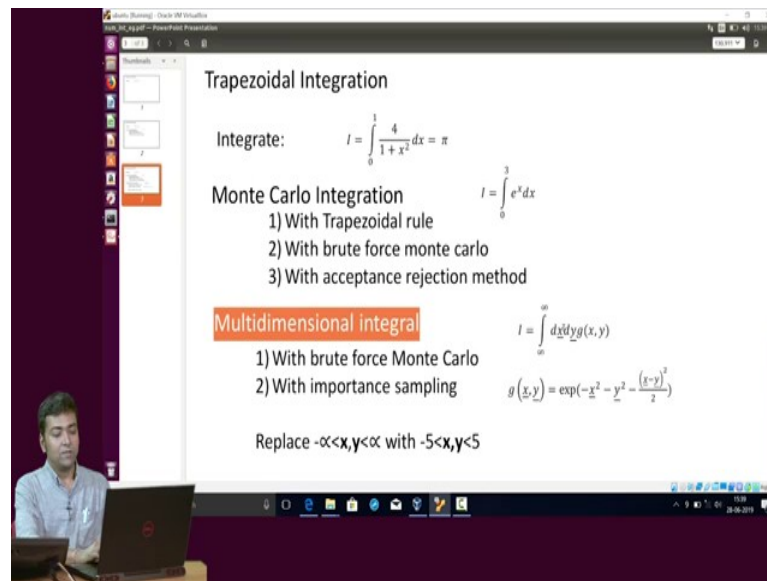


Computational Physics
Dr. Apratim Chatterji
Dr. Prasenjit Ghosh
Department of Physics
Indian Institute of Science Education and Research, Pune

Lecture - 15
Numerical Integration Part 09

(Refer Slide Time: 00:22)



The screenshot shows a presentation slide with the following content:

Trapezoidal Integration

Integrate: $I = \int_0^1 \frac{4}{1+x^2} dx = \pi$

Monte Carlo Integration $I = \int_0^3 e^x dx$

- 1) With Trapezoidal rule
- 2) With brute force monte carlo
- 3) With acceptance rejection method

Multidimensional integral $I = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x,y) dx dy$

- 1) With brute force Monte Carlo
- 2) With importance sampling

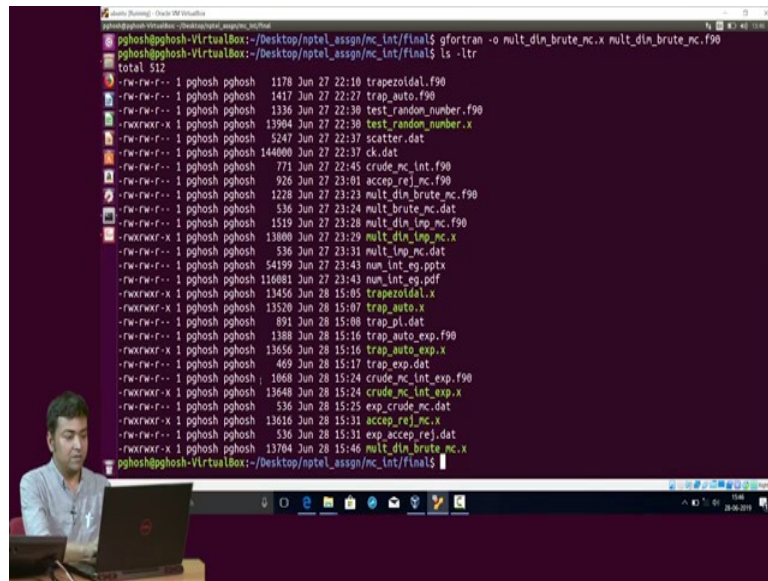
$g(x,y) = \exp(-x^2 - y^2 - \frac{(x-y)^2}{2})$

Replace $-\infty < x, y < \infty$ with $-5 < x, y < 5$

Now, we will do in go into multidimensional integral. What we will do is we will evaluate this 6-dimensional integral so, where basically x and y here are vectors which has three components; each has three components x and y. So, the total the integral the dimension of this integral is 6 and we will evaluate this particular integral.

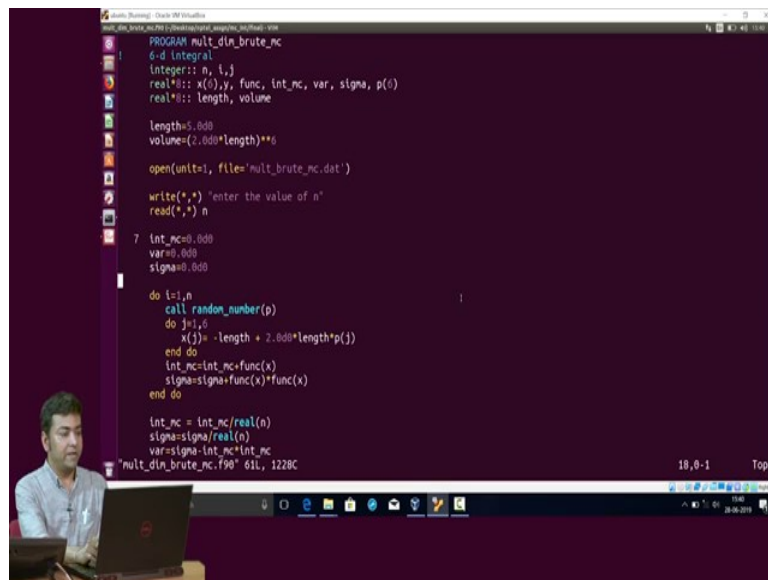
And, we will use it two use two Monte Carlo methods: one is the brute force one which is the simple Monte Carlo technique and then we will use what is called the important sampling which is a improved and more efficient way of evaluating this integrals using Monte Carlo method.

(Refer Slide Time: 01:01)



So, let us first look into the brute force way. So, this is my file multi dimension brute underscore f90.

(Refer Slide Time: 01:16)



So, this is the program which will do the brute force integral for us and the idea is exactly same as the previous examples which we looked for Monte Carlo integration for example, e to the power integration of the e to the power x, but there we had one variable x, here we have six variables; so, x 1, x 2, x 3, x 4, x 5 and x 6. So, for that what we need

is six random numbers instead of just one random number and this volume will be a 7-dimensional volume that is that is the value of the integral which we will get.

And, though I mean the limits of this integral are from sorry, there is a mistake at this will be from minus infinity to plus infinity, but we cannot do that on a computer. So, we will restrict the limits to minus 5 to plus 5 and this is a very valid assumption because, this function as you can see it is a form of a typical deform of a Gaussian which decays exponentially e to the power minus x square of that is or that is how it is decays. So, it decays very fast so, which has a long tail which will have negligible contribution to the integral.

So, let see how it is done. So, by length that here instead of the complete length of the integral I have used just one side from 0 the positive x 1 for example, or 0 to positive x 2 or 0 to positive x 3. So, the total length along each dimension say for x 1 the total length is twice this value that is 10, we have 2 into length and since this is a 6 dimensional integral so, the volume will be given by to the power 6. So, this is my volume. I need to multiply the average of the function with this volume to obtain the integral.

So, now, I start computing a again as before I am I will be doing it for several values of n I will be evaluating say the integral using 10 by computing the average 10 times, then 100 times, 1000 times so on and so forth and we will see how the number is accuracy is changing. So, what we and those values are stored in this particular file here then I asked the user to give me small value the starting value, then I initiate my variables to 0. This one stores the final value of the integral, this one stores the variance and this one stores the sigma.

