Accuracy and Speed

1) Floating point representation

In Python: IEEE 754 double precision representation:

\[ X = (-1)^{\text{sign}} \left( 1. b_{51} b_{50} \ldots b_0 \right) \times 2^{e-1023} \]

Equivalently:

\[ X = (-1)^{\text{sign}} \left( 1 + \sum_{i=1}^{52} b_{52-i} \times 2^{-i} \right) \times 2^{e-1023} \]

Where:

a) \( \text{sign} \) is either 0 or 1 (1 bit)

b) Fraction is the fractional part of a binary number (written in the form 1. fraction) and it is represented with 52 bits; the "1." in front is not explicitly represented to save space.

c) The exponent [e] is an 11-bit unsigned integer. The exponent is written in binary representation and can therefore take all integer values between 0 and 2047. Note that the actual final exponent is:

\[ [e - \text{bias}] \rightarrow \text{actual exponent} \]

where, in double precision, we set \( \text{bias} = 1023 \). Therefore we can represent, in principle, numbers that range from \( \rightarrow \)
\[ \text{from } 2^{-1023} \rightarrow 2^{+1024}. \text{ The actual range is instead from } 2^{-1022} \rightarrow 2^{+1023}. \text{ The reason is that the exponent with all zeros is used to represent "exact 0" (it would be } 2^{-1023} \text{ otherwise), whereas the exponent with all ones (that would be } 2^{1024} \text{) is used to define "infinity."} \]

**Example.** Consider the number \( x = 85.125 \) and write it in IEEE 754, 64-bit representation:

\[
\text{a) Binary: } 85 = 1010101 \\
\quad 0.125 = 0.001 \\
\quad \Rightarrow 85.125 = 1010101.001
\]

\[
\text{b) Need to write as I. fraction } \Rightarrow \quad 1010101.001 = 1.\overline{0101001} \times 2 \quad \text{unbiased exponent}
\]

\[
\quad \Rightarrow 1010101.001 = 1.\overline{0101001} \times 2
\]

\[
\text{fraction}
\]

\[
\text{c) To get the biased exponent just remember } 6 = e^{-1023} \quad \text{then } e = 1029. \text{ Need to write } e \text{ in base 2}
\]

\[
\quad \text{[
\begin{array}{c}
e = 1029 = 10000000101
\end{array}
\]}

\[
\text{d) The number is positive, hence } \text{Sign} = 0
\]

Therefore, we get the double precision representation of 85.125 as:

\[ 85.125 \rightarrow 0 \quad 10000000101 \quad 010101001 \quad \overline{10} \quad \ldots \ldots \quad 0 \quad 43 \quad \text{zeros} \]

**The two main things we learn are:**

\[
\text{a) The double precision representation allows to store numbers between -10^{-308} \text{ and } 10^{308}
\]

\[
\text{b) The number is rounded off with a precision of } 2^{-53}, \quad \text{corresponding to } \approx 16 \text{ decimal digits}
\]