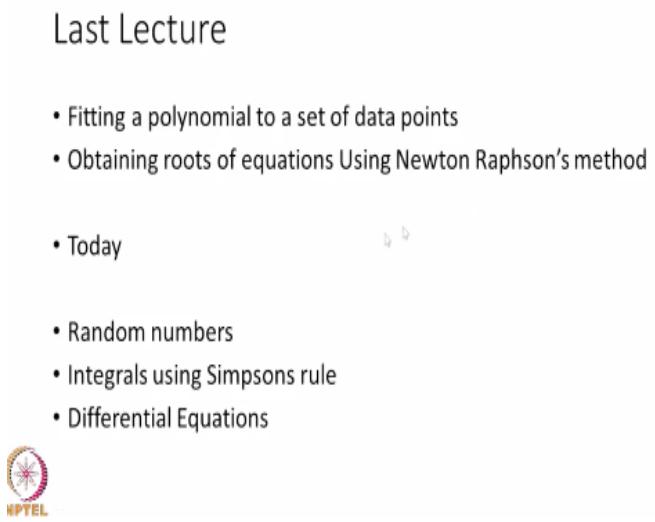
**Computational Chemistry & Classical Molecular Dynamics**
**Prof. B. L. Tembe**
**Department of Chemistry**
**Indian Institute of Technology – Bombay**

**Lecture - 21**
**Random Numbers, Numerical Integration Using Simpson's Rule**

Welcome to today's lecture. In the last lecture, we considered matrix operation and in particular use inverting a matrix to find the best fitting curve for a given set of data points. So this as well as other things we will consider in the practical session after this class. Today, I am going to do two more new items. One is we will discuss random numbers today. Look at the slide.

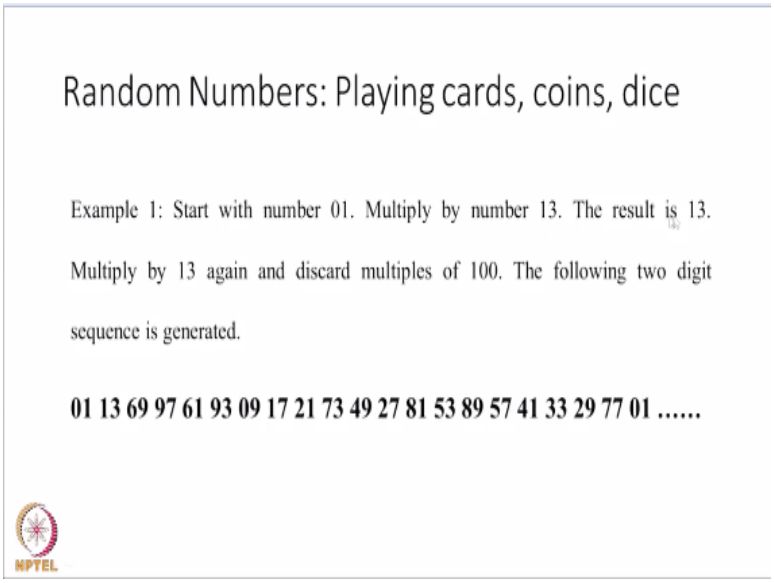
**(Refer Slide Time: 00:47)**




Fitting a polynomial to a set of data points, we studied in the last class. Also obtaining roots of equations using Newton Raphson's method, this also we considered in the last class. Today we will do Random numbers, Integral using Simpsons rule and Differential Equations. As I have been telling many times our emphasis has been to introduce you to all these numerical methods, they are not going to be comprehensive.

You just have to be comfortable with all the concepts and how to use very simple programs to do what you need. So today we will be discussing random numbers. These are extremely useful in doing computer simulation such as a Monte Carlo Simulation as well calculating integrals which cannot be calculated in any other way. So the next slide shows what I mean by random numbers.

**(Refer Slide Time: 01:38)**




So look at this slide. So when we play cards we know that there are 52 cards and each time you draw a card it is completely random anyone of the 52 can come and there is no formula to tell you which card will come when you pick one at random. Similarly, when you toss a coin you can never predict whether it is head or tail, but when you toss it say a million times you expect half the time it to be head and half the time it to be tail.

Same thing when you throw dice there are 6 options, so on the average you will get each face 1/6 of the time. So now this is conceptual but I want to use a computer to get a set of random numbers. So I shall illustrate it with a particular sequence. Look at this sequence, start with number 01. To this number 01 I multiply by number 13 so the first result will be 13. I started with 1 my next result is 13. In the next step I will multiply this 13 by 13.

