Seemingly Remarkable Mathematical Coincidences Are Easy to Generate

William H. Press
The University of Texas at Austin

June 1, 2009

1 Introduction

Gelfond's constant e^π can be expressed as the "equation"

$$e^\pi = 17 + \sqrt{3} + \sqrt[4]{172} + \log\log 9 \tag{1}$$

But, unsurprisingly, equation (1) is a fake: its two sides differ numerically by about 2×10^{-14}, as can readily be checked in a computer language that implements arbitrary precision arithmetic, for example *Mathematica*.[1] Similarly bogus, though also accurate to better than 2×10^{-14}, is

$$\pi = \left[\left(7 + \log\frac{7}{12}\right)^{1/3} + 1\right]^4 - 64 \tag{2}$$

or, accurate to about 2×10^{-16},

$$\pi = \left(\frac{12}{\log 889}\right)^2 + e^{-4} \tag{3}$$

or

$$\frac{1}{\pi} = \frac{3\sqrt{3}}{10}\left(5 - \sqrt{\frac{10}{13 - 6\sqrt[3]{9}}}\right) \tag{4}$$

whose error is only about 2×10^{-19}, several orders of magnitude smaller than the inherent error of most computers' IEEE double-precision arithmetic.[2]

As far as is known, equations (1)–(4) are purely numerical coincidences. As such they are, in some ways, less remarkable and less interesting than the mundane

$$\pi \approx 22/7, \quad \text{or} \quad \pi \approx 355/113 \tag{5}$$

which at least derive from a mathematical property of π, namely its continued fraction expansion. What *seems* remarkable about equations (1)–(4), and similar coincidental near-equations, is their blind, stubborn accuracy.

How hard is it to find such coincidences? Must one try on the order of 10^{19} formulas before getting one as good as equation (4)? Not at all. The standard cryptographic technique of "meet-in-the-middle attack"[3, 4] yields a computational complexity of order the square root of the inverse precision, requiring also about this much memory. So coincidences as good as equation (4) can readily be found in $O(10^9)$ operations on an ordinary desktop machine with $O(10^9)$ memory. The rest of this note gives the details of one such implementation.

2 Meet-in-the-Middle Attack

Suppose we generate at random a large number N of well-formed symbolic expressions and evaluate them numerically. As we will see below, it is not hard to arrange for the values to be roughly log-uniformly distributed over about $\Delta \sim 5$ e-folds, so the average density ρ of values per e-fold is

$$\rho = N/\Delta \tag{6}$$

If the values are independent and identically distributed (i.i.d.) random, then the logarithmic difference between consecutive values ϵ is exponentially distributed with c.d.f.

$$P(\epsilon < \epsilon_0) = 1 - \exp(\rho\epsilon_0) \approx \rho\epsilon_0 \tag{7}$$

Thus, among N (or, strictly, $N-1$) independent differences, the smallest difference should be on the order of

$$\epsilon_{\min} \sim \frac{1}{\rho N} \sim \frac{\Delta}{N^2} \tag{8}$$

If we repeat the entire process M times and take the smallest difference found, we should expect

$$\epsilon_{\min} \sim \frac{\Delta}{MN^2} \tag{9}$$

The above model makes two incorrect assumptions: First, it assumes that the N expressions are algebraically (and not just symbolically) distinct. This is wrong, because there are many equivalent ways of writing the same algebraic expression, so the effective value of N is reduced. Second, the model assumes that the generated values are i.i.d. This is wrong because expressions are composed of sub-expressions that may be common to many different expressions. There is thus a complicated structure of conditional dependency that generates clumpiness among the values. The effective value of ρ is thus increased. Figure 1 illustrates the clumpiness of expression values by plotting histograms of the counts of numerical values of random expressions at three different scales of resolution. Comparable fluctuations, hugely in excess of Poisson, are seen at the different resolutions.

As regards $\epsilon_{\min}$, the two effects act in opposite directions. Experimentally, they seem to be of comparable importance (within a factor 30, say), so that the estimate of equation (9) is also good to about a factor of 30. Our experiments have $\Delta \sim 5$, $N \sim 10^8$, $M \sim 10^2$, predicting $\epsilon_{\min} \sim 5 \times 10^{-18}$. In practice, we typically see 10^{-16} or 10^{-17}. Equation (4) is apparently just unusually lucky.

