What, if anything, is epsilon?

Dr. Tom Murphy VII Ph.D.*

1 April 2014

Abstract

We present a sample of the values of the programming constant `epsilon` as found on the internet, for several different programming languages and with a variety of visualizations.

Keywords: computational archaeology, epsilon, very-small and medium-small numbers

Introduction

Epsilon, the all-spelled-out version of ϵ, although properly "epsilon" because it's the *lowercase* Greek letter, though it's not like I'm going to start my paper with a lowercase letter even if it's technically correct, since I like to wait at least until the second or third letter of the paper before the reader starts doubting that I can write or spell or have shift-keys on my keyboard, anyway epsilon is a mathematical symbol denoting a *very small number*.

In mathematics, ϵ usually refers to the ϵ–δ formulation of limits. This is pretty simple and a reminder appears in Figure 1. This paper is not about that kind of math.

In computing, ϵ is used in a much more general sense to just mean some small number or error bound. For example, two numbers are often considered equal if their absolute difference is less than ϵ.

In IEEE-754 floating-point [2], the standard way that computers represent "real numbers," there is a specific formal value called "machine epsilon", or "unit roundoff". It is the maximum (relative) amount of error from a single rounding operation. This number is useful if you want to do numerical programming and be careful about what you're doing.

Most programmers find this subject too tedious and simply pick a number that seems pretty small. Thus in practice, ϵ is an application-specific choice. Of well-known "constants," ϵ may be the least agreed-upon, with values seen in the wild spanning more than *300 orders of magnitude*. This paper explores the practical values of ϵ in real software.

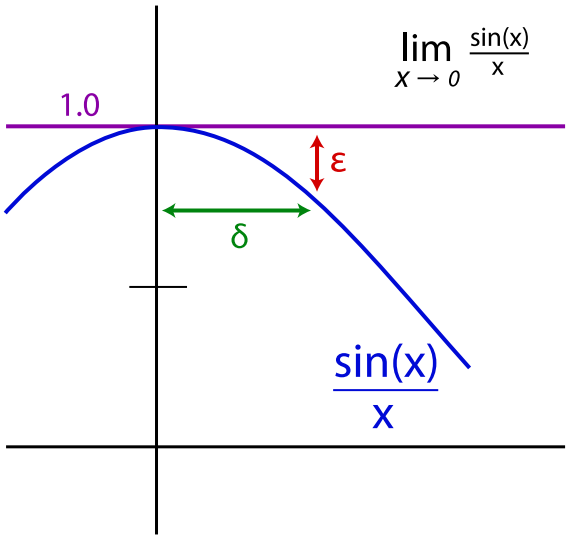

Figure 1: The curvy line is the function $\frac{\sin x}{x}$, which is undefined at 0. However, its limit as x approaches 0 is 1. The ϵ–δ formulation of the limit is this: For any positive choice of ϵ, there exists some δ such that $\frac{\sin(0+\delta)}{0+\delta}$ is less than $1+\epsilon$. In other words, for an arbirarily small error (your choice), I can produce a delta from 0 (my choice) that brings the result within the error of the limit. This has nothing to do with the subject of the paper.

1 Methodology

Github. Github is the hub that contains all gits, approximately 10 million of them, as of the beginning of

2014. The site has search functionality, which "allowed" me to scrape one hundred pages of results for queries like `"const double epsilon ="` for various languages, as long as I didn't do it too fast. I scraped the programming languages C, C++, C♯, JavaScript, and Objective C. Each language has its own idiosyncracies about how constants are defined, so I used one (or several) appropriate to each language. For example, in JavaScript, I looked for `"var epsilon ="`. From these HTML files I extracted all of the right-hand-side expressions, manually excluded the ones that could not be evaluated (for example because they depended on other symbols; see Figure 2 for some examples), and then computed the actual values for the rest. The source code to do the scraping, extract the expressions, and tally the results is available online.[1]

```
0.5/ELEC_REST_ENERGY
alpha/beta
4 / MULT32
exact_epsilon(true)
fmass_Epsilon * EPS_EXTRA
((Lj_Parameters*) parameters)->
scalar_traits<
EpsArray[prec]
hfwfn_->
fl.net_.opt_.epsilon
Tolerance
```

Figure 2: Other uninterpretable values of `const double epsilon` in C. Who knows what these are supposed to be?