So when I multiply 13 by 13 the answer is 169, out of the 169 I will remove 100 so what remains is 69. So look at the following sequence. First number was 1 next number was 13, 13*13 was 169 out of that 169 I removed 100 so I got 69, multiply 69/13 again. Keep only the digits between 1 and 99, okay. So that is called a mode operation Modulus operation is you divide a number by that 100 and only keep the reminder. It is a reminder operation.

So I got 97 then multiply 97 by 13 and remove all the multiple of 100 what remains is 61. Again multiply by 13 you get last two digits 93. And if you keep on repeating I start with 1 it spans through all different numbers between 1 and 100 and again after 77 I come back to 01. Once I come back to 01 and multiply by 13 I will again get 13 69 97 and so on. So after this 01 the entire sequence will repeat.

So in the range of this sequence we got all numbers between 1 and 100, you see there is a number between 10 and 20. There is a number between 60 and 70. There is a number between 90 and 100. There is a number between 0 and 10. There is a number between 20-30, so it is spanning my entire range in a more or less a uniform manner. So these are called Uniform random numbers. So this is a very elementary algorithm.

**(Refer Slide Time: 04:30)**

$$R_i = (mul * R_{i-1} + add) * mod\, m$$

$$R_i = (25173 * R_{i-1} + 13849) * mod\, 65536$$

$$R_i = (13 * R_{i-1} + 0) * mod\, 100$$

These are uniform random numbers

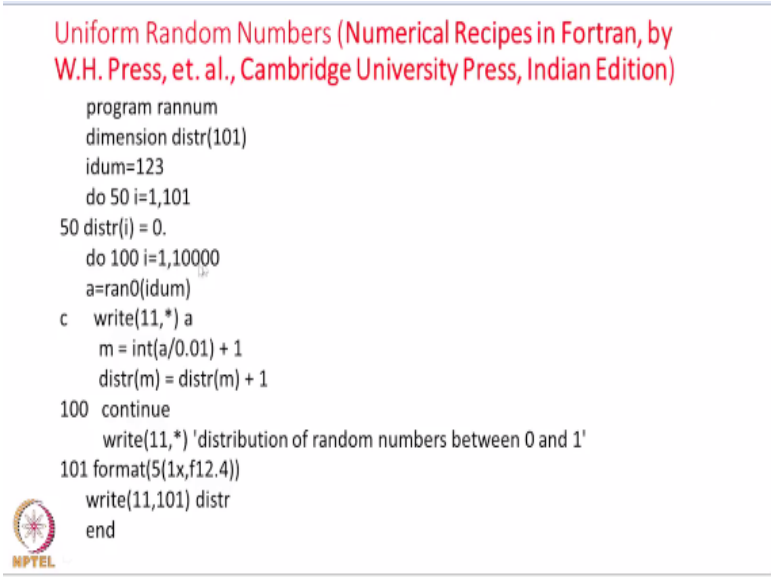

So the next slide shows a more compact algorithm. So what this algorithm says, random number i is obtained by old random number this was Ri-1 multiply that by some number in our case it was 13, if you want you can add or subtract something then modulus m, modulus m is the reminder function. In our earlier case, I use Ri=13 times Ri-1+0 I did not add anything mod 100. This is a modulus function.

So this particular algorithm which I call the uniform random numbers because they are distributed between, in this case it was between 0 and 100. I can always scale it between 0 and 1

dividing by 100. So they are uniformly distributed in a given range. These are uniformed random numbers. And better algorithm will be, look at this Ri will be 25173 Ri-1 + 13849 modulus of 65536.

So you use this a large enough number so that all my random numbers will lie between 1 and 65535. They will all lie between 1 and 65535. So this is the better algorithm then algorithm we considered here. So now what I want to use? I want to use a professional algorithm to obtain a large number of random numbers then we will see that to do with those random numbers.

**(Refer Slide Time: 06:05)**




So what I am going to use, I am going to have an algorithm for generating uniform random numbers. That particular algorithm we will not discuss in great detail because it is a little more complicated than our normal things that we considered earlier. So this particular algorithm I got from this Numerical Recipes in Fortran, by W. H. Press, et. al. There are four authors. It is a very well known book Numerical Recipes in Fortran, there is an Indian edition.

So this book has algorithms to solve many problem that you need in day-to-day analysis of chemical problems. So it is a very well-known book. These numerical recopies are there in Fortran; they are there in C, they are in other languages also, okay. So you may be able to download all the algorithms from the website. The actual algorithms may be available on the website.