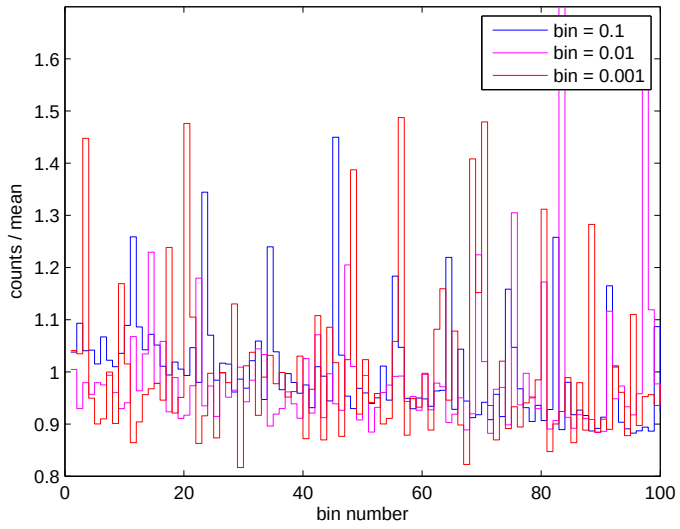

Figure 1: Histograms of the counts of numerical values of random expressions on three scales. The blue histogram spans the range 16.18 to 26.18; magenta, 16.18 to 17.18; red 16.18 to 16.28. Comparable fluctuations in counts, much larger than Poisson, are seen on all three scales.

3 Random Well-Formed Expressions

Here we discuss in more detail what we mean by a random symbolic expression. There is a lot of arbitrariness in our choices, so this section should be seen as an example, not a general discussion.

We consider expressions as being made from these atoms: numbers consisting of non-zero positive integers (e.g., 1 to 13), e, and π; the binary operations $+$, $-$, $\times$, and $\div$; and the unary operations $\sqrt{\ }$, $\sqrt[3]{\ }$, $(\quad)^2$, log, exp, and sin. (We will be able to restrict to a subset of the operations if desired.) Each atom is assigned a one-character representation, as follows:

atom	1	2	3	4	5	6	7	8	9	10	11	12	13
char	1	2	3	4	5	6	7	8	9	a	b	c	d

e	π	$+$	$-$	$\times$	$\div$	$\sqrt{\ }$	$\sqrt[3]{\ }$	$(\quad)^2$	log	exp	sin
e	p	+	−	*	/	q	t	Q	L	E	S

A string of characters is interpreted as reverse-Polish notation for an expression,[5] that is, like an RPN scientific calculator with a push-down stack for intermediate results. For example,

$$\text{“7p+qdt/”} \quad = \quad \frac{\sqrt{7+\pi}}{\sqrt[3]{13}} \tag{10}$$

Not all strings are well-formed. A string can fail to be well-formed with one of three exceptions: (1) A unary operator is encountered when the stack is empty. (2) A binary operator is encountered when the stack has fewer than two values. (3) The string ends with more than one value on the stack.

We generate strings from left to right, adding one character at a time. Exception (1) is obviated by a special rule for the first character. A convenient, though highly non-unique, way of obviating the other two exceptions is to specify as a parameter the number n_{bin} of binary operations that will be in the completed string. Then, when adding a character, we choose it to be a binary operation with probability

$$P_{\text{bin}} = \max\left(0, \min\left(1, \frac{i_s - 1}{n_{\text{bin}} - i_{\text{bin}}}\right)\right) \tag{11}$$

where i_s is the current stack size and i_{bin} is the number of binary operations already in the string.

A complete specification of the generative model is now obtained by specifying these additional parameters: P_{un}, the probability of a unary operation (versus a number); a vector of probabilities, summing to 1, for choosing among numbers; likewise, a vector for unary operations; and a vector for binary operations. Rather than taking n_{bin} as a fixed constant, we take it as being Poisson distributed with mean μ_{bin}, so this parameter replaces n_{bin}.

4 Details, Details

4.1 "Aiming"

Like Humpty-Dumpty's assertion on the meaning of words,[6] a unary or binary operation can mean "just what we choose it to mean". That is to say, whenever we encounter an operator, we already have on the stack the values of its operands. We can use this fact to advantage. For example, we can maintain positivity by redefining the subtraction operator:

$$a - b \equiv \begin{cases} a - b & \text{if } a > b \\ a + b & \text{if } a = b \\ b - a & \text{if } a < b \end{cases} \tag{12}$$

Note that this redefinition can be substituted both numerically and *symbolically* when a string is evaluated. Similarly,

$$\log a \equiv \begin{cases} \log a & \text{if } a > 1 \\ a & \text{if } a = 1 \\ -\log a & \text{if } a < 1 \end{cases} \tag{13}$$

Further redefinitions avoid excessively large or small results:

$$\exp a \equiv \begin{cases} \exp a & \text{if } |a| < 5 \\ a & \text{otherwise} \end{cases} \tag{14}$$

$$a^2 \equiv \begin{cases} a^2 & \text{if } 0.01 < a^2 < 100. \\ a & \text{otherwise} \end{cases} \tag{15}$$

In general, the purpose of these redefinitions is to "aim" the value of the expression into a limited (logarithmic) range of positive values, and thus to increase the density of values in that range. We are allowed to do this precisely because we don't really care what the expression is! We only care if it is close to another, different, expression.

In practice, we obtain a roughly log-normal distribution of expression values, with a standard deviation on the order of a factor of 10. Because memory is a scarce resource for meet-in-the-middle, we discard (don't store) expressions whose values lie more than one standard deviation (logarithmicaly) from the mean. Typically this results in keeping values between about 1 and 100.

