

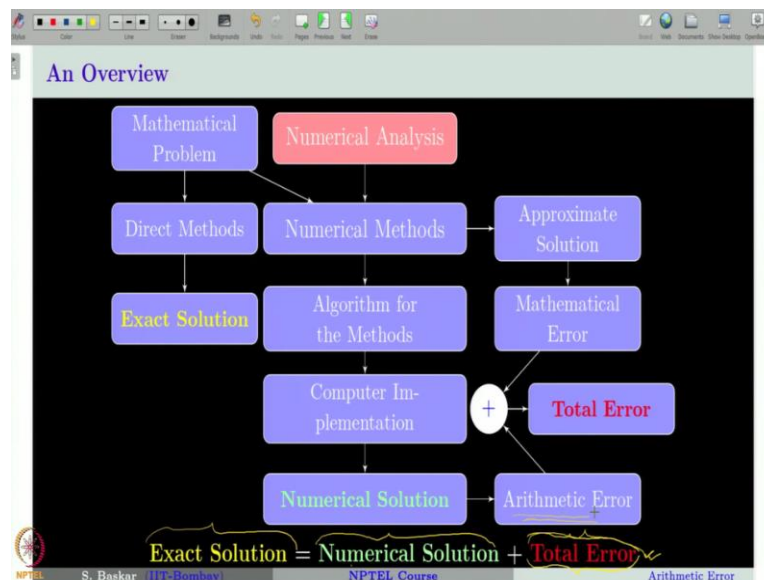
Numerical Analysis
Prof. S. Baskar
Department of Mathematics
Indian Institute of Technology – Bombay

Lecture – 04
Arithmetic Error: Floating-point Approximation

Hi in this lecture we will start a new chapter. This is the first chapter of our course. This is about the arithmetic errors. In this lecture we will first introduce what is meant by floating-point representation of a number on a computer and then we will see what is meant by floating-point approximation and then we will also introduce a notion called machine epsilon and define various errors.

Let us start our class with a brief overview of how the numerical methods are implemented on computers and how they give errors.

(Refer Slide Time: 01:05)



Let us start our discussion from the mathematical problem that we have in hand and we want to solve this mathematical problem obviously if we have a method that can give us exact solution that will be the highly desirable one because we have no error involved in our solution. Such methods are called direct methods, but as we spoke earlier not all mathematical problems can be solved to get exact solutions one needs to therefore go for an approximation.

One way to approximate is to go for numerical methods. The numerical methods can only give approximate solutions to our mathematical problem that shows that the method will obviously

involve a error when compared to the exact solutions of the problem. These are the errors which are coming due to the mathematical approximation of the problem and we call such errors as mathematical error in our course.

Once we devise the numerical methods then we will go to develop an algorithm based on the method and further we will be implementing that algorithm on a compute. Once we implement on the computer then the computer will generate numerical solution for our problem. Now at this level also we have one more error coming up due to the approximation of the numbers on a computer.

This is because computers cannot store numbers with their infinite precision due to their limited memory. Therefore, they in some sense truncate the number and only stores finitely many digits in the mantissa. This leads to a new type of error which is called the arithmetic error. Therefore, the numerical solution obtained on a computer will also involve the arithmetic error.

Remember the numerical method as such involves the error which is the mathematical error. Now the numerical solution obtained on the computer will therefore have mathematical error and arithmetic error both together is called the total error. Hence, our exact solution is nothing, but the numerical solution plus the total error. The problem is, in general we cannot obtain the total error exactly, if you can obtain then it is equivalent to getting the exact solution itself.

Often we do not have that luxury therefore this total error is not known to us. Of course, if we know exact solution then one can take the difference between exact solution and numerical solution and that will give you the total error, but that also is not possible because if we know exact solution why will we go for the numerical solution. Therefore, in practical applications we cannot get the total error.

However, we can get some idea about how the total error will behave in certain problems. For that we need to first understand how this arithmetic error will come into our calculation and how it goes from one step to the other step of a calculation and what kind of impact that it will give on the total error and this is what we are going to see in this chapter.

