Algorithms for Protein Modelling and Engineering

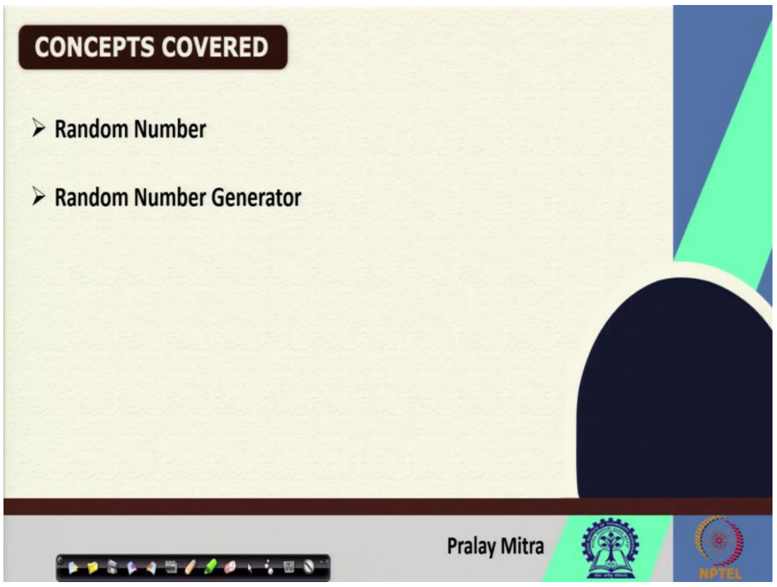
Professor Pralay Mitra
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture 17

Monte Carlo Method (Contd.) and Random Number

Welcome back, so we are discussing basically the Monte Carlo method and its application on the computation on Pi and in this lecture actually we will see the drawback or limitation of the Monte Carlo method if our random number generation is not that much random in nature, so that way we have to be careful that we are generating the random number which is purely random in nature or as much random as possible.

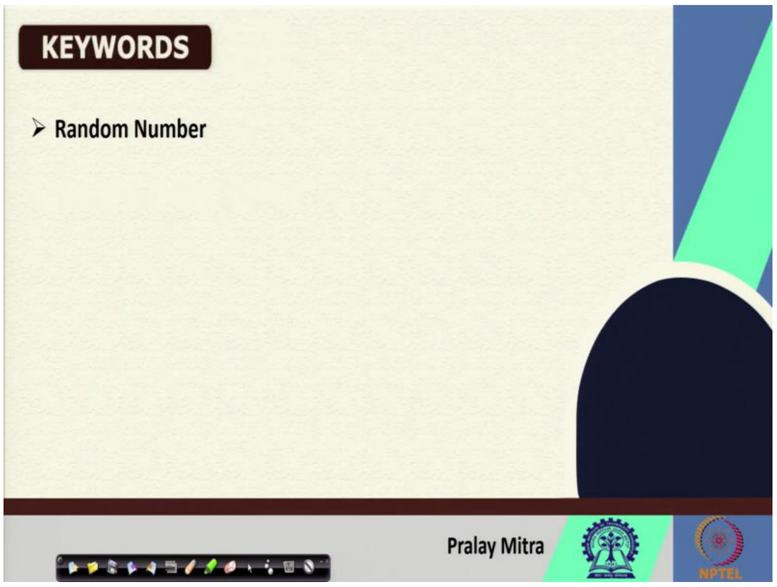
(Refer Slide Time: 00:48)



CONCEPTS COVERED

- Random Number
- Random Number Generator

Pralay Mitra



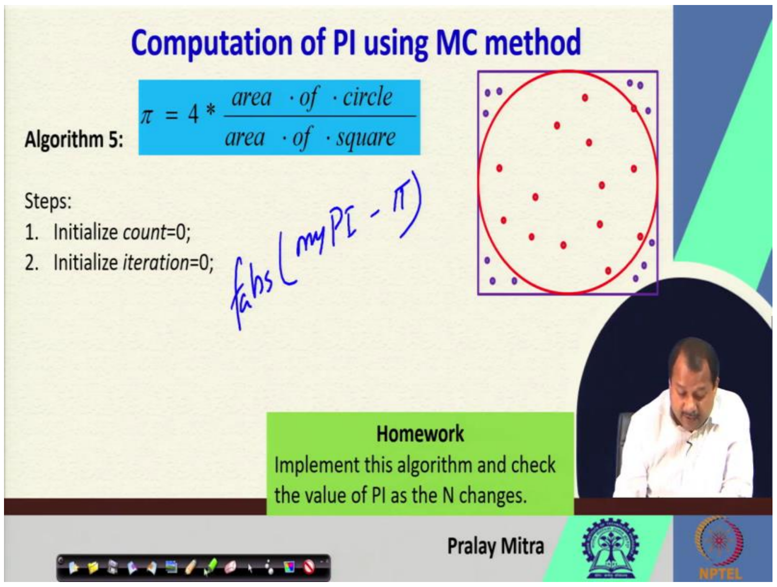
KEYWORDS

- Random Number

Pralay Mitra

So, the content we will be covering is this application of the Monte Carlo and the random number generation the random number generator, so that is our keyword also that random number.

(Refer Slide Time: 01:04)



Computation of PI using MC method

Algorithm 5: $\pi = 4 * \frac{area \cdot of \cdot circle}{area \cdot of \cdot square}$

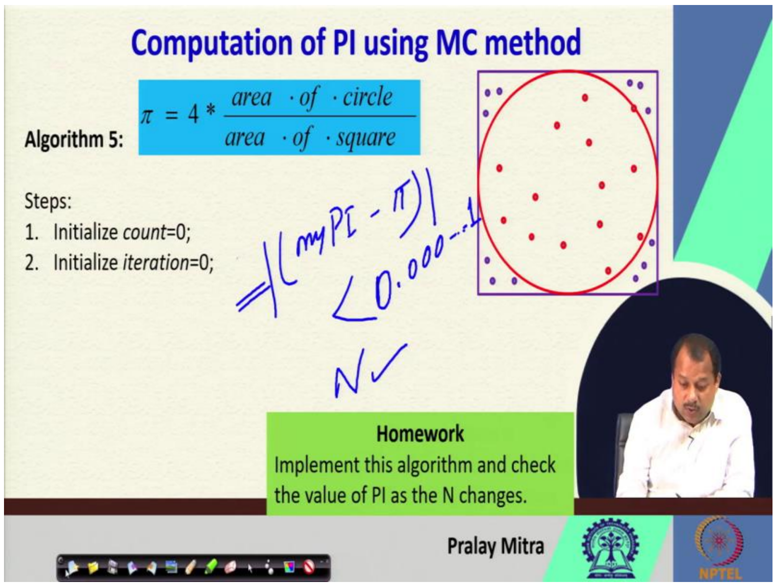
Steps:

1. Initialize *count*=0;
2. Initialize *iteration*=0;

$fabs(myPI - \pi)$

Homework
Implement this algorithm and check the value of PI as the N changes.

