A Geometric Explanation for Prime-rich Lines on the Ulam Spiral

Mathematics

How can a model based on the cause of prime-rich lines in the Ulam spiral be used to produce prime numbers?

3982 words

Contents

1 Introduction

On the Cartesian plane, the natural numbers are uniquely assigned to all points of integer coordinates (the lattice points) (x, y). The mapping goes: $(0,0) \mapsto 1$, $(1,0) \mapsto 2$, $(1,1) \mapsto 3$, $(0,1) \mapsto 4$, $(-1,1) \mapsto 5$, $(-1,0) \mapsto 6$, $(-1,-1) \mapsto 7$, $(0,-1) \mapsto 8$, $(1,-1) \mapsto 9$, $(2,-1) \mapsto 10$, $(2,0) \mapsto 11$, $(2,1) \mapsto 12$, $(2,2) \mapsto 13$, and continues infinitely outward, as described by Stanislaw Ulam (after whom the spiral is named), Myron Stein, and Mark Wells in *A Visual Display of Some Properties of the Distribution of Primes* (1964). In Figure 1, connecting points that have been assigned consecutive numbers results in a counterclockwise square spiral—the Ulam spiral.

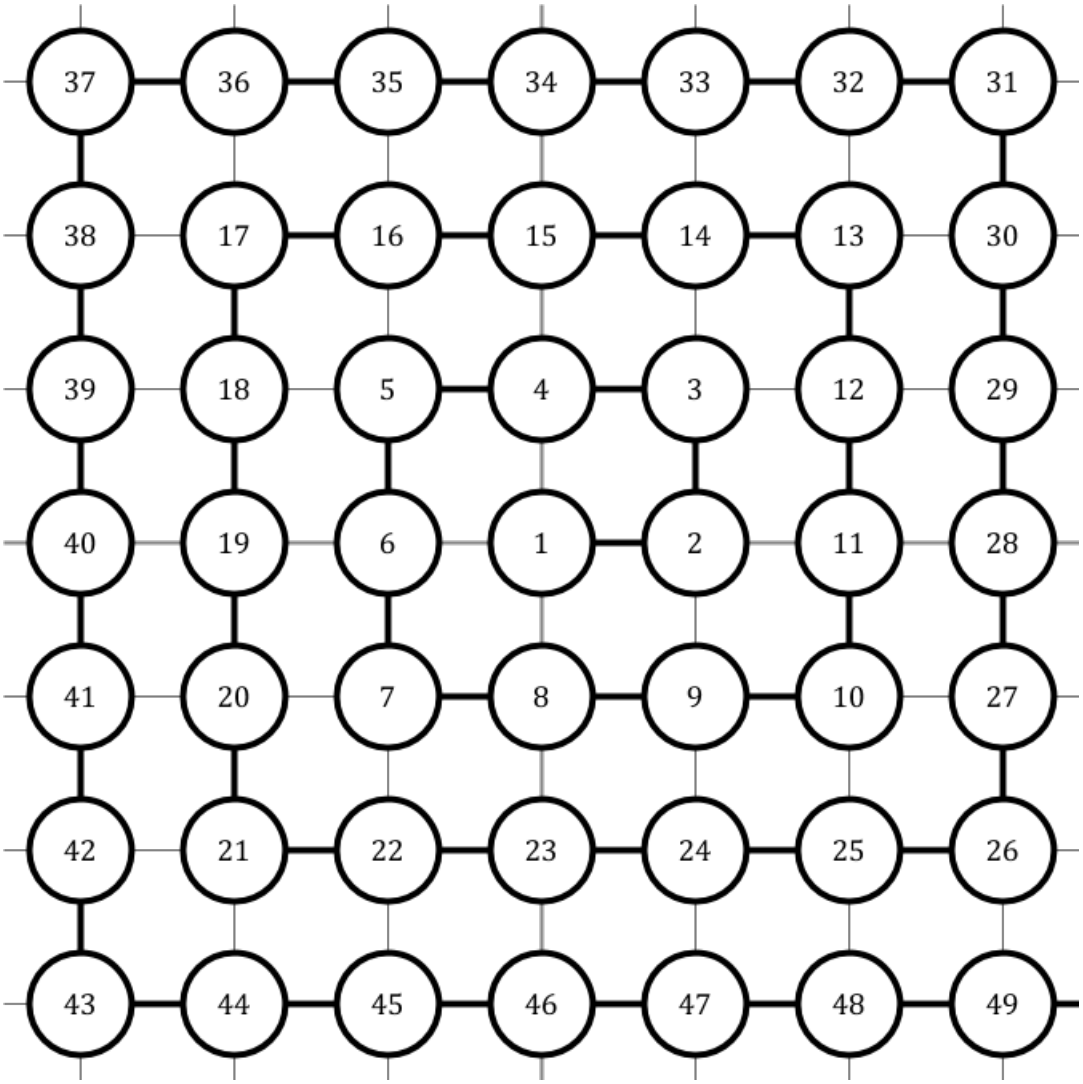


Figure 1: Lattice points in the domain $|x|, |y| \leq 3$ are shown as circles containing the number assigned to them.

In the most common image of the spiral, each lattice point is represented as a single pixel. "Prime points" have been assigned prime numbers, and are coloured black. The curiosity of the spiral is the presence of prominent "prime-rich lines", along which prime points occur with a greater-than-expected frequency. The existence of prime-rich lines is surprising to some, because it shows a pattern that is not intuitively expected. For this reason, the Ulam spiral is popularized as mysterious (Inglis-Arkell, 2011).

In the original paper, prime-rich lines are stated to be connected to the factorability of quadratic expressions (Stein et al., 1964). Robert Sacks thoroughly explains this connection using a round (Archimedean)

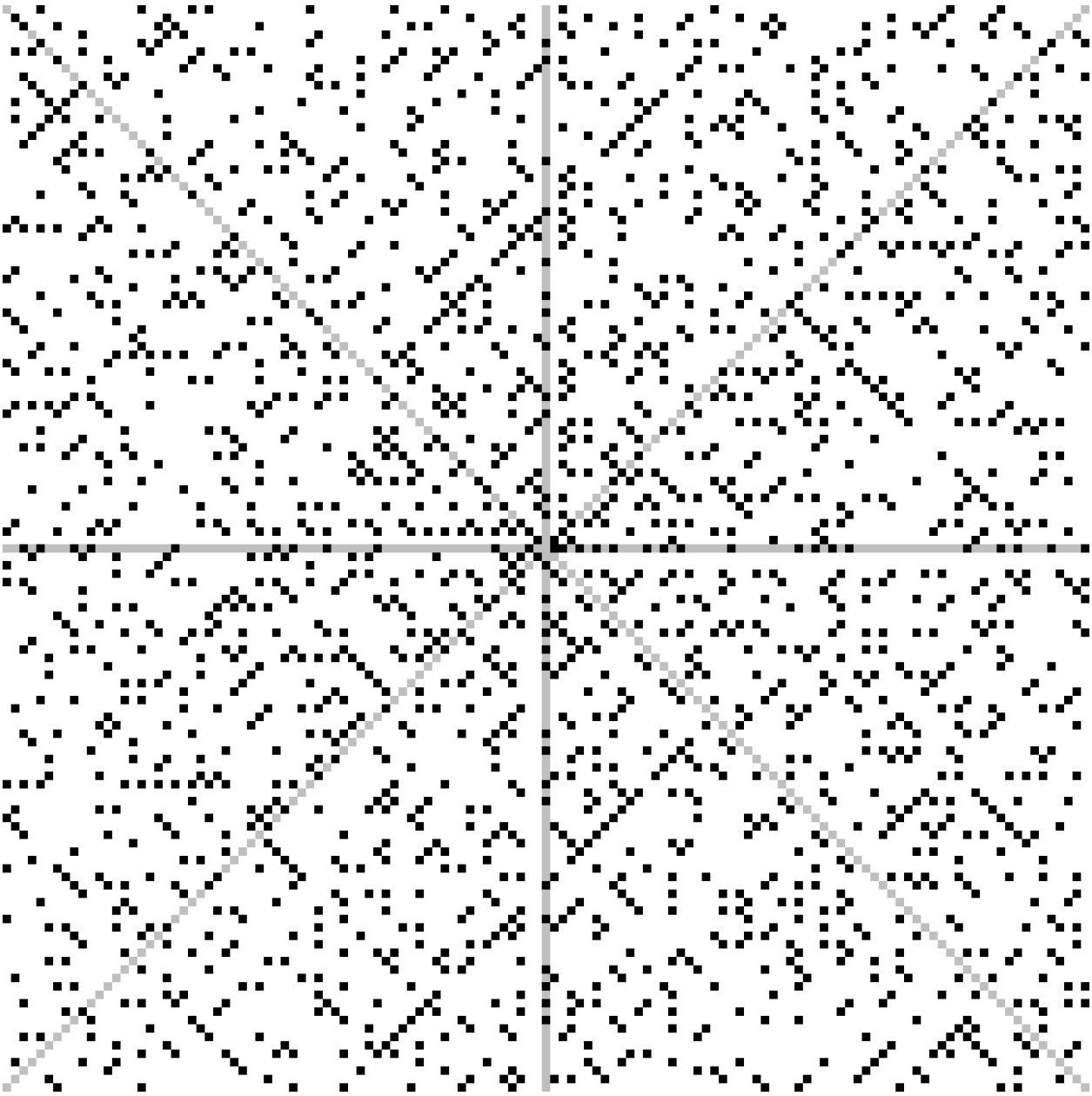


Figure 2: Points $|x|, |y| \leq 64$ in the Ulam spiral, coloured as described. Lines $x = 0$, $y = 0$, and $y = \pm x$ are also shown.

variant of the spiral (2018). Nonetheless, understanding of the Ulam spiral is incomplete. It receives little attention, the majority of which is from amateur mathematicians. This essay presents an in-depth study of an incomplete explanation first mentioned by Tom Bates (2018). It focuses on geometric patterns that appear when points assigned values divisible by a particular number are highlighted. The explanation is implemented in a model to predict the locations of prime points in the spiral. The success of this model leads reflects the success of the explanation, leading to the question: *How can a model based on the cause of prime-rich lines in the Ulam spiral be used to produce prime numbers?*

2 Modelling the spiral

2.1 Wheel factorization

For a factor p, the probability that a randomly chosen number X is indivisible by p is $\frac{p-1}{p}$. X is prime ($X \in \mathbb{P}$) if it is indivisible by every prime number between 2 and $\sqrt{X}$ (inclusive). Suppose X has a value close to x. Then, probabilistic arguments can be made (Granville, 2018):

$$P(X \in \mathbb{P}|X \approx x) \approx P((p_1 \nmid X) \cap (p_2 \nmid X) \cap \ldots \cap (p_k \nmid X))$$

Where p_n is the nth prime, and k is the number of primes less than $\sqrt{x}$. Each event $p \nmid X$ is independent:

$$\therefore P(X \in \mathbb{P}|X \approx x) \approx P(p_1 \nmid X) \times P(p_2 \nmid X) \times \ldots \times P(p_k \nmid X)$$

$$\therefore P(X \in \mathbb{P}|X \approx x) \approx \frac{p_1 - 1}{p_1} \times \frac{p_2 - 1}{p_2} \times \ldots \times \frac{p_k - 1}{p_k} = \prod_{i=1}^{k} \frac{p_i - 1}{p_i} \approx \frac{1}{\ln(x) - 1}$$

Then, one is given that X is not a multiple of p_n, where $1 \leq n \leq k$.

$$P(X \in \mathbb{P}|X \approx x, p_n \nmid X) = \frac{P(X \in \mathbb{P}|X \approx x) \cap (p_n \nmid X))}{P(p_n \nmid X)} = \frac{P(X \in \mathbb{P}|X \approx x)}{\frac{p_n - 1}{p_n}}$$

$$P(X \in \mathbb{P}|X \approx x, p_n \nmid X) = \frac{p_n}{p_n - 1} P(X \in \mathbb{P}|X \approx x)$$

Such given information is desirable, as it increases the probability of finding a prime number when X is chosen. Let Q represent the factor by which $P(X \in \mathbb{P}|X \approx x)$ increases. Then, the probability that X is prime is approximately $\frac{Q}{\ln(x)-1}$ (Ribenboim, 2004, p. 164). In the above case, $Q = \frac{p_n}{p_n - 1}$. Ideally, Q should be as large as possible. Wheels use multiple conditions to result in large values of Q:

1. $2 \nmid X \Leftrightarrow X \not\equiv 0 \pmod 2$.
2. $3 \nmid X \Leftrightarrow X \not\equiv 0 \pmod 3$.
3. The Chinese remainder theorem states that the conditions can be combined to find X modulo $2 \times 3 = 6$. Assume X can be congruent to anything modulo 6: $X \equiv 0, 1, 2, 3, 4, 5 \pmod 6$ (Wright, 2016, p. 7).
4. $X \not\equiv 0 \pmod 2$, $\therefore X \equiv 1, 3, 5 \pmod 6$.
5. $X \not\equiv 0 \pmod 3$, $\therefore X \equiv 1, 5 \pmod 6$.

$X \equiv 1, 5 \pmod 6$ is an example of a wheel. Selecting X to equal $1 + 6k$ or $5 + 6k$ introduces some information about X, such that $Q = \frac{2}{2-1}\frac{3}{3-1} = 3$. Adding more statements like $5 \nmid p$ can increase Q further. Similarly on the Ulam spiral, a more complicated process may locate lines that contain no points that have been assigned multiples of the first n primes—the prime-rich lines.

