Implementing Concrete Computation and Plotting Analytic Functions in $\mathbb{Q}_p$

Jay Reiter

May 2021

1 Introduction

The p-adic numbers are often studied abstractly through their algebraic and number theoretic properties. As we know from Ostrowski's theorem though, but the p-adic numbers are every bit as "real" as $\mathbb{R}$. Something we have in the real numbers that we lack in $\mathbb{Q}_p$ is a an instinct for basic numerical operations like $\cdot$ and $+$ and strong visual intuition for the action of functions. This project attempts to bring the p-adics into the familiar, concrete space of calculation and graphs.

We do this with a C++ implementation of the p-adic numbers which supports basic arithmetic operations like $+$ and $\cdot$ as well as additive and multiplicative inverses. No such utility exists in C/C++ yet (that could easily be found online). The most popular package to use for p-adic computation is `ExactpAdics` for the Magma Computational Algebra System. There are also some very limited packages in Python. A C++ implementation, though, is much more portable and serves our needs well in this project.

Finally, we use our implementation of the p-adics to generate visual plots of analytic functions like exp, sin, and cos on $\mathbb{Z}_p$. All the code for this project, along with several sample visualizations can be found on GitHub.

2 The p-adic Numbers

In this section, we will review some of facts about the p-adic numbers that are used in this project.

2.1 Construction

There are two constructions of the p-adic numbers that we are concerned with in this project. The first is given by the following:

Definition. *[1, p.10] Fix some prime number, p. Let the p-**adic numbers**, $\mathbb{Q}_p$, be the set of all two sided sequences*

$$\ldots a_2\, a_1\, a_0\, .\, a_{-1}\, a_{-2} \ldots$$

where each $a_i \in \{0, 1, \ldots, p-1\}$, and $a_{-n} = 0$ for all negative n.

*Let the p-**adic integers**, $\mathbb{Z}_p$, be the subset of such sequences where $a_n = 0$ for all $n < 0$.*

This definition is nice since it gives a very concrete intuition for what a p-adic number *is*. Though we are often unable to write down particular p-adic numbers since they may have infinitely many non-repeating, nonzero digits to the left (note: $\mathbb{Z}_p$ and $\mathbb{Q}_p$ are uncountable), this definition shows us how the p-ary expansion of any positive integer can immediately be interpreted as a p-adic number. We therefore have some familiar elements to play around with and use to understand $\mathbb{Z}_p$.

Note that $(\mathbb{Q}_p, +, \cdot)$ is a field where $+$ and $\cdot$ are natural extensions of the usual addition and multiplication on $\mathbb{Q}$, and that $(\mathbb{Z}_p, +, \cdot)$ is a commutative subring of $\mathbb{Q}_p$. So, playing with the positive integers in $\mathbb{Z}_p$ will yield few surprises. In fact, all of $\mathbb{Q}$ is (isomorphic to) a subfield of $\mathbb{Q}_p$ [1, p.10]. We will see later that though this isomorphism "preserves" the positive rationals with non-repeating decimals in the sense that their notation is identical in $\mathbb{Q}$ and $\mathbb{Q}_p$, other p-adic numbers present in less familiar ways.

(One topic I'm interested in but that I have not investigated here is the inclusion of rational numbers with infinitely repeating decimal expansions into $\mathbb{Q}_p$. For example, what does the rational number $1/3$ look like in $\mathbb{Q}_2$?)

We now introduce the p-adic norm:

Definition. *Let $a = \ldots a_2\, a_1\, a_0\, .\, a_{-1} \ldots \in \mathbb{Q}_p$. The **order** of a is the smallest integer m for which $a_m \neq 0$. Symbolically,*

$$\operatorname{ord}_p(\ldots a_2\, a_1\, a_0\, .\, a_{-1} \ldots) = \begin{cases} \infty & a_i = 0,\ \forall i \\ \min\{m : a_m \neq 0\} & \text{otherwise.} \end{cases}$$

*From this, we define the p-**adic norm** on $\mathbb{Z}_p$:*

$$|x|_p = p^{-\operatorname{ord}_p(x)}.$$

The p-adic norm satisfies the *strong triangle inequality*, that is, for all x and y in $\mathbb{Q}_p$, we have $|x + y|_p \leq \max\{|x|_p, |y|_p\}$. This distinguishes the p-adic norm from the usual Euclidean absolute value on $\mathbb{Q}$ which only satisfies the ordinary triangle in equality. In the p-adic numbers, $|1 + 1|_p \leq 1$, and for this reason we call the p-adic norm *non-Archimedean.*

Since the rationals are contained in $\mathbb{Q}_p$, it makes sense to talk about the p-adic norm of a rational number:

Theorem. *[1, p.11] Let x be a non-zero rational number. There exist integers s and t, both coprime to p, such that*

$$x = p^n \frac{s}{t}.$$

Then $|x|_p = p^{-n}$.

