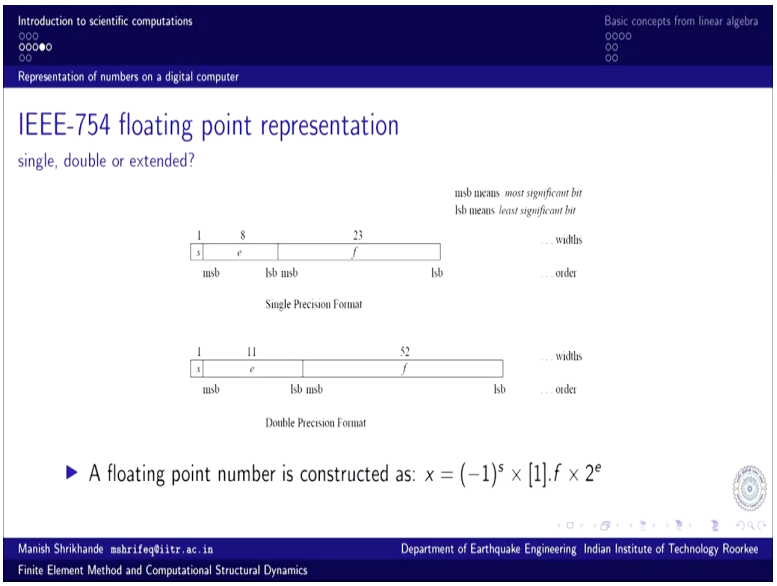
Finite Element Method and Computational Structural Dynamics
Prof. Manish Shrikhande
Department of Earthquake Engineering
Indian Institute of Technology, Roorkee

Lecture - 02
Introduction to Scientific Computations-II

Hello.

(Refer Slide Time: 00:31)



So, continuing with our discussion on floating point representation; so there are two standard representation, earlier IEEE-754 that is the floating point representation. And now all digital computers, all vendor and any make of digital computer comply with the IEEE-754 floating point representation. And this representation came into existence in late 80s, before that it was complete chaos; different computer vendors had their own representation of floating point and a code which gave some result on let us say IBM machine would give entirely different results let us say on wax machine. And it was a complete chaotic system, chaotic arrangement and lot of time and effort was wasted into porting of the code from one platform to another platform. And things became stabilized once a IEEE-754 floating point standard was adopted and all computer manufacturers started adhering to this standard for representing floating point numbers and operations.

So, there are two basic formats, one is a single precision and second one is double precision and there is also one more a format that is extended precision, that goes beyond double precision, but that is rarely ever used. So, double precision is more than enough for all scientific computations.

What we do we mean by single precision and double precision?. A single precision represents about an accuracy up to 7 places of decimal in a single precision representation. And in double precision, the accuracy of computation is a retained about 14 places of decimal. So, anything beyond 14th place of decimal is just garbage. So, there may be numbers, but there is no significance to those numbers.

Now, how it is done? Iinternal representation is of 32 bits. So, one floating point number on a digital computer occupies 32 bits and those 32 bits are arranged like you see here; we count from left side and move to the right hand side. So, the leftmost, so that is called the most significant bit, that is referred to as the sign bit. So, one bit, the most significant bit is reserved for the sign bit; because we need to have a representation of both positive numbers and negative numbers. So, we adopt that sign and modulus form. So, the most significant bit, the most left hand side a bit value is written as the sign bit. The next 8 bits are the exponent field in the biased representation; remember I talked about the shifting the zero line. So, shifting of origin, so that kind of representation is used for exponent field, that allows us to have negative exponents as well as positive exponents. So, we can have a very large numbers as well as very large small numbers. And then subsequently remaining 23 bits are the fractional bit; fractional bit of the mantissa, so a after the binary separator. So, the fractional point representing the coefficients of negative powers of 2, ie, $b_1 \times 2^{-1}$, $b_1 \times 2^{-2}$……. is the representation of single precision format.

