Algorithm for binary-to-decimal conversion of floating-point numbers

1 Introduction

The goal of this document is to describe an algorithm to convert iEEE-754 64-bit double- and 80-bit extended-precision numbers to decimal representation by using only integer operations and precomputed values, in order to then be printed by `printf` and the other '`printf`-family' functions. The most immediate application of this algorithm would be during the boot-up of a computer (when the libc is not yet loaded), or on basic (embedded) systems that do not have an FPU (Floating Point Unit) available to them, and so must rely solely on integer operations.

The reader is assumed to have a good understanding of the IEEE-754 norm. To recapitulate, a normal, non-zero IEEE-754 binary floating-point number x is represented as $x = (-1)^s \times m \times 2^E$ with $1 \leqslant m < 2$.

1. the IEEE-754 64-bit double format consists of 1 sign bit, 11 binary exponent bits and 52 binary mantissa bits (with an implicit 1 before the mantissa, unless the number is subnormal).

2. the IEEE-754 80-bit extended precision format consists of 1 sign bit, 15 binary exponent bits and 64 binary mantissa bits (there is no implicit 1 before the mantissa).

In decimal floating-point notation, x can be written as $x = (-1)^s \times n \times 10^F$ with $1 \leqslant n < 10$. We will henceforth suppose that $x > 0$, as once we have obtained a decimal representation of x, finding a decimal representation of $-x$ is trivial, as is the case when $x = 0$. The presented algorithm computes an unsigned integer $\tilde{n}$ and a signed integer $\tilde{F}$ such that

$$x = n \times 10^F \approx \tilde{n} \times 10^{\tilde{F}}$$

with a satisfactory degree of accuracy, as we will discuss. From this point, computing the decimal representation as required by floating point printing formats `%e`, `%f` and `%g` in C code is simple. This last step is not the subject matter of this document. Also not that 32-bit single-precision arguments (which we will suppose are represented by the `float` type in C) are promoted to `double` in variadic functions, therefore we need only concern ourselves with the two aforementioned binary floating-point types.

2 Decomposing floating point numbers and bounds on the binary exponent

Whether for the double- or extended-precision type, we can extract the bits that make up their decimal exponent and mantissa, by binding their binary representations to integers via `union` constructs in C. The 80-bit extended-precision format can represent a superset of the values that a double-precision format can represent. Therefore, we need only concern ourselves with being able to convert these values to decimal representation.

We start by considering a **normal** 80-bit floating-point number x (we will get to **subnormal** numbers shortly), from which we have extracted the 64 bits that indicate m and the 15 bits that indicate E.

Supposing that x is neither infinity nor "Not A Number" (NaN), in which case all exponent bits would be set to 1 denoting a value of $2^{15} - 1$, the highest possible value of the binary exponent is $2^{15} - 2$. However, this exponent includes a bias equal to $2^{14} - 1$. As such, the true highest exponent that can appear with the

80-bit format is $2^{15} - 2 - (2^{14} - 1) = 2^{14} - 1$. The lowest (biased) exponent that can appear is 1, which translates to $1 - (2^{14} - 1) = -2^{14} + 2$ once the aforementioned bias is subtracted (a biased exponent of 0 indicates a subnormal number).

For our algorithm, we would first like to compute a mantissa $\tilde{m}$ and a binary exponent $\tilde{E}$ such that :

$$x = \tilde{m} \times 2^{\tilde{E}} \qquad \text{where } m \in [\![2^{63}, 2^{64} - 1]\!]$$

The 64 bits that we extracted to obtain m conceptually have the binary point after the left-most bit given that for normal numbers we have $1 \leqslant m < 2$. This also means that for normal numbers, this leftmost bit is always set to 1.

To obtain $\tilde{m}$ in the range $[\![2^{63}, 2^{64} - 1]\!]$, we can simply take these 64 bits as an integer denoting the value of $\tilde{m}$. Simply put, this means that $\tilde{m} = m \times 2^{63}$ and that we have done away with the conceptual binary point. To maintain the equality $x = \tilde{m} \times 2^{\tilde{E}}$, we have to subtract 63 from E, meaning that $\tilde{E} = E - 63$.