(Refer Slide Time: 05:45)

Floating-Point Representation

Let $\beta \in \mathbb{N}$ and $\beta \geq 2$.
 Any real number can be represented exactly in base β as

$$(-1)^s \times (.d_1 d_2 \dots d_n d_{n+1} \dots)_\beta \times \beta^e, \quad (1)$$

with

- $d_i \in \{0, 1, \dots, \beta - 1\}$, $d_1 \neq 0$ or $d_1 = d_2 = d_3 = \dots = 0$,
- $s = 0$ or 1 is called the **sign**,
- an appropriate integer e called the **exponent**,
- $(.d_1 d_2 \dots d_n d_{n+1} \dots)_\beta = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_n}{\beta^n} + \frac{d_{n+1}}{\beta^{n+1}} + \dots$ is a β -fraction called the **mantissa** and

The representation (1) of a real number is called the **floating-point representation**.

NPTEL S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Let us start our discussion with what is meant by floating-point representation of a real number. Assume that we are given a number β which is some natural number greater than or equal to 2 then any real number can be represented exactly in base β as $(-1)^s$. Remember if s is equal to 0 that will lead to a positive number and $s = 1$ will give us negative number into $(.d_1 d_2 \dots)_\beta \times \beta^e$.

You can keep in mind either $\beta = 10$ which give us a decimal representation or $\beta = 2$ which will give us the binary representation. All these are well known to us I am just trying to define that. The only new thing is that the way we represent the number and more importantly this d_i 's should be some integer between 0 to $\beta - 1$ with the main assumption that d_1 should not be equal to 0 or if d_1 has to be 0, then all other digits have to be 0 in which case we will be representing the number 0 only.

As I told s is called the sign and it takes the value either 0 or 1 depending on whether the given number is positive or negative, e is an appropriate integer such that this representation is a floating-point representation means d_1 should not be equal to 0. For instance if you want to represent the number -0.045 , then how you can represent it, you can write it as -1 to the power of 1 into point see you cannot write 045 because the first digit should not be 0 therefore you have to write 45 and the base is 10. I am just writing it in the decimal form into 10 to the power of -1 . Therefore, this e has to be appropriately chosen to get the floating-point representation of the number that you want to work with. Now, this part of the representation is called the mantissa. And that can be written like this and the representation 1 is called the floating-point representation, often β is also called the radix.

(Refer Slide Time: 09:08)

Error Analysis: Floating-Point Representation (contd.)

Example:

- When $\beta = 2$, the floating-point representation

$$(-1)^s \times (.d_1 d_2 \cdots d_n d_{n+1} \cdots)_2 \times 2^e$$
 is called the **binary** floating-point representation.
- When $\beta = 10$, the floating-point representation

$$(-1)^s \times (.d_1 d_2 \cdots d_n d_{n+1} \cdots)_{10} \times 10^e$$
 is called the **decimal** floating-point representation.

Throughout this course, we always take $\beta = 10$.

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

So, when $\beta = 2$ the floating-point representation is called the binary floating-point representation and when $\beta = 10$ the floating-point representation is called the decimal floating-point representation. Throughout this course we will mostly consider only $\beta = 10$.

(Refer Slide Time: 09:36)

Error Analysis: Floating-Point Approximation

Definition (n -Digit Floating-point Number)

Let $\beta \in \mathbb{N}$ and $\beta \geq 2$. An **n -digit floating-point number** in **base β** is of the form

$$(-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e$$

where

$$(.d_1 d_2 \cdots d_n)_\beta = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_n}{\beta^n}$$

with $d_i \in \{0, 1, \dots, \beta - 1\}$, $d_1 \neq 0$ or $d_2 = d_3 = \cdots = 0$, $s = 0$ or 1 , and an appropriate exponent e .

Handwritten notes on the right: $\frac{1}{3} = .333\dots$, $= .\underline{33\dots}3[3]$, ∞

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Next is the floating-point approximation. As I told earlier the computers cannot hold a number in its exact form especially when the number consist of infinitely many digits in the mantissa. Say, for instance, if you want to store the number $\frac{1}{3}$. We know that it is 0.333 and so on, if you want to store this number on a computer it can only store finitely many numbers and it truncates all the other digits after that number.