We get the intuition that the p-adic norm is like a measure of how close a rational is to a large power of p. From this theorem, we get an alternative construction of the p-adic numbers which does not give us a notion of a "typical" p-adic number as did the previous construction.

Theorem. *$\mathbb{Q}_p$ is the completion of $\mathbb{Q}$ with respect to the metric induced by the p-adic norm on $\mathbb{Q}$.*

Of course, this discussion brings us to the following theorem which motivates the study of the p-adic numbers:

Theorem (Ostrowski)**.** *[1, p.22] Each non-trivial valuation on $\mathbb{Q}$ is equivalent to either the Euclidean absolute value, or some p-adic valuation, i.e. the only completions of $\mathbb{Q}$ are $\mathbb{R}$ or some $\mathbb{Q}_p$.*

So, in some sense, the study of the p-adic numbers is just as natural as the study of $\mathbb{R}$. This project, then, is a stubborn attempt to carry over intuitions we have for arithmetic and functions on $\mathbb{R}$ to concepts in the p-adic numbers. Before we try to force $\mathbb{Q}_p$ into our mental mold of $\mathbb{R}$, though, we will first discuss some more critical differences between the p-adics and the reals.

2.2 Power Series in the p-adics

Let $\{x_i\}$ be a sequence in $\mathbb{Z}_p$. We consider the series $\sum_{i=0}^{n} x_i$. Just like in $\mathbb{R}$, if we want to talk about the limit of the series as $n \to \infty$, we require $x_i \to 0$. But $x_i \to 0$ if and only if $|x_i|_p \to 0$, so the number of trailing zeros of elements of $\{x_i\}$ gets larger and larger! (In particular, the sequence $1, p, p^2, p^3 \ldots$ converges to 0 in $\mathbb{Q}_p$.)

Now let $\{x_i\}$ be a sequence in $\mathbb{Q}_p$ which converges to $x \neq 0$. Then $\{x_i\}$ is a Cauchy sequence, so for $\varepsilon > 0$, there exists N so $n, m > N$ implies $|x_n - x_m|_p < \varepsilon$. But $|x_n - x_m|_p$ small means $(x_n - x_m)$ has many trailing zeros. So if $x_n = \ldots a_2\, a_1\, a_0$ and $x_m = \ldots b_2\, b_1\, b_0$, we have $a_i = b_i$ for all $0 \leq i < -\log_p \varepsilon$.

As we shrink ε to 0, the upper bound on i grows, until eventually x_n and x_m have exactly the same number of trailing zeros. Hence, for sufficiently large m and n, $\mathrm{ord}_p(x_n) = \mathrm{ord}_p(x_m)$.

What we have shown (not entirely rigorously) is the first part of the following theorem:

Theorem. *[1, p.61] Let $\{a_i\}$ be a sequence in $\mathbb{Q}_p$. Then*

1. *If $\lim_{n\to\infty} a_n$ is some nonzero $a \in \mathbb{Q}_p$, there exists some $M \in \mathbb{N}$ such that $m > M$ implies $|a_m|_p = |a|_p$.*
2. *The series $\sum_{i=0}^{\infty} a_i$ converges if and only if $\lim_{n\to\infty} a_n = 0$.*

The second part of the theorem is true by the strong triangle inequality. For any integers $n < m$,

$$\left|\sum_{i=n}^{m} a_i\right|_p \leq \max\{|a_n|_p, \ldots, |a_m|_p\} \leq \max\{|a_i|_p : i > n\},$$

hence $\{\sum_{i=0}^{n} a_i\}$ is a Cauchy sequence.

This is very different from how series behave in $\mathbb{R}$. On one hand, the theorem means it is relatively easy to show that a series in $\mathbb{Q}_p$ converges at a point; on the other hand, it means the p-adic norm tells us very little about the limit of a convergent series.

