Computational Techniques

Prof. Dr. Niket Kaisare

Department of Chemical Engineering

Indian Institute of Technology Madras

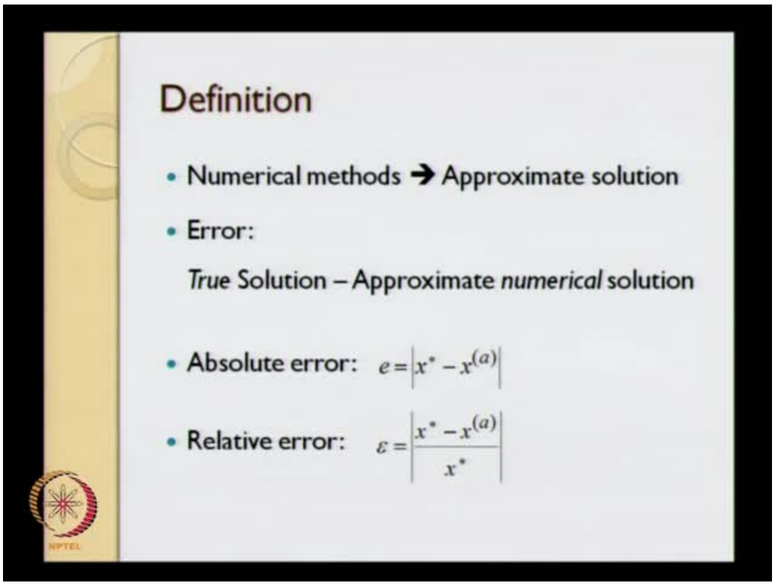
Module No. # 02

Lecture No. # 01

Computational and Error Analysis

Hello and welcome to the module 2 for the computational techniques course. In module 2, we are going to cover computation and error analysis. And what I will do here is just go through an overview of what we are going to cover in this particular module.

(Refer Slide Time: 00:27)



Definition

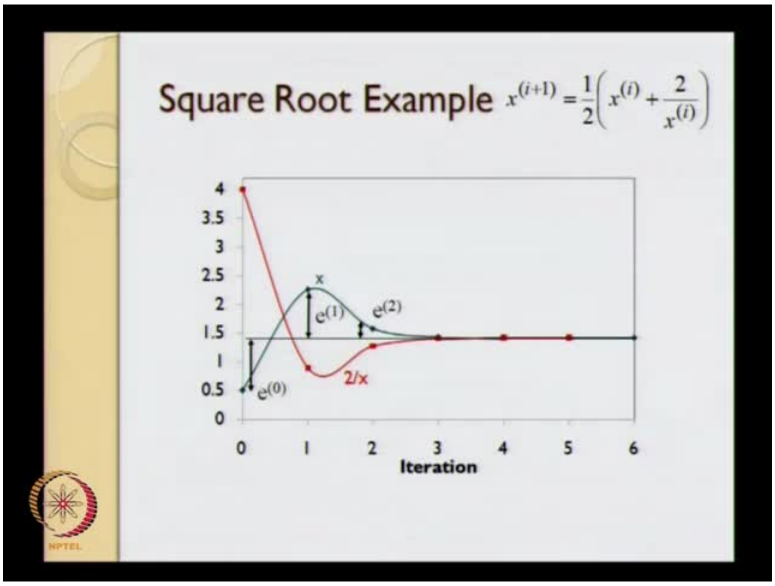
- Numerical methods → Approximate solution
- Error:

  True Solution – Approximate *numerical* solution
- Absolute error: $e = \left|x^* - x^{(a)}\right|$
- Relative error: $\varepsilon = \left|\dfrac{x^* - x^{(a)}}{x^*}\right|$

The definition of errors: as we had seen in the introduction module, numerical methods will lead to approximate solutions; so, what that means is that the solution will differ from the actual value of the solution. The difference between the true solution and the approximate numerical solution is what we call the error. There are various reasons why these errors come about; these errors are classified into two different types, what is known as round off errors and truncation errors. We will talk about that in this particular module. Before we do that, we will just define what we mean by - or how what we mean by - the error; we are going to use two different definitions of error - one is the small e over here is just the difference between x star, where x star is what we are representing as the true value of the solution, and x a over here represents the approximate value of the solution. So, the difference between the two is going to represent error.

Again we do not want to bias ourselves to positive e or negative values; and therefore, we are going to use the absolute value of that difference as our definition of error; this is what is known as the absolute error. Often what is important is not just the absolute value of error, but the value of error relative to the true value of the solution and that is what is known as the relative error. And the relative error is defined as: a difference in the true value minus the approximate value divided by the true value - the absolute value of that. So, the reason for doing this is because an error of 0.1 is very large, if the true value of the solution is say 0.2; but it is negligibly small, if the true value of the solution is say 1000. So, we need to differentiate between the error value, what they really mean, and that is the reason why we have an absolute value, absolute value error and the relative error.

(Refer Slide Time: 02:40)

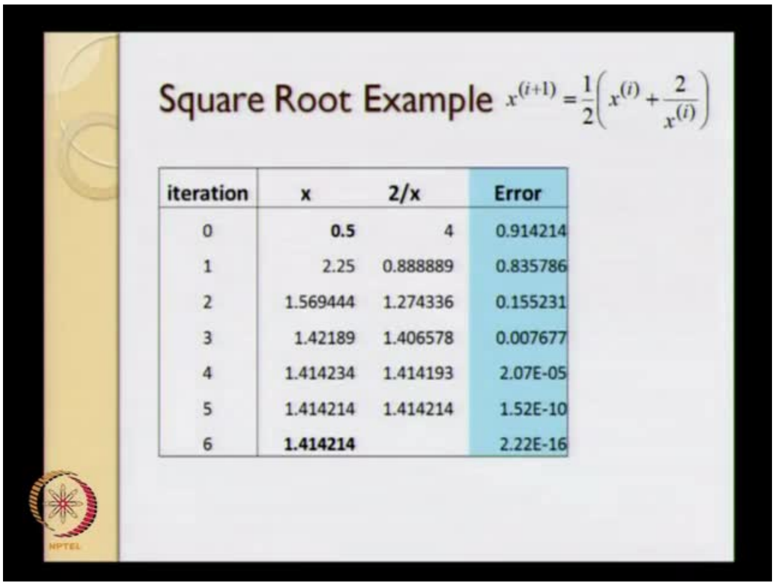


Square Root Example $x^{(i+1)} = \frac{1}{2}\left(x^{(i)} + \frac{2}{x^{(i)}}\right)$

This is the square root example from the Henon algorithm, that we had talked about in the introduction lecture; and the x i plus 1 over here is an average between the previous value x i and 2 divided by x i, it is just an algebraic mean of these two values; and this is the curve that we had seen - the blue and the red curves - we saw in the previous module, and these are the approximate values of x and how these approximate values of x change with the iteration number; the black line over here represents the true solution x star; the difference between the true value and the approximate value is what we call the error; so, e 0 over here is essentially this particular difference between 1.4 1 4 and 0.5; the value e 1 is essentially, this particular difference e 2 is essentially this difference, e 3 is this, e 4 is this, so on and so forth. So, the definition of error really is that, the absolute error is the difference between the true value and the approximate value; and the relative error is this difference divided by this particular true value of the solution.

So, that is as it goes with respective the definition; what we mean by error propagation is, the error e 1 depends on the previous value x and the value x star or in general, it depends on the error e 0; likewise, error e 2 depends on the value x 1 as well as the error e 1, so on and so forth. So, what we see over here is the error e 0 and the value x 0 are going to determine what the next error e 1 is going to be, that in turn determines what the next error e 2 is going to be, what the next error e 3 is going to be, and so on and so forth. An understanding of how this error propagates is going to be very important in analyzing these numerical techniques; so, that is what this numerical techniques we are going to do. In this particular lecture is define the error - define what we mean by error propagation - and try to find out what are the reasons? Why this particular error comes about.

