Uniformly distributed sequences and their discrepancies

Laura A. Pace
Carlos Salazar-Lazaro

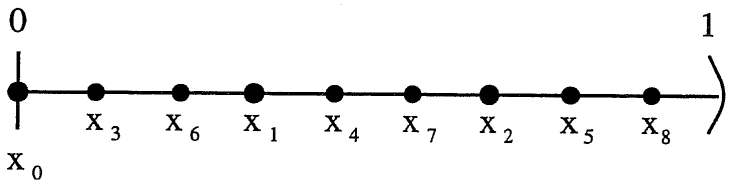
Faculty Advisor
Professor Robert Burton

REU Program
Oregon State University
August 1996

Chapter 1

Introduction

1.1 What is a uniformly distributed sequence?



The p-adic sequence, where p=3

The above sequence of points on the $[0,1)$ interval is an example of a *uniformly distributed sequence*. There are no "huge" gaps between the points or any large groups of points bunched together. In other words, the points are evenly spaced throughout the interval. A formal definition for a uniformly distributed sequence follows and can be found in [1].

Definition 1.1 *Let* $\{x_1, x_2, \ldots\}$ *be a sequence of points contained in the interval* $(0,1]$. *Then the sequence is uniformly distributed if for any interval* (a,b), $0 \leq a < b < 1$

$$\frac{\#\{x_i \in (a,b) \mid 1 \leq i \leq N\}}{N} \longrightarrow_{N \to \infty} (b-a).$$

Imagine an interval, I, of length $M < 1$. Then no matter where you put I on the interval $(0,1]$, the number of points contained in I will be nearly

constant. This guarantees that the sequence is uniformly distributed.

Throughout the rest of the paper, the notation $\#(a,b)$ will be defined as $\#\{x_i \in (a,b) \mid 1 \leq i \leq N\}$ and $\#[a,b]$ will be defined as $\#\{x_i \in [a,b] | \ 1 \leq i \leq N\}$ where N is the total number of points in the sequence.

1.2 What is Discrepancy?

Some sequences are more uniform than others, naturally leading to the question of how to measure the uniformity of a sequence. *Discrepancy* is the measure of how "non-uniform" a sequence is. When dealing with *Quasi-Monte Carlo Integration*, the lower the discrepancy of the sequence, the lower error of the calculations, which will be discussed further in Chapter 4. Because the discrepancy of a sequence plays a significant part in error calculations, a quantitative measure for discrepancy is needed.

Definition 1.2 *Let $\{x_1, x_2, \ldots, x_N\}$ be a sequence in $[0,1)$. Then the discrepancy of the sequence is defined as*

$$D_N = \sup_{0 \leq \alpha < \beta < 1} \left| \frac{\#(\alpha,\beta)}{N} - (\beta - \alpha) \right|$$

where $\#(\alpha,\beta)$ is the number of points in (α,β) [1].

By restricting the intervals over which the supremum is calculated, an alternative definition of discrepancy is created, which can be found in [1].

Definition 1.3 *Let $\{x_1, x_2, \ldots, x_N\}$ be a sequence in $[0,1)$. Then the discrepancy D_N^* is defined as*

$$D_N^* = \sup_{0 < \alpha \leq 1} \left| \frac{\#[0,\alpha)}{N} - \alpha \right|.$$

There are some general upper and lower bounds for the discrepancy of all sequences. Tighter bounds can be found for specific sequences. The following bounds for discrepancy and associate proof can be found in [1].

Theorem 1.1 *Let $\{x_1, x_2, \ldots, x_N\}$ be a sequence of numbers. Then*

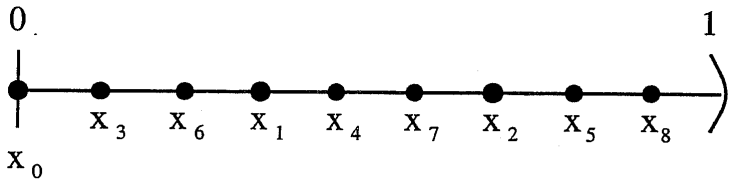
$$\frac{1}{N} \leq D_N \leq 1.$$

1.3 Examples of uniformly distributed sequences

There are several kinds of uniformly distributed sequences. Each of these sequences has its own discrepancy bound, depending on its behaviour as N gets large. *Low discrepancy sequences* are sequences for which D_N becomes small as N gets large. These sequences have bounded discrepancy, often of the order $c * \frac{\ln N}{N}$, where c is a constant. Two such sequences are the *p-adics* and the *rotations*. There are several more low discrepancy sequences; however these were the two that we worked with. Our plan was to use these sequences to develop a general method to form sequences and use this method to create a dynamical system. This will be discussed further in Chapter 3.

For both sequences, and the dynamical systems, we need a set of real numbers, called scales, such that $|\gamma_1| > |\gamma_2| > \ldots > |\gamma_n| > \ldots > 0$.

1.3.1 p-adic sequences



The p-adic sequence, where p=3

The above sequence is an example of the *p-adic* sequence. There are two algorithms for creating the sequence. One method involves using base p notation. The other method uses scales. We will mostly use the algorithm involving scales because it is easier to generalize to the dynamical system; however, both are equivalent. We will be creating the sequence on the $[0, 1)$ interval.

Let p be a prime number. The n^{th} scale, γ_n, is equal to $\frac{1}{p^n}$. The pointer, q, points to the x_i that you're on and end list, N, points to the last x_i added to the sequence. For each point x_i in the sequence let b_i be defined as the length between x_i and the point immediatly following it. In other words,

$b_i = \min_{k=1,2,..,N}\{(1-x_i),(x_k-x_i)|x_k-x_i>0\}$. Initially set $x_0 = 0$, $q = 0$ and $N = 0$. Then follow the following algorithm for $n = 1, 2, 3, ...$ to create the sequence.

If $b_q > \gamma_n$ then

- add $x_{N+1} = x_q + \gamma_n$ to the sequence
- add $b_{N+1} = b_q - \gamma_n$
- set $b_q = \gamma_n$
- set $q = q + 1$
- set $N = N + 1$

If $b_q \leq \gamma_n$ then

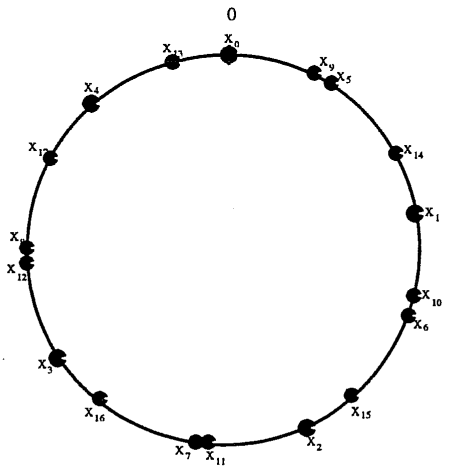
- If pointer is not the end of the list then set $q = q + 1$
- If pointer is end of the list then
  - set pointer to 0
  - replace γ_n by γ_{n+1}

As you can see from the *p-adic* sequence, the γ_n's are used to partition the intervals at each stage. A γ_n *stage* is when you go through and add points to the sequence for that γ_n . Unless stated otherwise, γ_n will be positive. $\gamma_n > 0$ will indicate foward direction, and $\gamma_n < 0$ will denote negative direction.

The second method of generating the *p-adic* sequence uses base p notation. First n is written in base p. In other words, $a_i \in \{0, 1, 2, \ldots, p_1\}$ and $n = \Sigma_{i=0}^{j} a_i p^i$. Then, x_n is defined as the reflection of n around the decimal point. Basically, $x_n = \Sigma_{i=0}^{j} a_i p^{-i-1}$ which means that $x_n \in [0, 1)$. It can be shown that this method and the algorithm for creating the *p-adic* sequence are equivalent [3].

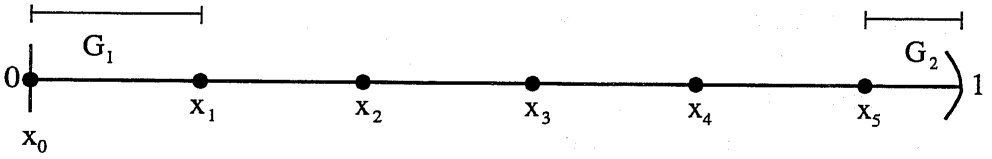
The *p-adic* sequence is a low-discrepancy sequence. It has been shown that the discrepancy of the *p-adic* sequence is bounded by $c * \frac{\ln N}{N}$, where c is a constant.