Pralay Mitra



Computation of PI using MC method

Algorithm 5: $\pi = 4 * \frac{area \cdot of \cdot circle}{area \cdot of \cdot square}$

Steps:

1. Initialize *count*=0;
2. Initialize *iteration*=0;

$|(myPI - \pi)| < 0.000\ldots1$

N ✓

Homework
Implement this algorithm and check the value of PI as the N changes.

Pralay Mitra

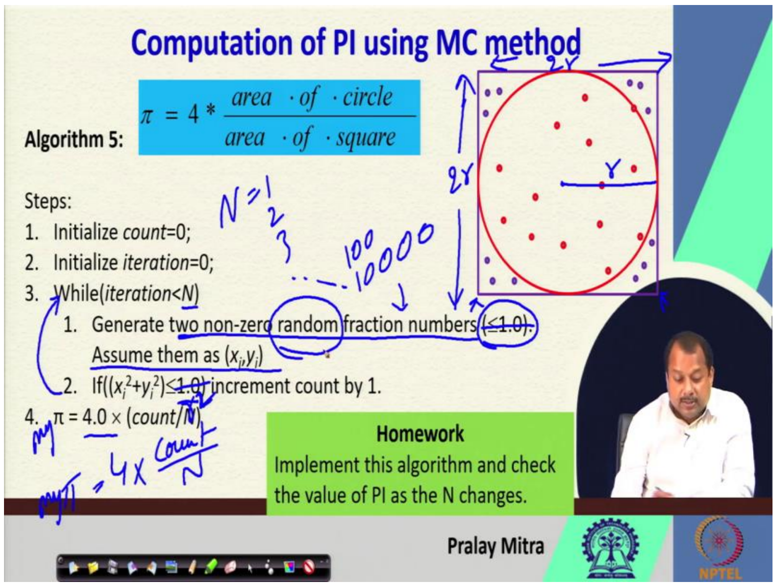
So, I promised on the last lecture that I will give you 1 algorithm that you can able to implement using any language and can have an idea of Monte Carlo simulation technique, so the theory behind this computing the Pi is Pi equals to 4 multiplied with area of circle divided by area of the square and the diagram is given on the right hand side which indicates actually there is a square and we are considering the circle whose diameter is same as the side of the square then only we can write this equation that Pi equals to 4 multiplied with area of circle divided by area square.

Now, the algorithm that we are going to give you, this is going to be your homework, so you have to be really careful while listening to this, so the steps will be initialised count equals to 0 similar to our last slide on the last lecture that time it was s equals to 0, so initialise you some count as 0 then, initialise iteration also as 0.

So, there can be 2 ways for the termination of the iterative process as you understand this is going to be an iterative process, one is that you know beforehand how many iterations will be there which means, N or the number of points that you are going to generate is known to you. Another situation is that, that is not known to you, but somebody says that the value or the accuracy of the Pi after Nth decimal, I mean the Nth position after the decimal point will be correct.

Then your termination criteria will be something like, you are calculating the value of Pi, so my Pi and then actual Pi if I say fabs or without the mathematical function actually, if I just write the modulus operation for this I mean I am interested to know only the absolute difference of this if it is within some limit that limit will be known to me 0.0000 some 1 something like, so that may be one termination point of the iteration of the loop, another is N, so where I know N number of points I am going to generate.

(Refer Slide Time: 04:04)



Computation of PI using MC method

Algorithm 5: $\pi = 4 * \frac{area \cdot of \cdot circle}{area \cdot of \cdot square}$

```
Steps:
1. Initialize count=0;
2. Initialize iteration=0;
3. While(iteration<N)
    1. Generate two non-zero random fraction numbers ~~(≤1.0)~~.
       Assume them as (x_i, y_i)
    2. If((x_i^2+y_i^2)≤~~1.0~~) increment count by 1.
4. π = 4.0 × (count/N)
```

$N = 1, 2, 3 \ldots 100, 10000$

$my\pi = 4 \times \frac{count}{N}$

Homework
Implement this algorithm and check the value of PI as the N changes.

Pralay Mitra

Now, you know that instead of this first one the second one is preferable at least when you are designing fast, so let us focus only on the N which means that you know that how many points you are going to generate, so if it is the case then after initialising count and the iteration then y number of iteration is less than N, I mean N is known to me and you know that what is the value of N or how many points you have to generate.

So, you have to generate two nonzero random fraction numbers, regarding this fracture numbers say less than 1.0 it is up to you why, I will come to that one later and assume them as xi, yi if that you assume that xi and yi then, if x square plus y square less than a 1.0 no, this will not be 1.0, so if two fraction numbers you generated less than 1 which means it will vary from here to here, then this r square will be actually this 1 divided by 2 which means 0.5 whole square.

