How to hunt wild constants

David R. Stoutemyer

March 30, 2021

Abstract

There are now several comprehensive web applications, stand-alone computer programs and computer algebra functions that, given a floating point number such as 6.518670730718491, can return concise nonfloat constants such as $3\arctan 2 + \ln 9 + 1$ that closely approximate the float. Usefully often such a result is the exact limit as the float is computed with increasing precision. Therefore these program results are candidates for proving an exact result that you could not otherwise compute or conjecture without the program. Moreover, candidates that are *not* the exact limit can be provable bounds, or convey qualitative insight, or suggest series that they truncate, or provide sufficiently close efficient approximations for subsequent computation. This article describes some of these programs, how they work, and how best to use each of them. Almost everyone who uses or should use mathematical software can benefit from acquaintance with several such programs, because these programs differ in the sets of constants that they can return.

1 Introduction

"*. . . you are in a state of constant learning.*"

– Bruce Lee

This article is about numerical *mathematical* constants that can be computed approximately.

For real-world problems, we often cannot directly derive exact closed-form results even with the help of computer algebra, but we more often can compute approximate floating-point results – hereinafter called **floats**. For some such cases there is an exact closed-form result that the float approximates and that form is simple enough so that we would like to know it, but we do not know how to derive it or to guess it as a prerequisite to a proof. If we had one or a few plausible concise nonfloat candidates that agree sufficiently closely with the float, then we could concentrate our efforts on attempting to prove that one of these candidates is the exact result.

This article describes text and software tools that provide such guesses. Table 1 lists several such tools in order of presentation, grouped by type. Figure 1 plots the initial

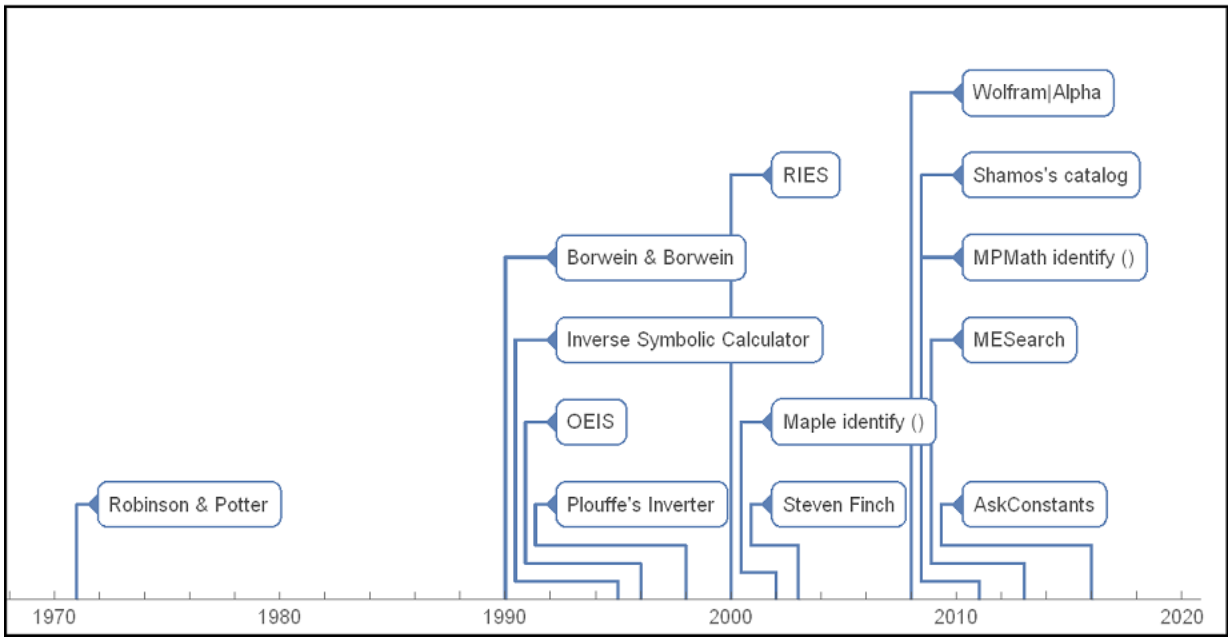
publication dates of these tools on a time line. All of the web-based and downloadable tools are free, but some of them require Java, a C compiler, Maple or *Mathematica*.[1]

Table 1: Web apps, standalone apps, CAS functions, CAS apps, and books with tables

Type	**Name**	**Sec.**	**Prerequisite**	**http://www. or https://www.**
Book table	Robinson & Potter	2.1	Library or buy	escholarship.org/uc/item/2t95c0bp
	Borwein & Borwein	2.2		springer.com/gp/book/9781461585121
	Steven Finch	2.2		sites.oxy.edu/lengyel/originals/0521818052ws.pdf
.pdf table	Shamo's Catalog	3	Smart-phone or computer	[18]
Web search	Google, etc.	4		Google.com, etc.
Web apps	OEIS	5.1		oeis.org
	WolframAlpha	5.2		wolframalpha.com
	ISC	5.3		wayback.cecm.sfu.ca/projects/ISC/ISCmain.html
Stand alone	MESearch	6.1	Java	xuru.org/mesearch/MESearch.asp
	RIES	6.2	C compiler	mrob.com/ries
In CAS	identify	7.1	Maple	maplesoft.com/support/help/Maple/view.aspx
	identify, findpoly	7.2	SymPy	docs.sympy.org/0.7.1/modules/mpmath/
CAS apps	AskConstants	8.1	*Mathematica*	AskConstants.org
	Plouffe's Inverter	8.2	Maple	plouffe.fr/Simon%20Plouffe.htm
PSLQ	Custom IR model	9	Fortran C_{++} or CAS	[1, 2]

[1]But *Mathematica* is free and pre-installed on some Raspberry Pi computers that cost only a few dollars.

Figure 1: Text tables, applications, applications and functions for identifying float constants



Float inputs for these tools often arise from computations such as numerical integration, iterative equation solving, approximate optimization, etc. The purposes of this article are to explain how best to use these tools, to explain how they work, and to explain how to decide if proposed candidates are promising or else probably impostors.

Section 8 discusses input magnitude limitations and Section 9 discusses some common causes of impostor results, with conclusions in Section 10.

2 Text tables

2.1 *Mathematical Constants* table by Robinson & Potter

Here is an excerpt from a hand-typed table of about 3000 constants by Robinson and Potter [15]:

...	...
4 .22755 35333 76265 40809	$-\psi(1/4) = \gamma + 3\ln 2 + \pi/2$
0 .22755 09577 68849 99385	$4/(\pi^2 e^\gamma)$
0 .22756 34054 87472 14332	root of $7xe^x = 2$
...	...

Notice that:

- Inverse to more common tables of constants, the inputs in the left column are floats and the corresponding results in the right column are corresponding exact constants.

- The digamma function of $1/4$ was negated to make the float positive. Most such tables do this **sign aliasing**, because it is easy to discard the sign of the float, then negate the corresponding nonfloat result. This doubles the potential coverage of the table.

- The inputs are sorted by the *fractional parts of the absolute values* of the input floats rather than by those entire input floats. For a given float $\tilde{x}$, you do a manual search for the fractional part of $|\tilde{x}|$, then decide whether or not the discrepancy with the fractional part of either bracketing entry or entries slightly further away is small enough to justify further consideration. This **fractional-part aliasing** makes the table applicable to many more examples and is easily inverted mentally to construct the true candidate. For example, we can also easily guess that a numerical result 5.22755 35334 might approach $1 - \psi(1/4)$ as the precision increases.

- The last entry in the above table is an implicit result. Since the popularization of the Lambert W function [5], this exact result can now be expressed explicitly as $W_0(2/7)$, but the table contains solutions to other equations that cannot yet be expressed explicitly, together with definite integrals, infinite series, and infinite products having no known closed form.

To view a photocopy of the entire table, visit

`https://escholarship.org/uc/item/2t95c0bp`

Definition. *Published constants* are publicly accessible closed-form and/or approximate float constants – either printed or on the web.

Definition. *Named constants* are constants having a widely-accepted name, such as Catalan's constant or the *twin prime* constant.

Definition. *Wild constants* are computed float constants for which **you** do not know an exact closed form for the limit as the precision of the approaches infinity.

Definition. *Tabulated constants* are formed by systematically applying sets of functions to systematic sets of arguments, Printed tables for approximate computation usually have equally-spaced arguments that are terminating decimal fractions such as 1.001, 1.002, 1.003, etc. In contrast, to return nonfloat results such as $\arctan(2/3)$, the curated and computer-generated tables described in this article often instead use the set of all reduced fraction arguments whose numerator and denominator magnitudes do not exceed some given integer – perhaps also multiplied by common irrational constants such as π, $\sqrt{2}$, etc.

Remark. The table by Robinson and Potter contains all of the above types.

2.2 Other extensive text tables

- Borwein and Borwein published a similar table of about 100,000 constants [3]. The table contains mostly computer-generated tabulated constants and some dimensionless physical constants.

- The award-winning book by Steven Finch [9] has not only a table of about 10,000 well chosen non-tabulated constants, but it and Volume II ([10]) contain short essays about those constants, with references to the literature about them.

3 .pdf and .html tables

Michael Shamos posted a .pdf file of a table of about 10,000 constants at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.366.9997&rep=rep1&type=pdf` Most of these constants are isolated pairs of a float value and a closed form defined by a definite integral, infinite series, or infinite product. Many floats list several corresponding formulas, such as various integrals, series and products that have the same value, which is helpful by suggesting other perhaps more efficient ways to compute more digits.

A **web search for "Mathematical constants"** can locate other such downloadable tables or .html tables that can be searched directly on the web.

4 Web browsers

Web browser search engines are helpful for finding candidate nonfloats for particular float inputs.

Here are some **tips** for using the Google Search Engine for this purpose[2]:

1. It seems better to use decimal fraction notation than scientific notation.

2. For negative numbers, try both the negative entry and its absolute value.

3. For numbers having magnitude less than 1, try both with and without a leading 0 digit before the decimal point.

4. Some publications have truncated the exact value and some have rounded it, so try both.

5. Most published constants have 16 or fewer significant digits, because most are computed with 16-digit significands or less, and the author of the published float might have further rounded or truncated the computed result to allow for computational errors or typesetting constraints.

