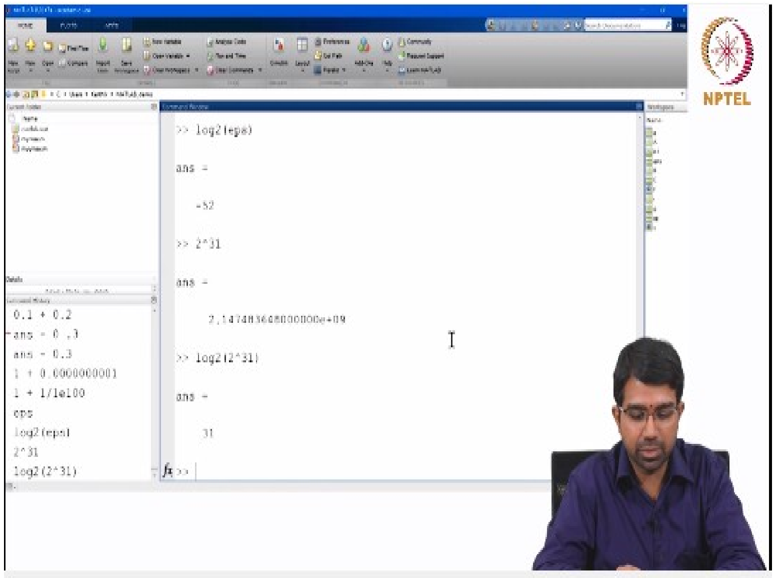
Computational Systems Biology

Karthik Raman
Department of Biotechnology
Indian Institute of Technology – Madras

Lecture – 12
Lab: MATLAB Basics

In this last MATLAB introduction video, we will focus on a very important topic namely floating point arithmetic and this has implications for how one writes numerical algorithms to solve different kinds of problems and so on although we won't be delving into great details, you should look it up and this video will give you at least an overview of the surprises one might expect, the surprises one might encounter, when you work with floating points on any computer.

(Refer Slide Time: 00:39)



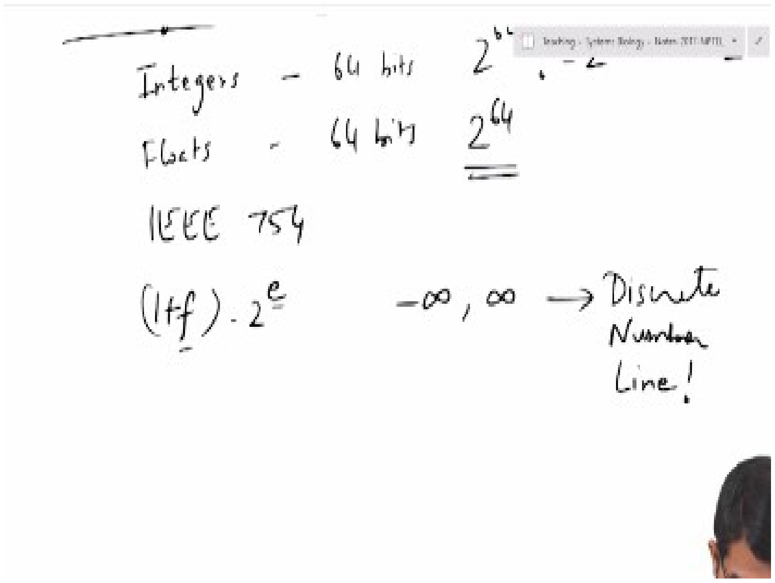
```
>> log2(eps)

ans =

   -52

>> 2^31

ans =

     2.147483648000000e+09

>> log2(2^31)

ans =

    31

>>
```

```
0.1 + 0.2
ans - 0 .3
ans - 0.3
1 + 0.0000000001
1 + 1/1e100
eps
log2(eps)
2^31
log2(2^31)
```

And as always, I must draw your attention to how floating points are represented, right you can always have some interesting, right, so oops! 0.1+ 0.2 - 0.3 is not 0, welcome to floating point world, right, so if I add 1 + 0.0000000001,a very small number, it will give me something, if I add 1 + 1/1e100, it will give me 1, right, so when does this change? There's something known as machine epsilon and this is typically 2^{-52} on most modern computers, it is 2^{-52}, right.