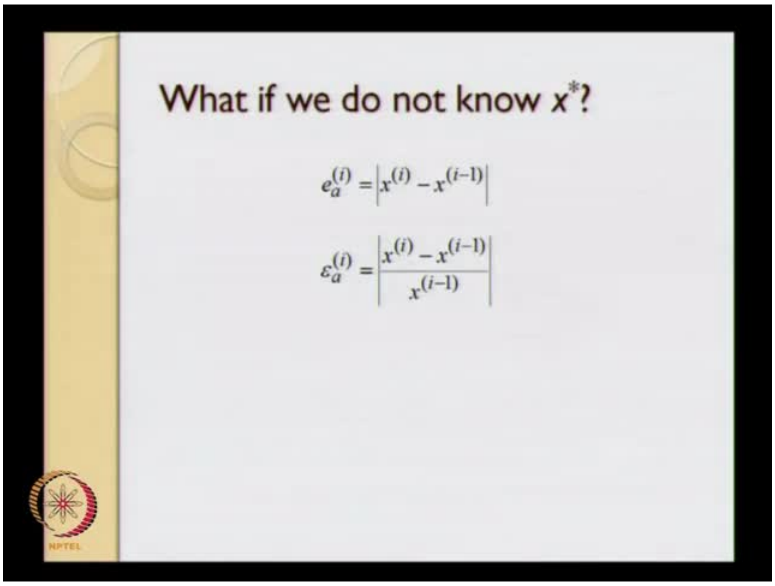
(Refer Slide Time: 05:06)



Square Root Example $x^{(i+1)} = \frac{1}{2}\left(x^{(i)} + \frac{2}{x^{(i)}}\right)$

iteration	x	2/x	Error
0	**0.5**	4	0.914214
1	2.25	0.888889	0.835786
2	1.569444	1.274336	0.155231
3	1.42189	1.406578	0.007677
4	1.414234	1.414193	2.07E-05
5	1.414214	1.414214	1.52E-10
6	**1.414214**		2.22E-16

(Refer Slide Time: 06:02)



What if we do not know x^*?

$$e_a^{(i)} = \left|x^{(i)} - x^{(i-1)}\right|$$

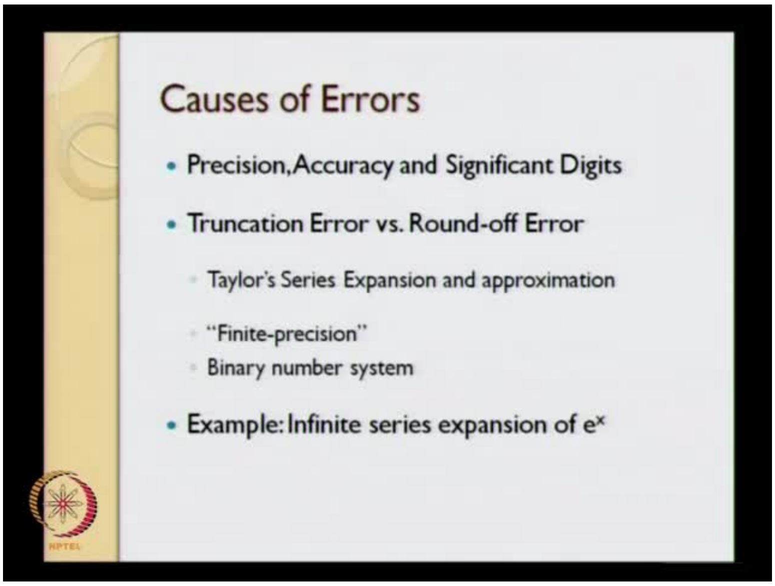
$$\varepsilon_a^{(i)} = \left|\frac{x^{(i)} - x^{(i-1)}}{x^{(i-1)}}\right|$$

Again this is the square root example and these are essentially that the absolute errors; so, this is 1.414 minus 0.5 the absolute value of that, 1.414 minus 2.25 the absolute value of that, so on and so forth; the column here is the last column over here, represents the error in the approximate solution and as you can see the error is quite significant, we start with an error of 0.9 and very quickly the error drops to 10 to the power minus 16. So, how this error behave as the iteration proceeds is going to be a very important property of any numerical technique and that is what we are going to cover in the various different numerical techniques, that we are going to talk about starting from the next module; and the question is, now what if we do not know x star.

You know, in this particular example that, we took we knew what the true value of x star was going to be; but in general, when we have we are going to use these numerical techniques, we do not know what this x star value is going to be. So, we define what is known as the approximate approximation error; so, we represent this as e a and again the e a at the i iteration is the difference between the value that we have from the i iteration and the value that we have from the i minus 1 iteration; and the relative error over there is again in the same way as in the previous case, we have defined that as epsilon a and it is the difference - divided by the true value of sorry - divided by the approximate value x i minus 1. And essentially, what that particular definition means is that, the difference between this value and this value is going to be e 1, difference between this and this value is going to be e 2, e 3, e 4, and so on; and if you remember from the module 1, what we had said is, we have stopped at iteration number 6, the reason why we stopped at iteration number 6, is because the result wasn't changing from the fifth iteration to the sixth iteration up to the desired accuracy, that we were seeking; and the desired accuracy we were seeking over here is that, the sixth decimal place should not change.

So, the stopping criteria that we decided, essentially was not based on whether the true value is reached or not, but whether there is a significant change with the iteration in the approximate value, we have or not and this is where the approximation error between i and i minus 1 iteration comes very handy, is when we do not know the value of x star.

(Refer Slide Time: 08:05)

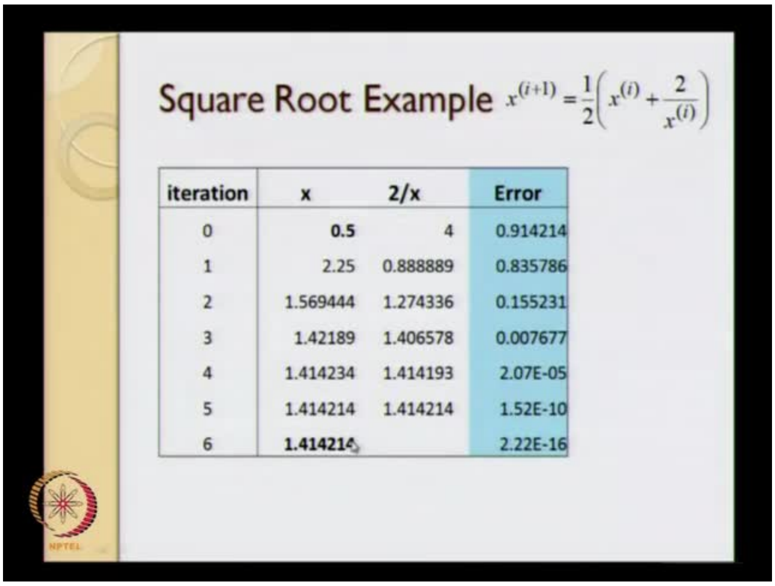


Causes of Errors

- Precision, Accuracy and Significant Digits
- Truncation Error vs. Round-off Error
  - Taylor's Series Expansion and approximation
  - "Finite-precision"
  - Binary number system
- Example: Infinite series expansion of e^x

Next, we are going to look at the causes of the errors. Before we do that, we will discuss about, what we mean by precision? What we mean by accuracy? What we mean by significant digits?