SPEC benchmarks. Did you know that the SPEC benchmarks [1] cost $800? Like they literally expect me to pay them money to download the source code so that I could grep for `const double epsilon` or test my compiler out on that. Many are even based on open-source software like Sphinx and POV-Ray. Ridiculous. I refuse. Values of epsilon for the SPEC benchmarks do not appear in Figure 3.

2 Results

The results of the analysis appear in several figures which are interspersed haphazardly with this text. Each figure presents the data in a different way, since this diversity in presentation should maximize the chance that

[1] In the Subversion repository at: `https://sourceforge.net/p/tom7misc/svn/HEAD/tree/trunk/epsilon/`

Figure 3: $800? Fuck that!

one of the charts makes sense to you. The C programming language has two numeric types that could reasonably be used to represent ϵ: `float` and `double`. The results for `double` appear in Figure 4 and for `float` in Figure 5. C++ has those same two types, but I decided arbitrarily to only look at `double`, which is in Figure 6. Programmers in C♯ are very creative; their results are presented in Figure 7. Objective C proved unpopular for use of ϵ, its sparse data are in Figure 8. Finally, the ineffable JavaScript has its results in Figure 9.

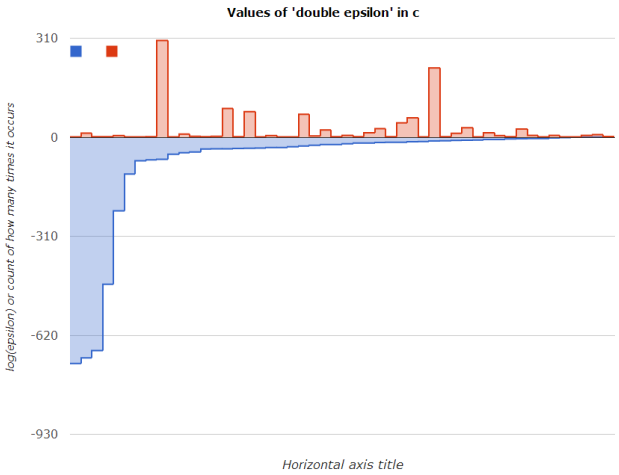

Figure 4: Values for `const double epsilon` in the C programming language. In this chart, the blue (lower) bars are the distinct values of epsilon seen. The vertically aligned red (upper) bar is its count. Epsilon values are plotted on a logarithmic scale, where the minimum observed value $\log(-708)$ is 303×10^{-308}, and the largest $\log(1.609)$ is 5. Notes: One programmer used the value `-1e10`, which is -10,000,000,000, probably meaning `1e-10`. This value was excluded because it has no real logarithm.