So, this is the smallest distance between 2 numbers on the computer's number line, right. In a real number line, you have infinitely many numbers between any 2 numbers. But computer's number line is basically discrete, right. So, how does the number line look like? So, how many different floats can you store on a computer? So, we have a 32-bit representation. How many integers can you store?

(Refer Slide Time: 02:59)


Integers - 64 bits 2^{64}

Floats - 64 bits $\underline{\underline{2^{64}}}$

IEEE 754

$(1+f).2^{e}$ $-\infty, \infty \rightarrow$ Discrete Number Line!

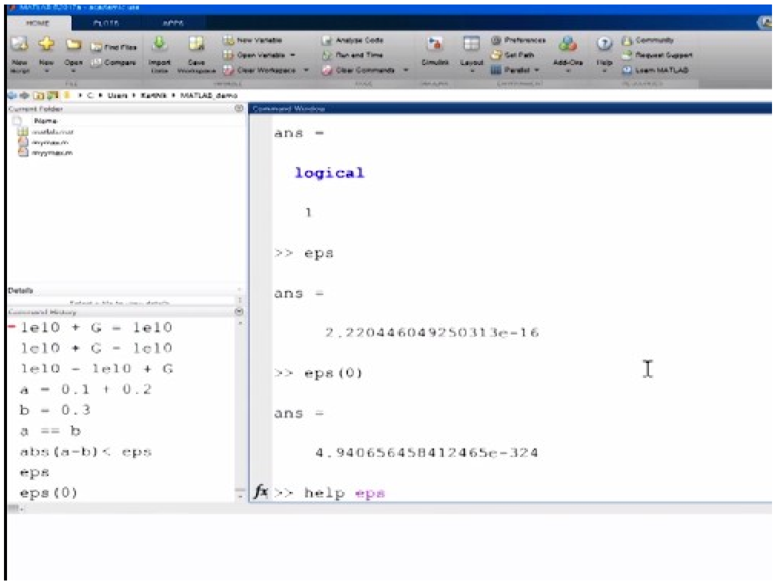
-52 is for 64-bit. How many integers can you store in 64 bits? How many integers can you store in 64-bits? It is not a trick question, it is basically 2^{64} typically, $2^{63}…2^{63}$-1.

So, how many floats in 64-bit? It is actually the easiest to think of. 2^{64} floats, if you have 64 bits, you can represent 64 different numbers using the 64-bits, as simple as that right, whether those binary numbers represent integers or floats or whatever else depends upon the you know how it's encoded and so on, if you have standards. So, there is something known as which I think everybody should try to quickly look at.

The IEEE 754 standard. The IEEE 754 floating point standard, this basically says that every number is represented as $(1+f).2^{e}$, so every number is actually represented as some f, e and the values, the maximum values of e and all of that vary depending upon how many bits you are using and so on, so in 64, there is a particular set of numbers and so on. I think it is worth understanding this.

Because the first thing you need to know is that because you have only 2^{64} bits, you are compressing all numbers between -infinity and +infinity to a discrete number line. Supposed to be continuous, right, 264 is big but you don't have every number there, so you can't, so let's look at some interesting things here.

(Refer Slide Time: 07:39)



```
1e10 + G - 1e10
1e10 + G - 1e10
1e10 - 1e10 + G
a = 0.1 + 0.2
b = 0.3
a == b
abs(a-b)< eps
eps
eps(0)
```

```
ans =

  logical

   1

>> eps

ans =

     2.220446049250313e-16

>> eps(0)

ans =

    4.940656458412465e-324

>> help eps
```