So, the number of digits in this mantissa will be decided based on the capacity of the computer. So, we will introduce the notion called floating-point approximation which we will call it as n -digit floating-point number to the base β . So, if we say that the number x is represented in n -digit floating-point form it means it has only n digits in its mantissa that is the idea. All other conditions remain the same.

(Refer Slide Time: 11:02)

Error Analysis: Floating-Point Approximation (contd.)

Example:
The following are examples of real numbers in the decimal floating point representation.

- The real number $x = 6.238$ can be represented as

$$6.238 = (-1)^0 \times 0.6238 \times 10^1,$$
 in which case, we have $s = 0$, $\beta = 10$, $e = 1$, $d_1 = 6$, $d_2 = 2$, $d_3 = 3$ and $d_4 = 8$.
- The real number $x = -0.0014$ can be represented in the decimal floating-point representation as

$$x = (-1)^1 \times 0.14 \times 10^{-2},$$
 Here $s = 1$, $\beta = 10$, $e = -2$, $d_1 = 1$ and $d_2 = 4$.

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Let us give few examples if you want to represent the real number 6.238 that can be written as $(-1)^0$ because it is a positive number and then we have the mantissa like this and then to make it 6.238 we have to multiply it with it $(10)^1$. Therefore, we are representing this number in the decimal form that is base 10 and therefore here $s = 0$, β is 10, e is 1 and d_1 is 6 and so on.

Similarly, if you want to represent the number $x = -0.0014$ then you have $s = 1$ and you have d_1, d_2 as 1 and 4 with exponent e as -2 .

(Refer Slide Time: 12:05)

Error Analysis: Floating-Point Approximation (contd.)

Remark:
 Note that there are **only finite number of digits** in the n -digit floating-point representation.

But a real number **may have infinitely many digits** and therefore infinitely many digits in mantissa.

For instance,

$$\frac{1}{3} = 0.33333\cdots = (-1)^0 \times (0.33333\cdots)_{10} \times 10^0.$$

Therefore, the representation

$$(-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e \quad +$$

is (in general) **only an approximation to a real number**.

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Note that there are only finitely many digits in the n -digit floating-point representation, but a real number in general can have many digits as we have seen, for instance, in $\frac{1}{3}$ we have infinitely many 3's in the mantissa. Therefore, in general a n -digit representation of a real number is only an approximate representation.

(Refer Slide Time: 12:33)

Error Analysis: Underflow and Overflow of Memory

- When the value of the exponent e in a floating-point number exceeds the maximum limit of the memory, we encounter the **overflow** of memory.
- When this value goes below the minimum of the range, then we encounter **underflow**.

Thus, for a given computing device, there are integers m and M such that the exponent e is limited to a range

$$m \leq e \leq M.$$

During the calculation,

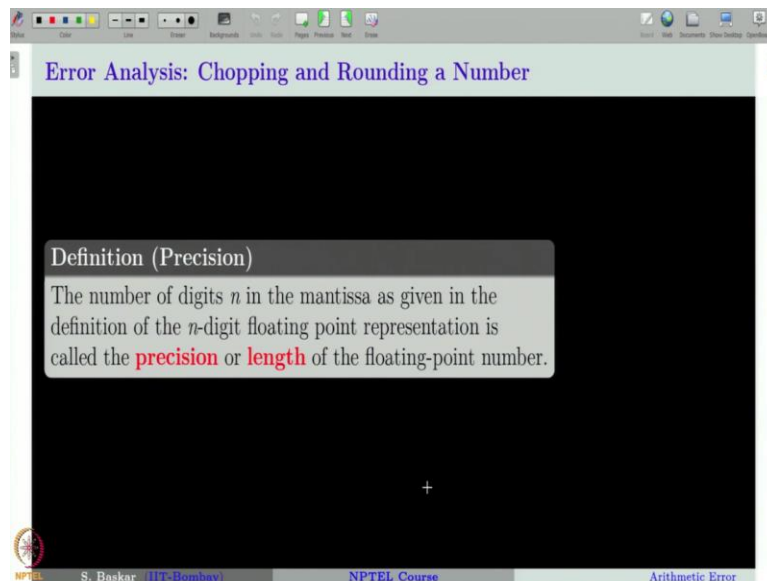
- if some computed number has an exponent $e > M$ then we say, the memory **overflow** occurs and
- if $e < m$, we say the memory **underflow** occurs.

