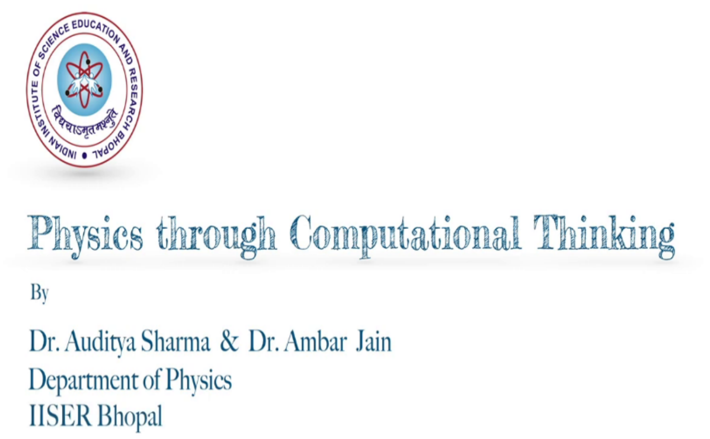
Physics through Computational Thinking
Professor Auditya Sharma and Ambar Jain
Department of Physics
Indian Institute of Science Education and Research Bhopal
Module 07 Lecture 39
The Monte Carlo Method 1

(Refer Slide Time: 0:20)



Physics through Computational Thinking

By

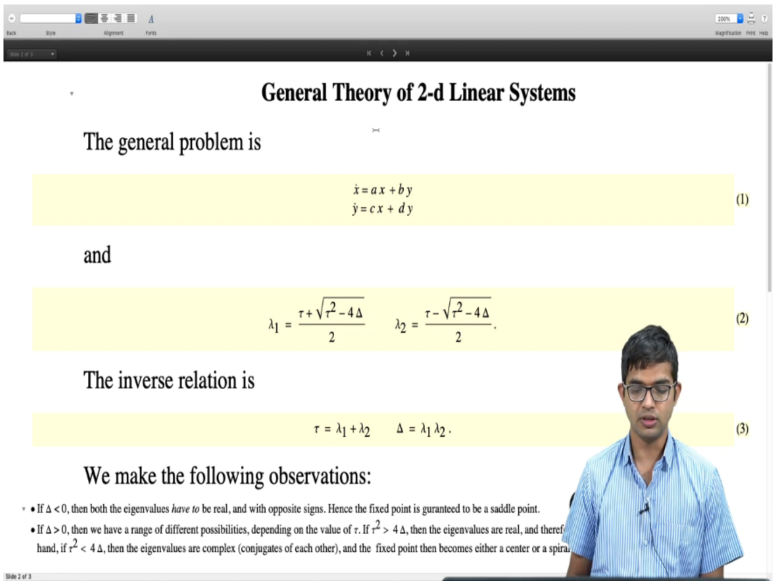
Dr. Auditya Sharma & Dr. Ambar Jain
Department of Physics
IISER Bhopal

Hi guys. So, we have been looking at linear systems, we have been looking at linear systems in 2D. And so, we defined the notion of a fixed point and then we argued for, you know why a good understanding of linear systems is useful and how they can actually help throw light even in you know more complicated nonlinear problems. And then we also offered a word of caution, when you know conclusions based on linearization may fail. Right. We gave examples of where linearization would work. And therefore, the power of a strong knowledge of linear systems is so useful and where linearization can fail, when you are the borderline cases.

So; and then we looked at some interesting examples; you know somewhat cooked up or artificial examples you might think, but they are interesting nonetheless. Differential equations, coupled differential equations, which were linear in nature. And then, that led us to, you know some fun analysis using the linear tool box. Now what I want to do is actually switch switch gears or switch a direction and get you an introduction to the Monte Carlo Method. But in order to do that, I will use our own study of the linear systems and you know the kind of things that we have been doing with 2D linear models; as a sort of bridge.