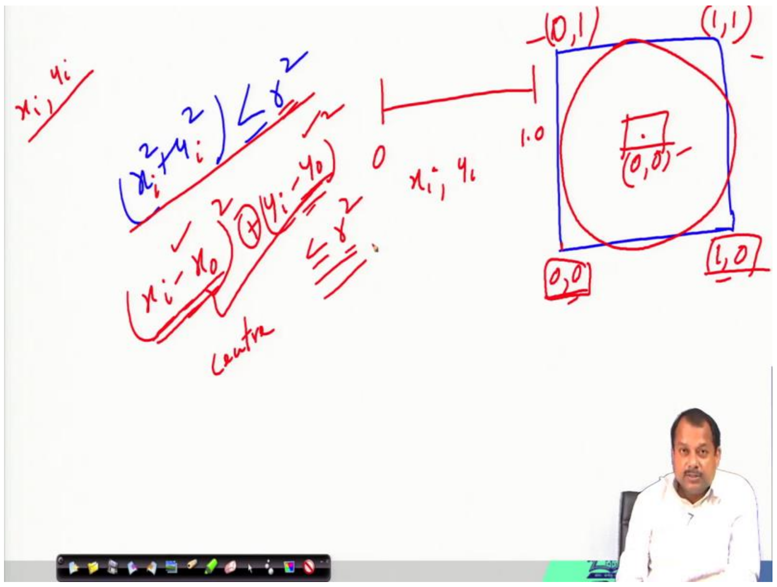
So, that is why what we can do that for the time being, let us not mention this one and here we write r square, where we know that this is my r and we also know that this is my 2r this is also my 2r, this is computational tricks that I will come later.

Now, if x square plus y square less than r square, r is the radius of the circle, then increment count by 1 after that one what we will do that this process will keep on moving, so this is a loop it will go here and the loop will keep on iterating until this N is reached, when N will be reached, then it will come out of the loop and it will count what it will count that 4 multiplied with count divided by N, so here it is N.

That is basically my Pi, so instead of this Pi you can write also my Pi, simple algorithm easy to implement consider this as your homework you can implement to check the value of Pi as the N changes, so you can give N as an input you can supply N as a 1 2 3 and then 100 then 10,000 then keep on increasing that value and you see that how it is changing.

Now, you have to make sure that random number is purely random in nature otherwise, there will be some bias, why that I will come later, but before that I wish to tell you one simple implementation tricks and that is look what you are doing here?

(Refer Slide Time: 07:41)



Basically, x square you are computing x square plus y square less than equals to r square. Now, you have one square and inside that square you have one circle, the point you are generating xi and yi randomly you are generating, now after generating that one this xi yi r is going to be fixed since r is going to be fixed, so you can mean some that what is the range say 1.0 within that one whether, I will generate xi and yi that way you are assuming this is your 0,0 this is your 1,0 this is your 0,1 and this is your 1,1 that is my quad that you are assuming.

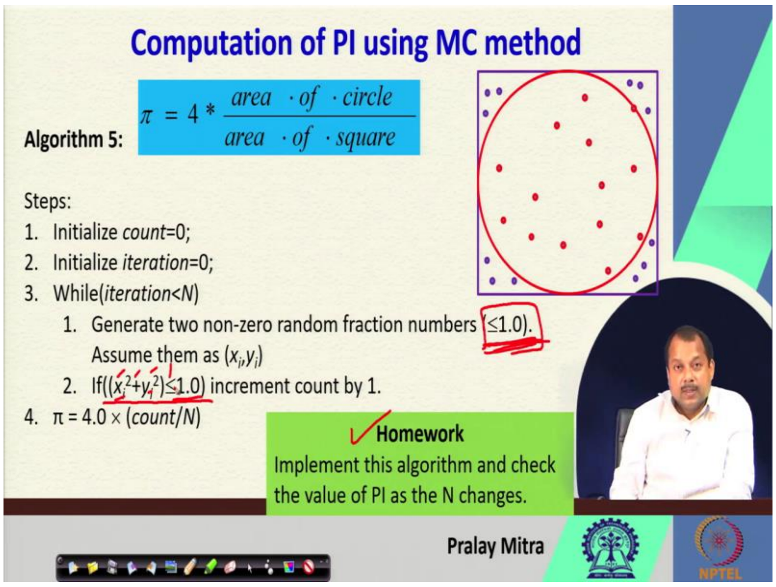
Another way is that if you assume that this is my origin and if you assume this is my origin that way actually this equation is xi minus x0 plus yi minus y0 less than equals to r square, so this x0 and y0 is the centre, so this is the centre of the circle.

And now you see that 1 subtraction a 2 subtraction this will be square, this will be also square, so 2 subtraction is there regarding this r square, so you know r once you define what is the side of the sum square, what is the radius of the circle you know r that you can pre-compute and store that one, so instead of computing this r square always, what you can do that you can pre-compute that one and use that value always as if a constant value, that way you can reduce one multiplication in each iteration, that is one thing.

Another you can think, I am not going to give you the solution right now, since I mentioned that it is going to be your homework, that how you can reduce this part also, so what will be your assumption regarding the point here, point here and point here these 3 will also tell what 2 others point will be.

Accordingly, if you can decide this, so this one, this one and this one then definitely you can able to decide what will be the value of r also that way you should try to reduce this subtraction also at each step and that way, if you can proceed then after generating the number, so what you can do directly you can compute the square, add them and check them, so one multiplication, one multiplication here, another multiplication here, one addition here and after that one comparison statement that is it, so you keep the computational time minimum by doing this process.

(Refer Slide Time: 11:22)



Computation of PI using MC method

Algorithm 5: $\pi = 4 * \frac{area \cdot of \cdot circle}{area \cdot of \cdot square}$

Steps:

```
1. Initialize count=0;
2. Initialize iteration=0;
3. While(iteration<N)
    1. Generate two non-zero random fraction numbers (≤1.0).
       Assume them as $(x_i, y_i)$
    2. If$((x_i^2+y_i^2) \leq 1.0)$ increment count by 1.
4. $\pi = 4.0 \times (count/N)$
```

Homework

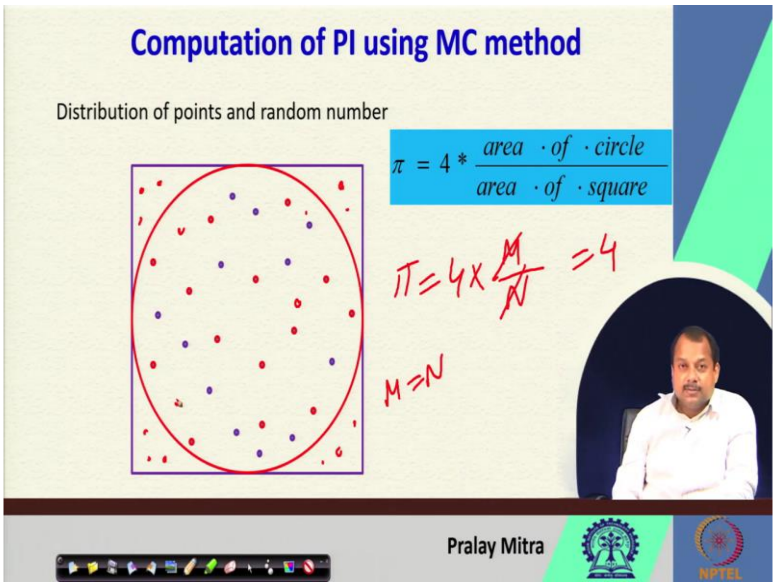
Implement this algorithm and check the value of PI as the N changes.