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Next we will try to understand what is meant by underflow and overflow of memory when the value of the exponent in a floating-point representation exceeds the maximum limit of the memory then we will encounter a overflow of memory. On the other hand, if the exponent e goes below minimum range then we encounter underflow of memory. Therefore, in any computer if you take there will be a minimum limit that it can hold and maximum number that it can hold. That depends on the exponent, if the exponent is less than or minimum limit then

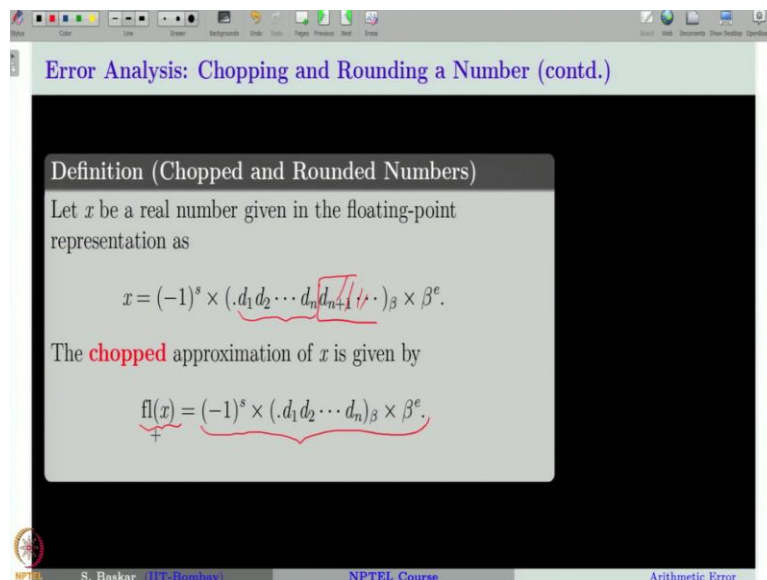
the number will be just considered as 0 and if the exponent exceeds its maximum limit say capital M then that number will be considered as infinity.

(Refer Slide Time: 13:36)



The number of digits n in the mantissa as given in the definition of the n -digit floating-point representation is called the precision or length of the floating-point number.

(Refer Slide Time: 13:50)



Now given that our computer can hold only finitely many digits in the mantissa how is it going to truncate a number which has more digits than it can hold. Well, the computers have very sophisticated algorithm of rounding a number, but for the sake of understanding here we will only consider two types of truncating number, one is chopping and another one is rounding a number and these are in their simplest form just for the sake of mathematical understanding.

Let us introduce first what is meant by chopping a number. Suppose, you are given a real number and its floating-point representation is given like this then chopping this number to n digit means you simply forget the digits from $(n + 1)$ th position onwards only the digits from 1 to n and therefore this representation that is the n -digit floating-point representation just by chopping this part is called the chopping approximation.

We will use the notation $\text{fl}(x)$ for any floating-point approximation of a number whether it is chopping or rounding. Let us next see what is meant by rounding a number.

(Refer Slide Time: 15:30)

Error Analysis: Chopping and Rounding a Number (contd.)

