined by the Halton sequence in R^{22} (displayed in dual consecutive dimensions.)

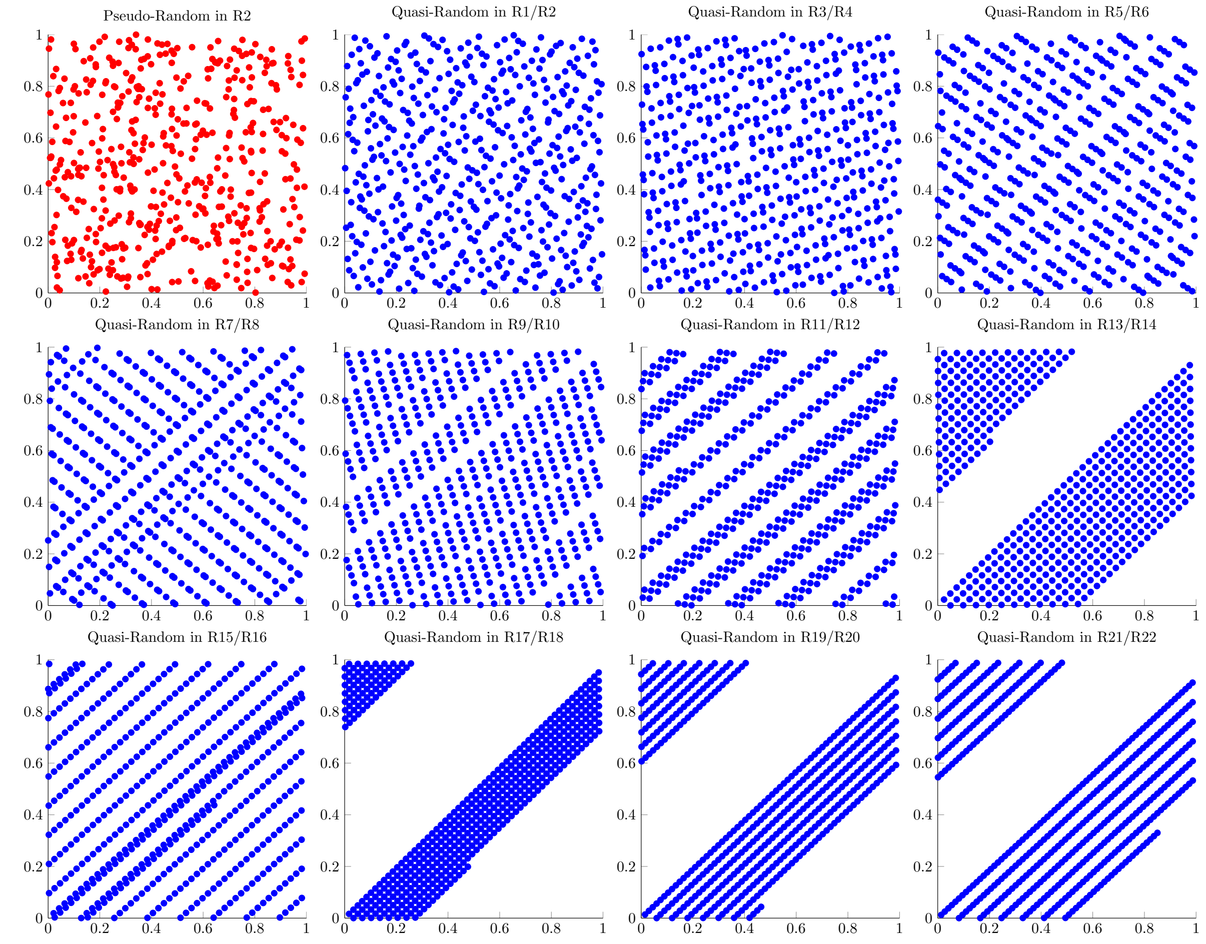


Figure 10: Halton sequence in dual consecutive dimensions

Lattice rules bring unique design to the forefront. In our example, we used a well-known lattice defined by a rank-1 Fibonacci lattice rule. We used the generating vector $z = (1, F_{k-1})$ where z_2 is the F_{k-1} Fibonacci number to generate F_k lattice points in the initial region [0,1). In general, a common way to construct a lattice rule in d dimensions is to define the generating vector in *Korobov* form

$$z = (1, a, a^2, ..., a^{d-1}) \bmod n, \text{ where } 1 \le a \le n-1 \text{ and } gcd(a, n) = 1 \text{ [2].}$$

Here, we see a recipe of how lattice rules can extend into higher dimensions. However, in practice the challenge is defining appropriate variables. Given a fixed n and d, an optimal a can be chosen in order to minimize star discrepancy, and therefore our error bound. Yet, if d or n were to change, then the original value of a may no longer be optimal in minimizing star discrepancy. The Korobov generating vector is far from a one size fits alls solution.