Finally, in the case of subnormal numbers where $0 < m < 1$, we need to adjust things : we again interpret the 64 bits corresponding to m as an integer, but now the leftmost bit is zero (and indeed the following ones may be too), meaning that the extracted integer is strictly less than 2^{63}. We can simply shift this integer left by however many bits necessary to put it in the range $[\![2^{63}, 2^{64} - 1]\!]$. Let L be the magnitude of the shift in bits. The lowest possible value of m in a 80-bit subnormal floating-point number is 2^{-63} meaning that the maximum value of L is 63. Of course, to maintain the equality $x = \tilde{m} \times 2^{\tilde{E}}$ as we did before, we will need to adjust the binary exponent further in the subnormal case, meaning that $\tilde{E} = E - 63 - L$.

Therefore, after converting x to the format $x = \tilde{m} \times 2^{\tilde{E}}$ with $\tilde{m}$ an integer in the range $[\![2^{63}, 2^{64} - 1]\!]$, the most extreme values that $\tilde{E}$ can take are:

- $\tilde{E}_{max} = 2^{14} - 1 - 63 = 16320$
- $\tilde{E}_{min} = -2^{14} + 2 - 63 - 63 = -16508$

Of course the decomposition process will be a different with the 64-bit double format, but as we said, the 80-bit format covers all the possible values of the former, and the object of this section was simply to find the most extreme binary exponent values that could appear while converting x to $x = \tilde{m} \times 2^{\tilde{E}}$ with $\tilde{m} \in [\![2^{63}, 2^{64} - 1]\!]$.

3 Approximating the decimal exponent

The object of the following sections will to obtain a sufficiently precise decimal notation of x. As we stated earlier, if the exact value of the binary floating-point number x in decimal notation is $x = n \times 10^F$ where n is a real number in the interval $[1, 10[$, our aim is to compute an unsigned integer $\tilde{n}$ and a signed integer $\tilde{F}$ such that $x \approx \tilde{n} \times 10^{\tilde{F}}$ to a reasonable degree of approximation. Again, we do not explicitly say here what this degree is, but the relative error will be covered in a later section. We can write $x = n \times 10^F = n^* \times 10^{F^*}$ where $n^* = n \times 10^{18}$ and therefore n^* is a real number in the range $[10^{18}, 10^{19}[$ and $F^* = F - 18$. Our goal will be to compute an integer $\tilde{n}$ that is close to n^*. First of all, however, we must first compute an approximation of F^*. We can write :

$$\begin{aligned} n \times 10^F &= \tilde{m} \cdot 2^{\tilde{E}} \\ 10^F &= \frac{\tilde{m} \cdot 2^{\tilde{E}}}{n} \\ \log_{10} 10^F &= \log_{10} \frac{\tilde{m} \cdot 2^{\tilde{E}}}{n} \\ F &= \log_{10} \left(\tilde{m} \cdot 2^{\tilde{E}} \right) - \log_{10} n \end{aligned}$$

However, we know that $1 \leqslant n < 10$ (indeed, we defined it in this way) and so $0 \leqslant \log_{10} n < 1$.
Furthermore, since F is an integer, it is apparent that :

$$\log_{10} \tilde{m} \cdot 2^{\tilde{E}} - \log_{10} n = \left\lfloor \log_{10} \tilde{m} \cdot 2^{\tilde{E}} \right\rfloor$$

We therefore have :

$$F^* = F - 18 = \left\lfloor \log_{10} \tilde{m} \cdot 2^{\tilde{E}} \right\rfloor - 18$$