So, use what we have already been doing And gently introduce you to the Monte Carlo Method. And then we will see examples drawn from you know other kinds of context; you know in future modules. So, today what to try, in this module what we will do is, we will first look at, just quickly recall what we did with linear systems; the the general picture we have.

(Refer Slide Time: 2:28)



General Theory of 2-d Linear Systems

The general problem is

$$\dot{x} = a\,x + b\,y \qquad (1)$$
$$\dot{y} = c\,x + d\,y$$

and

$$\lambda_1 = \frac{\tau + \sqrt{\tau^2 - 4\Delta}}{2} \qquad \lambda_2 = \frac{\tau - \sqrt{\tau^2 - 4\Delta}}{2}. \qquad (2)$$

The inverse relation is

$$\tau = \lambda_1 + \lambda_2 \qquad \Delta = \lambda_1 \lambda_2 . \qquad (3)$$

We make the following observations:

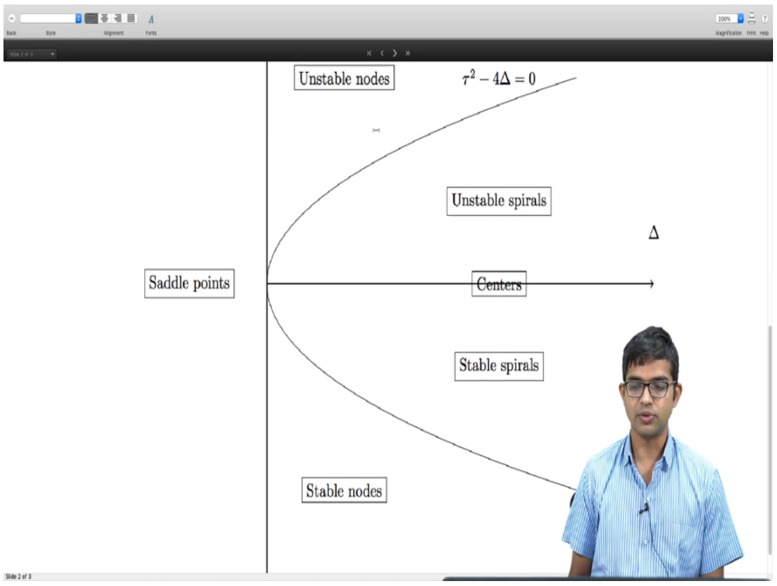
- If $\Delta < 0$, then both the eigenvalues *have to* be real, and with opposite signs. Hence the fixed point is guranteed to be a saddle point.
- If $\Delta > 0$, then we have a range of different possibilities, depending on the value of τ. If $\tau^2 > 4\,\Delta$, then the eigenvalues are real, and theref
hand, if $\tau^2 < 4\,\Delta$, then the eigenvalues are complex (conjugates of each other), and the fixed point then becomes either a center or a spira

Slide 2 of 3

Alright. So, we start with the General Theory of 2D Linear Systems. This is just pure recall. Right; so, we have seen this. So, we have a system, which is like$\dot{x} = a\,x + b\,y$, $\dot{y} = c\,x + d\,y$. So, all the information, all the (qua) qualitative features are contained inside the matrix a, b, c, d.

You diagonalize this matrix, obtain the eigenvalues; λ_1 and λ_2. And, or equivalently look at the trace of this matrix and the and the determinant of this matrix. Just these 2 numbers basically tell you all about the nature of the fixed point at the origin. So, we we have seen that this picture,

(Refer Slide Time: 3:08)



You know very compactly captures all possibilities; you know for your, the nature of the fixed point at the origin. So, you see that the large majority of systems actually give you saddle points. Saddle points are you know points, where your system is stable along one direction, but unstable along another direction.

And then, you see that there is another, you know fairly common, but not as common as saddle points is; is that of nodes. You know some, half of them are stable and the other half are unstable; you know roughly. And also you can have stable and unstable spirals. You can also have these borderline cases. You know borderline cases are the points, where you should be careful, if you are doing linearization of a nonlinear problem. We have seen all of this. Okay. So, now what I want to do is, pose this problem.

