The Pólya-Vinogradov inequality and the distribution of quadratic residues mod p

MATH 471 - Sam Gwydir

<2017-03-19 Sun>

Contents

1 Introduction

Generating random numbers is an important problem in computer science and applied mathematics and in particular, in cryptography. Many cryptographic algorithms depend on the users' ability to generate random numbers including generating keys for one-time pads or testing primality via Miller-Rabin, Solovay-Strassen or the Fermat primality test. While sources of "true" random numbers exist, such as measuring atmospheric or thermal noise, these sources are limited in the amount of entropy they can provide and therefore bottleneck the speed of routines which make use of them [1]. Thus most systems use "pseudo" random number generation which generate a seemingly random sequence of numbers that is "hard" to determine without knowing a secret key, or seed. Therefore finding methods of pseudo random number generation is of great interest to students of cyptography.

Cryptography makes great use of the concept of the quadratic residue in a variety of applications. An integer q is a quadratic residue mod p if there is another integer x such that $x^2 \equiv q \pmod{p}$ [2]. To date, the distribution of quadratic residues and non-residues is an open problem, though several bounds concerning the number of quadratic residues have been developed. In 1918, Pólya and Vinogradov independently showed that $\left(\frac{n}{p}\right)$ for consecutive values of n can be seen as random coin flips with values of ± 1. In particular they showed on an interval of length N, the number of consecutive quadratic residues modulo p is

$$\frac{1}{2}N + O\left(\sqrt{p}\log p\right)$$

. Thus we can state

$$\left|\sum_{n=M+1}^{M+N}\left(\frac{n}{p}\right)\right| < \sqrt{p}\log p$$

which gives us a bound on the number of identical consecutive "coin tosses".

In the following short paper we will investigate how "random" this sequence is, how we can best compute these values and what other related bounds exist.

[1] `https://en.wikipedia.org/wiki/Random_number_generation#Practical_applications_and_uses`

[2] `https://en.wikipedia.org/wiki/Quadratic_residue`

2 Pólya-Vinogradov Inequality

First we must evaluate what the Pólya-Vinogradov Inequality tells us. This inequality can be interpreted as a measure of the "statistical fluctuation" of the number of quadratic residues and non residues in the sequence of Legendre symbols on a prime p. In particular the inequality says that this count is bounded by $\sqrt{p}\log p$, though for p large, this becomes $\sqrt{p}$. In addition we know from the inequality that in an interval of length N, there are

$$\frac{1}{2}N + O\left(\sqrt{p}\log p\right)$$

consecutive quadratic residues $\pmod p$. Thus we know the least quadratic non-residue $\pmod p$, is bounded above by $O(\sqrt{p}\log p)$ [3].

Later, finer upper bounds were found, but they do not improve substantially upon Pólya-Vinogradov. In 1987 Cochrane obtained that

$$\left|\sum_{n=M+1}^{M+N}\left(\frac{n}{p}\right)\right| < \frac{4}{\pi^2}\sqrt{p}\log p + 0.41\sqrt{p} + 0.61$$

.

Other bounds can be found if we assume certain facts about the distribution of primes. Notably, Montgomery and Vaughan (1977) showed that if we assume the truth of the Generalized Riemann Hypothesis then the estimate of the character sum above becomes $O(\sqrt{p}\log\log p)$ [4] In addition, Schur (1918) provides a lower bound,

$$\left|\sum_{n=1}^{p-1}\left(\frac{n}{p}\right)\right| > \frac{1}{2\pi}\sqrt{p}$$

These bounds are visualized in Figure 1. In the following paragraphs we will compare the real world results against these bounds.