We will now show how to compute F^*, albeit with a margin of error of 1. The current expression is not suitable to be computed using only integer operations, so we develop it further :

$$\begin{aligned}
F^* &= \left\lfloor \log_{10} \tilde{m} \cdot 2^{\tilde{E}} \right\rfloor - 18 \\
&= \left\lfloor \tilde{E} \log_{10} 2 + \log_{10} \tilde{m} \right\rfloor - 18 \\
&= \left\lfloor \tilde{E} \log_{10} 2 + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor + \lfloor \log_{10} 2^{63} \rfloor \right\rfloor - 18 \\
&= \left\lfloor \tilde{E} \log_{10} 2 + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor \right\rfloor + \lfloor \log_{10} 2^{63} \rfloor - 18
\end{aligned}$$

We have $\log_{10} 2^{63} = 18$, therefore :

$$\begin{aligned}
F^* &= \left\lfloor \tilde{E} \log_{10} 2 + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor \right\rfloor \\
&= \left\lfloor \tilde{E} \log_{10} 2 + \left\lfloor \tilde{E} \log_{10} 2 \right\rfloor - \left\lfloor \tilde{E} \log_{10} 2 \right\rfloor + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor \right\rfloor \\
&= \left\lfloor \tilde{E} \log_{10} 2 \right\rfloor + \left\lfloor \tilde{E} \log_{10} 2 - \left\lfloor \tilde{E} \log_{10} 2 \right\rfloor + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor \right\rfloor \\
&= \left\lfloor \tilde{E} \log_{10} 2 \right\rfloor + \lfloor \alpha \rfloor
\end{aligned}$$

In the above expression, we have $\alpha = \tilde{E} \log_{10} 2 - \left\lfloor \tilde{E} \log_{10} 2 \right\rfloor + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor$, and we need to establish bounds on the value of $\lfloor \alpha \rfloor$:

$$\begin{aligned}
\alpha &= \tilde{E} \log_{10} 2 - \left\lfloor \tilde{E} \log_{10} 2 \right\rfloor + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor \\
&= \tilde{E} \log_{10} 2 - (\tilde{E} \log_{10} 2 - \delta) + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor \qquad \text{where } 0 \leqslant \delta < 1 \\
&= \delta + \log_{10} \tilde{m} - \lfloor \log_{10} 2^{63} \rfloor \\
&= \delta + \log_{10} \left(\frac{\tilde{m}}{2^{63}} \right) + \log_{10} 2^{63} - \lfloor \log_{10} 2^{63} \rfloor
\end{aligned}$$

We can obseve that :

- $\log_{10} 2^{63} - \lfloor \log_{10} 2^{63} \rfloor \approx 0.96$
- $0 \leqslant \log_{10} \left(\frac{\tilde{m}}{2^{63}} \right) < \log_{10} 2 \approx 0.3 \qquad$ because $2^{63} \leqslant \tilde{m} < 2^{64}$, therefore $1 \leqslant \frac{\tilde{m}}{2^{63}} < 2$

Therefore, we have $0 \leqslant \alpha < 3$ and $\lfloor \alpha \rfloor \in \{0, 1, 2\}$. To have a good approximation (at most within 1) of F^*, we take $\hat{F} = \left\lfloor \tilde{E} \log_{10} 2 \right\rfloor + 1$. However, this expression does not address our need to avoid floating point computations. We solve this by proving that $\forall \tilde{E} \in [\![\tilde{E}_{min}, \tilde{E}_{max}]\!]$,

$$\left\lfloor \tilde{E} \log_{10} 2 \right\rfloor + 1 = \left\lfloor \tilde{E} \cdot \lfloor 2^{40} \log_{10} 2 \rfloor \cdot 2^{-40} \right\rfloor + 1 \tag{1}$$

The latter expression can easily be precomputed using integer operations (in particular via multiplication, bit shifts and one addition) and the precomputed value $\lfloor 2^{40} \log_{10} 2 \rfloor$. We prove this equality by checking each case in the range $[\![\tilde{E}_{min}, \tilde{E}_{max}]\!]$ using the Sollya software tool (available here). The case by case-by-case check is done in the proof script `proof_for_E.sollya`.