Pralay Mitra

So, here in this case, so there is a typo here, so this will not be less than 1.0 but I am not giving you the solution, since I mentioned already that it is going to be your homework, so you fix this, so this 1.0 what will be it, so that you can use this xi square plus y square less than equals to 1.0.

And if you can use this then you see that this one multiplication, another multiplication, another addition, another logical operation that is it, if you wish to make this change that may not be a problem because this will be constant throughout all the iterations, but this will be costly and xi minus x0, yi minus y0 is going to be even costly, so you have to be careful regarding that one, so you fix this typo and then implement this as your homework.

(Refer Slide Time: 12:30)


Computation of PI using MC method

Distribution of points and random number

$$\pi = 4 * \frac{area \cdot of \cdot circle}{area \cdot of \cdot square}$$

$$\pi = 4 \times \frac{\cancel{M}}{\cancel{N}} = 4$$

$$M = N$$

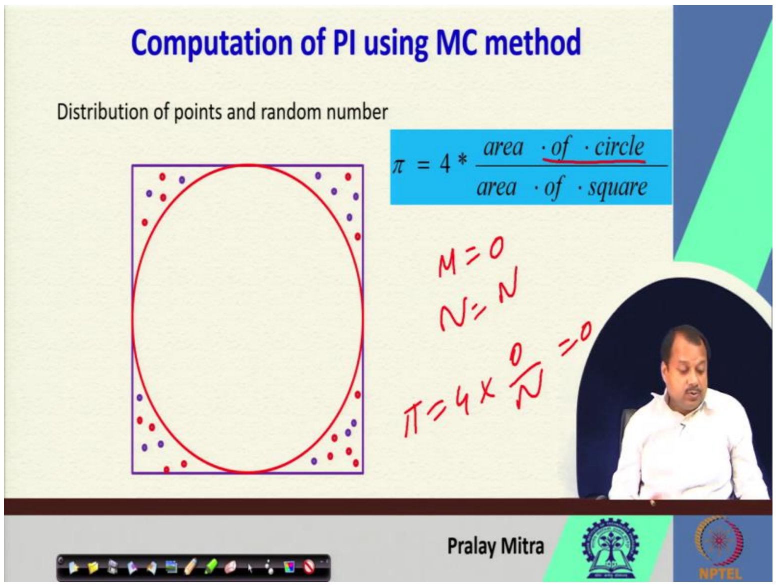
Next point is the randomness of the random number and I mentioned it matters, using one example say when n equals to 2, if both the points are inside the circle or both the points are outside the circle, how it will affect that I have mentioned, here you can see one such example.

you see, I generated a number of points all the points are inside the circle, if they are inside the circle, then you can understand in this equation Pi is going to be area of the circle, so number of points M and the number of points area of the square N and by this diagram, I can see that M is going to be N, so it will cross out so Pi is going to be 4 although the number of points are very high then also I am getting this.

This is because the random number that I have generated or the distribution of the random number is such that it is always inside the circle not outside the circle if you are (())(13:43) probably the distribution is correct, then you have to generate probably the more number of points, so two points to note which will also be useful for Monte Carlo based techniques in protein folding and protein engineering.

One is that during its iteration, you have to generate a random number, make it as much random as possible even with that one if you are unable to achieve the required solution, then you have to increase the number of iterations, I agree with you that by looking at this I may think that distribution of the point is not that much bad, but if I generate few more points, then probably some will go outside the circle then you will generate those, so which means that you increase the size of the N on the number of points, these number of points are not enough, so you explore the search space that is exactly what we will also do when we will go for the protein folding or protein engineering.

(Refer Slide Time: 14:56)



Computation of PI using MC method

Distribution of points and random number

$$\pi = 4 * \frac{area \cdot of \cdot circle}{area \cdot of \cdot square}$$

$M = 0$

$N = N$

$\pi = 4 \times \frac{0}{N} = 0$

But, how do you justify this kind of situation? If there is any bias less, then this will occur and it purely says that, there is some bias in your random number generation that is why it occurs and because of this occurrence, what will happen the area of the circle, your M is going to be 0, your N is N now, so Pi is 4 multiplied with 0 by N which is 0, which is not the correct result.

So, Monte Carlo technique will give you the correct result, that is why I also insisted that you please go for implementing this one, it will give you correct result if the number of iterations are high or good enough and your random number is sufficiently random in nature, so that no two points are coinciding.

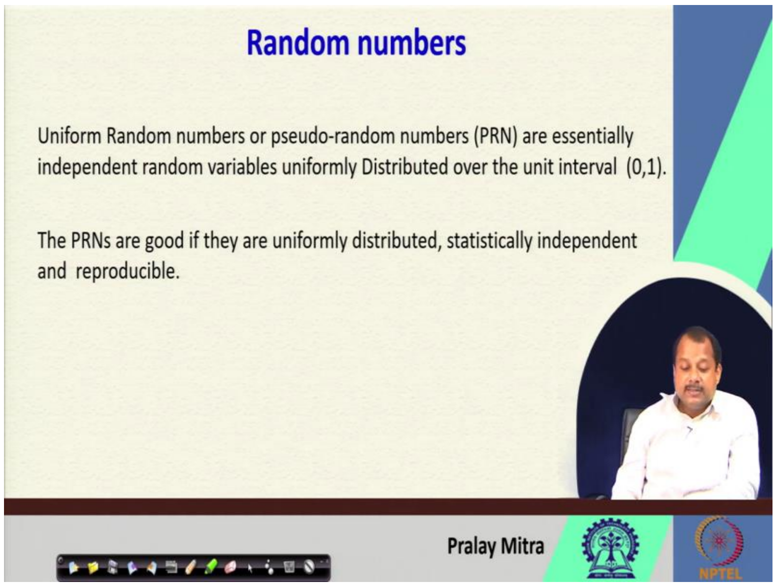
If two points are coinciding, so they are biasing basically, so they are not basically contributing to the area of the new area of the circle or square or both, because actually by this point, I am calculating the area of the circle and area of the square then I am taking their ratio and multiplying the 4. Now, if two points are generated at the same position, so the second point is not contributing to a new area, although it is contributing to the area of the square, because I mentioned that the number of points I am generating is going to be the area of the square.