2.2 Goals of a model

The primary goal of the model presented in this essay is to predict the locations of prime-rich lines in the Ulam spiral. This is evaluated based on how prime-rich these lines are, compared to a random selection of "average" lines. The greater the difference, the more effectively the model may be used to produce primes. The exploration is also driven by the goal to be as complete as possible in describing the mechanics of the spiral. Many amateur studies are incomplete due to a limited focus, such as on lines of a particular slope. This essay attempts to avoid specific cases as much as possible. Finally, computational complexity is an important theme in the development of algorithms used to find prime numbers. In this exploration, symmetries are studied as a means to minimize computation.

Along with the model, most images of the spiral that appear in this essay are generated with the assistance of original Java code, which may be found in the appendix.

3 Spiral anatomy

3.1 The spiral as a recursive function

f is a function accepting a lattice point ($(x, y) \in \mathbb{Z} \times \mathbb{Z}$) as its parameter, that outputs the number assigned to that point on the Ulam spiral. The recursive definition of f is similar to the way in which a person might construct the spiral.

$$f((x,y)) = \begin{cases} 1, & \text{if } (x,y) = (0,0) \\ f((x,y-1)) + 1, & \text{if } -x+2 \le y \le x \quad \text{in quadrant 1} \\ f((x+1,y)) + 1, & \text{if } -y \le x \le y-1 \quad \text{in quadrant 2} \\ f((x,y+1)) + 1, & \text{if } x \le y \le -x-1 \quad \text{in quadrant 3} \\ f((x-1,y)) + 1, & \text{if } y+1 \le x \le -y+1 \quad \text{in quadrant 4} \end{cases}$$

Four quadrants exist on the spiral. In each quadrant "the next point" is in a different direction relative to the previous point. The variable d represents the quadrant number, an arbitrary value from 1 to 4. In this essay, $d = D$ is used to signify that only quadrant D is being considered.

$$d = \begin{cases} 1, & \text{if } -x+2 \le y \le x \\ 2, & \text{if } -y \le x \le y-1 \\ 3, & \text{if } x \le y \le -x-1 \\ 4, & \text{if } y+1 \le x \le -y+1 \end{cases}$$

In all quadrants, some unit vector describes the displacement to a point from "the point before it". Let $\hat{v}$ represent this vector:

$$\hat{v} = (v_x, v_y) = \begin{cases} (0,1), & \text{if } d = 1 \\ (-1,0), & \text{if } d = 2 \\ (0,-1), & \text{if } d = 3 \\ (1,0), & \text{if } d = 4 \end{cases}$$

By definition, f can be rewritten as:

$$f((x,y)) = \begin{cases} 1, & \text{if } (x,y) = (0,0) \\ f((x,y) - \hat{v}) + 1, & \text{otherwise} \end{cases}$$

The spiral winds around the origin by one "layer" per complete revolution. A separate vector $\hat{u}$ represents the direction between successive layers of the spiral. For example, $(6, 2)$ is one layer beyond $(5, 2)$ in quadrant 1.

$$\hat{u} = (u_x, u_y) = \begin{cases} (1,0), & \text{if } d = 1 \\ (0,1), & \text{if } d = 2 \\ (-1,0), & \text{if } d = 3 \\ (0,-1), & \text{if } d = 4 \end{cases}$$

$a\hat{u} + b\hat{v}$ is a frequent representation of points in this essay. a is the number of layers between the point and the origin, and b represents how far anticlockwise the point is in the rotation of the spiral. a and b must be integers. When writing $a\hat{u} + b\hat{v}$, the location of the point is ambiguous, since the unit vectors depend on the undeclared quadrant number d. Writing $d = D : a\hat{u} + b\hat{v}$, there is no ambiguity, since the unit vectors are defined. Every point has a unique representation in the form $d = D : a\hat{u} + b\hat{v}$.

The generalized spiral The Ulam spiral can be generalized to begin with n at $(0, 0)$ and increment by m.

$$f_{m,n}((x,y)) = \begin{cases} n, & \text{if } (x,y) = (0,0) \\ f_{m,n}((x,y) - \hat{v}) + m, & \text{otherwise} \end{cases}$$

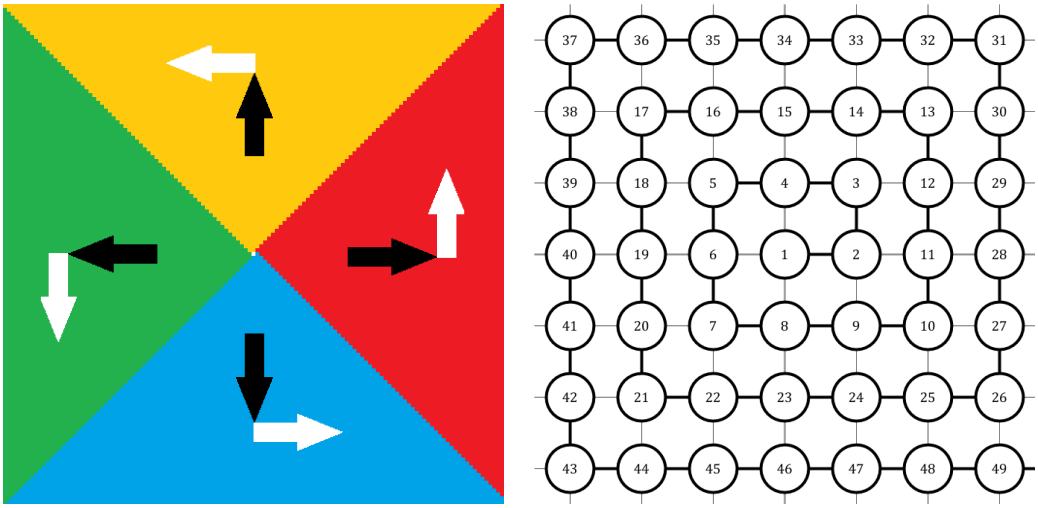


Figure 3: The quadrants: 1) red, 2) yellow, 3) green, 4) blue. Black arrows represent $\hat{u}$, white arrows represent $\hat{v}$. The point assigned 14 is located at $d = 2 : 2\hat{u} + (-1)\hat{v}$. In comparison, $d = 3 : f(2\hat{u} + (-1)\hat{v}) = 18$.

The original Ulam spiral is described by $f_{1,1}$. This complication is not solely for being general. The range of f can be set to any arbitrary arithmetic sequence: $f_{m,n}(P) \equiv n \pmod{m}$. Thus, wheel factorization may be directly implemented into the Ulam spiral. For example, the wheel $p \equiv 1, 5 \pmod{m}$ can be simulated with $f_{6,1}$ and $f_{6,5}$.

3.2 The spiral as an explicit function

Performing recursion is computationally intensive; an explicit function is more desirable. Examining the relationship between a and $f_{m,n}(a\hat{u})$, quadratic relationships are found.

a	$d = 1$	$d = 2$	$d = 3$	$d = 4$
1	$1m + n$	$3m + n$	$5m + n$	$7m + n$
2	$10m + n$	$14m + n$	$18m + n$	$22m + n$
3	$27m + n$	$33m + n$	$39m + n$	$45m + n$
4	$52m + n$	$60m + n$	$68m + n$	$76m + n$
5	$85m + n$	$95m + n$	$105m + n$	$115m + n$
General formula	$(4a^2 - 3a)m + n$ $(4a^2 - 5a + 2(1)a)m + n$	$(4a^2 - a)m + n$ $(4a^2 - 5a + 2(2)a)m + n$	$(4a^2 + a)m + n$ $(4a^2 - 5a + 2(3)a)m + n$	$(4a^2 + 3a)m + n$ $(4a^2 - 5a + 2(4)a)m + n$

It is concluded from inspection that $f_{m,n}(a\hat{u}) = (4a^2 - 5a + 2da)m + n$. This can be substituted into the recursive formula for the generalized Ulam spiral.

$$f_{m,n}(a\hat{u} + b\hat{v}) = \begin{cases} (4a^2 - 5a + 2da)m + n, & \text{if } b = 0 \\ f_{m,n}(a\hat{u} + b\hat{v} - \hat{v}) + m, & \text{otherwise} \end{cases}$$

Thus, $f_{m,n}(a\hat{u} + 1\hat{v}) = (4a^2 - 5a + 2da)m + m + n$, which then allows $f_{m,n}(a\hat{u} + 2\hat{v})$ to be defined, and so on. By inductive reasoning, the explicit function of the generalized Ulam spiral is obtained:

$$f_{m,n}(a\hat{u} + b\hat{v}) = (4a^2 - 5a + 2da)m + bm + n = (4a^2 - 5a + 2da + b)m + n$$

4 Divisibility plots

The famous image of the Ulam spiral is a graph of $f_{m,n}(P) \in \mathbb{P}$. This is not the only possible representation. In this section, the graph of $p \mid f_{m,n}(P)$ is explored in depth, called a "divisibility plot" modulo p. No specific value of p is chosen, but it is understood that p is some odd prime number, treated as a constant.

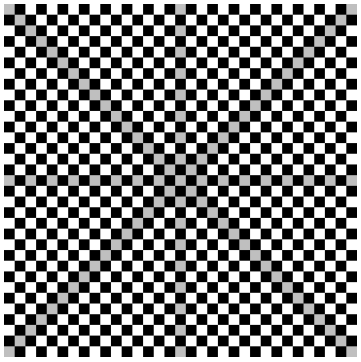


Figure 4: The plot of $2 \mid f_{1,1}(P)$ shows why the prominent prime-rich lines on the Ulam spiral are primarily diagonal. Only a diagonal line may contain no multiples of 2.

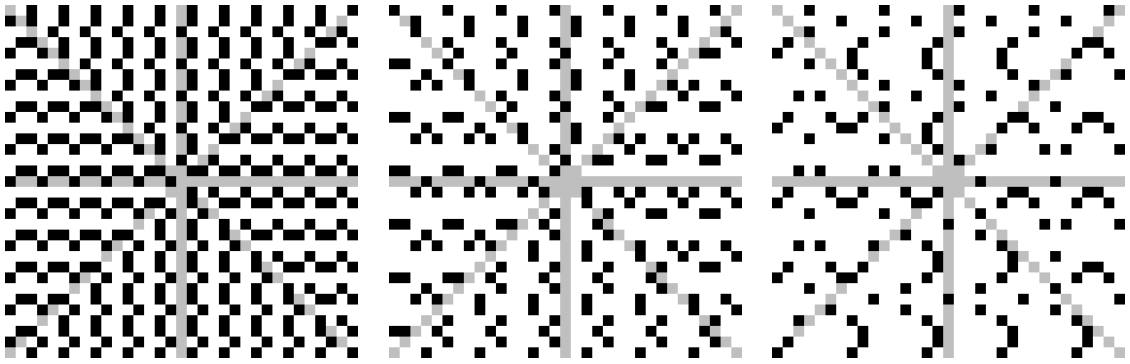


Figure 5: Various divisibility plots.

4.1 Exceptions

The mathematics presented in the following sections assume $p \nmid m$. Recall that $f_{m,n}(P) \equiv n \pmod{m}$. If $p \mid m$, then $f_{m,n}(P) \equiv n \pmod{p}$. If $p \mid n$, then every point is divisible modulo p, otherwise $p \nmid n$ and no point is divisible.

4.2 Parabolas

Divisibility plots are named so because every point where $p \mid f_{m,n}(P)$ is true has been assigned a multiple of p. These points are "divisible points". If $p \nmid f_{m,n}(P)$, P is an "indivisible point". Divisible points can be located explicitly:

$$p \mid f_{m,n}(a\hat{u} + b\hat{v}) \Leftrightarrow f_{m,n}(a\hat{u} + b\hat{v}) \equiv (4a^2 - 5a + 2da + b)m + n \equiv 0 \pmod{p}$$

$$(4a^2 - 5a + 2da + b)m \equiv -n \pmod{p}$$

$$4a^2 - 5a + 2da + b \equiv -m_p^{-1} n \pmod{p}$$

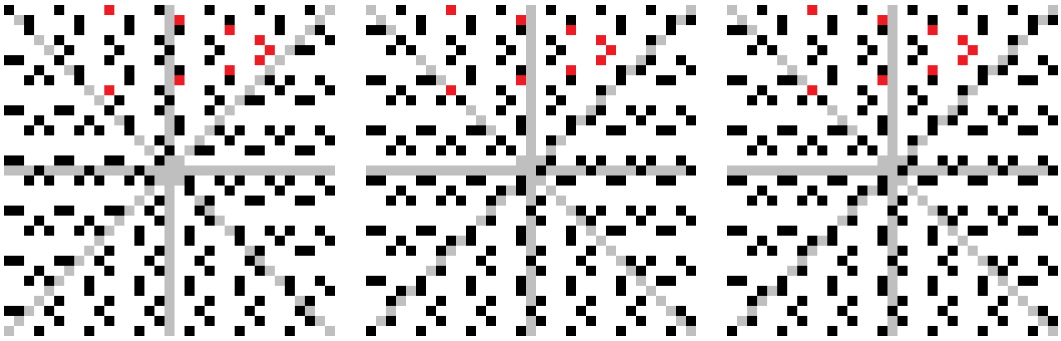
$$b \equiv -4a^2 + 5a - 2da - m_p^{-1} n \pmod{p}$$

$$b = (-4 + k_2 p)a^2 + (5 - 2d + k_1 p)a + (-m_p^{-1} n + k_0 p),\ k_0, k_1, k_2 \in \mathbb{Z}$$

Thus, if $a\hat{u} + b\hat{v}$ is a divisible point, then b is equal to a function g_{k_0,k_1,k_2} of a:

$$b = g_{k_0,k_1,k_2}(a) = (-4 + k_2 p)a^2 + (5 - 2d + k_1 p)a + (-m_p^{-1} n + k_0 p)$$

The notation m_p^{-1} is used throughout this essay to signify the least positive modular multiplicative inverse of m modulo p. It is the solution to $m_p^{-1}m \equiv 1 \pmod{p}$, $0 \leq m_p^{-1} < p$. m_p^{-1} does not exist if m and p share a common factor. As k_0, k_1, k_2, and a vary, every possible solution for b in $f_{m,n}(a\hat{u} + b\hat{v}) \equiv 0 \pmod{p}$ is obtained. That is, $a\hat{u} + g_{k_0,k_1,k_2}(a)\hat{v}$ describes every divisible point. Noting that $g_{k_0,k_1,k_2}(a)$ is a quadratic expression, $t\hat{u} + g_{k_0,k_1,k_2}(t)\hat{v}$, $t \in \mathbb{Z}$ is a parabola consisting solely of divisible points. Varying k_0, k_1, k_2 affects the behaviour of the parabola.