As a result, though our favorite power series are defined the same way in $\mathbb{Q}_p$ as in $\mathbb{R}$, their domains of convergence are different. We will consider the following three series on $\mathbb{Q}_p$:

$$\exp(x) = \sum_{n\geq 0} \frac{x^n}{n!}$$

$$\sin(x) = \sum_{n\geq 0} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

$$\cos(x) = \sum_{n\geq 0} (-1)^n \frac{x^{2n}}{(2n)!}.$$

Each of these series converge only on the small disc, [1, p.70]

$$E := \{x : |x|_p \leq p^{1/(1-p)}\}.$$

In $\mathbb{Q}_p$, this set E is just $p\mathbb{Z}_p$ since the value group of the p-adic norm on $\mathbb{Q}_p$ is $\{p^n : n \in \mathbb{Z}\}$. This precise definition of E becomes more important when we extend these series to the p-adic complex numbers, $\mathbb{C}_p$, which have value group $\{p^r : r \in \mathbb{Q}\}$. We will not be talking further about the p-adic complex numbers here since the notion of writing p-adic numbers as two-sided sequences vanishes in $\mathbb{C}_p$, and we no longer have any intuition for what a plot may look like. ([2] was a good resource for understanding this and enough to convince me to stick to $\mathbb{Q}_p$ for now.)

3 Calculation in the p-adics

We now discuss some of the details and difficulties of doing computation in the p-adic numbers. We start by deciding on a sufficient measure of accuracy.

3.1 Cosets of $\mathbb{Z}_p$

Let us note some properties of the additive cosets of $p^k\mathbb{Z}_p$ and discuss why estimation up to coset is a reasonable way to classify the accuracy of calculations.

We write p in $\mathbb{Z}_p$ as $\ldots 010$, so it is natural that for any p-adic integer $\ldots a_2\, a_1\, a_0$, multiplication by p amounts to adding a zero on the right, just like multiplication by 10 in any base-n number system:

$$(\ldots a_2\, a_1\, a_0) \cdot p = \ldots a_2\, a_1\, a_0\, 0.$$

Then $p\mathbb{Z}_p$ is the subset of p-adic integers ending in 0; it is a maximal ideal of $\mathbb{Z}_p$, and the set of all p-adic numbers with norm less than 1 since every $x \in p\mathbb{Z}_p$ has $\mathrm{ord}_p(x) > 0$.

Likewise, $p^k\mathbb{Z}_p$ is the subset of p-adic integers ending in k zeros (also an ideal of $\mathbb{Z}_p$). The p^k additive cosets of $p^k\mathbb{Z}_p$ partition $\mathbb{Z}_p$ in the following way: If a is an element of $j + p^k\mathbb{Z}_p$, then the last k digits of a and j are the same.

Therefore, when we talk about "computing something up to p^k coset," we mean computing the last k digits accurately. For p-adic numbers which may have infinitely many nonzero, non-repeating digits, this sort of approximation is all we can hope to do with a computational model.

3.2 Images of $\mathbb{Z}_p$

Luckily, the cosets of $p^k\mathbb{Z}_p$ lend themselves nicely to being represented geometrically. Below is our first image of the p-adic numbers:

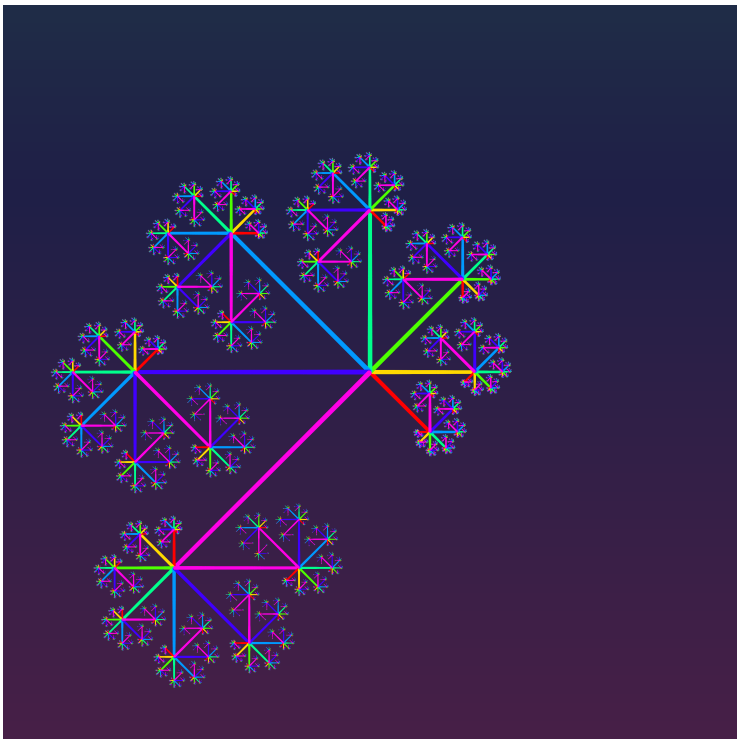

Figure 1: Plot of $\mathbb{Z}_7$.