A "componet-by-componet" construction (CBC) of z aims to extend d into higher dimensions as needed without having to reconstruct the vector completely. This process imposes a *greedy* algorithm where each component, z_i is selected to minimize error. The algorithm is greedy by definition because at each iteration the choice of z_i is solely decided based on minimizing the error of the current vector $z = z_1, z_2, ..., z_i$ and does not consider how future z_{i+1} will effect the resulting finalized z and its error. Most algorithms set $z_1 = 1$ as seen in the general Korobov form. Still, a CBC construction of z is nonextensible in n. A proof by Hickernell and Niederreiter in 2003 showed that an extensible lattice, in d and in n, does exist, yet provided no insight into how one could find such a vector z [2].

Even with a lattice defined by a CBC vector z or of the Korobov form, practical knowledge of error is still at best represented as a bound (Koksma-Hlawka inequality). One way to dismiss the deterministic quality of a lattice rule is to incorporate a *random shift* Δ. A *shifted lattice* takes the form $\{\frac{kz}{n}+\Delta\}$ where $k = 0, 1, ...n-1$ and $\Delta \in [0,1)^d$ where, again, the braces indicate the element's fractional representation. This shift is an IID random sample, equipped with a known probabilistic error estimation, seen similarly in classic Monte Carlo. Like all "non-classic" Monte Carlo techniques a shifted lattice rule aims simply to have a *better* convergence rate than $\mathcal{O}(n^{-1/2})$. In fact, under specific settings shifted lattice rules have an estimated error bound of $\mathcal{O}(n^{-1+\delta})$ where $\delta > 0$ [11]. This, of course, is independent of d and is more optimal than CMC's error estimation.

In conclusion, our simple example in d=2 dimensions, where $f(x) = \sqrt{r^2 - x^2}$, we saw that QMC methods out preformed CMC methods, especially for low values of n, as quasi-random generation samples points deterministically and therefore more uniformly. In summary, classic Monte Carlo approaches are dimension adapted; allowing construction into higher dimensions with ease. However, its appeal both start and stop at extendability as classic Monte Carlo approaches carry slow convergence rates in practicality. In certain settings, QMC methods are often favorable in the context of accuracy yet suffer the *curse of dimensionality*. Not only are QMC methods dependent of d, they loose their appeal completely at relatively low values of d. Lattice rules produced convincing estimations in d=2 and are of a special class of quasi-Monte Carlo. An extension of d however, causes a new construction of the vector z, which can be a costly task. (Unshifted) lattices still give no practical estimated error; notably its most unappealing aspect. A lattice defined by a CBC vector is still considered *embedded* as n is still bounded. Ideally, we hope to develop numerical integration techniques which are extensible in both d and in n yet have faster (known) convergence rate than $\mathcal{O}(n^{-1/2})$. The future strives to develop hybrid methods which house benefits of classic Monte Carlo quasi-Monte Carlo and lattice-rule approaches.

7 Appendix: MATLAB scripts

7.1 area_circle.m

```
%Method 1 and 2 MC Integration

% r = radius of circle, 0.5
% f = function for 1/4 circle
% n = number of points
% num = row vector representing various values of n
% bool = boolean matrix
% inside = total number of points that lie within concerned region
% rat = ratio of inside vs. total
% e1 = estimated area via MC "method 1"
% e2 = estimated area via MC "method 2"
% abserr = absolute error
% relerr = relative error

clear;clc;close all;

rng(2014); %define seed
format long
n = [10 100 1000 10000 100000];
r = 0.5;
actual = (pi*r^2)/4;

for i = 1:5;
    num = n(1,i);
    x{i} = rand(num,1)*0.5;  %x direction
    y{i} = rand(num,1)*0.5;  %y direction
    f{i} = sqrt(r.^2 - x{i}.^2); %function evaluations for random x's
    bool{i} = y{i}≤f{i};   %returns T/F
    inside(i) = sum(bool{i});
    ratio(i) = inside(i)/num;
    e1(i) = ratio(i)*r^2; %integral estimation via "method 1"
    e2(i) = (sum(f{i})/num)*r; %integral estimation via "method 2"
```

```matlab

%Determine Error
    abserr_e1(i) = abs(e1(i)-actual);
    abserr_e2(i) = abs(e2(i)-actual);
end

relerr_e1 = abserr_e1 / actual;
relerr_e2 = abserr_e2 / actual;


%Display Results
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('    N     |    ESTIMATE    |    ABSOLUTE ERROR    |    RELATIVE ERROR    \n');
fprintf('\n');
fprintf('                          Monte Carlo Method 1                            \n')
fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for k = 1:5
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n',n(k), e1(k), ...
        abserr_e1(k), relerr_e1(k))
end
fprintf('\n');
fprintf('                          Monte Carlo Method 2                           \n');
fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for k = 1:5
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n',n(k), e2(k), ...
        abserr_e2(k), relerr_e2(k))

end
return

%Plots :)
f1 = figure;
%  subplot(2,2,1)
xx = 0:0.0001:0.5;
yy = sqrt(r.^2-xx.^2);
plot(yy,xx,'Linewidth',2); hold on;
scatter(x{2},y{2},'g.');
title('Method 1 with n=100')
axis([0 0.5 0 0.5]);
axis square
matlab2tikz('figurehandle', f1, 'MC1_1.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);

f2 = figure;
%  subplot(2,2,3)
plot(yy,xx,'Linewidth',2); hold on;
scatter(x{2},f{2},'go');
title('Method 2 with n=100')
axis([0 0.5 0 0.5]);
axis square
matlab2tikz('figurehandle', f2, 'MC1_2.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);

f3 = figure;
%  subplot(2,2,2)
semilogy(n,abserr_e1)
xlabel('number of points')
ylabel('absolute error')
title('Method 1 Convergence')
%xlim([10,100000])
axis([0 10^5 10^-5 10^-1]);
axis square
matlab2tikz('figurehandle', f3, 'MC2_1.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);

