A class of functions defined by replacing digits of natural numbers in arbitrary bases

Sebastian Karlsson

December 2022

Contents

1 Introduction 1

2 Arbitrary bases and arbitrary digital roots 1

3 The function 2

4 Plots 2

A Python code for the graphs 6

1 Introduction

The main objective of this paper is to show how mathematics can be used in a simple and playful way to construct art. We will investigate a class of functions defined on the natural numbers whose main interest lies in the striking patterns that can be found in their graphs.

We let f_b be the function that takes a number n represented in base b and replaces each digit d of n with the base b digital root of $d \cdot n$. We will try to make the meaning of this definition clear.

2 Arbitrary bases and arbitrary digital roots

We normally represent our numbers in base 10. This means that we use 10 different digits, $0, 1, ..., 9$. When we say that, for example, $n = 871$, what we actually mean is that

$$n = 8 \cdot 10^2 + 7 \cdot 10^1 + 1 \cdot 10^0$$

In general, if the decimal digits of a number n are $d_m, d_{m-1}, ..., d_1, d_0$, then this means that

$$n = d_m \cdot 10^m + d_{m-1} \cdot 10^{m-1} + ... + d_1 \cdot 10^1 + d_0 \cdot 10^0$$

Every integer can uniquely be represented in this manner - something we often take for granted when we use mathematics in our everyday life. In fact, this property of numbers is not special for 10. We can do this for any base b ([1]). In the base b representation we use b different digits, say $a_0, a_1, ..., a_{b-1}$, such that a_k represents the number k, where $k = 0, 1, ..., b - 1$. Every number n has then a unique representation in base b of the form

$$n = d_m \cdot b^m + d_{m-1} \cdot b^{m-1} + ... + d_1 \cdot b^1 + d_0 \cdot b^0$$

Where $d_0, d_1, ..., d_m$ are among the digits we are using and d_m is nonzero. We signify with a lower index written in base 10 that a certain string of digits is the representation of a number in a base other than 10. For example, 101_2 is the representation of 5 in base 2 (binary) because $5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$.

The digital root of a number is the single digit one obtains after repeatedly adding the digits together. For example, the digital root of 871 is 7 because $8 + 7 + 1 = 16$ and $1 + 6 = 7$. Similarly, the digital root of a number n in base b is the number one obtains after repeatedly adding together the base b digits of n. We denote the digital root of n in base b with $dr_b(n)$.

Let's take an example of this in another base. In base 12 we use the digits $0, 1, ..., 8, 9, A$ and B. For example, $11 = A_{12}$, $12 = 10_{12}$ and $23 = 1A_{12}$. Now, the base 12 digital root of 23 is 1 because $1_{12} + A_{12} = 10_{12}$ and $1_{12} + 0_{12} = 1_{12} = 1$.

One could wonder whether there is a concise mathematical formula for the digital root. In fact, it is a theorem ([2]) that

$$dr_b(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 + (n-1) \bmod (b-1) & \text{otherwise} \end{cases}$$

where "mod" stands for the remainder function ([3]).

3 The function

We are now in position to define the class of functions this paper is concerned about. Let $d_m, d_{m-1}, ..., d_1, d_0$ be the base b representation of the non-negative integer n. Then

$$f_b(n) = dr_b(n \cdot d_m) \cdot b^m + dr_b(n \cdot d_{m-1}) \cdot b^{m-1} + ... + dr_b(n \cdot d_1) \cdot b^1 + dr_b(n \cdot d_0) \cdot b^0$$

In base 10 this gives rise to the integer sequence; 0, 1, 4, 9, 7, 7, 9, 4, 1, 9, 10, 22, 36, ..., which can be found in the online encyclopedia of integer sequences ([4], [5]). Let us again take an example: Let's calculate $f_{10}(12)$. We see that $1 \cdot 12 = 12$ and the digital root of 12 is 3. Furthermore, $2 \cdot 12 = 24$ and the digital root of 24 is 6. Hence, $f_{10}(12) = 36$.

4 Plots

Of course, this is a rather arbitrary and random definition of a function. It is also not evident what the point is. However, sometimes it is nice to just play around with irrelevant and useless definitions: They can lead one to something interesting.