Though we call this a plot of $\mathbb{Z}_7$, this diagram really shows all the cosets of $7^5\mathbb{Z}_7$ (a true plot of $\mathbb{Z}_7$ would recurse infinitely). We use colors to represent the digits of 7-adic numbers. For example, assign the digit 0 to the pink branches, 1 to the violet branches and so on, up to 6 for the red branches. If we want to represent the 7-adic number $\ldots 66510$ on this plot (or more accurately, the coset of 7-adic numbers $66510 + 7^5\mathbb{Z}_7$), we start from the center node, then follow the pink branch, then the violet, then yellow, red, red. With each node we jump to along the tree, we pass from a $7^k\mathbb{Z}_7$ to a $7^{k+1}\mathbb{Z}_p$ coset, and our estimation accuracy increases.

The program is able to produce a similar plot for other values of p with an arbitrary number of tree generations, and at higher image resolutions, but for values of p greater than 11 the tree begins to intersect itself.

Initially, the program created more "traditional" circular plots of $\mathbb{Z}_p$ where all the branches at each level had the same length (an image is included in the Appendix). Resolving the issue of self-intersection was then a matter of choosing an appropriate constant to scale the length of the branches on each generation. This problem is difficult to solve in general, though, and even after calculating such a constant for $p = 13$, I found the branches became too small far too quickly ([3] is a reference I used briefly when trying to implement this).

To avoid intersections and keep the branches visible, we make a plot which shows the $0 + p^k\mathbb{Z}_p$ coset largest and $(p^k - 1) + p^k\mathbb{Z}_p$ coset smallest. This is a logical choice since many of the power series we are concerned with have domains of convergence contained in $p\mathbb{Z}_p$ and images in $1 + p\mathbb{Z}_p$, so we will be able to make out more important details this way. (We still end up with some self intersections using this method—see the Appendix.)

3.3 Estimating Power Series up to Coset

Assuming we are able to do basic operations with p-adic numbers like multiplication and addition, approximating power series is easy. But how do we know when our approximation is accurate in the first k digits so we can plot its value? We use the following method:

Suppose we want to compute the value of the series $\sum_{n\geq 0} a_n x^n$ up to an error term in some p^k coset of $\mathbb{Z}_p$. Assume the series converges at x and that the sequence of norms of nonzero terms, $\{|a_n x^n|_p : a_n x^n \neq 0\}$, is non-increasing (this is the case for exp, sin, and cos).

We know $|a_n x^n|_p \to 0$, so there is some h for which $|a_h x^h|_p \leq p^k$. To simplify the computation, rewrite this as

$$\begin{aligned} k &\leq \operatorname{ord}_p(a_h x^h) \\ &= \operatorname{ord}_p(a_h) + h \operatorname{ord}_p(x), \end{aligned}$$

so the number of terms we need to compute is the minimal h which satisfies this inequality.

In the special case where $a_n = 1/n!$ as in exp, we apply the identity [4, p.15]

$\operatorname{ord}_p(n!) = n - s_p(n)$ where $s_p(n)$ is the p-adic digit sum of n:

$$\operatorname{ord}_p(x^n/n!) = n \operatorname{ord}_p(x) - s_p(n) + n.$$

Note that when $x = 0$, we define $\operatorname{ord}_p(x)$ to be ∞. There are limitations to implementing ∞ in C/C++, so the function for computing ord_p just returns the maximal value of `unsigned int`. To avoid the large integer multiplication (and potential overflow errors), we immediately return a_0 when evaluating a power series at $x = 0$.

A more subtle problem comes when we compute power series where some—but not all—coefficients are 0 like $\sin(x)$. In this case it is not sufficient to stop computing terms when one has order greater than k; we must also check that the term is nonzero.

3.4 Multiplicative Inverses: an Algorithm

A big computational hurdle of doing calculations in the p-adic numbers is finding multiplicative inverses. The general way to do this is long division, though I could not find anything but worked examples in my research. What follows is the algorithmic implementation I use:

Let $a \in \mathbb{Q}_p$. We will find a^{-1} to k digits of accuracy.