So, I am going to zoom this out. So, consider this linear system, where you know you have this vector xy and you have matrix $A = \{a, b, c, d\}$. And where, I do not know what these numbers ‘a, b, c, d’ are. I am just telling you that these numbers are are real.. And they are independently, they are drawn independently and from a uniform distribution, in the interval (-1 to 1).

And suppose I do this. Now I can ask; what is the probability that you know the the fixed point, that I just randomly pull out of this box? You can think of this as an ensemble of matrices. So, in some sense, this is how you are considering an ensemble of random matrices. So, the theory of random matrices itself is quite an advanced and beautiful theory. And you can, you know get many of those insights even by working, starting with something as simple as 2 by 2 matrices.

So, this is actually, you know these kinds of games that we will play, you can play more such games and actually uncover many beautiful results. And then there are ways to extend this to our arbitrary n by n matrices; you know. And so, in fact the whole theory of random matrices is itself a very, you know deep and beautiful theory. And a lot of it can be experimentally discovered.

You can basically play with Mathematica and uncover many of these results, which are already known. And then there are new results waiting to be discovered and so on. So, for our purposes, we start very small. Just consider random matrices, which are 2 by 2 and *a*, *b*, *c*, *d*'s are all real. You can; and we do not impose any constraints on the structure of the matrix. Sometimes, it is useful to demand that your random matrix is real symmetric; you know.

And these are, or you can demand that your random matrix is is complex, hermition and so on. So, but we here, we are not looking at anything of that kind. We just allow *a*, *b*, *c*, *d* to be whatever they want, as long as they are real. And which they lie in the interval (-1 to 1). Right? So, one way off-course of generating a real symmetric matrices out of something like this is to take, generate 1 random matrix *A B A* at random and then add add its transpose to itself and divide it by 2.

So, this is one standard approach and this will yield for you something called the Gaussian Orthogonal Ensemble. It is possible provided you choose your distribution of the elements appropriately. So, if you make them Gaussian, then it gives you this famous Gaussian Orthogonal Ensemble, which has some very very nice properties and so on. But that is a completely different story altogether.

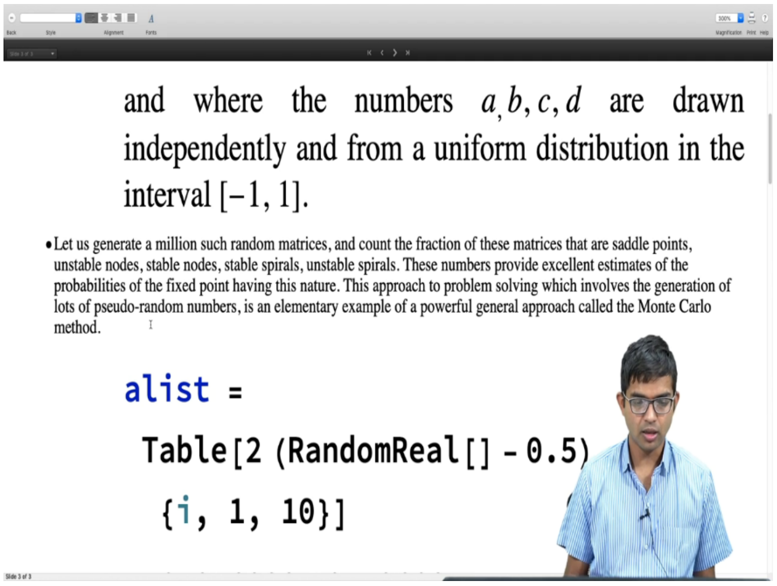
Here, we are just doing, we are playing a game, right; where all these number *a*, *b*, *c*, *d* are drawn from a uniform distribution. Now let us generate a million such random matrices. And well, the font has changed, because I am trying to switch between, you know this mode in Mathematica, where it is meant for, you know typesetting of a certain kind. And then I also want to evaluate. And so, in the middle of all this, sometimes the font gets messed up.