$f_{1,1}$ $f_{3,7}$ $f_{8,2}$

Figure 6: Various divisibility plots modulo 5. Example parabolas are highlighted in red.

4.3 Symmetries

Observations are made of three symmetries on the divisibility plot. They are transformations applied to a point, which guarantee that the transformed point is divisible, if the original is. The point resulting from the transformation is thus said to be "redundant" to the original. Redundancy is an equivalence relation on a divisibility plot:

- Let $A \rightleftharpoons B$ signify that B is divisible if A is. That is, B is redundant to A. In terms of points, this means that a transformation following one of the three symmetries can move A to the location of B.
- Reflexivity: $A \rightleftharpoons A$.
- Symmetry: $A \rightleftharpoons B \Rightarrow B \rightleftharpoons A$.
- Transitivity: $(A \rightleftharpoons B \wedge B \rightleftharpoons C) \Rightarrow A \rightleftharpoons C$.

4.4 Translation symmetry

Tiling conjecture $P = a\hat{u}+b\hat{v}$ and $P' = P+\Delta a\hat{u}+\Delta b\hat{v}$ are two points within the same quadrant. When Δa and Δb are multiples of p, P' must be a divisible point if P is; $P \rightleftharpoons P'$. Formally:

$$P \rightleftharpoons P + k_a p\hat{u} + k_b p\hat{v} \ , \ k_a, k_b \in \mathbb{Z}$$

Proof Assume that $P = a\hat{u} + b\hat{v}$ is a divisible point. That is,

$$f_{m,n}(P) = f_{m,n}(a\hat{u} + b\hat{v}) = (4a^2 - 5a + 2da + b)m + n \equiv 0 \pmod{p}$$

P' is a divisible point if $f_{m,n}(P') \equiv 0 \mod p$.

$$P' = P + \Delta a\hat{u} + \Delta b\hat{v} = (a + \Delta a)\hat{u} + (b + \Delta b)\hat{v}$$

When Δa and Δb are multiples of p, they can be written as $k_a p$ and $k_b p$ respectively ($k_a, k_b \in \mathbb{Z}$).

$$P' = (a + k_a p)\hat{u} + (b + k_b p)\hat{v}$$

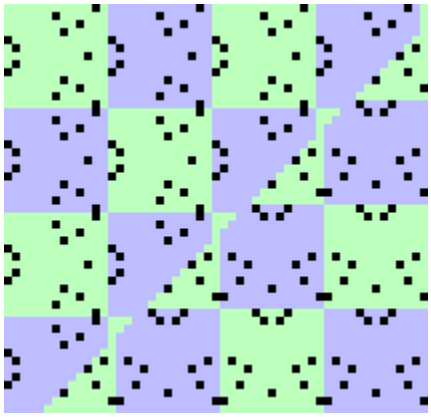


Figure 7: The boundary between quadrants 1 and 2 in the divisibility plot modulo 13. In each quadrant, a pattern of divisible points repeats in both dimensions as tessellating 13×13 square "tiles". This observation leads to the tiling conjecture.

$$f_{m,n}(P') = f_{m,n}((a + k_a p)\hat{u} + (b + k_b p)\hat{v})$$
$$f_{m,n}(P') = (4(a + k_a p)^2 - 5(a + k_a p) + 2d(a + k_a p) + (b + k_b p))m + n$$
$$f_{m,n}(P') = (4a^2 - 5a + 2da + b)m + (8ak_a + k_a^2 p - 5k_a + 2dk_a + k_b)pm + n$$
$$f_{m,n}(P') \equiv (4a^2 - 5a + 2da + b)m + n \pmod{p}$$
$$\text{It is known that } (4a^2 - 5a + 2da + b)m + n \equiv 0 \pmod{p}$$
$$\therefore f_{m,n}(P') \equiv 0 \pmod{p}$$

Therefore P' is a divisible point and the conjecture is proven.

4.5 Reflection symmetry

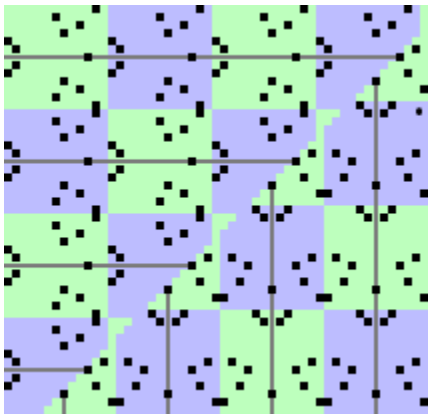


Figure 8: The same plot from Figure 7, with lines of reflection drawn. The lines are parallel to $\hat{v}$, separated by distances of p. This observation leads to the reflection conjecture.

Reflection conjecture Lines of reflection can be described in vector form by $A\hat{u} + t\hat{v}$, $t \in \mathbb{Z}$, or $a = A$. $P = (A + \Delta a)\hat{u} + b\hat{v}$ and $P' = A - \Delta a)\hat{u} + b\hat{v}$ are two points in the same quadrant. Δa is the distance from P and P' to $a = A$. If P is a divisible point, then P' is as well.

$$(A + \Delta a)\hat{u} + b\hat{v} \rightleftharpoons (A - \Delta a)\hat{u} + b\hat{v}$$

Proof Every divisible point P is part of a parabola $a\hat{u} + g_{k_0,k_1,k_2}(a)\hat{v}$. Every parabola has an axis of reflection symmetry; P can be reflected across this axis to the location P'. Since P' is part of the parabola, it must be divisible. Thus, the axis of reflection of the parabola is the line of reflection described by the conjecture.

The axis of reflection of any parabola passes through its vertex, where the rate of change of g_{k_0,k_1,k_2} is zero:

$$\therefore g'_{k_0,k_1,k_2}(A) = 2(-4 + k_2p)A + (5 - 2d + k_1p) = 0$$
$$-8A + 2k_2pA + 5 - 2d + k_1p = 0$$
$$8A \equiv 5 - 2d \pmod{p}$$
$$A \equiv 8^{-1}(5 - 2d) \pmod{p}$$
$$A = 8_p^{-1}(5 - 2d) + kp\ ,\ k \in \mathbb{Z}$$

Infinitely many lines of reflection exist separated by distances of p, as suggested by the observations.

Reflection is generally unused in this essay due to the relative difficulty of implementing it into the model. However, the importance of these findings is that the lines of reflection always pass through the vertices of the parabola $a\hat{u} + g_{k_0,k_1,k_2}(a)\hat{v}$, at the points $A\hat{u} + B\hat{v}$, where B represents $g_{k_0,k_1,k_2}(A)$.

$$B = g_{k_0,k_1,k_2}(8_p^{-1}(5 - 2d) + kp)$$
$$B = (-4 + k_2p)(8_p^{-1}(5 - 2d) + kp)^2 + (5 - 2d + k_1p)(8_p^{-1}(5 - 2d) + kp) + (-m_p^{-1}n + k_0p)$$
$$B \equiv (-4)(8_p^{-1}(5 - 2d))^2 + (5 - 2d)(8_p^{-1}(5 - 2d)) + (-m_p^{-1}n) \pmod{p}$$
$$B \equiv (-4)(8_p^{-1})^2(5 - 2d)^2 + 8_p^{-1}(5 - 2d)^2 + (-m_p^{-1}n) \pmod{p}$$
$$B \equiv 8_p^{-1}(1 - 8_p^{-1}4)(5 - 2d)^2 - m_p^{-1}n \pmod{p}$$

This information is used to describe rotation symmetry. The location of the vertex depends on the quadrant, since d is present in the expressions for A and B. Hence, let $A_D \equiv 8_p^{-1}(5 - 2D) + k_Dp\ ,\ k_D \in \mathbb{Z}$, and $B_D \equiv 8_p^{-1}\left(\frac{1-p}{2}\right)(5-2D)^2 - m_p^{-1}n \mod p$. $A_D\hat{u} + B_D\hat{v}$ represents the locations of the vertex in quadrant D.

4.6 Rotation symmetry

The patterns of divisible points in the quadrants are mutually similar. In the previous proofs, the assumption *let d be constant* is successfully applied, suggesting that pattern-generating mechanisms operate similarly everywhere. A rotation from quadrant D_1 to D_2 can be described as taking the point $d = D_1 : a\hat{u} + b\hat{v}$ and transforming it to $d = D_2 : a\hat{u} + b\hat{v}$. The change in d serves to redefine the directions of $\hat{u}$ and $\hat{v}$. Doing so for every point in a quadrant, the points are rotated together, bringing the pattern of divisible points with them. However, changing d has an additional effect on divisible points:

$$\text{Original: } \mathbf{d = D_1} : a\hat{u} + g_{k_0,k_1,k_2}(a)\hat{v} = a\hat{u} + ((-4 + k_2p)a^2 + (5 - 2\mathbf{D_1} + k_1p)a + (-m_p^{-1}n + k_0p))\hat{v}$$

$$\text{Rotation: } \mathbf{d = D_2} : a\hat{u} + g_{k_0,k_1,k_2}(a)\hat{v} = a\hat{u} + ((-4 + k_2p)a^2 + (5 - 2\mathbf{D_1} + k_1p)a + (-m_p^{-1}n + k_0p))\hat{v}$$

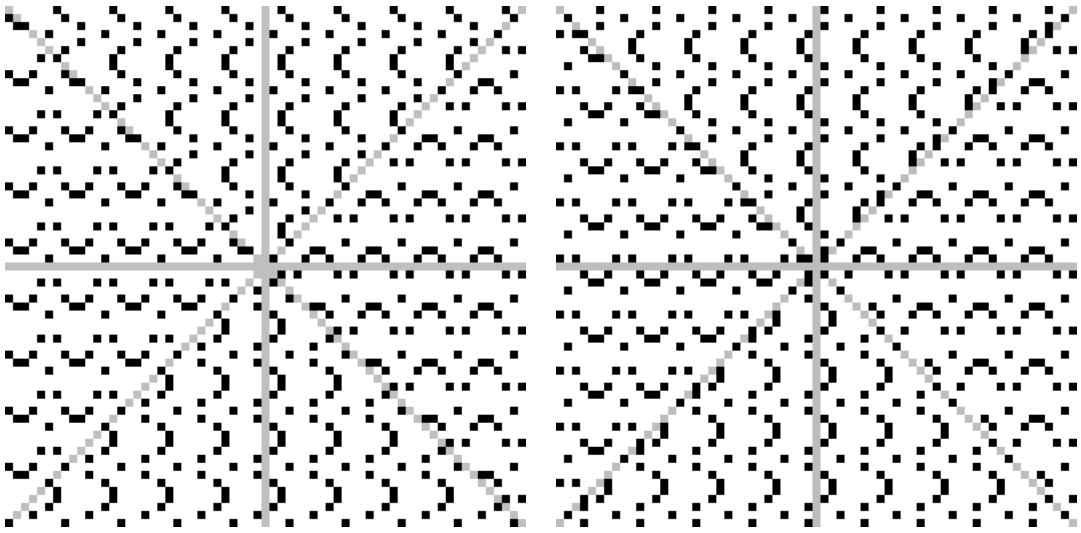
$$\text{Actual: } \mathbf{d = D_2} : a\hat{u} + g_{k_0,k_1,k_2}(a)\hat{v} = a\hat{u} + ((-4 + k_2p)a^2 + (5 - 2\mathbf{D_2} + k_1p)a + (-m_p^{-1}n + k_0p))\hat{v}$$

The coefficient $(5-2d+k_1p)$ of the term with degree 1 changes. On the graph of any quadratic relationship, this results in a translation of the parabola. This signifies that the similarity between two quadrants relies not only on a rotation, but a translation applied afterwards. The location of a parabola can be determined by its vertex, $A\hat{u} + B\hat{v}$. A translation of the vertex is also a translation of all divisible points on the parabola.

$$\text{Original: } d = D_1 : A_{D_1}\hat{u} + B_{D_1}\hat{v}$$

$$\text{Rotated: } d = D_2 : A_{D_1}\hat{u} + B_{D_1}\hat{v}$$

$$\text{Actual: } d = D_2 : A_{D_2}\hat{u} + B_{D_2}\hat{v}$$



A) Actual B) Rotation only (Incorrect)

Figure 9: In B, divisible points from quadrant 1 are rotated about the origin—no translation added, resulting in subtle differences between quadrants 2, 3, and 4 of the two plots.