Then, what I do is this is the place where I am executing this do loop helps me in computing the average. So, each time this do loop is executed what it does is it calls random 6 random numbers. So, now, unlike the previous cases here you see this p argument which is used which is a dummy argument for my random number generator it has a dimension six. So, what it means is that it will return 6 random numbers. And, once I got this 6 random numbers so, what I need to do is these random numbers are 0 4 2 1 and again as before now I need to have these random numbers as space from minus 5 to plus 5 and this is how that is done.

So, the p's I am converting to p's which lie between 0 to 1 I am now here is converting it from minus 5 to plus 5.

(Refer Slide Time: 05:02)

```

int_nc=volume*int_nc
sigma=volume*sqrt(var/real(n))

write(i,*) n, " ", int_nc, " ", sigma

begin automation
n=n*10
if (n .lt. 1000000000) goto 7
end automation
END PROGRAM mult_dim_brute_mc

real*8 function func(x)
implicit none
real*8::x(3), xx, yy, xy
real*8:: a,b
a=1.000
b=0.500

xx=x(1)*x(1)+x(2)*x(2)+x(3)*x(3)
yy=x(4)*x(4)+x(5)*x(5)+x(6)*x(6)
xy=(x(1)-x(4))**2+(x(2)-x(5))**2+(x(3)-x(6))**2
func=exp(-a*xx-a*yy-b*xy)

end function
  
```

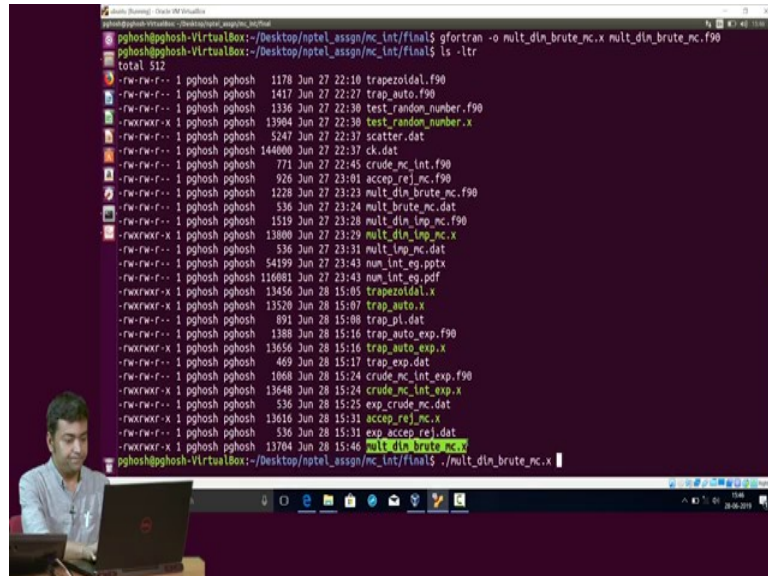
And, once I have that value of p I call this function so, this function does nothing, but evaluate the integrand. So, basically this is what it does e to the power minus a x square minus a y square minus b x y, where a and b are 1.0 and 0.5. So, it is nothing, but just basically this particular function. So, if you compare this with this thing yeah. So, this is a function it is trying to evaluate here. So, this is e to the power minus x square minus y square.

So, my x square will be x the x component x 1, x 2, x 3 that the mod of that vector and then that is what is here, the x square x 1 square plus x 2 square plus x 3 square; similarly, y square will be given by x 4 square x 5 square and x 6 square and then I have x minus y whole square. So, this is how the function is evaluated and once I get the value of the function at a given at a given set of values of x I add to this particular variable and I keep on doing it 10 times.

Similarly, for the same way I do for the sigma and once that is done so, for the integral what I do is I compute the average and once the average is computed then I compute the I multiply it by volume to get the value of the integrand. So, once that is done what I do is I write my n, then the value of the integrand and then the standard deviation. So, this I

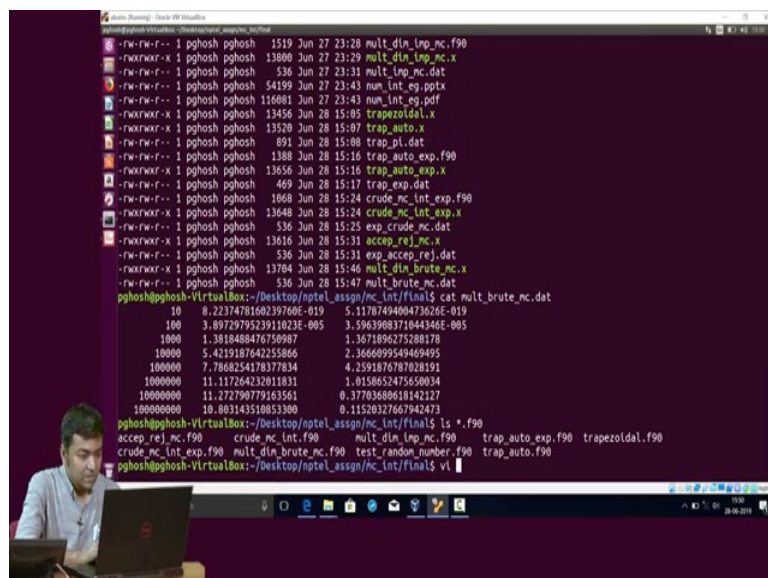
print into the file and this is done till a certain number of n takes a certain value. So, this is how it is done and then let us see how efficient how it performs.

(Refer Slide Time: 07:23)



So, I compile it as before dimensional mult dim underscore brute f90. See, if I compile it then this is the executable that has been produced here. You can see, then I run it.

(Refer Slide Time: 07:33)



So, it asks for a value of 10, I start with a ridiculously small value though it is meaningless, but let us start with that. And, so, if the code is now doing the calculations so, let us wait for some time for it to finish the calculation and, then let us we will see

how the results are. So, where particularly the last two points take a significant amount of time so, that is why it is it is still doing it is done. So, now, if I do I_s minus I_{tr} , so, what I see is that this is the file where I have my data which is generated by the code.

So, if I pull that out so, I get the following. So, again just reminder so, this is the value of n that is the number of random points I used to compute the average of my function this is the value of the integral and this is the standard deviation. So, what you see is that so, so for example, with 10 grid points random numbers or it is 100 random numbers you get some ridiculous values of the integral, it is completely wrong you see.

And, to get somewhere close to the very small so, in I mean in Monte Carlo integration the accuracy of the of the integral typically is determined by the a standard deviation here and you see that even with so many points with these many points also the standard deviation is still pretty high which you do not want. You want a very low standard deviation, the lower the value, the more accurate will be your calculations.

So, now to improve this what we will do is we will use the we will evaluate the same integral here using this importance sampling. So, if just to recollect what importance sampling is what we do in importance sampling we modify the function in such a way or in other in importance sampling we what we do is we have a distribution of random numbers which has a uniform distribution, we change the distribution in such a way that the shape of the distribution matches the shape of the function.

So, for example, here this function is sort of like a Gaussian function. So, what we will do is we will do a change to two things: one is we will change the random number generator which gives us uniform distribution we will use that random number generator, generate a set of random numbers with uniform distribution and change the distribution to that of a Gaussian deviator. And, once it is done we will also do another thing that is the change of the variable.

So, when we do the change of the variable that is basically what it means is we will divide this function by e to the power minus x square and minus y square that way. So, once we divide that this by that function so, what will happen is these two terms will go away. So, my modified function will just be the exponential minus x minus y whole square ok. So, this is what.