```

```
f4 = figure;
%  subplot(2,2,4)
semilogy(n,abserr_e2);
xlabel('number of points')
ylabel('absolute error')
title('Method 2 Convergence')
%xlim([10,100000])
axis([0 10^5 10^-5 10^-1]);
axis square
matlab2tikz('figurehandle', f4, 'MC2_2.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);
```

7.2 MCmethod3.m

```
%Method 3 MC Integration

clear;clc;close all;

rng(2014); %define seed
r = 0.5;
xmidd = 0.025:0.05:0.475;
xinterval = 0:0.05:r;
f = @(x) sqrt(r.^2 - x.^2);
ymidd = f(xmidd);
ysum = sum(ymidd);
actual = (pi*r^2)/4;

%Preallocate **is this really necessary?
e3 = zeros(1,5);
p = zeros(1,10);

for n = 1:5
    num = 10^n;
    for i = 1:10
        p(i) = f(xmidd(i))/ysum;
        pts_int(i) = p(i)*num;
    end

    %pts_int needs to be all integer values
    rnd_pts = round(pts_int);

%-----------If simple round function does not sum to n...---------------%

    if sum (rnd_pts) < num %need to add a point in appropriate panel
        for k = 1:10;
            abs_diff(k) = abs(rnd_pts(k) - pts_int(k));
        end
    [val index] = max(abs_diff);
    rnd_pts(index) = rnd_pts(index) + 1; %point added
    end

    if sum (rnd_pts) > num %need to subract a point in appropriate panel
        for k = 1:10;
            abs_diff(k) = abs(rnd_pts(k) - pts_int(k));
        end
    [val index] = max(abs_diff);
    rnd_pts(index) = rnd_pts(index) - 1; %point subtracted
    end

    s = sum(rnd_pts); %check

 %----------------------------------------------------------------------%

        %Create "Random" X Values
    srt = 1;
    for j = 1:10 %# of panels
```

```
        last = srt + rnd_pts(j) - 1;
        xm3(srt:last) = rand(rnd_pts(j),1)*0.05 + xinterval(j);

        %Evaluate Function Values
        ym3(srt:last) = f(xm3(srt:last));
        area(j) = (0.05)*((sum(ym3(srt:last)))/rnd_pts(j));
        srt = last + 1;
    end
    e3(n) = sum(area);

    %Determine Error
    abserr_e3(n) = abs(e3(n)-actual);
end

relerr_e3 = abserr_e3 / actual;

%Display Results
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('                          Monte Carlo Method 3                           \n')
fprintf('\n');
fprintf('    N     |    ESTIMATE     |    ABSOLUTE ERROR     |    RELATIVE ERROR    \n');
fprintf('----------|----------------|----------------------|----------------------\n');
for k = 1:5
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n',10^k, e3(k), ...
        abserr_e3(k), relerr_e3(k))
end

%Plot Results
f1 = figure;
semilogy([10,100,1000,10000,100000],abserr_e3)
xlabel('number of points')
ylabel('absolute error')
title('Method 3 Convergence')
axis square
matlab2tikz('figurehandle', f1, 'MC3_1.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);
```