So for our today's program we will use the Uniform Random Numbers given in this book. So I will just give you an outline of how to use those random numbers. So I shall call this program random, that is the name of the program. I have given a dimension distribution 100; 101. So what I want to do I want to generate random numbers between 0 and 1. And as you go along you will see what this 101 will do for us. So idum, this is a variable to start this random number program.

I have to give you a starting number idum make sure it is not 0 because if it is 0 it will create a problem. So in that book several algorithms are given for uniform random numbers. I am going to use the first one. So this is my starting it is called a seed then since I had 101 values of distr variable I will assign all those to 0, so this do loop do 50i going from 1 to 101; 50 distr i=0. So all the values of this variable are adjusted to 0 or set to 0 at the beginning.

Now is my main loop. This do 100 i going from 1 to 10000 so this is my do loop, so that is the statement up to 100 all the lines up to line 100 continue are repeated 10000 times. So what I want to do is to I want to get 10000 random numbers then arrange them in a particular way, okay. So do 100 i going from 1 to 1000. So a=ran0 idum, okay. This is a function; this is a function ran0 in the bracket idum. I already defined idum ones.

So it calls this random number function and then value of the random number is given as a. So if we want we can write all those random numbers in some file, but it will be a very long file because there are 10000 numbers in that so which we do not want to do that, it is a comment card. And once I have this random number this random a will lie between 0 and 1. So what I want to do with this program I want to know how many numbers are between 0 and 0.1.

How many between 0 and 0.2. So I want to make 100 boxes, I want to make 100 boxes each of space 0.1 and I want to know how many random numbers will fall into each of the box. For example, suppose that random was 0.5 so what this is doing m= this is 0.5 divided by 0.01 so that will be 50 and I add 1. So from that 0.5 I got a number 51 that m will be 51. So the 51st box will be incremented by that 1. So next time let us say my random number is 0.4.

That 0.4 divided by 0.01 is 40, 40+1 is 41. The 41st box will be put will be incremented by 1. So this way if I generate 10000 random numbers and my range is between 0 and 1, so if they are completely uniformly distributed each bin of 0.01 size should have 100 random numbers. There should be 100 between 0 and 0.1; 100 between 0.01 and 0.02 so each one of this should contain 100 numbers. So we will see through the program whether each one will contain 100.

So if each of the bins contains 100 it will be an absolutely perfect random number between 0 and 1 but in general that does not happen. In fact, if it is too perfect it is not probably as good. So we will see the results in a later situation. Again in this do loop what I have done? I have obtained 10000 random numbers each is between 0 and 1 and to find out in which part of 0 and 1 it is I am dividing by 0.01, so when I divide that by 0.01 I get a integer between 1 and 100.

So each random number would be put in one of those bins and my distr, d i s t r will contain this distribution of random numbers, okay. So after this whole thing is over it will write fine 11 distribution of random numbers, okay. And it will write all that array distr is an array now, array of 101 points. So when I say write 11, 101 distr it will write all the 101 values of that array file 11, file 11 here I have not given. If nothing is given it will create some file by itself, okay.

4.11 fort.11. it will create a file fort.11. But you can always open unit=11 some file name, you can do that as well. So in this case it will create a file by itself. So now all I have to show you is how this particular function works.

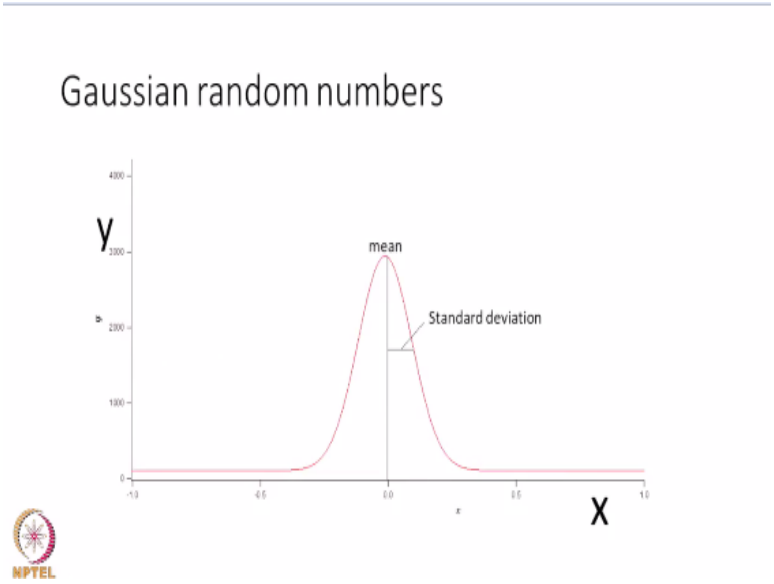
**(Refer Slide Time: 12:35)**

```
function ran0(idum)
integer IA,IM,IQ,IR,MASK
real ran0,AM
PARAMETER(IA=16807,IM=2147483647,AM=1./IM,IQ=127773,IR=2386,
1MASK=123459876)
integer k
idum=ieor(idum,MASK)
k=idum/IQ
idum = IA*(idum-k*IQ)-IR*k
if (idum.lt.0) idum = idum + IM
ran0 = AM*idum
idum = ieor(idum,MASK)
return
end
```