But, if you have any mechanism that if two same points are generated, then discard that point, then it is fine otherwise, that can be a problem, but discarding that one will again be costly, why you have to generate those points, then you have to test those points for their coincidence and then you have to discard that one that is a costly affair and I will prefer to avoid that one, so what is the solution that is why we have to look into the random number generation.

I know that there are several library functions for generating random numbers, so there are seed value you mentioned a seed value and based upon that seed value, it generates some random number, some of the random number generators actually uses the system clock in that case, the problem will be if your technique is very fast, then within a fraction of second if you are calling the random number, which depends on the system time then basically that will create some problem.

If it uses some hardware information like temperature or say other characteristic information, then perhaps it is going to be good, but again if I call that function frequently and there is not much difference between two function call then there is a very less chance that the parameter on which I am relying like system time or hardware feature may be the same between two consecutive calls, so we have to be sufficiently random in nature.

(Refer Slide Time: 18:23)



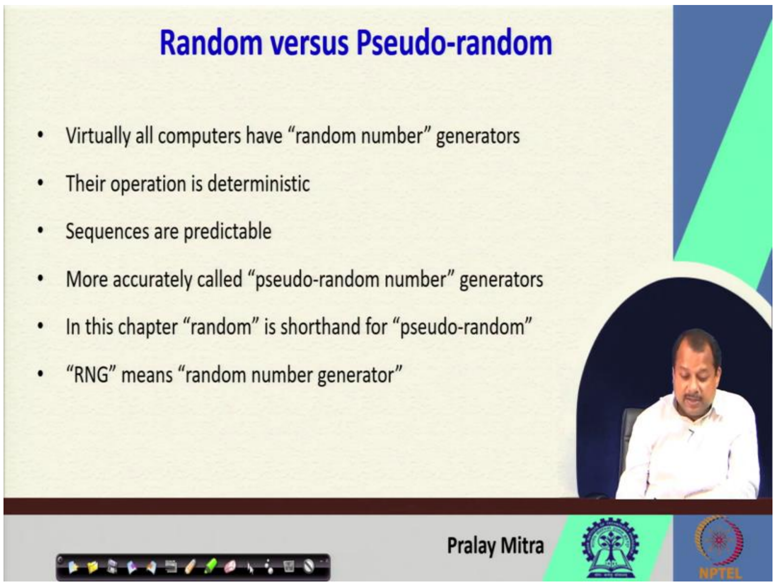
Random numbers

Uniform Random numbers or pseudo-random numbers (PRN) are essentially independent random variables uniformly Distributed over the unit interval (0,1).

The PRNs are good if they are uniformly distributed, statistically independent and reproducible.

Usually, uniform random numbers or pseudo-random numbers are essentially independent and independent random variables, uniformly distributed over the unit interval 0 and 1. Now, you see that 0 and 1 that can be actually scaled for N number that you are interested in. The PRN or pseudo-random numbers are good if they are uniformly distributed statistically independent and reproducible.

(Refer Slide Time: 18:53)



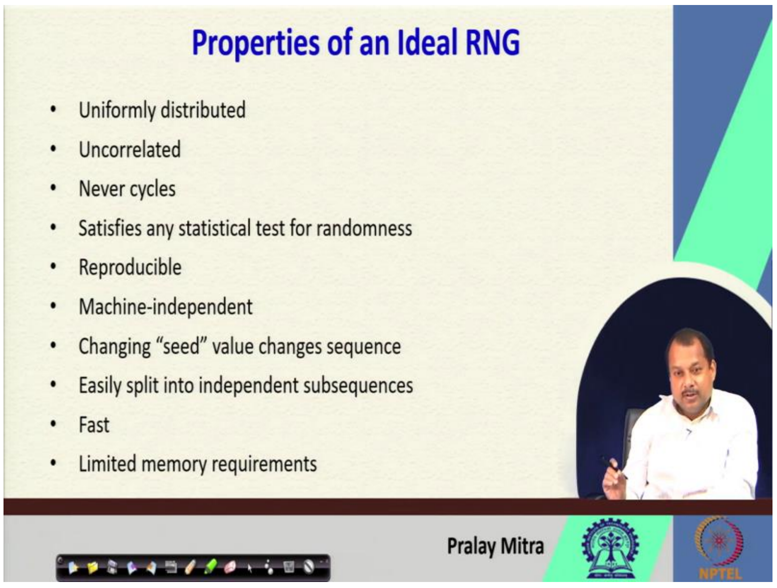
Random versus Pseudo-random

- Virtually all computers have "random number" generators
- Their operation is deterministic
- Sequences are predictable
- More accurately called "pseudo-random number" generators
- In this chapter "random" is shorthand for "pseudo-random"
- "RNG" means "random number generator"

Pralay Mitra

Random versus Pseudo-random number virtually all computers have random number generators, their operation is deterministic, sequences are predictable, more accurately called pseudo random number generators, and in this case, random is shorthand for pseudo-random. So, it says that what we are actually talking is not purely random number, but pseudo random number and RNG means random number generator.

(Refer Slide Time: 19:35)



Properties of an Ideal RNG

- Uniformly distributed
- Uncorrelated
- Never cycles
- Satisfies any statistical test for randomness
- Reproducible
- Machine-independent
- Changing "seed" value changes sequence
- Easily split into independent subsequences
- Fast
- Limited memory requirements

Pralay Mitra

So, the properties of the ideal random number generator is, it is uniformly distributed and you know that, if it is uniformly distributed for the calculus of the Pi, then it will cover the entire surface of the square using the exactly same number of points which you required and there we no coincidence, it will be uncorrelated there should not be any correlation no patterns will be found there, it never cycles, so after each interval it will keep on coming, so it will not occur like that way it satisfies any statistical test for randomness.