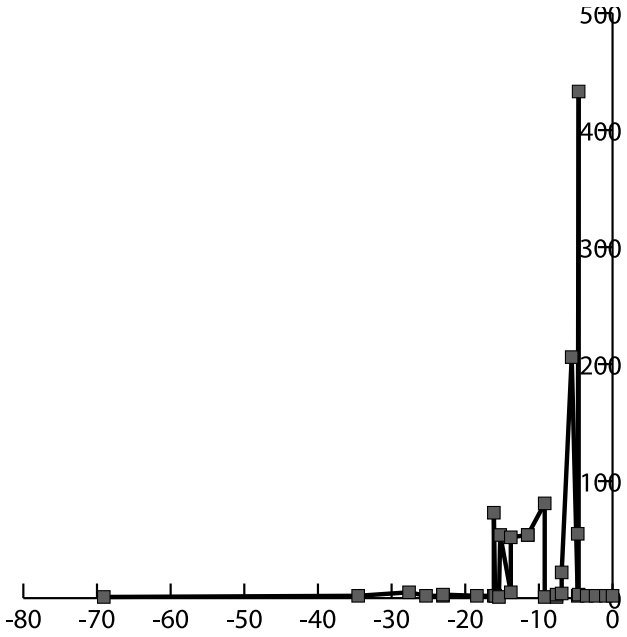

Figure 5: Values for `const float epsilon` in the C programming language. An x–y scatter plot where the y coordinate is the count of the number of times that specific value occurred, and the x coordinate is the log of the value. Values take on a less extreme range than with type `double`, naturally, ranging "only" 31 orders of magnitude from 3.9×10^{-31} to 0.

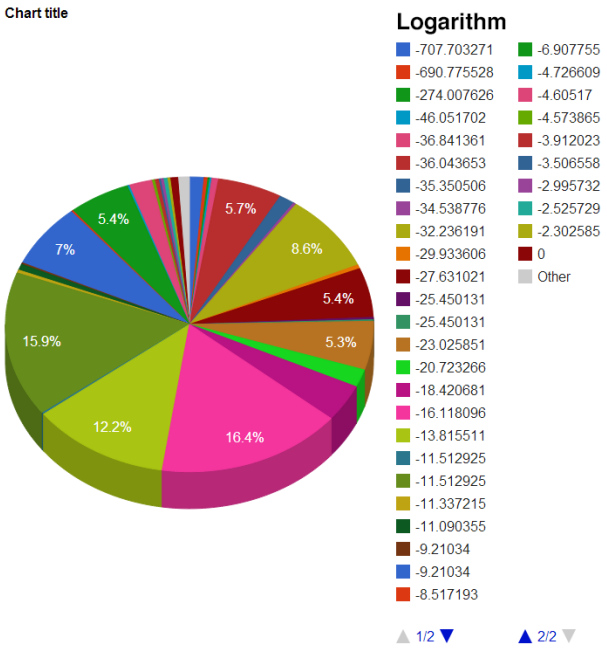

Figure 6: Values for `const double epsilon` and `constexpr double epsilon` in the C++ programming language. The `constexpr` qualifier, a new feature of C++11, is used very rarely (less than 1% of the time). Notes: Five times, the programmer used 0.0 for epsilon, which is possibly the only unjustifiable value. The value `pow(10,-13)` is annotated in German "Genauigkeitsziel bei der Nullstellensuche," or "Accuracy goal in the search for zeros."

References

[1] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, September 2006.

[2] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic.* IEEE, New York, NY, USA, August 1985. Revised 1990.