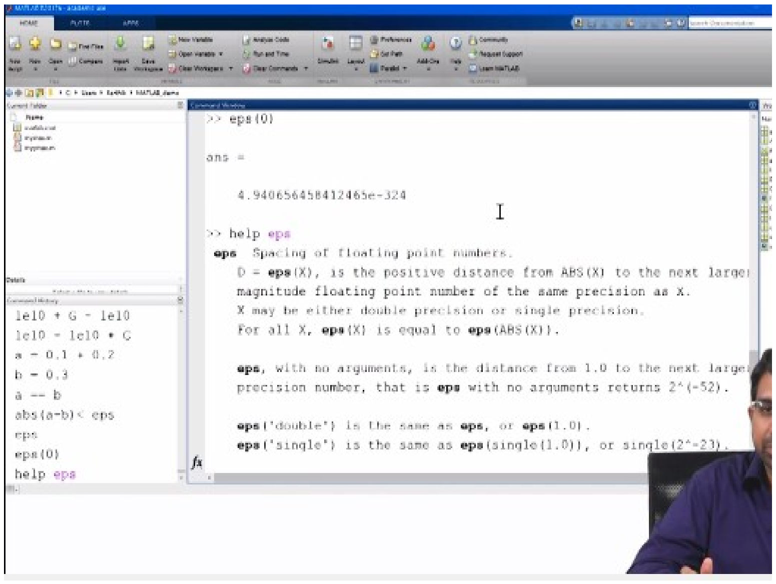
What is epsilon(eps)? Epsilon is this very small number, 2^{-52}. Can I compute eps/100? Well, certainly not, if eps/100 were 0, then you can never say represent gravitational constant, G as 6.6734*1e-34, right? So, you need those small numbers to be represented, so basically you don't have a uniform number line, the number line increases in resolution in some places, decreases in resolution in some places.

Obviously, I can't add 1e10+G –1e10. It will be 0 but 1e10– 1e10+G, will be G, so addition is no longer associative, welcome to floating point arithmetic. a+b+c is not the same as a+(b+c), so these are some of the things you need to be vary of. Why is this important? This is mostly not important for us, it is not important when you look at network biology, it is not important when you look at constraint-based modelling.

It becomes very important when you look at ODE solving, ODE solving will have all kinds of weird things, when you use solvers and these numerical inaccuracies will blow out and so on. So, you need to be quite careful with this and in general, if you are doing any numerical computation, you should never compare if 2 floats are equal, you should not let's say a=0.1+ 0.2, b=0.3, you should not ever say a = = b.

It says a is not equal to b, you should always check, if abs(a-b)<eps, very good. And you often don’t use something as low as epsilon, you will use a smaller number like 1e- 6 or something like that. So, what is therefore, eps? So, eps of, oh! eps on MATLAB is a cool quantity, it is the smallest number when added to a number, will make it different.

(Refer Slide Time: 10:50)

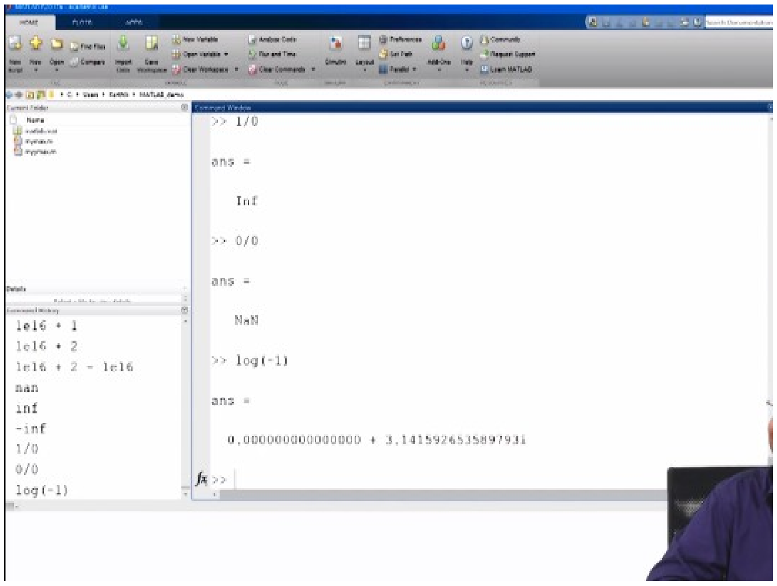


```
>> eps(0)

ans =

   4.940656458412465e-324

>> help eps
 eps  Spacing of floating point numbers.
   D = eps(X), is the positive distance from ABS(X) to the next large
   magnitude floating point number of the same precision as X.
   X may be either double precision or single precision.
   For all X, eps(X) is equal to eps(ABS(X)).

   eps, with no arguments, is the distance from 1.0 to the next large
   precision number, that is eps with no arguments returns 2^(-52).

   eps('double') is the same as eps, or eps(1.0).
   eps('single') is the same as eps(single(1.0)), or single(2^-23).
```

```
1e10 + G - 1e10
1e10 - 1e10 + G
a = 0.1 + 0.2
b = 0.3
a == b
abs(a-b)< eps
eps
eps(0)
help eps
```