The next slide shows details of this function. This function ran0(idum) this is a function. That function uses several integer variables. It uses real variables. It uses many, many parameters, okay. And it does all these operations and every time it returns you a random number. It return you a random number which will be called; in our earlier program it was a=ran0 idum each time it gives you a random number as well as a new value of idum, okay i d u m.

So you note this program, execute exactly what it is using g Fortran and we will not discuss too much about this because there are many, many professional algorithms which gives you random numbers. I have already discussed the very simple algorithm for you and you know what random numbers are.

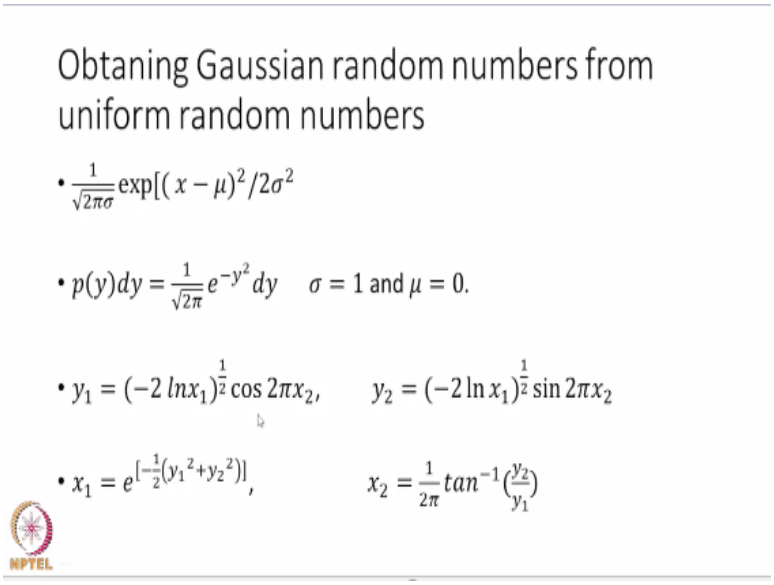
**(Refer Slide Time: 13:27)**




So once I do this now I will consider another set of random numbers. Our earlier sets of random numbers were uniformly distributed between 0 and 1. Now this is a different distribution, these are called Gaussian random numbers these are distributed between some – value to + value. The range is quite wide but they are all picked around that mean. They are picked around the mean and this is the standard deviation. These are called Gaussian random numbers.

I shall also tell you the algorithm to get this, okay. You can study that in great detail.

**(Refer Slide Time: 14:03)**



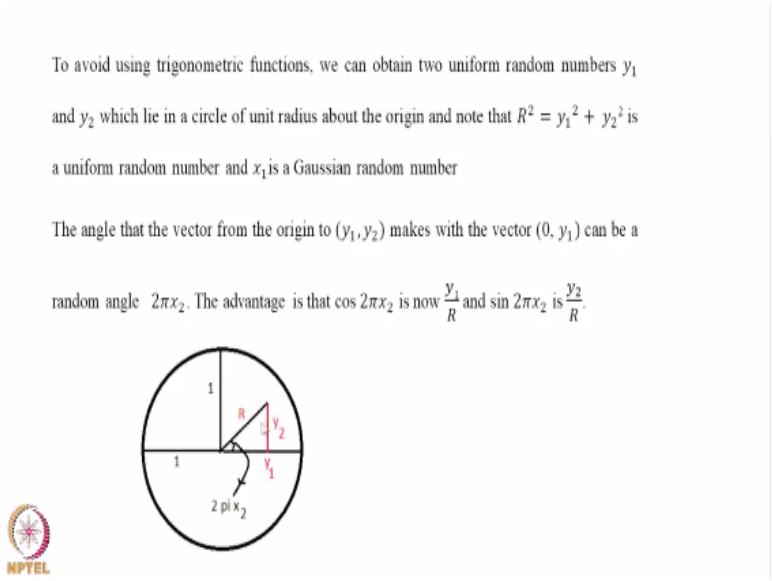

I will give you the algorithm to get Gaussian random numbers. What are Gaussian random numbers? Now they are distributed between –infinity to infinity because this x can take any

value but it is all picked it is all picked around 0 with a standard deviation of sigma, okay. So this py dy is a Gaussian random number distribution so it will be distributed as 1/2 pi e to the –y square dy sigma=1 Mu=0.

