Irish Math. Soc. Bulletin
Number 71, Summer 2013, 59–66
ISSN 0791-5578

TO WHAT DOES THE HARMONIC SERIES CONVERGE?

DAVID MALONE

Abstract. We consider what value the harmonic series will converge to if evaluated in the obvious way using some standard computational arithmetic types.

1. Introduction

In Niall Madden's article about John Todd [5], he mentions a topic we might bring up in an introductory course on analysis: the question of convergence of the harmonic series and what would happen if it was naïvely evaluated on a computer using finite precision. In particular, the obvious way to evaluate

$$s_N = \sum_{n=1}^{N} \frac{1}{n}$$

is to use an algorithm similar to Algorithm 1. If we run this algorithm using integer arithmetic where division commonly rounds down, then the sum converges to 1, as after the first term the reciprocal becomes zero. If we use a different rounding, where 1/2 rounds to 1, then the sum will converge to 2.

Algorithm 1 Simple algorithm for calculating the harmonic series.

$\mathtt{s} \leftarrow 0$, $\mathtt{N} \leftarrow 1$
for ever **do**
  $\mathtt{s} \leftarrow \mathtt{s} + \frac{1}{\mathtt{N}}$, $\mathtt{N} \leftarrow \mathtt{N} + 1$
end for

2010 *Mathematics Subject Classification.* 65G50.
Key words and phrases. roundoff error, series.
Received on 3-5-2013.

2. Floating Point

Of course, no one is (deliberately) likely to try and evaluate this sum using integer arithmetic. However, using floating point arithmetic seems more promising. Students, when learning numerical programming, sometimes assume that the answer given by the computer is exact, and the harmonic series can be a good example to help them understand what can go wrong.

We usually ask them to reason as follows. We know that if `s` and `N` were double precision floating point numbers, the sum will gradually grow while what we add gradually decreases. Also, because the sum and most reciprocals cannot be represented exactly as floating point numbers, the reciprocal and the sum will be rounded. Thus, if we round to the nearest representable number, we see that the sum must eventually become constant. We reach a similar conclusion if we round toward zero or toward $-\infty$.

This is an existence proof for the limit, but it might be of interest to know what value the sum converges to, or at what term the sequence becomes constant. Floating point behaviour is well defined [2, 3], so can we use this to make an estimate?

Let's consider the commonly encountered format of *binary64* double precision floating point numbers. In this format, one bit is used to store the sign of the number, 11 bits are used to store a binary exponent and the remaining 52 bits are used to store bits of the binary expansion of the number. The value of a particular pattern of bits is, usually, interpreted as:

$$(-1)^S \left(1 + \sum_{i=1}^{52} \frac{B_i}{2^i}\right) 2^{E-1023},$$

where S is the sign bit, $B_1, \ldots B_{52}$ are the bits of the binary expansion of the number and E is the value of the exponent bits interpreted as positive binary integer. Note, the leading bit is not stored, because if the number is non-zero then it must always be one. Some numbers, including zero, are stored as special cases, which are indicated by making all the exponent bits zero or one. Other special types of number stored in this way include plus/minus infinity, NaN values (to indicate results of invalid operations) and special small *denormalised* numbers.

For floating point numbers of the format described above, we can consider the change in the value resulting from flipping the least

significant bit, B_{52}. This change is a change in the *unit of least precision*, and a change of this size is often referred to as one ULP. Where the difference between the floating point value and the exact value is important, we will use the typewriter font for `N` and `s`.

3. Estimates

As a crude estimate, since we know a double precision floating point has a significand of 52 bits, and the partial sums $\mathtt{s}_N$ are always bigger than 1, it must converge before term 2^{52+1}, as at this point adding each $1/\mathtt{N}$ will give a result that is small enough so it won't be rounded up to the next number.