Then, it is reproducible, it is missing independent again the problem again will be that I just mentioned to you that sometimes it depends upon the hardware for the random number generator now, if it depends upon the hardware and the features you are implementing is not compatible from one system to another system then it might create a problem say when you are simulating in one system and then moving to another system, because whatever programme you have written it is supposed to be executed in all the systems, it will not change in other system.

That way if you do then say you wrote some Monte Carlo simulation based protein folding technique in one desktop system it is giving you some result and in another say workstation or server it is giving another result and you will be clueless, your algorithm is same, your programme is same, your score function is same, everything is same, but what is the difference. So, it should be machine independent also.

Changing seed value changes sequence that must be there, so that at different time when you say start executing, so you just change a seed value at the beginning again that seed value change you can also do randomly and based upon that one the sequence will also change and easily split into independent sub sequences, because there is no pattern, there is no cycle, they are uniformly distributed, so if I chop it into different sub sequences, so they will be independent from each other, the computation needs to be fast, because as you understand that it is a core requirement of Monte Carlo simulation technique.

But, apart from that one other score function, other calculations will be there for the simplest Pi calculation, we noted that x square plus y square less than equals to r square assuming that the centre is actually 0,0 but, if it is not then it will be xi minus x0, so subtraction, addition, multiplication, that is the simplest, trust me.

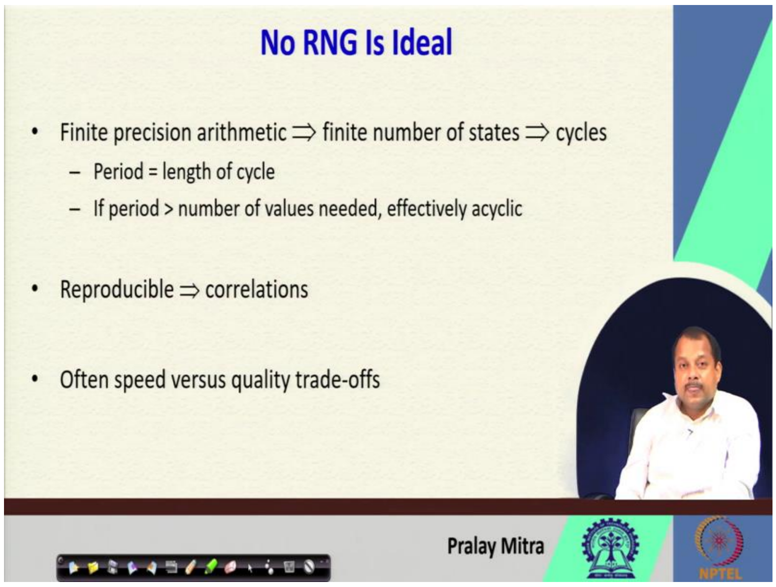
But, when it is in the context of protein folding or protein engineering or protein design, etc, then other score function calculation will also come into play and if that is the situation then I cannot afford too much time on random number generation, so it must be fast and of course, we want that everything must be fast, so not only this one.

And it should not again require the huge amount of memory because other applications, other means that say when Monte Carlo simulations are taking place, then it is basically performing some simulation on some problem.

So, for that problem a lot of features, score value, difference instances or states in order to store that we need a huge amount of memory, so for this random number generation although we understand it is the code of the Monte Carlo simulation, but again we will try to make it such fast, less memory required, but there should not be cycle, there should not be pattern, they should be uniformly distributed, they will be random as much as possible.

So, those are a lot of things we are asking, but we are not ready to give time or memory or storage requirement, so that way if you can design then it will be very beneficial for the Monte Carlo simulation.

(Refer Slide Time: 24:01)



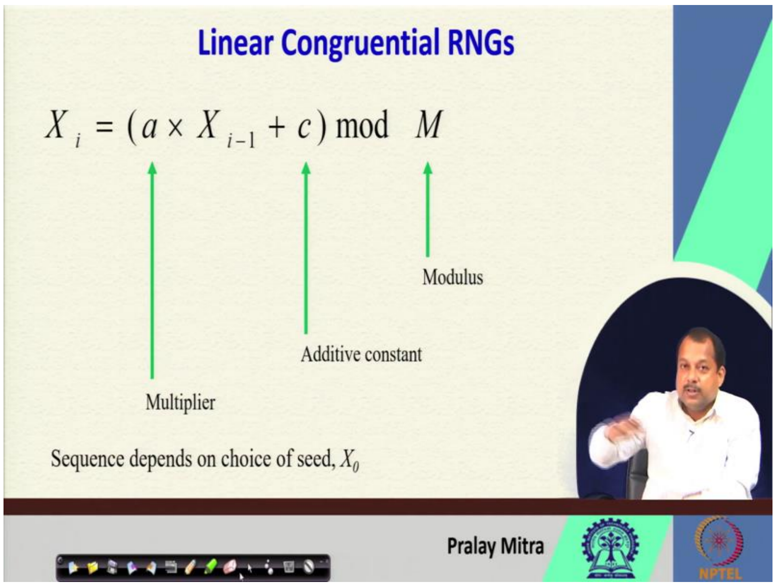
No RNG Is Ideal

- Finite precision arithmetic $\Rightarrow$ finite number of states $\Rightarrow$ cycles
  - Period = length of cycle
  - If period > number of values needed, effectively acyclic
- Reproducible $\Rightarrow$ correlations
- Often speed versus quality trade-offs

Pralay Mitra

So, no RNG is ideal in nature, so in finite precision arithmetic finite number of states, cycles are there, so period is the length of the cycle and if period is greater than number of values needed effectively it is a cyclic, so reproducible means it will be correlations might be there and often speed versus quality trade of is also required.