But hopefully, you can, you are able to see what is written there. But anyway, let me also explain what is going on. So, we simply generate a very large number of random matrices and do this analysis, which already we have; you know, find τ, find Δ. And based on the value of τ and Δ, you can compute, you know you can tell what is the nature of the fixed point there.

And then, by doing this repeatedly for many many many matrices, you can ask; what fraction of these random matrices are going to be, you know stable nodes and what fraction would be unstable nodes and so on. For every possibility on that 2-dimensional phase diagram that we had, we can just do a counting with random matrices and figure out what fraction are going to give you which, you know cover which zone of your phase diagram.

So, in order to do this, I am going to use a command called or a function called Random Real.

(Refer Slide Time: 8:31)



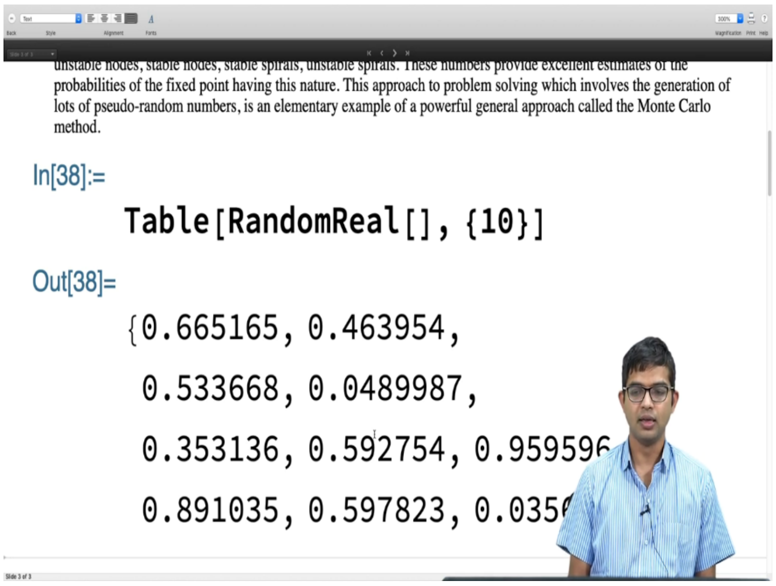
and where the numbers a, b, c, d are drawn independently and from a uniform distribution in the interval $[-1, 1]$.

- Let us generate a million such random matrices, and count the fraction of these matrices that are saddle points, unstable nodes, stable nodes, stable spirals, unstable spirals. These numbers provide excellent estimates of the probabilities of the fixed point having this nature. This approach to problem solving which involves the generation of lots of pseudo-random numbers, is an elementary example of a powerful general approach called the Monte Carlo method.

```
alist =
 Table[2 (RandomReal[] - 0.5)
  {i, 1, 10}]
```

I suppose you have already seen this. But anyway, Random Real is just a function, which will return for you a uniform distribution, between 0 and 1; if you are just giving no arguments. But there is a way for you to give it, you know shift your origin to somewhere else, or in other words you can ask it to give you a random number with mean, something other than 0 and with whatever variants or whatever. There are ways to set this. Then you will have to set the argument. But if I just simply do not give any argument, then Random Real is going to yield for me. So, let me show you,

(Refer Slide Time: 9:08)