4 Approximating the decimal mantissa

Now that we have an approximation $\hat{F}$ of F^* in $x = n^* \times 10^{F^*}$ (which is either exact, or off by ± 1), we will turn our attention to computing the corresponding decimal mantissa.
First of all, recall that $x = \tilde{m} \times 2^{\tilde{E}}$. We can write :

$$\begin{aligned} \tilde{m} \times 2^{\tilde{E}} &= n^* \times 10^{F^*} \\ n^* \times 10^{F^*} &= \hat{n} \times 10^{\hat{F}} \\ \hat{n} &= \tilde{m} \times 2^{\tilde{E}} \times 10^{-\hat{F}} \\ \hat{n} &= \tilde{m} \times 2^{\tilde{E}} \times 2^{-\hat{F}} \times 5^{-\hat{F}} \\ \hat{n} &= \tilde{m} \times 2^{\tilde{E}-\hat{F}} \times 5^{-\hat{F}} \end{aligned} \tag{2}$$

We can then define the following variables where $\lfloor z \rceil$ indicates rounding z to the nearest integer (we round down when z is equidistant from $\lfloor z \rfloor$ and $\lceil z \rceil$) :

$$t = 5^{-\hat{F}} \cdot 2^{126-\left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor} = 5^{-\hat{F}} \cdot 2^{126-\log_2(5^{-\hat{F}})+\delta} \qquad \text{where } 0 \leqslant \delta < 1$$

$$\hat{t} = \lfloor t \rceil = t + \delta_t \qquad \text{where } -\frac{1}{2} \leqslant \delta_t < \frac{1}{2}$$

From the above, we can see that $2^{126} \leqslant t < 2^{127}$ and therefore $2^{126} \leqslant \hat{t} \leqslant 2^{127}$, meaning that $\hat{t}$ can be represented as a 128-bit unsigned integer (t and $\hat{t}$ are of course functions of $\hat{F}$). We now write :

$$\begin{aligned} \hat{n} &= 2^{\tilde{E}-\hat{F}-126+\left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor} \cdot \tilde{m} \cdot t \quad \text{(equal to } \tilde{m} \times 2^{\tilde{E}-\hat{F}} \times 5^{-\hat{F}}\text{)} \\ \hat{\hat{n}} &= 2^{\tilde{E}-\hat{F}-126+\left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor} \cdot \tilde{m} \cdot \hat{t} \\ \hat{F}_{min} &= \left\lfloor \tilde{E}_{min} \log_{10} 2 \right\rfloor + 1 = -4950 \\ \hat{F}_{max} &= \left\lfloor \tilde{E}_{max} \log_{10} 2 \right\rfloor + 1 = 4913 \\ T(\hat{F}) &= 5^{-\hat{F}} \cdot 2^{126-\log_2(5^{-\hat{F}})+\delta} = \hat{t} \end{aligned}$$

For all values $\hat{F} \in [\![\hat{F}_{min}, \hat{F}_{max}]\!]$, we can precompute $T(\hat{F})$ $(= \hat{t})$. We also define $\Delta = \left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor$. In order to compute $\hat{\hat{n}}$, we still need $2^{\tilde{E}-\hat{F}-126+\Delta}$. In particular, Δ does not seem to be easily computable using only integer operations. However, $\forall \hat{F} \in [\![\hat{F}_{min}, \hat{F}_{max}]\!]$, we have :

$$\begin{aligned} \Delta &= \left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor \\ &= \left\lfloor (-\hat{F}) \cdot \log_2 5 \right\rfloor \\ &= \left\lfloor (-\hat{F}) \cdot \lfloor 2^{40} \cdot \log_2 5 \rfloor \cdot 2^{-40} \right\rfloor \end{aligned}$$

The proof for the final step is based on a case-by-case check for the values of $\hat{F}$, using the Sollya software tool and the script `proof_for_delta.sollya`. The last expression for Δ can be computed using only integer arithmetic and the precomputed value $\lfloor 2^{40} \cdot \log_2 5 \rfloor$. We also define :

$$\sigma = \tilde{E} - \hat{F} - 126 + \Delta \in [\![-130, -127]\!]$$

The proof for the bounds $[\![-130, -127]\!]$ on σ is given (again, via case-by-case check for the values of $\hat{F}$) by the script `proof_for_sigma.sollya`. Finally, we define :

$$\hat{\hat{n}} = \left\lfloor \hat{\hat{n}} \right\rfloor = \lfloor 2^{\sigma} \cdot \tilde{m} \cdot \hat{t} \rceil$$