1.3.2 Rotation sequences

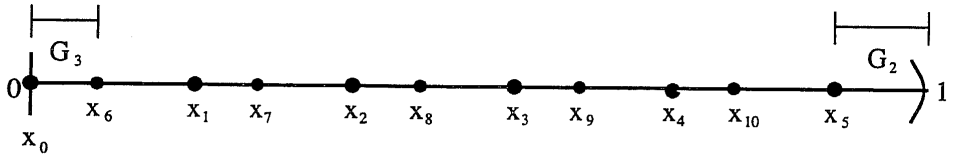


The above sequence of points is the *rotation* sequence. This sequence is formed similiarly to the *p-adic* sequence except the $\gamma_n's$ are defined differently.

In the *rotation* sequence, each γ_n is defined recursively. Pick an irrational number, α. Then $\gamma_0 = 0$, $\gamma_1 = \alpha \pmod 1$, $x_0 = 0$ and $x_1 = \gamma_1$. Continue adding points to the sequence by "going foward" a distance γ_1. Since γ_1 is irrational, the points will not evenly divide the interval $[0, 1)$, so there will eventually be a point x_i such that $1 - x_i < \gamma_1$. The remaining distance will be γ_2.



Then fill up each subinterval of $[0, 1)$ with points by going "backward" from each point a distance of γ_2 until there is no more room. The remaining distance will be γ_3



By continuing in this manner, we get the recursive definition for γ_n,

$$\gamma_{n+1} = (-1)^{n+1} * \left(\gamma_{n-1} - \gamma_n * \left\lfloor \frac{\gamma_{n-1}}{\gamma_n} \right\rfloor\right)$$

where $\gamma_1 > 0, \gamma_2 < 0, \gamma_3 > 0, \ldots, \gamma_{2n} < 0, \gamma_{2n+1} > 0, \ldots$. Thus the γ_n's are alternating in sign. We informally call this an *alternating* sequence.

For the algorithm to generate the *rotation* sequence, b_i, the pointer q and end list N are defined the same as in the *p-adic* sequence. Let c_i be defined as the length between x_i and the point immediately preceding it. Then $c_i = \min_{k=1,2,..,N}\{(1 - x_i), (x_i - x_j) | x_i - x_k > 0\}$. Let r_i and l_i be the indices for the points immediately right and left of x_i. In other words, $r_i = k$ such that $x_k - x_i = b_i$ and $l_i = k$ such that $x_i - x_k = c_i$. Initialize $x_0 = 0, N = 0, q = 0$, and $c_0 = 0$. Then the following algorithm generates the *rotation* sequence.

For the stages where $\gamma_n > 0$

- If $b_q > \gamma_n$ then
  - add $x_{N+1} = x_q + \gamma_n$ to the sequence
  - add $b_{N+1} = b_q - \gamma_n$ and $c_{N+1} = \gamma_n$
  - set $b_q = \gamma_n$ and $c_{r_q} = b_{N+1}$
  - set $q = q + 1$ and $N = N + 1$
- If $b_q < \gamma_n$ then
  - If pointer is not the end of the list then set $q = q + 1$
  - If pointer is end of the list then
    - set pointer to 0
    - replace γ_n by γ_{n+1}

For the stages where $\gamma_n < 0$

- If $c_q > \gamma_n$ then

- add $x_{N+1} = x_q - \gamma_n$ to the sequence
- add $b_{N+1} = \gamma_n$ and $c_{N+1} = c_q - \gamma_n$
- set $c_q = \gamma_n$ and $b_{l_q} = c_{N+1}$
- set $q = q + 1$ and $N = N + 1$

• If $c_q < \gamma_n$ then

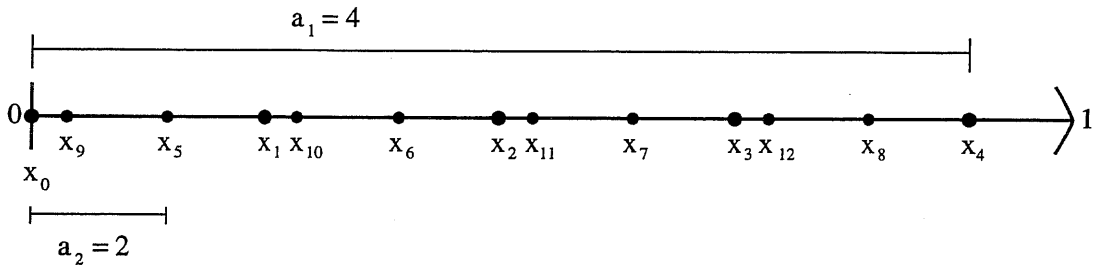
- If pointer is not the end of the list then set $q = q + 1$
- If pointer is end of the list then
  * set pointer to 0
  * replace γ_n by γ_{n+1}

Another way to describe the algorithm is as follows. Wrap the unit interval $[0, 1)$ into a circle, S'. Let $x_0 = 0$ and $x = \alpha \pmod 1$. Then "rotate" from 0 to x and let $x_1 = x$. Continue rotating foward distance x. From this rotation we can see that $x_n = n\alpha \pmod 1$.

Any irrational number α can be written in continued fraction form as below.

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ddots}}}$$

$a_0 = \lfloor \alpha \rfloor$. When going through the γ_1 stage you can get a certain number of points in the $[0, 1)$ interval before running out of room. The number of points added to the sequence at that stage is a_1. In the second stage, the number of points added between 0 and x_1 is a_2. From this we get our continued fraction.



The *continued fraction* sequence is also a low discrepancy sequence. In [4], Niederreiter proves that if there is a postitive integer K such that for $i = 1, 2, 3, \ldots$, $a_i < K$ then the discrepancy of the sequence is bounded by $O(\frac{\ln N}{N})$.

Chapter 2

Theorem on Finite Calculations

Recall that discrepancy was given in Definition 1.2 as

$$D_N = \sup_{0 \le \alpha < \beta < 1} \left| \frac{\#(\alpha, \beta)}{N} - (\beta - \alpha) \right|$$

for a given sequence $\{x_1, x_2, \ldots, x_N\}$. This definition involves a *supremum* over an infinite number of intervals. Unfortunately, this makes estimating discrepancy on the computer impossible. For this reason we developed a theorem which computes D_N over a finite number of intervals, rather than an infinite number. This makes computing D_N possible.

2.1 Formulas for finite calculations of discrepancy

According to Definition 1.2 there is at least one interval that causes the discrepancy of the sequence to be at its worse. Let $(a, b) \subseteq [0, 1)$ such that

$$D_N = \left| \frac{\#(a, b)}{N} - (b - a) \right|$$

and $0 \le \ldots < x_{i-1} \le a < x_i \le \ldots \le x_j < b \le x_{j+1} < \ldots < 1$. This then leads to the question of what happens to the discrepancy on the intervals

(x_{i-1}, x_{j+1}) and $[x_i, x_j]$. The number of points in each of these intervals would be the same as the number of points in (a, b) but the length of the intervals is either increasing or decreasing, which would imply that the D_N of these intervals would also be increasing or decreasing. Then the intervals (x_{i-1}, x_{j+1}) and $[x_i, x_j]$ would be the extreme D_N for the points in the interval (a, b). This leads to the following Theorem.

Theorem 2.1 *Let $\{x_1, x_2, \ldots, x_N\}$ be a sequence in $[0, 1)$. Then the discrepancy*

$$D_N = \sup_{0 \le a < b < 1} \left| \frac{\#(a, b)}{N} - (b - a) \right|$$

is equivalent to

$$D_N = \max_{i \le j} \left\{ \frac{\#[x_i, x_j]}{N} - (x_j - x_i) \right\} \bigvee \max_{i < j} \left\{ (x_j - x_i) - \frac{\#(x_i, x_j)}{N} \right\}$$

proof:
Let $\{x_1, x_2, \ldots, x_N\}$ be a sequence in $[0, 1)$ such that the discrepancy D_N of the sequence is M. Then $\exists (a, b) \subseteq [0, 1)$ such that

$$D_N = \left| \frac{\#(a, b)}{N} - (b - a) \right| = M$$

From Theorem 1.1 we know that $D_N \ge \frac{1}{N}$.
Then $\frac{\#(a,b)}{N} \ne (b - a)$ which implies $\frac{\#(a,b)}{N} > (b - a)$ or $\frac{\#(a,b)}{N} < (b - a)$
Let $0 \le \ldots < x_{i-1} \le a < x_i \le \ldots \le x_j < b \le x_{j+1} < \ldots < 1$.

