On calculating Friedman number formulas

Ron Kaminsky*

October 13, 2015

Abstract

Heuristic arguments are given to justify the hypothesis that $f_b(n)$, the fraction of Friedman numbers to the base b among the positive integers less than n, is very close to 1 even for n with modest numbers of digits (hundreds or more), independent of b. The arguments immediately provide a practical approach to generating computational results which support the hypothesis.

1 Introduction

An integer n is a Friedman number to the base b if n can be non-trivially represented using the digits of its representation to the base b and the operations of unary minus, addition, subtraction, multiplication, division, exponentiation, and the joining of a set of digits into the representation of an integer to the base b (without leading zeros). Grouping of results using parentheses is also allowed. For example, 128 is a Friedman number to the base 10 since

$$128 = 2^{8-1}$$

and 1250 is a Friedman number to the base 8 since

$$1250_8 = 5_8 * 210_8 \, .$$

The seminal work on Friedman numbers can be found on the web[1]. Located there is the following conjecture due to Friedman:

> If $F_{10}(n)$ is the number of Friedman numbers to the base 10 among the integers $\{1, 2, \ldots, n\}$ then
>
> $$\lim_{n\to\infty} \frac{F_{10}(n)}{n} = 1 \, . \tag{1}$$

*Contact info: my email address ID at gmail.com is the name above, without spaces.

This conjecture was proven by Brand and published in 2013[2]. The mechanism of proof used there, albeit rigorous and elegant, does not shed much light on the density of Friedman numbers among numbers likely to be amenable to computation by humanity in the near future. This work presents non-rigorous arguments which indicate that it is probable that the density of Friedman numbers among numbers with hundreds or more digits is also very close to one, and in addition yields a practical method for computing Friedman formulas for large numbers.

2 Basis

The justification of (1) which comes immediately to mind is based on combinatorics, not number theory[3]. In general, the number of formulas which can be generated by using the k digits of the number n and the seven allowed operations grows much larger than 10^k as k increases. This in and of itself is not a very good justification for several reasons.

First of all, many of the formulas may include invalid operations, such as division by zero, exponentiation of zero by zero, or the joining of digits to form a number with leading zeroes. The number of valid formulas constructable from k zeroes is much smaller than the number constructible from k ones.

Second, even if a formula consists of only valid operations, its value may well be non-integral because of the presence of division and the possibility of exponentiation to a negative or non-integral power.

Third, even if the number of formulas which evaluate to integers exceeds 10^k, there is no guarantee that many of their values may not be identical, yielding many less distinct integral formula values. In the case of k zeroes, for example, all such valid formulas evaluate to zero.

And fourth, the range of the formula values is much larger than $[0, 10^k]$ because of the presence of exponentiation. Therefore it is not clear that any great part of the integers in $[0, 10^k]$ are among the values.

Notwithstanding the problems with this preliminary argument, the conjecture has been proven correct and will be justified here without rigor using a somewhat different line of reasoning than given above or previously used in [2]. Although the following discussion will address the problem for Friedman numbers to the base 10 only, the arguments certainly generalize to all larger bases, and probably work for all bases. For the case of smaller bases the practicality of computation is unchecked, however.

3 A possible justification of the conjecture

The (non-rigorous) justification will be presented as a series of observations.

3.1 Combinatorial explosion happens

If we consider all of the formulas constructable from any k digits (not just those of a particular number n) and examine the behavior of $g(k, M)$, defined as the number of distinct integral formula values generated in the interval $[0, M]$, then when $M \gg 10^k$, we find that $g(k, M)$ grows faster than 10^k. This was checked for $k \leq 5$ and $M = 2^{665} \approx 10^{200}$ (where the formulas are constrained to only those with integral intermediate results). The results can be seen in the following table.[1]

Digits	Values	Useful Values
1	10	0
2	133	43
3	2983	2135
4	76524	69926
5	2098860	2084052

Table 1: Table of formula value statistics. (The "Useful Values" column counts only formula values which might possibly generate digit profit when used as the divisor of a product-sum representation).

3.2 Remainder is distributed uniformly

The remainder left on division of a large number N by a divisor d can be presumed to be uniformly distributed on $[-d/2, d/2]$ if there is no a priori relationship between N and d, and $N \gg d$. The probability that the remainder r of the division of N by d will have m less digits than d is approximately $4 \cdot 10^{-m}$ (averaged over subsets with a fixed number of digits).