We know that $\forall r \in \mathbb{R}, \lceil r \rceil = \lfloor r + 0.5 \rfloor$. Therefore, $\forall k \in \mathbb{Z}$, we have :

$$\left\lceil 2^k \cdot r \right\rceil = \left\lfloor 2^k \cdot r + 0.5 \right\rfloor = \left\lfloor (r + 2^{-k-1}) \cdot 2^k \right\rfloor$$

Therefore we have :

$$\hat{\tilde{n}} = \left\lceil 2^\sigma \cdot \tilde{m} \cdot \hat{t} \right\rceil = \left\lfloor 2^\sigma \cdot \tilde{m} \cdot \hat{t} + 0.5 \right\rfloor = \left\lfloor (\tilde{m} \cdot \hat{t} + 2^{-\sigma-1}) \cdot 2^\sigma \right\rfloor$$

Using the rightmost expression, we can compute $\hat{\tilde{n}}$ using only integer arithmetic.
Hence, if we take $\tilde{n} = \hat{\tilde{n}}$ and $\tilde{F} = \hat{F}$, we have reached our goal and computed an approximation $x \approx \tilde{n} \times 10^{\tilde{F}}$ using only integer arithmetic (and precomputed values for $\hat{t}$).

5 Error analysis

Recall from (2) that $x = \hat{n} \times 10^{\hat{F}}$. We have two layers of approximation relative to $\hat{n}$:

$$\hat{\tilde{n}} = 2^\sigma \cdot \tilde{m} \cdot \hat{t} = \hat{n} \cdot \frac{\hat{t}}{t}$$
$$\hat{\tilde{n}} = \left\lfloor 2^\sigma \cdot \tilde{m} \cdot \hat{t} \right\rceil = \left\lfloor \hat{\tilde{n}} \right\rceil$$

As we said before,

$$\begin{aligned} \hat{t} &= t + \delta_t && \text{where } -\frac{1}{2} \leqslant \delta_t < \frac{1}{2} \\ &= t \cdot \left(1 + \frac{\delta_t}{t}\right) \\ &= t \cdot (1 + \varepsilon_t) && \text{where } \varepsilon_t = \frac{\delta_t}{t} \end{aligned}$$

Given that $2^{126} \leqslant t < 2^{127}$, we know that :

$$\begin{aligned} |\varepsilon_t| &\leqslant \frac{1/2}{2^{126}} \\ &\leqslant 2^{-127} \end{aligned}$$

Secondly, we have :

$$\begin{aligned} \hat{\tilde{n}} &= \left\lfloor \hat{\tilde{n}} \right\rceil \\ &= \hat{\tilde{n}} + \delta_{\hat{\tilde{n}}} && \text{where } -\frac{1}{2} \leqslant \delta_{\hat{\tilde{n}}} < \frac{1}{2} \\ &= \hat{\tilde{n}} \cdot \left(1 + \frac{\delta_{\hat{\tilde{n}}}}{\hat{\tilde{n}}}\right) \\ &= \hat{\tilde{n}} \cdot \left(1 + \varepsilon_{\hat{\tilde{n}}}\right) && \text{where } \varepsilon_{\hat{\tilde{n}}} = \frac{\delta_{\hat{\tilde{n}}}}{\hat{\tilde{n}}} \end{aligned}$$

We know that $\hat{\tilde{n}} = 2^\sigma \cdot \tilde{m} \cdot \hat{t}$ and that :

$$\begin{aligned} 2^{-130} &\leqslant 2^\sigma \leqslant 2^{-127} \\ 2^{63} &\leqslant \tilde{m} < 2^{64} \\ 2^{126} &\leqslant \hat{t} \leqslant 2^{127} \end{aligned}$$