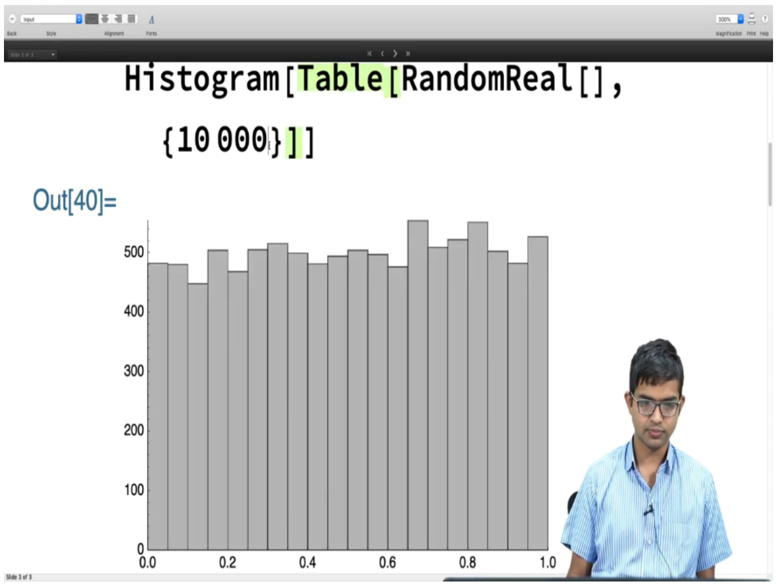
unstable nodes, stable nodes, stable spirals, unstable spirals. These numbers provide excellent estimates of the probabilities of the fixed point having this nature. This approach to problem solving which involves the generation of lots of pseudo-random numbers, is an elementary example of a powerful general approach called the Monte Carlo method.

```
In[38]:=
        Table[RandomReal[], {10}]
Out[38]=
        {0.665165, 0.463954,
         0.533668, 0.0489987,
         0.353136, 0.592754, 0.959596
         0.891035, 0.597823, 0.035
```

let me just do this; Random Real. I have to make sure that this is set to input. So, if I do this input, there you go. So, I can evaluate this. By hitting shift enter, I get a number 0.29. I can evaluate it again. I get some other number. So, I get a range of this. So, I can in fact do a table of this. Let us say I make a table of 10. So, it gives me a bunch of numbers. You can see that, all of them lie between 0 and 1. And I am telling you that, it is it is a uniform distribution.

So, one way to check this is; let me generate even larger number and then I do not want to print this out. But let me make use of another function called Histogram.

(Refer Slide Time: 10:06)



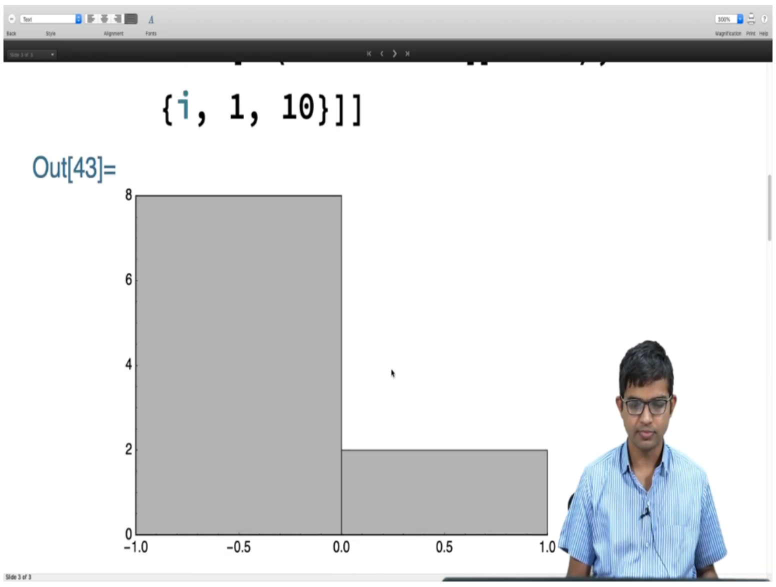
```
Histogram[Table[RandomReal[],
  {10 000}]]
Out[40]=
```

If I do this Histogram of this table, then you see, you see that Histogram is corresponds to roughly that of a uniform distribution. And so, the more such numbers you take, the closer it will become to the uniform distribution.