3.3 Some product-sum representations will be profitable

If we consider each of the distinct formula values of section 3.1 as a trial divisor of a large number N, then given the probability estimate of section 3.2 we should be able to find among the formulas using k digits, for large enough k, values d such that

$$\#(N) > \#(\frac{N-r}{d}) + k + \#(r)$$

where $\#(n)$ stands for the number of digits in n's decimal representation, and r is the remainder (in $[-d/2, d/2]$) of the division of N by d. Any such product-sum representation

$$N = q\Phi + r, \ (q = \frac{N-r}{d})$$

[1] There exist small discrepancies between Table 1 (which was generated years ago using C++) and the results generated by the newer software written in Julia. These discrepancies are small enough that they do not endanger any of the arguments stated here, and can possibly be caused by the Julia version using a more stringent comparison with M (the C++ version used approximate comparison of logarithms).

where Φ stands for the formula generating d will be said to generate a "digit profit" of size

$$\#(N) - \#(q) - k - \#(r) \, .$$

A hidden assumption here is that each of the trials is independent, and this is not really correct since if two formula values are correlated their remainders are correlated in some fashion, just not in a way which is likely to cause the digit distributions to be (highly) correlated. In practice, however, by using the formula values of formulas on 5 or less digits with values less than approximately 10^{200} it is possible to generate an average digit profit of approximately 1.1 digits per product-sum representation.

3.4 Product-sum representation can be applied iteratively

The quotient q of the profitable product-sum representation of section 3.3 still has at least close to $\#(N) - \#(M)$ digits and can be itself represented in the same fashion. By repeating this process it is possible to generate formulas for large numbers which have digit profits of size linear in the number of digits of said numbers. Such a formula is not necessarily a valid Friedman formula for the large number, since the profit is a total value over the 10 digits $\{0, 1, \ldots, 9\}$ and there may well be one or more digits which occur more frequently in the formula than in the number being represented. Even if this is not true, the left-over numbers not used in the formula must be absorbed somehow without changing its value.

3.5 Enough left-over digits can always be absorbed

Most sets of 3 decimal digits, and any set of 4 or more decimal digits can be absorbed into a formula without changing its value. In addition, any set of left-over digits can be absorbed into a formula which contains at least one number whose value is 1.

3.6 Digit profit grows faster than digit frequency deviations

The average deviations of the frequencies of single digits in a randomly chosen large number N (with a given number of digits) from the expected value of $\#(N)/10$ grow according to $\sqrt{\#(N)}$. In contrast, the digit profit we can generate using iterative product-sum representation grows linearly in $\#(N)$. This means that for large enough numbers it is almost certain that we can generate a Friedman formula using product-sum representation.

This argument of course rests heavily on the assumption that the average digit distribution of the formulas we are using to generate the profit is itself uniform. While it seems reasonable that the set of profitable formulas could be chosen to have a uniform digit distribution, in the limit as the number of digits in the formulas increases, for formulas of 5 digits or less it is demonstrably untrue.

As we will see later in section 4.1, it is possible to reuse the idea of product-sum representation to compensate for the imbalance generated by using an unbalanced fixed set of profitable formulas.

3.7 Independent of b

It should be noted that all of these points can be generalized to the case of Friedman numbers to an arbitrary base b. For very small bases, the minimum number of digits in the trial formulas used to generate profit will probably be larger (even in a relative sense), which may affect the practicality of doing actual calculation.

4 Practical algorithms for computing Friedman formulas

The heuristic justification of section 3 immediately implies that the following algorithm can be used to generate a Friedman formula of a large number. The simplest algorithm would seem to be:

Algorithm StraightRPSR

```
quo = <large number to be proven Friedman>
while (<quo is larger than M, the upper bound
        on the divisor formula values>)
{
   for f in divisor formulas
   {
      if (<the product-sum representation using
           f yields a Friedman formula>)
         <stop - the algorithm succeeds!>
      else if (<the product-sum representation using
                f yields digit profit>)
         quo = the new quotient of the product-sum representation
   }
}
<stop - the algorithm fails>
```

Even here it is not clear how the product-sum representation is chosen — there may very well be several product-sum representations with digit profit. The working heuristic examines all of the divisor formulas, and chooses the one which generates the most profit relative to the decrease in the number of digits in "quo".

It is, of course, possible to imagine much more sophisticated algorithms, where the number is initially broken into several independent "pieces" using unprofitable product-sum representations: for example, in the form of a binary tree. This could greatly aid parallelization of the computation, and enable the use

of much more powerful optimization algorithms. The problem in this case becomes a multi-dimensional multiple-choice knapsack problem (more or less), which multiple researchers have shown (in various variations) to be in general NP-hard without the possibility for a fully polynomial-time approximation scheme. However, in this particular case, particular features of the problem, namely those of the (presumably) random distribution of vectors of the digit frequencies, could likely enable the problem to be efficiently solved in practice.