6. Your entry must often have exactly the same digits as a publication – no more and no fewer.

[2]The Microsoft Edge and Apple Safari browsers currently allows a choice of Google, Bing, Yahoo or DuckDuckGo as a search engine .

7. Many relevant publications tend to have exactly 11 digits, including a leading zero before the decimal point if the magnitude is less than 1, because they were taken directly or indirectly from the table in Steven Finch's book described in subsection 2.2.

8. As you decrease the number of digits down to a few, the number of browser results tends to increase, but so does the number of results that are nearby mathematical constants or are not mathematical constants.

9. Thus one strategy is to use up to the first 17 digits of your decimal float (using a sign if negative and a leading 0 digit for magnitudes < 1), then repeatedly truncate successive digits (and also try rounding where it is different), until you obtain at least one promising result on the first page or two of the results.[3] Then if those results include a name or other helpful clues (such as "Artin's constant"), branch into a search using that clue. At each digit length you can optionally experiment with omitting a leading minus sign and or 0 digit.

- If there is more than one plausible result and the associated texts do not indicate equivalence, then repeat the search starting with more digits if you can compute them.

- Somewhat different tips might be appropriate for other search engines, and Google's search algorithm might evolve to make some of the tips here unnecessary. To learn, experiment with float approximations to a few known mathematical constants are not too obscure or too well known.

5 Free web applications

If you have internet access as you read this, then you might want to try each of the applications in this section as you read the subsections. With good eyesight, most of them are usable even with most smart phones.

To become acquainted with them, using simple then less simple nonfloat constants from books and articles, try entering their floating-point approximations of varying precisions. Also try random floats having differing numbers of significant digits. As the number of significant digits increases, random floats are decreasingly likely to be the truncated or rounded value of any nonfloat expression simple enough for any of these programs to guess in a reasonable amount of time. An application is **generous** if it often suggests expressions for such random-number entries, versus **parsimonious** if it rarely does. It is alright for an application to be generous, but it is important for you to realize it. Generosity can be helpful for determining bounds, efficient sufficiently accurate approximations or for suggesting the dominant terms of a longer exact result.

Every float is exactly representable as an easily computed rational number, but floats cannot represent all rational numbers and floats are often approximations to irrational

[3]Beware that, for example, rounding 1.546 to 1.56, then subsequently rounding that to 1.6 is different than round 1.546 directly to 1.5 .

numbers. If your float approaches a limit as the working precision of the calculation approaches infinity and that limit is expressible as a nonfloat expression, then that limit is what you want. **Perfect agreement with a mere approximation prevents that desired result**.

5.1 The On-line Encyclopedia of Integer Sequences

One of the main purposes of the OEIS site at https://oeis.org/ is to identify integer sequences such as proposing that the truncated sequence 8, 16, 32, 64, . . . might be a subsequence of the sequence whose nth element is 2^n. Another goal of this database is to provide accurate and comprehensive information about these sequences.

The successive digits of a decimal fraction can be regarded as an integer sequence, and an under-advertised capability of OEIS is to propose nonfloat constant expressions that closely approximate your float. OEIS has sequences of up through about 100 digits or more for about 10,000 constants regarded as important by the many contributors who posted them. To attempt identifying nonfloat candidates for your float, you can enter a comma-separated list of some leading digits of the *absolute value* of its decimal fraction value, such as 3, 1, 4, 1, 5, 9, 2, 6 . Ignoring any leading 0 digits, OEIS then attempts to match your other entered successive digits with any subsequence of its sequences. It then lists information about each of its integer sequences that contain your sequence as a subsequence. However, it is better to enter your float digits as a positive decimal fraction with a decimal point, such as 3.1415926. (Do not use scientific notation.)

Any matched table entries can contain hyperlinks to references and contain computer-algebra program fragments to compute the constant to any number of digits efficiently. There is also usually a Keyword link labeled "cons" that leads to the precomputed value of the constant up to a median of about 100 significant digits.

This exact-match *radix search* requires you to *truncate* trailing digits of the absolute value of your float *rather than round*. However, the hardware or software that computed your float is approximate and almost certainly used rounding rather than truncation. Therefore you should discard enough of your trailing digits to be reasonably confident that your submission matches exactly the leading digits at OEIS.

If your computed float ends with a sequence of digits 9, then also try what you get by adding 1 to the last digit. If your computed float ends with a sequence of digits 0, then also try what you get by subtracting 1 from the last digit, because those small relative changes to your approximate float can dramatically alter the digit sequence.

If you enter too few digits then you will obtain numerous matches, most or all of which are irrelevant. However, with about 10^5 numbers in the data base, entering 8 or more significant digits is usually sufficient to achieve at most one match, and almost all of the OEIS sequences for floats contain at least that many digits. Therefore you can usually truncate to about 8 significant digits without much chance of false matches, which might be revealed by comparison with your truncated digits.

Since OEIS ignores the position of the decimal point, entering 31.41592 returns π rather than 10π. It is your responsibility to notice this and multiply the proposed exact constant by the appropriate power of 10. This base-ten **significand aliasing** increases the number of floats that can be matched.

Since OEIS can match any *subsequence* of their digit sequences, you can also exploit fractional part aliasing by entering only the fractional part of the absolute value of your float. For example, if your truncated mystery float is 9.141592654, then entering it produces no result, whereas entering 0.141592654 produces π and gives its float value, enabling you to recognize that your mystery float is probably $6 + \pi$. Thus it is worth trying all of your float, then only its integer part if different.

Beware that fractional part aliasing suffers catastrophic cancellation when the absolute value of the fractional part is small compared to the absolute value of the float, whereas significand aliasing suffers no loss of precision. Therefore it is best if 8 or more significant digits remain after omitting the integer part and truncating dubious trailing digits.

Other than you exploiting these types of aliasing, OEIS makes no effort to identify more general transformations of its constants, such as $2/3$ of a named constant or $2/3$ plus a named constant.

All OEIS integer sequences and float digit sequences are assigned an OEIS name consisting of the letter "A" followed by six digits. Although such names are difficult to remember and do not acknowledge the discoverers, they are unique and serve as a *de facto* standard – particularly when there is no other widely recognized name. In contrast, traditional names can have various spellings, subsets, and orderings of all the discoverers, as well as values differing by a factor of 2, etc, on account of one author using a radius where another uses a diameter, for example.

A complimentary way to use OEIS for named constants is to enter a name, such as *donkey*, *grazing goat*, or *A075838*. If successful, you will find the same information as for entering the truncated float value 0.952847864 .

Brief sorted descriptions of all the constants together with links to their pages are at

`https://oeis.org/wiki/Index_to_constants#Description`

Some of these constants are tabulated, but most are isolated published constants, drawn from many areas by many contributors.

Even if another tool returns a plausible result, OEIS and Steven Finch's books are good starting points for learning more about an OEIS constant in your result.

As a training example, try to compute

$$\int_0^\infty \frac{(\coth(\pi x)+1)\left(2x\ln\left(x^2+\frac{1}{4}\right)-2\arctan(2x)\right)}{4x^2+1}\,dx$$

symbolically and numerically, then enter the latter at `https://oeis.org/` ., truncated to about 8 digits

5.2 WolframAlpha

If you enter a float into WolframAlpha at `https://www.wolframalpha.com/`, then the results usually include up to three guesses for a nonfloat constant that the float approximates.

A way to test the tools described in this article is to approximate a nonfloat constant to a generous number of significant digits, then see how many of those digits are necessary to produce an equivalent expression, if any. Accordingly, I approximated $1+3/(1+\sqrt{3}+\pi)$ to 18 digits, then tried entering that and successively fewer digits into Alpha. Table 2 lists the results for floats *rounded* to 10 and to 11 significant digits, together with entries having the 11 correct digits followed by one and two intentionally incorrect digits. The underlined digits in the entries differ from the exact expression by at most 0.5 units in the last place.

Table 2: WolframAlpha test for identifying $1 - \dfrac{3\sqrt{3}}{1+\sqrt{3}+\pi}$ from a float approximation

Entry	**digits**	**Result (Agrees with input to 0.5 in last bold digit)**	≡
0.1153442566	10	$\dfrac{11-7\pi+2\pi^2}{\pi(21+\pi)} \approx \mathbf{0.1153442564}9366$	
		$\dfrac{1}{\text{root of } x^4-75x^2+10x-99 \text{ near } x=8.6697} \approx \mathbf{0.1153442564}6239$	
		root of $99x^4-10x^3+75x^2-1$ near $x=0.115344$ $\approx \mathbf{0.1153442564}6239$	
0.11534425658	11	$\dfrac{1-2\sqrt{3}+\pi}{1+\sqrt{3}+\pi} \approx \mathbf{0.11534425658}4835$	✓
		$\dfrac{-2e\,e!+124-35e+5e^2}{136e} \approx \mathbf{0.1153442565}9706$	
		root of $92x^4+49x^3-27x^2+37x-4$ near $x=0.115344$ $\approx \mathbf{0.1153442565}8563$	
0.1153442565800	12	$\dfrac{1-2\sqrt{3}+\pi}{1+\sqrt{3}+\pi} \approx \mathbf{0.1153442565}814835$	✓
		$\dfrac{2(15\sigma_S-1)}{5(6\sigma_S+73)} \approx \mathbf{0.1153442565}818196$ where σ_S is Somos's quadratic recurrence constant	
		π (root of $100x^4-152x^3-x^2-27x+1$ near $x=0.0367$) $\approx \mathbf{0.1153442565}828298$	
0.11534425658800	13	$\dfrac{45419\pi}{1237062} \approx \mathbf{0.1153442565800}221712$	
		$\dfrac{3e\,e!+295-86e-9e^2}{94e} \approx \mathbf{0.1534425657}9947$	
		$\dfrac{1}{45}(4C+30-11\pi-40\pi\log(2)+27\pi\log(3))$ $\mathbf{0.1153442565800}$78601, where C is Catalan's constant	

Notice that:

1. Alpha proposed an equivalent expression for 11 and 12 correct significant digits, but not for 10.

2. Too many units of discrepancy in the last decimal place can mislead the algorithm into trying to fit those incorrect digits.