7.3 halton.m

```
function [ u ] = halton( p,n )
%Halton sequence in base (prime) p.
%Generates n quasi-random points in [0,1].

if n ==1
    u = 1/p;
    return;
end

b = zeros(ceil(log(n)/log(p)),1);  %largest number of digits
for j = 1:n
    i = 1;
    b(1) = b(1)+1;             %add one to current integer;
    while b(i) > p-1+eps       %this loop does carry out in base p
        b(i) = 0;
        i = i + 1;
        b(i) = b(i) + 1;
    end
    u(j) = 0;
    for k = 1:length(b(:))     %add upp reverse digits
        u(j) = u(j) + b(k)*p^(-k);
    end
end


end
```

7.4 QMCmethods123.m

```
% QMC Integration: Methods 1,2 and 3

% r = radius of circle, 0.5
% f = function for 1/4 circle
% n = number of points
% inside = total number of points that lie within concerned region
% e1q = estimated area via QMC "method 1"
% e2q = estimated area via QMC "method 2"
% e3q = estimated area via QMC "method 3"
% abserrq = absolute error
% relerrq = relative error

clear;clc;close all;

rng(2014); %define seed
format long
k = 5;
r = 0.5;
n = 10^k;
actual = (pi*r^2)/4;
x = halton(2,n)'*r;
y = halton(3,n)'*r;
f = sqrt(r.^2 - x.^2);
inside = (x.^2 + y.^2) <= r^2;

%Method 1 and Method 2
for i = 1:k;
%integral estimation via "method 1"
    e1q(i) = r^2 * sum(inside(1:10^i)) / (10^i);
%integral estimation via "method 2"
    e2q(i) = r*sum(f(1:10^i))/(10^i);

%Determine Error
    abserr_e1q(i) = abs(e1q(i)-actual);
    abserr_e2q(i) = abs(e2q(i)-actual);
end

relerr_e1q = abserr_e1q / actual;
relerr_e2q = abserr_e2q / actual;

%Display Results
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('    N      |    ESTIMATE    |    ABSOLUTE ERROR     |    RELATIVE ERROR    \n');
%fprintf('\n');
fprintf('                       Quasi Monte Carlo Method 1                           \n')
%fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for k = 1:5
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n',10^k, e1q(k), ...
        abserr_e1q(k), relerr_e1q(k))
end
%fprintf('\n');
fprintf('                       Quasi Monte Carlo Method 2                           \n');
%fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for k = 1:5
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n',10^k, e2q(k), ...
        abserr_e2q(k), relerr_e2q(k))

end

%Method 3
xmidd = 0.025:0.05:0.475;
```

```
xinterval = 0:0.05:r;
f = @(x) sqrt(r.^2 - x.^2);
ymidd = f(xmidd);
ysum = sum(ymidd);

%Preallocate
e3q = zeros(1,5);
p = zeros(1,10);

for n = 1:5
    num = 10^n;
    for i = 1:10
        p(i) = f(xmidd(i))/ysum;
        pts_int(i) = p(i)*num;
    end

    %pts_int needs to be all integer values
    rnd_pts = round(pts_int);

%-----------If simple round function does not sum to n...---------------%

    if sum (rnd_pts) < num %need to add a point in appropriate panel
        for k = 1:10;
            abs_diff(k) = abs(rnd_pts(k) - pts_int(k));
        end
    [val index] = max(abs_diff);
    rnd_pts(index) = rnd_pts(index) + 1; %point added
    end

    if sum (rnd_pts) > num %need to subract a point in appropriate panel
        for k = 1:10;
            abs_diff(k) = abs(rnd_pts(k) - pts_int(k));
        end
    [val index] = max(abs_diff);
    rnd_pts(index) = rnd_pts(index) - 1; %point subtracted
    end

    s = sum(rnd_pts); %check

    % since we know how long the sequence will need to be, create it and store it
    h = halton(2,max(rnd_pts));
 %-----------------------------------------------------------------------%

        %Create "Q-Random" X Values
    srt = 1;
    for j = 1:10 %# of panels
        last = srt + rnd_pts(j) - 1;
        xm3(srt:last) = h(1:rnd_pts(j))'*0.05 + xinterval(j);

        %Evaluate Function Values
        ym3(srt:last) = f(xm3(srt:last));
        area(j) = (0.05)*((sum(ym3(srt:last)))/rnd_pts(j));
        srt = last + 1;
    end
    e3q(n) = sum(area);

    %Determine Error
    abserr_e3q(n) = abs(e3q(n)-actual);
end

relerr_e3q = abserr_e3q / actual;

%Display Results
%fprintf('\n');
fprintf('                       Quasi Monte Carlo Method 3                           \n')
%fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for k = 1:5
```

```
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n',10^k, e3q(k), ...
        abserr_e3q(k), relerr_e3q(k))
end

%Plot convergence rates for QMC
figure
num = [10 100 1000 10000 100000];
semilogy(num,abserr_e1q,'r',num,abserr_e2q,'g',num,abserr_e3q,'b');
xlabel('number of points')
ylabel('absolute error')
title('QMC Convergence Rates')
axis square
legend('QMC Method 1', 'QMC Method 2', 'QMC Method 3')

%Plot rates for QMC and CMC
abserr_e1 = [0.028650459150638   0.001349540849362   0.003849540849362   0.002300459150638   ...
    0.000047959150638];
abserr_e2 = [0.006682937312325   0.003727526218243   0.002002912520977   0.000642413785594   ...
    0.000161776561111];
abserr_e3 = [0.004316000562404   0.000270011858544   0.000225372123501   0.000076972525891   ...
    0.000002222747864];

f5 = figure;
semilogy(num,abserr_e1q,'r', num,abserr_e2q,'g', num,abserr_e3q,'b', num,abserr_e1,'r:', ...
    num,abserr_e2,'g:', num,abserr_e3,'b:', 'LineWidth', 0.65);
xlabel('number of points')
ylabel('absolute error')
title('CMC and QMC Convergence Rates')
axis square
legend('QMC Method 1', 'QMC Method 2', 'QMC Method 3','CMC Method 1', 'CMC Method 2', 'CMC ...
    Method 3')
matlab2tikz('figurehandle', f5, 'QMCvsCMC.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);
```