Therefore $2^{59} \leqslant \hat{\hat{n}} < 2^{64}$, and $|\varepsilon_{\hat{\hat{n}}}| \leqslant 2^{-60}$. Finally, we can write :

$$\begin{aligned}\hat{\hat{\hat{n}}} &= \hat{n} \cdot (1 + \varepsilon_{\hat{\hat{n}}})(1 + \varepsilon_t) \\ &= \hat{n} \cdot (1 + \varepsilon_{\hat{\hat{n}}} + \varepsilon_t + \varepsilon_{\hat{\hat{n}}}\varepsilon_t) \\ &= \hat{n} \cdot (1 + \mathcal{E})\end{aligned}$$

Thus, we know that $|\mathcal{E}| \leqslant 2^{-60} + 2^{-127} + 2^{-187} \approx 2^{-60} \approx 10^{-18.06}$. In other words, the relative error is roughly on the order of the 18-th digit after the decimal point.

6 Bipartite tables for precomputed values

The aforementioned algorithm requires us to precompute the 128-bit integer $\hat{t}$ for all 9864 possible values of $\hat{F}$ (recall that $\hat{F}_{min} = -4950$ and $\hat{F}_{max} = 4913$). This alone requires $9864 \times 16 = 157824$ bytes, or ≈ 154 KB of memory. Given that a potential application of this algorithm is for embedded systems, we would like to reduce this requirement.
First, we take the Euclidean division of $\hat{F}$ by 256, meaning that :

$$\hat{F} = 256 \cdot \hat{F}_h + \hat{F}_l \qquad \text{where } 0 \leqslant \hat{F}_l < 256$$

Given that $\hat{F} \in [\![\hat{F}_{min}, \hat{F}_{max}]\!] = [\![-4950, 4913]\!]$, $\hat{F}_h$ is in the range $[\![-20, 19]\!]$.

We now re-write the formula for t :

$$\begin{aligned}t &= 5^{-\hat{F}} \cdot 2^{126-\left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor} \\ &= 5^{-(2^8 \cdot \hat{F}_h + \hat{F}_l)} \cdot 2^{126-\left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor} \\ &= 5^{-(2^8 \cdot \hat{F}_h + \hat{F}_l)} \cdot 2^{126-\left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor - \left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor - \left\lfloor \log_2(5^{-\hat{F}_l}) \right\rfloor + \left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor + \left\lfloor \log_2(5^{-\hat{F}_l}) \right\rfloor} \\ &= \left(5^{-2^8 \cdot \hat{F}} \cdot 2^{126-\left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor}\right) \left(5^{-\hat{F}_l} \cdot 2^{126-\left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor}\right) 2^{2-128-\left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor + \left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor + \left\lfloor \log_2(5^{-\hat{F}_l}) \right\rfloor} \\ &= t_1 \cdot t_2 \cdot 2^{\tau - 128} \qquad \text{where } \tau = 2 - \left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor + \left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor + \left\lfloor \log_2(5^{-\hat{F}_l}) \right\rfloor\end{aligned}$$

We also define 2 functions :

$$\begin{aligned}T_1(\hat{F}_h) &= \left\lfloor 5^{-2^8 \cdot \hat{F}} \cdot 2^{126-\left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor} \right\rceil = \lfloor t_1 \rceil \\ T_2(\hat{F}_l) &= \left\lfloor 5^{-\hat{F}_l} \cdot 2^{126-\left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor} \right\rceil = \lfloor t_2 \rceil\end{aligned}$$

We can precompute $\hat{t}_1 = \lfloor t_1 \rceil$ and $\hat{t}_2 = \lfloor t_2 \rceil$ for all possible values of $\hat{F}_h$ and $\hat{F}_l$ respectively. It is easy to prove (as we did with t in the previous algorithm), that $2^{126} \leqslant t_1, t_2 < 2^{127}$, and so that $2^{126} \leqslant \hat{t}_1, \hat{t}_2 \leqslant 2^{127}$, which can be represented as 128-bit unsigned integers. This therefore amounts to a new total of 296 precomputed 128-bit values, as $\hat{F}_h \in [\![-20, 19]\!]$ and $\hat{F}_l \in [\![0, 255]\!]$. This requires $296 \times 16 = 4736$ bytes, or roughly 4.6 KB, down from ≈ 154 KB for the previous algorithm, a 33-fold difference.