3. The returned candidates are usually nonequivalent.

4. Alpha almost always finds expressions that differ from your input by at most a few units in the last place that you entered. Therefore those differences alone are usually not good criteria for favoring one alternative over the others.

Alpha mostly uses the PSLQ integer relation algorithm[4] [7, 8]: Given a basis vector $\mathbf{c} = [c_1, c_2, \ldots]$ of nonzero symbolic and/or float constants, the algorithm returns either an indication of failure or a minimal 2-norm vector of integers $[n_1, n_2, \ldots]$ not all 0 such that the inner product $n_1 c_1 + n_2 c_2 + \cdots \simeq 0.0$ and $\gcd(n_1, n_2, \ldots) = 1$.

1. The most straightforward way to exploit this algorithm is to make c_1 be the given float $\tilde{x}$ and choose commonly occurring nonfloat constants such as 1, π, $\sqrt{2}$, and $\ln 2$ as the other constants. If the algorithm returns a vector $[n_1, n_2, \ldots]$, then a close-fitting candidate is the *rational linear combination*

$$\tilde{x} \simeq -\frac{n_2}{n_1} c_2 - \frac{n_3}{n_1} c_3 - \cdots .$$

   The last entry in Table 2 is of this type.

2. Another way to exploit this algorithm is if

$$(n_1 c_1 + n_2 c_2 + \cdots)\, \tilde{x} + (n_k c_k + n_{k+1} c_{k+1} + \cdots) \simeq 0.0,$$

   then

$$\tilde{x} \simeq -\frac{n_1 c_1 + n_2 c_2 + \cdots}{n_k c_k + n_{k+1} c_{k+1} + \cdots}.$$

   This *linear fractional* category of model was used for the correct identifications and some of the incorrect ones in Table 2.

3. Another way to exploit this algorithm is if $n_1 + n_2 \tilde{x} + \cdots + n_{m+1} \tilde{x}^m \simeq 0.0$, then $\tilde{x}$ is approximately one of the zeros of the polynomial $n_1 + n_2 x + \cdots + n_{m+1} x^m$. This can be tried for $m = 1, 2, \ldots$ up to some maximum degree that increases with the number of significant digits. If PSLQ returns a vector of integers, then an approximate polynomial zero algorithm can then be used to determine which zero is closest to the given float. Several entries in Table 2 report such *algebraic number* candidates.

[4]The LLL algorithm can also be used for this purpose.

4. For a power product model of the form $\tilde{x} \simeq b_1^{r_1} b_2^{r_2} \cdots$ with given positive nonfloat base constants b_k and unknown rational exponents r_k, if $n_1 \ln \tilde{x} + n_2 \ln b_1 + n_3 \ln b_2 \cdots \simeq 0.0$, then

$$\tilde{x} \simeq \exp\left(-\frac{n_2 \ln b_1 + n_3 \ln b_2 + \cdots}{n_1}\right) = b_1^{-n_2/n_1} b_2^{-n_3/n_1} \cdots .$$

   The irrational factors of an irrational exponent can be considered part of the base to make this category fit a model such as $\tilde{x} \simeq 2^{r_1} 3^{r_2\sqrt{5}} e^{r_3\pi} = 2^{r_1} \left(3^{\sqrt{5}}\right)^{r_2} (e^{\pi})^{r_3}$.

5. For a model of the form $f(\ldots)$, where f is invertible with respect to at least one of its arguments and that argument model is one of the above types or recursively of this functional type:

   - Using predetermined constants for any other arguments, apply an inverse of f to the given float $\tilde{x}$ giving $\tilde{y}$.
   - If the argument model is successfully fit to $\tilde{y}$, giving a nonfloat candidate constant y, then return $f(y)$.

The computing time grows rapidly with the number of elements m in the basis vector and the number of words in the float significands. Moreover, the PSLQ algorithm can require the input float to have mn significant digits of precision to return a correct m component vector having integer components of up through n digits even if many of these integer coefficients have much smaller magnitudes, including 0 . Therefore, although specialized efforts to identify particular floats have used thousands of digits and hundreds of vector components, the general-purpose programs described here use mostly basis vectors having ≤ 8 elements.

Advise for using Alpha to identify floats:

1. If you enter 18 or more significant digits, including any trailing 0 digits, then Alpha uses arbitrary precision rather than 16-digit machine floats. Arbitrary-precision floats are slower but more than 18 digits are necessary for PSLQ to determine many published constants.

2. Alternatively, you can append a suffix of the form ‘n to the significand of your input to indicate that your best guess for the Precision is the value of n, which can be a decimal fraction. For example, if your input for the example in Table 2 was computed by a iterative equation solving using 16-digit IEEE binary64 hardware and you guess that most probably about 14 digits are correctly rounded, then you could start by entering 0.1153442565814834‘14 . This has the advantage that you can enter all of the computed digits, which which are more likely to be correct than the value rounded or truncated to 14 digits, but without having Alpha assume that all but perhaps the last one are correct. In the other direction, if your computed 16-digit machine float is 0.1615000000000000 and you are confident that all are correct, then you can enter it more concisely as 0.1615‘16 . The “‘” character is near the upper left corner on many keyboards.

3. Software does not always display all of the digits in its significands, and internet browsers paste into web applications such as Alpha only the *characters* that you highlight when copying. For example, *Mathematica* displays by default only 6 of the approximately 16 significant digits in the widespread IEEE binary64 machine floats, and displays none of the 8 or more stored *guard* digits in arbitrary-precision floats. Often some of those hardware or guard digits are correct and therefore helpful to try entering. Therefore, learn how to display all of the digits in your software's floats.[5]

4. Unless you are an exceptionally fast accurate typist, highlight then **copy the float rather than type it**.

5. Then *round* to as many of all those computed digits as you think have an absolute error of most nearly 1/2 unit in the last place: Cancellation and rounding errors often lose one or more digits of precision. Discretization errors such as from quadrature often lose more, and each of these causes can lose all of the digits. Interval arithmetic can provide an upper bound on cancellation and rounding errors, but not necessarily on discretization errors. Significance arithmetic such as *Mathematica's* arbitrary-precision arithmetic provides cancellation and rounding error estimates that are occasionally underestimates but more frequently overestimates.

6. Try rounding to successively fewer and more of the digits that you computed.

7. Compared to impostors, promising candidates often exhibit the property of persisting throughout a larger range of rounding or truncation levels up through and perhaps slightly beyond their correct digits.

8. PSLQ overfits often contain many digits in their rational numbers. Several terms containing only simple rational numbers is less suspicious than one term having a complicated rational number. Information theory suggests that **candidates having nearly as many or more total digits than the float input are particularly dubious**.

5.3 Inverse Symbolic Calculator

Inverse Symbolic Calculator (ISC) implemented by Simon Plouffe working with Jonathon and Peter Borwein and others is currently hosted at

`http://wayback.cecm.sfu.ca/projects/ISC/ISCmain.html` .

Plouffe [12] describes its origins.

ISC provides four different techniques to apply to the float that you enter:

[5] One way to do this in *Mathematica* is to copy all of the displayed digits then paste into a *notebook*, then copy all of that fully-displayed float. Another way is to copy the result of InputForm [*float*].

5.3.1 Simple Lookup and Browser

This technique searches a precomputed table of sorted **truncated** 16-digit base-ten **significands** of the **absolute values** of the floats, each paired with a corresponding nonfloat constant.[6]

That table has about 30 million pairs that are a mixture of published and tabulated constants. The sorted significands and the corresponding nonfloats are subdivided into about 9000 files whose names include some leading digits of all the significands in that file, with each file containing all of the significands with those leading digits. The search begins with a binary search on the file names in RAM, followed by a search in the appropriate file to perfectly match or bracket the significand of the float that you entered, followed by retrieval of close neighbors of those entries. Here are some **input tips**:

- ISC Simple Lookup defines a **match** as **all** of the entered digits of the significand being **identical** to that leading portion of a tabulated significand, but ignoring any entered digits beyond 16.

- The last one or more digits of your computed float are often not the correctly truncated ones. To maximize your chance of success, try to compute as many digits as is practical to be reasonably confident that the first 16 digits are correctly truncated, then enter those 16 digits. Your float was probably computed with software that rounds rather than truncates, so at least truncate the last computed digit.

- If you cannot compute more than 16 digits, then truncate to as many as your best guess for how many are correctly truncated digits of the limit you seek.

- If the most significant truncated digit is a 9, then also try truncating what you get by adding 1 to that 9 digit. If the most significant truncated digit is a 0, then also try truncating what you get by subtracting 1 from that 0 digit. Those minimal relative changes to your entered float can dramatically alter the digit sequence.

- You can enter as few as 5 significant digits, but with about 30 million entries having significands 0.1000000000000000 through 0.9999999999999999, the mean distance between table significands is

$$\frac{0.9999999999999999 - 0.1000000000000000}{30 \times 10^6} \simeq 3.3 \times 10^{-8}.$$

  Thus the expected number of matches for 5 entered significant digits is about 1000. It is burdensome to assess more that a few matches even if they contain the limit you seek; and they often do not, because a known closed form does not exist or it is not in the tables or you have exceeded the ISC bound on the number of matches.

- You are offered the opportunity to browse nearby table entries even if there are no matches. Doing so might reveal one or more candidate nonfloats that agree closely and are plausible considering your knowledge of the problem domain.

[6] Mentally truncating fractional digits correctly is easier than mentally rounding them correctly.

- The table contains many instances of nonfloats that all have the same 16-digit significand value. The corresponding nonfloats are often equivalent to each other within a factor that is a power of 10. Approximating the nonfloats to higher precision can reveal nonequivalence.
- If you do not obtain at least one promising result, then try also entering fewer and more truncated digits covering what you guess spans most of the possible actual correctly truncated digits of your mystery float.

As a test example, the correctly truncated 16-digit value of

$$-10\sqrt{\frac{2}{37}\left(8\sqrt{3}-9\right)} = -5.123557917376186\,. \tag{1}$$

Table 3 summarizes the results of entering various further truncated and rounded leading digits to expose the consequences of rounding rather than truncating – and of appending incorrectly truncated digits beyond the sixteenth.