7.5 lattice.m

```
% Plots the N lattice points based upon the input vector z
%
% USAGE: lattice(34);

function X = lattice(N)
    [N, ii] = fib(N);  % this makes N the closest Fibonacci number
    ii = ii - 1;

    % this makes z(2) the previous Fibonacci number
    z = [ 1 floor(((1+sqrt(5))^ii-(1-sqrt(5))^ii)/(2^ii*sqrt(5)))];

    X = latticepts(z,N);
    %  fprintf('We used z = [%d %d] and N = %d.\n',z,N);
    %  plot(X(:,1),X(:,2),'.b');
end

% computes the N lattice points based upon the vector z
function X = latticepts(z,N)
    X = [1:N]'*z/N;
    X = X - floor(X);
end

% computes the nearest Fibonacci number
function [N, ii] = fib(N)

    ii = 0;
    c = 0;
    a = 0;
    b = 1;
    while (c ≤ N)
```

```
        ii = ii + 1;
        c = a + b;
        a = b;
        b = c;
    end

    if ((N-a) > (c-N))
        N = c;
        ii = ii + 1;
    else
        N = a;
    end
end
```

7.6 Sobol.m

```
%Solbol Squence
clear;clc;close all;

rng(2014); %define seed
format long
r = 0.5;
actual = (pi*r^2)/4;


sol = sobolset(2);  %dementions

k=5
X = net(sol,10^k)*r;
x = X(:,1)
y = X(:,2);
inside = (x.^2 + y.^2) ≤ r^2;

for i = 1:k
        if i == 2      %Plot example
            figure
            plot(x(1:100),y(1:100),'g.')
            title('Sobol Sequence with N = 100')
        end
    eS(i) = r^2 * sum(inside(1:10^i)) / 10^i;
    abserrS(i) = abs(eS(i)-actual);
end
    relerrS = abserrS / actual;

%Display Results
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('                  Monte Carlo Method via Sobol "Lattice/Sequence?" ...
                  \n')
fprintf('    N     |    ESTIMATE    |    ABSOLUTE ERROR    |    RELATIVE ERROR    \n');
%fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for i = 1:k
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n', 10^i, eS(i), ...
        abserrS(i), relerrS(i))
end
```