So the way to do it would be you define two uniform random numbers y1 and y2 and from y1 and y2 you get x1 and x2. How do I get x1 and x2 from y1 and y2; x1 will be e to the -1/2 y1 square + y2 square, this is 1 and x2 is 1/2pi tan inverse y2/y1. This arithmetic you can work out. So all you have to do is to generate two random numbers y1 and y2 between 0 and 1 and from that generate two Gaussian random numbers x1 and x2.

My next slide will give you greater picture of what this y1 and y2 are.

**(Refer Slide Time: 15:16)**



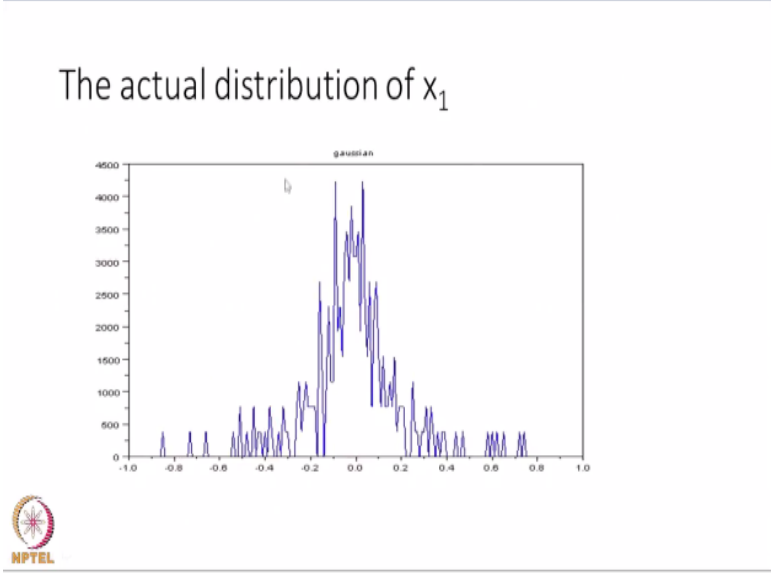

So look at this. Suppose I have two random numbers y1 and y2 and these random numbers both are between 0 and 1. So y1 lies between 0 and 1; y2 lies between 0 and 1, so since both are lying between 0 and 1 in principal this R square which is a sum of y1 square and y2 square. So in principal this R square can be 2 because y1 could be 1 and y2 could be 1 so R square could be 1 square + 1 square that is 2 square.

So if R square is 2 that means the number will lie somewhere in that box it will lie in a square box of edge length 2. But I want R square to be inside that circle. How to I ensure that R square

to be inside the circle? When the sum of y1 square and y2 square is < 1 then surely ask where is <1? So what I will do I will get two random numbers y1 and y2. If the square of the two random numbers is < 1 then only I will consider my remaining operations.

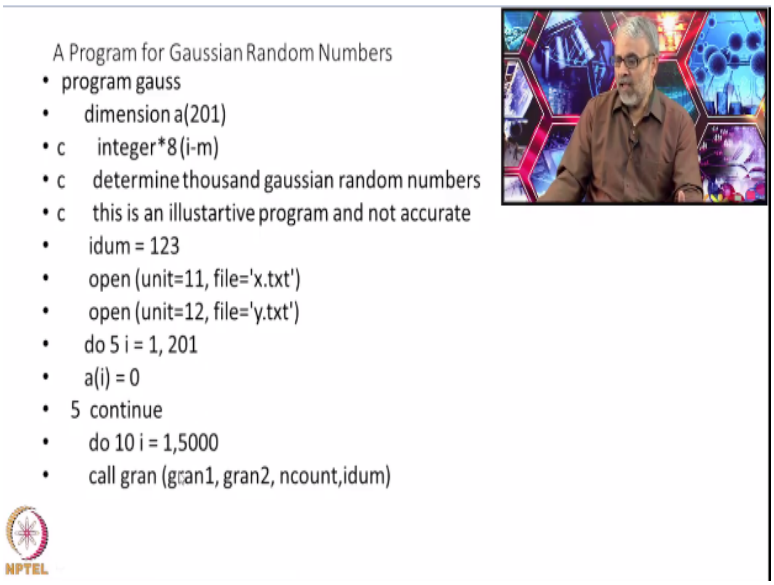
So only when R square is <1 I will proceed further. So what are these y1 and y2? This is my y1, this is my y2. So x1, I already told you how to get x1 in the previous slide and how do you get x2? This angle s 2pi x2, okay. So cos 2pi x2 is y1/R, trigonometry will tell you that y1/R is cos 2pi x2 and y2/R is sin 2pi x2. So this way I generate two Gaussian random numbers x1 and x2 okay.