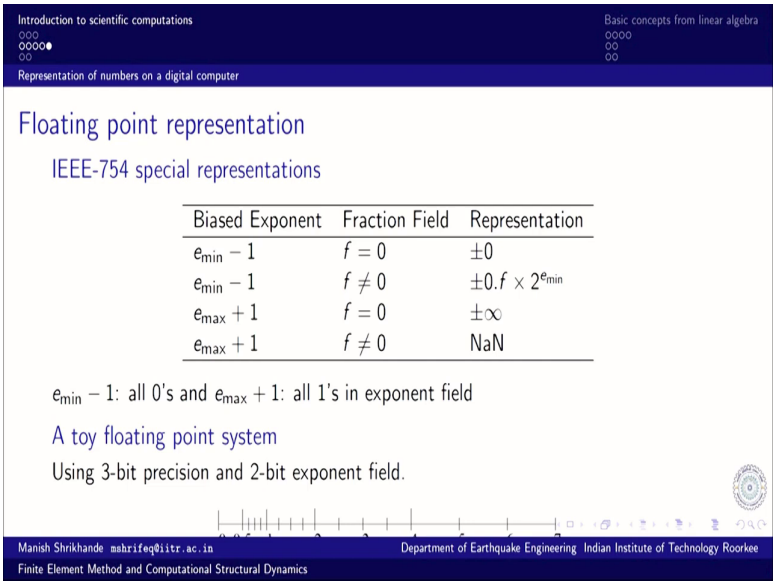
Double precision, it is structurally it is similar, except that the field bits are a more, sign bit of course you do not need anything more than 1 bit; but exponent field instead of 8 bits, it is 11 bits and the fractional field which is what controls the precision, that is now 52 bit long. So, that gives us the basic representation and a floating point number is constructed. Let us say x is a floating point number, I declare it as x as a float. Then that number is interpreted as -1^S , wheres is the sign bit. So, if s is equal to 0 that represents 1; because minus 1 raised to the

power 0 is 1. And if s is 1, then it is a negative number; it becomes -1^1 is -1 and the number becomes negative.

Then we have a hidden bit 1 as I said, we always assume that whole part of 1 is always there. So, that 1 is a hidden bit is represented in the square bracket and it is followed by the fractional part. So, [1].f. is the fractional part of the mantissa and that is of course multiplied by 2^e ; e is the exponent, whatever is the integer representation in the exponent field, so that gives us the magnitude, the scale of the number. So, we can have a very large number, we can have a very small number.

Now, you can see that it is a binary, it is a discrete representation. So, binary field I mean 0 0 0 0 0 1 0 1 0 0 1 1 etcetera, so that these are all discrete numbers, right.

(Refer Slide Time: 06:47)



So, what we have here is, as I said 1 is the hidden bit and biased exponent. So, we have also a spatial representation; if exponent field is minimum exponent minus 1, so that is the all representations as a 0 fractional representation, the exponent field is all 0. Then and the exponent field is all 0, all exponent bits are 0 and the fraction field is also 0; then that representation is taken to represent 0 and it can be plus or minus depending on the sign bit whatever it is. So, that is a assigned value. So, this representation is assigned a value of 0;

there is no way we can ever arrive at this number by computation, I again reiterate this sentence.

And if the exponent field is all 0 and fractional field is non zero, then this spatial representation is considered as 0.f x multiplied $2^{e\text{-}min}$. And that is called subnormal representation and it is an attempt to fill up the gap between assigned value of 0 and the smallest standard normal representation. So, this sub normal representation that is an attempt to fill up this gap with some numbers; I mean we have a representation, why waste it, use it some way, but its precision is reduced. So, you can have different levels of precision, it is not really all 24 bit precision that is available in standard representation. And then if we have exponent field of all 8 bits or 11 bits are all 1 and the fractional field is 0. Then that is taken to represent plus or minus infinity and again depending on sign bit.

And then if the fractional field is non zero for all exponent field of unity; then that is taken to represent not a number. So, not a number is used to flag numerical operations those are not defined; like trying to compute logarithm of a negative number. So, during a process of computation, if you ever encounter a situation; you are taking logarithm of a variable and that variable happens to store a negative number, that result would be labelled as not a number, because this operation is not defined. So, that is the a standard representation IEEE 754. These are very helpful in understanding the results that we get during the process of our computations and makes sense of something if something goes wrong.