The discrepancy between the two is the translation $\Delta A_{D_1 \to D_2}\hat{u} + \Delta B_{D_1 \to D_2}\hat{v}$ that is needed to move the rotated vertex to its expected location.

$$\Delta A_{D_1 \to D_2} \equiv A_{D_2} - A_{D_1} \pmod{p}$$

$$\Delta B_{D_1 \to D_2} \equiv B_{D_2} - B_{D_1} \pmod{p}$$

The translation only needs to be calculated once, then rotation symmetry can be expressed for any point:

$$d = D_1 : a\hat{u} + b\hat{v} \rightleftharpoons d = D_2 : (a + \Delta A_{D_1 \to D_2})\hat{u} + (b + \Delta B_{D_1 \to D_2})\hat{v}$$

4.7 Testing symmetries

To confirm that the symmetries and their consequences are accurately understood, divisibility plots are constructed following two distinct methods. The first method uses the recursive definition of f and tests each point's divisibility. The second method finds some divisible points, and transforms those points to fill the plane. In Figure 10, the plots are identical by visual comparison, validating the symmetries.

5 Lines

Every line L on the Ulam spiral can be expressed in vector form, starting at some initial point $a_0\hat{u} + b_0\hat{v}$, and extending along a direction vector $\vec{\omega} = \omega_a\hat{u} + \omega_b\hat{v}$, $\omega_a, \omega_b \in \mathbb{Z}$. Lines are assumed to be confined to a single quadrant.

$$\textbf{Vector form of } L : d = D : (a_0\hat{u} + b_0\hat{v}) + t(\omega_a\hat{u} + \omega_b\hat{v}),\ t \in \mathbb{Z}$$

$$\textbf{Parametric form of } L : d = D : \begin{cases} a = a_0 + t\omega_a \\ b = b_0 + t\omega_b \end{cases} = (a_0 + t\omega_a)\hat{u} + (b_0 + t\omega_b)\hat{v},\ t \in \mathbb{Z}$$

A "divisible line" modulo p contains at least one divisible point; an "indivisible line" contains none. Indivisible lines are desirable in the model, as they are more likely to contain prime points. *Which lines are indivisible modulo p?* The question appears difficult to answer, since there are infinitely many possible lines.

When a transformation from the previous section is applied to the line L, it is applied to every point in L. When a divisible point is transformed, its new location is also divisible; when an indivisible point is

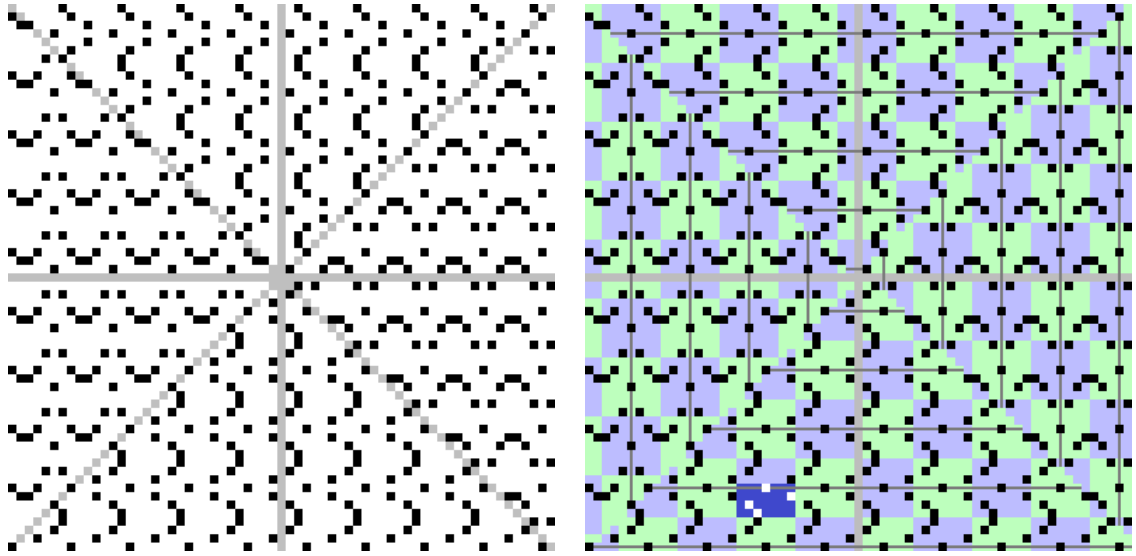
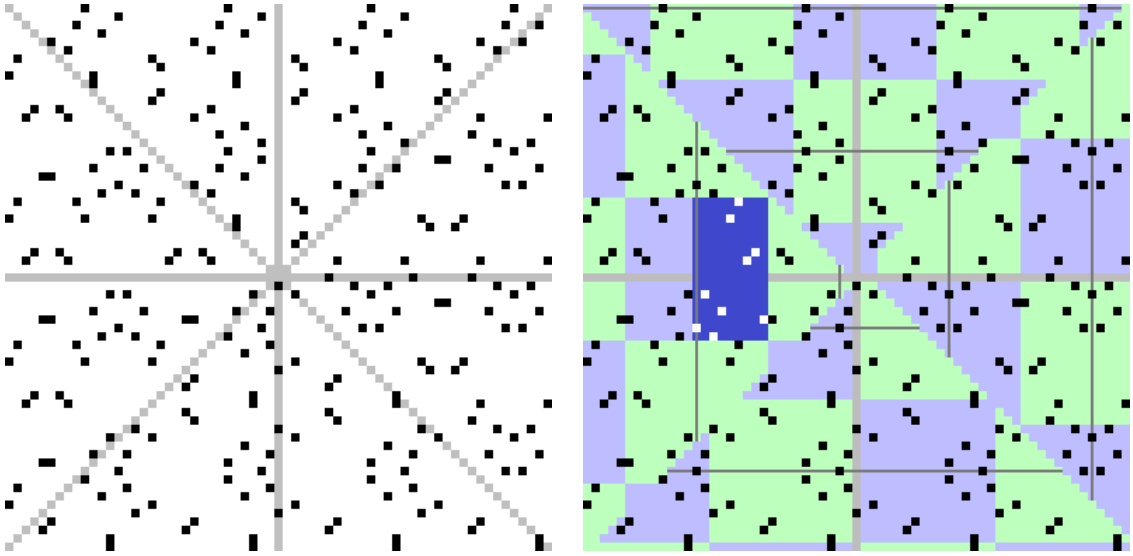


Figure 10: Above, $7 \mid f_{5,4}(P)$. Below, $17 \mid f_{2,3}(P)$. The plots on the right are generated by the half-tiles highlighted on the plot.



transformed, its new location is indivisible. The transformed line L' is composed only of these transformed points. Thus, L' must have the same divisibility as L. In this way, L and L' are redundant, and knowing the divisibility of L automatically grants knowledge about the divisibility of L'.

$$L \rightleftharpoons L'$$

In the following sections, symmetry is used to show how every line is redundant to a simpler line expressible in the form $d = D : (sb + c)\hat{u} + b\hat{v}\ ,\ s, c \in \mathbb{Z}\ ,\ 0 \leq s, c < p$. One may find the divisibility of each line in the simplified form, then use symmetry to determine the divisibility of every possible line.

5.1 Translational symmetry

Recall the tiling conjecture: $P \rightleftharpoons P + k_a p\hat{u} + k_b p\hat{v}\ ,\ k_a, k_b \in \mathbb{Z}$.

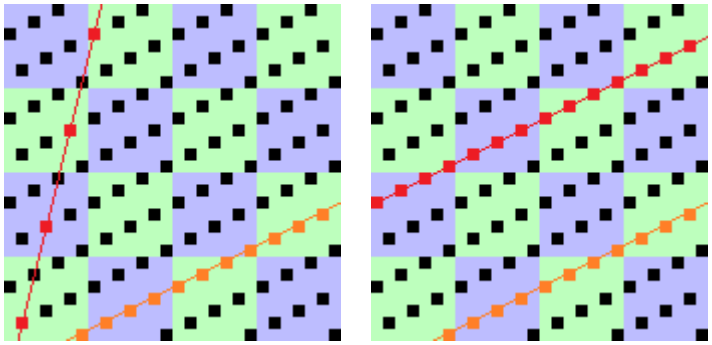


Figure 11: In each divisibility plot modulo 7, the red line is redundant to the orange line by translational symmetry.

$$L : d = D : \begin{cases} a = a_0 + t\omega_a \\ b = b_0 + t\omega_b \end{cases} \rightleftharpoons L' : d = D : \begin{cases} a = a_0 + t\omega_a + k_a p = (a_0 + k_a p) + t\omega_a \\ b = b_0 + t\omega_b + k_b p = (b_0 + k_b p) + t\omega_b \end{cases}$$

$$L : d = D : \begin{cases} a = a_0 + t\omega_a \\ b = b_0 + t\omega_b \end{cases} \rightleftharpoons L' : d = D : \begin{cases} a = a_0 + t\omega_a + tk_a p = a_0 + t(\omega_a + k_a p) \\ b = b_0 + t\omega_b + tk_b p = b_0 + t(\omega_b + k_b p) \end{cases}$$

As shown, adding multiples of p to any of $a_0, b_0, \omega_a, \omega_b$ results in a line redundant to the original.

Translational symmetry principle of lines A If $a_0 \equiv a_0'$, $b_0 \equiv b_0'$, $\omega_a \equiv \omega_a'$, $\omega_b \equiv \omega_b' \pmod{p}$, then

$$L : d = D : \begin{cases} a = a_0 + t\omega_a \\ b = b_0 + t\omega_b \end{cases} \rightleftharpoons L' : d = D : \begin{cases} a = a_0' + t\omega_a' \\ b = b_0' + t\omega_b' \end{cases}$$

Thus, every line is redundant to some line where $0 \leq a_0, b_0, \omega_a, \omega_b < p$.

Lines where $\omega_b \equiv 0 \pmod{p}$ make up a small portion of all lines on the plot. Ideally, these lines may be studied further, but in this essay they are assumed unnoteworthy and ignored for brevity.

Lines where $\omega_b \not\equiv 0 \pmod{p}$ can be simplified by the substitution $t = t'\omega_{b_p}^{-1}$.

$$L : d = D : \begin{cases} a = a_0 + t\omega_a = a_0 + t' b_p^{-1}\omega_a = a_0 + t'(\omega_{b_p}^{-1}\omega_a) \\ b = b_0 + t\omega_b = b_0 + t' b_p^{-1}\omega_b = b_0 + t'(\omega_{b_p}^{-1}\omega_b) \end{cases}$$

Consider $L' : d = D : \begin{cases} a = a_0 + t\omega_b^{-1}{}_p\omega_a \\ b = b_0 + t \end{cases}$. This line is redundant to L if:

$$a_0 \equiv a_0 \ , \ b_0 \equiv b_0 \ , \ \omega_{b_p}^{-1}\omega_a \equiv \omega_{b_p}^{-1}\omega_a \ , \ \omega_{b_p}^{-1}\omega_b \equiv 1 \pmod{p}$$

All of these statements are true, thus every line where $\omega_b \not\equiv 0 \pmod{p}$ can be converted to a line where $\omega_b = 1$.

When $\omega_b = 1 \pmod{p}$, the substitution $b = b_0 + t \rightarrow t = b - b_0$ can be used:

$$L : d = D : \begin{cases} a = a_0 + t\omega_a \rightarrow a = a_0 + (b - b_0)\omega_a = (a_0 - b_0\omega_a) + b\omega_a \\ b = b_0 + t \rightarrow b = b_0 + (b - b_0) = b \end{cases}$$

Let s be the slope of the line with respect to $\hat{v}$: $s = \frac{\omega_a}{\omega_b} = \frac{\omega_a}{(1)} = \omega_a$. c is the a-intercept of the line (when $b = 0$): $c = \omega_a(0) + a_0 = a_0$. The line can be written in the form $L : d = D : a = sb + c$.

Translational symmetry principle of lines B The translational symmetry principle of lines can be adapted to lines of this form. If $s \equiv s'$, $c \equiv c' \pmod{p}$, then

$$L : d = D : a = sb + c \rightleftharpoons L' : d = D : a = s'b + c'$$

Thus, every line is redundant to some line where $0 \leq c, s < p$.

5.2 Rotational symmetry

In rotational symmetry,

$$\Delta A_{D_1 \to D_2} = A_{D_2} - A_{D_1}$$
$$\Delta B_{D_1 \to D_2} = B_{D_2} - B_{D_1}$$
$$d = D_1 : a\hat{u} + b\hat{v} \rightleftharpoons d = D_2 : (b + \Delta A_{D_1 \to D_2})\hat{u} + (b + \Delta B_{D_1 \to D_2})\hat{v}$$

This can be applied to lines.

$$L : d = D_1 : a = sb + c \rightleftharpoons L' : d = D_2 : (a - \Delta A_{D_1 \to D_2}) = s(b - \Delta B_{D_1 \to D_2}) + c$$

$$L' : d = D_2 : a = sb + (c - s\Delta B_{D_1 \to D_2} + \Delta A_{D_1 \to D_2})$$

Letting $D_2 = 1$, every line is redundant to some line in quadrant 1. In conclusion, every line may be converted to the form $d = 1 : a = sb + c$, $0 \leq c, s < p$. In total, there are p^2 unique lines.

5.3 Divisible lines modulo 2

First, this special case is considered, to demonstrate the process of finding an indivisible line.

