§2.1 - Floating-Point Numbers

Definition #1 (Base-2)
Let $x \in \mathbf{R}$.

Then
$$(x)_2$$
is x's binary decimal expansion.

Definition #2 (Integer and Fractional Part)
Let $x \in \mathbf{R}$ have the following decimal expansion

$$x = x_0.(x_1x_2x_3\ldots)$$

where $x_0 \in \mathbf{Z}$ and $x_k \in \{0, 1, \ldots 9\}$ for each k.

Then x_0 is the **integer part** of x and $x_1x_2x_3\ldots$ is the **fractional part** of x.

Definition #3 (Rounding and Chopping)
Let $n \in \mathbf{N}$ and $x \in \mathbf{R}$ have the following decimal expansion

$$x = x_0.(x_1x_2x_3\ldots).$$

Then

$$\tilde{x} = x_0^*.(x_1^*x_2^*\ldots x_n^*)$$

is the n-digit **rounded** approximation of x. While

$$\hat{x} = x_0.(x_1x_2\ldots x_n)$$

is the n-digit **chopped** approximation of x.

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 2.71828\ldots$$

$n = 2$

$$\tilde{e} = 2.72 \quad \& \quad \hat{e} = 2.71.$$

$$13 = 2^3 + 2^2 + 2^0, \quad (13)_2 = 1101.0$$

Now

$$(0.2)_2 = (1/5)_2 = 0.001100110011\ldots = 0.\overline{0011}$$

has infinitely many binary digits but only one decimal digit.

We will avoid expansions ending in all 9's in base-10 or all 1's in base-2.

$$(1)_2 = 0.\overline{1} \qquad (1)_{10} = 0.\overline{9}.$$

Remark

To compute x_k^* in
$$\tilde{x} = x_0^*.(x_1^* x_2^* \dots x_n^*)$$
we will use the following rules for rounding to n decimal places:

- Round up if the $(n+1)$th decimal digit is greater than or equal to 5.
- Round down if the $(n+1)$th decimal digit is less than 5

Lemma #1
Let $n \in \mathbf{N}$ and $x \in \mathbf{R}$.

If $\tilde{x}$ and $\hat{x}$ are the n-digit rounded and chopped approximations of x then

$$|x - \tilde{x}| \leq 1/2 \times 10^{-n} \quad \text{and} \quad |x - \hat{x}| < 10^{-n}.$$

Pf) Let $n=0$ and $x = x_0.x_1x_2\ldots$

Then
$$x \in [x_0, x_0+1] \quad \text{and} \quad \hat{x} = x_0 \in [x_0, x_0+1]$$
thus
$$|x-\hat{x}| < 1$$
because
$$|x-\hat{x}| = \ell([\hat{x}, x]) < \ell([x_0, x_0+1]) = 1.$$
Now $\tilde{x} = x_0$ or $\tilde{x} = x_0+1$.
If $\tilde{x} = x_0$ then
$$x \in [x_0, x_0+1/2)$$
and therefore
$$|x-\tilde{x}| < 1/2.$$
While $\tilde{x} = x_0+1$ implies
$$x \in [x_0+1/2, x_0+1]$$
or
$$|x-\tilde{x}| \leq 1/2.$$
If $n>0$ multiply by 10^n and repeat the argument on
$$y = (x_0x_1\ldots x_n).x_{n+1}\ldots$$

Definition #4 (Normalized Scientific Notation)
Let $x \in \mathbf{R}$ be nonzero.

Then

$$x = \pm r \times 10^n$$

where $r \in [1/10, 1)$ and $n \in \mathbf{Z}$ is the **normalized scientific notation** of x in base-10.

In base-2, we have

$$x = \pm q \times 2^m$$

where $q \in [1/2, 1)$ and $m \in \mathbf{Z}$.

$e = 0.271828\ldots \times 10^1$, shift right #10

$(1/5)_2 = 0.\overline{0011} = 0.\overline{1100} \times 2^{-2}$, shift left

Marc-32: left shift so we have "1.q"

$$(1/5)_2 = 1.\overline{1001} \times 2^{-3}$$

$$1.\underbrace{10011001100110011001100}_{q} \times 2^{-3}$$

Here $m=-3$ so

$$e = m + 127 = 124$$

$$(124)_2 = (2^6 + 2^5 + 2^4 + 2^3 + 2^2)_2 = 0111110.$$

s	e	q
0	0111110	10011001100110011001100

$$x = 1/5 \approx .19999988079$$

So $1/5$ is not a machine number and

$$x_- = .19999988079$$

is a machine number just to the left of x.