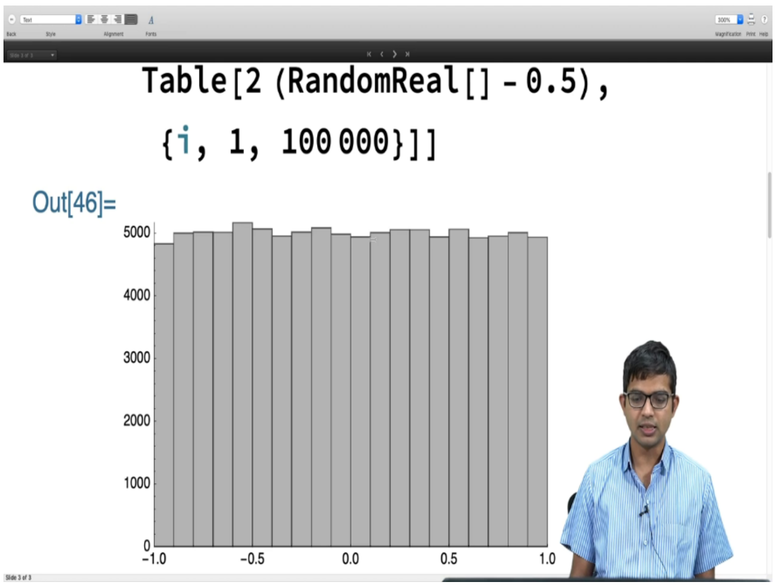
So, basically all regions of this interval; from 0 to 1 are covered uniformly. So, that is what is meant by a uniform distribution. And that is what we want here. Right? So, using a random variable, which runs between 0 and 1, it is easy to generate a bunch of random numbers, which are going to uniformly cover the region from -1 to 1. So, in order to do that, I have taken 2*RandomReal[]-0.5. So, this is one way of doing it. And there, surely there are other ways of doing it.

So, there you go; I have. And I am going to call this some list. I have called it 'a list'. You will see in a moment why I am doing this. So, you can look at this and you play this game for yourself. So, let me actually reduce this number and then hide this. Okay. So, now I have 'a list'; is equal to So, this, for you will give you numbers like this. If you want, you can; maybe I do not even want to equate it to 'a list'. So, let me again do histogram of this.

(Refer Slide Time: 11:34)



```
{i, 1, 10}]]
Out[43]=
```



```
Table[2 (RandomReal[] - 0.5),
  {i, 1, 100 000}]]
Out[46]=
```

Histogram, Histogram. So, it is covering. Yeah. So, there are too few numbers. So, it is not, it is not so illuminating yet. But if I increase this number, you will see that it is indeed a uniform distribution and which run from -1 to +1.

So, this is one way of doing it. So, you can use you know other techniques like using of a unit step function and so on. Okay, this is simple and I am using this. So, my solution is the following. So, by the way; so, the question is; can you generate a million such random matrices? And already, even without my showing you the solution, can you use your skill in

Mathematica, to find out what is the probability, that your fixed point is going to be a saddle point? what is the probability that your fixed point is a, an unstable spiral and so on; every possibility. So, by the way; so, this is this is a, an example of the powerful Monte Carlo Method. So, what is Monte Carlo, what does it involve?

So, Monte Carlo; the name itself comes from, you know the town Monte Carlo, which is famous for gambling. So, you know people would bet their money on outcomes of various kinds. And so, there is a probability, which is associated with events. Some events would be desirable. And so, they want to have a strong hold on you know, maximizing the probability of certain kinds of events and reducing the probability of other kinds.

And that is where people put in money and so on. So, that is; since there is a stochastic element to this approach, since there is a randomness involved; right, so, you simply get a very large number of random matrices. And you count, what fraction of it is going to give you of a certain kind and so on. And so, that is why it is a, an example of the use of the Monte Carlo Method. Right; so, there are sophisticated ways of using this and so on.

Maybe some of which, we might look at. Some examples for sure, we will look at. But I am going to show you my solution now. You can pause the video and crack this problem on your own. Or you can just allow this video to continue. So, my solution is the following. I am not claiming that this is the best solution available. This is one solution. So, let me zoom this to 125%. Yeah.