So, I have to integrate this new function with a distributional random numbers which has a Gaussian distribution now instead of a uniform distribution and the way we have also learnt that how to generate a Gaussian distribution. For a Gaussian to generate a Gaussian distribution what we need is two random numbers at a with uniform distribution at the given time and then we will use the Box-Muller method to do that. So, let us look at how the code is. So, we go so, this method is called the important sampling.

(Refer Slide Time: 11:34)

```

PROGRAM mult_dln_lnp_nc
66 integrat

implicit none
integer:: n, i
real*8:: x(6), y, func, int_nc, var, sigma, p(i)
real*8:: length, volume, sqrt2, gauss_dev

length=5.000
volume=acos(-1.000)**3
sqrt2=1.000/sqrt(2.000)

open(unit=1, file='mult_lnp_nc.dat')

write(*,*) 'enter the value of n'
read(*,*) n

7 int_nc=0.000
var=0.000
sigma=0.000

do i=1,n
do j=1,6
call random_number(p)
x(j)=gauss_dev(p)*sqrt2
end do
int_nc=int_nc+func(x)
sigma=sigma+func(x)*func(x)
end do

"mult_dln_lnp_nc.f90" 77L, 1519C
5,22 Top

```

This is what my program does. So, again as before I have a set of integer variables which are used for the same purpose. Here I have one more extra counter, then again I need 6 basically I will be needing 6 numbers. So, that is stored in this variable x and each time I need 2 for each number random number with Gaussian distribution I need 2 numbers with an uniform distribution which are stored in this variable p 2.

And, then I have this particular function which generates my Gaussian distribution random numbers with Gaussian distribution and this is the function which evaluates the value of the integrand at those set of 6 numbers. So, again this length is given by this. Now, if we do the algebra your volume we have to multiply it by a factor of pi cube. So, this what is given here and the in the Box-Muller method the standard deviation of my Gaussian distribution is different from the standard deviation of the Gaussian distribution we need to use.

So, and if one does the algebra as we have seen before in the lecture so, we need to multiply this or divide the number by square root 2. So, this is what and since we need to do it every time so, instead of evaluating this every time we evaluate it and the very beginning and store it in this variable and then we as before we write the value the values of my integral and the error in this file for different values of n. We ask the user to supply us the starting value of n and then I have read that same thing then I set my different variables to 0, and then I start doing the average.

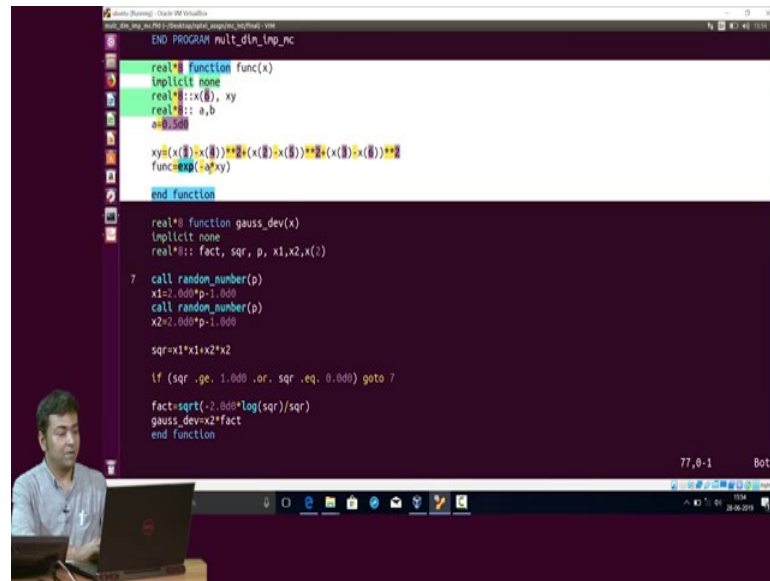
(Refer Slide Time: 13:33)

So, basically what I am doing here is I am doing the average the first loops unlike before now I have two do loops. So, this is the first loop here starting with do i equals to 1 to n and it answer and then I have the second do loop what it thus is it generates six random numbers with the Gaussian distribution starting from.

So, what I am doing here is I am calling my random number generator to return me 2 random numbers with uniform distribution these a 2 random numbers I am giving as a input to my function Gaussian underscore dev and that returns me one random number with which forms a uniform Gaussian distribution and that I multiplied by square root of 2, so that my sigma of the Gaussian distribution remains the same as y desire to do my integration and not the original Box-Mullar sigma for the original Box-Mullar.

And, once I have this 6 numbers here through this do loop what I do is I fit them to my function. The function is evaluated using this particular function here.

(Refer Slide Time: 14:49)



```
END PROGRAM mult_dtn_tnp_nc

real*8 function func(x)
  implicit none
  real*8 :: x(3), xy
  real*8 :: a,b
  a=0.5d0
  b=0.5d0
  xy=(x(1)*x(1))**2*(x(2)*x(2))**2*(x(3)*x(3))**2
  func=exp(-a*xy)
end function

real*8 function gauss_dev(x)
  implicit none
  real*8 :: fact, sqr, p, x1,x2,x(2)
  7 call random_number(p)
  x1=2.0d0*p-1.0d0
  call random_number(p)
  x2=2.0d0*p-1.0d0

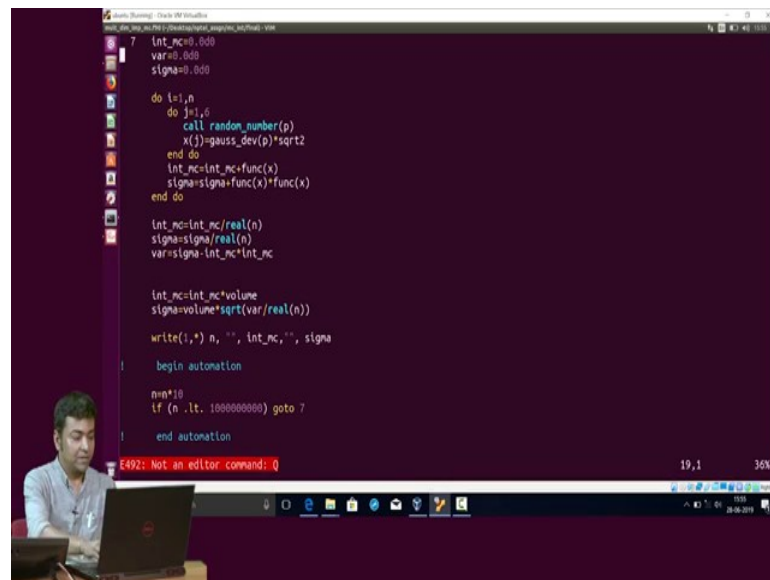
  sqr=x1*x1+x2*x2

  if (sqr .ge. 1.0d0 .or. sqr .eq. 0.0d0) goto 7

  fact=sqrt(-2.0d0*log(sqr)/sqr)
  gauss_dev=x2*fact
end function
```

So, this is the function where I am evaluating it. So, now, you see the function is just minus half xy. So, only that part left.