1. First convert a to a unit in $\mathbb{Z}_p$. Let $\tilde{a} = p^{\operatorname{ord}_p(a)}a$. Then $\tilde{a} \in \mathbb{Z}_p^\times$ and
$$\tilde{a}^{-1}p^{-\operatorname{ord}(a)} = a^{-1}$$

2. Since $(p, \tilde{a}) = 1$, use Fermat's Little Theorem to find x so
$$\tilde{a}x \equiv 1 \pmod{p}.$$
We take
$$x = (\tilde{a}^{p-2}) \,\%\, p.$$
Note: The symbol $\%\, p$ represents taking the integer remainder upon division by p, as in many programming languages. I avoid writing $\pmod{p}$ here to avoid ambiguity since it is important that x have as few (nonzero) digits as possible to reduce the computational complexity of multiplication by x.

3. Let $d_0, d_1 \ldots$ be the sequence of dividends. Initialize d_0 to $\ldots 001$.

4. Let w_i be the first nonzero digit of d_i (this will be the digit of d_i at index $\operatorname{ord}_p(d_i)$). Then the ith digit of the quotient is $(xw_i \,\%\, p)$. We set
$$d_{i+1} = d_i - (xw_i \,\%\, p)\tilde{a}.$$

5. Repeat (4) to obtain the desired number of digits. We then have
$$\tilde{a}^{-1} \approx \ldots (xw_k \,\%\, p)(xw_{k-1} \,\%\, p) \ldots (xw_1 \,\%\, p).$$

6. We return $a^{-1} \approx \tilde{a}^{-1} p^{-\operatorname{ord}(a)}$.

Below is a Python-style pseudo-code implementation of the algorithm:

```
a_tilde = a * (p ** ord(a))
x = (a_tilde ** (p - 2)) % p
d = ...001   # dividend
q = ...000   # quotient

for i in range(k):
    w = d[ord(d)]
    q[i] = (x * w) % p
    d -= q[i] * a_tilde

return q * (p ** -ord(a))
```

Notice that each loop iteration contains 2 multiplications, 1 subtraction, and 1 reduction mod p. Thus, the number of operations to compute a^{-1} to k digits is $O(k)$.

3.5 Complexity of Power Series Evaluation

Let us now consider the computational complexity of computing $\exp(x)$ to k accurate digits. Suppose x has ℓ nonzero digits, and h is minimal satisfying

$$k \leq \operatorname{ord}_p(x) - s_p(h) + h,$$

so we need to compute h terms of the power series.

Computing each $m!$ is done in $O(m)$ time here, and we have shown above that computing the inverse to k digits is done in $O(k)$ time. Thus, computing each coefficient $1/m!$ is $O(mk)$.

So the complexity of computing all the coefficients is

$$\begin{aligned}\sum_{m=0}^{h} O(mk) &= O\left(k \sum_{m=0}^{h} m\right) \\ &= O(kh^2).\end{aligned}$$

Once we have all the coefficients, we have $2h-1$ multiplications for computing each $a_r x^r$ (we store the value of x^r on the rth iteration to easily compute x^{r+1} on the next iteration) and $h-1$ additions, so the whole computation is done in $O(h\ell^2 + kh^2)$ time. Since we are only computing up to k digits, we only worry about at most k digits of x; the complexity becomes $O(hk^2 + kh^2)$.

Finally, note that $\operatorname{ord}_p(x) = c \log_p(x)$ for some coefficient c, so we get $O(k) = O(h \log_p(x) - \log_p(h) + h) = O(h)$. We conclude that the computation of $\exp(x)$ to k digits is done in $O(k^3)$ time.

Note that we use a few naive implementations here for simplicity. The implementation of p-adic–p-adic multiplication here is $O(\ell^2)$, but with more sophisticated algorithms, could maybe be improved. Integer–integer multiplication can be done as fast as $O(\ell \log \ell)$, though I'm not sure how this will translate to the p-adic datatype where each digit is stored as an element of an array.

We can also reduce the run time by storing the value of $1/(r!)$ on the rth iteration so we can easily compute $1/(r+1)!$ on the next iteration. For now this optimization is omitted in favor of a more adaptable power series interface in our program. We discuss this next.

4 Implementation

In this section, we will discuss in some more detail the design features of the implementation of the p-adic numbers. We will omit a detailed discussion about the actual p-adic "graphing" utility I made because, while complicated, there is not much mathematical to say about it. All the code and function headers are commented.