Let us try to plot these functions. To do that, we implement the function in Python and use the matplotlib-library to create a graph. The python code can be found in the appendix.

Having the natural numbers on the x-axis and the value of f_b on the y-axis, we notice that we get a square-shape consisting of a lot of smaller intricate patterns. We also notice that the squares are located on the intervals $\{b^m + 1, b^m + 2, ..., b^{m+1}\}$ for each natural number m. Furthermore, for large m it seems like the pattern is more or less the same on $\{b^m + 1, b^m + 2, ..., b^{m+1}\}$ as on $\{b^{m+1} + 1, b^{m+1} + 2, ..., b^{m+2}\}$. Proving that this is true for all bases b (or just one base) could be a challenge for further research.

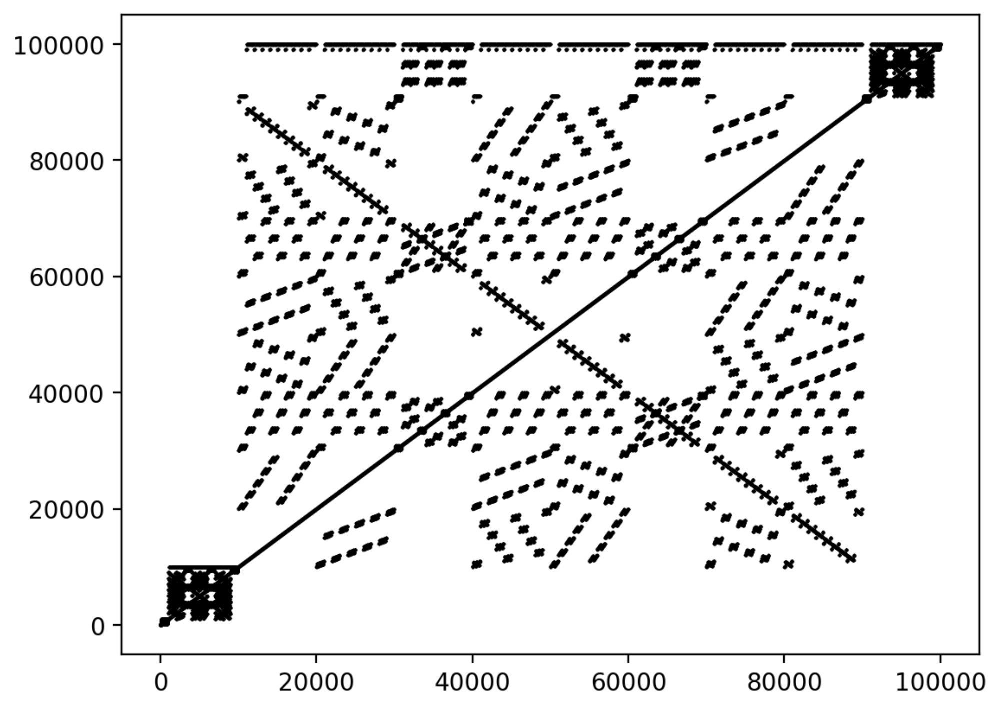

Figure 1: f_{10} on the interval $\{0, 1, ..., 10^5\}$

.