(Refer Slide Time: 15:07)



```
7 int_mc=0.0d0
  var=0.0d0
  sigma=0.0d0

  do i=1,n
    do j=1,3
      call random_number(p)
      x(j)=gauss_dev(p)*sqrt2
    end do
    int_mc=int_mc+func(x)
    sigma=sigma+func(x)*func(x)
  end do

  int_mc=int_mc/real(n)
  sigma=sigma/real(n)
  var=sigma-int_mc*int_mc

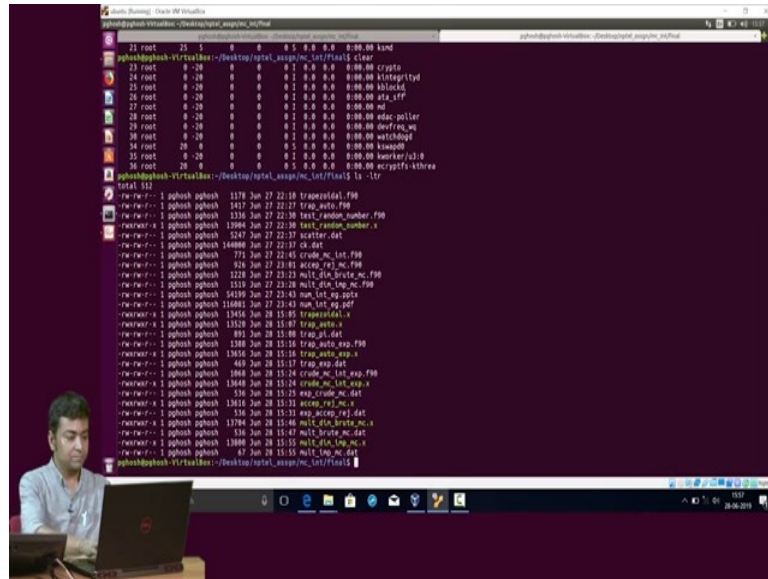
  int_mc=int_mc*volume
  sigma=sigma*sqrt(var/real(n))

  write(i,*) n, " ", int_mc, " ", sigma

  begin automation
  n=n*10
  if (n .lt. 1000000000) goto 7
  end automation
```

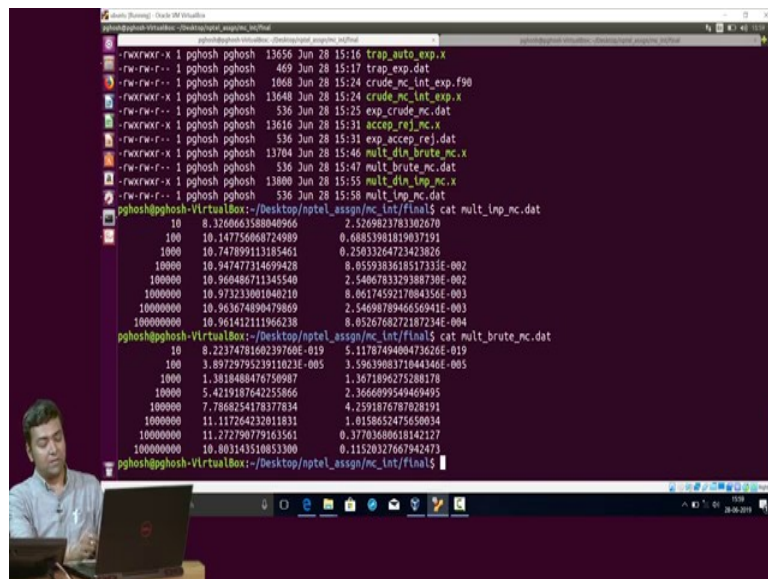
So, once this function is evaluated here. So, what I do is I take compute the average in this line, then similar way I compute the sigma by computing the average, then I compute the variance and then what I do is I multiply it by the factor of pi whole cube and the sigma also by volume and then square root of n and then I print them out in the file. So, this is how it is how the code works.

(Refer Slide Time: 18:11)



Still doing something.

(Refer Slide Time: 18:29)



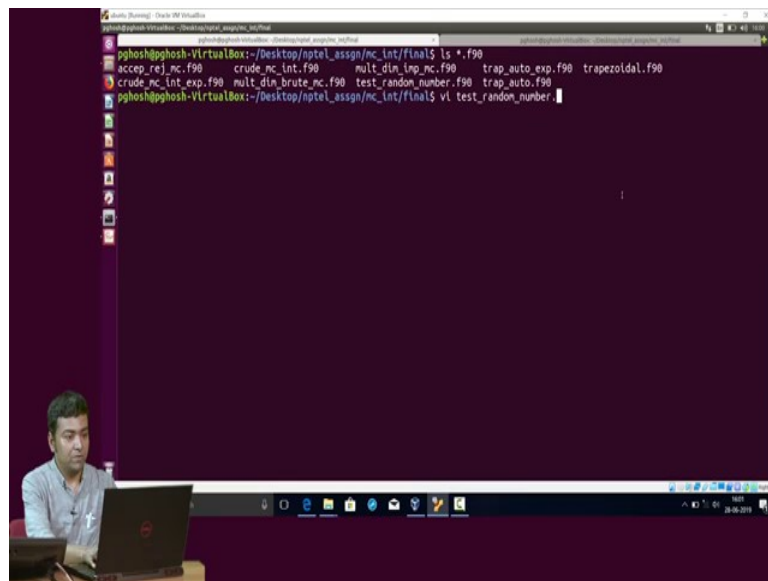
So, it is taking time because it is done, but because it is now we were doing an extra step that is it is generating a new distribution of random number. So, that way it is a bit expensive. So, once it is done then what we do is, so, this is the file which is been produced, where the data is stored and let us look at it. So, imp dat. So, again this is the first column is my n, the second column is the value of the integral and the third column

is the sigma. So, as we see and let us compare this with the brute force one which we did in the previous thing.

So, if we see that in the brute force one the lowest value of the sigma that we can get is 0.11 which is just one order of magnitude, smaller than the value of integral. For the same number of points here and here now with my important sampling you see it is converging very fast, the sigma is decreasing very fast and if you do the algebra you can see that the sigma will go up as $\sigma \propto 1/\sqrt{n}$ that also you can check and see whether it is valid.

So, this is a clear example that in doing this type of integrals the important sampling, though it might be slightly expensive than the brute force method performs or gives a much more accurate estimate of the value of the integral.

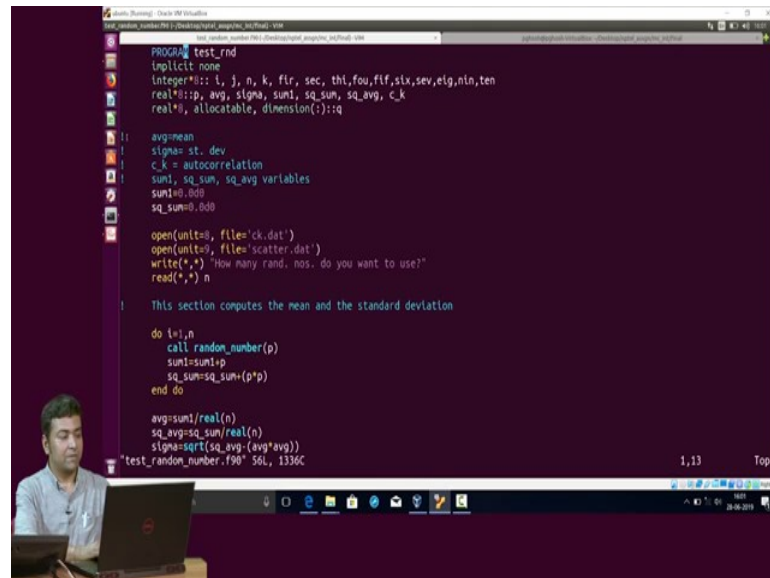
(Refer Slide Time: 20:12)



So, we will see how to test the random number with. So, I will take you through a code which has been written by me which what it does is it does the different steps of the random number. So, in the lectures what I have told is so, we have to check whether one gets the expected value or average value, the expected value of sigma, then the correlation plot one should do, one should do the scatter plot, one should do the histogram.