Here is a non-exhaustive list of some of the more important files and their contents:

`p_adic.h`: Templated class header for the p-adic numbers. Describes all the class features and member functions.

`p_adic.hpp`: Implementation of the p-adic class.

`p-adic_draw.cpp`: Implementation of graphing utility.

`main.cpp`: Handles terminal input and makes function calls.

4.1 Interface

The goal for this implementation of the p-adic numbers is to make a package that can easily be dropped into any other C++ program and work in an intuitive way. The class `p_adic<p>` takes an `unsigned` template parameter `p`, and will therefore immediately work for any prime p. An added benefit here is that the compiler will stop users from attempting operations between a p-adic number and q-adic number where $p \neq q$ since they have different types.

Operator overloading in C++ is also very helpful here and makes arithmetic between `p_adic<p>` objects feel very natural. The `+`, `*`, and `-` operators are each overloaded so, for example, if we have two instances of `p_adic<p>` called `a` and `b`, we can compute their sum with `a+b`.

Other common computations are implemented in low-friction ways. The order, norm, and inverse of `a` can each be computed with `a.ord()`, `a.norm()`, and `a.inv()`. The `[]` and `<<` operators are also overloaded so we can easily access digits of `a` and stream `a` to `stdout` or any other file.

The most interesting feature here is the ability to define and evaluate arbitrary power series. The general power series function looks like this:

```
p_adic<p> power_series(p_adic<p> x, unsigned k,
                       p_adic<p> a(unsigned));
```

To evaluate a power series at `x` to `k` accurate digits, all the user has to do is define a coefficient function `a()` which takes an index and returns a `p_adic<p>` object. This function can be passed into the graphing function and the code will generate a plot of the function.

4.2 Implementation Details

We have discussed how the computations are implemented already, so in this section, we will just mention how the `p_adic<p>` datatype is stored.

Since `p` is an arbitrary prime, the digits of `p_adic<p> a` can be arbitrarily large, so it does not make sense to store `a` as a single `unsigned` or other primitive datatype (this would also impose a pretty small limit on the size of `a`). We therefore use a `std::vector<unsigned>` to store the digits.

For p-adic integers, this design choice is intuitive, since the index i digit appears at index i in the `vector`. Generalizing this implementation to all p-adic numbers is not difficult, though; we just have to store a number `m` which indicates the index in the `vector` of the digit with index 0, i.e. `m` is the number of "decimal" places. For p-adic integers, this means `m=0`. For the 7-adic number $\ldots 014.2$, we have `m=1`, and so on.

This is all the data we have to store for a single `p_adic<p>` object, which makes the class lightweight.

5 Plots of Power Series

Finally, let us take a look at the results. What do the plots of p-adic power series actually look like?

Since we already use colors to indicate the cosets of $\mathbb{Z}_p$, the most practical way to show how the value of a power series like $\exp(x)$ varies with different inputs is to display one pair $(x, \exp(x))$ at a time and cycle through them. We therefore only show individual frames of the plots here. The full animations can be accessed on the project's GitHub. Note that since the final `.gif` files are each over 500 MB in size, only lower resolution versions are available online (they still capture the "motion" of the functions).