(Refer Slide Time: 24:28)



Linear Congruential RNGs

$$X_i = (a \times X_{i-1} + c) \bmod M$$

Multiplier
Additive constant
Modulus
Pralay Mitra
NPTEL

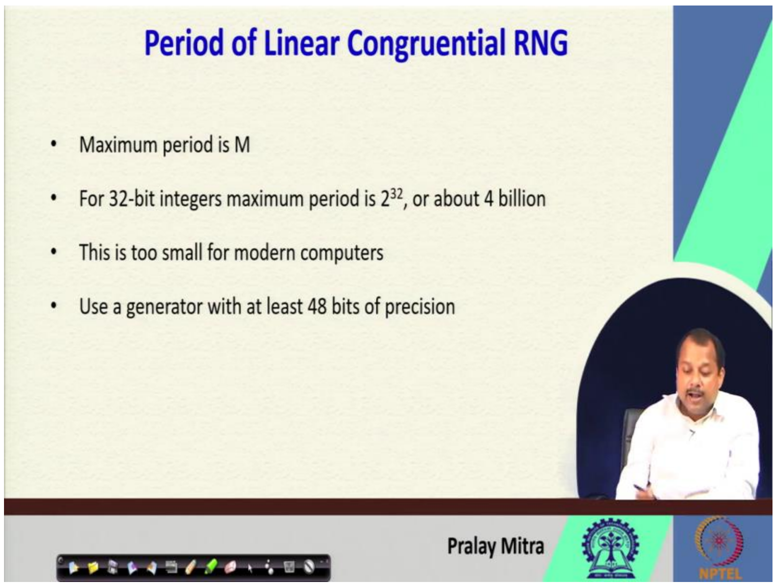
Sequence depends on choice of seed, X_0

Here, is one example of random number generator which is based upon linear congruential technique that is xi equals to a multiplied with x i minus 1 plus c mod M and what is my assumption regarding this mod ca, etc.

So, that I will go but before that, this is a basically the multiplier, so I will multiply some value with the previous random number and after multiplying that value then I will add some constant value and after that one I will go for some modulus operation.

Now, this modulus operation as you understand will tell me some pattern however, if I multiply and then add something, then because of that multiplication and the addition, so that pattern will also change, so that way there is a possibility that I am getting a random number, so sequence depends on the choice of the seed that is the x0 the first one, so you take the first one and based upon that rest will be built.

(Refer Slide Time: 25:50)



Period of Linear Congruential RNG

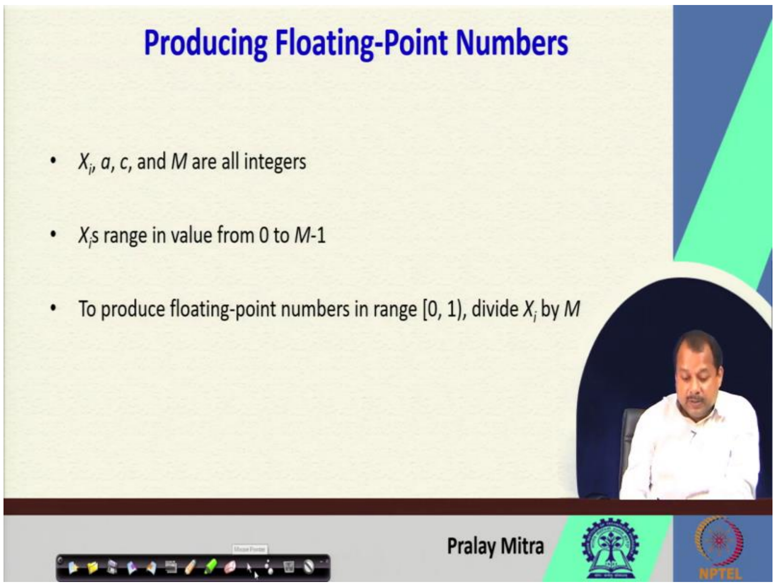
- Maximum period is M
- For 32-bit integers maximum period is 2^{32}, or about 4 billion
- This is too small for modern computers
- Use a generator with at least 48 bits of precision

Pralay Mitra

Period of linear congruential method as you understand since mod M is there, so it will be M maximum period is M now, if M is very large, so you are interested to generate say M number of steps and N number of steps and n is very less or less than M if it is the case then even there is a cycle you are well inside one particular cycle that way it will not be repeated.

For 32 bit integers maximum period is to 2 to the power 32 or about 4 billion, now you see if the period is 4 billion and during your simulation you are not generating more than few millions then 4 billion is much more and that way if there is a pattern or cycle after 4 billion, then also you are not affected this is too small for modern computers. Use a generator with at least 48 bits of precision, so if you wish to increase this cycle of frequency you go for higher precision that is 48 bit or even higher.

(Refer Slide Time: 27:09)



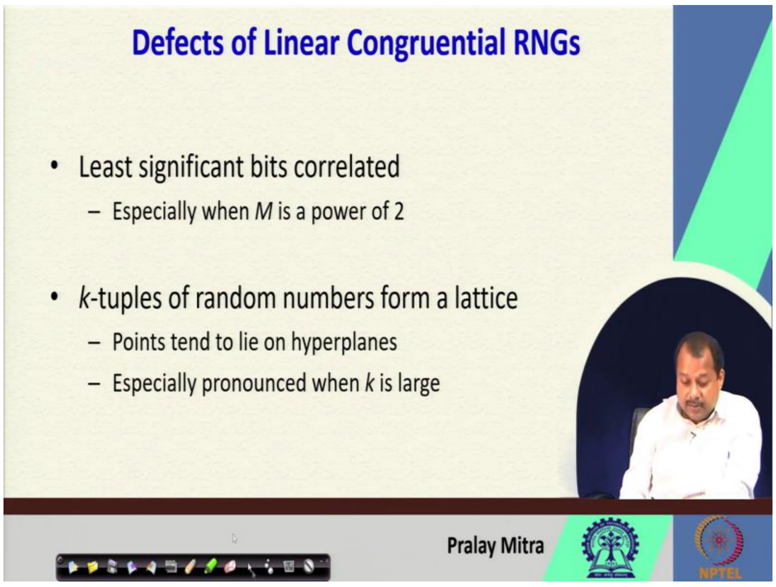
Producing Floating-Point Numbers

- X_i, a, c, and M are all integers
- X_is range in value from 0 to M-1
- To produce floating-point numbers in range [0, 1), divide X_i by M

Pralay Mitra

Or you generate the floating-point numbers, so xi, a, c and M all are integer as of now, so xi is range in value from 0 to M minus 1 and to produce floating point numbers in the range 0 to 1, so it is the open interval divide xi by M and you get that required number.