So, these are the five tests one is to do. So, I will not go the histogram. I will not show how to do the histogram, but I will show a code which does the other stuff. For so, example this is my code test random number dot f90.

(Refer Slide Time: 21:05)



```
PROGRAM test_rnd
  implicit none
  integer :: i, j, n, k, flr, sec, thi, fou, flf, six, sev, etg, nin, ten
  real :: p, avg, sigma, sum1, sq_sum, sq_avg, c_k
  real, allocatable, dimension(:) :: q

  avg=mean
  sigma=st_dev
  c_k = autocorrelation
  sum1, sq_sum, sq_avg variables
  sum1=0.000
  sq_sum=0.000

  open(unit=0, file='ck.dat')
  open(unit=1, file='scatter.dat')
  write(*,*) "How many rand. nos. do you want to use?"
  read(*,*) n

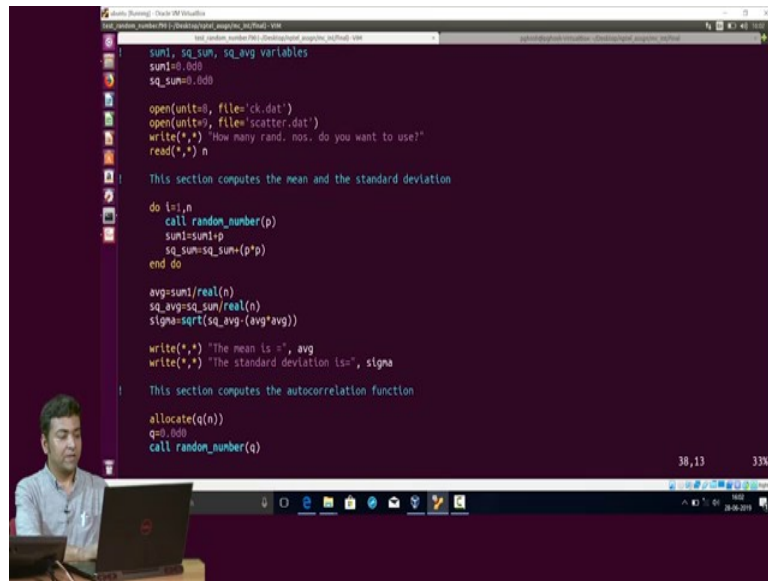
  ! This section computes the mean and the standard deviation
  do i=1,n
    call random_number(p)
    sum1=sum1+p
    sq_sum=sq_sum+(p*p)
  end do

  avg=sum1/real(n)
  sq_avg=sq_sum/real(n)
  sigma=sqrt(sq_avg-(avg*avg))
  "test_random_number.f90" 56L, 1336C
```

So, I start with the Fortran keyword program and then I have the program name. And, then I have a set of variables and it is a long list variables. Some of them are temporary variables which reused in the code and others are others the significant ones I have within comments I have kept here what it means. For example, average avg – variable stores the mean of distribution; sigma variable stores the standard deviation; c underscore k stores the auto co-relation and these are some again dummy variables.

So, to begin with I make it 0 , I set the sum1 and square of sum to be 0. So, what I will do is; so, I need to have two files. One file will store my co-relation coefficient which we have defined already in the lecture as a function of k and in other one I will store x 1 and x 2 that I mean the basically what I am going to do is I am going to store the data for the scatter plot. Then in this part what I am doing is I am asking the user to give me to tell me how many random numbers I want to use and then I read that number.

(Refer Slide Time: 22:26)



```
sum1, sq_sum, sq_avg variables
sum1=0.000
sq_sum=0.000

open(unit=0, file='ck.dat')
open(unit=9, file='scatter.dat')
write(*,*) "How many rand. nos. do you want to use?"
read(*,*) n

! This section computes the mean and the standard deviation
do i=1,n
  call random_number(p)
  sum1=sum1+p
  sq_sum=sq_sum+(p*p)
end do

avg=sum1/real(n)
sq_avg=sq_sum/real(n)
sigma=sqrt(sq_avg-(avg*avg))

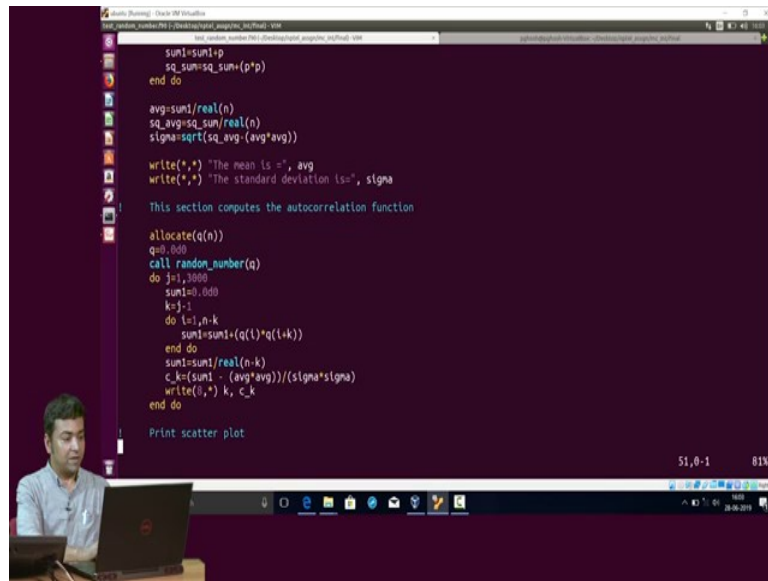
write(*,*) "The mean is ", avg
write(*,*) "The standard deviation is", sigma

! This section computes the autocorrelation function
allocate(q(n))
q=0.000
call random_number(q)
```

So, this particular section what it does is, it will compute for me the average and the standard deviation. So, what it does is it will these first two loop what it will done do is it will call the random numbers n random numbers that depends on the value of n and these random numbers are I sum them up and store in this variable and then I also take the square of the random numbers and sum them up and store in this variable.

Once that is done what I do is the average is computed by dividing the sum the number stored in sum1 by the number of times or the number of random numbers I used that is n. And, then similarly I compute the I take I compute the average of the square sum that is given by square underscore sum divided by real n and once that is done I get the sigma using this formula and then it prints out the average value and then the sigma.

(Refer Slide Time: 23:34)



```
sum1=sum1+p
sq_sum=sq_sum+(p*p)
end do

avg=sum1/real(n)
sq_avg=sq_sum/real(n)
sigma=sqrt(sq_avg-(avg*avg))

write(*,*) "The mean is =", avg
write(*,*) "The standard deviation is=", sigma

This section computes the autocorrelation function

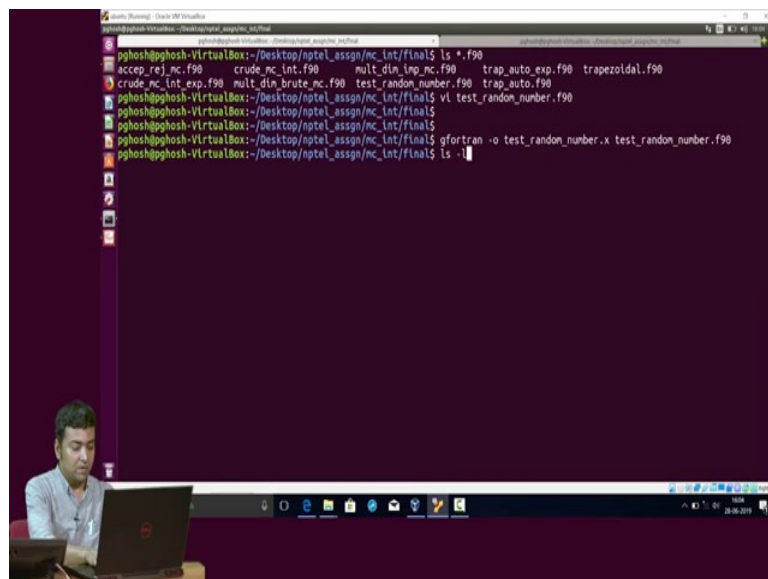
allocate(q(n))
q=0.000
call random_number(q)
do j=1,3000
sum1=0.000
k=j-1
do l=1,n-k
sum1=sum1+(q(l)*q(l+k))
end do
sum1=sum1/real(n-k)
c_k=(sum1 - (avg*avg))/(sigma*sigma)
write(0,*) k, c_k
end do

Print scatter plot
```