So, it basically gives you the spacing of floating point numbers and this spacing is not uniform.

(Refer Slide Time: 11:02)



```
>> 1/0

ans =

   Inf

>> 0/0

ans =

   NaN

>> log(-1)

ans =

  0.000000000000000 + 3.141592653589793i

>>
```

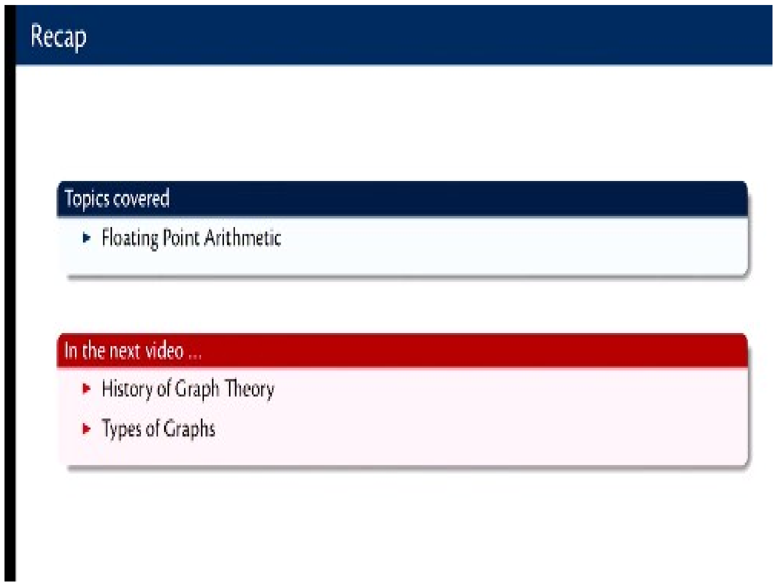
```
1e16 + 1
1e16 + 2
1e16 + 2 - 1e16
nan
inf
-inf
1/0
0/0
log(-1)
```

Yes, so 0+eps(0), no problem, 0+eps(0)/2, this is the smallest number, why by 2, 0+eps(0)*0.9999.

1e16+1 is 1e16. 1e16+2 is 1e16. But, 1e16+2-1e16 makes it different from 1e16. Interesting scale, that is the IEEE 754 notation that people came up with, this is to make sure that you maximise the number of possible computations. You also have some special floats, NaN, Inf, -Inf, these are all special floats. NaN is not a number, 0 by 0 indeterminate quantities.

log (-1), oh wait, here, it goes to complex, so, I think this is about what I wanted to cover, so I hope you got a quick, very quick introduction to MATLAB, so I think the most important things to know are logical indexing and ways to not use, think of ways to not use loops in your MATLAB code.

(Refer Slide Time: 12:42)



Recap

Topics covered

- Floating Point Arithmetic

In the next video ...

- History of Graph Theory
- Types of Graphs

So, I hope today, you had a very interesting introduction to floating point arithmetic and it will prompt you to go and read a little more about these interesting topics. In the next video, we will go back to graph theory and we will start to the history of the graph theory and the different types of graphs.