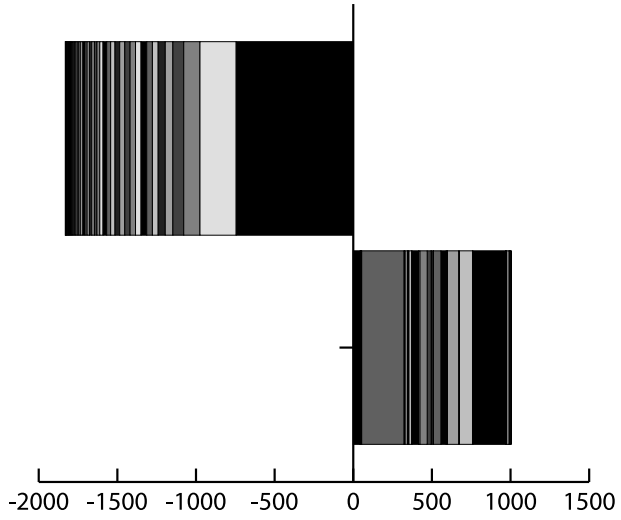

Figure 7: Values for `const double epsilon` in the C♯ programming language. In this chart, the left edge of greyscale blocks on the left-facing flag represent the cumulative value of the logarithm of all values of epsilon seen so far. The right edge of the block with the exact same greyscale value on the bottom-right flag is the cumulative count of all definitions of epsilon so far. It is easy to see that there are 1,000 total definitions, as expected. Notes: One programmer defined epsilon as `1.0-18`, that is, -17, probably meaning `1.0e-18`. Four used literal `0` for epsilon. Another defined epsilon as `1 / 100000`, which uses integer division and results again in 0. Owing to the strength and creativity of C♯ developers, here we saw our largest value of epsilon so far, `700`, and the declaration with the most significant digits: `0.00000000000000022204460492503130808472633361816406 25`.

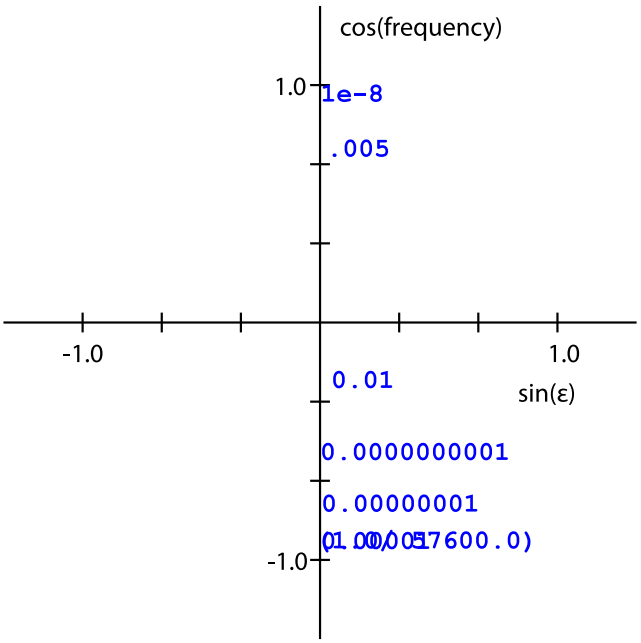

Figure 8: Results for the language Objective C for `const double epsilon`. Perhaps because Objective C programmers do not use this idiom for defining constants, or because it is only used to make iPhone games where epsilon is not a concept of interest, there were only 8 distinct values of epsilon observed. This allows us to present the data completely in the chart. The x axis is $\sin(\epsilon)$ and the y axis is $\cos(f)$ where f is the total number of times that the given value occurred in the code.

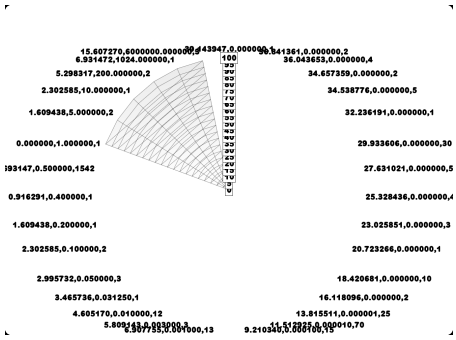

Figure 9: Finally, JavaScript, the official language of the Internet. In this "radar" plot, the data are arranged around a circle, as tuples of $\langle \log(\epsilon), \epsilon, f \rangle$ where f is the frequency of the value being observed. Numbers in boxes ascend along the 0 deg axis, displaying each of the multiples of 5 between 0 and 100, inclusive. Boxes stack so that only part of the number is visible, but you know what's under there if you've looked at numbers before. A spider-web-like network of interlacing lines ascribe some additional meaning to some of the points on the clock face. The smallest nonzero value observed was 10^{-32}. Notes: JavaScript programmers have the highest tolerance for error of all languages tested, with over 1,000 using epsilon of 0.5 or higher. One programmer used $\epsilon = 1024$, and another 6,000,000!