Table 3: ISC Simple Lookup test results for approximated $\left|-10\sqrt{\frac{2}{7}\left(8\sqrt{3}-9\right)}\right|$

Input	# digits	Treatment	Result	?
5.1235	5	**truncated**	**38 matches** include $12^{1/4}/\sqrt{8+9^{3/4}}$	Too many
5.1236		rounded	**360 matches**. They **don't** include $12^{1/4}/\sqrt{8+9^{3/4}}$	Too many
5.123557	7	**truncated**	Matches only $12^{1/4}/\sqrt{8+9^{3/4}}$	✓+
5.123558		rounded	6 matches that **do not** include nearby $12^{1/4}/\sqrt{8+9^{3/4}}$	All bad
5.1235579173	11	**truncated**	Matches only $12^{1/4}/\sqrt{8+9^{3/4}}$	✓+
5.1235579174		rounded	No match, but browser lists adjacent $12^{1/4}/\sqrt{8+9^{3/4}}$, matching 10 digits	✓
5.12355791730	12	incorrect last digit	No match, but browser lists adjacent $12^{1/4}/\sqrt{8+9^{3/4}}$, matching 10 digits	✓
5.12355791737618	15	**truncated**	Matches only $12^{1/4}/\sqrt{8+9^{3/4}}$	✓+
5.12355791737619		rounded	No match, but browser lists adjacent $12^{1/4}/\sqrt{8+9^{3/4}}$, matching 14 digits	✓
5.1235579173761861	17	**incorrect last digit**	Matches only $12^{1/4}/\sqrt{8+9^{3/4}}$	✓+
5.1235579173761869		**incorrect last digit**	Matches only $12^{1/4}/\sqrt{8+9^{3/4}}$	✓+

For each match, ISC lists the truncated 16-digit base-10 significand (without a decimal point) equated to a corresponding nonfloat, such as

$$512355791737618 = 12\text{^}(1/4) \ / \ (8 + 9\text{^}(3/4))\ \text{^}\ \ (1/2)\ .$$

To process this match:

1. The ratio of your input 5.123557917376186 to the significand 0.5123557917376186 is 10, so multiply the candidate nonfloat by 10 to de-alias the significand.

2. Then negate that, because our float of interest in equation (1) is negative and we necessarily entered the negative of that value, giving our final result

$$-\,10\,\frac{12^{1/4}}{\sqrt{8+9^{3/4}}}. \tag{2}$$

3. For verification, we should approximate expression (2) to slightly more than 16 significant digits, then truncate to 16 digits, giving -5.123557917376186, which agrees with the left side of equation (5.3.1) to 15 digits.

Beware that the table was assembled over many years from many computed and printed sources that used varying syntactic conventions. For example:

- Gam, Gamma, GAM and GAMMA all denote the Gamma function, whereas gamma denotes the Euler-Mascheroni constant.
- sqrt and sr both denote $\sqrt{\ldots}$.
- A polynomial in x or root ($a\_polynomial\_in\_x$) denotes the zero of that minimal polynomial nearest the given float.
- Non-real expressions often implicitly denote the real parts of those expressions.
- As with our regrettably ambiguous traditional mathematics notation, For example, sometimes 16^3/4 means 16^(3/4) rather than (16^3)/4, $\sin e + 3$ means $\sin(e+3)$ rather than $\sin(e)+3$, and $7/3\pi$ means $7/(3\pi)$ rather than $(7/3)\pi$, etc.

  Step 3 above enables you to verify or refute your interpretation of the ambiguous nonfloat syntax, then try another interpretation if necessary.

Many nonfloats in the table contain either a common name constant such as Bernstein or a red OEIS web link labeled with an OEIS constant having a name of the form "A" with a six-digit suffix. Either way, you can search for their definitions at `https://oeis.org/`, as described in subsection 5.1.

5.3.2 Smart Lookup

This technique uses the same lookup table about 100 times, but precedes each lookup with a transformation of your float. For example, smart lookup might add 3/7 or multiply by 3/7 or apply ln(...). For a given number of input digits, the average number of matches is about 100 times as many as for simple lookup, and some transformations lose precision. Therefore smart lookup requires you to enter at least 10 digits, and the transformation of your input is done using 26-digit significands. Thus it is beneficial to enter floats having up through 26 correctly rounded digits even though the forward table floats have only 16-digit significands.

For each match found, ISC lists the transformation that it used, with K representing your float input, followed by the transformed float significand, the number of leading digits that are identical to the matching table entry, and the number of matches for that transformation. For example, computed to 26 significant digits,

$$-\frac{1}{4}-10\,\frac{12^{1/4}}{\sqrt{8+9^{3/4}}}\simeq -5.37355791737618667154532 32\,. \tag{3}$$

Applying simple lookup to the absolute value of this float truncated to 16 digits returns no match, and the ISC browser lists no result that agrees to more than 5 digits. However, applying smart lookup to the absolute value of the right side of equation (3), computed with arithmetic that rounds, returns the table

Function	Result	Precision	Matches
K − 1/4	5.12355791737618667154532 32	16	1

In the Function entry, K−1/4, K represents your input 5.37355791737618667154532 32. The different Result entry is the corresponding value of the formula in the Function entry, as computed with 26-digit significands.

If you click on the Function entry K − 1/4, then ISC shows the result of doing simple lookup with the truncated significand of the transformed Result float, which gives

$$\frac{12^{1/4}}{\sqrt{8+9^{3/4}}}, \tag{4}$$

together with an opportunity to browse around it.

If you want to further investigate one of these nonfloats that have acceptable agreement, then, for example, you should

1. Numerically approximate the nonfloat, preferably to at least 16 digits or slightly more, giving
$$\mathrm{K}=12^{1/4}/\sqrt{8+9^{3/4}}\simeq 0.512355791737618667\,.$$
2. From that value and the value in the Result entry, determine a power of 10 by which to multiply the nonfloat:
$$\frac{5.12355791737618667154532 32}{0.512355791737618667}=10.0000000000000000,$$

giving

$$10\,\frac{12^{1/4}}{\sqrt{8+9^{3/4}}}$$

3. Solve $y = \mathrm{K} - 1/4$ for K, giving $K = y + 1/4$, giving

$$K = \frac{1}{4} + 10\frac{12^{1/4}}{\sqrt{8+9^{3/4}}}.$$

4. Negate this result giving the nonfloat candidate for our float of interest

$$-\left(\frac{1}{4} + 10\frac{12^{1/4}}{\sqrt{8+9^{3/4}}}\right).$$

More generally there might be several rows for several different transformations that resulted in matches. Click on each of the Function entries to see the corresponding nonfloats and optionally browse neighbors.

For this example, entering fewer than 16 digits produced no smart lookup result.

5.3.3 ISC Integer relation Algorithms

ISC provides an Integer Relation alternative that is worth trying too. It requires inputs of 16 through 32 significant digits, and it checks for algebraic numbers through fifth degree and for rational linear combinations of the basis vectors

$$[e, \pi, \gamma, \mathrm{Ei}\,(1), W\,(1), 1]\,,$$
$$\left[\sqrt{3}\pi, \ln 3, \ln 2, \gamma, \sqrt{2}\pi\right],$$
$$\left[\pi^2, \mathrm{Catalan}, \pi \ln 2, \sqrt{2}\pi^2, (\ln 2)^2\right],$$
$$\left[\pi^3, \zeta\,(3), \pi^2 \ln 2, (\ln 2)^3, \sqrt{3}\pi^3, \sqrt{2}\pi^3\right].$$

These basis vectors might seem rather specialized. However, a noticeable portion of wild constants in the printed tables of Section 2 are representable with a subset of at least one of these vectors or with algebraic numbers of degree ≤ 5.

It is wise to compute as many digits as you can up through 32. There is no need to truncate, and it is OK for a few of the last digits to be incorrectly rounded.

5.3.4 Generalized Expansions

You must enter at least 16 significant digits for this option, and it is most appropriate to round your approximate float to where you are reasonably confident that the last digit is correctly rounded.

This option computes a truncated continued fraction from your input float, and computes similar truncated infinite representations such as a truncated infinite product and a truncated Egyptian fraction. This can be helpful because these alternatives might suggest patterns that might be useful for computing the float to arbitrarily many digits more efficiently than the method you used. Moreover, the GFUN package [17] is then

applied to these truncated representations to try guessing a generating function for the infinite representation. (It is for you to prove that it *is* a generating function.)

Also, you can copy the comma-separated sequence of integers for one of these representations, then paste it at https://oeis.org/ to attempt learning more about it, possibly including a closed form. This is most likely to be successful for the continued-fraction representation. A relevant OEIS hint for the sequence 1, 1, 2, 4, 9, 21, 51, 127 is

> "Enter about 6 terms, starting with the second term. Leave off the first term or two, because people may disagree about where the sequence begins. Don't enter too many terms, because you may have more terms than are in the OEIS data base.

Generalized Expansions and two related tools GCF.TXT by Dougherty-Bliss and Zeilberger and [6] and the Ramanujan Machine by a team of nine authors [14] are intriguing, but adjacent to the topic of this article, so they are not described further here.

5.4 Inverse Symbolic Calculator Plus

Inverse Symbolic Calculator Plus at https://isc.carma.newcastle.edu.au/ was a different interface to essentially the same tables and integer-relations models as ISC, ISC+ has had the message "down for maintenance indefinitely" from late 2018 through at least March 2021. However, there is currently a link there titled "The original ISC" that links to the ISC site already discussed in subsection 5.3.

6 Freely downloadable standalone applications

The implemented integer relation models and precomputed table-lookup entries are based on their implementer's perception of what patterns of nonfloat constant expressions are most likely to fit the combined needs of the intended users. There is a consequent efficiency to the extent that the implementer's choices are appropriate for you. However, such an implementation is almost certain to miss some very concise results fitting unimplemented patterns.

