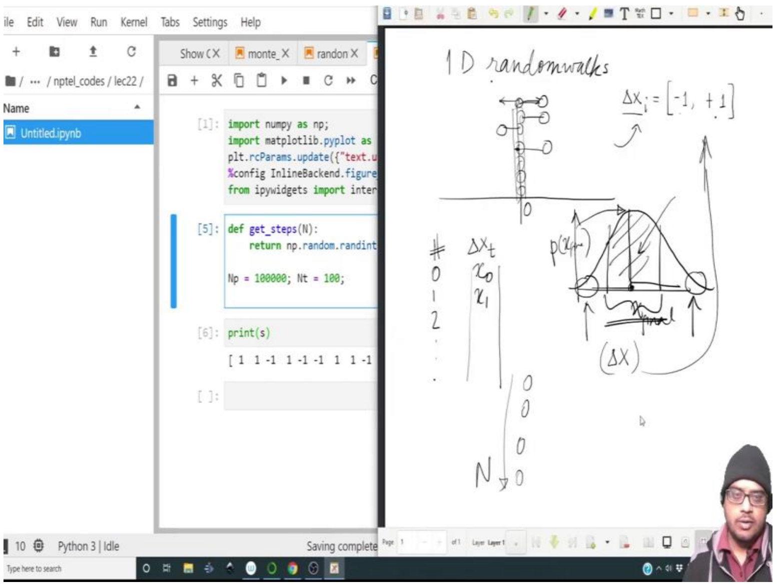
Tools in Scientific Computing
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 22
ID Random Walks

(Refer Slide Time: 00:33)



```
[1]: import numpy as np;
     import matplotlib.pyplot as
     plt.rcParams.update({"text.u
     %config InlineBackend.figure
     from ipywidgets import inter

[5]: def get_steps(N):
         return np.random.randint

     Np = 100000; Nt = 100;

[6]: print(s)

     [ 1  1 -1  1 -1 -1  1  1 -1
```

Everyone, welcome to this lecture; in this particular lecture, we are going to consider One-Dimensional Random Walks. In the last lecture, we had considered Monte Carlo simulations in which we were generating certain state of the entire event. So, we were not necessarily taking the entire process into considerations. We just were worrying about how the final state looks like.

But in this particular lecture we are going to look at the process in particular. So, let us begin by the very simple process that is going to be a set of particles which are originally located at the origin, ok. So, there are a bunch of particles all stacked on top of each other. Well, it does not matter whether they are stacked or not, but we have say a lot of particles at the origin.

Then, at the next time instant there is we sample a step, ok. So, for each particle there is a step it can be either -1 or it can be plus +1. And so, the point is depending on whether the sample step is -1 or +1, we will either move the particle to this side or this side. And we will do this for all the particles. So, this sampling is different for each particle. But we are necessarily sampling from either this or this. So, this kind of a sampling is called as the binomial sampling or the Bernoulli trial.

So, this particle suppose it gets over here, this particle also gets over here, this particle gets over here, this particle may get over here, and so on. Then, over the next time instant again you find out what step increment it must have by sampling from this and keep doing this. And the question is for large times how does this particular.

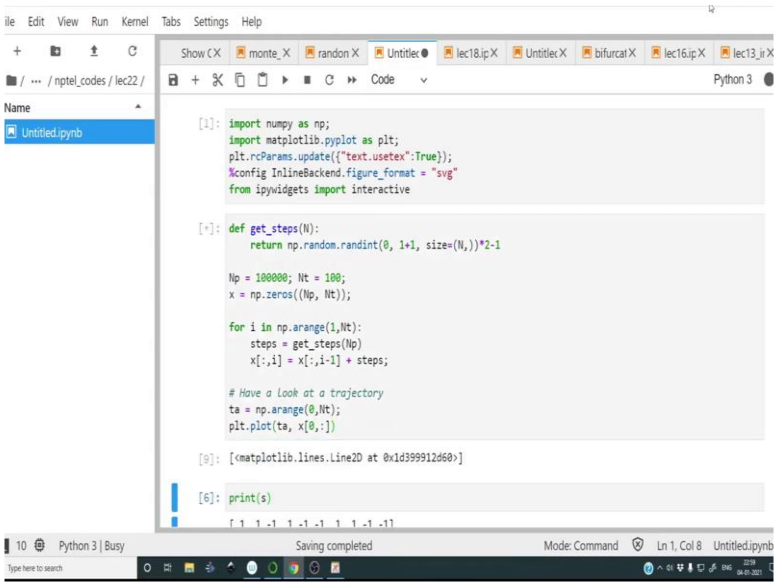
So, let me call this as particle in number. So, particle number 0, 1, 2, and so on and this is Δx at some time. So, maybe we will have x_0, x_1. So, these are all at some final time t. So, we are interested to know what is the distribution of all the locations of all the particles at the final time whether all the. So, if the distribution of the final location looks something like this, it means that all the particles are sort of accumulated over here.