Case 1: $\frac{\#(a,b)}{N} > (b - a)$
Then $\frac{\#(a,b)}{N} - (b - a) > 0$
Because $0 \le \ldots < x_{i-1} \le a < x_i \le \ldots \le x_j < b \le x_{j+1} < \ldots < 1$

$$\#(a, b) = \#[x_i, x_j] \text{ and } (x_j - x_i) \le (b - a).$$

Then

$$0 < \frac{\#(a, b)}{N} - (b - a) \le \frac{\#[x_i, x_j]}{N} - (x_j - x_i)$$

But

$$\sup_{0 \le \alpha < \beta < 1} \left| \frac{\#(\alpha, \beta)}{N} - (\beta - \alpha) \right| = M$$

which means that

$$0 < \frac{\#(a,b)}{N} - (b-a) \not< \frac{\#[x_i,x_j]}{N} - (x_j - x_i)$$

therefore

$$\frac{\#[x_i,x_j]}{N} - (x_j - x_i) = M$$

Case 2: $\frac{\#(a,b)}{N} < (b-a)$

Then $(b-a) - \frac{\#(a,b)}{N} > 0$

Because $0 \leq \ldots < x_{i-1} \leq a < x_i \leq \ldots \leq x_j < b \leq x_{j+1} < \ldots < 1$

$$\#(a,b) = \#(x_{i-1}, x_{j+1}) \text{ and } (x_{j+1} - x_{i-1}) \geq (b-a).$$

Then

$$0 < (b-a) - \frac{\#(a,b)}{N} \leq (x_{j+1} - xi - 1) - \frac{\#(x_{i-1}, x_{j+1})}{N}$$

But

$$\sup_{0 \leq \alpha < \beta < 1} \left| \frac{\#(\alpha,\beta)}{N} - (\beta - \alpha) \right| = M$$

which means that

$$0 < (b-a) - \frac{\#(a,b)}{N} \not< (x_{j+1} - x_{i-1}) - \frac{\#(x_{i-1}, x_{j+1})}{N}$$

therefore

$$(x_{j+1} - x_{i-1}) - \frac{\#(x_{i-1}, x_{j+1})}{N} = M$$

Then

$$D_N = \max_{i \leq j} \left\{ \frac{\#[x_i,x_j]}{N} - (x_j - x_i) \right\} \bigvee \max_{i<j} \left\{ (x_j - x_i) - \frac{\#(x_i,x_j)}{N} \right\} \square$$

The sequence of points $\{x_1, x_2, \ldots, x_N\}$ can be ordered without any loss of generality. Thus if we let the sequence be such that $0 = x_0 < x_1 < x_2 < \ldots < x_N < x_{N+1} = 1$, it is easy to see that we will get the the same discrepancy. Thus we may assume that the sequence of numbers are ordered. With this we get a more general form of Theorem 2.1.

Corollary 2.1 *Let* $\{x_1, x_2, \ldots, x_N\}$ *be a sequence in* $[0,1)$ *such that* $0 = x_0 < x_1 < x_2 < \ldots < x_N < x_{N+1} = 1$. *Then the discrepancy* D_N *is equivalent to*

$$D_N = \max_{i \leq j} \left| (x_i - \frac{i}{N}) - (x_j - \frac{j}{N}) \right| + \frac{1}{N} \tag{2.1}$$

proof:
From Theorem 2.1 we have that

$$D_N = \max_{i \leq j} \left\{ \frac{\#[x_i, x_j]}{N} - (x_j - x_i) \right\} \bigvee \max_{i < j} \left\{ (x_j - x_i) - \frac{\#(x_i, x_j)}{N} \right\}.$$

Since the elements of the sequence are ordered we have that $\#[x_i, x_j] = j - i + 1$ and $\#(x_i, x_j) = j - i - 1$. Then,

$$D_N = \max_{i \leq j} \left\{ \frac{\#[x_i, x_j]}{N} - (x_j - x_i) \right\} \bigvee \max_{i < j} \left\{ (x_j - x_i) - \frac{\#(x_i, x_j)}{N} \right\}$$

$$= \max_{i \leq j} \left\{ \frac{j - i + 1}{N} - (x_j - x_i) \right\} \bigvee \max_{i \leq j} \left\{ (x_j - x_i) - \frac{j - i - 1}{N} \right\}.$$

Then,

$$D_N = \max_{i \leq j} \left\{ (x_i - \frac{i}{N}) - (x_j - \frac{j}{N}) + \frac{1}{N} \right\} \bigvee \max_{i \leq j} \left\{ (x_j - \frac{j}{N}) - (x_i - \frac{i}{N}) + \frac{1}{N} \right\}$$

Therefore,

$$D_N = \max_{i \leq j} \left| (x_i - \frac{i}{N}) - (x_j - \frac{j}{N}) \right| + \frac{1}{N} \square.$$

The discrepancy D_N^* also has a general form involving only finite calculations. The following Corollary shows this and the proof can be found in [1].

Corollary 2.2 *Let* $\{x_1, x_2, \ldots, x_N\}$ *be a sequence in* $[0,1)$ *such that* $0 = x_0 < x_1 < x_2 < \ldots < x_N < x_{N+1} = 1$. *Then the discrepancy* D_N^* *is given by*

$$D_N^* = \max_{i=1,\ldots,N} \max \left\{ \left| x_i - \frac{i}{N} \right|, \left| x_i - \frac{i-1}{N} \right| \right\}.$$

For our computer programs calculating discrepancy we used another version of Theorem 2.1 given in the corollary below. This formula allows for lower computer computation time. It can formed directly from Corollary 2.1. Niederreiter also presented this as a theorem in [4] and his proof can be found there.

Corollary 2.3 *Let* $0 = x_0 < x_1 < x_2 < \ldots < x_N < x_{N+1} = 1$ *be a sequence in* $[0,1)$. *Then the discrepancy,* D_N, *is equivalent to*

$$D_N = \frac{1}{N} + \max_{1 \le i \le N} \left(\frac{i}{N} - x_i \right) - \min_{1 \le i \le N} \left(\frac{i}{N} - x_i \right). \tag{2.2}$$

proof:
From Corollary 2.1 we know that

$$D_N = \max_{i,j} \left| (x_i - \frac{i}{N}) - (x_j - \frac{j}{N}) \right| + \frac{1}{N}$$

To get the maximum D_N we want the first value, $x_i - \frac{i}{N}$, to be as large as possible and the second value, $x_j - \frac{j}{N}$, to be as small as possible. Then we get

$$D_N = \frac{1}{N} + \left| \max_{1 \le i \le N} \left(x_i - \frac{i}{N} \right) - \min_{1 \le i \le N} \left(x_i - \frac{i}{N} \right) \right|.$$

Then,

$$D_N = \frac{1}{N} + \max_{1 \le i \le N} \left(x_i - \frac{i}{N} \right) - \min_{1 \le i \le N} \left(x_i - \frac{i}{N} \right) \square$$

2.2 Calculating D_N in higher dimensions

So far, we have talked about discrepancy in the first dimension over intervals. Discrepancy can be applied to several dimensions as well. Before we define discrepancy for multiple dimensions we need to introduce some notation.

$$R((a,b),(c,d)) = \left\{ (x,y) \;\middle|\; \begin{array}{l} a < x < c \\ b < y < d \end{array} \right.$$

is the interval over the open cube in $\mathbf{R}^2$.

$$R[(a,b),(c,d)] = \left\{ (x,y) \;\middle|\; \begin{array}{l} a \le x \le c \\ b \le y \le d \end{array} \right.$$

is the interval over the closed cube in $\mathbf{R}^2$.

The definition of discrepancy can naturally be extended to the multi-dimensional case [1].

Definition 2.1 *Let* $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N\}$ *be a sequence in* $\mathbf{R}^2$. *Let* $J = (\alpha_1, \beta_1)X(\alpha_2, \beta_2) \in [0,1)X[0,1)$ *and* $J = (0, \beta_1)X(0, \beta_2) \in [0,1)X[0,1)$. *Then* D_N *and* D_N^* *are defined as follows*

$$D_N = \sup_J \left| \frac{\#R(J)}{N} - |J| \right|$$

$$D_N^* = \sup_{J^*} \left| \frac{\#R(J^*)}{N} - |J^*| \right|$$

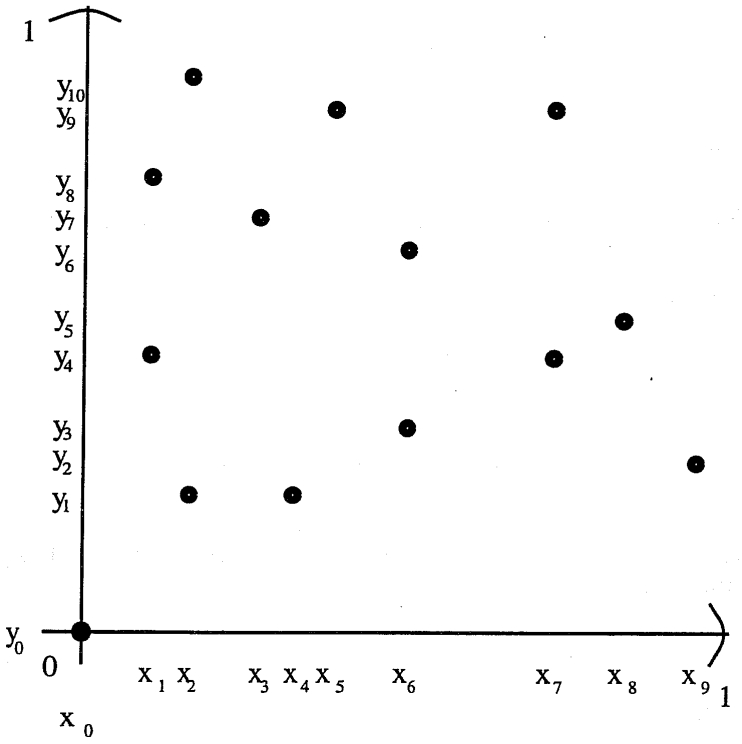
Similiarly, D_N and D_N^* can be reduced to open and closed regions of points as in Theorem 2.1. The proof is similiar to the one in Theorem 2.1.