The MESearch and RIES programs address this issue by trying all possible expressions composed of selected rational numbers, symbolic constants, operators and functions – up to a certain total complexity, time limit or memory limit. Both programs use *bidirectional search*: The **forward table** is like the ISC table, and the **backward table** is like the ISC Smart Lookup transformations. However, rather than using *precomputed* forward and transformation tables, for each given float both applications build new tables from your selected components in a breadth-first way, interleaving the searching with this table building. The forward table begins with a selected set simple rational numbers and named constants together with a set of arithmetic operators and functions. More complicated forward expressions are formed by applying all chosen functions and operators to existing entries in an order such that the predicted expression complexity grows approximately monotonically. The backward table starts with the given float, and applies the selected functions in an order such that the estimated resulting complexity

also grows approximately monotonically. For functions and operators of more than one operand, all but one of the operands can be taken from the current forward table.

Growth alternates between the forward and backward tables so that at any one time they have roughly the same number of entries.

Even though the initial set of rational numbers is typically quite simple, arithmetic combinations of rational subexpressions can lead to larger numerators and/or denominator magnitudes.

Both programs use only hardware floating point but do a good job of making the best of that limitation.

Both programs use a few simplification rules such as employing 0 and 1 identities, doing rational arithmetic, and canceling composition of a function with its inverse. However, neither program uses a computer algebra system, so sometimes the results can be usefully simplified manually or by computer algebra.

Computation ceases at a preset expression complexity limit, memory limit, time limit, or the limit of your patience when you finally interrupt a search. Both programs initially use RAM for the tables, but can resort to a much slower secondary-storage mode when allocated RAM is exhausted.

The fact that the tables are exhaustively built and searched in approximate order of increasing expression complexity has a benefit that simple candidates composed of the selected components are almost always quickly found. However, the open-ended exponential growth of the table sizes and computing time with expression complexity makes it impractical to generate expressions having complexity that is easily achievable by specific integer relation models with sufficiently large-precision inputs. With a medium sized set of selected components, exhaustive search can require overnight to generate an expression containing about 11 or 12 instances of rational numbers, symbolic constants, arithmetic operators, and function invocations.

But the peace of mind knowing that the search was exhaustive for expressions composed of the selected rational numbers, symbolic constants, operators and functions is worth the wait.

In contrast, the other software discussed in this article have a fixed number of table entries and/or integer relation models, and the latter can accommodate expressions having any number of predetermined symbolic constants, with arbitrarily large integers determined by PSLQ. But being non-exhaustive, they can and do miss some very simple results that MESearch and/or RIES can determine.

Ignoring the fact that applying some of the transformations to the given float might produce unusable underflow overflow, or non-real results, if the backward table has m entries and the forward table has n entries, then the number of potentially recognizable expressions is mn. If both kinds of table entries averaged the same memory space, then for a given amount of memory, mn is maximized by having $m = n$. For example to recognize 10^{12} expressions, it suffices to have only 10^6 entries in the forward table and 10^6 entries in the backward table.

6.1 MESearch

MESearch is implemented in Java and freely downloadable from

`http://www.xuru.org/mesearch/MESearch.asp`.

The Java Runtime Environment is freely downloadable from `https://www.java.com/en/download/`, and that is often already installed on most computers.

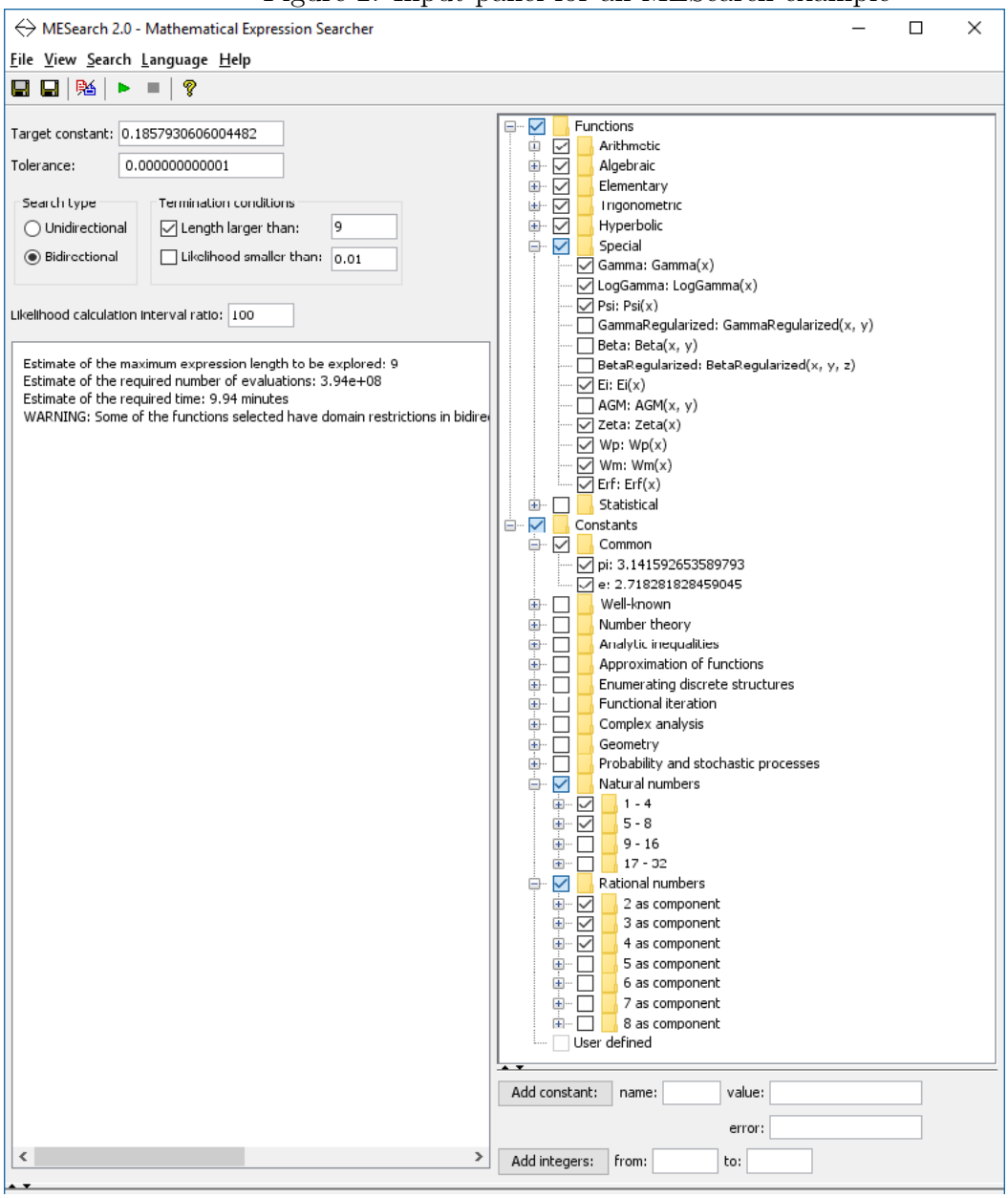
MESearch uses only invertible functions for its backward table, and automatically applies those inverses to the nonfloats in the forward table whose float values closely match the transformed float input.

Figure 2 shows the MESearch input panel for the float input

$$\frac{1}{e^{\operatorname{arccosh}(5)/2} + \sqrt{5}} \approx 0.1857930606004482. \tag{5}$$

MESearch measures expression complexity by **Length**, defined as the number of occurrences of rational numbers, named constants, functions and operators. MESearch offers a choice of operators, functions, named constants, natural numbers and rational numbers from which to build expressions, and Figure 2 shows my most common choices for these. Choosing only a few components permits you to search to longer expressions before exhausting memory or your patience, but it is important to include components that you expect have a nonnegligible probability of occurring in your problem domain. Functions and operators with more than one argument are particularly costly, and that is why I often initially avoid selecting such special functions and statistics distributions. Memory and computing time also grow quickly with the cardinality of the initial set of named constants, natural numbers, and rational numbers. That is why I often initially select only π and e together with very simple natural numbers and fractions.

Figure 2: Input panel for an MESearch example



MESearch only displays candidates whose float value are within ±Tolerance, and the default MESearch Tolerance is 1/2 unit in the last entered digit of your Target constant, which is difficult to achieve if your Target constant is all 16 digits of the hardware floats or nearly so. That is why I often choose about 1000 to 10,000 units in the last entered place for Target floats having 16 significant digits, down to about 10 units in the last place for Target floats having 7 or less significant digits. Generous tolerances change the likelihood estimates and are more likely to generate impostors, but unlike overly frugal tolerances, generous tolerances do not tend to lose true limits. I would rather obtain a true limit together with a *few* impostors, then decide for myself which are most plausible and recompute those with smaller tolerances.

MESearch uses your entered **absolute tolerance** rather than the number of entered digits, so you should enter all of your computed digits if there are less than 16 or rounded to 16 otherwise, even if any number of the resulting trailing digits are dubious. Therefore I entered all 16 digits of 0.1857930606004482, for which my entered tolerance was 10^{-12}.

Here is a summary of the most important information in the resulting output:

Length	Expression	Difference	Likely_1	Likely_2	Likely_3
8	((e^arccosh (5))^(1/2) + 5^(1/2))^-1	0.0	0.5103	0.9135	0.9187
9	arctanh (cos (tan (sin (4/3 / sinh (cos (4))))))	6.769e-13	3.3e-11	0.8515	
9	arctan(ψ(2 - tan(Ei(logGamma((3/4)^2)))))	6.769e-13	3.3e-11	0.8515	

Watching the output panel evolve provides some entertainment while you wait and helps you decide whether and when to terminate the search manually.

The first expression that met the tolerance was at complexity Length level 8. The three Likelihood estimates are based on different mathematical arguments. These likelihood estimates are based on the fact that the expressions are all possible expressions composed of the selected components, computed in approximately monotonic non-decreasing order of complexity. The reason for the two vacant Likely_3 entries is that Likely_3 can be computed only upon **completion** of searching its Length level, and I manually halted the search during Length 9.

Clicking a column heading sorts the rows by the values in that column, so I usually click the middle Likelihood_2, which the implementer Salsamendi states is most often most correct.

If this input float was a mystery float, then having the Difference between the entered float and the value of the Expression, together with two likelihoods of greater 90% and one of greater than 50% for the first row would lead me to strongly suspect that the corresponding expression is the true limit, which it is. In contrast, the very low values for half the likelihoods of the two Level 9 candidates, with their significantly greater Differences would lead me to strongly suspect that they are impostors.

Salsamendi [16] describes the data structures, algorithm, and time complexity.