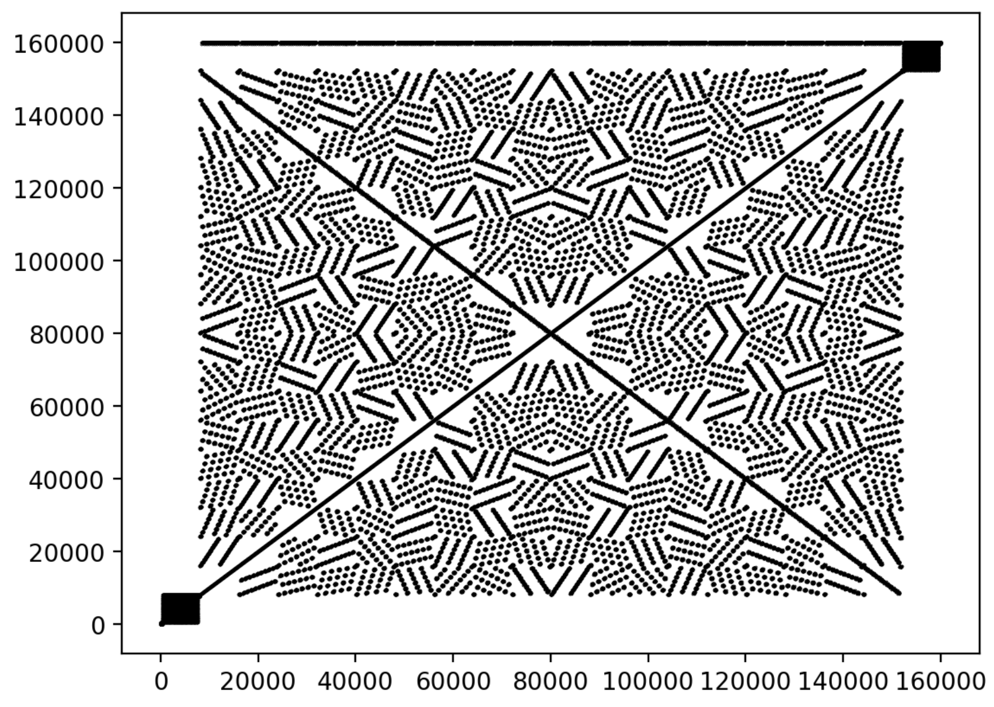

Figure 2: f_{20} on the interval $\{0, 1, ..., 20^4\}$. Note that the numbers on the axes are written in base 10.

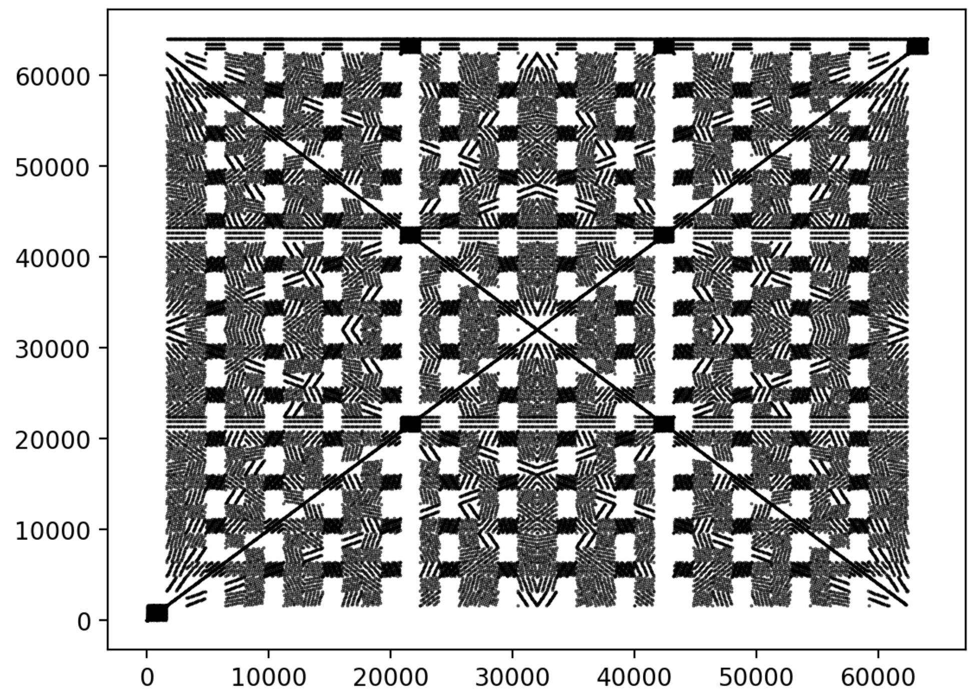

Figure 3: f_{40} on the interval $\{0, 1, ..., 40^3\}$. Note that the numbers on the axes are written in base 10.