7.7 MClattice_sobol.m

```
clear;clc;close all;
%  colormap lines;
%  CMap = colormap;
CMap = [
    0.000 0.447 0.741
    0.850 0.325 0.098
    0.929 0.694 0.125
    0.494 0.184 0.556
    0.466 0.674 0.188
    0.301 0.745 0.933
    0.635 0.078 0.184
    ];

rng(2014); %define seed
format long
r = 0.5;
actual = (pi*r^2)/4;

%---------------------------Fibonacci Lattice--------------------------%

% Compute the first 26 Fibonacci numbers
fib(1) = 0; fib(2) = 1;
for i = 3:26,
    fib(i) = fib(i-1)+fib(i-2);
end

for i = 1:length(fib)-2
    L = lattice(fib(i+2))*r;
    x = L(:,1);
    y = L(:,2);
        if i == 10      %Plot example
            figure
            plot(x,y,'r.')
            title('Fibonacci Lattice with N = 89')
        end
    inside = (x.^2 + y.^2) ≤ r^2;
    eL(i) = r^2 * sum(inside) / fib(i+2);
    abserr(i) = abs(eL(i)-actual);
end
    relerr = abserr / actual;

%Display Results
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('                     Monte Carlo Method via Fibonacii Lattice                       \n')
fprintf('    N     |    ESTIMATE    |    ABSOLUTE ERROR    |    RELATIVE ERROR    \n');
%fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for i = 1:length(fib)-2
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n', fib(i+2), eL(i), ...
        abserr(i), relerr(i))
end

%------------------------------Product Rule Lattice--------------------%

%N=6;
Prod = [fib(3:12) 10^2 fib(13:21) 10^4 fib(22:end) 316^2];
for i= 1:length(fib)+1 %2:2:N
    [x y] = meshgrid(0:r/(sqrt(Prod(i))-1):r, 0:r/(sqrt(Prod(i))-1):r);
    [mx nx] = size(x);
    Ppts(i) = mx*nx;
    x = reshape(x,[mx*nx 1]);
    y = reshape(y,[mx*nx 1]);
    inside = (x.^2+y.^2) ≤ r^2;
        if i == 10
            figure
            plot(x, y,'.') %Plot example%
            str_title = sprintf('Product Rule with N=%d points',Ppts(i));
            title(str_title);
        end
     %estimate%
     eLgrid(i) = r^2*sum(inside)/Ppts(i);
     abserr_eLgrid(i) = abs(eLgrid(i)-actual);
  end
  relerr_eLgrid = abserr_eLgrid / actual;

  %Display Results
fprintf('\n');
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('                  Monte Carlo Method 1 via "Product Rule" Lattice ...
                          \n')
fprintf('    N     |    ESTIMATE    |    ABSOLUTE ERROR    |    RELATIVE ERROR    \n');
%fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for i = 1:length(fib)+1
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n', Ppts(i), eLgrid(i), ...
        abserr_eLgrid(i), relerr_eLgrid(i))
end

%-----------------------------Sobol Squence QMC Method 1----------------------%
%sol = sobolset(2);  %dementions

k=5
%X = net(sol,10^k)*r;
%x = X(:,1)
%y = X(:,2);
%inside = (x.^2 + y.^2) ≤ r^2;

%for i = 1:k
%        if i == 2      %Plot example
%            figure
%           plot(x(1:100),y(1:100),'g.')
%            title('Sobol Sequence with N = 100')
%        end
%    eS(i) = r^2 * sum(inside(1:10^i)) / 10^i;
%   abserrS(i) = abs(eS(i)-actual);
%end

%If statistical toolbox not available, here are the need vectors
eS =  [0.225000000000000   0.202500000000000   0.196500000000000   0.196375000000000  ...
    0.196347500000000];
abserrS = [0.028650459150638   0.006150459150638   0.000150459150638   0.000025459150638 ...
    0.000002040849362]
relerrS = abserrS / actual;


%Display Results
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('                 Quasi Monte Carlo Method via Sobol Sequence                       \n')
fprintf('    N     |    ESTIMATE    |    ABSOLUTE ERROR    |    RELATIVE ERROR    \n');
%fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for i = 1:k
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n', 10^i, eS(i), ...
        abserrS(i), relerrS(i))
end


%Plot Convergence Rates for Fib Latt,Prod Latt, CMC Method 1, and QMC Method 1
f6 = figure;
abserr_e1 = [0.028650459150638   0.001349540849362   0.003849540849362   0.002300459150638   ...
    0.000047959150638];
abserr_e1q = [0.000676173729491   0.001406437708150   0.000323521389740   0.000054725639127   ...
    0.000006721893680];
num = [1 10000];
num = 10.^[1:5];
semilogy(fib(8:end),abserr(6:end),'Color', CMap(1,:), 'LineWidth', .6); hold on;
semilogy(Ppts ,abserr_eLgrid,'Color', CMap(2,:), 'LineWidth', .6);
semilogy(num, abserr_e1,'Color', CMap(3,:), 'LineWidth', .6);
semilogy(num, abserr_e1q,'Color', CMap(4,:), 'LineWidth', .6);
semilogy(num, abserrS,'Color', CMap(5,:), 'LineWidth', .6);
xlabel('number of points')
ylabel('absolute error')
title('Method(s) 1 Convergence Rates')
axis square
%xlim([1,10000])
legend('Fibonacci Lattice', 'Product Rule Lattice', 'CMC', 'QMC-Halton', 'QMC-Sobol')
matlab2tikz('figurehandle', f6, 'LatticevsMC2.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);

f7 = figure;
x = linspace(0,10^5,1000);
%Fibonacci
F = fib(8:end)'; E = abserr(6:end)';
C = [ ones(length(F),1) log(F)]\log(E);
B = C(1)+C(2)*log(x);
semilogy(F,E,'+','Color',CMap(1,:));hold on;
p1 = semilogy(x,exp(B),'-','Color',CMap(1,:));

% Product Rule
F = Ppts(8:end)'; E = abserr_eLgrid(8:end)';
C = [ ones(length(F),1) log(F)]\log(E);
B = C(1)+C(2)*log(x);
semilogy(F,E,'+','Color',CMap(2,:));
p2 = semilogy(x,exp(B),'-','Color',CMap(2,:));

% CMC
F = 10.^[1:5]'; E = abserr_e1';
C = [ ones(length(F),1) log(F)]\log(E);
B = C(1)+C(2)*log(x);
semilogy(F,E,'+','Color',CMap(3,:));
p3 = semilogy(x,exp(B),'-','Color',CMap(3,:));

% QMC
F = 10.^[1:5]'; E = abserr_e1q';
C = [ ones(length(F),1) log(F)]\log(E);
B = C(1)+C(2)*log(x);
semilogy(F,E,'+','Color',CMap(4,:));
p4 = semilogy(x,exp(B),'-','Color',CMap(4,:));

%Sobol
F = 10.^[1:5]'; E = abserrS';
C = [ ones(length(F),1) log(F)]\log(E);
B = C(1)+C(2)*log(x);
semilogy(F,E,'+','Color',CMap(5,:));
p5 = semilogy(x,exp(B),'-','Color',CMap(5,:));

xlabel('number of points')
ylabel('absolute error')
title('Method(s) 1 Convergence Rates Linear Fit')
legend([p1 p2 p3 p4 p5],'Fibonacci Lattice', 'Product Rule Lattice', 'CMC', 'QMC-Halton', ...
    'QMC-Sobol');
xlim([10 100000]);

matlab2tikz('figurehandle', f7, 'LatticevsMC2_LinearFit.tex', 'showInfo', false, ...
    'checkForUpdates', false,'height', '\figureheight', 'width', '\figurewidth', ...
    'standalone', true);
```