So, this is, what I am doing is, I am first of all generating a, an ‘a list’, a ‘b list’. I want to generate these 4 lists; *a*, *b*, *c* and *d*. Right; so, these are the 4 numbers that I, I want to you know generate and create a matrix. So, in order to do this, I will just say that let me generate some large number of *A*s, large number of *B*s, large number of *C*s and *D*s and corresponding to each index *i*; will be the one whole matrix; involving all 4 numbers.

So, the nice thing about Mathematica is, you can just take a whole array and you know; make arrays operate on each other arrays, as if they are like numbers. You can just deal with them as any regular variables. So, if I want to create a list, which is, which is a list of discriminants, ultimately all the information that is there in these matrices is, you know it contains some more information than actually what we need. All we need is information about delta and tau.

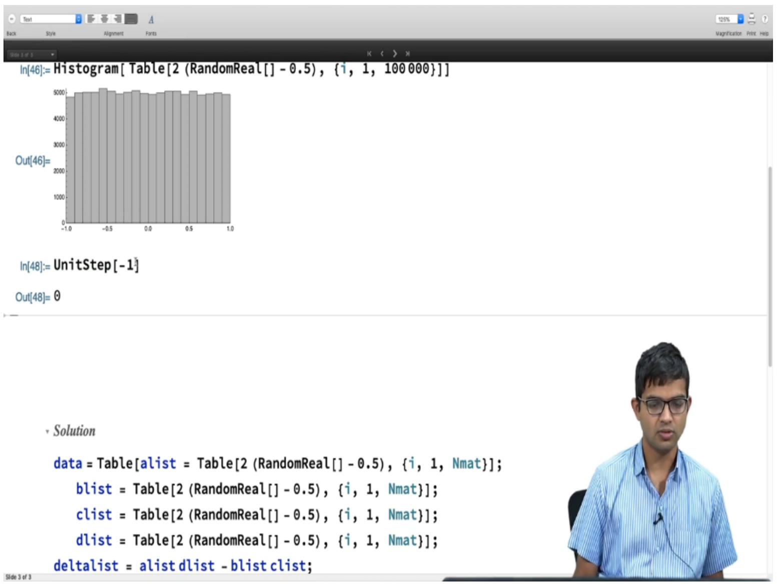
So, I am going to create this delta list and tau list. And how to create delta list? Very easy. All I have to do is I do, *alist*dlist - blist* clist*. Very simple; it is as if I am computing $a * d - b * c$. And so, if I do this, it automatically means that you do it for every i; i going from 1 to *Nmat*. Then tau list is simply $a^2 + d^2$. Right; so, that is what it is $a + d$. It is not even square. There is no square. It is just the trace of this matrix. So, it does, it computes the trace of each of these matrices and creates a list and puts them altogether. And then there is a discriminant list. So, discriminant list is $\tau^2 - 4\Delta$. Right; so, Δ is determinant.

Now discriminant of your quadratic equation, which was $\tau^2 - 4\Delta$. Once again, I will just simply use this trick to define a new list called 'disc list'; is equal to $\tau^2 - 4\Delta$. That is it. And then, in order to get the fraction of saddle point; so, I am using a simple one-line code to get this. You see; all I do is, is find the length of this thing called 'delta list'. It just tells you total number of elements.

And so, then I have, I have to find the sum and then divide it by the length. That is the total number of matrices. So, length of *deltalist* is going to be basically the same as length of *alist* or length of *blist*. All of them are going to be the same. It is just *Nmat*. I got to directly use this. But it is okay. I mean length is a useful function, for you all to be aware of.

So, unit step is a useful function. Right; I could have actually used unit step also to generate this shift, that I did to to get my uniform distribution between minus 1 and 1. So, what does this unit step do? It says that; so, let us look at what unit step does. So, I can take an example; unit step.