We can improve on this estimate, as we know that the sum will be somewhat bigger than one. Denoting by $\lfloor x \rfloor_2$, the largest power of two less than x, the sequence will stop increasing when

$$\frac{1}{\mathtt{N}} < 2^{-52-1} \lfloor \mathtt{s}_N \rfloor_2$$

By using Euler's estimate for the harmonic sum [1], we know $\mathtt{s}_N \approx s_N \approx \log(N) + \gamma$ where

$$\gamma = 0.57721566490153286060651209008240243104215933593992\ldots$$

We can convert this to

$$2^{53} < N(\lfloor \log(N) + \gamma \rfloor_2),$$

which happens when $N > 2^{48}$. We expect a value of $\mathtt{s}_N$ close to

$$\log(2^{48}) + \gamma = 33.8482803317789077\ldots,$$

though there may have been significant accumulation of rounding errors in the sum. We can, however, get a bound on this error; in round-to-nearest mode the error in each operation should be bounded by 0.5 ULP. Concentrating on the errors associated with the sum[1], for each term when the sum is between 1 and 2, we will have an error of at most 2^{-53}. Similarly, for each term when the sum is between 2 and 4, the error will will be at most 2^{-52}, and so on. Thus, if T_r is

$$T_r = \min\left\{n \in \mathbb{N} : \mathtt{s}_n > r\right\},$$

[1]We could also calculate the error associated with evaluating the reciprocals, however this will be considerably less than 2^{-53} for most terms. Consequently, the error terms from the sum dominate.

then we can accumulate these error bounds and bound the error between $\mathtt{s}_n$ and the exact sum by

$$\sum_{n=1}^{\infty} \left(\min(N, T_{2^n}) - \min(N, T_{2^{n-1}})\right) 0.5 \times 2^{-52+n-1}. \tag{1}$$

Here, the first expression in parentheses counts the number of terms before N with the given error. Again, if we are willing to make the approximation $\mathtt{s}_n \approx \log(n) + \gamma$ then we find $T_r \approx e^{r-\gamma}$, and we can easily evaluate this expression numerically and find that we expect the error at term 2^{48} to be less than $0.921256\ldots$.

One might also attempt to consider the rounding errors to be independent random variables. Using $\mathrm{ULP}^2/4$ as a bound on their variance, one might hope to use the central limit theorem to predict a distribution for the error. However, as n becomes large, the bits of $1/n$ and $1/(n+1)$ that are being rounded are increasingly similar, and so independence appears a poor assumption.

4. Numerical Results

Evaluating a sum with 2^{48} terms is feasible on modern computer hardware. Naturally, if parallelisation of the calculation was possible, the sum could be evaluated on a cluster of computers or CPUs. However, the intermediate values are important in establishing what rounding occurs and the final value of the sum.

We evaluated the sum using an AMD Athlon 64 CPU, clocked at 2.6GHz. To avoid the use of extended (80-bit) precision, which is common on platforms based on the i387 floating point processor, the operating system was in 64-bit mode, where i387 instructions are not used. A C program, which explicitly selected the to-nearest rounding mode, was used to find when the sum converged and to what value. As predicted, the sum became constant when $N > 2^{48}$. In this environment, the calculation took a little more than 24 days.

The value of the sum was

$$\mathtt{s}_{2^{48}} = 34.122035668047871581620711367577314376831054687 5,$$

or, the bits of the floating point number were `0x40410f9edd619b06`. The difference between this and Euler's approximation is $0.2737\ldots$, within our estimated error bound of 0.921256.

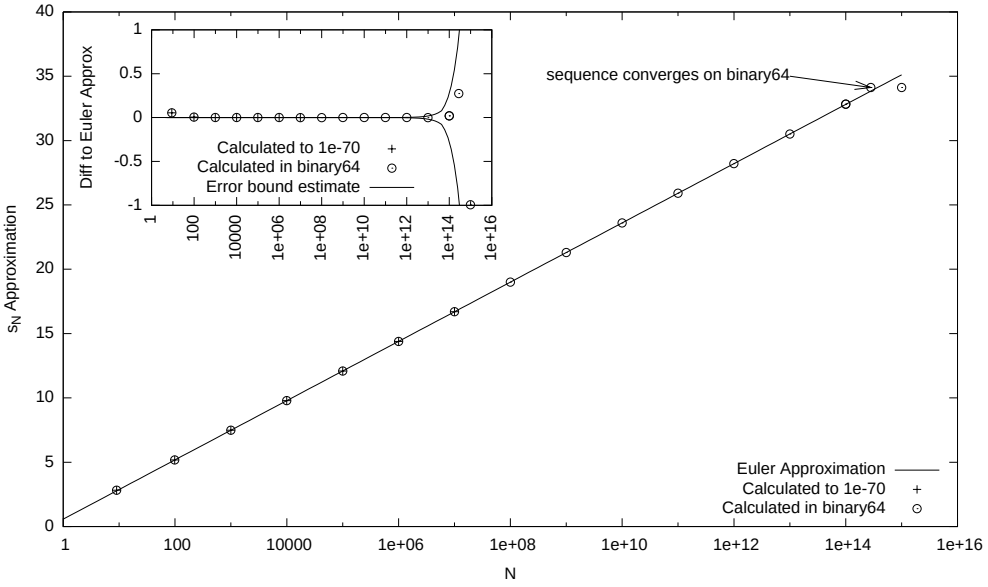

FIGURE 1. Approximations to the partial harmonic sum. The inset graph shows the distance between Euler's estimate and the numerical estimates.

