Question 1: What is machine epsilon?

Answer: Machine epsilon, $\in_{mach}$ is defined as the smallest number such that

$$1+ \in_{mach} > 1$$

It is the difference between 1 and the next nearest number representable as a machine number.

Let's look at this concept for a hypothetical word of 8 bits that stores a real number where the first bit is used for the sign of the number, the next three bits for the biased exponent, and the last four bits for the magnitude of the mantissa. The number $(1)_{10}$ is represented as

$(1)_{10} =$

0	0	1	1	0	0	0	0

The next number in the binary word after $(1)_{10} =$ would be

0	0	1	1	0	0	0	1

and in base-10, that number is $(1.0625)_{10}$.

This gives

$$\in_{mach} = 1.0625 - 1$$
$$= 0.0625$$

Question 2: How do I find the machine epsilon using a MATLAB code?

Answer: The MATLAB code to find the machine epsilon is given below. As an initial guess of the machine epsilon, we choose a positive number which will definitely give 1+number>1. So we choose a number 1. Then we keep halving the number till we find that the sum of 1 and the number does not get recognized by the computer as greater than one.

```
epsmach = 1;
while (1+epsmach) > 1
   epsmach = epsmach/2;
end
epsmach= epsmach*2
```

Question a: Why do we multiply by 2 in the last line of the MATLAB code?

Question b: Why do we get the machine epsilon to be 2.2204e-16 in MATLAB 2013b, when we know that the machine epsilon for a single precision machine is 0.11921e-6.

Question 3: How is machine epsilon related to the number of bits used to represent a floating point number?

Answer: To represent a real number, if there are m bits used for the magnitude of the mantissa, then the machine epsilon is 2^{-m}.

For a hypothetical word of 8 bits that stores a real number where the first bit is used for the sign of the number, the next three bits for the biased exponent, and the last four bits for the magnitude of the mantissa, $(1)_{10}$ is represented as

$(1)_{10} =$

0	0	1	1	0	0	0	0

The next number in the hypothetical word would be

0	0	1	1	0	0	0	1

and in base-10, that number is $(1.0625)_{10}$.

This gives

$$\begin{aligned} \in_{mach} &= 1.0625 - 1 \\ &= 0.0625 \\ &= 2^{-4} \end{aligned}$$

Question 4: What is the significance of machine epsilon for a student in an introductory course in numerical methods?

Answer: The machine epsilon is an upper bound on the absolute relative true error in representing a number. Let y be the machine number representation of a number x, then we can show that the absolute relative true error in the representation is

$$\left|\frac{x-y}{x}\right| \leq \in_{mach}$$

For example, let a real number be represented in a hypothetical word of 8 bits where the first bit is used for the sign of the number, the next three bits for the biased exponent, and the last four bits for the magnitude of the mantissa. A base-10 number of 6.7 would be represented approximately as

$(6.7)_{10} \approx$

0	1	0	1	1	0	1	0

The above floating point representation is an exact representation of a base-10 number of 6.5. Hence, the absolute relative true error is

$$\left|\in_t\right| = \left|\frac{6.7-6.5}{6.7}\right|$$
$$= 0.02985$$
$$< \in_{mach} = 0.0625$$

Go ahead and try any other number between the smallest (-31) and largest number (31) that can be represented by the above hypothetical 8-bit word, and convince yourself that the absolute relative true error in the representation of a chosen number is always less than the machine epsilon.

Question 5: What is the proof that the absolute relative true error in representing a number on a machine is always less than the machine epsilon?

Proof: If x is a positive number we want to represent, it will be between a machine number x_b below x and a machine number x_u above x. If

$$x_b = (1.b_1b_2....b_m)_2 \times 2^k$$

where

m = number of bits used for the magnitude of the mantissa, then

$$\begin{aligned} x_u &= \left[(1.b_1b_2....b_m)_2 + (0.00.....1)_2\right] \times 2^k \\ &= \left[(1.b_1b_2....b_m)_2 + 2^{-m}\right] \times 2^k \end{aligned}$$

Since x is getting represented either as x_b or x_u,

$$\begin{aligned} |x - y| &\le |x_b - x_u| \\ &= 2^{-m+k} \\ \left|\frac{x-y}{x}\right| &\le \frac{2^{-m+k}}{x} \\ &\le \frac{2^{-m+k}}{x_b} \\ &= \frac{2^{-m+k}}{(1 \cdot b_1b_2....b_m)_2 2^k} \\ &= \frac{2^{-m}}{(1 \cdot b_1b_2....b_m)_2} \\ &\le 2^{-m} = \varepsilon_{mach} \end{aligned}$$

The above proof is for positive numbers that use chopping for machine representation. We can prove the same for negative numbers as well as when one uses rounding for machine representation.