So, just a example of a toy number system. So, instead of 23 bit precision, I just use 3 bit precision and an exponent of a 2 bit system and this is what the number system looks like. So, 0 is the assigned number and then we have 0.1 and 0.5 and you can see the important thing is, you can represent here exactly between powers of 2.

So, 1.000 and multiplied by 2^{-1}. So, that is the number 0.5 that you have and then the next number that we have is 1, that is hidden bit; then 1.001 multiplied by 2^{-1}. So, that adds one more increment and so on until we go to a number 1 and these are the different range. And you can see that between two integer powers of 2; so 2.5 is 2^{-1}, 1 is 2^{0}, we have exactly 3 numbers, 3 distinct numbers that are represented. And same happens between another next powers of 2; so from 2^{0} to 2^{1}, we have 3 numbers in between. And then from 2^{1}; so that is 2

to 2^2, that is 4, we again have 3 numbers. So, that is what the problem is, we always have fixed set of numbers that are available between the numbers, between two integer powers of 2.

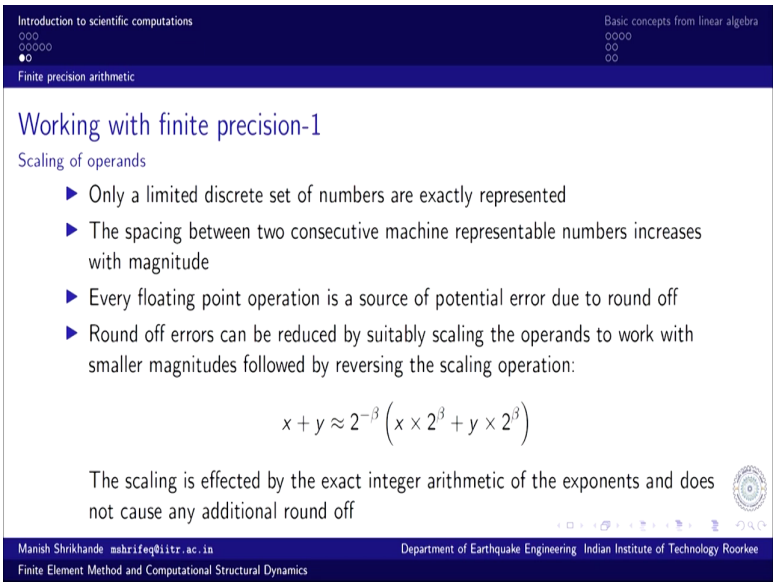
And more importantly you should try to appreciate this that the numbers they are the separation between the numbers, the 3 numbers that are available they are closely spaced for smaller powers of 2. As the power of 2 increases, the separation between the consecutive numbers that can be represented increases. What is the significance of this representation? Anything in between the numbers that are labelled here that does not exist on the computer. So, this is what I meant; how many numbers can I fit between two consecutive numbers? Ideally in a real number system, there is infinite set that is available; but on a digital computer, it is not so. I only have a finite set of numbers that I can pack in between two consecutive numbers. And those are discrete set of numbers, those are a referred to as machine represented numbers and anything between them is black hole. The machine does not know anything in between, anything between other than these discrete representation. So, if the number of the actual result of computation happens to be something in between these numbers that are represented; then it will be rounded off, the result will be rounded off depending on what is the round off strategy. And that round off error, the magnitude of round off error depends on what is the magnitude of the number that we are working on. If it is a large magnitude number, then the chances of round off error are also very large; because the distance between two consecutive numbers is very large.

And therefore, it is important to scale the exponents field; see scaling the numbers is based on manipulation of the exponent and that is an integer operation, integer operation (addition, subtraction) is an exact operation as long as it does not overflow. So, there is nothing involved in that in exponent manipulation. So, I can actually reduce the round off error if I choose to operate, instead of operating at this range that is the large error which can have; if result of computation is somewhere here, it will be either rounded off to this number or it will be rounded off to this number depending on what is the round off strategy in force. So, either case it is a large round off error; but if I can somehow scale it back to this range, then the round off error is that much smaller. And after computation, I can again scale it back to

original magnitude whatever was required and that is what is important in understanding this floating-point operation.