(Refer Slide Time: 08:21)



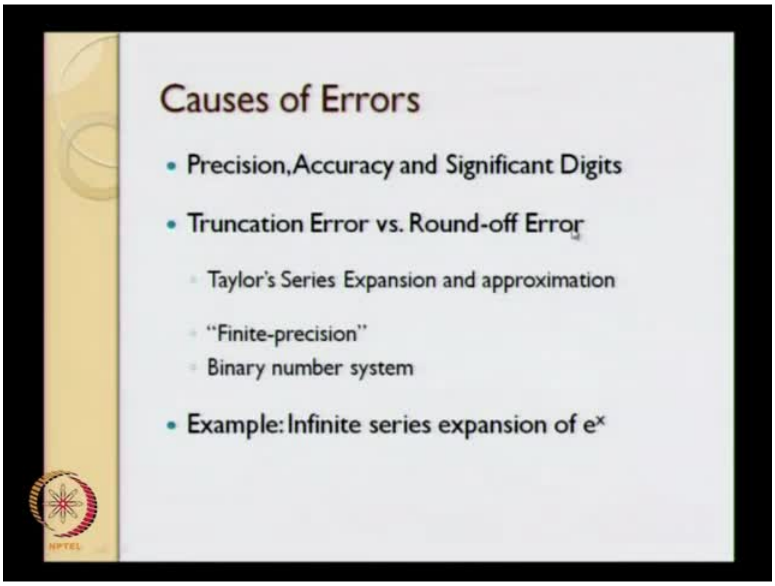
Square Root Example $x^{(i+1)} = \frac{1}{2}\left(x^{(i)} + \frac{2}{x^{(i)}}\right)$

iteration	x	2/x	Error
0	**0.5**	4	0.914214
1	2.25	0.888889	0.835786
2	1.569444	1.274336	0.155231
3	1.42189	1.406578	0.007677
4	1.414234	1.414193	2.07E-05
5	1.414214	1.414214	1.52E-10
6	**1.414214**		2.22E-16

In the previous example, the significant digits that we looked at are essentially 7 significant digits. We know for sure that the solution is correct up to this particular number of digits, we do not know what happens beyond this particular point.

So, we are interested essentially in six digits after the decimal point; in this particular case, there is 1 digit before the decimal point and we are interested in the result up to the seventh significant digits.

(Refer Slide Time: 08:48)



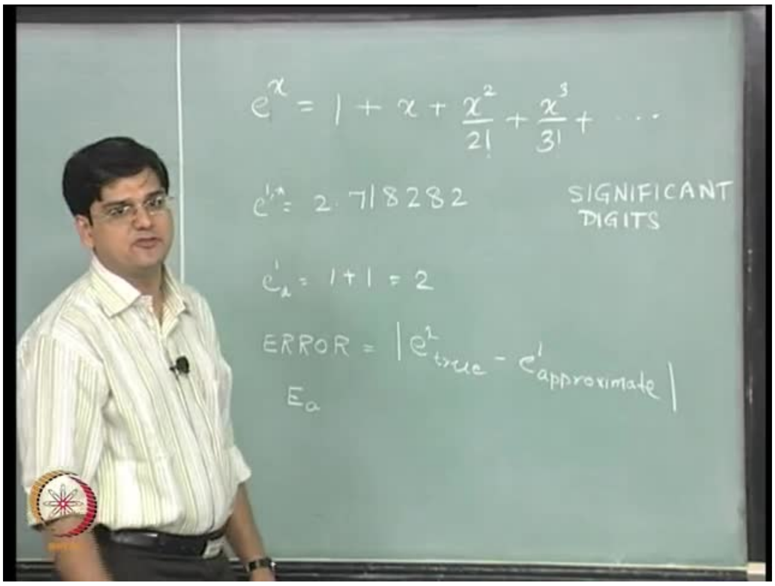
Causes of Errors

- Precision, Accuracy and Significant Digits
- Truncation Error vs. Round-off Error
  - Taylor's Series Expansion and approximation
  - "Finite-precision"
  - Binary number system
- Example: Infinite series expansion of e^x

Next, we are going to look at what truncation error and round off error means; again we will go back to the work horse Taylor’s series expansion and approximation of the Taylor series.

We will talk about the computer, the binary number system and the fact that the computer is what is called a finite precision machine; and we will take an example of an infinite series of e to the power x in order to use - in order to continue with this particular module. So, that is what we are going to do in this particular module.

(Refer Slide Time: 09:30)

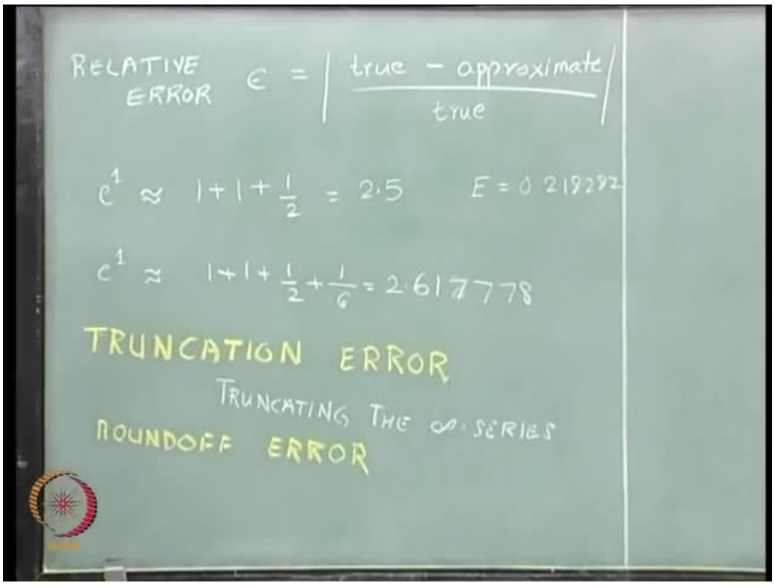

$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$
$e^{1*} = 2.718282$
SIGNIFICANT DIGITS
$e^1_a = 1 + 1 = 2$
$\text{ERROR} = |e^1_{true} - e^1_{approximate}|$
E_a

Let us just look at the example of the infinite series expansion of e to the power x; so, as we had said before, e to the power x can be written as 1 plus x plus x square by 2 factorial plus x cube by 3 factorial and so on; so, this is an infinite series, the greater the number of terms that you use, more accurate your results are going to be.

So, let us look at the problem of e to the power 1; now, e to the power 1, it is 2.718282, so, we are stopping at essentially this particular point and what this signifies is the number of significant digits, I will come to that later - number of significant digits. Now, this is the actual value of e to the power 1; we will call this as star, star to represent that the true value. Now, we will try to get the approximate values of e to the power 1 by including additional terms in this infinite series.

So, if we use only the first two terms of this infinite series, we will get e to the power 1 approximate - the subscript a stands for approximate - is 1 plus 1 that is going to be equal to 2. So, if you see that the difference between the true value of e to the power 1 and the approximate value of e to the power 1, is going to be significant when you are just going to include only the first term over here; this difference between the two is what is known as error; so, error is going to be defined as the difference between the true value minus e to the power 1 approximate value.