[3] `http://thales.doa.fmph.uniba.sk/macaj/skola/teoriapoli/primes.pdf`

[4] `https://en.wikipedia.org/wiki/Generalized_Riemann_hypothesis`

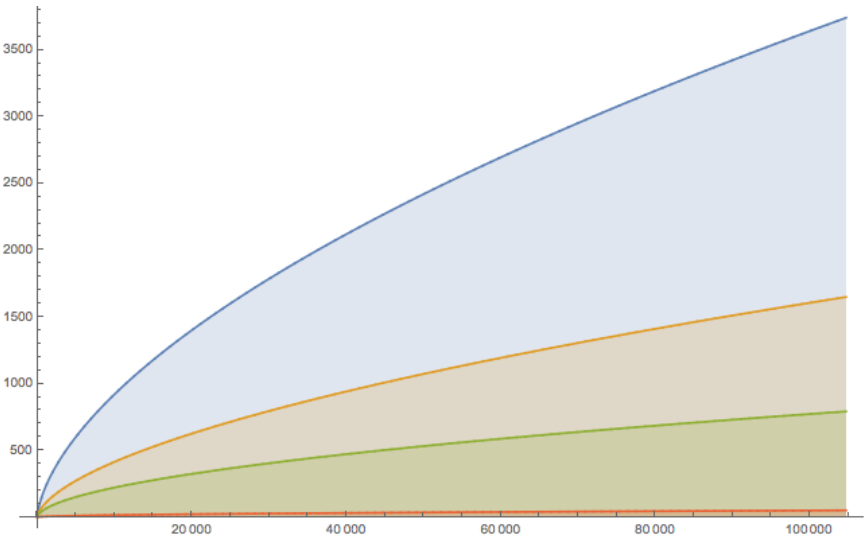

Figure 1: These bounds visualized. Polya-Vinogradov (Blue), Cochrane(Orange), MontgomeryVaughan(Green), Schur(Red)

3 Findings

3.1 Computing Quadratic Residues

The Legendre symbol is defined as[5]:

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \pmod{p} = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \text{ and } a \not\equiv 0 \pmod{p}, \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p, \\ 0 & \text{if } a \equiv 0 \pmod{p}. \end{cases}$$

This indicates if we were to naively evaluate the sum in the Pólya-Vinogradov inequality, the sequence to be summed would be made up of 3 values namely, 0, 1 and -1. This would require a modular exponentiation which are expensive. Obviously (and at the suggestion of the professor) this is not the direction to go first.

Firstly when evaluating the sum we will choose a range from 1 to N for some $N > 1$. If we choose our N to be $p - 1$ then instead of calculating the Legendre symbol, we can directly find the quadratic residues in this range by squaring the numbers in the interval $\frac{p-1}{2}$ taken $\pmod{p}$. If our interval from 1 to N is smaller than p we know there are no zeroes, thus all other

[5] `https://en.wikipedia.org/wiki/Legendre_symbol`

values are -1. If this interval is larger than p, we leverage the fact that the Legendre symbol is periodic in p. Finding the quadratic residues is the same as before but we should then find the first integer in our interval, n such that $n|p$. We can then create an array the size of our interval, and place a zero at this index. After this index we can fill in the values $1^2 ... \frac{p-1}{2}^2, 0$ repeating until the array is filled.

However, if we are taking some random interval in 1... that doesn't contain an n that divides p then we can just directly calculate the Legendre symbol. To speed up our calculations we should not use the direct calculation using a modular exponentiation but instead use the supplements to the law of quadratic reciprocity like one would do when evaluating a Legendre symbol by hand. This means pulling off powers of 2 from a in $\left(\frac{a}{p}\right)$ then we have $\left(\frac{2^n}{p}\right)\left(\frac{a}{p}\right)$ and using the law of quadratic reciprocity to simplify our calculations. This is exactly what is done in Victor Shoup's NTL library.

In the next section we will analyze the "randomness" of the sequence produced above. First we will cast our outcomes as "coin flips". Since we know we only have two outcomes, 1 or -1, we can map values of -1 onto 0. Thus our program output pure binary data, without needing to deal with negatives. We can now quickly return blocks of coin flips.

Note carefully that this sequence will repeat on p, thus if this algorithm were to be used as a pseudorandom number generator such as `/dev/random` on UNIX systems, then a new p should be found each time a 0 is reached in a sequence.