4.1 Cleanup

Algorithm StraightRPSR will often fail because of disparity between the random non-uniformity in the digit distribution produced in the generated formula versus the original number, and also because of the non-uniformity of the divisor formula digit distribution. The vector of digit frequency differences for digits which occur more often in the generated formula than in the original integer will be called the digit deficit. An additional heuristic can often reduce this deficit to zero, greatly improving the success rate of the algorithm.

The remainders generated can themselves be represented using product-sum representation. Some of them may be too small for this to generate profit, but even unprofitable representations can decrease the total amount of digit deficit.

The working heuristic used for cleanup is to sort all the numbers used in the formula at the end of StraightRPSR in decreasing order, and sequentially represent them using product-sum representation using the divisor formulas (actually, for cleanup there is no pressing need for profit to be generated, so even formulas which cannot possibly generate profit can be used). At each step the representation which best decreases the total amount of digit deficits is chosen (a simple greedy algorithm). For 1600-digit numbers, this heuristic almost always will succeed to reduce the digit deficit to zero, after which it only remains to absord the surplus digits.

The choice to use product-sum representation is, of course, one of convenience: any form of representation which changes the digit distribution without changing the value of a sub-part of the formula could be used in the cleanup step, as long as it does not increase the number of digits by too much.

In addition, if the cleanup rerepresentations are independent of each other, it is again possible to use much more powerful optimization algorithms, exactly as mentioned in the previous section.

5 Applicability to orderly (nice) Friedman numbers

One could imagine attempting to adapt the justification of Section 3 to the case of orderly (also known as "nice") Friedman numbers[1], where the order of digits in the formula must match the order of digits in its value, by using product-sum representations of the form

$$N = \Phi q + r$$

instead. Brand has shown that for bases 2, 3, and 4, orderly Friedmans are also dense[4]. It is interesting to note that in the case of using product-sum representation, Section 3.1 gives rise to exactly the same considerations which are discussed there in Theorem 1, which means this cannot succeed for any bases greater or equal to 28.

6 The anti-Friedman conjecture

Let an "anti-Friedman" number be defined as a positive integer which is not a Friedman number in any integral base (Trevor Green invented this concept[1] without naming such numbers). The results presented here indicate that, in the limit, only integers whose digit frequency distributions are sufficiently atypical can fail to be Friedman numbers. This makes it almost certain that there are only a finite number of anti-Friedman numbers.

I offer a bounty[5] of US$200 for a rigorous proof that there are only finitely many anti-Friedman numbers, and US$2000 to anyone who can rigorously calculate the largest anti-Friedman number, or prove that such a number does not exist.

7 Summary

The technique of product-sum representation using an appropriately chosen set of trial divisor formulas is very powerful, and enables the computation of Friedman formulas for large random numbers. The technique may be applicable to other problems involving either Friedman numbers or formula design.

Iterated digit-profitable product-sum representation provides a non-rigorous justification of why almost all numbers of hundreds or more digits are Friedman numbers, and a conjecture about anti-Friedman numbers was proposed.

8 Acknowledgements

The original software used to confirm the algorithmic concept was developed in C++ using LiDIA's bigint class[6], and run using computer time graciously donated by Orbotech, Ltd. The current, much more concise implementation is written in Julia[7] and hosted on Github at URL `https://github.com/RonKaminsky/ff-compute` .

References

[1] Erich Friedman, URL `http://www2.stetson.edu/~efriedma/mathmagic/0800.html`.

[2] Michael Brand, "Friedman numbers have density 1", Discrete Applied Mathematics, 161(16–17), Nov. 2013, pp. 2389-2395. doi:10.1016/j.dam.2013.05.027 .

[3] Rafi Brada, personal communication, circa Dec. 1998.

[4] Michael Brand, "On the density of nice Friedmans", arXiv:1310.2390 [math.NT], http://arxiv.org/abs/1310.2390

[5] Details of the bounty conditions can be found at https://sites.google.com/site/ronkaminsky/the-anti-friedman-conjecture-bounties

[6] LiDIA version 1.3.1, developed at TU Darmstadt, Darmstadt, Germany.

[7] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman, "Julia: A Fast Dynamic Language for Technical Computing", arXiv:1209.5145 [cs.PL], http://arxiv.org/abs/1209.5145.