Figure 1 shows Euler's approximation of the partial harmonic sums, $\log(N) + \gamma$, along with the numerical values of $\mathtt{s}_N$. For comparison, we also used an arbitrary precision calculator to estimate the value of s_N when the rounding error at each step is bounded by 10^{-70} for smaller values of N. We see that the values agree well, until we come close to the point where the numerical $\mathtt{s}_N$ converges.

Also shown in Figure 1 is the difference between Euler's approximation and the numerical values. For comparison, the error bound from (1) is plotted. The differences all fall within the error bound estimates. We also note that until $N > 10^{13}$ the accumulated errors are unlikely to be a serious concern. Finally, observe that both numerical values diverge from Euler's approximation when N is small. This is because the error in Euler's formula is known to be $O(1/N)$. If the approximation is improved by including the next term, $\gamma + \log(N) + 1/2N$, this discrepancy is corrected.

5. Rounding Up

We can also ask what happens in other rounding modes. If we round down to the nearest representable floating point number, then the story will be similar to the previous case. In this case, since the

sum is always rounded down, the sum will become constant when

$$\frac{1}{\mathtt{N}} < 2^{-52}\lfloor \mathtt{s}_N \rfloor_2$$

which we can estimate happens when $N > 2^{47}$. Again, we could make an estimate of the range of possible numerical values. Using Euler's formula directly we can get the upper end of the possible range. For the lower end, calculating the error due to rounding, we can use an expression similar to eq. 1, but where the error per term can be 1ULP, rather than 0.5ULP. Checking numerically[2], we find the sum converges to

$$32.813730134256800852199376095086336135864257812 5$$

with bit pattern `0x404068284f1d338a` at term 2^{47}. Figure 2 shows this to be within the range we expect.

If we round up, the situation is a little more interesting. In this case, when

$$\frac{1}{\mathtt{N}} < 2^{-52}\lfloor \mathtt{s}_N \rfloor_2$$

we get an increase of one ULP each time the reciprocal is added. Once this condition is met, at term 2^{47}, the sum begins to act like a counter, beginning at $\mathtt{s}_{2^{47}}$ and incrementing by one ULP for each value of N, beginning at 2^{47}. However, when N reaches 2^{53}, not all integers can be exactly represented as floating point numbers. So, because we are in round-up mode, each increment of `N` actually results in `N` being increased by *two* (which is one ULP for `N`). Thus, after a further 2^{52} steps, we reach 2^{54} when the increments in `N` result in `N` being increased by 4, and so on.

Thus, after 2^{52} loop iterations, `N` and `s` are both incremented by one ULP for each iteration. At iteration 2^{52}, `N` has value 2^{52} and `s` has value of $\mathtt{s}_{2^{47}}$ increased by $2^{52} - 2^{47}$ ULP increments. As $\mathtt{s}_{2^{47}} \ll 2^{47}$, we see that `N` begins at a larger value, and so will remain ahead in this curious race. The largest non-special case value of the exponent is 1023, and will increase by one every 2^{52} iterations, so following another $(1024-52)\times 2^{52}-1$ iterations `N` will reach the largest finite double value of $2^{1024} - 2^{1024-53}$.

On the next iteration `N` will become `inf`, and then $1/$`inf` is zero. At this point, the sum will no longer increase! We conclude that

[2]Some additional care is required here, because if the compiler does constant folding, it will probably be done in the default mode (round to nearest). To avoid such issues, we disabled optimisation when compiling.