Definition (Chopped and Rounded Numbers)

Let x be a real number given in the floating-point representation as

$$x = (-1)^s \times (.d_1 d_2 \cdots d_n d_{n+1} \cdots)_\beta \times \beta^e.$$

The **rounded** approximation of x is given by

$$\text{fl}(x) = \begin{cases} (-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e & , 0 \leq d_{n+1} < \frac{\beta}{2} \\ (-1)^s \times (.d_1 d_2 \cdots (d_n + 1))_\beta \times \beta^e & , \frac{\beta}{2} \leq d_{n+1} < \beta \end{cases}$$

where $(-1)^s \times (.d_1 d_2 \cdots (d_n + 1))_\beta \times \beta^e :=$

$$(-1)^s \times \left((.d_1 d_2 \cdots d_n)_\beta + (. \underbrace{00 \cdots 0}_{{(n-1)\text{-times}} 1})_\beta \right) \times \beta^e.$$

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Again you are given a real number which is written in this floating-point form then rounding means you take the $(n + 1)$ th digit if $n + 1$ th digit is greater than $\frac{\beta}{2}$ of course, but if it is less than $\frac{\beta}{2}$ then we will just follow the chopping idea whereas the $(n + 1)$ th digit d_{n+1} if it lies between $\frac{\beta}{2}$ and β then we will add 1 to the previous digit. So, that is called the rounding of a number. We have more sophisticated algorithms for this, but this is the simplest idea of rounding a number and we will consider only in this form in our course.

(Refer Slide Time: 16:29)

Error Analysis: Arithmetic Using n -Digit Rounding and Chopping

Procedure of performing arithmetic operations using n -digit rounding:

Let \odot denote any one of the basic arithmetic operations '+', '-', '×' and '÷'.

Let x and y be real numbers.

The process of computing $x \odot y$ using n -digit rounding is as follows.

Step 1

Get the n -digit floating-point approximation $fl(x)$ and $fl(y)$ of the numbers x and y , respectively.

Handwritten note: $x, y \rightarrow fl(x) fl(y)$

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

With this, let us try to understand the procedure of performing an arithmetic operation using n -digit rounding. Remember we have four basic arithmetic operations '+', '-', '×' and '÷'. The procedure that we are going to give here holds for any of these four operations. Therefore, I will not restrict the notation to any one of this. I will take a symbol \odot which represents any of this four operations.

Let us see how to perform an arithmetic operation between two numbers x and y that is we want to compute this number $x \odot y$. Remember \odot can be either '+', '-', '×' or '÷'. So, what you have to first do is to get the floating-point approximation of x and the floating-point approximation of y . What we are going to do here is simply to understand how a computer does an arithmetic operation like this.

Suppose, you give x and y to a computer. It first takes x and write its floating-point approximation as per its capacity, that is, it will truncate the digits which it cannot hold in its memory, it is instructed to do, mostly it is instructed to do a rounding approximation and then it also takes the number y and writes $fl(y)$. So, it takes the number x and y and write in its memory only its approximate value as per the rounding algorithm and then it will go to perform the arithmetic operation.

(Refer Slide Time: 18:44)

Error Analysis: Arithmetic Using n -Digit Rounding and Chopping

Procedure of performing arithmetic operations using n -digit rounding:

Let \odot denote any one of the basic arithmetic operations '+', '-', '×' and '÷'.

Let x and y be real numbers.

The process of computing $x \odot y$ using n -digit rounding is as follows.

Step 2

Perform the calculation $\text{fl}(x) \odot \text{fl}(y)$ using exact arithmetic.

So, therefore in the step 2 the computer will do the arithmetic operation of the approximated value of x and the approximated value of y .

(Refer Slide Time: 18:55)

Error Analysis: Arithmetic Using n -Digit Rounding and Chopping

Procedure of performing arithmetic operations using n -digit rounding:

Let \odot denote any one of the basic arithmetic operations '+', '-', '×' and '÷'.

Let x and y be real numbers.

The process of computing $x \odot y$ using n -digit rounding is as follows.

Step 3

Get the n -digit floating-point approximation $\text{fl}(\text{fl}(x) \odot \text{fl}(y))$ of $\text{fl}(x) \odot \text{fl}(y)$.

Handwritten notes: x, y, z and $(x+y)z$ with a plus sign. An arrow points from the expression to the text "final answer".

And then it will take the final value that is $\text{fl}(x) \odot \text{fl}(y)$ and then makes a floating-point approximation of that number and that is what the final answer that a computer will give to us if you want it to perform this calculation. Therefore, if a human being performs this calculation the value maybe different from the value that is given by the computer.