As for computing τ, notice that we can compute $\left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor$ and $\left\lfloor \log_2(5^{-\hat{F}_l}) \right\rfloor$ in the same way that we computed $\left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor$ earlier on : recall that we proved via our `proof_for_delta.sollya` script that $\forall \hat{F} \in [\![\hat{F}_{min}, \hat{F}_{max}]\!] = [\![-4950, 4913]\!]$,

$$\Delta = \left\lfloor \log_2(5^{-\hat{F}}) \right\rfloor = \left\lfloor (-\hat{F}) \cdot \lfloor 2^{40} \cdot \log_2 5 \rfloor \cdot 2^{-40} \right\rfloor$$

We can use the same scheme to compute $\left\lfloor \log_2(5^{-2^8 \cdot \hat{F}_h}) \right\rfloor$ and $\left\lfloor \log_2(5^{-\hat{F}_l}) \right\rfloor$ using only integer arithmetic and the precomputed value $\lfloor 2^{40} \cdot \log_2 5 \rfloor$, by substituting $2^8 \cdot \hat{F}_h$ and $\hat{F}_l$ for $\hat{F}$ in the above formula.

The only hindrance is that $-2^8 \cdot \hat{F}_h$ can reach a minimum of $-20 \cdot 2^8 = -5120$. Hence we extend the minimum bound of our script's checking to -5120 and verify that this property still holds. We now define :

$$\Delta_h = \left\lfloor (-2^8 \cdot \hat{F}_h) \cdot \left\lfloor 2^{40} \cdot \log_2 5 \right\rfloor \cdot 2^{-40} \right\rfloor$$
$$\Delta_l = \left\lfloor (-\hat{F}_l) \cdot \left\lfloor 2^{40} \cdot \log_2 5 \right\rfloor \cdot 2^{-40} \right\rfloor$$

After computing $\tau = 2 - \Delta + \Delta_h + \Delta_l$ at runtime and looking up $\hat{t}_1$ and $\hat{t}_2$ in our precomputed table, we can compute :

$$\hat{t}^{'} = \left\lfloor \hat{t}_1 \cdot \hat{t}_2 \cdot 2^{\tau - 128} \right\rfloor$$

Once again, we can compute this at runtime using only integer arithmetic (multiplications and bit-shifts), given that $\hat{t}_1$, $\hat{t}_2$ and τ are all integers. In `proof_for_T.sollya`, we prove that $2^{126} \leqslant \hat{t}^{'} \leqslant 2^{127}$. Finally, we define :

$$\begin{aligned} \hat{\hat{n}}^{'} &= 2^{\sigma} \cdot \tilde{m} \cdot \hat{t}^{'} \\ \hat{\hat{n}}^{'} &= \lfloor \hat{\hat{n}}^{'} \rceil \\ &= \left\lfloor (\tilde{m} \cdot \hat{t}^{'} + 2^{-\sigma-1}) \cdot 2^{\sigma} \right\rfloor \end{aligned}$$

This can also be computed at runtime using integer arithmetic, as σ, $\tilde{m}$ and $\hat{t}^{'}$ are all integers. We can prove, given the bounds that we have on these, that $2^{59} \leqslant \hat{\hat{n}}^{'} \leqslant 2^{64} - 1$. If we take $\tilde{n} = \hat{\hat{n}}^{'}$, our final approximation of x is now $\hat{\hat{n}}^{'} \times 10^{\hat{F}}$.