**(Refer Slide Time: 17:13)**




So its distribution is shown here. So how is x1 distributed by using the previous algorithm? It is distributed such that all values are allowed between large negative value and then large positive value and the random numbers are distributed which are picked around x=0 it is sharply picked at the origin and with a standard deviation also 1, okay. So this is a distribution of x1, okay.

**(Refer Slide Time: 17:39)**




So now this slide and the next slide gives you an algorithm to generate these Gaussian random numbers, okay. So all I have done here just as in the earlier case I define a new array 201. Now if I want very accurate results I need to what is called double precision arithmetic. I have discussed it in one of the earlier classes. So double precision arithmetic uses more spaces than then single precision arithmetic, okay.

So I am not using higher precision here. I just I am using a normal precision. Again idum = 123. I opened two files 11 and 12, okay. And I set all my A values to 0 do i, do 5 i going from 1 to 201 a(i)=0 5 continue. It sets all the initial values of my array a to be 0 then is my next point. So 10 i going for 1 to 5000. It calls this Gaussian random number program 5000 times. So I generate 5000 Gaussian random numbers by calling this function 5000 times.

So call gran, gran1, gran2, ncount, idum; idum was my initial value and as I mentioned before do not ever set this initial value to be 0 then there will be a problem. Always it should be non-zero. Instead of 1,2,3 it could be 2,3,1. It could be something else but never 0. So I call my Gaussian random number function many, many times. It is a subroutine; I call it many times.

**(Refer Slide Time: 19:20)**

```
c      write (11, *) gran1
       ii= (gran1 * 10) + 100
       if ((ii .gt. 200).or.(ii.lt.0)) go to 10
       a(ii) = a(ii) + 1
  10 continue
       write(12,99) a
  99  format(6(1x,f10.3))
      end
```

So once I call the Gaussian random number now again I want to put that random number in some array because we know that this Gaussian random number is going to be anywhere from some large negative value to large positive value. So I want to put them in an array. So when I put them in an array, so gran*10 suppose that Gaussian random number is 0.5 if it is 0.5 that 0.5*10 is 50, okay. 50+100 is 150. So if the Gaussian random number is 0.5 this ii will be 0.5.

If the Gaussian random number is 0.1, 0.1*10 this 1+100 so it will be 101. So this way wherever the Gaussian random number lies between a – value and a + value I get an integer corresponding to that and I update my array ii. I update my array a so if that random number comes many times so that array will have values of that index. So I will execute it and show you but it may so happen that my array had only a size of 200.

So if this ii is >200 or if ii is <0 it will not be able to accommodate in this array a. So therefore, I just want to skip that because if by chance the ii becomes 202 it will give an error because a has as size of only 201. And if ii is <0 or 0 again there is a problem okay so it will not accept. ii will never be 0 because even if the random number is 0 whatever happens to that random number that, okay. So if ii is less; in principal if ii is 0 it could give you some trouble.

But I can say; so to avoid that I can say ii < 1; instead of 0 if I make ii < 1 it will still execute. So it ensures that ii lies between 0 and 200; 1 and 200 so I will also execute and show you this