So, once that is done, in the next part I compute the co-relation function. So, if you refer to the lecture notes the co-relation function is given by it again involves a sum over n and that is what I am doing here. And, this part of the code computes the co-relation function and this now I am using q number of I using a different set of random numbers to do that.

And, then again I use the same set and to write the data in such a way I write I plot i versus i plus 1-th random how the plot looks like. So, that is what is my scatter plot.

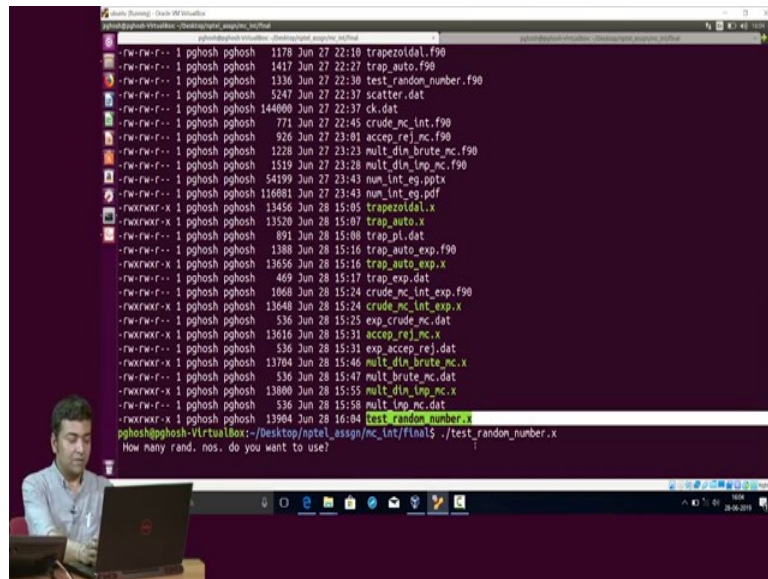
(Refer Slide Time: 24:11)



```
gghosh@gghosh-VirtualBox:~/Desktop/nptel_assign/mc_int/final$ ls *.f90
accep_rej_mc.f90      crude_mc_int.f90      mult_dim_inp_mc.f90  trap_auto_exp.f90    trapezoidal.f90
crude_mc_int_exp.f90 mult_dim_brute_mc.f90 test_random_number.f90 trap_auto.f90
gghosh@gghosh-VirtualBox:~/Desktop/nptel_assign/mc_int/final$ vi test_random_number.f90
gghosh@gghosh-VirtualBox:~/Desktop/nptel_assign/mc_int/final$
gghosh@gghosh-VirtualBox:~/Desktop/nptel_assign/mc_int/final$
gghosh@gghosh-VirtualBox:~/Desktop/nptel_assign/mc_int/final$ gfortran -o test_random_number.x test_random_number.f90
gghosh@gghosh-VirtualBox:~/Desktop/nptel_assign/mc_int/final$ ls -l
```

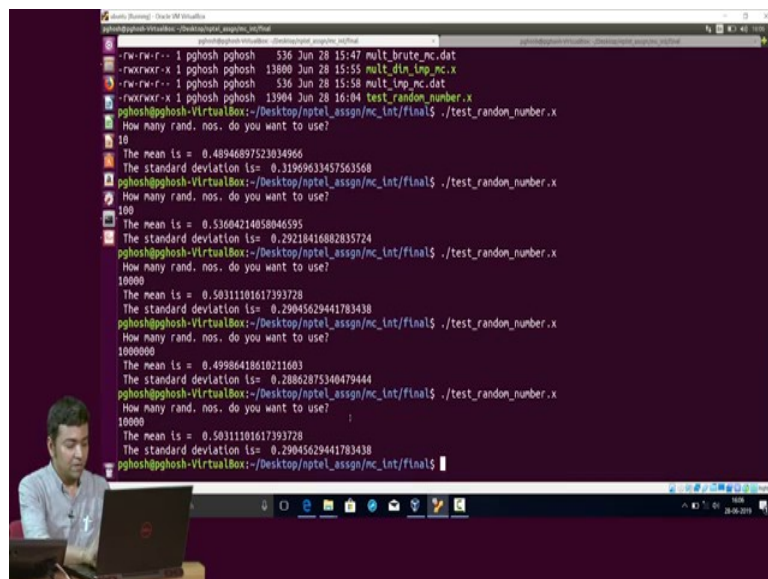

So, let us compile this. So, it is done.

(Refer Slide Time: 24:32)



And, then if you see do ls minus ltr we have this executable to run it. So, we run this. So, let us start with a very small set of random numbers say 10. So, we know that the average value of a uniform distribution is 0.5 and the standard deviation sigma is 0.2886. So, let us see what values we got.

(Refer Slide Time: 24:59)

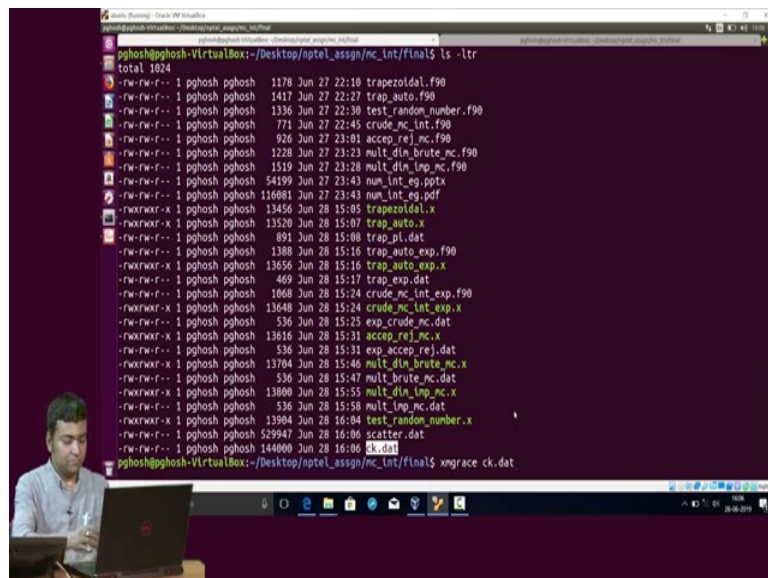


So, you see with 10 random numbers here the value of the mean is quite far away from 0.5; similarly, the standard deviation is also quite far away from 0.2886. Now, if I

increase the number of random numbers, say it is from 10 I go to 100. So, you see it has still away from both the values are still away from the exact value, now we make it say 10000. Now, you see it is slowly gradually going towards 0.5 and this is gradually going towards 0.2886. Now, if I increase it further. So, then you see it is much closer to that value. So, it is taking time because it is computing the correlation function also .