6.2 RIES (Rilybot Inverse Equation Solver)

RIES is implemented in C, and the source code is freely downloadable from `https://mrob.com/ries` . Your computer might already have a C or C++ compiler that you are unaware of, and there are several good free ones for Mac OS X, the Unix family, and Windows.

RIES optionally exploits 19-digit IEEE binary80 arithmetic if your CPU and C compiler support it, and 19 digits is a valuable increment over the typical 16-digit hardware. IEEE binary80 is supported by Intel x86 processors, but not by current or past AMD or ARM processors. The Intel C++ compiler and the free GNU C compiler support this arithmetic, and Microsoft Visual C can be made to do so by inserting _controlfp_s function calls at appropriate places in the source code.

Here is an example by Bill Gosper (private communication):

Mathematica 12.1 cannot determine a closed form for

$$\sum_{n=0}^{\infty} \frac{(-1)^n \cos\left(\sqrt{\frac{1}{2}+\left(\frac{1}{2}+n\right)^2}\,\pi\right)}{\left(\frac{1}{2}+n\right)\left(\frac{1}{2}-\frac{1}{\sqrt{2}}+n\right)\left(\frac{1}{2}+\frac{1}{\sqrt{2}}+n\right)}$$

Suppose that you somehow compute a 20-digit approximate value 7.0895773641597344051, and you invoke RIES as the command line

```
ries  -l7  7.0895773641597344051  -s
```

where -l7 and -s are option settings.

The following lines (slightly edited) are displayed as they are computed:

```
Your target value: T = 7.08957736415973441  mrob.com/ries

x = pi + 4 for x = T + 0.0520153 {49}
x = 4 sqrt(pi) for x = T + 0.000238039 {58}
x = 1/(8 + pi) + 7 for x = T + 0.000176411 {81}
x = (sqrt(pi) + pi)/ln(2) for x = T - 0.000106841 {87}
x = e^(e^sqrt(phi))/5 for x = T - 8.60481e-05 {90}
x = ln(ln(6^8))^2 for x = T + 7.23949e-05 {93}
x = (phi^2)^sqrt(1 + pi) for x = T - 4.72474e-05 {85}
x = (e^(pi - 2))^2 - e for x = T + 1.25979e-05 {90}
x = (1/pi + sqrt(pi))^2 + e for x = T - 2.53137e-06 {92}
x = 3"/phi + 3^phi for x = T + 2.27427e-06 {98}
x = e (sinpi(1/ln(9)) + phi) for x = T + 2.0796e-06 {109}
x = 3"/(log_3(1 + pi)) + 6 for x = T - 1.32453e-08 {111}
x = 3/pi + 8 - pi"/x for x = T + 5.08817e-10 {113}
sinpi(sqrt(x)/e) = (1/(8/phi - 1))^2 for x = T - 3.68034e-10 {131}
x = 3((pi"/4 - 1/7) + phi) - 2 for x = T - 3.51782e-11 {141}
x = 2 e"/(1 + e) - 1/(1 - 3"/2) for x = T + 1.65079e-11 {147}
x = ((log_9(1/2 + 8))/4)/e + 7 for x = T - 1.3897e-11 {150}
x = -(pi/cospi(1/sqrt(2))) - pi cospi(1/sqrt(2)) for x = T - 5.28657e-16 {148}
x = pi (1/sinpi(1/sqrt(2) - 1/2) - cospi(1/sqrt(2))) for x = T + 3.05745e-16 {154}
x = (1/tanpi(1/sqrt(8))^4 + 1)(x/2 + pi) for x = T - 1.30104e-18 {164}

(Stopping now because best match is within 3.07e-18 of target value.)

log_A(B) = logarithm to base A of B = ln(B) / ln(A),  cospi(X) = cos(pi * x)
e = base of natural logarithms = 2.71828...,  sinpi(X) = sin(pi * x)
ln(x) = natural logarithm or log base e,  tanpi(X) = tan(pi * x)
phi = the golden ratio,  (1+sqrt(5))/2 sqrt(x) = square root
A"/B = Ath root of B  pi = 3.14159...
```

	--LHS--	--RHS--	-Total-		
max complexity:	86	81	167		
dead-ends:	363904522	872434773	1236339295	Time: 204.127	„
expressions:	25836915	59427019	85263934	Memory: 1351040KB	
distinct:	7974910	9198455	17173365		
Total tested:	(7.336e+13)				

The integers in braces at the right ends of lines beginning "x =" are the complexities, which are almost monotonic too.

Notice that

- RIES rounded the 20 digit input to 1 digit less than the 19 digit arithmetic it is using on this computer.
- RIES displays a candidate only if its absolute difference from the entered float is less than that for the previously displayed candidate. Since candidates are generated in approximate order of non-decreasing complexity, the sequence of displayed candidates is approximately the Pareto set of optima for the conflicting objectives of large agreement and small complexity. Thus usually all but the last few of the displayed candidates are possibly-useful approximations rather than plausible candidates for the limit you seek. But as with the subsection 6.1, the very last few can be overfit impostors.
- For the line `sinpi(sqrt(x)/e) = (1/(8/phi - 1))^2` it is your responsibility to solve this equation for x, giving

$$x = \frac{e^2}{\pi^2}\left(\arcsin\left(\frac{-3-\sqrt{5}}{5\left(-23+3\sqrt{5}\right)}\right)+2\pi n\right)$$

$$x = \frac{e^2}{\pi^2}\left(\arcsin\left(\frac{-3-\sqrt{5}}{5\left(-23+3\sqrt{5}\right)}\right)+2\pi(n+1)\right)$$

  where n represents an arbitrary integer, then approximate these for various integer n values to determine which alternative equation and what value of n corresponds to your input float. The first alternative with $n = 0$ agrees very closely with the input float.
- For the line `x = (1/tanpi(1/sqrt(8)))^4 + 1)(x/2 + pi)`, it also your responsibility to solve this equation for x, giving

$$x = -\frac{2\pi\left(1+\cot\left(\frac{\pi}{2\sqrt{2}}\right)^4\right)}{-1+\cot\left(\frac{\pi}{2\sqrt{2}}\right)^4}.$$

- In all of the other result lines, RIES was able to uniquely invert the backward table entries to produce an explicit solution directly.

- Approximation of the explicit formulas for the last three candidates to larger precision suggests that they are almost certainly equivalent. The fact that the reported discrepancy is two orders of magnitude less for the last result is probably attributable to the fact that the formulas were not computed with larger precision arithmetic than the input precision.

RIES uses a particularly compact data structure that allows it to build particularly large tables before having to resort to slower secondary storage. Compiling with extended precision reduces the maximum achievable table sizes and is slightly slower, but the greatly increased number of recognizable constants makes it worthwhile. To maximize your coverage, compile both a 19-digit and 16-digit version, then try your second choice if your first choice doesn't return a satisfying result.

7 Functions built into computer algebra systems

7.1 The Maple identify (...) function

Sometimes you want to use a result of a downloadable application in another piece of software. For that purpose, functions are usually better than applications or web applications – because functions have a single output and have no side effects, making them composable, with results that are the same regardless of temporal reorderings due to exploiting commutativity, associativity, etc. Also, functions do not require interactive keystrokes.

The Maple computer algebra system has a function named identify (...) that can take a float as an argument and returns either one nonfloat constant or the input float if the function cannot determine a nonfloat that the function judges sufficiently likely to be the limit you seek. This function is an adaptation by Kevin Hare of a Maple application by Alan Meichsner [11, 4], based on the PSLQ integer relation algorithm.

Default Maple floats are arbitrary-precision with only 10-digit significands. Therefore you must almost always use more precision to have a reasonable chance of success.

The fact that identify (...) results are either the input float or a nonfloat that has very nearly the same value enables identify (...) to map automatically over the parts of non-real constants and over non-constant expressions, trying to replace all or at least some of any floats therein with close nonfloat constants. For example,

$$\text{identify}\left(\frac{0.27675082 - 1.0369278\,I}{8.3118730x^2 + 11.94535128y}\right) \rightarrow \frac{0.27675082 - I\zeta(5)}{\sqrt{7}\pi x^2 + 4e\ln(3)y}$$

where $I = \sqrt{-1}$ and ζ is the Riemann zeta function.

The one float that identify did not convert was a random float. In contrast to applications that allow you to inspect alternatives and reject them all, a function such as identify (...) automatically replaces any number of floats in an expression without your inspection of their complexity and agreement. Therefore it is appropriate for such functions to be parsimonious about the qualifications of replacements it accepts. This makes it a good sign that the random float was not replaced by a candidate that would almost certainly be an impostor.

7.2 The MPMath, SymPy and Sage identify (...), findpoly (...), and nsimplify (...), functions

Sage is a software system freely downloadable from `http://www.sagemath.org/` that includes NumPy, SciPy, matplotlib, SymPy, Maxima, GAP, FLINT, R and other packages. The SymPy computer algebra system in turn includes MPMath, which has an identify (...) function analogous to Maple's, except that identification of algebraic numbers is done by a separate function named findpoly (...). The SymPy system also has an nsimplify (...) function somewhat analogous to the NSimplify [...] component of AskConstants described in subsection 8.1.

8 Functions and applications for CAS

8.1 AskConstants and associated functions for *Mathematica*

AskConstants is an application that I implemented in *Mathematica*, freely downloadable from `AskConstants.org` . AskConstants has about 3000 integer relation models, and bidirectional search with a choice of precomputed tables for which the largest backward table has about 5 million entries and the largest forward table has about 14 million entries. The table lookup exploits sign and base two significand aliasing, with automatic dealiasing and inversion for close matches.[7] Thus ignoring overflow underflow, nonreal compositions and the aliasing bonus, the largest tables cover about $(5\times10^6)(14\times10^6 = 7\times10^{13}$ expressions. The backward tables and half of the largest forward table are tabulated constants. The other half of the largest forward table was generated by exhaustive table building with the elementary functions and the most commonly-occurring special functions to exploit advantages of precomputed tables such as ISC and exhaustive breadth-first tables of MESearch and RIES.