$L : d = D : a = sb + c$ is divisible if there exists some point on L that is divisible—there is some value of b for which the point $d = D : (sb + c)\hat{u} + b\hat{v}$ on the line is divisible. Whether b exists may be determined by attempting to construct it. Begin by assuming that $d = D : (sb + c)\hat{u} + b_0\hat{v}$ is a divisible point.

$$d = D : 2 \mid f_{m,n}((sb + c)\hat{u} + b\hat{v})$$

$$(4(sb + c)^2 - 5(sb + c) + 2D(sb + c) + b)m + n \equiv 0 \pmod{2}$$
$$bm(s + 1) + cm + n \equiv 0 \pmod{2}$$

If m is even, then $n \equiv 0 \pmod{2} \Rightarrow 2 \mid n$. This is always false (has no solution) if n is odd. When m is odd:

$$b(s + 1) \equiv c + n \mod 2$$

$$b \equiv \begin{cases} (s+1)_2^{-1}(c+n) & \text{if } 2 \nmid s+1 \Rightarrow 2 \mid s \\ \text{any value} & \text{if } 2 \mid s+1, 2 \mid c+n \\ \text{no solution} & \text{if } 2 \mid s+1, 2 \nmid c+n \Rightarrow 2 \nmid s, 2 \nmid c+n \end{cases} \pmod{2}$$

Therefore, L is indivisible if m is even but n is odd, or if s and $c + n$ are odd.

5.4 Divisibility of a line

Just as in the previous section, to determine if $L : d = 1 : a = sb + c$ is divisible, begin by assuming that $d = 1 : (sb + c)\hat{u} + b\hat{v}$ is a divisible point.

$$d = 1 : f_{m,n}((sb + c)\hat{u} + b\hat{v}) \equiv 0 \pmod{p}$$

$$(4(sb + c)^2 - 5(sb + c) + 2(1)(sb + c) + b)m + n \equiv 0 \pmod{p}$$
$$4s^2b^2 + (8sc - 3s + 1)b + 4c^2 - 3c + m_p^{-1}n \equiv 0 \pmod{p}$$

Let $4s^2 = \alpha$, $8sc - 3s + 1 = \beta$, and $4c^2 - 3c + m_p^{-1}n = \gamma$. $\alpha, \beta, \gamma \in \mathbb{Z}$.

$$\alpha b^2 + \beta b + \gamma b \equiv 0 \pmod{p}$$

$$4\alpha^2 b^2 + 4\alpha\beta b + 4\alpha\gamma \equiv 0 \pmod{p}$$
$$(2\alpha b + \beta)^2 - \beta^2 + 4\alpha\gamma \equiv 0 \pmod{p}$$
$$(2\alpha b + \beta)^2 \equiv \beta^2 - 4\alpha\gamma \pmod{p}$$

If $\beta^2 - 4\alpha\gamma$ is a quadratic nonresidue modulo p, then there is no value of r that can satisfy $r^2 \equiv \beta^2 - 4\alpha\gamma \pmod{p}$ (Wright, 2016). Thus, there is no solution for $2\alpha b + \beta$, and no value of b for which the assumption "$d = 1 : (sb + c)\hat{u} + b\hat{v}$ is a divisible point" is true. In this case, L is indivisible.

To manipulate $\beta^2 - 4\alpha\gamma$ so that it is a quadratic nonresidue, quadratic nonresidues modulo p must first be found. This is done quickly through brute force. Numbers from 0 to $p - 1$ are listed. Every number that is congruent to a perfect square modulo p is a quadratic residue, and is removed. The $\frac{p-1}{2}$ values that remain in the list are quadratic nonresidues modulo p. Next, for each value q from the list, one considers the congruence:

$$q \equiv \beta^2 - 4\alpha\gamma \pmod{p}$$
$$q \equiv (8sc - 3s + 1)^2 - 4(4s^2)(4c^2 - 3c + m_p^{-1}n) \pmod{p}$$
$$q \equiv (64s^2c^2 - 48s^2c + 16sc + 9s^2 - 6s + 1) - (64s^2c^2 - 48s^2c + 16s^2m_p^{-1}n) \pmod{p}$$
$$q \equiv 9s^2 - 16s^2m_p^{-1}n + 16sc - 6s + 1 \pmod{p}$$
$$-16sc \equiv 9s^2 - 16s^2m_p^{-1}n - 6s + 1 - q \pmod{p}$$
$$c \equiv -16_p^{-1}9s + sm_p^{-1}n + 8_p^{-1}3 + (q-1)16_p^{-1}s_p^{-1} \pmod{p}$$

This suggests that there are many ways to manipulate a line so that it is indivisible. For every value of s except $s \equiv 0 \pmod{p}$, there is a possible solution for c. In translational symmetry, it is shown that only $0 \leq s < p$ needs to be studied to know the divisibility of every line. Thus, s may be varied from 1 to $p - 1$, evaluating c in each case. This is a finite process, involving $\frac{p-1}{2}$ possible values for q, and $p - 1$ possible values for p, a total of $\frac{(p-1)^2}{2}$ times the congruence needs to be evaluated to determine every indivisible line in the divisibility plot modulo p.

$$c \equiv -16_7^{-1}9s + s(2)_7^{-1}(1) + 8_7^{-1}3 + ((3) - 1)16_7^{-1}s_7^{-1} \pmod{7}$$
$$c \equiv -(4)9s + s(4)(1) + (1)3 + (2)(4)s_7^{-1} \equiv 3s + 3 + s_7^{-1} \pmod{7}$$

Colour	s	c
red	1	0
orange	2	6
yellow	3	3
green	4	3
blue	5	0
violet	6	6

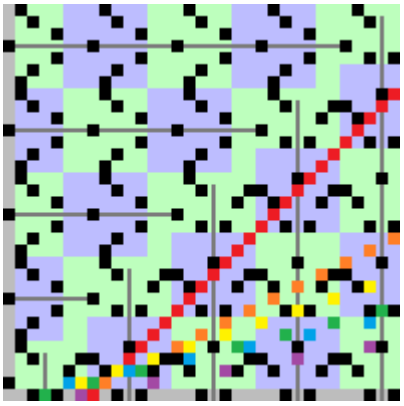


Figure 12: When searching for indivisible lines on the divisibility plot of $f_{2,1}$ modulo 7, it is found that 3 is a quadratic nonresidue modulo 7. Each resulting indivisible line $d = 1 : a = sb + c$ is shown.

6 Finding prime-rich lines

After the procedure in section 5, the model possesses a vast collection of statements: $L : d = 1 : a = sb + c$ is indivisible modulo p. This can be expressed by the notation $L : d = 1 : a = sb + c \ (p)$.

Statements can be combined. Choose k statements, each about a different divisibility plot:

$$L_1 : d = 1 : a = s_1 b + c_1 \ (p_1)$$

$$L_2 : d = 1 : a = s_2 b + c_2 \ (p_2)$$

$$\dots$$

$$L_k : d = 1 : a = s_k b + c_k \ (p_k)$$

The goal of processing this information is to find a line L' that is redundant to every line presented in the statements. L' would be indivisible modulo every prime from p_1 to p_k. As explained in section 2, L' would be prime-rich. Successfully predicting its existence answers the research question affirmatively. Specifically,

$$Q = \frac{p_1}{p_1 - 1} \times \frac{p_2}{p_2 - 1} \times \ldots \times \frac{p_k}{p_k - 1} = \prod_{i=1}^{k} \frac{p_i}{p_i - 1}$$

1. As with every line, the target line L' is expressible in the form $L' : d = D : (a'_0 + t\omega'_a)\hat{u} + (b'_0 + t\omega'_b)\hat{v}$.

2. Suppose one wishes to find L' in quadrant D. Every statement $L_k : d = 1 : a = s_k b + c_k \ (p_k)$ is converted into $L_k : d = D : a = s_k b + (c_k - s_k \Delta B_{1\to D} + \Delta A_{1\to D}) \ (p_k)$, following rotational symmetry.

3. The line $L_k : d = D : a = s_k b + c_k$ can be converted into a vector form, by reversing the substitutions described in section 5.1. In this way, every statement returns to one in the form

$$L_k : d = D : (a_{0_k} + t\omega_{a_k})\hat{u} + (b_{0_k} + t\omega_{b_k})\hat{v} \ (p_k)$$

4. From the translational symmetry principle of lines, it is known that L' is redundant to L_k if

$$a_{0_k} \equiv a'_0 \ , \ b_{0_k} \equiv b'_0 \ , \ \omega_{a_k} \equiv \omega'_a \ , \ \omega_{b_k} \equiv \omega'_b \pmod{p_k}$$

   Therefore,

$$a_{0_1} \equiv a'_0 \pmod{p_1}$$
$$a_{0_2} \equiv a'_0 \pmod{p_2}$$
$$\dots$$
$$a_{0_k} \equiv a'_0 \pmod{p_k}$$

   And similarly for b_0, ω_a, ω_b.

5. For each of $a'_0, b'_0, \omega'_a, \omega'_b$, the system of congruences may be solved by implementing the Chinese remainder theorem. In fact, a solution must exist, therefore the statements may always be combined to find L'.

Thus, in the model, arbitrarily many statements can be combined to predict the location of a line that is indivisible by an arbitrary number of primes. Considering the number of statements available for each p, the number of solutions to L' is so vast that it would be impractical to use a model that finds them all.

6.1 Algorithm

The model generates a random solution for L' in the plot of $f_{m,n}$ which does not contain multiples of the first k primes.

1. Generate a list of the first k primes. For each prime p_i:

2. If $p_i \mid m$, then if $p_i \mid n$, the plot will contain no prime-rich lines, and the algorithm may abort. If $p_i \nmid n$, the algorithm may continue, and this value of p_i may be skipped, since the plot of $f_{m,n}$ contains no divisible points modulo p_i.

3. For $D = 2, 3, 4$, $\Delta A_{1\to D}$ and $\Delta B_{1\to D}$ are calculated.

4. The guess-and-check method determines all quadratic nonresidues q modulo p_i.

5. For each possible q, for each value s from 1 to $p-1$, $c \equiv -16_{p_i}^{-1}9s+sm_{p_i}^{-1}n+8_{p_i}^{-1}3+(q-1)16_{p_i}^{-1}s_{p_i}^{-1} \pmod{p_i}$ is evaluated and stored in a list of statements about p_i.

6. One statement is selected randomly from the list of statements about p_i.

7. D is selected at random, and all statements are converted accordingly.

8. For each statement, b_0 is chosen between 0 and p_i-1, and ω_b is chosen between 1 and p_i-1. $d = D : a = sb+c$ is rewritten as $d = D : \begin{cases} a = st' + c \\ b = t' \end{cases}$, where $t' = b_0+t\omega_b$. Then $a = s(b_0+\omega_b t)+c = sb_0+s\omega_b t+c$, thus $a_0 = sb_0 + c$ and $\omega_a = s\omega_b$.

9. The Chinese remainder theorem is used to solve for $a'_0, b'_0, \omega'_a, \omega'_b$.

6.2 Results

$k = 3$ $Q = 3.75$	$k = 4$ $Q = 4.375$	$k = 5$ $Q = 4.8125$
$d = 3 : (0 + 23t)\hat{u} + (-14 + -7t)\hat{v}$	$d = 1 : (183 + 11t)\hat{u} + (49 + 101t)\hat{v}$	$d = 3 : (487 + 851t)\hat{u} + (1109 + -293t)\hat{v}$
$d = 3 : (28 + 23t)\hat{u} + (14 + -7t)\hat{v}$	$d = 4 : (119 + 71t)\hat{u} + (-3 + 89t)\hat{v}$	$d = 3 : (2172 + 61t)\hat{u} + (-540 + -61t)\hat{v}$
$d = 2 : (19 + 29t)\hat{u} + (1 + -7t)\hat{v}$	$d = 1 : (115 + 131t)\hat{u} + (87 + -11t)\hat{v}$	$d = 3 : (1549 + 13t)\hat{u} + (155 + -377t)\hat{v}$
$d = 2 : (5 + 17t)\hat{u} + (11 + 17t)\hat{v}$	$d = 1 : (37 + 167t)\hat{u} + (-33 + -59t)\hat{v}$	$d = 3 : (579 + -989t)\hat{u} + (609 + -1873t)\hat{v}$
$d = 4 : (1 + 19t)\hat{u} + (3 + -1t)\hat{v}$	$d = 3 : (36 + 107t)\hat{u} + (-102 + -101t)\hat{v}$	$d = 2 : (345 + 353t)\hat{u} + (627 + 821t)\hat{v}$

Figure 13: These theoretically prime-rich lines in the plots of $f_{1,1}$ were produced near instantaneously by the computer.

