Lecture 8

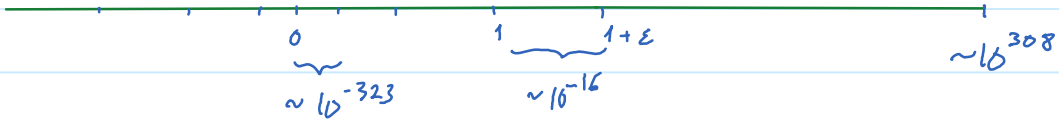
Friday, January 24, 2020

A bit sequence is what a computer stores.

$c_0 \,|\, b_1 b_2 \dots b_{11} \,|\, a_1 a_2 \dots a_{52}$ (IEEE 754-1985 standard, or double precision floating-point format)

A floating-point format is an interpretation of the bit sequence. It is of the form $x = \sigma \cdot \bar{x} \cdot 2^e$

In the IEEE floating-point format, the numbers that can be precisely represented by the bit sequences are dense near 0 and sparcer at we go further from 0.



The smallest positive number that can be represented (precisely) by a bit sequence is

$$(0.\underbrace{00\dots01}_{51})_2 \times 2^{-1022} = 2^{-1074}$$

How small is this number? Let us find a such that $2^{-1074} = 10^{-a}$.
Take natural log of both sides:

$$-1074 \ln 2 = -a \ln 10$$

We get $$a = \frac{1074 \ln 2}{\ln 10} \approx 323.$$

The smallest number larger than 1 that can be represented with exactness by a bit sequence is

$$(1.\underbrace{00\dots01}_{51})_2 \times 2^0 = 1 + 2^{-52}$$

The gap between this number and 1 is $\varepsilon = 2^{-52} \approx 10^{-16}$ This is

called the machine epsilon of the floating-point format

The largest number that can be represented by a bit sequence is

$$(1.\underbrace{11\ldots1}_{52})_2 \times 2^{1023} \approx 2 \times 2^{1023} = 2^{1024} \approx 10^{308}.$$

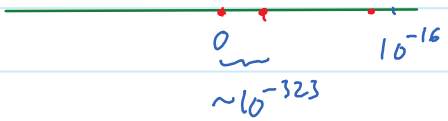
One can do the following experiments on Matlab:

```
>> 10^(-323)
>> 10^(-324)
>> 10^308
>> 10^309
```

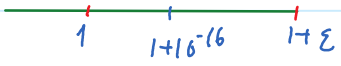
The commands

```
>> 0 + 10^-16 - 0
>> 1 + 10^-16 - 1
```

gives different results: the first one gives a number close to 10^{-16} but the second one gives 0.



Matlab performs the command $0 + 10^{-16} - 0$ from left to right. It will first add 10^{-16} to 0. Note that computers do computation only on binary numbers. They have to convert 10^{-16} into binary system (the double precision floating point format), perform the operation and convert the result to decimal format to give as output. Thus, 10^{-16} is first approximated by the nearest double precision floating-point number (the red dot). Then it is added to zero.



On the other hand $1 + 10^{-16}$ will be approximated as 1 before the subtraction. Therefore, the result is equal to 0.

* Issues caused by arithmetic of floating-point numbers :

We consider some consequences of working with floating-point format.

1) Loss of significant digits :

It is easy to see that the operations (addition, multiplication, subtraction, division) on floating-point numbers are not exact. A step of rounding is always required. Rounding can cause the loss of important digits. This leads to arithmetic mistakes such that

$$x + y = x$$

when y is too small relative to x. In this case, y is "absorbed" into x.

Ex :

$$1 + 10^{-16} = 1$$

Ex.

We know that $\lim_{h \to 0} \frac{(1+h)^2 - 1}{h} = \lim_{h \to 0} (2+h) = 2$.

We can hope that when we enter a very small value of h on the computer, it will produce a number very close to 2. Let's do an experiment.

```
>> ((1+h)^2 - 1)/h
```

For $h = 10^{-6}, 10^{-8}, 10^{-10}$, the results are quite good. For $h = 10^{-16}$, we get 0. This is because $1+h$ is rounded to 1 before being squared.

Ex :

We know that $\lim_{n \to \infty} n\left(1 + \frac{1}{n} - 1\right) = 1$. But if n is sufficiently big ($\sim 10^{16}$), matlab gives answer 0.

2) Overflow and underflow :

This issue is caused by dealing with too big or too small numbers.

Ex :

Consider a diagonal matrix A of size 400×400 where every entry on the diagonal is equal to 0.1.

$$A = \begin{bmatrix} 0.1 & & \\ & 0.1 & \\ & & \ddots \\ & & & 0.1 \end{bmatrix}_{400}$$

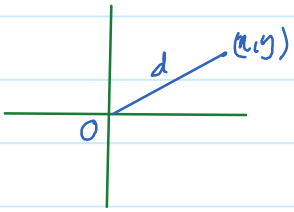
The dimension of A is not too big. In application, it is common to deal with even bigger matrices. For example, in the method called Finite Element method, one deals with a matrix called "stiffness matrix" This is usually a very big matrix. The size of each entry of A is not too small. A is obvious not a singular matrix because $A = (0.1)I_{400}$. However, Matlab considers as singular because

$$\det(A) = (0.1)^{400} = 10^{-400} \approx 0.$$

This phenomenon is called underflow.

Ex: the distance from a point (x,y) on the plane and the origin is

$$d = \sqrt{x^2+y^2}$$



Mathematically,

$$\sqrt{x^2+y^2} = x\sqrt{1+\left(\frac{y}{x}\right)^2}$$

However, these expressions are different from a computation perspective.

When x and y are big, say $xy \sim 10^{200}$, the LHS is equal to ∞. But the right hand side is $10^{200}\sqrt{2} \approx \infty$. (still within the range that double precision floating-point format can represent).

3) Noise caused by the randomness of rounding

Consider two expressions

$$f_1(x) = (x-1)^3$$

$$f_2(x) = x^3 - 3x^2 + 3x - 1$$

They are mathematically equivalent However, they are different computationally It takes 2 multiplications to compute f_1, but 5 multiplications to compute f_2. More arithmetic operations being done lead to more roundings being made. Let's take a look at a narrow interval around 1, say $[1-10^{-5}, 1+10^{-5}]$

The computation of f_2 involves more rounding steps. The rounding errors at each step (x, x*x, x*x*x, 3*x, 3*x*x) are relatively independent of each other Moreover, the (total) errors when x varies in the interval are relatively random. Thus, one can observe random fluctuations of the value of $f_2(x)$ as x varies on the interval. Test the following code on Matlab.

```
h = 10^(-6);
x = 1-10^(-5) : h : 1+10^(-5);
y1 = (x-1).^3;
y2 = x.^3 - 3*x.^2 + 3*x - 1;
plot(x,y1,'.b',x,y2,'.r')
```