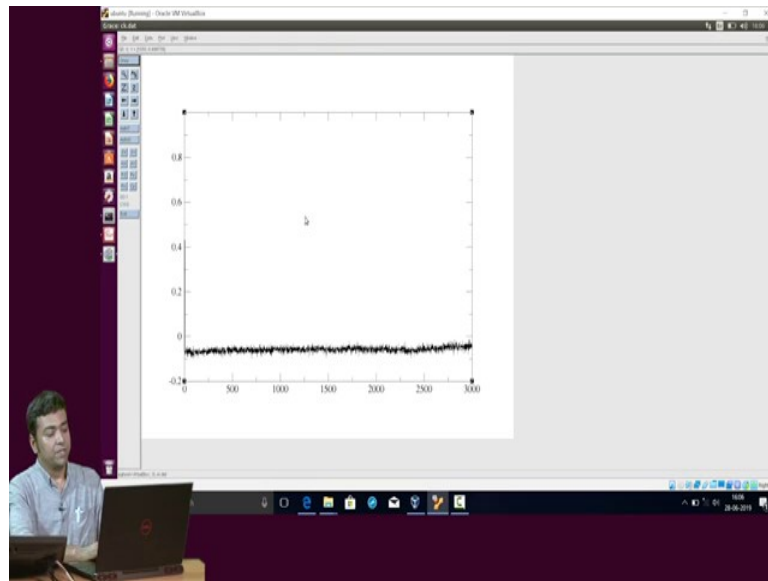
Now, let us look at what happens to the co-relation plot. So, let us do it with first say 10000.

(Refer Slide Time: 26:21)



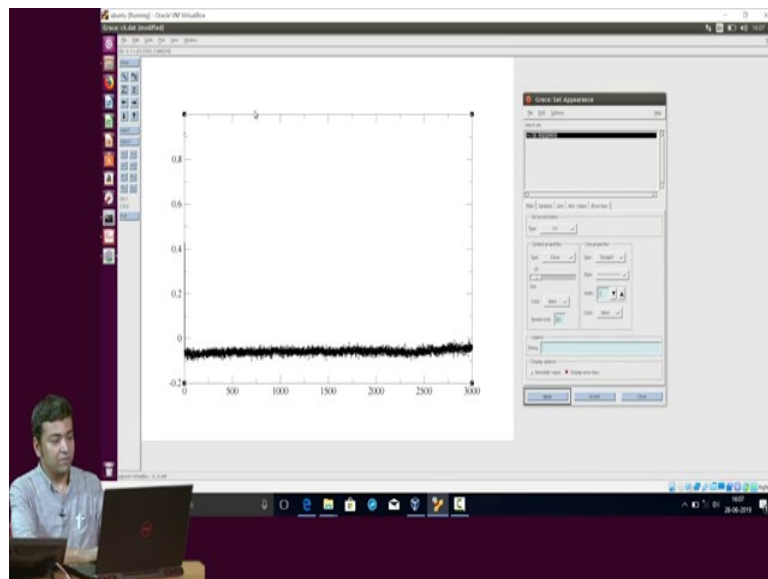
So, if I run it then this is the data is stored here. So, you can use any plotting software you want. So, in my machine I have xmgrace. So, I will use xmgrace to plot it ok.

(Refer Slide Time: 26:39)



So, this is what my data will look like.

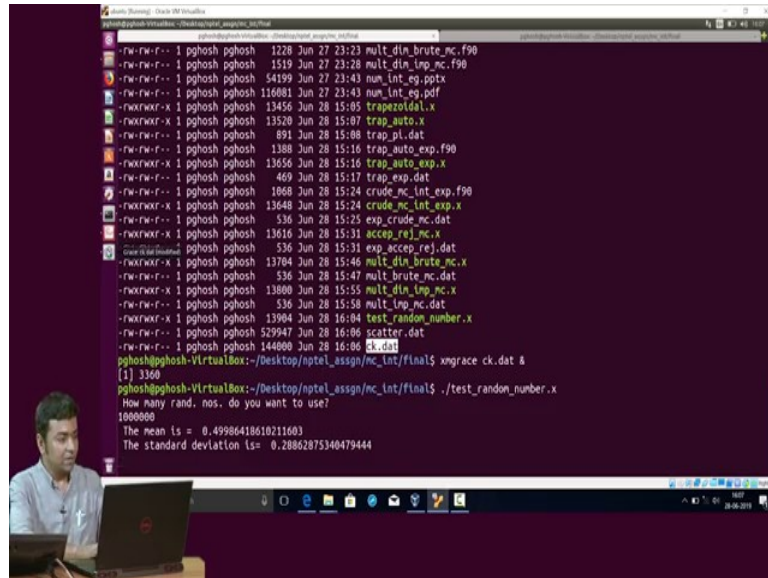
(Refer Slide Time: 26:47)



So, if I instead of if I put the just the symbols here along with the lines so, at k ; so, in the x-axis I have k and in the y-axis I have ck . So, for k equals to 0 it is nothing, but the auto co-relation and it should be one. So, you should expect a point here and it somewhere which is close to 1. The reason it is not going to 1 is because the number of points are very few.

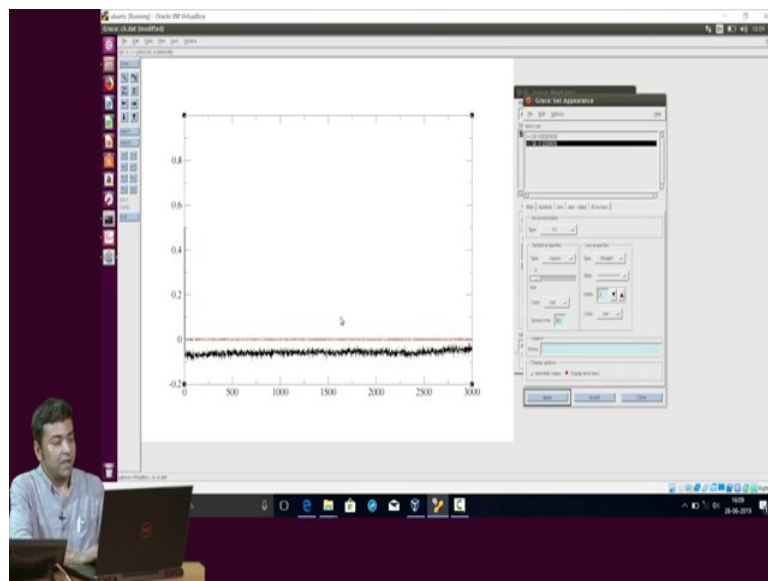
Now, let us try to run it again with a few more points and see how the things are changing.

(Refer Slide Time: 27:25)



So, what we do here is, so instead of 10,000 so, rather than that so, I will put 10 lakh part and let it run. So, once it is done what I will do is I will plot it in this one only. Let me try to plot it in this graph see it is done.