(Refer Slide Time: 12:43)



RELATIVE ERROR $\epsilon = \left| \frac{\text{true} - \text{approximate}}{\text{true}} \right|$

$e^1 \approx 1 + 1 + \frac{1}{2} = 2.5$ $E = 0\ 219282$

$e^1 \approx 1 + 1 + \frac{1}{2} + \frac{1}{6} = 2.617778$

TRUNCATION ERROR

TRUNCATING THE ∞. SERIES

ROUNDOFF ERROR

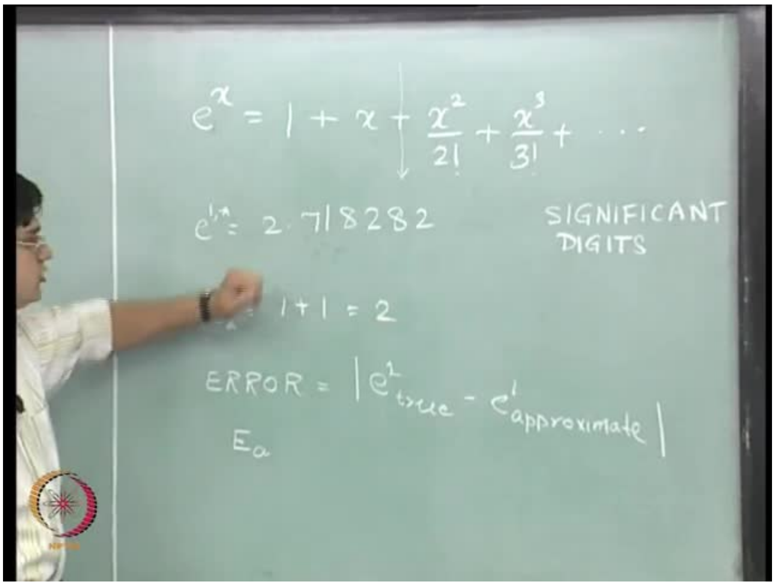
Now, we do not want to bias ourselves saying that, the positive error is acceptable or negative error is acceptable and so on; so, error in either direction is not acceptable, what we want to do is, we want to really reach the value 2.718282 as closely as possible. So, the error is actually going to be defined as the absolute value of the difference between the two; this is known as the absolute error; and we will represent it as E a, the absolute error. Now, the absolute error is not the only criterion that we are interested in, what does some times more useful is to find out how far the two numbers deviate compared to what the true value of e to the power 1 is; and in that particular case the error which is known as the relative error, which we will represent using the term epsilon, that we will define as the absolute value of the true value minus the approximate value divided by the true value; so, this is the relative error that we are interested in tracking.

Now, let us involve instead of just the first two terms, we will involve the first three terms over here and in that particular case, e to the power 1. Or I should actually we writing at approximately equal to is 1 plus 1 plus 1 by 2 which is equal to 2.5. So, now what you see is the error has decreased, when you are increasing the number of terms in this expansion - the infinite series expansion of e to the power x. So, the error over here is going to be 2.718 minus 2.5, that is 0.218282, that is what the error is going to be when we include three terms.

Now, we can include additional term; when we include the fourth term, what we will find is the error, is going to be 1 by 2 plus 1 by 6, it is going to be 2.5 plus 0.18; so, that will be 2.617778. So, now this value is getting closer to the value 2.71828; so, what we see over here is, as we include additional terms in this infinite series expansion; we are - - going - getting closer and closer to the true value of e to the power 1. So, that leads me to the first definition that is of a importance to us and that definition is truncation error.

Truncation Error comes up because we are not going to use infinite number of terms in the series expansion, but at some pint of time we will say that, we are not going to use any additional term and this current approximate value is sufficient for our purpose; so the truncation error comes about because we are truncating this infinite series at certain point.

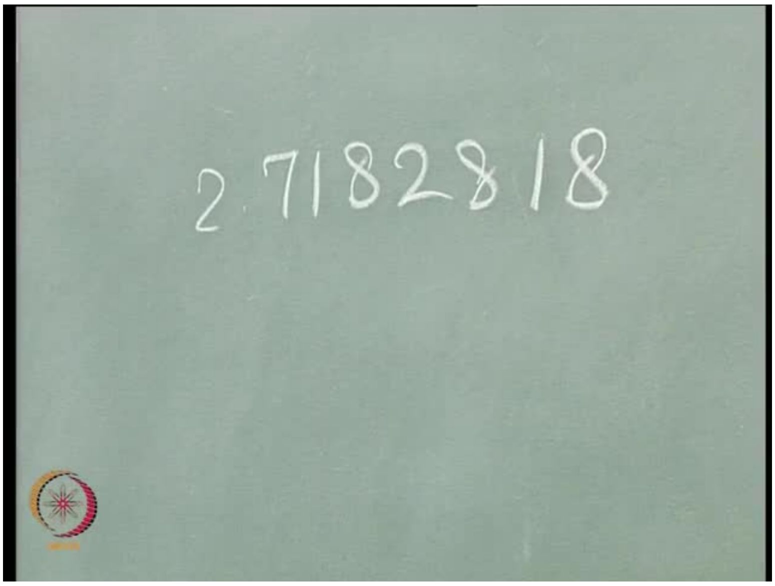
(Refer Slide Time: 15:54)


$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$
$e^{1} = 2.718282$
SIGNIFICANT DIGITS
$1 + 1 = 2$
ERROR = $|e^1_{true} - e^1_{approximate}|$
E_a

So, what happens is, when we truncate the series at this particular location, the difference between the approximate value of e to the power 1 and true value of e to the power 1 is arising, because we are truncating at the second term; in the second case, this truncation error is dropping; the reason why this truncation error is dropping down is because, we have used an additional term in this error, in this approximate value of e to the power 1; so, the idea about truncation error is because it arises because we are truncating the infinite series; so, that is where the truncation error arises.

Now, the second type of error which is of vital importance to us is, what is known as the round-off error? So, for example, if you are using a calculator like I am using, the maximum number of digits in this particular calculator are going to be 10 digits; so, you can only represent numbers up to 10 significant digits using a calculator of the kind, that I am using.

(Refer Slide Time: 17:48)

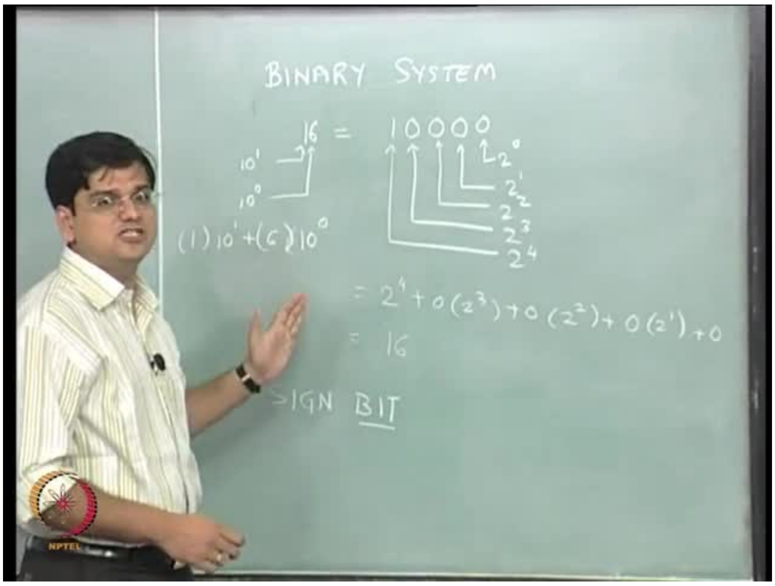


2.7182818

So, any error that comes up, because you are chopping off the remaining part of the number come is what we call as the round off error. For example, e to the power x is actually not 2.718282; if you include additional terms it might end up being 2.7182818 and then additional terms beyond that particular value, so when we are stopping at this particular point, what we are essentially doing is we are neglecting all the data beyond this particular value.

So, that round off error now comes up, because we are chopping in our mathematical representation; we are chopping off the way we are representing a complete number.