3.2 Randomness of Sequence

We've discussed how to implement a pseudorandom number generator using this inequality and implied that it is "random", but we should quantify the quality of our pseudorandom number generator.

There are many ways to measure randomness. Using the `ent`, the "Pseudorandom Number Sequence Test Program", we can deduce several properties of a sequence easily. `ent` produces a measure of entropy, "the information density of the contents of the file" by attempting to compress the input. Next an arithmetic mean is produced. Since our sequence is made up of purely one's and zero's this should always be exactly 0.5. Next a monte carlo simulation estimating Pi is performed using our coin flips as input. This should yield a small error as the input grows. After this a serial correlation coefficient is found. For random sequences this should be 0, meaning terms cannot be predicted using the previous terms. Lastly, and most importantly

a chi-square test is computed. This test is often used to measure randomness of data and is extremely sensitive. We will discuss this measure shortly.

First the first several measures can be showcased. For 400 primes of 6-29 bits, we produced:

n-bit prime	Entropy	Arthmetic Mean	Monte Carlo Error (Pi)	Serial Correlation
6	0.986625	0.5000	27.32%	0.0
7	0.996071	0.5000	27.32%	0.0
8	0.998983	0.5000	23.81%	0.0
9	0.999724	0.5000	12.76%	0.0
10	0.999936	0.5000	10.65%	0.0
11	0.999985	0.5000	6.86%	0.0
12	0.999995	0.5000	4.50%	0.0
13	0.999999	0.5000	3.64%	0.0
14	1.	0.5000	2.33%	0.0
15	1.	0.5000	2.067%	0.0
16	1.	0.5000	1.46%	0.0
17	1.	0.5000	0.96%	0.0
18	1.	0.5000	0.52%	0.0
19	1.	0.5000	0.47%	0.0
20	1.	0.5000	0.32%	0.0
21	1.	0.5000	0.23%	0.0
22	1.	0.5000	0.12%	0.0
23	1.	0.5000	0.12%	0.0
24	1.	0.5000	0.10%	0.0
25	1.	0.5000	0.09%	0.0
26	1.	0.5000	0.09%	0.0
27	1.	0.5000	0.04%	0.0
28	1.	0.5000	0.02%	0.0
29	1.	0.5000	0.01%	0.0

Thus primes over about 30-bits are needed the be random enough to calculate Pi using a monte carlo simulation. If we do not need cryptographically secure random bits, anything over about 14-bits primes are useful.

Finally the Chi-square results. The Chi-Square results should be near 50% plus or minus a large percentage. For example, random numbers generated by radioactive decay had a 41% chance of being random. Here our pseudo random number generator gets outed for what it is – pseudo random. The average case percentage is 99.943%. This puts our generator into what

the `ent` website refers to as "almost certainly not random" range. However, we are in good company. For instance the rand() function from the C standard library is pegged at 99.99%.

Why might this be? Here is where are bounds come back into play. While the legendre symbol outputs sequences that have many of the properties that are desirable of random numbers such as those in the table above, the legendre symbol is tends to have "runs" of 1s or -1s (0's in our implementation), as opposed to flipping back and forth. These "runs" can be described, albeit loosly, by the bounds we are investigating. This is in stark contrast to true random sequences which have no such bounds.

This means while the legendre symbol is interesting for investigating the distribution of quadratic residues, and it is somewhat random, it will not do as a pseudo random number generator.

4 Future Work

In "On the Randomness of Legendre and Jacobi Sequences" Damgard (1998) suggests that Jacobi symbols generate stronger randomness properties than a Legendre sequence. This is worth exploring.

5 Links

- The programs I developed can be found here: `https://github.com/gwydirsam/Polya-Vinogradov-Paper`
- DIEHARDER:

`http://www.phy.duke.edu/~rgb/General/dieharder.php`

- The Number Theoretic Library, NTL: `http://www.shoup.net/ntl/index.html`
- GNU Multi-Precision Arithmetic Library, GMP: `https://gmplib.org/`
- ENT - pseudorandom number sequence test: `https://www.fourmilab.ch/random/`