7 Error analysis for bipartite table algorithm

We have four layers of approximation for this second algorithm :

1. $\hat{t}_1 = \lfloor t_1 \rceil = t_1 + \delta_1 \qquad$ where $-\frac{1}{2} \leqslant \delta_1 < \frac{1}{2}$
2. $\hat{t}_2 = \lfloor t_2 \rceil = t_2 + \delta_2 \qquad$ where $-\frac{1}{2} \leqslant \delta_2 < \frac{1}{2}$
3. $\hat{t}^{'} = \left\lfloor \hat{t}_1 \cdot \hat{t}_2 \cdot 2^{\tau-128} \right\rfloor = \hat{t}_1 \cdot \hat{t}_2 \cdot 2^{\tau-128} + \delta_3 \qquad$ where $0 \leqslant \delta_3 < 1$
4. $\hat{\hat{n}}^{'} = \left\lfloor \hat{\hat{n}}^{'} \right\rceil = \hat{\hat{n}}^{'} + \delta_4 \qquad$ where $\frac{1}{2} \leqslant \delta_4 < \frac{1}{2}$

We know that :

1. $t_1 \geqslant 2^{126}$ therefore $\hat{t}_1 = t_1 \left(1 + \frac{\delta_1}{t_1}\right) = t_1(1 + \varepsilon_1)$ where $|\varepsilon_1| \leqslant 2^{-127}$
2. $t_2 \geqslant 2^{126}$ therefore $\hat{t}_2 = t_2 \left(1 + \frac{\delta_2}{t_2}\right) = t_2(1 + \varepsilon_2)$ where $|\varepsilon_2| \leqslant 2^{-127}$
3. $\mathcal{T} = \hat{t}_1 \cdot \hat{t}_2 \cdot 2^{\tau-128} \geqslant 2^{126}$ therefore $\hat{t}^{'} = \mathcal{T}\left(1 + \frac{\delta_3}{\mathcal{T}}\right) = \mathcal{T}(1 + \varepsilon_3)$ where $|\varepsilon_3| \leqslant 2^{-126}$
4. $\hat{\hat{n}}^{'} \geqslant 2^{59}$ therefore $\hat{\hat{n}}^{'} = \hat{\hat{n}}\left(1 + \frac{\delta_4}{\hat{\hat{n}}^{'}}\right) = \hat{\hat{n}}^{'}\,(1 + \varepsilon_4)$ where $|\varepsilon_4| \leqslant 2^{-60}$

Recall once more that

$$x = \hat{n} \times 10^{\hat{F}}$$
$$x \approx \hat{\hat{n}} \times 10^{\hat{F}} \qquad \text{which is our final approximation.}$$

The relative error of our final approximation is therefore :

$$\begin{aligned}
\hat{\hat{n}}^{'} &= \left\lfloor 2^{\sigma} \cdot \tilde{m} \cdot \hat{t}^{'} \right\rceil \\
&= 2^{\sigma} \cdot \tilde{m} \cdot \hat{t}^{'} \cdot (1+\varepsilon_4) \\
&= 2^{\sigma} \cdot \tilde{m} \cdot \mathcal{T} \cdot (1+\varepsilon_3)(1+\varepsilon_4) \\
&= 2^{\sigma} \cdot \tilde{m} \cdot \hat{t}_1 \cdot \hat{t}_2 \cdot 2^{\tau-128} \cdot (1+\varepsilon_3)(1+\varepsilon_4) \\
&= 2^{\sigma} \cdot \tilde{m} \cdot t_1 \cdot t_2 \cdot 2^{\tau-128} \cdot (1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3)(1+\varepsilon_4) \\
&= 2^{\sigma} \cdot \tilde{m} \cdot t \cdot (1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3)(1+\varepsilon_4) \\
&= \hat{n} \cdot (1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3)(1+\varepsilon_4) \\
&= \hat{n} \cdot (1+\mathcal{E}^{'})
\end{aligned}$$

Given that ε_1, ε_2 and ε_3 are negligible before ε_4, we have $\mathcal{E} \approx \varepsilon_4$ and $\varepsilon \leqslant 2^{-60}$. As with the previous algorithm, given that $2^{-60} \approx 10^{-18.06}$, we have a relative error bounded by roughly 10^{-18}, which is on the order of the 18th digit after the decimal point.