Now, if we have some complicated expressions, like I have x , y and z and I want to find $(x + y)z$ then you can just generalize this idea and you can see how a computer will find the value of this expression. How will it find? First it will find $\text{fl}(x)$, $\text{fl}(y)$ and $\text{fl}(z)$ then it will add $\text{fl}(x) + \text{fl}(y)$ and then it again makes a floating-point approximation of that sum and then it will

multiply that floating-point approximation of the sum with $\text{fl}(z)$. And then the final answer will further be approximated using the floating-point algorithm. So, that is the idea of performing any expression.

(Refer Slide Time: 20:34)

Error Analysis: Arithmetic Using n -Digit Rounding and Chopping (contd.)

Example:
 Consider the function $f(x) = x(\sqrt{x+1} - \sqrt{x})$.
 Let us evaluate $f(100000)$ using a **six-digit rounding**.

- $f(100000) = 100000 (\sqrt{100001} - \sqrt{100000})$.
- $\sqrt{100001} \approx 316.229347 = 0.316229347 \times 10^3$.

The 6-digit rounded approximation of 0.316229347×10^3 is

$$0.316229 \times 10^3.$$

$$\Rightarrow \text{fl}(\sqrt{100001}) = 0.316229 \times 10^3.$$

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Let us take this example. You have $f(x) = x(\sqrt{x+1} - \sqrt{x})$. Let us say we want to compute the value of this function at the point $x = 100000$ using 6-digit rounding. How will we do? Of course, we have to find this value for that first we have to round this number to 6 digits anyway it has only 6 digits. Therefore, you can just plug in that and take the square root and the square root is coming to be like this, you write it in the floating-point form and that comes to be like this.

Of course, you have to put a $(-1)^0$, but I have just avoided putting it here then the 6- point rounding approximation of that number happens to be this. So, what you are doing is you are keeping up to 6 digits 1, 2, 3, 4, 5, 6 and then 347 you have just truncated since the leading number is 3 it just goes like a chopping approximation only. Therefore, the floating-point approximation of $\sqrt{100001}$ is this.

(Refer Slide Time: 22:03)

Error Analysis: Arithmetic Using n -Digit Rounding and Chopping (contd.)

Example:

- Similarly, $\text{fl}(\sqrt{100000}) = 0.316228 \times 10^3$.
- Therefore, $\text{fl}(\text{fl}(\sqrt{100001}) - \text{fl}(\sqrt{100000})) = 0.1 \times 10^{-2}$.
- Finally, we have

$$\begin{aligned} \text{fl}(f(100000)) &= \text{fl}(100000) \times (0.1 \times 10^{-2}) \\ &= (0.1 \times 10^6) \times (0.1 \times 10^{-2}) = 100. \end{aligned}$$

Using six-digit **chopping**, the value of $\text{fl}(f(100000))$ is **200**.

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Now you go to do the next step that is you get the floating-point approximation of this number minus this and that happens to be 0.1×10^{-2} . Finally, we have floating-point approximation of the function value at 100000 which is what we want to find is given like this and that happens to be 100. So, remember we have done this calculation with 6-digit rounding at every step of the calculation. And we got the value 100.

You can see that instead of doing rounding if you do the chopping approximation then you will get the value 200 here. But now what is the exact value? Well, we will see that exact value is entirely different from this. It is something like 158 or so. Therefore, you can see that the calculation using 6-digit rounding or chopping has magnified the error to a greater extent.