If the distribution looks something like this then we can say that the majority of the particles. So, this is what the pdf is, the pdf of the final location, the probability density function. So, the probability density function having a high value over here means there is a higher probability that the particle is inside this range and a lower probability that the particle is either too far from this mean or too away from this mean.

So, the question is how can we perform this particular random walk, ok. This kind of process, it is a random walk and what can we infer from it. In general, random walks are not just some synthetic simulations like this, but they often have a very deep physical understanding for various processes occurring in nature and for various engineering and science applications as well, ok.

So, that notwithstanding let us begin. So, let us go over here I have already pasted this particular bit of text that we will need, alright. So, now, let me just yeah. So, what we need is to sample from a given distribution, ok. So, there are various functions available inside numpy which can help us do that. So, let me first create a function, which will help me obtain various steps, ok.

(Refer Slide Time: 05:13)


```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive

[*]: def get_steps(N):
         return np.random.randint(0, 1+1, size=(N,))*2-1

     Np = 100000; Nt = 100;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np)
         x[:,i] = x[:,i-1] + steps;

     # Have a look at a trajectory
     ta = np.arange(0,Nt);
     plt.plot(ta, x[0,:])

[9]: [<matplotlib.lines.Line2D at 0x1d399912d60>]

[6]: print(s)

     [ 1  1 -1  1 -1 -1  1  1 -1 -1]
```

So, let me define a function called as get_steps(N) and the input will be N, that is the N over here I would pass it as the number of particles. So, if all the particles are like this. So, there are N number of particles. So, I will ask the computer to give me N number of such increments and they would necessarily be sampled from this either it will be +1 or it will be -1, ok, so fine.

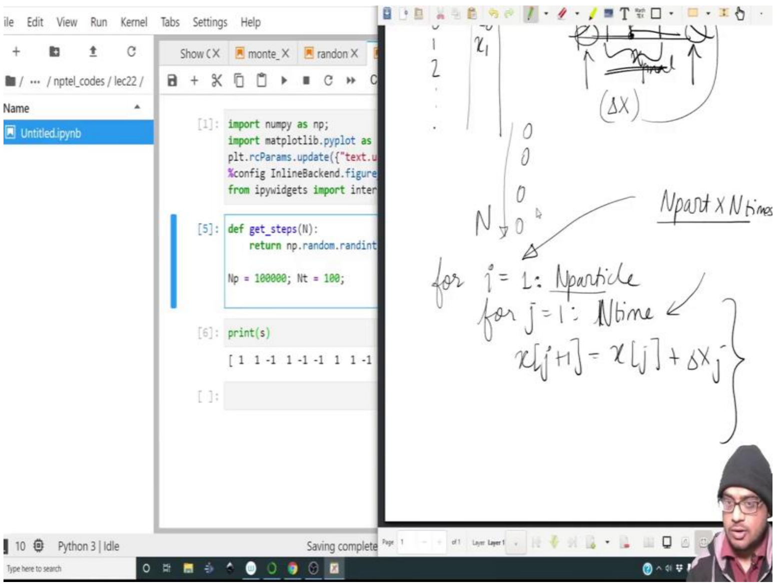
And I must return np.random.randint(0, 1+1) because randint does not include the end number. So, this is the end number. This is the number I want and +1 is because I have to increment the range by one it excludes the outer range, ok. And I must give size=(N,). So, it is a one-dimensional array of size N. So, this is how I have the definition of this. So, in fact, let me show you how we can do that.