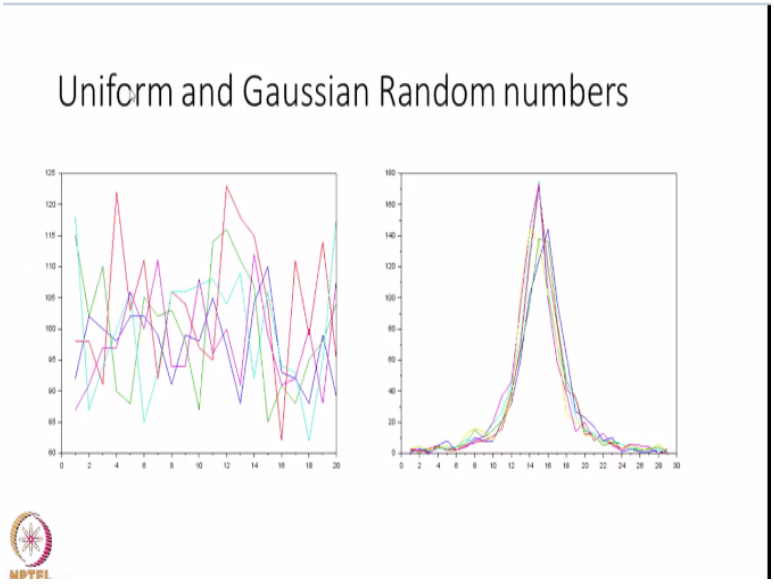
program. So then after I calculate all my random numbers in this case there are 5000 it will write it in that file 12, okay.

**(Refer Slide Time: 21:50)**

```
      subroutine gran(gran1, gran2, ncount,idum)
      ncount=0
10    x1 = ran0(idum)
       x2 = ran0(idum)
      w1 = (2 * x1 - 1.)
      w2 = (2 * x2 - 1.)
c      write(11,*) w1,w2
      rsq = w1 * * 2 + w2 ** 2
      ncount = ncount + 1
      if (ncount .gt. 1000) stop
      if (rsq .ge. 1.0) go to 10
      fac = sqrt (-2.0 * log (rsq))/rsq
       gran1 = w1 * fac
       gran2 = w2 * fac
      return
      end
```

So this is the rest of the Gaussian random number function. This is the rest of the Gaussian random number function. And this also need ran0, ran0 was the uniform random number.

**(Refer Slide Time: 22:01)**




So that is also shown here. ran0 is exactly what was before. So I will now show you ran0 again. So my program consists of, okay. So this is the subroutine gran, it calls ran0, ran0 I have already explained. You add all the lines for ran0 function in this program. So the result of this two

programs are as such when I obtain a uniform random number remember first program was uniform random number. See it is uniformly distributed in some range.

Whatever is the range it is uniformly distributed although it shows lot of fluctuation, these fluctuations really are not too much because this is 100, 105, 110, 120. So since I had generated 10,000 random numbers each box have to contain 100 there should be 100 random number between 0 and 0.02; 100 between 0.01 and 0.02. But in each internal it is lying somewhere between -90, -80 to +120. So they are more or less uniformly distributed across the entire range.

Whereas the Gaussian random numbers see how nicely they are picked at that midpoint and with the standard deviation 1. So this is how you will generate random numbers using this two functions and subroutines. This uniform random numbers are required in Monte Carlo simulation and Gaussian random numbers are needed when you want to assign all the velocities of the particles in a gas.

You know that molecules in a gas are distributed according to the Maxwell Boresman distribution of velocities. So one way to get the Maxwell Boresman distribution of velocities is through your Gaussian random number.

**(Refer Slide Time: 23:51)**

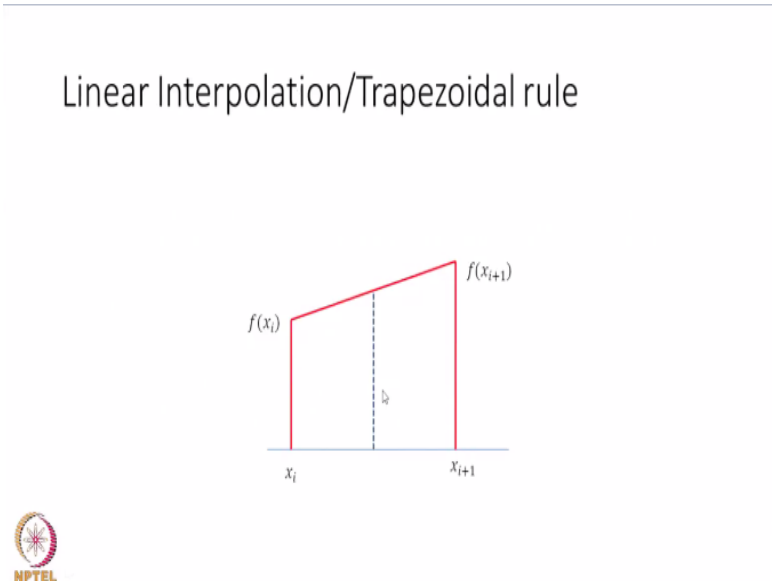
## NUMERICAL INTEGRATION

- Given a set of data points $(x_0, f(x_0)), (x_1, f(x_1)), \ldots . (x_n, f(x_n))$
- We want an estimate of the integral I,
- $I = \int_{x_0}^{x_n} f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx$

So now we will move to Numerical Integration. So we will do briefly numerical integration and differential equations. Just a very simple two algorithms for numerical integration and two algorithms for differential equation. So what is numerical integration? Numerical integration is I have a set of data my data is x naught f x naught, x1 f x1, xn f x1 that is like xi yi.