(Refer Slide Time: 18:23)



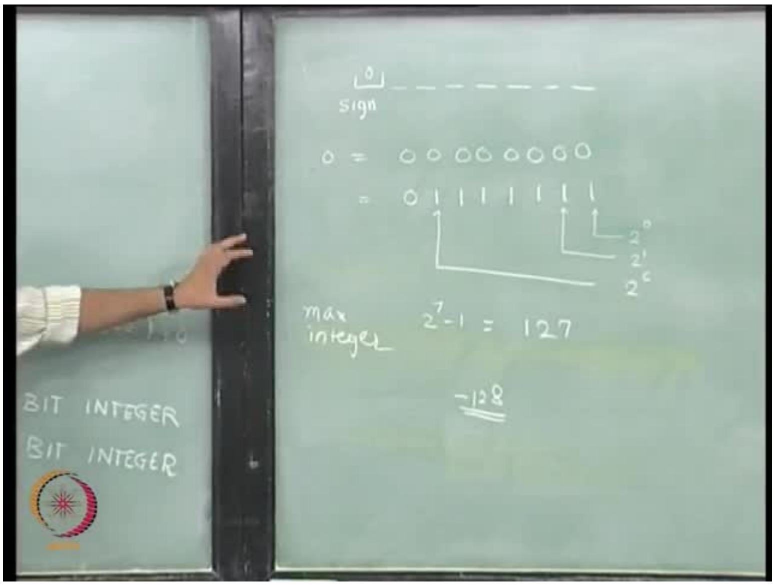
So, the binary system uses numbers 0 and 1 to represent any number within the computer. So, for example, let us say, we have the number 16, this number 16 in binary is going to be represented as 10000; this particular digit represents 2 to the power 0; this one represents 2 to the power 1; this represents 2 to the power 2; this is 2 to the power 3; and this is 2 to the power 4.

So, this number what it means is, 2 to the power 4 plus 0 multiplied by 2 to the power 3 plus 0 multiplied by 2 to the power 2 plus 0 multiplied by 2 to the power 1 plus 0 which is equal to 16; so, this particular number is the binary representation of the number 16 in in the decimal representation. Likewise, in the decimal representation also, the way we represent a decimal number is really this is going to be 10 to the power 1 and this represents 10 to the power 0; so, 16 really is, 1 multiplied by 10 to the power 1 plus 6 multiplied by 10 to the power 0; so that is 10 plus 16.

This is how, essentially, the numbers the integer numbers are going to be represented, if we have only interested in 0 and positive integer numbers; if we are interested in negative integer numbers also, in addition to this particular, these 5 numbers we will require one more digit to represent, whether it is a positive number or a negative number; and that particular digit is what is known as a sign bit.

Bit stands for a binary digit and the sign bit represents that it determines, whether the sign of that particular number is positive or negative.

(Refer Slide Time: 21:08)



So, in general, what we are going to have in a computer representation is that, an integer number will be represented by a finite number of bits; the first bit represents the sign bit and the remaining bits represent the actual number; the first bit over here will be a sign bit and the remaining 5 6 7 remaining 7 bits are going to represent a number.

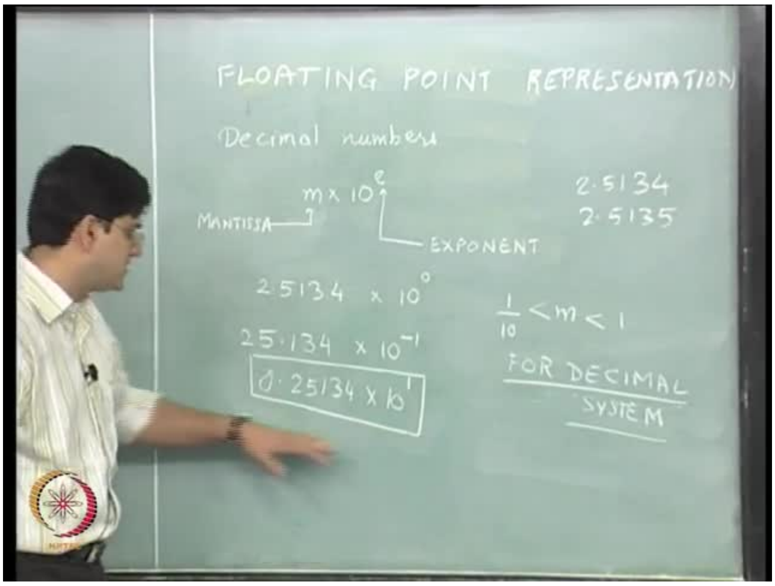
So, let us first talk about a positive number; for a positive number, the sign bit is going to be 0, and then remaining bits are going to represent the rest of the numbers; the smallest non-negative number that you can represent is, of course, going to be 0 and the 0 is going to be equal to 00000000.

So, we have eight 0s representing the number 0, the highest positive number that we will able to represent in this binary system, is going to be 0 which stands for the sign bit representing a positive number followed by seven 1s.

Now, what the number does this represent, this represents 2 to the power 0 plus 1 plus 2 to the power 1 plus 1 and so on up to 2 to the power 6 plus 1, right 2 to the power 0 1 2 3

4 5 6. So, the maximum number this is going to represent is, 2 to the power 0 plus 2 to the power 1 plus 2 to the power 2 up to 2 to the power 6, which we know can be written as 2 to the power 7 minus 1; so, the maximum integer that can be represented using this particular notation is, 2 to the power 7 minus 1, which turns out to be 127.

(Refer Slide Time: 23:46)



So, 127 is the largest number that can be represented using an 8 bit integer representation. Likewise, the smallest number that can be represented is going to be actually minus 128; and I would not go into the details of how the negative numbers are represented. Let us just stick to the fact that, for the negative numbers in the sign bit, we will have the number as 1; and for the positive numbers the sign bit, we will have the number 0. When we go to real numbers, what happens is, the things get a little bit more complicated and the reason why it gets complicated is because a number is going to be always represented in a computer using what is called finite precision.

So, I will talk about floating point representation for decimal numbers. The floating point representation will always be written as, some number m multiplied by 10 to the power e; this particular m carries the name mantissa; and e is called the exponent.

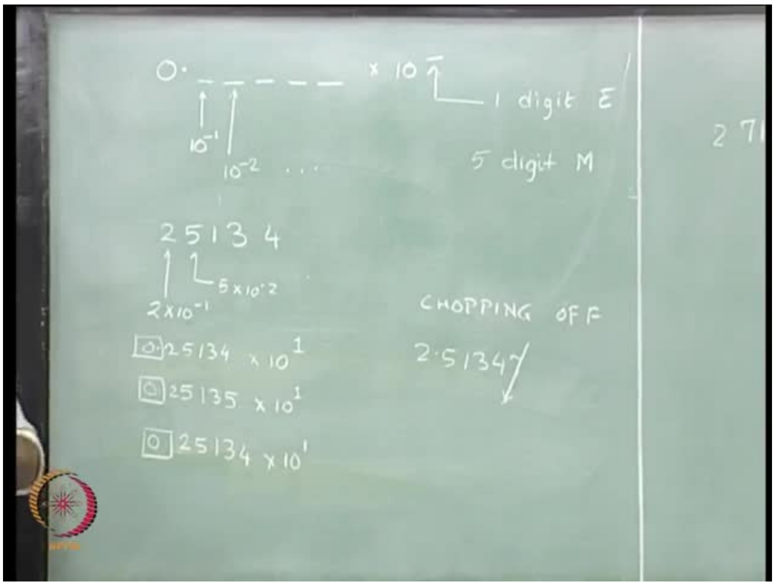
So, let us take this particular number 2.5134; so, 2.5134 in this kind of representation will be represented, let us say, as 2.5134 into 10 to the power 0.