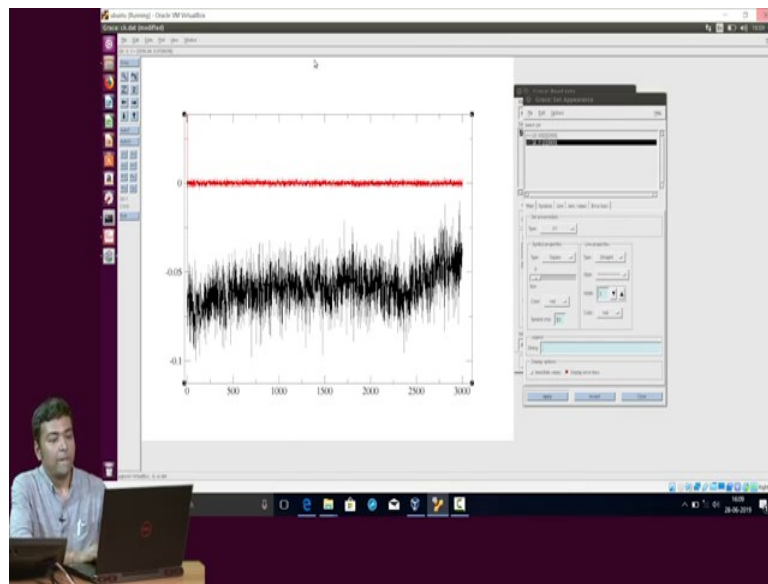
(Refer Slide Time: 27:51)



So, let me just try to reload the data ok. So, the two things to notice here ok, let me just put some make this 25. This is not. So, the two things to notice so, if so, ideally if there is

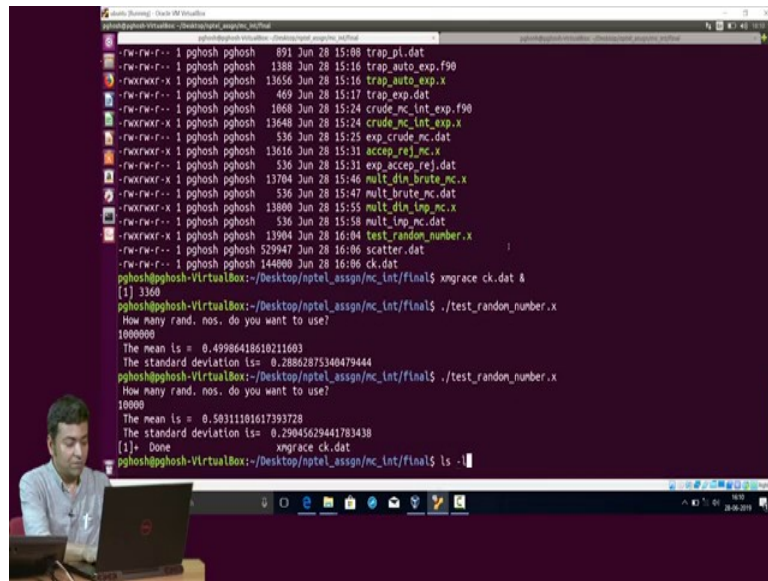
a completely random the numbers are completely random you expect the co-relation to be zero. So, what you would expect is that these numbers this plots will hover around zero, but when we have a few numbers for the co-relation plot what we see is that it is having a certain non-zero value. And, the moment you increase the number of random numbers that goes into the evaluation of the co-relation plot you see that it starts hovering around zero nicely.

(Refer Slide Time: 29:27)



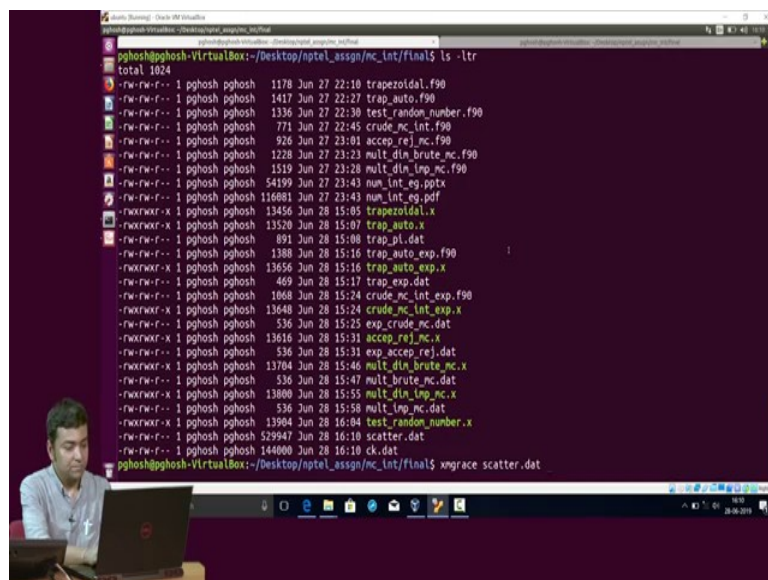
So, this also tells you another thing that in addition to that in these type of methods. So, what is also very important is the average. So, one needs to do a very good average I mean good average I mean average over sufficient to large number of points to get into some meaningful and realistic number.

(Refer Slide Time: 29:51)



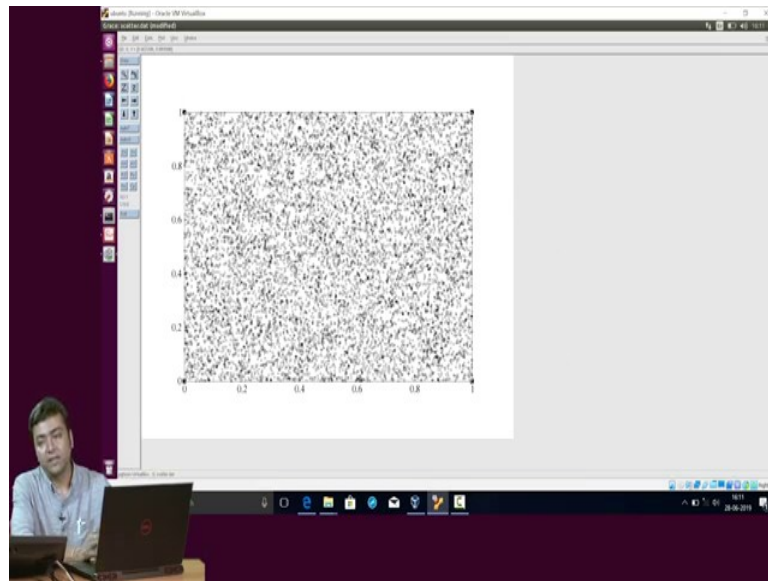
Now, we will also look at the scatter plot. So, for the scatter plot what we will do is we will use fewer random numbers, so that you can see. So, let us put say 10,000 random numbers and do the evaluation.

(Refer Slide Time: 30:03)



So, let us open it in xmgrace.

(Refer Slide Time: 33:10)



So, if your numbers are co-related then what your scattered would have look like? Let me just first make this plot proper. So, circle slightly straight line I do not want, I just want to see the dots here. So, this is how it you see it is I mean the points are spread all over this area from x-axis is from 0 to 1 and y axis again is from 0 to 1. Had there been co-relations, then what you would have seen is that certain parts of this area is covered while the other parts are empty or it would have been something like a diagonal one. So, this says since in all the points are scattered all over the space, all over the extent of my plot this tells that the numbers are not co-related and it is genuinely a good random number.

There is one more test which I am not going to show it to you and I leave it to you as exercise that is you what you can do is you can generate say a million random numbers divide them into small bins and once you have them in small bins you can plot a histogram that is in each bin how many random numbers are there. So, and what you would see is that the height of the histogram and of all of them are same that is what it tells you that is a uniform distribution because each random number has an equal probability of there is an probability of finding each random number in a uniform distribution.

So, if you use very few random numbers in plotting the histogram then you will you will find that some bin heights are larger, some bin heights are smaller, but then gradually as

you increase the number more and more as you increase the more and more random numbers that is as you enhance or better the sampling what you will find is that the heights of all the bins are same.