Theorem 2.2 *Let* $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N\}$ *be a sequence in* $\mathbf{R}^2$. *Then* D_N *and* D_N^* *are equivalent to*

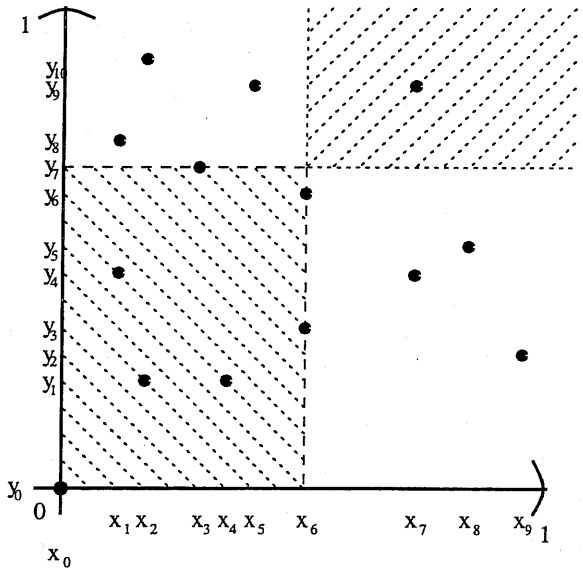
$$D_N = \max_{i,j} \left\{ \frac{\#R[\mathbf{x}_i, \mathbf{x}_j]}{N} - |R[\mathbf{x}_i, \mathbf{x}_j]| \right\} \bigvee \min_{i,j} \left\{ |R(\mathbf{x}_i, \mathbf{x}_j)| - \frac{\#R(\mathbf{x}_i, \mathbf{x}_j)}{N} \right\}$$

$$D_N^* = \max_{i} \left\{ \frac{\#R[\mathbf{0}, \mathbf{x}_i]}{N} - |R[\mathbf{0}, \mathbf{x}_i]| \right\} \bigvee \min_{i} \left\{ |R(\mathbf{0}, \mathbf{x}_i)| - \frac{\#R(\mathbf{0}, \mathbf{x}_i)}{N} \right\}$$

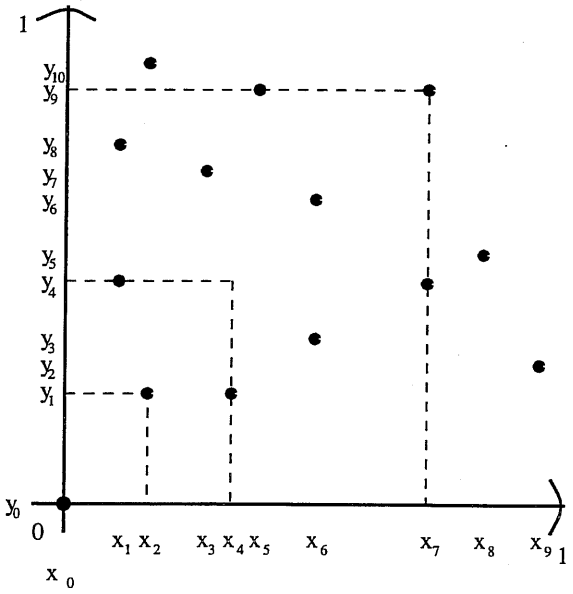
In $\mathbf{R}$, by ordering the points in the sequence it is easy to count the number of points in the intervals. Unfortunately, for $\mathbf{R}^2$, this is not so easy.



The graph below shows a sequence of points in $\mathbf{R}^2$. The x and y points are ordered, thus the points in the sequence are of the form (x_i, y_j). Our plan was to find a way to calculate the number of points in the shaded regions. However, we ran into several problems. When trying to calculate the number of points in the bottom left region, we found that there's no way to separate off the rest of the points without doubling the number of points in the top right region.



Another problem found was if you have two points, such as (x_i, y_j), (x_i, y_k) or (x_i, y_j), (x_k, y_j), then the points would only get counted once in the x or y indexing, but there are actually two points. This causes difficulty in finding an exact formula for counting the points.



In Theorem 2.2 we need a method for counting points in both open and closed intervals. In open intervals, the boundry does not present a problem since all points on the boundry are not counted. However, in the closed interval the points on the boundry are included. As seen from the above example, there is a possibility of 1, 2, or 3 points being on the boundry and as of yet we have not found a way to count the number of points.

2.3 Using Matlab and C to approximate D_N

Theorem 2.1 and Corollary 2.1 are useful when calculating the discrepancy of sequences, because they only require a finite number of calculations. In fact, using matrices on MATLAB^{TM} allows for simple computations of D_N. We wrote Matlab programs to calculate discrepancies of different sets of points. The first programs used equation 2.1. Later on, equation 2.2 was found for the one dimensional discrepancy in Neideireiter [4], which takes less amount of computations than the previous formula. Most of the programs written, used equation 2.2. Matlab was used to produce plots of discrepancies, C programs to calculate discrepancies and reproduce sequences. The C programs were based on linked lists, which were used to reproduce sets of points using a given algorithm. The complete program can be found in Appendix A.

Chapter 3

Dynamical Systems and Discrepancy

The main goal of our project was to generalize the *p-adic* and *continued fraction* sequences into a dynamical system. Hopefully, we could then find some conditions on the γ_n's that would guarantee low-discrepancy sequences.

3.1 Introduction to dynamical systems

3.1.1 What is a dynamical system?

We will be working with dynamical systems on S' or $[0,1)$. S' is the interval $[0,1)$ wrapped around into a circle with the origin at the top. $\mathbf{B}$ is the class of *Borel* sets. Borel sets are the natural class of sets for which length or Lebesque measure may be defined. *Lebesque measure*, λ, on the unit interval is the usual definition of length.

Definition 3.1 *Lebesque measure is defined as* $\lambda(a,b) = b - a$ *for* $0 \leq a \leq b < 1$ *and has the following properties:*

1. $\lambda(0) = 0$ *and* $\lambda(S') = 1$
2. *If* $B_1, B_2, ... \in \mathbf{B}$ *are pairwise disjoint, then* $\lambda(\bigcup_{n=1}^{\infty} B_n) = \Sigma_{n=1}^{\infty} \lambda(B_n)$

There are several methods for describing dynamical systems. We will be using one which involves Borel sets and Lebesque measure.

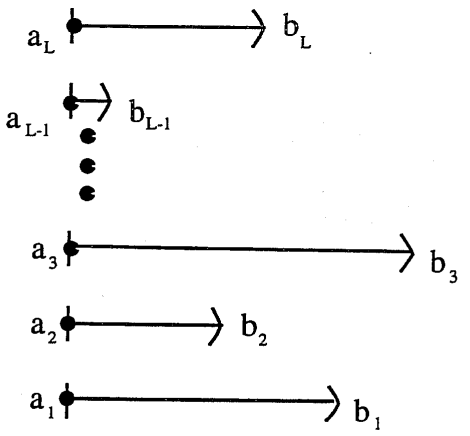
Definition 3.2 *If $T : S' \longrightarrow S'$ is a dynamical system, then*

1. *T is 1-1 and onto on a set of Lebesque measure 1*
2. *$\forall B \in \mathbf{B}$, $\lambda(B) = \lambda(T^{-1}(B))$, where $T^{-1}(B) = \{y | Ty \in B\}$*

Example 3.1 *Let $\alpha \in S'$ be an irrational number and let $T(x) = x + \alpha \ (mod\ 1)$. Then $\lambda(a,b) = b - a$ and $\lambda(T^{-1}(a,b)) = \lambda(a - \alpha, b - \alpha) = b - a$. When two measures agree on intervals, they are the same. Thus, this is a dynamical system.*