So, this particular number is 2.5134. Now, we can also represent this number as 25.134 multiplied by 10 to the power minus 1. We can also represent this as, 0.25134 multiplied by 10 to the power 1.

Now, the question is which of these representations is correct. Now, when we are talking about, you know, doing things by humans; all these representations mean one and the same thing you can interchangeably use any of this representation and you know really what that particular representation means and you can do your calculations. However, when you are going to use a floating point representation, we require certain ways of representing the numbers, that is uniform and that is standard over various different types of machines.

So, the standard that is used in floating point representation is that, this mantissa should necessarily lie between 1 divided by 10 and 1 for decimal systems. Again I am taking some pedagogical liberties over here, so please bear with me and I will try to summarize all of this after I finish explaining the floating point representation.

(Refer Slide Time: 27:23)



So, with this particular definition, that m lies between 0.1 and 1; the correct representation of our number is going to be this; so, instead of having a binary computer, if we had a decimal computer, the floating point number 2.5134 would be represented as, 0.25134 multiplied by 10 to the power 1. Any number in our decimal computer is going to be represented as 0 point dash dash dash dash dash, so we have five blanks over here, so we are using five digit mantissa multiplied by 10 to the power dash. So, we are using 1 digit exponent. In order to not complicate things, we will just look at the positive values of exponent, we will only look at the positive values of mantissa; the reason is, because in binary notations, for example, you will use a sign bit in order to determine the sign of the entire number and you will use a signed way of representing the exponent. We will come to that later, but let us just assume for the time being for the sake of simplicity, that we are only going to represent positive numbers with positive exponent.

So, what does, what do actually these five blanks really mean; this blank really means that, this particular number is multiplied by 10 to the power minus 1, just the way we had it in the binary notations of the integers; likewise, the decimal notation of the mantissa, is this guy multiplied by 10 to the power minus 1, this multiplied by 10 to the power minus 2 and so on.

So, when we have the five digit representation 2 5 1 3 4; this actually means, 2 multiplied by 10 to the power minus 1 plus 5 multiplied by 10 to the power minus 2 and so on, which will give us 0.25134; keep in mind that, our decimal computer is not going to waste any space in representing this 0 dot, it is only going to represent this five numbers. The first digit is going to be multiplied by 10 to the power minus 1; second digit 10 to the power minus 2; third digit 10 to the power minus 3, so on and so forth.

Now, let us say that, this one, this is the kind of representation we are going to use. So, the first, that the number that we can represent the 2.5134, the way we will represent is as we said is 0.25134 multiplied by 10 to the power 1; the number 2.5135, we will represent that as 0 point and again I am putting this in a bracket, because in our decimal computer we are not going to spend any waste space in representing the 0 point; the next number will be represented as 0.25135 into 10 to the power 1.

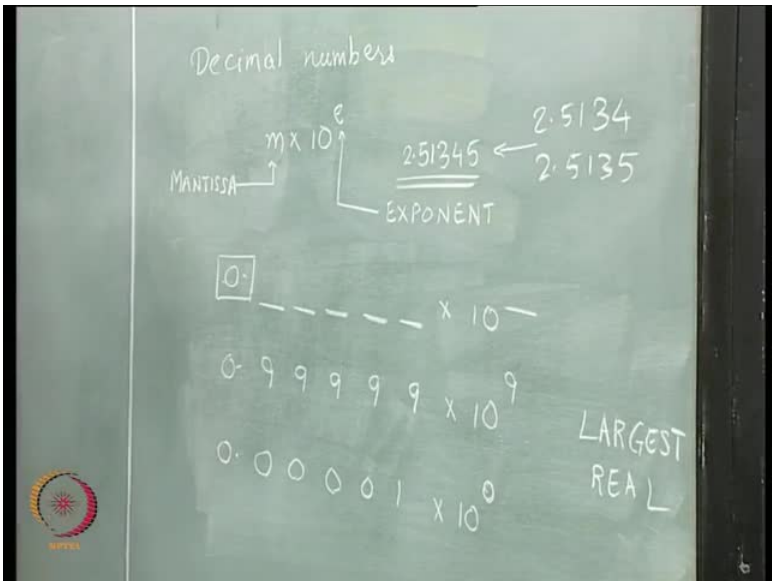
Now, let us say, we had a number between these two numbers; let us take that number as 2.1345; the question is, how do we represent 2.51345, is if we try to do that we will have 0 point 25134 and now we have run out of space to represent that particular number.

So, what is happening over here is, if we have said we are going to save five spaces to represent a mantissa and one space to represent the exponent, we will not be able to represent a number 2.51345. So, this you will either be able to represent that as, two point five one 0.25134 into 10 to the power 1 or you will be the representing this as two point five 0.25135 into 10 to the power 1, you will not be able to represent one more digit over here.

Now, what we will try to do in our decimal computer is, we will use what is known as chopping off; what we mean by this is, for example, if I give you a number say 2.51347 and I tell you that, I want you to just give me five significant digits for this particular number, you will give the number as 2.5135, why because you are rounding off this number; what a computer, what our decimal computer will do? It will just throw away this number, this number is just discarded off and any number between 2.5134 and 2.5135 will be represented as 2.5134 itself, because we are discarding all the numbers or chopping off all the numbers.

So, to summarize, if we have represented a decimal number through using five digit mantissa and one digit exponent, we can at most represent the number of the form 2.5134 or 2.5135; we cannot represent any number between those two numbers that is what is known as a finite precision.

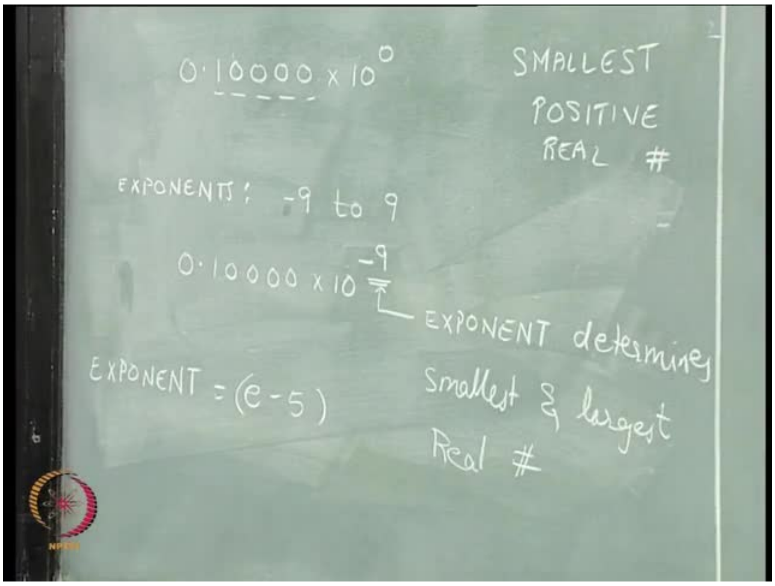
(Refer Slide Time: 33:57)



So, coming back to what we were talking about or where we got started off from; we got started off from talking about errors and we said they are two types of errors: truncation error and round off error. The round off error comes up, because a finite precision machine is chopping off the numbers beyond what it is capable or what you have asked it to store; you have asked to store a real number using only five decimal digits, as a result of this, anything that requires six decimal digit is going to be chopped off. So, what I want you to do is, just think about you have the representation 0.5 digit mantissa multiplied by 10 to the power 1 digit exponent. Think about what is the largest number, you can represent using this particular notation; and what is the smallest number, that you can represent using this particular notation.