Figure 4 shows the use of version 4.1 to propose a closed form for the definite integral

$$\int_0^\infty \frac{dx}{\sqrt{1+x^2}+(1+x^2)^{7/2}}$$

that the *Mathematica* 12.1 Integrate function cannot determine.[8]

The input field contains the *Mathematica* NIntegrate function to compute the approximate float value displayed below the digit ruler. More commonly, the numerical integration would be done in a notebook, with the float result copied then pasted into the AskConstants input field

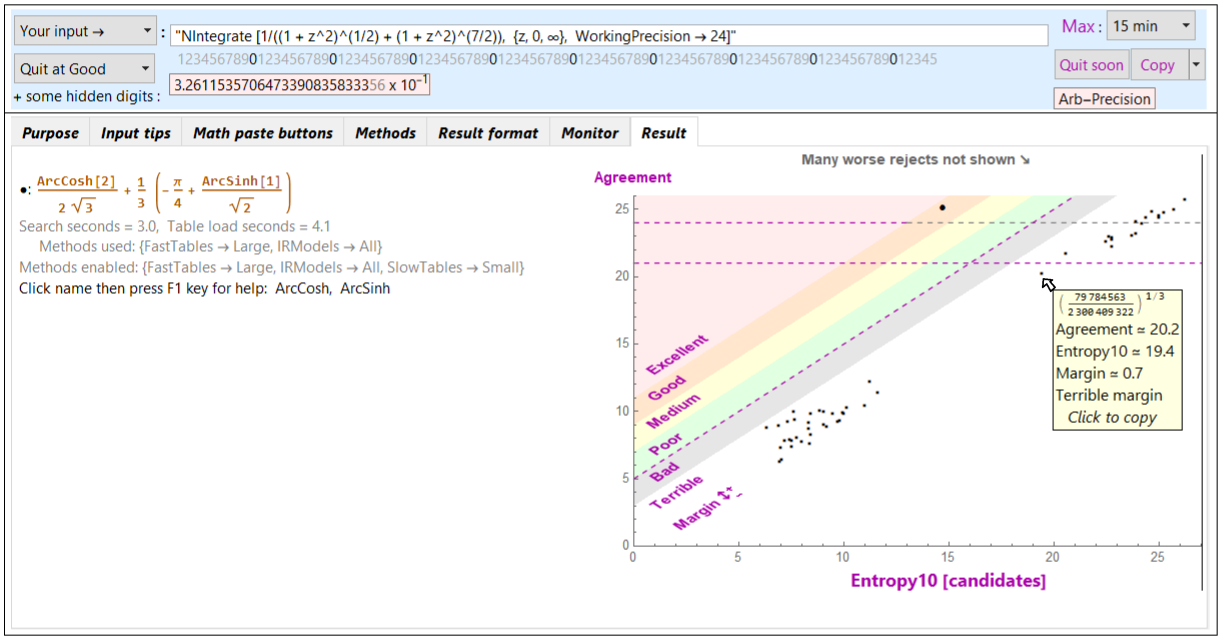
AskConstants displays the candidate result

$$\frac{\text{ArcCosh}\,[2]}{2\sqrt{3}}+\frac{1}{3}\left(-\frac{\pi}{2}+\frac{\text{ArcSinh}\,[1]}{\sqrt{2}}\right). \tag{6}$$

[7]The bonus for base 2 aliasing exceeds that of base 10 aliasing because 2 divides a random reduced numerator or denominator more often than 10 does.

[8]I thank Daniel Lichtblau for this example.

Figure 3: AskConstants proposes a formula that agrees with 3.26115... to 24 digits



on the left side of the Result tab and displays an associated scatter plot of Agreement versus Entropy10 on the right side, where:

- The **Agreement** of a candidate nonfloat is **the number of digits that agree** with the input float within 1/2 unit, up to the length of the input significand, including any stored digits that are not ordinarily displayed.

- **Entropy10** is a measure of nonfloat expression complexity that is the sum of the base-ten logarithms of the absolute values of all the nonzero integers occurring in an expression, plus about 1.0 per occurrence of a named constant, function or operator – counting duplicates. The implicit units of Entropy10 are also digits.

- The scatter plot shows the Agreement versus Entropy10 for the one accepted candidate as a large dot, with small dots for some computed but rejected candidates for comparison.

- It is desirable to have high agreement, but it is also desirable to have low expression complexity because floats can be approximated arbitrarily well by sufficiently complicated nonfloats that are impostors rather than the limit you seek.

- Therefore

$$\text{Margin} := \text{Agreement} - \text{Entropy10} \tag{7}$$

  combines these two competing objectives into one of maximizing Margin, with the largest Margin in the upper left corner of the plot and the smallest Margin in the lower right corner.

- Diagonal Margin intervals are rainbow color-shaded and labeled with adjectives from "Terrible" through "Excellent" as a qualitative guide to the likelihood that a candidate is the limit you seek rather than an impostor.[9]

- Typically, a lower left cluster of rejects is mostly from table lookup, and an upper right cluster is mostly from integer relations.

- As illustrated, hovering the mouse over a plot point lists the nonfloat candidate together with its coordinates and Margin.

- The dashed diagonal line is the lowest acceptable Margin.

- The upper horizontal dashed line is the Precision of the input float estimated by *Mathematica*'s arbitrary-precision significance arithmetic.

- The horizontal dashed line three digits of Agreement below it is the smallest acceptable Agreement for the Precision of the given float.

- The large accepted plot point has an Agreement one digit greater than the input Precision because the first normally-undisplayed guard digit in the significand also agreed with the value of the nonfloat computed to larger precision. This "home team advantage" for arbitrary precision floats computed in the AskConstants input field or copied and pasted in the same *Mathematica* session does not occur for floats copied from outside *Mathematica* or typed without guard digits.

- Notice that the reject in the extreme upper right corner has even larger Agreement than the accepted result. However, its nonfloat is

$$\pi \text{ArcTan}\left[\frac{168443027133}{1616842785493}\right].$$

  Although this could be the exact expression we seek, such large rational number numerators and denominators compared to any in the integrand or integration limits are typical of integer relation overfits. That is why basing acceptance on Agreement alone is especially problematic for integer relation results.[10]

- More generally there might be several accepted candidates, and ones judged equivalent by the *Mathematica* PossibleZeroQ function are joined by line segments, which might help you assess them more efficiently.

Automatic of the backward transformation requires an inverse function for every function in the transformation; and as with most computer algebra systems, *Mathematica* has very few inverse special functions – probably because they are extremely difficult to implement for nonreal arguments. However, AskConstants directly addresses only real floats, and

[9]Candidates are not produced in approximate order of increasing complexity, so the helpful likelihood estimates of MESearch are inapplicable. The qualitative adjectives "Terrible" through "Excellent" are based on my experience with test examples.

[10]It can also be problematic for table lookup, but less frequently.

it is not too difficult to implement most inverse special functions for real arguments and results. Consequently I did that for about 40 special functions. Many of those are multi-branched, such as for BesselJ, Gamma, and Zeta. Consequently I also implemented corresponding functions that return the abscissa and ordinates of the infima and suprema of those functions to accomplish piecewise monotonic decompositions of the functions being inverted. The real inverse and infima or suprema functions are separately usable, as are some additional functions analogous to the *Mathematica* BesselJZero function.

To overcome the limitation to constant expressions, real arguments and results, AskConstants also contains a ProposeBestOrInput function that maps over general expressions and over the real and imaginary parts of non-real constants similar to the Maple identify function.

8.1.1 NSimplify [...]

Anyone who tests the programs described here with constants having known nonfloat representations might eventually notice that surprisingly often an equivalent proposed candidate is simpler than the published expression. This is because:

- Manual derivations and default computer algebra simplification often produce incompletely simplified expressions.[11]
- Sometimes there is no composition of builtin optional computer algebra transformations that can produce a particularly simple representable equivalent that exists.

Consequently the AskConstants download includes an NSimplify function that supplements the *Mathematica* FullSimplify function by approximating a nonfloat input expression as a float, then applying the ProposeBestOrInput function to that float. If it returns a nonfloat having smaller Entropy10 than the given nonfloat and the *Mathematica* PossibleZeroQ function judges that the difference between the original expression and proposed replacement is 0, then the original is replaced. If that doesn't succeed, then this process is recursively applied to subexpressions. NSimplify also tries FullSimplify so that the result is at least that successful. This brute-force combination is slow, but it can achieve some dramatic simplifications. For example, applying NSimplify to

$$\frac{\sqrt{\frac{1}{2}-\frac{1}{4\sqrt{\frac{2}{4+\sqrt{7-\sqrt{5}+\sqrt{30-6\sqrt{5}}}}}}}}{\text{Root}\left[-97+448\,\#1-672\,\#1^2+560\,\#1^3-280\,\#1^4+84\,\#1^5-14\,\#1^6+\#1^7\,\&,1\right]}$$

produced the result

$$\frac{\sin\left(\frac{7\pi}{120}\right)}{2-31^{1/7}};$$

and the optional trace reported the steps

[11] Sometimes dangerously so, as described in [19].

• Level 2: ProposeBestOrInput $\left[N\left[\frac{1}{2}-\sqrt{\frac{1}{4\sqrt{\cdots}}},26\right]\right]\mapsto\sin\left(\frac{7\pi}{120}\right)$,
reducing Entropy10 by 14.1; and PossibleZeroQ [difference] $\mapsto$ True.

• Level 2: ProposeBestOrInput $[N[\text{Root}[-97+\cdots+\#1^7\,\&,1]],38]\mapsto\frac{1}{2-31^{1/7}}$,
reducing Entropy10 by 18.6; and PossibleZeroQ [difference] $\mapsto$ True.

8.2 Plouffe's inverter for Maple

After developing ISC, Simon Plouffe developed Plouffe's Inverter [13], a similar Maple application with larger tables. As of August 2019, the backward tables contains 405 transformations. The largest version of the forward tables current contains 69×10^9 entries on his largest computer, and it continues to grow. Ignoring overflow, underflow, non-real values, sign aliasing and significand aliasing, that version can potentially match $405\times69\times10^9\approx3\times10^{13}$ expressions. That version is too large for practical distribution, but if you have already tried other systems and desperately want to exhaust your possibilities, then you can try sending your float constant to him to try.