$k = 6$ $Q = 5.2135416$	$k = 7$ $Q = 5.53938802083$
$d = 4 : (4190 + -13781t)\hat{u} + (11282 + -29647t)\hat{v}$	$d = 1 : (87308 + -64321t)\hat{u} + (-70090 + -275453t)\hat{v}$
$d = 4 : (26505 + 9241t)\hat{u} + (-13949 + -11191t)\hat{v}$	$d = 4 : (371753 + -455761t)\hat{u} + (203541 + -510481t)\hat{v}$
$d = 4 : (14181 + -14813t)\hat{u} + (1285 + -26977t)\hat{v}$	$d = 2 : (418792 + 212521t)\hat{u} + (175024 + -56471t)\hat{v}$
$d = 1 : (2055 + 193t)\hat{u} + (-3359 + 4423t)\hat{v}$	$d = 3 : (200891 + -262517t)\hat{u} + (229281 + -307253t)\hat{v}$
$d = 4 : (27360 + 22531t)\hat{u} + (-13520 + -12541t)\hat{v}$	$d = 4 : (133831 + -321137t)\hat{u} + (-104149 + -400481t)\hat{v}$

$a'_0, b'_0, \omega'_a, \omega'_b$ grow rapidly in magnitude with k, while Q grows slowly. For $k > 8$, the values of f for points along the lines produced are in the quadrillions. This is the greatest disadvantage of this model.

$$f_{m,n}(a\hat{u} + b\hat{v}) = (4a^2 - 5a + 2da + b)m + n \equiv 0 \mod p$$

$$f_{m,n}(a\hat{u} + b\hat{v}) \approx 4ma^2$$

$$f_{m,n}((a'_0 + t\omega'_a)\hat{u} + (b'_0 + t\omega'_b)\hat{v}) \approx 4m(a'_0 + t\omega'_a)^2$$

$$P(X \in \mathbb{P}|X \approx f_{m,n}((a'_0 + t\omega'_a)\hat{u} + (b'_0 + t\omega'_b)\hat{v})) \approx P(X \in \mathbb{P}|X \approx 4m(a'_0 + t\omega'_a)^2)$$

This is approximately $\frac{1}{\ln(4m(a_0+t\omega'_a)^2)-1}$. When a'_0 or ω_a are large, the probability that X is prime is small despite a large Q, and the model still fails to produce an abundance of prime numbers.

This is avoided somewhat by altering steps 2a and 2c. In 2a, one chooses statements where $s = 1$. In 2c, one chooses $\omega_b = 1$. Thus, $\omega'_a \equiv 1 \pmod{p_i}$, to which $\omega'_a = 1$ is always a solution.

6.3 Evaluation

250 lines are produced by the simplied model. On each line, the number of prime points first 250 points with the least a value is recorded. This is repeated without the model, checking 250 lines generated randomly.

R is the actual ratio of primes found by the model divided by primes found without the model. R varies somewhat from its expected values, however its general trend matches Q.

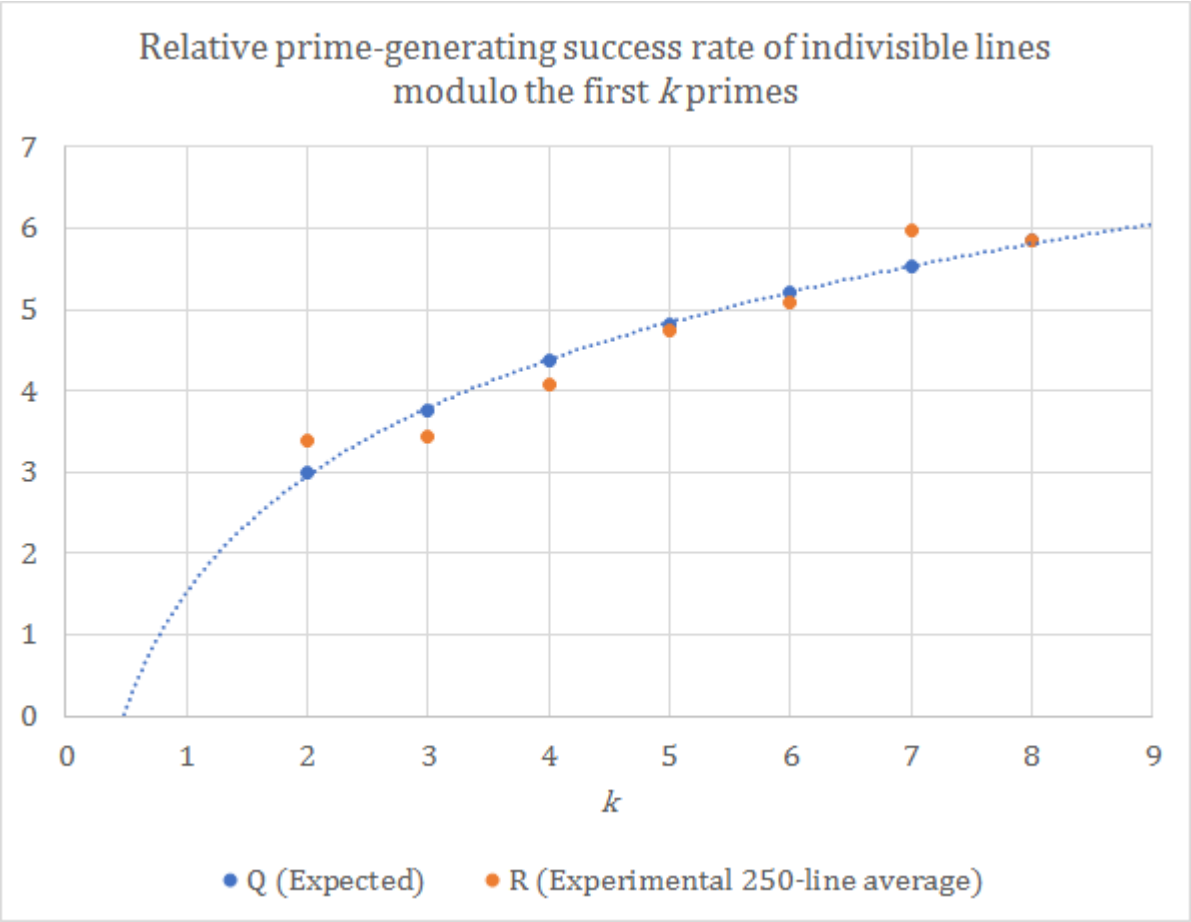


Figure 14: The above graph uses information gathered from $f_{1,1}$. All lines considered were in the form $(a_0\hat{u} + b_0\hat{v}) + t(\hat{u} + \hat{v})$.

7 Conclusion

Mathematically, it was discussed how a model may locate every possible line indivisible by arbitrarily many primes, on any spiral. The lines that have been checked yield primes at about the same rate as the model predicts. The model can achieve a fivefold increase in the probability of generating a prime number. This is not particularly large. However, it is sufficient to show that the Ulam spiral's "mystery" has yet another valid explanation. The significance of a structure perceived in imperfect randomness is usually the existence of more underlying structures.

A Indivisible lines modulo primes up to 17

$d = 1 : a = sb + c$ is an indivisible line modulo p on the divisibility plot of $f_{1,1}$.

p	**r**	**s**	**c**
3	2	1	2
3	2	2	1
5	2	1	4
5	2	2	3
5	2	3	4
5	2	4	3
5	3	1	0
5	3	2	1
5	3	3	1
5	3	4	2
7	3	1	4
7	3	2	0
7	3	3	1
7	3	4	5
7	3	5	6
7	3	6	2
7	5	1	5
7	5	2	4
7	5	3	6
7	5	4	0
7	5	5	2
7	5	6	1
7	6	1	2
7	6	2	6
7	6	3	5
7	6	4	1
7	6	5	0
7	6	6	4
11	2	1	5
11	2	2	3
11	2	3	4
11	2	4	3
11	2	5	10
11	2	6	10
11	2	7	6
11	2	8	5
11	2	9	6
11	2	10	4

cont.

p	**r**	**s**	**c**
11	6	1	8
11	6	2	10
11	6	3	5
11	6	4	1
11	6	5	4
11	6	6	5
11	6	7	8
11	6	8	4
11	6	9	10
11	6	10	1
11	7	1	6
11	7	2	9
11	7	3	8
11	7	4	6
11	7	5	8
11	7	6	1
11	7	7	3
11	7	8	1
11	7	9	0
11	7	10	3
11	8	1	4
11	8	2	8
11	8	3	0
11	8	4	0
11	8	5	1
11	8	6	8
11	8	7	9
11	8	8	9
11	8	9	1
11	8	10	5
11	10	1	0
11	10	2	6
11	10	3	6
11	10	4	10
11	10	5	9
11	10	6	0
11	10	7	10
11	10	8	3
11	10	9	3
11	10	10	9
13	2	1	9
13	2	2	9
13	2	3	12
13	2	4	6
13	2	5	12
13	2	6	11
13	2	7	6
13	2	8	5
13	2	9	11
13	2	10	5
13	2	11	8

cont.

p	**r**	**s**	**c**
13	2	12	8
13	5	1	10
13	5	2	3
13	5	3	8
13	5	4	3
13	5	5	7
13	5	6	9
13	5	7	8
13	5	8	10
13	5	9	1
13	5	10	9
13	5	11	1
13	5	12	7
13	6	1	6
13	6	2	1
13	6	3	11
13	6	4	2
13	6	5	1
13	6	6	4
13	6	7	0
13	6	8	3
13	6	9	2
13	6	10	6
13	6	11	3
13	6	12	11
13	7	1	2
13	7	2	12
13	7	3	1
13	7	4	1
13	7	5	8
13	7	6	12
13	7	7	5
13	7	8	9
13	7	9	3
13	7	10	3
13	7	11	5
13	7	12	2
13	8	1	11
13	8	2	10
13	8	3	4
13	8	4	0
13	8	5	2
13	8	6	7
13	8	7	10
13	8	8	2
13	8	9	4
13	8	10	0
13	8	11	7
13	8	12	6
13	11	1	12
13	11	2	4
13	11	3	0

cont.

p	**r**	**s**	**c**
13	11	4	10
13	11	5	10
13	11	6	5
13	11	7	12
13	11	8	7
13	11	9	7
13	11	10	4
13	11	11	0
13	11	12	5
17	3	1	2
17	3	2	13
17	3	3	12
17	3	4	8
17	3	5	13
17	3	6	14
17	3	7	3
17	3	8	10
17	3	9	12
17	3	10	2
17	3	11	8
17	3	12	9
17	3	13	14
17	3	14	10
17	3	15	9
17	3	16	3
17	5	1	0
17	5	2	12
17	5	3	0
17	5	4	16
17	5	5	16
17	5	6	8
17	5	7	10
17	5	8	14
17	5	9	8
17	5	10	12
17	5	11	14
17	5	12	6
17	5	13	6
17	5	14	5
17	5	15	10
17	5	16	5
17	6	1	16
17	6	2	3
17	6	3	11
17	6	4	3
17	6	5	9
17	6	6	5
17	6	7	5
17	6	8	16
17	6	9	6
17	6	10	0

cont.

p	**r**	**s**	**c**
17	6	11	0
17	6	12	13
17	6	13	2
17	6	14	11
17	6	15	2
17	6	16	6
17	7	1	15
17	7	2	11
17	7	3	5
17	7	4	7
17	7	5	2
17	7	6	2
17	7	7	0
17	7	8	1
17	7	9	4
17	7	10	5
17	7	11	3
17	7	12	3
17	7	13	15
17	7	14	0
17	7	15	11
17	7	16	7
17	10	1	12
17	10	2	1
17	10	3	4
17	10	4	2
17	10	5	15
17	10	6	10
17	10	7	2
17	10	8	7
17	10	9	15
17	10	10	3
17	10	11	12
17	10	12	7
17	10	13	3
17	10	14	1
17	10	15	4
17	10	16	10
17	11	1	11
17	11	2	9
17	11	3	15
17	11	4	6
17	11	5	8
17	11	6	7
17	11	7	14
17	11	8	9
17	11	9	13
17	11	10	8
17	11	11	15
17	11	12	14
17	11	13	16
17	11	14	7

cont.

p	**r**	**s**	**c**
17	11	15	13
17	11	16	11
17	12	1	10
17	12	2	0
17	12	3	9
17	12	4	10
17	12	5	1
17	12	6	4
17	12	7	9
17	12	8	11
17	12	9	11
17	12	10	13
17	12	11	1
17	12	12	4
17	12	13	12
17	12	14	13
17	12	15	5
17	12	16	12
17	14	1	8
17	14	2	16
17	14	3	14
17	14	4	1
17	14	5	4
17	14	6	15
17	14	7	16
17	14	8	15
17	14	9	7
17	14	10	6
17	14	11	7
17	14	12	1
17	14	13	4
17	14	14	8
17	14	15	6
17	14	16	14

B Model test data

k	2	3	4	5	6	7	8
Q	3	3.75	4.375	4.8125	5.213541667	5.539388021	5.8471318
Primes found with model	20211	20356	23416	19384	16036	13486	11442
Primes found without model	5955	5913	5749	4093	3153	2253	1954
R	3.39395466	3.442584137	4.073056184	4.735890545	5.085949889	5.985796715	5.855680655