The exact real numbers, we do not have infinite set available and there are only a finite set of numbers that can be represented on computers. And the spacing between these two numbers, two consecutive numbers that are represented on the computer, they are actually logarithmic spacing. So, with magnitude this spacing between two consecutive numbers also keeps on increasing. And therefore, in order to reduce the round off error, in order to keep the round off errors in check; it is important to scale the operations, to scale the numbers before performing any numerical operations. And this, this operation of scaling is very crucial and also the operation how do we sequence our numerical operations. Sometimes just the sequence also makes a lot of difference; whether I should add to numbers first or whether I should perform the subtraction operation first, that also makes a lot of difference in the ultimate result of calculation. So, once we understand that what are the sources of errors. Because of this very peculiar system of a representing real numbers on computers, it will help us to define on device algorithms for scientific computation which will be much more reliable and robust.

(Refer Slide Time: 17:09)



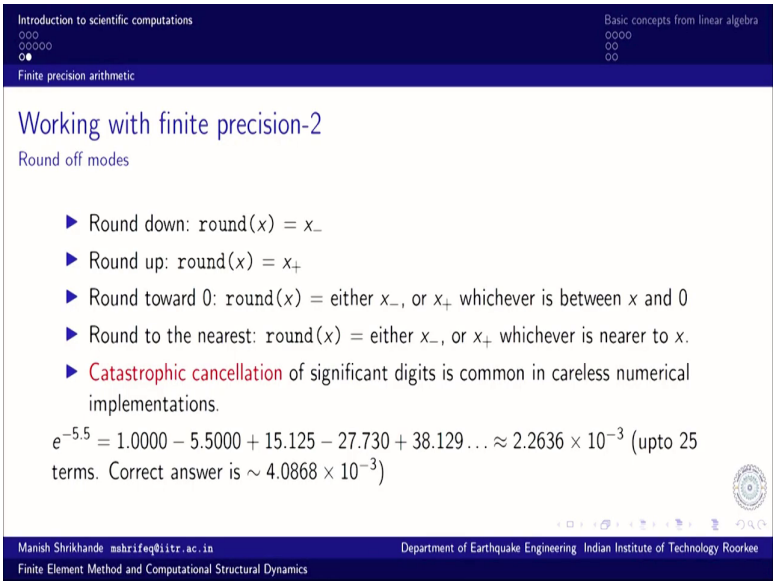
So, as I said, we need to scale how do we work with the finite precision. As I said, only a set of limited discrete set of numbers are exactly represented on the machine. And this spacing

between two consecutive machine numbers increases with magnitude. And every floating-point operation, the chance; I mean you can imagine there are infinite possibilities and the chance of probability of getting a number which is exactly represented on the computer is almost 0. So, every floating point operation that I perform on a digital computer is a possible source of error, it is a potential error due to round off. And if I use the result of floating-point operation again in subsequent computation, it is compounding error onto another already existing errors and that keeps on happening.

But as I said earlier the scaling is important; the round off errors can be reduced by suitably scaling the operands to work with the smaller magnitude. So, I scale the numbers to a smaller magnitude and do they perform the operations, so that the round off error is happening at a range in the range where the density of numbers is very large, the numbers are packed closer together. Then the round off error is very small and then the results can be a reverted back. So, just as in this case any number x plus y, two numbers addition of two numbers x and y it can be represented as; I can scale x by a factor 2^{β}. So, this again as you can see, it will only affect the exponent part of x and that is an integer exact arithmetic. And similarly, y can be scaled by the same factor 2^{β}. So that x and y both of them are scaled back to a number which is small in magnitude and the addition can then be performed. And subsequently the result can be again scaled back by negating the scaling. So, the result is again multiplied by $2^{-\beta}$. So, that will negate this a scaling operation.