In contrast, his portable version of the application contains 201 million forward table entries. Thus his portable version can match about $405\times201\times10^6\approx8\times10^{10}$ expressions, which is about 24 times as many expressions as Inverse Symbolic Calculator, which can match about $100\times30\times10^6=3\times10^9$ expressions. The Maple program without the tables is freely downloadable text that you can copy from `http://plouffe.fr/PlouffeInverter2017.txt` then copy into a plain text editor such as the Windows accessory Notepad, then save as file PlouffeInverter.

There are two choices for the tables. The easiest one, which you might want to try first, is to insert the following fragment into your PlouffeInverter Maple program to use the ISC forward table remotely:

```
with(Sockets):
server := ‘wayback.cecm.sfu.ca‘;
s := Sockets:-Open(server, 80);
Sockets:-Write(s, cat(‘GET /cgi-bin/ipcgi/lookup.pl?number=‘,
valeur,‘&lookup_type=simple\n‘));
text := ReadPage(s);
Sockets:-Close(s);
```

For this alternative the inverter has about four times as many transformations as ISC, so you will be able to match about four times as many expressions.

The other choice is to download about 9,000 GNU-zipped forward table files totaling about 370 gigabytes from `http://plouffe.fr/ip/`, then unzip them with a compatible unzipper. When unzipped they require a total of slightly more than a terabyte of secondary storage. You will need a fast reliable internet connection and a sufficiently large fast USB-connected external drive, which costs less than $100 USA in 2019.

Either way, there are some file-paths that must be changed, as indicated in comments near the beginning of the program.

Also, his inverter uses the Unix look command, which does a surprisingly fast search for a leading substring in a sorted file of strings that are not necessarily all the same length. If your computer does not already have that part of the Unix-family operating system, then one easy compact way to acquire the look command is to download the free Sage mathematical software system, which is independently advantageous.

9 Custom Integer Relation Models

Mystery floats often occur at the high end of a family of problems depending on a parameter n, with known nonfloat values for the low end. For example, a noticeable number of definite integrals have nonfloat representations that can be expressed as a rational linear combination of terms with cofactors that are small positive integer powers of $\ln 2$, $\ln 3$, $\zeta(m)$ with small positive integer m, and low-order polylogarithms with simple arguments such as $1/2$ or $1/4$, perhaps multiplied by $\sqrt{2}$ or $\sqrt{3}$. The known closed forms for small n permits a guess of possible cofactors for the next value of n.

An increasing number of computer algebra systems have builtin integer-relation solvers, and David Bailey and David Broadhurst [[1, 2]] describe some particularly efficient implementations in Fortran 90 and C++. Such models are too time consuming to make them part of the set of more general-purpose integer relation models in tools such as WolframAlpha, AskConstants, or the Maple and MPMath identify functions. However, it is easy to simply invoke a built-in or stand-alone PLSQ function with an input vector containing your float and a set of cofactors.

10 The curse of extreme magnitude

The software described here is most successful at proposing candidate nonfloats for floats whose magnitudes are not extremely different from 1.0. A reason for this is that representations of nonfloat constants having extreme magnitudes often require high-complexity extreme magnitudes of the numerators or denominators of some rational numbers therein, which requires extreme precision input floats for integer relation algorithms or prohibitively extreme magnitude integers in any lookup tables. Infrequent exceptions are functions whose nonzero magnitude can be extremely large or small for arguments that are not extremely large or small, such as $\Gamma(98/3) \approx 8.26 \times 10^{34}$ or $\operatorname{erfc}(9) \approx 4.14 \times 10^{-37}$.

Fortunately, most published mathematics constants do not have extreme magnitudes. For example, in Steven Finch's table of about 10,000 such constants, nonzero magnitudes vary from 0.000111582 through 137.0359 with median about 0.9 and quartiles about 0.4 and 1.9 .

11 Some causes of impostors

"1.0000001 *is a crowd*"

– adapted from James Thurber

Reasons for impostors include:

1. Many functions $f(x)$ have a stationary value of 1 for some value of x. For example, $\cos x$, $\sec x$ and $\cosh x$, at $x = 0$; or $\tanh x$ and $\operatorname{erf}(x)$ as $x \to \infty$. In a neighborhood of the stationary point, such constants that are modeled are likely to occur as impostors for any that are not modeled, because very low complexity x can produce values very close to 1.0 and hence each other. For example, sec(1/999) and cosh(1/999) differ by only 2 units in the 14th place.

2. If for some relatively low complexity nonfloat constants x a modeled expression $f(x)$ agrees closely with an unmodeled expression $g(y)$, then that makes it easy for the modeled expression to be an impostor for the unmodeled expression. For example the low complexity expressions, e^{18}, $2\sinh 18$, and $2\cosh 18$ differ by only 1 unit in the last of 16 places. If a proper subset of these is modeled and the true limit is unmodeled, then one of the modeled ones will almost certainly occur as an impostor having nearly the same agreement and complexity as the correct limit.

12 Conclusions

There are many good tools that can propose nonfloat candidates that your float closely approximates. Usefully often one of those candidates is the limit that your float would approach as the working precision increases. Some of the tools are easy to use directly on the internet, some are built-into a computer algebra system, and others are easy to download and install.

Most of the programs discussed in this article can propose correct candidates that none of the others can propose. Search engines, radix search, integer relation models, precomputed tables and exhaustive run-time tables are each best at overlapping kinds of expressions. Therefore it is worthwhile to try as many of these programs as is reasonably convenient.

Therefore it is wise to try at least:

- the Online Encyclopedia of Integer Sequences,
- at least one of the web search engines on your computers and smart phones,
- one of the tools that identify algebraic numbers, returning results such as Root [*polynomial*, *n*] and have numerous integer-relation models (AskConstants, Maple identify, SymPy identify and WolframAlpha),
- one that has precomputed lookup tables (AskConstants, Inverse Symbolic Calculator, and Plouffe's Inverter),
- one that uses exhaustive breadth-first search (MESearch and RIES)

At the very least you should try all that are built into the computer algebra systems that you already have, together with all of the web-based tools (a browser's search engine, the Online Encyclopedia of Integer Sequences, Inverse Symbolic Calculator and WolframAlpha) because that is so easy to do.

The success of your efforts depends strongly on knowing how to groom your float for input, interpret the results, and possibly de-alias and transform the result. These details vary greatly among the tools, and the descriptions of these tools here are intended partly as brief user guides that you can review for tools that you are not already familiar with.

When none of these programs accessible to you can propose a plausible constant, then at least you can have the peace of mind of greatly reducing the chance that your problem has a simple nonfloat result that you did not find.

Acknowledgments

Thank you Daniel Lichtblau, Robert Munafo, Simon Plouffe, Neil Sloane and Michael Trott for your helpful suggestions.

References

[1] Bailey, D. H., The PSLQ Algorithm: Techniques for Efficient Computation (slides), 2010, `https://www.davidhbailey.com/dhbtalks/dhb-carma-20100824.pdf`

[2] Bailey, D. H. and Broadhurst, D. J., Parallel Integer Relation Detection: Techniques and Applications, 1999, `https://arxiv.org/pdf/math/9905048.pdf`

[3] Borwein, J. and Borwein, P., *A Dictionary of Real Numbers*, Wadsworth Inc., 1990.

[4] Borwein, P., Hare, K. G., and Meichsner, A., Reverse Symbolic computations, the identify function, `https://www.researchgate.net/publication/267425704_REVERSE_SYMBOLIC_COMPUTATIONS_THE_IDENTIFY_FUNCTION`

[5] Corless, R, Gonnet, G., Hare, D., Jeffrey, D., Knuth, On the Lambert W function, *Advances in Computational Mathematics*, 5, (1996), pp. 329–359.

[6] Dougherty-Bliss, R. and Zeilberger, D., Automatic conjecturing and proving of exact values of some infinite families of infinite continued fractions, 2020, https://arxiv.org/pdf/2004.00090.pdf

[7] Ferguson, H. R. P. and Bailey, D. H., A Polynomial Time, Numerically Stable Integer Relation Algorithm, RNR Technical Report RNR-91-032, July 14, 1992.

[8] Ferguson, H. R. P., Bailey, D. H., and Arno, S., Analysis of PSLQ, an integer relation finding algorithm, NAS Technical Report NAS-96-005, NASA Ames Research Center, Moffet Field, CA. April 1996 and *Mathematics of Comput*ation 68 (1999) 351–369.

[9] Finch, S. R., *Mathematical Constants, Encyclopedia of Mathematics and its Applications*, Cambridge University Press, 2003.

[10] Finch, S. R., *Mathematical Constants II, Encyclopedia of Mathematics and its Applications*, Cambridge University Press, 2018.

[11] Meichsner, A., *Integer relation algorithms and the recognition of numerical constants*, M.S. Thesis, Simon Fraser University, 2001. http://www.collectionscanada.gc.ca/obj/s4/f2/dsk3/ftp04/MQ61592.pdf

[12] Plouffe, S., Credits and References for Inverse Symbolic Calculator. http://wayback.cecm.sfu.ca/projects/ISC/credits.html, 1995.

[13] Plouffe, S., L'Inverseur, March 1998, http://vixra.org/pdf/1409.0151v1.pdf

[14] Raayoni, G. et al., The Ramanujan Machine: Automatically generated conjectures on fundamental constants, 2020, https://arxiv.org/pdf/1907.00205.pdf

[15] Robinson, H. P. and Potter, E., *Mathematical Constants*, UCRL-20418, UC-32 Math. and Comp., TID-4500 (57th Ed.), March 1971.

[16] Salsamendi, J. Z., An efficient mathematical expression searcher for constant recognition and some conjectures and theorems discovered with it, 2013, http://www.xuru.org/downloads/papers/MESearch.pdf

[17] Salvy, B. and Zimmermann, P., GFUN: a Maple package for the manipulation of generating and holonomic functions in one variable, ACM Transactions on Mathematical Software, 20(2), June 1994, pp. 163-177.

[18] Shamos, M. I., Shamos's, Shamos's Catalog of the Real Numbers, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.366.9997&rep=rep1&type=pdf

[19] Stoutemyer, D. R., Ten commandments for good default expression simplification, *Journal of Symbolic Computation* 46(7), July 2011, pp 859-887.