7.8 Method2n3_Lattice.m

```
%Method 2 and 3 Lattice Integration

clear;clc;close all;
rng(2014); %define seed
format long
r = 0.5;
```

```
actual = (pi*r^2)/4;
f = @(x) sqrt(r.^2 - x.^2);
k = 5;

%Method 2
for i = 1:k;
    xold = [0: r/10^i : .5];
    x = xold(1:10^i);
    e2L(i) = r*sum(f(x))/(10^i);

%Determine Error
    abserr_e2L(i) = abs(e2L(i)-actual);
end

relerr_e2L = abserr_e2L / actual;

%Display Results
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('    N     |    ESTIMATE    |    ABSOLUTE ERROR    |    RELATIVE ERROR    \n');
%fprintf('\n');
fprintf('                       Lattice Monte Carlo Method 2                           \n');
%fprintf('\n');
fprintf('----------|----------------|----------------------|----------------------\n');
for k = 1:5
    fprintf('%8.0f  |    %1.6f    |     %1.6e     |     %1.6e      \n',10^k, e2L(k), ...
        abserr_e2L(k), relerr_e2L(k))

end

abserr_e2 = [0.006682937312325   0.003727526218243   0.002002912520977   0.000642413785594   ...
    0.000161776561111];
abserr_e2q = [0.015010416574067   0.002490746973378   0.000300146098825   0.000040985324138   ...
    0.000005139678088];
num = 10.^[1:5];

%Plot Results
f1 = figure;
semilogy(num,abserr_e2L, num, abserr_e2, num, abserr_e2q)
xlabel('number of points')
ylabel('absolute error')
title('CMC, QMC and Lattice Method 2 Convergence')
axis square
legend('Lattice Method 2', 'CMC Method 2', 'QMC Method 2')
matlab2tikz('figurehandle', f1, 'Method2_Lattice.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);


%--------------------------- Lattice - Method 3 ----------------------------------%

xmidd = 0.025:0.05:0.475;
xinterval = 0:0.05:r;
f = @(x) sqrt(r.^2 - x.^2);
ymidd = f(xmidd);
ysum = sum(ymidd);

%Preallocate
e3L = zeros(1,5);
p = zeros(1,10);

for n = 1:5
    num = 10^n;
    for i = 1:10
        p(i) = f(xmidd(i))/ysum;
        pts_int(i) = p(i)*num;
    end

    %pts_int needs to be all integer values
    rnd_pts = round(pts_int);

%-----------If simple round function does not sum to n...---------------%

    if sum (rnd_pts) < num %need to add a point in appropriate panel
        for k = 1:10;
            abs_diff(k) = abs(rnd_pts(k) - pts_int(k));
        end
    [val index] = max(abs_diff);
    rnd_pts(index) = rnd_pts(index) + 1; %point added
    end

    if sum (rnd_pts) > num %need to subract a point in appropriate panel
        for k = 1:10;
            abs_diff(k) = abs(rnd_pts(k) - pts_int(k));
        end
    [val index] = max(abs_diff);
    rnd_pts(index) = rnd_pts(index) - 1; %point subtracted
    end

    s = sum(rnd_pts); %check

    % since we know how long the sequence will need to be, create it and store it
    h = linspace(0,0.05,max(rnd_pts));
 %----------------------------------------------------------------------%

        %Create X Values according to "Lattice" with z = {1}
    srt = 1;
    for j = 1:10 %# of panels
        last = srt + rnd_pts(j) - 1;
        %  xm3(srt:last) = halton(2,rnd_pts(j))'*0.05 + xinterval(j);
        xm3(srt:last) = h(1:rnd_pts(j))' + xinterval(j);

        %Evaluate Function Values
        ym3(srt:last) = f(xm3(srt:last));
        area(j) = (0.05)*((sum(ym3(srt:last)))/rnd_pts(j));
        srt = last + 1;
    end
    e3L(n) = sum(area);

    %Determine Error
    abserr_e3L(n) = abs(e3L(n)-actual);

    %Plot Procedure
    if n == 2;
        f2 = figure;
        offset = 0;
        plot(xm3,ym3,'.'); hold on;
        for ii = 1:length(rnd_pts)
            plot(xm3(1+offset),ym3(1+offset),'r.');hold on;
            title('Method 3 via 1-Dimensional "Lattice" with n=100')
            offset = offset+rnd_pts(ii);
        end
        axis([0 0.5 0 0.5]);
        axis square;
        matlab2tikz('figurehandle', f2, 'MC3_Lattice.tex', 'showInfo', false, ...
            'checkForUpdates', false,'height', '\figureheight', 'width', '\figurewidth', ...
            'standalone', true);
    end;
end

relerr_e3L = abserr_e3L / actual;

%Display Results
fprintf('Actual Value of Integral: %1.6f\n', actual);
fprintf('\n');
fprintf('                    Lattice Monte Carlo Method 3            \n')
fprintf('\n');
fprintf('    N      |     ESTIMATE     |     ABSOLUTE ERROR     |     RELATIVE ERROR     \n');
fprintf('----------|-----------------|----------------------|----------------------\n');
for k = 1:5
    fprintf('%8.0f  |     %1.6f     |      %1.6e      |      %1.6e      \n',10^k, e3L(k), ...
        abserr_e3L(k), relerr_e3L(k))
end


abserr_e3 = [0.004316000562404   0.000270011858544   0.000225372123501   0.000076972525891   ...
    0.000002222747864];
abserr_e3q = [0.000676173729491   0.001406437708150   0.000323521389740   0.000054725639127   ...
    0.000006721893680];
num = 10.^[1:5];

%Plot Results
f3 = figure;
semilogy(num,abserr_e3L, num, abserr_e3, num, abserr_e3q)
xlabel('number of points')
ylabel('absolute error')
title('CMC, QMC and Lattice Method 3 Convergence')
axis square
legend('Lattice Method 3', 'CMC Method 3', 'QMC Method 3')
matlab2tikz('figurehandle', f3, 'Method3_Lattice.tex', 'showInfo', false, 'checkForUpdates', ...
    false,'height', '\figureheight', 'width', '\figurewidth', 'standalone', true);
```