So, since this scaling is performed in exact arithmetic, integer arithmetic of the exponent, so no loss happens there as long as it is not operating on the boundary of the integer range. So, this does not cause any additional round off and we can keep the round off error under control. So, this is one of the standard tricks that is employed. So, if you ever see any scientific computation code and then see the scaling and balancing operation. Then you may realize that you may relate to this a particular a aspect that this is done to minimize to keep the round off error in check.

(Refer Slide Time: 20:38)



So, there are different round off modes. So, it is a the round off is for round down. So, 2.4 is rounded off to 2, something like that and round up is 2.4 is rounded up to the 3, so that will be round up. So, round towards 0, so that is depending on whichever the sign of the number; if it is a negative number, then it is rounded to a larger number and if it is a positive number, then it is rounded to the smaller number. So, essentially whichever is the closest number between x and 0 and round to the nearest that is whichever is the nearest neighbor. So, that would be the round off strategy; most of the time we adopt a round to the nearest in most of the operations unless otherwise specified.

Now, as I said the sequence of operations, arithmetic operations is very important and that is what leads to what is known as catastrophic cancellation of significant digits. And this is very common in careless implementation of a numerical computation; if we are not careful enough, our results may be corrupted beyond redemption.

A simple example a just to drive home the point; the series expansion of e^x. So, if x is just some number -5.5, so it would be just $1 + x + x^2 + (x^3/3!)$ so on. So, that is a series expansion of e to the power x. So, if I calculate this again retaining these 5 places of significant digits, 5 significant digits in the calculation. So, if I add this and expanded up to 25 terms; then result a begins to convert somewhat up to this number. So, 2.2636 multiplied by 10 raise to the power minus 3, you can check that out. And you can also check that out the

calculator result would be about 4.08 multiplied by 10 to the power minus 3, a difference of almost 100 percent. Now, calculator also does the calculation by using the series expansion. So what is going on here, what is happening? So, the problem is sequential alternate addition and subtraction.

If you see, if I subtract two large numbers; I mean that is what you will see here as the series term go in the series terms, higher or higher terms, you will see that subtraction of very large numbers is happening. Even in this case you can see that a -27.7 and that is added to 38.129. So, two large numbers and then subsequent numbers are even larger. So, large numbers when they are subtracted; for example, if I subtract 9998 from 9999. So, four 9's minus 9998 and the result is going to be 0001. What is the implication? I had two numbers which were having four significant digits and the subtraction of those two 4 significant digits has resulted into a number, which has only one significant digit and that is a loss of precision. And that is what we call as catastrophic cancellation of significant digits. If two numbers of similar magnitude are subtracted that leads to a massive loss of significant digits andour computation can never recover from that error and that is why it is very important to guard against this kind of error.

So, what is the solution to this? How do I, I mean this is something that we do and all all approximate calculations are based on a series expansion. So, if this kind of thing happens, what is the correct way of doing it? The solution is very simple. I can write $e^{-5.5}$ as $1 / (e^{-5.5})$. So, I compute e raise to the power 5.5, so that would not be any alternate change of signs here, it will all be addition. So, 1 + 5.5 +15.125 + 27.730 + 38.129 and so on; I get this number, this converges this series converges very quickly and then I just take the inverse of that 1 over this number. And then you can see that the series converges very quickly; I do not really need to go up to 25 terms or 30 terms or 50 terms, it converges very quickly. And then I can naturally get 1 over that number and that actually $1 / (e^{-5.5})$ converges very quickly and you can get the result and avoid catastrophic cancellation.

So, the key idea at avoiding catastrophic cancellation is reorganizing the arithmetic operation in such a way that you can track, you can identify the potential sources of loss of significant digits or loss of precision in the computation. And then rearrange the computation, so as to avoid getting into that kind of mode.

So, with that I wrap up my discussion on this possible sources of error in scientific computation and we will keep all these basic considerations at the back of our mind during all through the course, through the discussions of our scientific computations and approximate solutions.

So, with this we wind up our discussion on errors in floating point operations. And we will discuss next the basis of approximation and some concepts on linear algebra, which will allow us to understand how the approximations are constructed and how to measure the errors of approximation. So, that will be the a topic next topic of our discussion and we will meet and discuss the basically linear algebra in our next meeting.

Thank you.