(Refer Slide Time: 17:37)



```
In[46]:= Histogram[ Table[2 (RandomReal[] - 0.5), {i, 1, 100000}]]
Out[46]=

In[48]:= UnitStep[-1]
Out[48]= 0
```

Solution

```
data = Table[alist = Table[2 (RandomReal[] - 0.5), {i, 1, Nmat}];
   blist = Table[2 (RandomReal[] - 0.5), {i, 1, Nmat}];
   clist = Table[2 (RandomReal[] - 0.5), {i, 1, Nmat}];
   dlist = Table[2 (RandomReal[] - 0.5), {i, 1, Nmat}];
deltalist = alist dlist - blist clist;
```

So, I have to convert this into a So, if I do UnitStep[1] is 1. But unit step of any number less than 0, is 0. That is all. And so why do I want to use unit step here? So, basically, I want to look at all these, you know all these entries in *deltalist*, which are less than 0. Whenever unit step is negative; whenever delta is negative, unit step of a negative number will be 0.

So, 1 - 0 will be 1. So, I just count this. Total of all of this will give me 1. So, whenever Δ is negative, all of that is counted here and then I divide it by length; which is actually nothing but *Nmat*, in this case. And then I have to do slash slash N, to get a number out of this. So, likewise I can do fraction of unstable node. So, what all, what all conditions must hold, I must have; Δ should be positive.

So, I take UnitStep[Δ]. Then it, it should also simultaneously give me; I should also have, simultaneously have tau to be positive. So, I take the product with τ and then *taulist*; UnitStep[*disclist*]. And so, I must also have a discriminant to be positive. So, all 3 conditions, when they are simultaneously met, then the simple way to count all of this is to just take the product of these 3. And then I just total all of them up; done. Very very simple.

Right; so, if you were to write this type of code in some of the language, surely you will have to spend many more lines. And it is, it is very nice, very convenient for quick calculations. Right; and you can do even big numbers. You know; so, look at *Nmat*. So, then finally what do I want. I am computing only these 2 quantities. But I will allow you guys to play with, you know improvise on this code and get fraction of; for example, stable node.

It should be the same as fraction of unstable nodes and so on. Right? So, there is some symmetry in the problem. So, let us look at what happens here. I am going to run my *Nmat*, all the way from 100, 000 up to 1 million, in steps of 100, 000. And then I will finally plot all of this. So, this is I am going to run this. Then I have list line plot of. So, when I say 'all and 1, 3', it gives me basically the plot of data with respect to the 1st and the 3rd. So, the 3rd here is frac unstable node.

Let us look at that one first. So, see; it is going to some value like 0.09. Right? So, you can make this even more sophisticated. Try to get error bars, like using, using the methods that we have described earlier; for calculating error bars, when you are using. Whenever you have any stochastic element and when you have lots of samples, there is a way to estimate the error. And you can try to plot all of them.

So, let us look at what happens with our other case; fraction of saddle points. There it goes to some number close to half and this is not a surprise. It is not a surprise, because geometrically we can see that, you know half of this plane is pretty much for every value of delta less than 0. It is a saddle point.

Although, I mean this kind of geometric picture you should not blindly believe, because it is not always clear that just because a geometric picture shows that, you know one half of the region is covered by points of a certain kind, it is not necessary. In this case, perhaps it is, there is a way to argue for it more rigorously. It is not necessary that, you know the fraction of your sample space is the same is the geometric area, that you are seeing.

So, this has to be argued out carefully. But in this case, it does turn out that the fraction of matrices, which give me saddle nodes is exactly half. So, perhaps there is some symmetry argument you can give and and, in any way, justify this. But I am just giving you a word of caution on blindly interpreting some geometric area.

So, it is not necessary that there is a uniform distribution involved. It is not like all parts of your plane are to be treated with, on the same footing. Maybe in this case it is, but not not in every case of this kind. Alright; so, what have we seen in this module; we have looked looked. We have made a connection between linear systems, the general theory of linear systems in 2D and used it to open up, open a window to the Monte Carlo Method.

Using 2 by 2 matrices, generating a large number of random matrices and checking that, you know indeed, with with a probability of half, you are going to get saddle nodes. Saddle nodes are the most common kind of fixed points, in linear systems.

On this note, let us end this module. We will see more Monte Carlo in future module. Thank you.