3.1.2 Cutting and Stacking

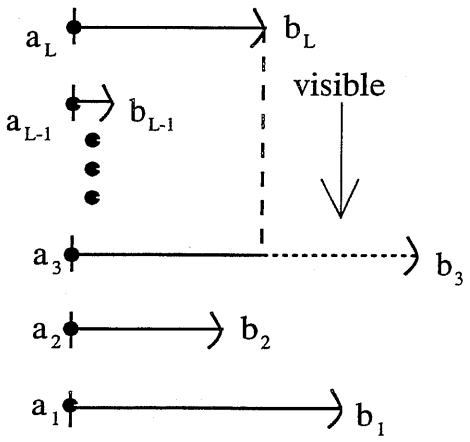
Defining the stack



Definition 3.3 *A stack is a finite collection of disjoint intervals $I_1, I_2, \ldots, I_L \subseteq S'$, where $I_i = [a_i, b_i)$.*

Each stack is accompanied by a transformation T. If $x \in I_i$, then $T(x) = a_j + (x - a_i)$, where $a_j + (x - a_i) < b_j$. Otherwise $T(x)$ is not defined. In other words, when looking "up" the stack, if there is an interval above the interval containing x, then $T(x)$ is defined to be the point directly above x in the interval above x. If $T(x)$ is not defined at x, then we say x

is *visible* from above. Simply stated, if you look down the stack, you can see the point x on its interval.

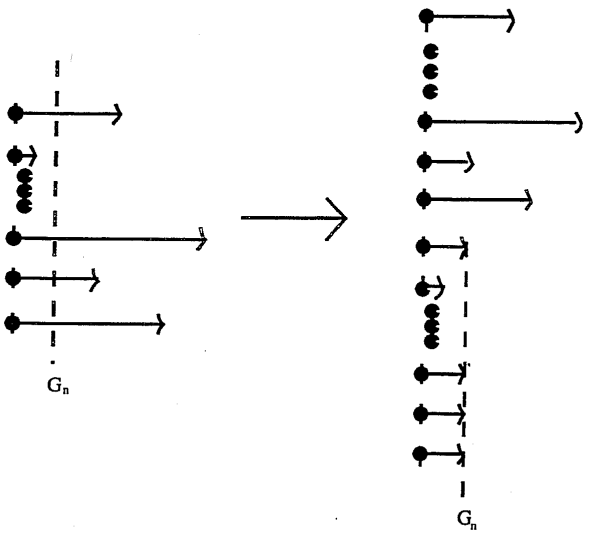


The *width*, w_L, of the stack is $\max_{i=1,..,L}(b_i - a_i)$. Basically, the width of the stack is the width of the longest interval. If $T(x)$ is undefined, we will define $\lambda(x) = w_L$. Then $T(x)$ will preserve measure when it is defined and visible from above.

Similiarly we need to take care of when $T(x)$ fails to have an inverse. $T(x)_{-1}$ does not exist when there is no y such that $T(y) = x$. We say that these points are *visible* from below. A point, x, is visible from below if you can look "up" the stack and see x. If the point, x, is visible from below, then $\lambda(x) = w_L$. Then $T(x)$ will preserve measure when visible from below. Thus $T(x)$ is a measure preserving transformation and the stack is a dynamical stystem.

Cutting the stack

In order to use the stack to create sequences we need to *cut* the stack. Let $I_i = [a_i, b_i)$ for $1 \leq i \leq L$ be a stack of intervals. Given $\gamma_n > 0$, we form the new stack by "cutting at γ_n and stacking on top."



For each I_i, we define $I_i' = [a_i, \min(b_i, a_i+\gamma_n\ ))$. If $a_i+\gamma_n < b_i$ then we get $I_i'' = [a_i+\gamma_n\ , b_i$, otherwise, $I_i'' = 0$. Then we have a stack $I_1', I_2', \ldots, I_L', I_1'', I_2'', \ldots, I_L''$, where $I_i'' = 0$ for some i. Then we remove all I_i'' such that $I_i'' = 0$. The reindex the stack to get, $I_1', I_2', \ldots, I_{L'}'$, the new stack.

Note that $L' = L$ only if $\gamma_n < w_L$. Otherwise, $L' > L$. Also, the $T(x)$ associated with the stack $\{I_i' | 1 \leq i \leq L'\}$ extends the $T(x)$ with $\{I_i | 1 \leq i \leq L\}$. The sequence generated from the *cutting and stacking* method is of the form $\{a_i\}_{i=1}^{i=L}$, where a_i is from $I_i = [a_i, b_i)$.

3.2 Effects of the direction of the scales

In Section 3.1.1 we defined the dynamical system using $\gamma_n > 0$. If γ_n is negative for any n, then we have to decide how to cut the stack. We tried to adjust the transformation so that it went downwards rather than upwards. However, in order to do this, the intervals would need to be lined up on the left rather than the right, otherwise the γ_n's would not cut properly. Then we decided to leave the issue of negative γ_n's out of the dynamical system. We adjusted the formation of the *continued fraction* sequence to also be a non-alternating sequence. Instead of having the γ_n's alternate in sign, we set all γ_n's > 0. The spacing of the points is not changed so the discrepancy of

the sequence is not affected by this redefinement.

Before deciding to consider only γ_n's > 0, we did some testing of the discrepany of sequences formed with alternating γ_n's and strictly positive γ_n's . We were hoping that the discrepancies of the two sequences would be identical for all N, but unfortunately this was not so. We used the following set of scales to test discrepancies of the two sequences.

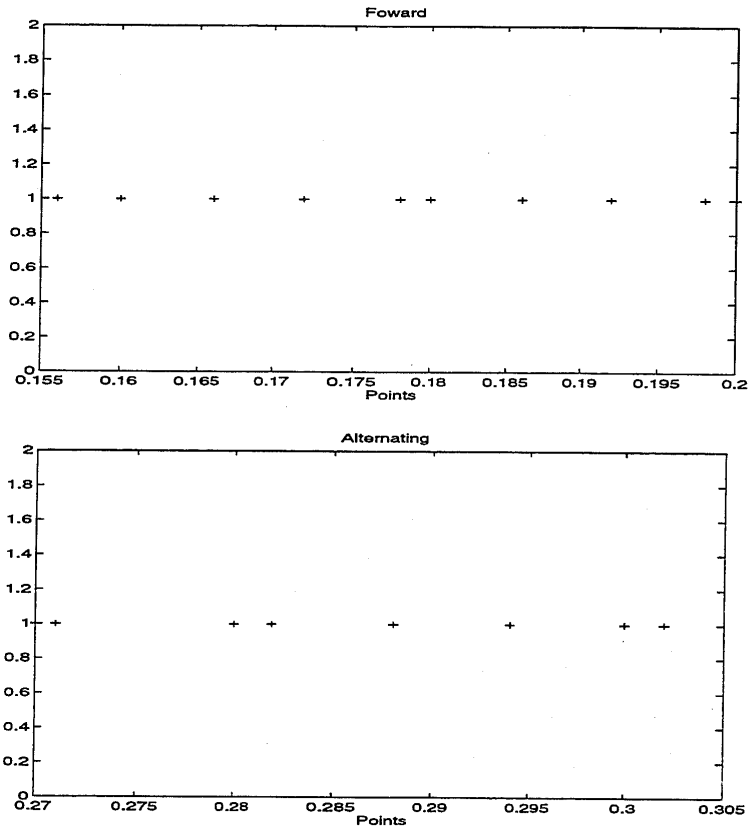
$$\gamma_1 = 0.5,\ \gamma_2 = 0.2,\ \gamma_3 = 0.09,\ \gamma_4 = 0.07,\ \gamma_5 = 0.03,\ \gamma_6 = 0.006$$

The discrepancies were calculated for both of the sequences at the end of each γ_n stage. The table of the discrepancies is given below.

γ_n	*foward*	*alternating*
0.5	0.5	0.5
0.2	0.2333	0.2333
0.09	0.1300	0.1300
0.07	0.0908	0.0908
0.03	0.0470	0.0470
0.006	0.0116	0.0124

The table shows that the discrepancy of the sequences is not alike. Thus, $\gamma_n > 0$ makes a difference in the discrepancy. The discrepancies of the two sequences differed throughout the γ_n stages and were only exactly equal at the end of the γ_n stages.

At γ_6 both sequences have tiny "slivers" of space. Basically, there ends up being a small gap between two points. In the alternating sequence these gaps are closer together than in the foward sequence, resulting in a higher discrepancy.



The results from this example suggest that if you modify the γ_n's so that little slivers between points don't occur, you can keep the dicrepancies closer together.

3.3 Generalizing the scales

Keeping the spirit of subsection 1.3.2, we can generalize the continued fraction sequence. In other words, we can generalize the algorithm in subsection 1.3.2 to one that takes a sequence of $\{\gamma_n\}$ as input and outputs a sequence $\{x_n\}_{n=0}^{\infty}$.