(Refer Slide Time: 23:25)

Error Analysis: Machine Epsilon

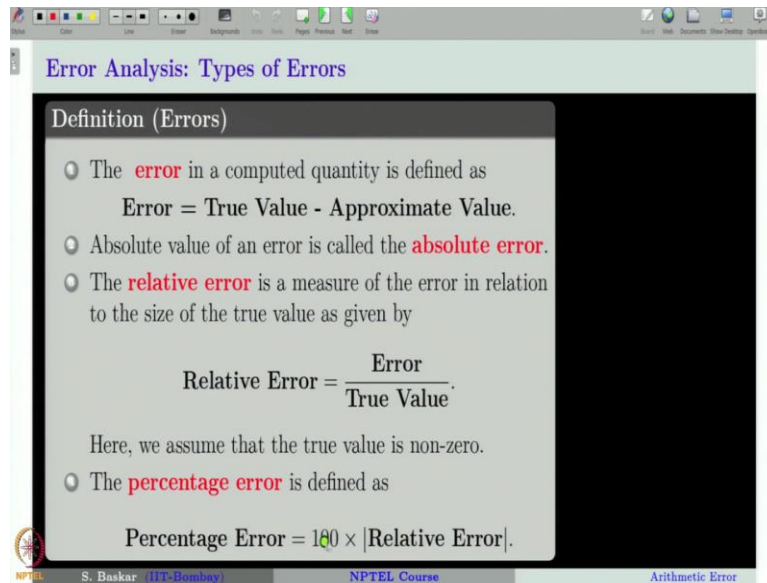
Definition (Machine Epsilon)

The **machine epsilon** of a computer is the smallest positive real number δ such that $\text{fl}(1 + \delta) > 1$. Thus, for any real number $0 < \hat{\delta} < \delta$, we have $\text{fl}(1 + \hat{\delta}) = 1$, and $1 + \hat{\delta}$ and 1 are identical within the computer's arithmetic.

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Let us see what is meant by machine epsilon? Machine epsilon of a computer is the smallest positive real number δ such that $\text{fl}(1 + \delta) > 1$. Thus, for any new number which is less than δ we will have the floating-point approximation of that number to be 1 only and $1 + \hat{\delta}$ and 1 are identical within the computer arithmetic. So, such a number is called the machine epsilon.

(Refer Slide Time: 24:10)



Error Analysis: Types of Errors

Definition (Errors)

- The **error** in a computed quantity is defined as
$$\text{Error} = \text{True Value} - \text{Approximate Value}.$$
- Absolute value of an error is called the **absolute error**.
- The **relative error** is a measure of the error in relation to the size of the true value as given by
$$\text{Relative Error} = \frac{\text{Error}}{\text{True Value}}.$$

Here, we assume that the true value is non-zero.

- The **percentage error** is defined as
$$\text{Percentage Error} = 100 \times |\text{Relative Error}|.$$

S. Baskar (IIT-Bombay) NPTEL Course Arithmetic Error

Let us quickly go through the definition of error. We all know what is mean by error. Error is nothing but the true value minus the approximate value. By absolute error we simply mean the absolute value of this error and next the important concept is the relative error. Relative error is nothing but the error divided by the true value and that will give us in fact the percentage error which can be obtained by multiplying the relative error by 100.

Of course, the absolute value of the relative error into 100 and that is what we call as the percentage error.

(Refer Slide Time: 25:00)

The image is a screenshot of a presentation slide. The title bar at the top reads "Error Analysis: Types of Errors (contd.)". The main content area has a black background with white text. It starts with a "Remark:" followed by a sentence: "Let x_A denote the approximation to the real number x . We use the following notations:". Below this, three mathematical definitions are listed:
1. $E(x_A) := \text{Error}(x_A) = x - x_A$.
2. $E_a(x_A) := \text{Absolute Error}(x_A) = |E(x_A)|$.
3. $E_r(x_A) := \text{Relative Error}(x_A) = \frac{E(x_A)}{x}, x \neq 0$.
The slide footer contains the NPTEL logo, the name "S. Baskar (IIT-Bombay)", the text "NPTEL Course", and "Arithmetic Error".

We also use some notations, $E(x_A)$ if we say it means the error involved in the approximate value x_A when compared to the true value x and that is given by $x - x_A$ and absolute error we often denote by $E_a(x_A)$ that is defined as $|x - x_A|$. For the relative error we often use the notation $E_r(x_A)$ and that is nothing but the error in x_A when compared to x divided by the true value x .

Of course, x should be not equal to 0. With this, we will now go into see what is meant by significant digits and what is the danger in losing this significant digits in our calculation. Thank you for your attention.