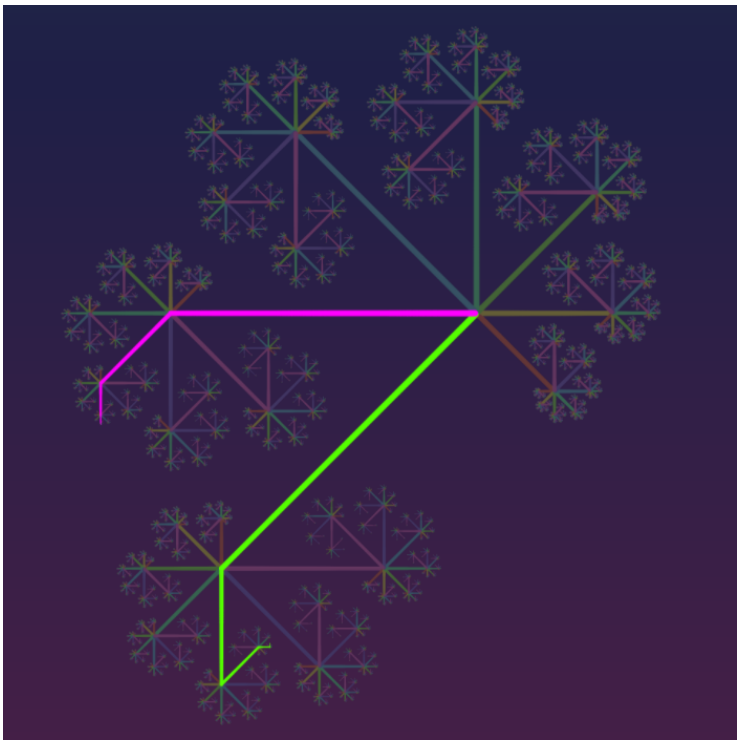

Figure 2: A plot of $\exp(x)$ (pink) on $\mathbb{Z}_7$. Watching the full animation of the plot, it is easy to see that for smaller values of x (green), the map $\exp(x)$ is close to $x+1$. For larger values of x, the pink branch occasionally flips around more rapidly, however it never strays too far from $x+1$ and never repeats values ($\exp(x)$ is a surjective isometry [1, p.80]).

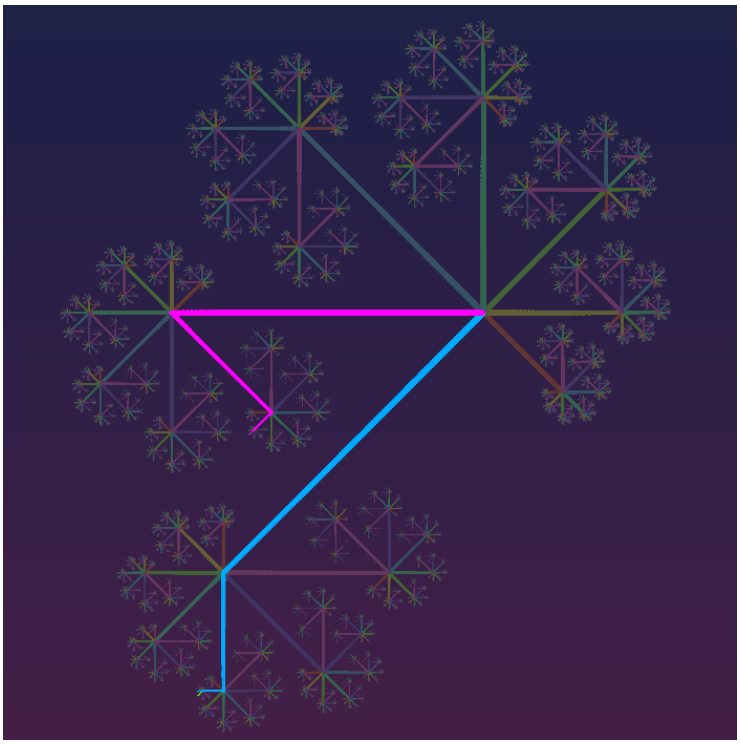

Figure 3: A plot of $\sin(x)$ (blue) and $\cos(x)$ (pink) on $\mathbb{Z}_7$. Note that x (green) is almost entirely covered by $\sin(x)$ in this picture. Since the power series for $\sin(x)$ does not have a constant term, every term in the sum is in $\mathbb{Z}_p$, an ideal of $\mathbb{Z}_p$, so $\sin(x)$ never "escapes" $p\mathbb{Z}_p$. The contribution of terms $a_n x^n$ is also very small for $n \geq 3$, so it makes sense that $\sin(x)$ is close to x.

Note that the series for $\cos(x)$ has only even powers of x and a constant term 1. Since x is always in $p\mathbb{Z}_p$, any even power of x is in $p^2\mathbb{Z}_p$, another ideal of $\mathbb{Z}_p$. As a result, $\cos(x)$ will always be in $1 + p^2\mathbb{Z}_p$. Visually, this means we never see the pink branch "escape" the $p^2\mathbb{Z}_p$ coset where it appears in this picture.

Thus, the images of sin and cos are disjoint in $\mathbb{Z}_p$. Interestingly though, it does appear that the two functions often make "jumps" between adjacent branches at the same time. A future problem will be to justify this observed behavior rigorously.

6 Appendix

6.1 Getting Started with the Code

Create a function `p_adic a(unsigned)` to define coefficients of a power series. Use this function with `power_series(...)` in `thread_process_image(...)` in `p-adic_draw.cpp`.