Generalized Algorithm

Let $\{\gamma_n\}_{k=1}^{\infty}$ be given, such that each $0 < |\gamma_n| < 1$ and $\{|\gamma_n|\} \downarrow 0$. Define the following functions:

$S(i, \{x_n\}_{k=1}^N)$ = returns the index of z
where z is such that
$z - x_i = \min_k\{x_k - x_i | x_k - x_i > 0\}$

$P(i, \{x_n\}_{k=1}^N)$ = returns the index of z
where z is such that
$x_i - z = \min_k\{x_i - x_k | x_i - x_k > 0\}$

Let N be the index of the last point in the current list of points. Set $N = 0, x_0 = 0, n = 1$.

Algorithm

STEP 1

Case $\gamma_1 > 0$

Set list of points to be:

$$\{x_n\}_{k=0}^m = \{x_k = k * \gamma_1\}$$

Where $m = \lfloor \frac{1}{\gamma_1} \rfloor$. Set $N = m$

Case $\gamma_1 < 0$

Set list of points to be:

$$\{x_n\}_{k=0}^m = \{0, 1 - k * \gamma_1 | k \in \{1..m\}\}$$

Where $m = \lfloor \frac{1}{\gamma_1} \rfloor$. Set $N = m, n = 2$

STEP 2

Let $P(i) = P(i, \{x_n\}_{k=0}^N)$ where N is understood to be the index of the last point in the current list of points. Similarly, define $S(i)$. Let $\gamma = \gamma_n, N_{old} = N + 1$. Do the following loops

While $(N_{old} \neq N)$
$N_{old} = N$
For ($i = 0$ to N)
If ($x_{S(i)} - x_i > \gamma$ and $\gamma > 0$ and $i \neq N$)
$N = N + 1$
$x_N = x_i + \gamma$

Continue
If ($x_i - x_{P(i)} > \gamma$ and $\gamma < 0$ a $i \neq 0$)
$N = N + 1$
$x_N = x_i + \gamma$
Continue

STEP 3
Set $n = n + 1$, repeat **STEP 2**

3.4 Possible conditions to assure low discrepancy

Our strong belief that sequences generated by the Generalized Algorithm (GA) would be similiar to those generated by the algorithm of subsection 1.3.2, led us to believe that most of conditions that guaranteed low discrepancy in continued fraction sequences will be sufficient to guarantee low discrepancy in GA sequences.

Using a result by Niederreiter [4],

Theorem 3.1 *If the irrational z is such that there exists a positive integer K with $a_i \leq K$ (a_i are the continued fraction coeficients) for all $i \geq 1$, ten*

$$D_N(S(z)) < G(K)N^{-1}log(N+1)$$

for all $N \geq 1$, where $G(K)$ is a constant dependent on K alone

If we set the $\{\gamma_n\}$ to be generated by some continued fraction of some number, then theorem 3.1 becomes

Theorem 3.2 *If $\{\gamma_n\}$ are from a continued fraction and are such that $|\frac{\gamma_n}{\gamma_{n+1}}| < K$, for some $K \geq 1$ then*

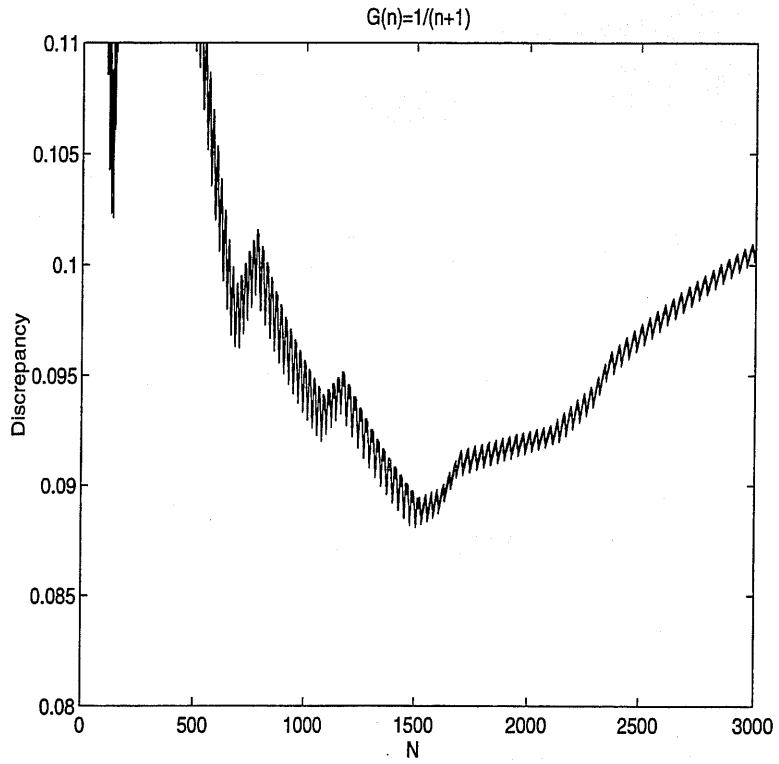
$$D_N(\{x_n\}) = O(\frac{\ln N}{N})$$

Where $\{x_n\}$ is the sequence generated by GA algorithm

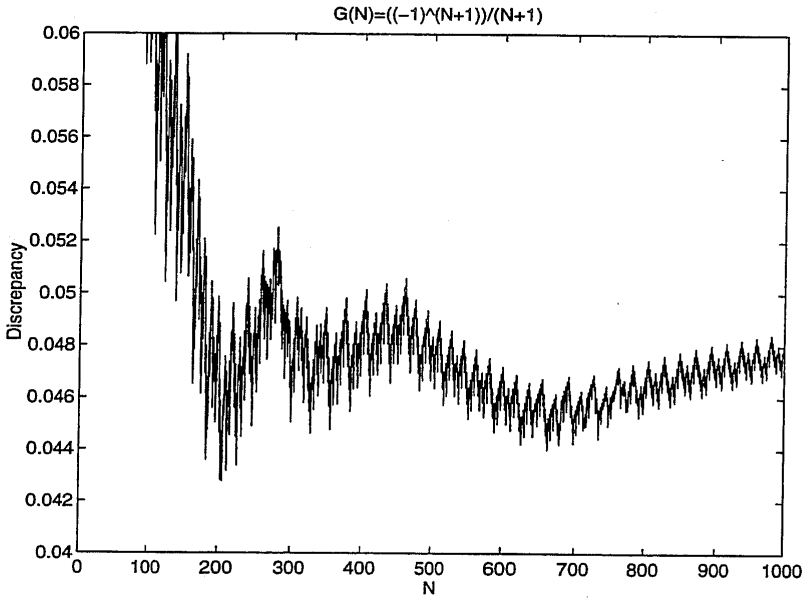
Thus keeping intuition in one hand and luck in the other, we conjectured

Conjecture 3.1 *If* $\{\gamma_n\}$ *is such that* $|\frac{\gamma_n}{\gamma_{n+1}}| < K$ *, then the GA algorithm produces a low discrepancy sequence.*

However we were unlucky, because if $\{\gamma_n\} = \frac{1}{n+1}$, the GA algorithm produces a sequence whose discrepancy seems to be above .01, most of the time.



And even if we let the γ_n's alternate, the discrepancy seems to stay above .04 most of the time.