So, the various possible answers that people had come up with the largest numbers, actually, I think most of the people got it right; the largest number that you can represent is going to be 0.9 9 9 9 9, because those are the largest numbers, each place holders can hold multiplied by 10 to the power 9. This is the largest real number that our floating point representation is going to hold. Now, what is going to be the smallest real number, the smallest real number is perhaps going to be 0.00001 multiplied by 10 to the power 0; that is the smallest real positive number that can be represented.

(Refer Slide Time: 35:40)



$0\cdot\underline{10000} \times 10^{0}$

SMALLEST POSITIVE REAL #

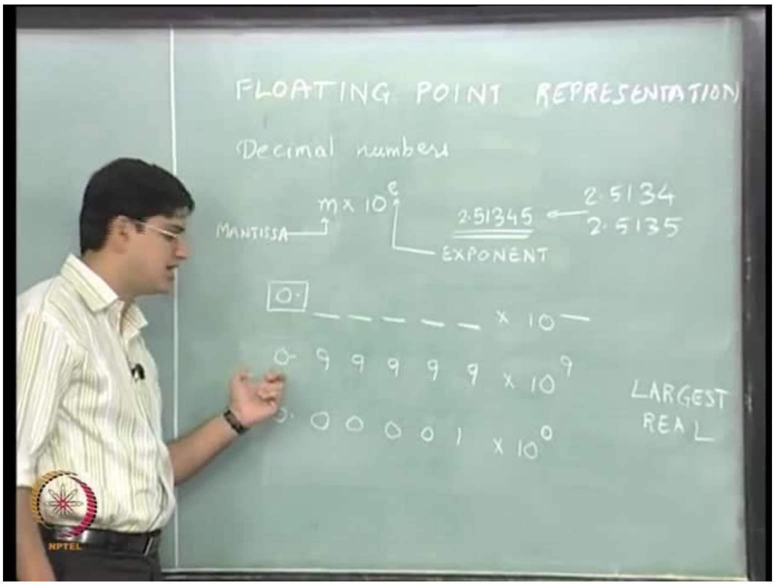
EXPONENTS : -9 to 9

$0\cdot 10000 \times 10^{-9}$

EXPONENT determines Smallest & largest Real #

EXPONENT = (e - 5)

(Refer Slide Time: 35:55)



If you wrote this as your answer, unfortunately your answer is incorrect, the reason why this is incorrect is remember what we said about the mantissa; the mantissa will lie between 0.1 and 1; keep in mind there is an equality sign over here, there is no equality sign over here; the largest number that is represented by the mantissa is 0.9 9 9 9 9 and not 1.0, 1.0 remember would be represented as 0.1 multiplied by 10 to the power 1; so, the mantissa lies between 0.1 less than and equal to mantissa less than 1. So, this particular number will not be represented in this way; this particular number has to be represented as 0.1000 multiplied by 10 to the power minus 4.

Now, the requirement we said with respect to the exponent is, we said we did not want to represent negative exponents over here. So, this particular number is not admissible in our scheme of things - in our decimal computer; so, the smallest number that is actually admissible in our decimal computer is going to be 0.10000 into 10 to the power 0, that is the smallest positive real number; so, the largest positive real number was 0.9 9 9 9 9 into 10 to the power 9; the smallest positive real number is 0.10000 into 10 to the power 0. Keep in mind that, we have are going to use all the five digits; so, that means, the number is not 0.1 into 10 to the power 0, the number is 0.10000 into 10 to the power 0.

Now, if you relax our assumption of only positive exponents and to say that even the negative exponents are admissible.

So, let us say that, the exponents that are possible will range from say minus 9 to plus 9; this is again for the argument sake and there would not be represent necessarily a representation in the decimal system, the reason is because we need one storage for this particular negative number and if you are going to use a decimal digit in order to do this storage, is going to be a waste of a lot of space.

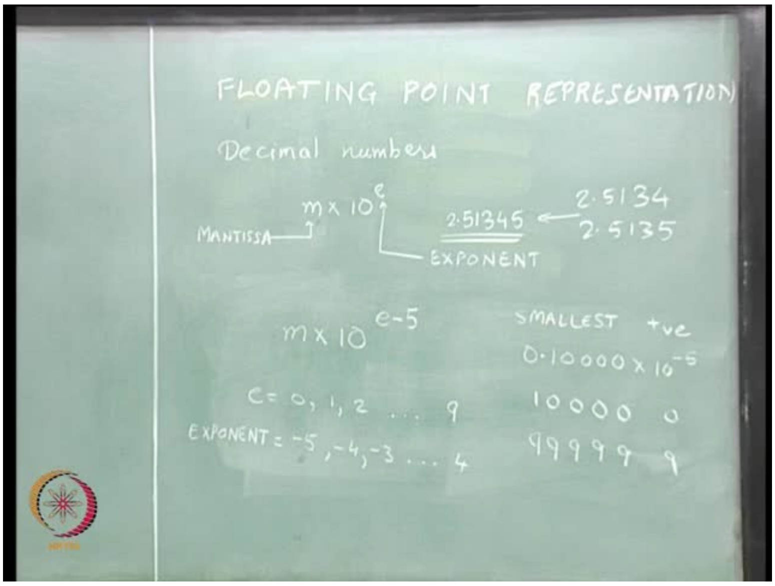
So, this is perhaps not something that we would want to do in the decimal system. But, let us say, for argument sake for now just that, the exponent that is allowed is from minus 9 to 9; in that particular case, the smallest positive real number, if the exponents were allowed from minus 9 to plus 9 is going to be 0.10000 into 10 to the power minus 9.

So, the crux of matter is that, the smallest and the largest real numbers are actually determined through what your exponent is; the exponent determines smallest and largest real numbers that can be represented using our floating point notation.

Now, as I said, the use of number minus 9 to plus 9 is very wasteful. Now, if you want to admit negative exponent also using a single digit, what can we possibly you do? Well, one thing that we can possibly do is, in order at admit positive and negative numbers also is the exponent, we will represent rather than saying e is the exponent, we will actually say exponent is actually e minus some constant; and let us use for this particular constant 10 divided by 2 equal to 5.

So, again from where we have gone, we have gone from trying to represent this number 2.5134 in our decimal computer, we then we talked about how the mantissa and the exponent is going to be represented; and then we talked about how the representation of the exponent can be wasteful. And now we are going to talk about how we can admit both the negative values of exponent and the positive values of exponent.

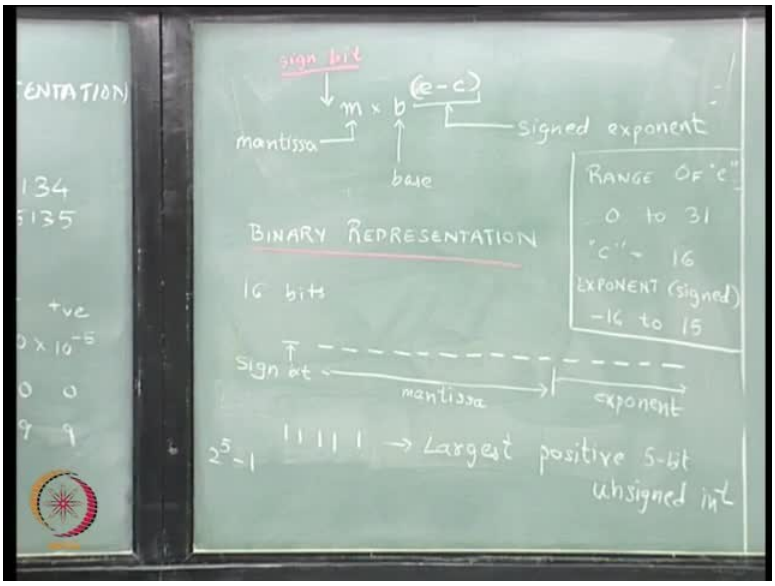
(Refer Slide Time: 41:02)



So, again, now what we have over here based on this particular redefinition, is instead of defining our number as m into 10 to the power e; if we were to use this particular definition we are going to represent this as m multiplied by 10 to the power e minus 5; so, when we have this particular representation, we have five digits to store the mantissa, one digit to store the exponent; the exponent e, the value of e, the admissible values of e are 0 1 2 and so on up to 9; as a result, the admissible values of exponent become minus 5 minus 4 minus 3 and so on up to plus 4; so, this revised definition of our floating point representation has the smallest positive number as 0.10000 multiplied by 10 to the power minus 5.

