

In the IEEE standard (which most chip manufacturers now follow), a (double precision) “normal” number x is represented using 64 bits (binary digits, *i.e.* 0 or 1) as follows:

$$x = (-1)^s (1 + j * \text{eps}) 2^{a-1023}$$

where $s = 0$ or 1 is a sign bit, j is a 52-bit *mantissa*: $j \equiv d_{51}d_{50} \dots d_1d_0$ with $d_k = 0$ or 1 , such that $j = d_{51}2^{51} + d_{50}2^{50} + \dots + d_12^1 + d_02^0$, and $\text{eps} = 2^{-52} = 2.220446049250313 \cdot 10^{-16}$. Finally, a is an 11-bit exponent $a \equiv d_{10} \dots d_0$ such that $a = d_{10}2^{10} + \dots + d_02^0$, however $a = 0$ and $a = 2^{11} - 1$ are reserved for special *definitions* (**zero** when $a = 0$ and $j = 0$, “denormalized numbers” when $a = 0$, $j \neq 0$, **NaN** “Not-a-Number” when $a = 2047$ and $j \neq 0$ and **Inf** “infinity” when $a = 2047$ and $j = 0$), so the range of a is $1 \leq a \leq 2^{11} - 2 = 2046$. These computer numbers are called *double precision floating point numbers*. There are also *integer numbers* defined in a similar way but without eps .

With this “normal” representation there are exactly $2^{52} - 1$ equidistant numbers between two consecutive powers of two because the largest j is $j = 2^{51} + 2^{50} + \dots + 2^1 + 1 = 2^{52} - 1$. So the computer knows of the numbers $1 + \text{eps}$, $1 + 2\text{eps}$, $1 + 3\text{eps}$, $\dots$, $1 + (2^{52} - 1)\text{eps}$ between 1 and 2, but it doesn’t know about any other numbers between 1 and 2 besides those! Likewise the computer knows only of $2^{52} - 1$ equidistant numbers between 4 and 8, those numbers are $4(1 + j * \text{eps})$ so their spacing is now 4 eps not eps . The distance from 4 to the next largest number is 4 eps . This is why eps is called *relative accuracy*. Note that the largest number in the IEEE format is $(1 + j_{\max}\text{eps}) 2^{a_{\max}-1023} = (2 - \text{eps}) 2^{1023} = 1.797693134862316e + 308$ (see `help realmax` in Matlab). The smallest number that you can represent in this “normal” way is $2^{1-1023} = 2^{-1022} = 2.225073858507201 \cdot 10^{-308}$ (see `help realmin` in Matlab).

There are other subtleties in the IEEE format. Note for instance that there is both $+0$ and -0 and they are defined to equal each other in arithmetic operations. There is also $+\text{Inf}$ and $-\text{Inf}$ for $\pm\infty$. The IEEE format also provides for “denormalized numbers” which are numbers smaller than those representable by the “normal” representation. These numbers are represented as

$$x = (-1)^s j \times 2^{-1074}$$

with $j \neq 0$ and $a = 0$ as mentioned above. The gap between two consecutive denormalized numbers is 2^{-1074} and this is chosen to equal the gap between the “normal” numbers (as defined earlier) in the smallest normal range $[2^{-1022} \ 2^{-1021}]$ which is $\text{eps} 2^{-1022} = 2^{-1074}$. The point of these denormalized numbers is to make sure that if $x - y$ is 0 then the floating point number x is indeed exactly equal to the floating point number y . For instance, $(1 + \text{eps}/2) - 1 = 0$ because $(1 + \text{eps}/2)$ does not exist on the computer so $1 + \text{eps}/2$ is truly “equal” to 1 on the computer, but $(1 + \text{eps}) - 1 = \text{eps} \neq 0$ because $1 + \text{eps}$ is indeed different from 1 on the computer. However if we calculate $(1 + \text{eps}) 2^{-1022} - 2^{-1022}$ (*i.e.* the difference between the two smallest “normal” numbers) we get a result that is smaller than the smallest normal number (2^{-1022}), so if we only had normal numbers we would conclude that this difference is zero even though the two numbers are different when represented with 64 bits. The denormalized numbers save the day because that difference is equal to 2^{-1074} , which is the smallest denormalized number, instead of 0. It can be verified also that the difference between two distinct denormalized numbers cannot be zero.