References

[1] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[2] R. Cools, F. Y. Kuo, and D. Nuyens. Constructing embedded lattice rules for multivariate integration. *SIAM J. Sci. Comp*, 28:2272256, 2006.

[3] I. Dalal, D. Stefan, and J. Harwayne-Gidansky. Low discrepancy sequences for monte carlo simulations on reconfigurable platforms. In *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on*, pages 108–113, July 2008.

[4] F. Kuo, C. Schwab, and I. H. Sloan. Quasi-Monte Carlo finite element methods for a class of elliptic partial differential equations with random coefficients. Technical Report 2011-52, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2011.

[5] F. Y. Kuo and I. H. Sloan. Lifting the curse of dimensionality. *Notices of the AMS*, 52:1320–1329, 2005.

[6] D. Nuyens. Fast construction of good lattice rules. 2007.

[7] D. P. O'Leary. Multidimensional integration: partition and conquer. *Computing in Science & Engineering*, 6(6):58–66, 2004.

[8] S. H. Paskov and J. F. Traub. Faster valuation of financial derivatives. Working papers, Santa Fe Institute, 1995.

[9] T. Sauer. *Numerical Analysis.* Addison-Wesley Longman, Incorporated, 2006.

[10] V. Sinescu. *Construction of lattice rules for multiple integration based on a weighted discrepancy.* PhD thesis, The University of Waikato, 2008.

[11] V. Sinescu. Shifted lattice rules based on a general weighted discrepancy for integrals over euclidean space. *Journal of computational and applied mathematics*, 232(2):240–251, 2009.

*Note that Section 2.1, Section 2.2 and Section 2.3 were heavily guided through a project idea proposed in [7].