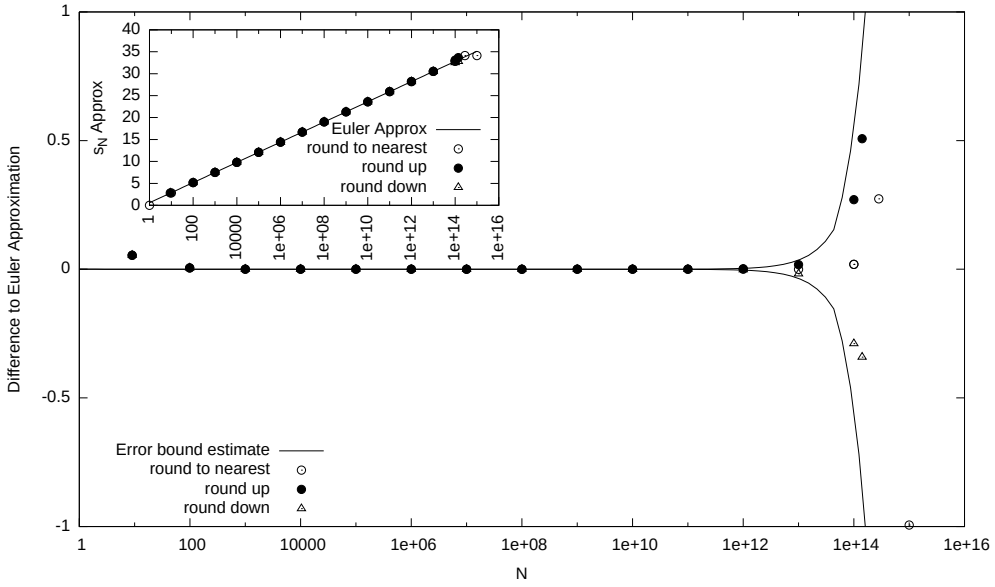

FIGURE 2. Differences between floating point estimates of the partial harmonic sum and Euler's approximation in round-to-nearest, round-up and round-down mode. The inset graph shows the actual values of the estimates.

$\mathtt{s}_N$ converges to a fixed value at this point. This happens after $2^{52} + (1024 - 52) \times 2^{52}$ loop iterations, when $\mathtt{s_N}$ will have increased by $2^{52} - 2^{47} + (1024 - 52) \times 2^{52} = 973 \times 2^{52} - 2^{47}$ ULPs above its value at $\mathtt{s}_{2^{47}}$.

While evaluating 937×2^{52} loop iterations is not practical, we can evaluate $\mathtt{s}_{2^{47}}$ numerically in round-up mode, and we get

$$33.662471610236330832321982597932219505310058593 75,$$

with bit pattern `0x4040d4cbdea63f2b`. If we treat this pattern as a hexadecimal integer, then each ULP increment will increase it by one, giving
`0x4040d4cbdea63f2b` + $937 \times 2^{52} - 2^{47}$ = `0x7d1054cbdea63f2b`.
Interpreting this as a floating point number, we get

$$4596834217901867 \times 2^{926}.$$

Figure 2 shows the values of the sum in round-up, round-down and round-to-nearest mode compared to the value of Euler's approximation, up to the point where they stop increasing or increase by one ULP. The upper error estimate is for the sum in round-up mode and the lower error estimate is the corresponding value for round-down mode. Unsurprisingly, at a particular N values the rounded-up $\mathtt{s}_N$

values are larger than the round-to-nearest values, which are larger than the round-down values.

6. Conclusion

We have looked at the behaviour of the floating point versions of the harmonic series, and shown that we can actually give bounds on how they should behave. Modern PCs are capable of evaluating enough terms of the sequence to establish the values that the numeric sequence converges to in each rounding mode. We leave open the question of what happens if we use Algorithms other than Algorithm 1, for example if we sum the terms in the order suggested by Huffman Coding [4].

References

[1] L. Euler. De progressionibus harmonicis observationes. *Commentarii academiae scientiarum Petropolitanae*, 7:150–161, 1740.
[2] IEEE Working Group for Floating-Point Arithmetic. 754-1985 — IEEE standard for binary floating-point arithmetic. pages 1–20, 1985.
[3] IEEE Working Group for Floating-Point Arithmetic. 754-2008 — IEEE standard for floating-point arithmetic. pages 1–82, 2008.
[4] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40:1098–1101, 1952.
[5] Niall Madden. John Todd and the development of modern numerical analysis. *Bulletin of the Irish Mathematical Society*, (69):11–23, Summer 2012.

David Malone received B.A.(mod), M.Sc. and Ph.D. degrees in mathematics from Trinity College Dublin. He is a Stokes lecturer at Hamilton Institute, NUI Maynooth where he works on applications of mathematics to networks.

Hamilton Institute, National University of Ireland Maynooth
E-mail address: David.Malone@nuim.ie