(Refer Slide Time: 27:40)



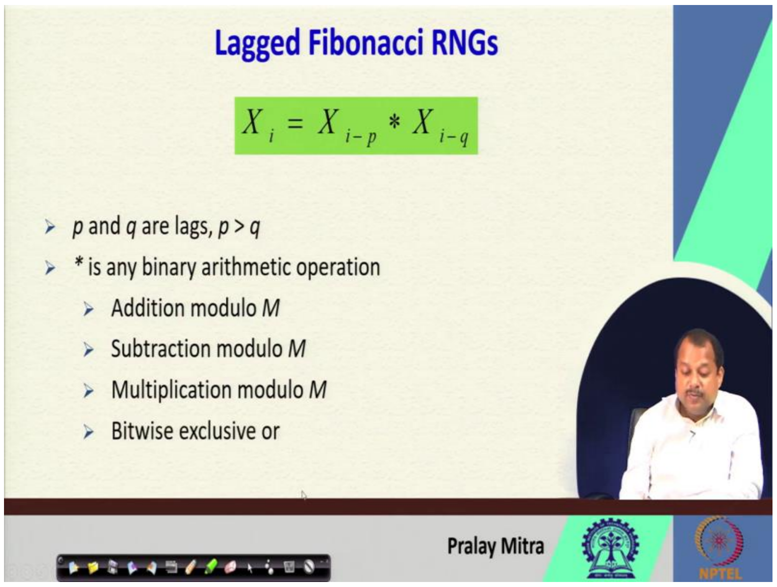
Defects of Linear Congruential RNGs

- Least significant bits correlated
  - Especially when M is a power of 2
- k-tuples of random numbers form a lattice
  - Points tend to lie on hyperplanes
  - Especially pronounced when k is large

Pralay Mitra

Now, there are certain defects also of linear congruential this random number generator, so least significant bits correlated, so if it is a bit then it will be 0 or 1 especially when M is a power of 2, so it will be 0 at 1 so, there will be some correlation, if that is not a problem, then it is fine otherwise, there might be some problems, so be careful about that. k-tuples of random numbers form a lattice, so points tend to lie on hyperplanes, especially pronounced when k is large.

(Refer Slide Time: 28:24)



Lagged Fibonacci RNGs

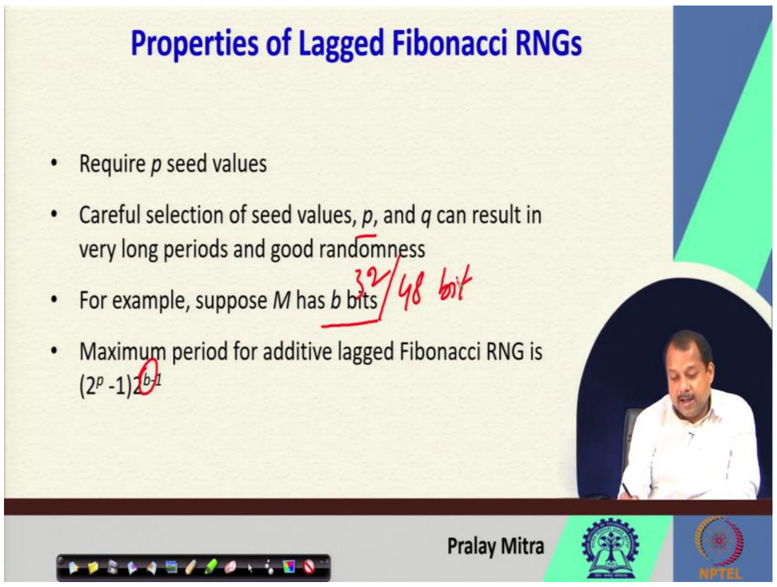
$$X_i = X_{i-p} * X_{i-q}$$

- *p* and *q* are lags, *p* > *q*
- * is any binary arithmetic operation
  - Addition modulo *M*
  - Subtraction modulo *M*
  - Multiplication modulo *M*
  - Bitwise exclusive or

Pralay Mitra

And here it is a lagged Fibonacci random number generator which is xi equals to x i minus p multiplied with xi minus q, so I hope you know what is Fibonacci number, so I am not going into that Fibonacci number, here P and Q are lags assuming p is greater than q and multiplication is in binary arithmetic operation, then addition, so this actually the star it can be multiplication, it can be other arithmetic operations, so some suggestions are like addition modulo M then subtraction modulo M, then multiplication modulo M, then bitwise exclusive or, so those are the possibilities.

(Refer Slide Time: 29:18)



Properties of Lagged Fibonacci RNGs

- Require *p* seed values
- Careful selection of seed values, *p*, and *q* can result in very long periods and good randomness
- For example, suppose *M* has *b* bits 32/48 bit
- Maximum period for additive lagged Fibonacci RNG is $(2^p - 1)2^{b-1}$

Now, what are the properties of the lag Fibonacci random number generator? So, first of all it requires p seed values and p greater than q careful selection of seed values p and q can result in very long periods and good randomness that is a very good property regarding this lag Fibonacci number again I am mentioning that if the period is very long and your requirement is well within that period then you are in the safe side and perhaps you need not have to worry too much in that particular context.

Now, for example, suppose M has b bits, maximum period for additive lagged Fibonacci random number generator is 2 to the power p minus 1 whole multiplied with 2 the power b minus 1 that is a huge range if I assume that the number of bits this b.

So, this number of bits b bits is this b actually is very high, so it is not 32 it is say 48 or even more and 2 to the power p, so this p is my seed value, so based upon that one it will vary, we will see that our requirement is not much so, this is in the context of random number generator where, we are interested to generate a huge random numbers, but within say 4 billion, etc that is 32 bit or say 48 bit at most, we are fine, we can do inside that.

So, that is all about this random number generator, so what we have discussed is that Monte Carlo simulation is going to be one technique specifically for problems where the solution is kind of NP hard in nature and in those situations, we see that the classic example on the computation of the Pi and we also demonstrated that the role of the random number in the context of the Monte Carlo simulation, because in the Monte Carlo simulation, so we will be making some random moves and based upon that random moves final solution will be outputted, will output. Now, if the moves are not sufficiently random in nature, then there is a possibility some bias will be introduced. So, we need to avoid this bias as much as possible by incorporating the as much randomness as possible in the random number generator, thank you very much.