I have set of data xi yi and I want to integrate the function x numerically because I do not know the formula so I want to numerically integrate that. So how do I do that this integral? x naught to x1; fx dx so this is my integral. So I split that integral into n segments, so there are n segments. In each segment I integrate xi to xi+1 okay. So this is my integral which is split into n segments.

**(Refer Slide Time: 24:52)**




So one way to split the segment is to use what is called a Trapezoidal rule. What is the Trapezoidal rule? This is my f xi; this is my f xi+1 I assume that the function varies linearly between these two points. The functions varies linearly between these two points, so I vary linearly; the function is a linear function. So what is the linear function?

**(Refer Slide Time: 25:17)**

- $P_1^i(x) = f(x_i) + \frac{\Delta f(x_i)}{h}(x - x_i) \ \forall \ x_i \le x \le x_{i+1})$
- where $\Delta f(x_i) = \frac{[f(x_{i+1}) - f(x_i)]}{}$

- $I_i = \int_{x_i}^{x_{i+1}} P_1^i(x)dx = \frac{\Delta f(x_i)}{h} \int_{x_i}^{x_{i+1}} (x - x_i)dx + f(x_i)h$

- $f(x_i)h + \frac{\Delta f(x_i)}{h} \cdot \frac{h^2}{2} = \frac{h}{2}(f(x_{i+1}) + f(x_i))$

- $I = h\sum_{i=0}^{n-1} \frac{f(x_{i+1}) + f(x_i)}{2} = \frac{hf(x_0)}{2} + h\sum_{i=1}^{n-1} f(x_i) + \frac{hf(x_n)}{2}$

My linear polynomial f xi + delta f xi/h x-xi for all xi going from xi to xi+1, okay. So delta f xi is a different between f xi+1 and f xi. So when I integrate this, this is the first order polynomial, when I integrate this function, what I get is f xi*h + f xi/h*h square/2, so this is my integral. h/2 f xi+1+f xi. So this is really the area of this Trapezoidal, what is the area of Trapezoidal? f xi, f xi+1-f xi take the difference, take the average, average value is this point.

And that average value you divide by this interval. That interval I have called h. So area of the interval is half of the height between these two multiplied by the base width, okay so that is my integral. So since I have n such intervals my total integral will be h*f0/2 and h*f xn/2 and all the intermediate points they have to be multiplied by h because the first point is only h/2 times fx0, mass point is h2*f xn. All the intermediate points will come twice.

One from the left side and one from the right side, so h* all the values of the function, so now it is very, very easy to program it. How will I program it? Read all the data.

**(Refer Slide Time: 26:58)**

## Derivation of Simpson's rule

- $I = \int_{x_{2i}}^{x_{2i+2}} P_2^i(x)dx$
- $I_i = \int_{x_{2i}}^{x_{2i+2}} [f(x_{2i}) + \frac{\Delta^1 f(x_{2i})}{h}(x - x_{2i}) + \frac{\Delta^2 f(x_i)}{2h^2}(x - x_{2i})(x - x_{2i+1})]dx$
- Where $\Delta^1 f(x_i) = f(x_{i+1}) - f(x_i)$
- And $\Delta^2 f(x_i) = \Delta^1 f(x_{i+1}) - \Delta^1 f(x_i) = f(x_{i+2}) - 2f(x_{i+1}) - f(x_i)$
- $I_i = h[2f(x_{2i}) + 2\Delta^1 f(x_{2i}) + \frac{1}{3}\Delta^2 f(x_{2i})]$
- $= h/3[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})]$


So read all the data, okay. Sum from 1 to n-1. Sum the values of xi and multiply by h. So that is one term. The first term and the last term, you add the two terms okay divide by 2 and multiply by h. So this integral is a sum of quantities, what are those quantities? All the functions added from 1 to n-1 multiply by h and the first one and the last one add the two and multiply by h/2 that sum is the integral using the Trapezoidal rule.

So I will conclude at this point. And in the next our I will consider Simpson's Rules and also how to solve differential equations. I will consider briefly and then do the detail programming for the same. So I will conclude this one here. Thank you.