From the command line, run:

```
./image p_adic <width> <p> <children> [target_file_name]
```

(The frames we generated here are done with `./image p_adic 5000 7 5`)

This will take a while to generate each frame of the `.gif` (around an hour running 7 threads on my machine with an i5-9400F processor at 2.90 GHz, 6 cores). When its done, the images are stored in `./pngs/`. My $7^5 = 2401$ images at 5000×5000 pixels each take up as much as 1 GB. Compile and run `./rename_files.c` to prepare file names (this erases their descriptive names).

Convert the files to a `.gif`:

```
ffmpeg -i pngs/%04d.png out.gif
```

Optionally, resize the `.gif`:

```
ffmpeg -y -i out.gif -pix_fmt rgb8 -r 10 -s 380x380 out__low_res.gif
```

6.2 Other Iterations of the p-adic Plots

In this section, we include some images of previous iterations and styles of the plots of $\mathbb{Z}_p$.

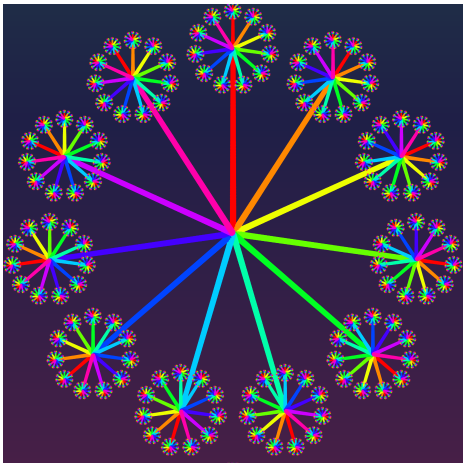

Figure 4: "Circular" plot of $\mathbb{Z}_{13}$. This style of plot was the first that I implemented and feels very traditional. It was ultimately abandoned since the branches become too small too quickly. It is also difficult to tell which branches indicate which coset because of the rotational symmetry.

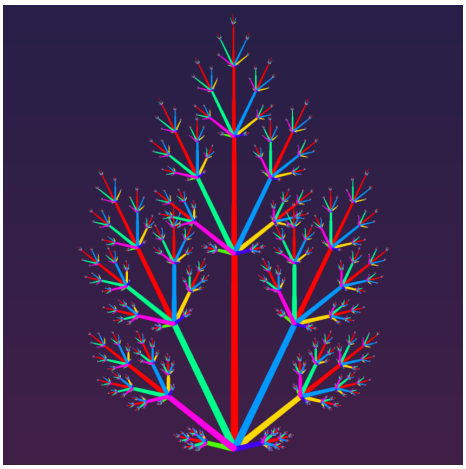

Figure 5: "Spikey" plot of $\mathbb{Z}_7$. This is the second style of plot I tried. This one looks pretty, but there is no mathematical reason to emphasize the "middle" coset of each generation; it is also very quick to intersect itself.

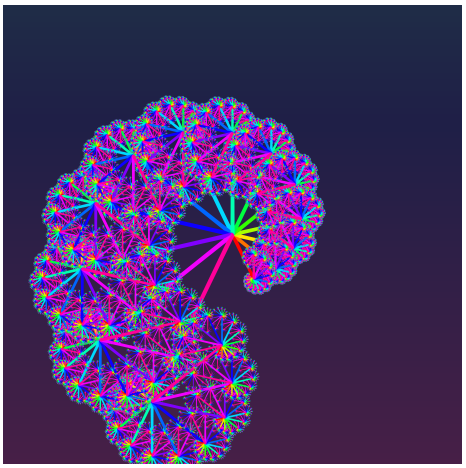

Figure 6: A self-intersecting plot of $\mathbb{Z}_{13}$ made in the final "swirly" style. While certainly visually striking, the self-intersections are too overwhelming to make out details. Creating a non-intersecting plot of $\mathbb{Z}_{13}$ is easy, though it would mean very tiny branches that are too small to see.

References

[1] W. H. Schikhof. *Ultrametric Calculus: an Introduction to p-adic Analysis*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2006.

[2] Daniel Litt (`https://mathoverflow.net/users/6950/daniel-litt`). How to picture $\mathbb{C}_p$? MathOverflow. `https://mathoverflow.net/q/51912` (version: 2018-07-28).

[3] L. Riddle. Classic iterated function systems: Symmetric binary tree. Agnes Scott College, 2021. `https://larryriddle.agnesscott.org/ifs/pythagorean/symbinarytree.html`.

[4] K. Conrad. Infinite series in p-adic fields. University of Connecticut, 2020. `https://kconrad.math.uconn.edu/blurbs/gradnumthy/infseriespadic.pdf`.