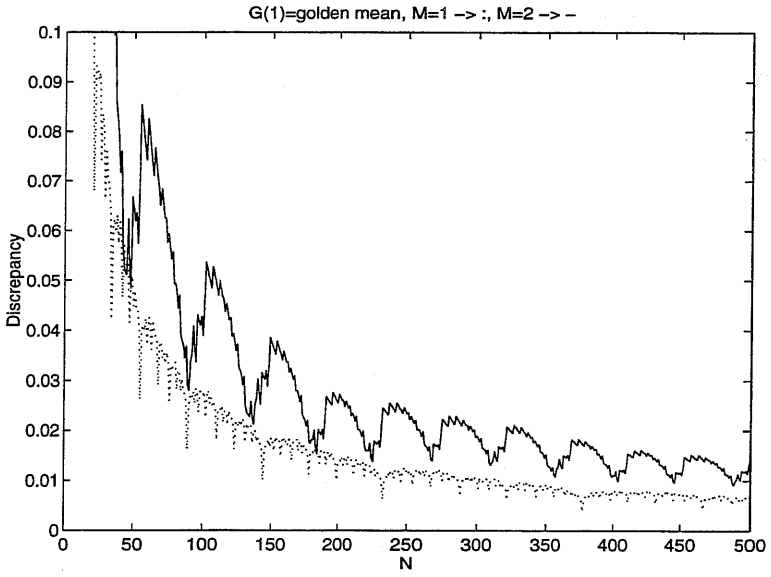


3.5 Sequences similar to a continued fraction

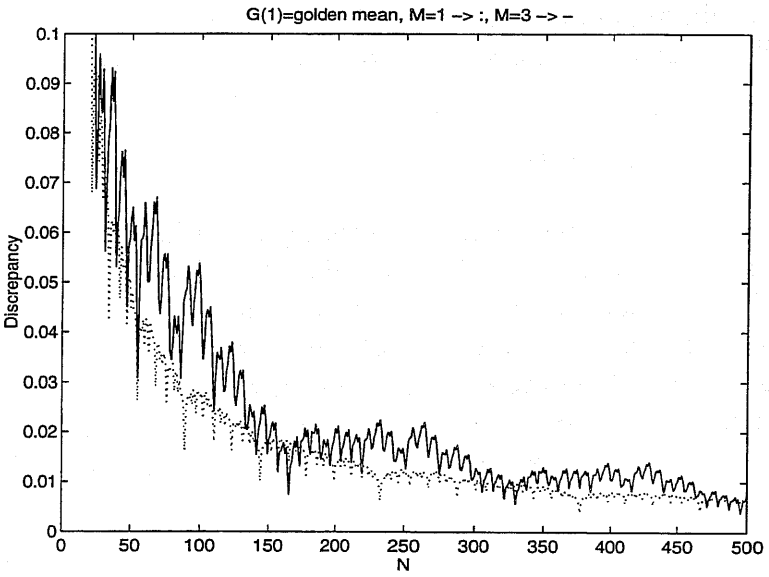
We tried to set more stronger conditions on the $\{\gamma_n\}$ so that it will behave more like a continued fraction. Namely, we put the following condition to the γ_n's

$$\gamma_{n+2} = \frac{(-1)^{n+1}}{M}\{\gamma_n - \lfloor\frac{\gamma_n}{\gamma_{n+1}}\rfloor * \gamma_{n+1}\}$$

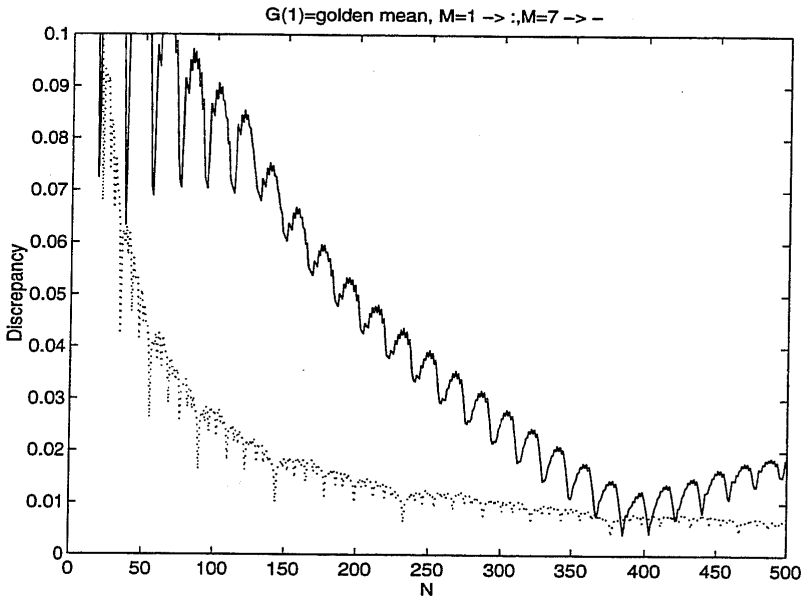
Where M is a fixed integer, and $\gamma_0 = 1, \gamma_1 = x$. For instance, if we let $x = \frac{1+5^{\frac{1}{2}}}{2}$, and let $M = 2$, we can see how much the discrepancy's behavior differs from the continued fraction case $M = 1$



Here is the comparison for $M = 3$



Here is the comparison for $M = 7$



In general, we saw that discrepancies $M = n \neq 1$ will eventually come back and touch the discrepancy graph for $M = 1$ and try to follow it. However, there is still a chance that for some $\gamma_1 = x_o$ this may not happen.

3.6 Error conditions

Given a sequence of γ_n's, we tried to impose another condition which will assure us a low discrepancy sequence. Whether it does is still an open question. Before we show the condition, we define some functions.

Let X_N be the sequence of the first N points generated by GA. Given X_N, we can form a partition of $[0,1)$ (add 1 to X_N before creating the partition), define $f(n,N)$ to be the number of intervals of length γ_n in the partition generated by X_N, let $S(n)$ to be the value N for which $f(n,N)$ reaches a maximum.

Let $e(i,0) = \gamma_{i-1}$, $e(i,k) = M(e(i,k-1),\gamma_k)$, where $M(x,y) = x - \lfloor \frac{x}{y} \rfloor y$, also let $\gamma_0 = 1$. Let $G(n) = f(n,S(n))$

Conjecture 3.2 *if* $\lim_{n\to\infty} G(n)\gamma_n = 1$ *then GA generates a Low discrepancy sequence*

Unfortunatly, we were not able to show that conjecture 3.2 was true or false. However, in the process of trying we found the following fact.

Theorem 3.3 $G(n) = \sum_{i=1}^{n} \lfloor \frac{e(i,n-i)}{\gamma_n} \rfloor G(i-1)$ *where,* $G(0) = 1$ *by convention.*

Proof:
By induction

Case $n = 1$
This case reduces to claiming

$$G(1) = \lfloor \frac{1}{\gamma_1} \rfloor$$

Which says that the most number of intervals of length γ_1 generated by partitions generated by sequences using GA is $\lfloor \frac{1}{\gamma_1} \rfloor$. This is being true, since such a maximum is reached by $X_{\lfloor \frac{1}{\gamma_1} \rfloor}$

Case $n = k + 1$
Define $G(l, n)$ be the number of intervals of length l generated in the partition generated by x_n. Note that in general, $G(k)$ is reached by $X_{S(k)}$, for all k. Let the partition generated by $X_{S(k)}$ be $P(k)$. $P(k)$ contains intervals of different lengths. Those lengths are of the form $\{e(i, k+1-i) | i = 1..k+1\}$ thusthere are $k+1$ intervals of different lengths in $P(k)$. In $P(k)$ the biggest interval is of length $e(k+1, 0) = \gamma_k$, in $P(k+1)$ al the intervals of $P(k)$ got partitioned so as to make γ_{k+1} the biggest interval, in the process of going from $P(k)$ to $P(k+1)$, each of the $k+1$ different intervals of $P(k)$ will have to get partitioned with the γ_{k+1}'s,after this is done, we get $P(k+1)$, and in this new partition there will be $k+2$ intervals,namely intervals of length $\{e(i, k+2-i) | i = 1..k+2\}$. Thus we can caculate the total number of intervals of length γ_{k+1} generated in P(k+1):

$$G(k+1) = \sum_{i=1}^{k+1} \lfloor \frac{e(i, k+1-i)}{\gamma_{k+1}} \rfloor G(e(i, k+1-i), S(k))$$

However, $G(e(i, k+1-i), S(k)) = G(i-1)$, for $i = k+1$ is obvious (use inductive hypothesis) , for the other possible i values, use the fact $G(e(i, M), SS(M)) = G(e(i, M-1), SS(M-1))$, where $SS(M)$ is the value of N that maximizes $G(e(i, M), N)$. Thus , the formula follows for $k+1$

□

$G(n)$ is a measure of frequency of the biggest spacing between points in $X_{Sn)}$. The last theorem, tells us that the behavior of $G(n)$, to be at least exponential, at most super exponential. We hoped that may be this exponential behavior might allow $|S(n) - G(n)| < K$, for some K, and all $n > N_o$

If $|S(n) - G(n)| < K$, for some K and all $n > N_o$, we can assure that most of the points in $X_{S(n)}$ will be taken up by the set of points W generating the intervals of length γ_n, such a set W has low discrepancy (it somewhat looks like a set of evenly spaced poins), the remaining points, there can only be at most K of them, a finite number, which could increase the discrepancy slightly, but by making n big enough, such an increase could be still pull down (we hope), and eventually be made as small as we could. This reasoning lead us to believe conjecture 3.2.

Chapter 4

Quasi-Monte Carlo Integration

Monte Carlo integration is a method of calculating integrals using random points. In *quasi-Monte Carlo* integration, instead of trying to use random points, deterministic points are used. It turns out uniformly distributed points work best for quasi-Monte Carlo integration [3]. Let $I^s = [0,1)^s$. Then the quasi-Monte Carlo approximation is as follows:

$$\int_{I^s} f(\mathbf{t})d\mathbf{t} \approx \frac{1}{N}\Sigma_{n=1}^{N} f(\mathbf{x}_n).$$

In $\mathbf{R}^n$, the approximation of most integrals involves n cross products. The quasi-Monte Carlo approximation, however, requires no cross products and ends up being a much simplier calculation. This is an advantage of quasi-Monte Carlo approximations. The preciseness of the approximation is determined by the discrepancy of the sequence, as shown by the theorem below.