C Code

```
import java.awt.Color; import java.awt.Desktop; import java.awt.Graphics2D;
import java.awt.image.BufferedImage; import java.awt.image.RenderedImage;
import java.io.File; import java.io.IOException;
import javax.imageio.ImageIO;
import java.util.Scanner;

class ulam_spiral {
 static Scanner scanner = new Scanner(System.in);
 static Color axis = new Color(191, 191, 191);
 static Color line = new Color(127, 127, 127);
 static Color tile1 = new Color(191, 255, 191);
 static Color tile2 = new Color(191, 191, 255);

 public static void main (String[] args) {
  System.out.println("d: divisibility plot modulo p");
  System.out.println("m: the model itself");
  System.out.println("s: divisibility plot modulo p, generated using the symmetries");
  System.out.println("u: prime points are coloured black");
  switch (in("> Plot type (d/m/s/u):").charAt(0)) {
   case 'd': d(); break;
   case 'm': m(); break;
   case 's': s(); break;
   case 'u': u(); break;
  }
 }

 static String in (String prompt) {System.out.print(prompt); return scanner.nextLine();}
 static int inInt (String prompt) {return Integer.parseInt(in(prompt));}

 // returns an int between a and b inclusive
 static long rand (long a, long b) {return a + (long) (Math.random() * (b - a + 1));}

 // returns a value from 0 to p-1
 static long mod (long a, long p) {long b = a % p; return b < 0 ? b + p : b;}

 // returns an array where array[i] is true if i is not prime
 static boolean[] notEratosthenes (int max) {
  boolean[] notPrime = new boolean[max];
  notPrime[0] = notPrime[1] = true; int maxI = (int) Math.sqrt(max);
  for (int i = 2; i <= maxI; i ++) {
   if (!notPrime[i]) {for (int j = i * i; j < max; j += i) {notPrime[j] = true;}}
  }
  return notPrime;
 }

 // returns an array where array[i] is the ith prime
 static int[] primes (int k) {
  int[] p = new int[k];
  p[0] = 2; p[1] = 3;
  for (int i = 2; i < k; i ++) {
   int candidate = p[i - 1];
   boolean prime = false;
   while (!prime) {
    candidate += 2;
    prime = true;
    for (int j = 0; j < i; j ++)
     if (candidate % p[j] == 0) {prime = false; break;}
   }
   p[i] = candidate;
  }
  return p;
 }
```

```
// evaluates f for the point in quadrant d: au+bv
static long f (int m, int n, int d, long a, long b) {
 return (4 * a * a + (2 * d - 5) * a + b) * m + n;
}

// determines if the point au+bv is actually in the quadrant d
static boolean isValid (int d, long a, long b) {
 if ((a == 0) && (b == 0)) {return true;}
 switch (d) {
  case 1: return ((-a + 2 <= b) && (b <= a));
  case 4: return ((-a + 1 <= b) && (b <= a + 1));
  default: return ((-a <= -b) && (-b <= a-1));
 }
}

static boolean isPrime (long n) {
 if (n < 2) {return false;}
 if ((n % 2 == 0) && (n > 2)) {return false;}
 for (long i = 3; i * i <= n; i ++) {if (n % i == 0) {return false;}}
 return true;
}

// returns the product of the array p
static long prod (int[] p) {
 long m = p[0];
 for (int i = 1; i < p.length; i ++) {m *= p[i];}
 return m;
}

// returns the least residue solution to the CRT for the list of residues a modulo p
static long crt (int[] a, int[] p) {
 long m = prod(p); long x = 0;
 for (int i = 0; i < a.length; i ++) {
  long mi = m / p[i]; x += a[i] * mi * inv(mi, p[i]);
 }
 return x % m;
}

static long inv (long a, long p) {return eea(a, p)[0];} // returns a^-1 mod p, may be negative

// returns x and y, where ax + by = gcd(a,b)... gcd(a,b) always equals 1 in this case
static long[] eea (long a, long b) {
 long x; long x1 = 0; long x2 = 1; long y; long y1 = 1; long y2 = 0; long q; long r;
 while (b > 0) {
  q = a / b; r = a - q * b; x = x2 - q * x1; y = y2 - q * y1;
  a = b; b = r; x2 = x1; x1 = x; y2 = y1; y1 = y;
 }
 long[] results = {x2, y2}; return results;
}

// returns a list containing all (p - 1) / 2 quadratic nonresidues modulo p
static int[] nonresidues (int p) {
 boolean[] residue = new boolean[p]; int[] nonresidue = new int[p / 2];
 for (int i = 0; 2 * i < p; i ++) {residue[i * i % p] = true;}
 int index = 0;
 for (int i = 0; i < p; i ++) {
  if (!residue[i]) {nonresidue[index] = i; index ++;}
 }
 return nonresidue;
}
```

```
 static void d () {
  int m = inInt("> m ="); int n = inInt("> n ="); int p = inInt("> p =");
  int depth = inInt("> |x|,|y| <="); int res = inInt("> Zoom factor:");
  BufferedImage img = new BufferedImage(2*depth+1, 2*depth+1, BufferedImage.TYPE_USHORT_GRAY);
  img = axes(img, depth);
  img = plotD(img, p, depth, m, n);
  img = stretch(img, (2 * depth + 1) * res, false);
  savePNG(img, "Renders/Divisibility Plots/m"+m+" n"+n+" p"+p+", depth"+depth+" res"+res+".png");
 }

 static void m () {
  int m = inInt("> m ="); int n = inInt("> n =");
  int k = inInt("> k ="); int lines = inInt("> Lines:");
  boolean simple = in("> Force wa=1 (y/n):").charAt(0) == 'y';

// Step 1
  int[] p = primes(k); int coprimes = k;
  double q = 1; for (int i = 0; i < k; i ++) {q *= ((double) p[i]) / ((double) p[i] - 1);}
  System.out.println(q);

// Step 2
  for (int i = k-1; i >= 0; i --) {
   if (m % p[i] == 0) {
    if (n % p[i] == 0) {
     System.out.println(m + " and " + n + " are both multiples of " + p[i] + ".");
     return;
    }
    p[i] = 0;
    coprimes --;
   }
  }
  int[] mIn = new int[k]; int[] inv8 = new int[k]; int[] inv16 = new int[k];
  for (int i = 0; i < k; i ++) {
   if (p[i] > 0) {
    mIn[i] = (int) inv(m,p[i]) * n % p[i];
    inv8[i] = (int) inv(8,p[i]);
    inv16[i] = (int) inv(16,p[i]);
   }
  }

// Step 3
  int deltaA1[][] = new int[5][k]; int deltaB1[][] = new int[5][k]; // d-2 k
  for (int d = 1; d < 5; d ++) {
   for (int i = 0; i < k; i ++) {
    if (p[i] > 0) {
     deltaA1[d][i] = (int) mod(inv8[i] * (2 - 2 * d), p[i]);
     deltaB1[d][i] = (int) mod(inv8[i] * (1 - 4 * inv8[i]) * ((5 - 2*d) * (5 - 2*d) - 9), p[i]);


    }
   }
  }
  int[][][] c = new int[k][p[k-1]/2][p[k-1]];
  for (int i = 1; i < k; i ++) {
   if (p[i] > 0) {

// Step 4
    int[] nonresidue = nonresidues(p[i]);

// Step 5
    for (int j = 0; j * 2 < p[i] - 1; j ++) {
     for (int s = 1; s < p[i]; s ++) {
```

```
      c[i][j][s] = (int) mod(-inv16[i] * 9 * s + s * mIn[i] + inv8[i] * 3 + (nonresidue[j] - 1) *
inv16[i] * inv(s,p[i]), p[i]);
     }
    }
   }
  }
  long product = 1;
  int pTotal = 0;
  for (int l = lines; l > 0; l --) {

// Step 6
   int[] select = new int[coprimes];
   int index = 0; for (int i = 0; i < k; i ++) {if (p[i] > 0) {select[index] = i; index ++;}}

// Step 7
   int[] pSelect = new int[coprimes];
   int[] sSelect = new int[coprimes];
   int[] cSelect = new int[coprimes];
   for (int i = 0; i < coprimes; i ++) {
    pSelect[i] = p[select[i]];
    if (simple) {sSelect[i] = 1;} else {sSelect[i] = (int) rand(1,pSelect[i]-1);}
    cSelect[i] = c[select[i]][(int) rand(0,(pSelect[i]-3)/2)][sSelect[i]];
   }
   if (p[0] == 2) {sSelect[0] = 1; cSelect[0] = (1 + n) % 2;}

// Step 8
   int d = (int) rand(1, 4);
   for (int i = 0; i < coprimes; i ++) {
    cSelect[i] = (int) mod(cSelect[i] - sSelect[i] * deltaB1[d][select[i]] +
deltaA1[d][select[i]], pSelect[i]);
   }

// Step 9
   int[] a0 = new int[coprimes];
   int[] b0 = new int[coprimes];
   int[] wa = new int[coprimes];
   int[] wb = new int[coprimes];
   for (int i = 0; i < coprimes; i ++) {
    b0[i] = (int) rand(0, pSelect[i]-1);
    if (simple) {wb[i] = 1;} else {wb[i] = (int) rand(1, pSelect[i]-1);}
    a0[i] = (sSelect[i] * b0[i] + cSelect[i]) % pSelect[i];
    wa[i] = (sSelect[i] * wb[i]) % pSelect[i];
   }

// Step 10
   product = prod(pSelect);
   long waprime = crt(wa, pSelect);
   if (waprime < 0) {waprime += product;}
   long wbprime = crt(wb, pSelect);
   if (wbprime < -product / 2) {wbprime += product;}
   else if (wbprime > product / 2) {wbprime -= product;}
   long a0prime = crt(a0, pSelect);
   if (a0prime < 0) {a0prime += product;}
   long b0prime = crt(b0, pSelect);
   if (b0prime < -product / 2) {b0prime += product;}
   else if (b0prime > product / 2) {b0prime -= product;}
   if ((simple) && (b0prime > 0)) {b0prime -= product;}
   if (!(((0 <= wbprime) && (wbprime <= waprime)) || ((0 >= wbprime) && (wbprime >= waprime)))) {
    waprime -= product; wbprime -= product;
   }
   System.out.println("d = "+d+": ("+a0prime+"+"+waprime+"t)u + ("+b0prime+"+"+wbprime+"t)v");
   if (simple) {
```

```
    long t = (- a0prime - b0prime) / 2 - 2;
    int pCount = 0;
    int points = 250;
    while (points > 0) {
     long a = a0prime + waprime * t;
     long b = b0prime + wbprime * t;
     if (isValid(d,a,b)) {
      points --;
      long f = f(m, n, d, a, b);
      if (isPrime(f)) {pCount ++;}
     }
     t ++;
    }
    pTotal += pCount;
   }
  }
  if (simple) {
   System.out.println("Primes found using model: " + pTotal);
   int pFakeTotal = 0;
   for (int i = 0; i < lines; i ++) {
    int d = (int) rand(1, 4);
    long a0fake = rand(0, product);
    long b0fake = rand(-product, 0);
    long t = (- a0fake - b0fake) / 2 - 2;
    int pFakeCount = 0;
    int points = 250;
    while (points > 0) {
     long a = a0fake + t;
     long b = b0fake + t;
     if (isValid(d,a,b)) {
      points --;
      long f = f(m, n, d, a, b);
      if (isPrime(f)) {pFakeCount ++;}
     }
     t ++;
    }
    pFakeTotal += pFakeCount;
   }
   System.out.println("Primes found randomly: " + pFakeTotal);
  }
 }

 static void s () { // p must be odd, m must not be a multiple of p
  int m = inInt("> m ="); int n = inInt("> n ="); int p = inInt("> p =");
  int depth = inInt("> |x|,|y| <="); int res = inInt("> Zoom factor:");
  int mIn = n; while (m * mIn % p != n % p) {mIn += n;}
  int inv8 = 1; while (8 * inv8 % p != 1) {inv8 += 1;}
  int ar1 = (int) mod(inv8 * 3, p) + p; int br1 = (int) mod(inv8 * (1 - p) / 2 * 9 - mIn, p);
  int ar2 = (int) mod(inv8 * 1, p) + p; int br2 = (int) mod(inv8 * (1 - p) / 2 * 1 - mIn, p);
  int ar3 = (int) mod(inv8 * -1, p) + p; int br3 = (int) mod(inv8 * (1 - p) / 2 * 1 - mIn, p);
  int ar4 = (int) mod(inv8 * -3, p) + p; int br4 = (int) mod(inv8 * (1 - p) / 2 * 9 - mIn, p);
  System.out.println("Vertices of the parabola au+g(a)v\\) are found at:");
  System.out.println("d = 1 : " + ar1 + "u+" + br1 + "v = (" + ar1 + "," + br1 + ")");
  System.out.println("d = 2 : " + ar2 + "u}+" + br2 + "v = (" + -br2 + "," + ar2 + ")");
  System.out.println("d = 3 : " + ar3 + "u+" + br3 + "v = (" + -ar3 + "," + -br3 + ")");
  System.out.println("d = 4 : " + ar4 + "u+" + br4 + "v = (" + br4 + "," + -ar4 + ")");
  System.out.println("Using " + (p+1)/2 + " divisible points on the half-tile " + ar1 + "<=x<=}"
+ (ar1 + p/2) + ",0<=y<p:");
  int[] a1 = new int[(p + 1) / 2]; int[] b1 = new int[(p + 1) / 2];
  a1[0] = ar1; b1[0] = br1; System.out.println("(" + a1[0] + "," + b1[0] + ")");
  for (int i = 1; i <= p / 2; i ++) {
   a1[i] = i + ar1; b1[i] = (int) mod(-4 * a1[i] * a1[i] + 3 * a1[i] - mIn, p);
```

```
   System.out.println("(" + a1[i] + "," + b1[i] + ")");
  }
  BufferedImage img = new BufferedImage(2*depth+1, 2*depth+1, BufferedImage.TYPE_USHORT_555_RGB);
  img = tiles(img, p, depth, ar1, br1, ar2, br2, ar3, br3, ar4, br4);
  img = plotS(img, p, depth, ar1, br1, ar2, br2, ar3, br3, ar4, br4, a1, b1, n);
  img = stretch(img, (2 * depth + 1) * res, true);
  img = drawS(img, p, depth, res, ar1, ar2, ar3, ar4);
  Graphics2D graphics = img.createGraphics();
  graphics.setPaint(Color.WHITE);
  for (int i = 0; i < a1.length; i ++)
   graphics.fillRect((depth + a1[i]) * res, (depth - b1[i]) * res, res, res);
  savePNG (img, "Renders/Symmetry Plots/m"+m+" n"+n+" p"+p+", depth"+depth+" res"+res+".png");
 }

 static void u () {
  int m = inInt("> m ="); int n = inInt("> n =");
  int depth = inInt("> |x|,|y| <="); int res = inInt("> Zoom factor:");
  BufferedImage img = new BufferedImage(2*depth+1, 2*depth+1, BufferedImage.TYPE_USHORT_GRAY);
  img = axes(img, depth);
  img = plotU(img, depth, m, n);
  img = stretch(img, (2 * depth + 1) * res, false);
  savePNG (img, "Renders/Prime Plots/m"+m+" n"+n+", depth"+depth+" res"+res+".png");
 }

 static BufferedImage axes (BufferedImage img, int depth) {
  Graphics2D graphics = img.createGraphics();
  graphics.setPaint(Color.WHITE);
  graphics.fillRect(0, 0, img.getWidth(), img.getHeight());
  graphics.setPaint(axis);
  graphics.drawLine(0, depth, img.getWidth(), depth);
  graphics.drawLine(depth, 0, depth, img.getHeight());
  graphics.drawLine(0, 0, img.getWidth()-1, img.getHeight()-1);
  graphics.drawLine(0, img.getHeight()-1, img.getWidth()-1, 0);
  return img;
 }

 static BufferedImage plotD (BufferedImage img, int p, int depth, int m, int n) {
  int x = 0; int y = 0; int points = 0;
  int max = n + m * img.getWidth() * img.getHeight();
  for (int i = n; i < max; i += m) {
   if (i % p == 0) {img.setRGB(depth + x, depth - y, Color.BLACK.getRGB()); points ++;}
   if (y > -x) {if (y < x) {y ++;} else {x --;}} else {if (y > x) {y --;} else {x ++;}}
  }
  System.out.println(points + " points plotted.");
  return img;
 }

 static BufferedImage plotS (BufferedImage img, int p, int depth, int ar1, int br1, int ar2, int
br2, int ar3, int br3, int ar4, int br4, int a[], int b[], int n) {
  for (int i = 0; i < a.length; i ++) {
   int a1 = a[i] % p;
   int a2 = (a[i] + ar2 - ar1 + p) % p;
   int a3 = (a[i] + ar3 - ar1 + p) % p;
   int a4 = (a[i] + ar4 - ar1 + p) % p;
   int b1 = b[i];
   int b2 = p - (b[i] + br2 - br1 + p) % p;
   int b3 = p - (b[i] + br3 - br1 + p) % p;
   int b4 = (b[i] + br4 - br1 + p) % p;

   for (int y = (depth % p < b1) ? depth / p * p - p + b1 : depth / p * p + b1; y >= -depth; y -=
p) { // d = 1
```

```
    for (int x = (depth % p < a1) ? depth / p * p - p + a1 : depth / p * p + a1; ((-x + 2 <= y)
&& (y <= x)); x -= p)
     img.setRGB(depth + x, depth - y, Color.BLACK.getRGB());
    if (i > 0) {
     a1 = (2 * ar1 - a1 + p) % p;
     for (int x = (depth % p < a1) ? depth / p * p - p + a1 : depth / p * p + a1; ((-x + 2 <= y)
&& (y <= x)); x -= p)
      img.setRGB(depth + x, depth - y, Color.BLACK.getRGB());
    }
   } for (int x = (depth % p < b2) ? depth / p * p - p + b2 : depth / p * p + b2; x >= -depth; x
-= p) { // d = 2
    for (int y = (depth % p < a2) ? depth / p * p - p + a2 : depth / p * p + a2; ((-y <= x) && (x
<= y - 1)); y -= p)
     img.setRGB(depth + x, depth - y, Color.BLACK.getRGB());
    if (i > 0) {
     a2 = (2 * ar2 - a2 + p) % p;
     for (int y = (depth % p < a2) ? depth / p * p - p + a2 : depth / p * p + a2; ((-y <= x) &&
(x <= y - 1)); y -= p)
      img.setRGB(depth + x, depth - y, Color.BLACK.getRGB());
    }
   } for (int y = (depth % p < b3) ? depth / p * p - p + b3 : depth / p * p + b3; y >= -depth; y
-= p) { // d = 3
    for (int x = -((depth % p < a3) ? depth / p * p - p + a3 : depth / p * p + a3); ((x <= y) &&
(y <= -x - 1)); x += p)
     img.setRGB(depth + x, depth - y, Color.BLACK.getRGB());
    if (i > 0) {
     a3 = (2 * ar3 - a3 + p) % p;
     for (int x = -((depth % p < a3) ? depth / p * p - p + a3 : depth / p * p + a3); ((x <= y) &&
(y <= -x - 1)); x += p)
      img.setRGB(depth + x, depth - y, Color.BLACK.getRGB());
    }
   } for (int x = (depth % p < b4) ? depth / p * p - p + b4 : depth / p * p + b4; x >= -depth; x
-= p) { // d = 4
    for (int y = -((depth % p < a4) ? depth / p * p - p + a4 : depth / p * p + a4); ((y + 1 <= x)
&& (x <= -y + 1)); y += p)
     img.setRGB(depth + x, depth - y, Color.BLACK.getRGB());
    if (i > 0) {
     a4 = (2 * ar4 - a4 + p) % p;
     for (int y = -((depth % p < a4) ? depth / p * p - p + a4 : depth / p * p + a4); ((y + 1 <=
x) && (x <= -y + 1)); y += p)
      img.setRGB(depth + x, depth - y, Color.BLACK.getRGB());
    }
   }
  }
  if (n % p == 0) {img.setRGB(depth, depth, Color.BLACK.getRGB());}
  System.out.println("Plotted " + (p+1)/2 + " redundant sets.");
  return img;
 }

 static BufferedImage plotU (BufferedImage img, int depth, int m, int n) {
  int x = 0; int y = 0; int points = 0;
  int max = n + m * img.getWidth() * img.getHeight();
  boolean[] notPrime = notEratosthenes(max);
  for (int i = n; i < max; i += m) {
   if (!notPrime[i]) {img.setRGB(depth + x, depth - y, Color.BLACK.getRGB()); points ++;}
   if (y > -x) {if (y < x) {y ++;} else {x --;}} else {if (y > x) {y --;} else {x ++;}}
  }
  System.out.println(points + " primes plotted.");
  return img;
 }
```

```
 static BufferedImage tiles (BufferedImage img, int p, int depth, int ar1, int br1, int ar2, int
br2, int ar3, int br3, int ar4, int br4) {
  int a1 = ar1 - (p - 1) / 2; int a2 = ar2 - (p - 1) / 2; int a3 = ar3 - (p - 1) / 2; int a4 =
ar4 - (p - 1) / 2;
  int b2 = br2 - br1; int b3 = br3 - br1; int b4 = br4 - br1;
  for (int x = -depth; x <= depth; x ++) {
   for (int y = -depth; y <= depth; y ++) {
    if ((-x + 2 <= y) && (y <= x))
     img.setRGB(depth + x, depth - y, (((Math.floor(((x - a1) / (double) p) + Math.floor(y /
(double) p))) % 2 != 0) ? tile1 : tile2).getRGB());
    else if ((- y <= x) && (x <= y - 1))
     img.setRGB(depth + x, depth - y, (((Math.floor(((y - a2) / (double) p) + Math.floor((-x -
b2) / (double) p))) % 2 != 0) ? tile1 : tile2).getRGB());
    else if (x <= y) // D == 3 the condition (y <= -x - 1) is redundant, since any point where
that is false is part of the previous quadrants
     img.setRGB(depth + x, depth - y, (((Math.floor(((-x - a3) / (double) p) + Math.floor((-y -
b3) / (double) p))) % 2 != 0) ? tile1 : tile2).getRGB());
    else
     img.setRGB(depth + x, depth - y, (((Math.floor(((-y - a4) / (double) p) + Math.floor((x -
b4) / (double) p))) % 2 != 0) ? tile1 : tile2).getRGB());
   }
  }
  Graphics2D graphics = img.createGraphics();
  graphics.setPaint(axis);
  graphics.drawLine(0, depth, img.getWidth(), depth);
  graphics.drawLine(depth, 0, depth, img.getHeight());
  return img;
 }

 static BufferedImage stretch (BufferedImage img, int size, boolean colour) {
  BufferedImage stretched = (colour) ? new BufferedImage(size, size,
BufferedImage.TYPE_USHORT_555_RGB) : new BufferedImage(size, size,
BufferedImage.TYPE_USHORT_GRAY);
  Graphics2D graphics = stretched.createGraphics();
  graphics.drawImage(img, 0, 0, size, size, null);
  return stretched;
 }

 static BufferedImage drawS (BufferedImage img, int p, int depth, int res, int ar1, int ar2, int
ar3, int ar4) {
  Graphics2D graphics = img.createGraphics();
  graphics.setPaint(line);
  if (res % 2 == 0) {
   for (int x = ar1 % p; x <= depth; x += p) {
    for (int y = -x + 2; y <= x; y ++) {
     if (img.getRGB(((depth + x) * res), (depth - y) * res) != Color.BLACK.getRGB())
      graphics.fillRect((depth + x) * res + res / 2 - 1, (depth - y) * res, 2, res);
    }
   } for (int y = ar2 % p; y <= depth; y += p) {
    for (int x = -y; x <= y - 1; x ++) {
     if (img.getRGB(((depth + x) * res), (depth - y) * res) != Color.BLACK.getRGB())
      graphics.fillRect((depth + x) * res, (depth - y) * res + res / 2 - 1, res, 2);
    }
   } for (int x = -ar3 % p; x >= -depth; x -= p) {
    for (int y = x; y <= -x - 1; y ++) {
     if (img.getRGB(((depth + x) * res), (depth - y) * res) != Color.BLACK.getRGB())
      graphics.fillRect((depth + x) * res + res / 2 - 1, (depth - y) * res, 2, res);
    }
   } for (int y = -ar4 % p; y >= -depth; y -= p) {
    int max = (-y + 1 > depth) ? depth : -y + 1;
    for (int x = y + 1; x <= max; x ++) {
     if (img.getRGB(((depth + x) * res), (depth - y) * res) != Color.BLACK.getRGB())
```

```
      graphics.fillRect((depth + x) * res, (depth - y) * res + res / 2 - 1, res, 2);
     }
    }
   } else {
    for (int x = ar1 % p; x <= depth; x += p) {
     for (int y = -x + 2; y <= x; y ++) {
      if (img.getRGB(((depth + x) * res), (depth - y) * res) != Color.BLACK.getRGB())
       graphics.fillRect((depth + x) * res + res / 2, (depth - y) * res, 1, res);
     }
    } for (int y = ar2 % p; y <= depth; y += p) {
     for (int x = -y; x <= y - 1; x ++) {
      if (img.getRGB(((depth + x) * res), (depth - y) * res) != Color.BLACK.getRGB())
       graphics.fillRect((depth + x) * res, (depth - y) * res + res / 2, res, 1);
     }
    } for (int x = -ar3 % p; x >= -depth; x -= p) {
     for (int y = x; y <= -x - 1; y ++) {
      if (img.getRGB(((depth + x) * res), (depth - y) * res) != Color.BLACK.getRGB())
       graphics.fillRect((depth + x) * res + res / 2, (depth - y) * res, 1, res);
     }
    } for (int y = -ar4 % p; y >= -depth; y -= p) {
     int max = (-y + 1 > depth) ? depth : -y + 1;
     for (int x = y + 1; x <= max; x ++) {
      if (img.getRGB(((depth + x) * res), (depth - y) * res) != Color.BLACK.getRGB())
       graphics.fillRect((depth + x) * res, (depth - y) * res + res / 2, res, 1);
     }
    }
   }
   return img;
  }

  static void savePNG (BufferedImage img, String path) {
   try {
    RenderedImage render = img;
    ImageIO.write(render, "png", new File(path));
    System.out.println("Saved as " + path + ".");
    Desktop.getDesktop().open(new File(path));
   } catch (IOException e) {e.printStackTrace();}
  }
 }
```