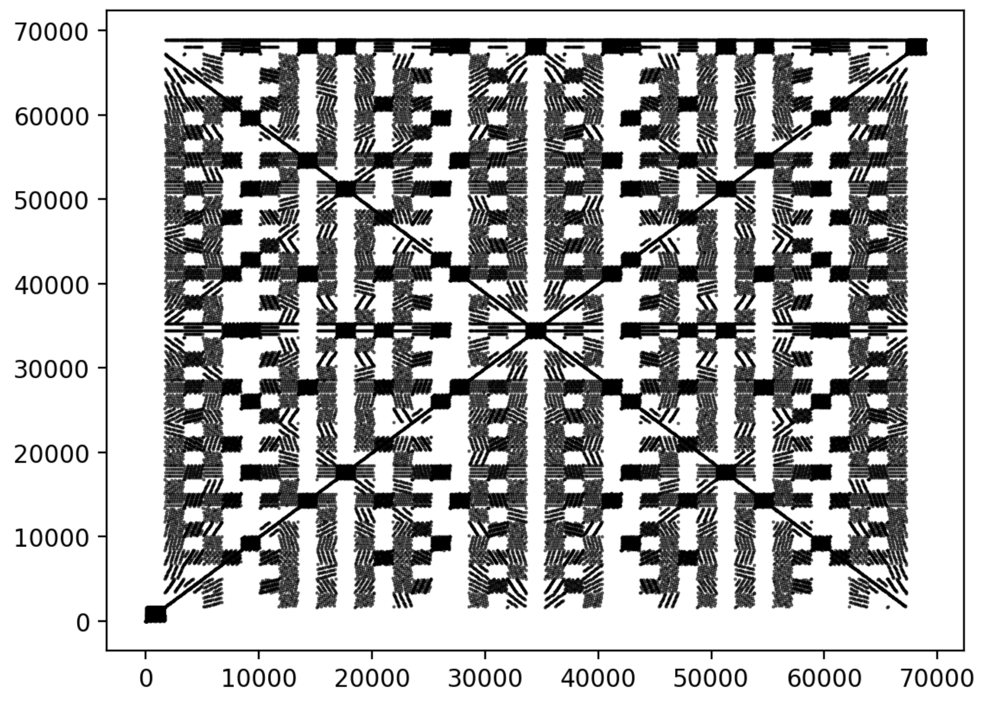

Figure 4: f_{41} on the interval $\{0, 1, ..., 41^3\}$. Note that the numbers on the axes are written in base 10.

References

[1] *Positional notation*. https://en.wikipedia.org/wiki/Positional_notation. Accessed 04.12.2022.

[2] *Digital root*. https://en.wikipedia.org/wiki/Digital_root. Accessed 29.11.2022.

[3] *Modulo operation*. https://en.wikipedia.org/wiki/Modulo_operation. Accessed 29.11.2022.

[4] *OEIS Foundation Inc. (2022), The On-Line Encyclopedia of Integer Sequences*. https://oeis.org/.

[5] Sebastian Karlsson. *OEIS Foundation Inc. (2022), The On-Line Encyclopedia of Integer Sequences*. A339023. Replace each digit d in the decimal representation of n with the digital root of n*d.

A Python code for the graphs

```python
import matplotlib.pyplot as plt
from functools import reduce

# Convert the number n to a list of digits in base b.
def to_base(n, b):
    return [n] if n < b else to_base(n//b, b) + [n%b]

# Convert a list of digits in base b to decimal.
def to_dec(l, b):
    return reduce(lambda x, y: b*x + y, l)

# The digital root in base b.
def dr(n, b):
    return 0 if n == 0 else 1 + (n-1)%(b-1)

# The function replacing digits.
def f(n, b):
    return to_dec([dr(n*d, b) for d in to_base(n, b)], b)

if __name__ == "__main__":
    base = int(input("Base: "))
    upper = int(input(f"Exponent of {base} for the upper bound: "))
    file_name = input("Enter filename: ")

    X = range(0, base**upper)
    Y = [f(x, base) for x in X]

    fig = plt.figure()
    plt.plot(X, Y, ".k", markersize=0.6)
    plt.show()
    fig.savefig(f"{file_name}.png", dpi=200, bbox_inches="tight")
```