Theorem 4.1 *If f is a function of bounded variation V(f) on $[0,1)$ and $\{x_1, x_2, \ldots, x_N\}$ is a sequence on $[0,1)$ with discrepancy D_N^*, then*

$$\left|\int_{I^s} f(\mathbf{t})d\mathbf{t} - \frac{1}{N}\Sigma_{n=1}^{N} f(\mathbf{x}_n)\right| \leq V(f)D_N^*$$

The proof can be found in Niederreiter [3]. This shows that the uniform distribution of the sequence determines the accuracy of the approximation. Thus, finding low discrepancy sequences has applications in quasi-Monte Carlo integration.

Appendix A

```
// ---------------------->Main Program of the project<----------------------
//   other programs were built but this one is the most general one
//    all the other programs were put into REU96CPP.BOB.tar.gz located at
// /amaterasu/sd2e/reu96/salazc/
//    which is a gziped tar file containing all the programs of the
//    project
// ------------------------------------------------------------------------
// Program confrac1.1.cc
// usage: confrac1.1 <filename>
//        <filename> is the name of a file that contains
//          as first number the size of Gamma vector
//          second number Maximun number of points on the sequence to be
//          generated, the rest of the numbers are numbers of the Gamma
//          vector
// function: Calculates discrepancies of sequences generated using the
//     Generalized Algorithm given the Gamma_n's, the biggest
//      sequence generated is of size Maximun number of points
//          given in the input file
//       The program generates a double linked list in a foward fashion
//          the main element of the list is a struct num, which contains two
//          self referencial pointers and a double
//           The linked list are generated by the class Array, and all
//     the functions transforming the double linked list are elements
//          of the class Array.

#include <iostream.h>
#include <fstream.h>
#include "confrac1.1.h"
#include "discrep.h"
#include "sort.h"

void main(int argc,char* argv[])
{
  ifstream in(argv[1]);
  int MAX_PTS,SIZE,SIZE_SEQUENCE,i;
```

```
   double* GAMMA;
   double* SEQUENCE;
   double* Discrep;
   double* temp;

   GAMMA=load(in,SIZE,MAX_PTS);

   array a;

   SEQUENCE=a.confrac(GAMMA,SIZE,MAX_PTS);
   SIZE_SEQUENCE=a.len();

   Discrep=new double[MAX_PTS];

   for(i=0;i<MAX_PTS;i++)
     {
       temp=a.toarray(2,0,i+1);
       qsort(temp,0,i);
       Discrep[i]=discrepancy(temp,i+1);
     }

   ofstream seq("confracseq.txt");
   ofstream disc("confracdiscrep.txt");

   for(i=0;i<SIZE_SEQUENCE;i++)
     seq<<SEQUENCE[i]<<"\n";
   for(i=0;i<MAX_PTS;i++)
     disc<<Discrep[i]<<"\n";

}

/*************************************************************************
*************** confrac1.1.h ***********************************
*************** header file  ***********************************
*************************************************************************/

#include <fstream.h>
```

```
// Base structure of the list
struct num{
  double val;
  num* next;
  num* o_next;
  num(){val=0.0;next=(num *)NULL;o_next=(num *)NULL;}
  ~num(){delete &val;delete next;delete o_next;}
};

// Structure that manipulates the list
struct array{
  int length;
  int freq;
  num* base;
  num* ptr;
  num* lastptr;

  array();
//  ~array();
  double* toarray(int a,int start,int end);
  int len(){return (length-1);}
  int how_many(num* current,double GAMMA);
  num* add(num* current,double GAMMA);
  double* confrac(double* GAMMA,int GAMMA_SIZE,int MAX_PTS,int format);
};

 array::array()
{
  length=2;
  freq=0;
  base=new num;
  base->val=0.0;
  base->o_next=new num;
  base->o_next->val=1.0;
  ptr=base; // not sure yet
  lastptr=base;
```

```
}
/*
 array::~array()
{
  delete base;
  delete ptr;
  delete lastptr;
  delete &freq;
  delete &length;
}
*/

// prints out elements of the list to an array,
// example: if the list has N elements
//     toarray(1,3,N-5)
//    will print out elments 3..N-5 to array in format 1
//    the first input is the format,
//     format=1 prints out in magnitudinal order
//     format=2 prints out in sequential order (order in which the
// point was added)

double* array::toarray(int a,int start=0,int end=-1)
{
  if (end==-1)
    end=len();

  double* out;
  num* temp;
  int i;
  temp=base;

  out=new double[end-start];

  if (a==2)
    {
      for(i=0;i<start;i++)
temp=temp->next;
```

```
      for(i=start;i<end;i++)
 {
   out[i-start]=temp->val;
   temp=temp->next;
 }
    }
   else if (a==1)
     {
        for(i=0;i<start;i++)
  temp=temp->o_next;

       for(i=start;i<end;i++)
  {
    out[i-start]=temp->val;
    temp=temp->o_next;
  }
     }
   return out;

}
double abs(double x)
{
  return ( ( x >= 0.0 )? x: -x );
}

// This function decides whether to add point to list or not
int array::how_many(num* current,double GAMMA)
{
  if (current==(num* )NULL)
    return (-1);
   if (( ( (current->o_next->val)-(current->val) ) - abs(GAMMA) )<= 0.00000000
    // WATCH OUT THIS STATEMENT IS MADE SO THAT THE ROUND OFF ERROR WONT KILL T
      return (-3);
  else
    return (1);
}
```

```
// This functions adds one point to the list properly
num* array::add(num* current,double GAMMA)
{

  num* temp;


  temp=current;
  double x;

  if(GAMMA >= 0.0)
    x=temp->val;
  else
    x=temp->o_next->val;

  x+=GAMMA;

//  cout<<"\n adding "<<x<<"\n";

  ptr=current->o_next;
  current=current->next;


  temp->o_next=new num;
  temp->o_next->val=x;

  temp->o_next->o_next=ptr;

  lastptr->next=temp->o_next;
  lastptr=temp->o_next;

  freq=1;
  length++;


/*  if ( (current->val) == 1.0 )
```

```
    current=(num *)NULL;

   int i;
   double* out1;
   out1=toarray(2);

   cout<<"\n";
   for(i=0;i<len();i++)
     cout<<out1[i]<<"\n";
*/
  return current;
}


// Adds points to the list in recursive fashion
double* array::confrac(double* GAMMA,int GAMMA_SIZE,int MAX_PTS,int format=2)
{
  num* current=base;
  num* last1=lastptr;
  int h,i,j;
//  double* out1;

  for(i=0;((i<GAMMA_SIZE)&&(len() < MAX_PTS));i++)
    {
do
  {
    freq=0;
    current=base;


    do
      {
h=how_many(current,GAMMA[i]);
/* cout<<"\n h values was: "<<h;
cout<<" current= "<<current->val<<"\n";
*/
if((h==-1)||(h==-2))
```

```
  current=(num* )NULL;
else if (h==-3)
  current=current->next;
else
  current=add(current,GAMMA[i]);
      }while( (current)&&(len()<MAX_PTS) );

//    if (i!=0)// this could be erased if doubt of efficiency
  //    freq=0;
/*

    out1=toarray(1);
    cout<<"\n here is the sequence after finishing a cycle";
    cout<<" Gamma was: "<<GAMMA[i]<<"\n";
    for(j=0;j<len();j++)
      cout<<out1[j]<<"\n";
*/
        }while(freq!=0);

    }
  return toarray(format);

}

// Loads the Gamma vector
double* load(ifstream& in,int& SIZE,int& MAX_PTS)
{
  in>> SIZE;
  in>>MAX_PTS;
  double* out=new double[SIZE];
  for(int i=0;i<SIZE;i++)
    in>>out[i];
  return out;
}
```

Bibliography

[1] L. Kuipers and H. Niederreiter. *Uniform distribution of sequences.* Wiley, New York, 1974.

[2] W. J. Morokoff and R. E. Caflisch. Quasi-monte carlo integration. *Journal of Computational Physics*, 122:218–230, 1995.

[3] H. Niederreiter. Quasi-monte carlo methods and pseudo-random numbers. *Bulletin of the American Mathematical Society*, 84(6):957– 1041, 1978.

[4] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* Society for Industrial and Applied Mathematics, 1992.

[5] H. S. Wall. *Continued Fractions.* D. Van Nostrand Company, Inc., 1948.