4.2 Tautology Filtering

Most near-coincident expression values are simply algebraic tautologies, for example,

$$\begin{gathered} \text{“}\texttt{5L2L+E}\text{”} = \text{“}\texttt{45*2/}\text{”} \\ \exp(\log 5 + \log 2) = 4 \times 5/2 \end{gathered} \tag{16}$$

Unfortunately, because of roundoff error, these are not generally exactly equal computationally. There are two possible ways to proceed:

(1) We can reject as "probably tautological" coincidences that are close enough to be roundoff differences only. In this case, we will be unable to find by secondary processing (see below) coincidences that are near, or smaller than, the machine's roundoff precision, on the order of 10^{-15} for 64-bit `double`. A mitigation strategy would be to do all calculations in 128-bit `long double`, which hardly exists in current compilers and is very slow in current processors.

(2) We can use additional special-case information to reject tautologies. For example, equations (1)–(4) were found by the special rule "one expression must contain exactly one π, and the other must contain no π's". This is not perfect, because a small number of expressions will contain a π in a negligibly small sub-expression. But in practice it is good enough.

4.3 Regeneration

Memory is the scarce resource. We generate, then sort into numerical order, as many expression values as we can fit. It would be wasteful to store the actual strings. Since the strings are generated by calls to a known random number generator, exactly the same string will be generated from the same random seed. We therefore use a random generator whose initialization is fast,[7] and re-initialize it with a different seed for each string. What we store, and later use to regenerate the small number of interesting strings, is the seed, as an 8-byte `long long`.

4.4 Post-Processing

As first encountered, a coincidence looks like this:

$$\begin{aligned}&\text{“}\texttt{151cLa-+1+-7L+Et*}\text{”} - \text{“}\texttt{1pa16-1+*+4+qq-}\text{”}\\ &\qquad\approx 1.862515913491208 - 1.862515913491209 \approx -2.38435\text{e-}016\end{aligned} \tag{17}$$

It is easy to re-evaluate the strings symbolically instead of numerically and output *Mathematica* notation:

```
(1*(((((1+(10-Log[12]))+1)-5)+Log[7]))^(1/3))
                - (((((Pi+(10*((6-1)+1)))+4))^(1/2))^(1/2)-1)
```
(18)

In turn, we can send expressions like this, in batch, through a *Mathematica* function defined as something like

```
Sho[expr_] := {Print[FullSimplify[expr]], Print[N[expr, 20]]}
```
(19)

That is, we needn't re-invent wheels as complex as the `FullSimplify[]` and (arbitrary precision) `N[]` functions, since only a small number of candidate expressions must be post-processed in this way. In the above case, we get

$$\begin{aligned}&1 - (64+\pi)^{1/4} + \left(7 - \text{Log}\left[\frac{12}{7}\right]\right)^{1/3}\\ &- 1.6982395465876682008 \times 10^{-16}\end{aligned} \tag{20}$$

which is readily seen to be a form of equation (2).

4.5 Density versus Aesthetics

In the above model, we specify the average length of expression strings through the parameter n_{bin} (or μ_{bin}). We typically use $\mu_{\text{bin}} = 9$. If μ_{bin} is too small, then the number of distinct expression values becomes too small, both because of identical strings and tautologies. On the other hand, if it is too large, then the resulting expressions become large and ugly – they start to look somehow like “constructions” instead of “coincidences”. This purely psychological, not mathematical, effect is a thread that, if pulled, can unravel the whole present enterprise.

It is not really surprising that long strings can represent π, for example,

$$3 + 1/10 + 4/10/10 + 1/10/10/10 + 5/10/10/10/10 + \ldots \tag{21}$$

Indeed, none of our coincidences are surprising in an information theoretic sense: they all have more symbols, drawn from a larger alphabet, than the corresponding decimal representation.

The space we are searching is thus fundamentally aesthetic, not mathematical. While equations (3) and (4) are indeed about the most accurate that we

have turned up, the less-accurate equations (1) and (2) were selected on aesthetic grounds from among 100 or so candidates of comparable accuracy. The fact that we find all of equations (1)–(4) multiple times suggests that many closer coincidences can be found by increasing $\mu_{\rm bin}$, N and M. But if these are ugly, then who will care?

References

[1] *Mathematica*, Wolfram Research Inc., http://www.wolfram.com

[2] "Standard for Floating-Point Arithmetic", IEEE 754-2008.

[3] W. Diffie and M. E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard", *Computer* 10 (6), 1977.

[4] Wikipedia, "Meet-in-the-middle attack," (seen May 30, 2009).

[5] Wikipedia, "Reverse Polish notation," (seen May 30, 2009).

[6] L. Carroll, *Through the Looking Glass*, Chapter 6, at http://www.gutenberg.org/files/12/12-h/12-h.htm#2HCH0006

[7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. (New York: Cambridge), 2007, §7.1.3 (routine `Ranq1`).