This with the 6 decimal digits, that we are going to use in our computer, is going to be represented as 100000, this is m, this is e and our representation is m multiplied by 10 to the power e minus 5; so, that is the smallest positive number; the largest positive number is going to be represented as 9 9 9 9 9 9; so, this represents 0.9 9 9 9 9 into 10 to the power 9 minus 5 or 10 to the power 4.

(Refer Slide Time: 43:22)



So, we have talked about the floating point representation. Now, we will talk about the floating point representation in binary based or any base for that matter, so rather than writing the number as m times e 10 to the power e, we will write it as m times b to the power capital E capital, E stands for the exponent or we can instead of this we will write it as e minus c.

So, over here, m is the mantissa, b is the base and this becomes our signed exponent. And in our binary representation that we use in our computers - in the binary representation what we will have is, in addition to this mantissa we will have a sign bit also for our binary and I will write this with the red chalk.

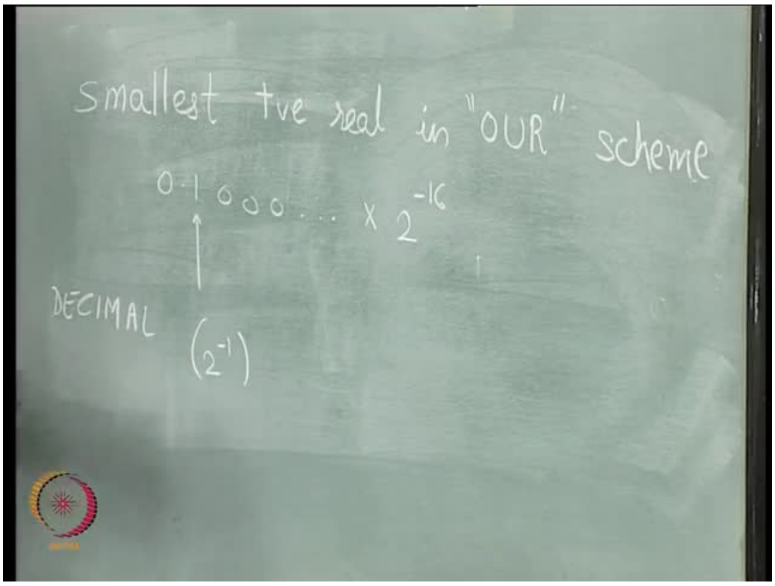
So, now how a binary representation of a flowing point number would be. So, let us say that, we are that we will have a total of 16 bits for representing a real number; in that particular case, the first bit is going to be the sign bit, that will represent the sign of the complete real number, whether the real number is positive or negative, will be given by the value of the sign bit 0 would mean, it is a positive number 1 would mean, it is a negative number, we will then out of the remaining 15. We will keep some of these digits, some of these bits for the mantissa and some of these bits for the exponent.

Let us say, again for argument sake, let us say that, we have kept 10 bits for the mantissa; so, 123456789 and 10 ,this will be kept for the mantissa, and the last five will be kept for the exponent; the exponent is an integer, so this the rules that we actually used for the integer numbers are the same rules that we will use for the exponent except, because we are representing this as e minus c, we will only worry about the positive representation of the numbers; so, if we are going to talk about positive integer, the largest positive integer in this particular case is going to be 11111.

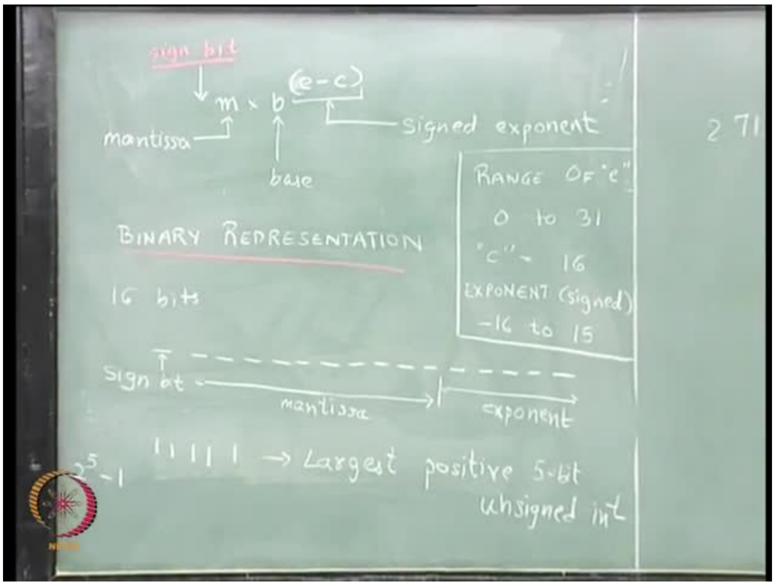
So, what I have written over here is, if we have five locations to store, then 11111 is the largest positive five bit unsigned integer; unsigned basically means, we are only admitting positive numbers; in this particular representation, there is no digit, no bit safe as a sign bit. So, this number is going to be 2 to the power 0 plus 2 to the power 1 plus 2 to the power 1 plus 2 to the power 2, 3, 4, which is equal to 2 to the power 5 minus 1. So, the range that e can take using this unsigned integer is from 0 which will would be 00000 to 11111, which is 2 to the power 5 minus 1 0 to 31, because this range goes from 0 to 31; what we will do is, we will choose c as 32 divided by 2 that is 16.

So, we will choose c in this particular representation as 16; so, with e going from 0 to 31 and with the value of c equal to 16, the exponent or rather I should be saying signed exponent, the signed exponent will go from minus 16 to plus 15; this is the range of exponent that we are going to get with this particular representation, if we have 16 bit floating point numbers with one sign bit 10 mantissa and 5 exponent; so, the smallest number that can be represented is going to be 0.100000000 multiplied by 10 to the power 2 to the power minus 16, because we have this as the base, the base is no longer 10 the base is b.

(Refer Slide Time: 49:39)



(Refer Slide Time: 49:57)



So, with this particular representation, why I emphasize the term because this is not a standard notation, I am just using it as pedagogical tool; in general, the real numbers are going to be represented as 32 bit or double precision numbers are going to be represented as 64 bit numbers; I

will come to that later, but in this particular representation, the smallest positive real number is going to be 0.1000 multiplied by 2 to the power, that is b to the power e minus c e minus c is going to be minus 16 2 to the power minus 16; when we represent this as, 1 the decimal equivalent is 2 to the power minus 1; this is 2 to the power minus 2 and so on; so, the decimal equivalent of this is 2 to the power minus 1 is what the mantissa gives us multiplied by 2 to the power of minus 16, that is what the exponent gives us.

So, the smallest positive real number is going to be determined through the exponent part of the representation.