Works cited

Bates, T. (n.d.). Ulam Spiral. Retrieved February 05, 2018, from http://www.betweenartandscience.com/ulamspiral_words.html

Granville, A. (n.d.). Harald Cramér and the Distribution of Prime Numbers. Retrieved February 5, 2018, from https://www.dartmouth.edu/~chance/chance_news/for_chance_news/Riemann/cramer.pdf
Athens, Georgia, USA.

Inglis-Arkell, E. (2011, July 09). The Bizarre Mathematical Conundrum Of Ulam's Spiral. Retrieved February 05, 2018, from https://www.gizmodo.com.au/2011/07/the-bizarre-mathematical-conundrum-of-ulams-spiral/

Ribenboim, P. (2004). The little book of bigger primes (2nd ed.). New York: Springer. Accessed February 5, 2018

Sacks, R. (n.d.). NumberSpiral.com. Retrieved February 05, 2018, from https://www.numberspiral.com/index.html

Stein, M. L., Ulam, S. M., & Wells, M. B. (1964). A Visual Display of Some Properties of the Distribution of Primes. The American Mathematical Monthly, 71(5), 516-520. doi:10.2307/2312588

Wright, S. (2016). *Quadratic residues and non-residues: selected topics*. Switzerland: Springer. Accessed February 5, 2018