If we increment the last mantissa bit

s	e	q
0	0111110	10011001100110011001101

$$x_+ = .200000002980$$

is a machine number just to the right of x. Choose the nearest machine number to represent x:

$$fl(x) = .200000002980.$$

Remark

If $x \neq 0$ then some base-2 decimal is nonzero:

$$(x)_2 = 0.0001101\ldots$$

so normalized form

$$(x)_2 = (.1101\ldots) \times 2^{-3}.$$

but this nonzero entry is **always** 1!

We can exploit this if we are trying to store the number.

Definition #5 (Marc-32 Floating Point Number)

Sign - s	Biased exponent - e	Normalized mantissa - f
1 bit	8 bits	23 bits

Let

$$x = (-1)^s q \times 2^m$$

where $q = (1.f)_2$ and $m = e - 127$.

Definition #6 (IEEE 64-bit Double Floating Point Number)

Sign - s	Biased exponent - e	Normalized mantissa - f
1 bit	11 bits	52 bits

Let

$$x = (-1)^s q \times 2^m$$

where $q = (1.f)_2$ and $m = e - 1023$.

Definition #7 (Machine Number)
Let $x \in \mathbf{R}$.

If

$$x = \pm q \times 2^m$$

for machine appropriate m and q then x is a **machine number**.

Lemma #2

Let x be a machine number.

For Marc-32:

$$1.2 \times 10^{-38} \approx 2^{-126} \leq x \leq 2^{128} \approx 3.4 \times 10^{38}$$

For IEEE 64-bit:

$$2.2 \times 10^{-308} \approx 2^{-1022} \leq x \leq 2^{1024} \approx 1.8 \times 10^{308}$$

Definition #8 (Machine Epsilon)
Let ϵ be a machine number.

The smallest ϵ such that

$$1 + \epsilon \neq 1$$

is the **machine epsilon**.

Remark

There are more machine numbers near 0 than near 100:

As $|x|$ increases the smallest machine number ϵ_x such that

$$x + \epsilon_x \neq x$$

increases.

Lemma #3 (Marc-32 Rounding Errors)
For Marc-32:

$$\epsilon = 2^{-23} \approx 1.2 \times 10^{-8}.$$

So for 32-bit computations, we should expect 7-8 digits of accuracy.

For IEEE 64-bit:

$$\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}.$$

So for 64-bit computations, we should expect 15-16 digits of accuracy.

Question

Can we trust that MATLAB has implemented machine 0 correctly?

Definition #9 (Nearby Machine Numbers)
Let $x \in \mathbf{R}$.

Then x_- and x_+ are the **largest** and **smallest** machine numbers such that

$$x_- \leq x \leq x_+.$$

and $x^* = \mathrm{fl}(x)$ is the **closest** machine number.

Lemma #4

Let $x \in \mathbf{R}$ be nonzero.

In the Marc-32 we have

$$\left|\frac{x - x_-}{x}\right| \le 2^{-24} \quad \text{and} \quad \left|\frac{x - x_+}{x}\right| \le 2^{-24}$$

and so

$$\left|\frac{x - x^*}{x}\right| \le 2^{-24}$$

since x^* is either x_- or x_+.

Definition #10
Let $x \in \mathbf{R}$.

If

$$x = \pm q \times 2^m$$

where m is too large for the machine then an **overflow** error has occurred.

If m is too small for the machine then an **underflow** error has occurred.

Remark
In MATLAB:

Overflow computations are represented by $\pm$Inf.

Underflow computations are stored as 0.

Remark
Machine computations such as

$$1/0, \quad 3/\sin(0), \quad \text{and} \quad -2/0$$

will generate $\pm\infty$ or $\pm$Inf and denote unbounded numbers.

While indeterminate computations,

$$0/0, \quad \infty - \infty, \quad \text{and} \quad \infty/\infty,$$

will generate NaN or **Not a Number**.

Question

Imagine we were computing

$$\sum_{k=1}^{n} a_k.$$

Will the order of the machine addition matter?

In 64 bit precision

$$2^{52} + 1 = 2^{52}$$

so it is wise to sum numbers of the same size.
To prevent loss of precision, sum from smallest to the largest terms.