So, let me say s = get_steps(10). So, let me now print what s is. So, its sampling between 0 and 1 yeah that is what we see. So, now, I want to sample between -1 and 1. So, I will multiply this by 2 and subtract 1, ok. So, let me run this. So, now, you have a bunch of numbers or or an array of size n here in this case 10 which are which are comprised of 1 and -1, ok.

So, now, let us go into the other part of the logic. So, we have to now initialize, so we have to declare how many particles we have we have to declare how many time steps we need to have and so on. So, let us define Np as the number of particles as say 10000, ok. Well, this is not 10000 this is 100000 or 1 lakh. Let me define the number of time steps.

Let me put it on one line. So, number of time steps is, let us say it is 100. So, now, what do we do?

(Refer Slide Time: 07:48)



```
[1]: import numpy as np;
     import matplotlib.pyplot as
     plt.rcParams.update({"text.u
     %config InlineBackend.figure
     from ipywidgets import inter

[5]: def get_steps(N):
         return np.random.randint

     Np = 100000; Nt = 100;

[6]: print(s)

     [ 1  1 -1  1 -1 -1  1  1 -1

[ ]:
```

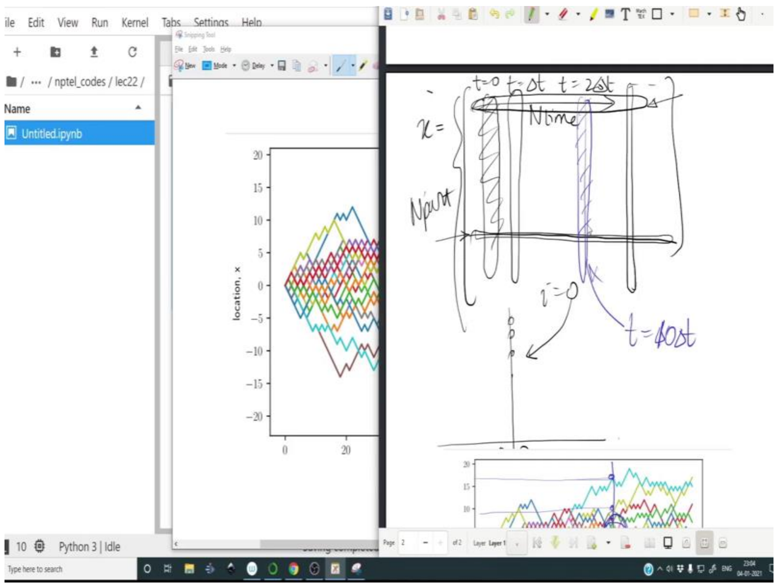
```
for i = 1: Nparticle
    for j = 1: Ntime
        $x[j+1] = x[j] + \Delta x_j$
```

We will initialize all the particle locations. So, we want to vectorize the program. You can do the following, you can do the following pseudo code for i = I mean this is just a pseudo code, it is not Python code; 1 2 number of particles, ok.

For, j = 1 2 time steps, so n time. Meaning I am grabbing hold of one particle in a loop and I am sort of iterating in time to find out the new location of the particle. So, $x[j+1] = x[j] + \Delta x_j$ for that particle. I can do this, right. So, it is like for each particle I have to do these many number of operations, so the complexity would be n particle multiplied by number of times.

So, that is rather expensive. But in Python we know that we can sort of vectorize everything. We can do all of this without having to write this double loop. In fact, you can do a lot of things without having to write loops and that is the that is the part where you achieve speed, ok. In C, there is no option to do that, but in Python that is what exactly you are aiming for.

(Refer Slide Time: 09:08)



So, we will make a matrix x which will be something like this. So, at a given Δt , so this will be Δt = 0, not Δt = 0, this will be t = 0 and these will all be the particle locations.

So, you must create a matrix whose number of rows will be n part and whose number of columns will be N time, this will be t = Δt , t = 2Δt and so on. So, for a given row inside this matrix we would essentially be singling out one particle and having its positions at all times. If we look at a column in this matrix, we are freezing time and looking at the positions of all the particles at that given time, ok. I hope this much is clear. This is essential to understand and what is going to follow.

So, x = np.zeros((Np, Nt)). So, here it will be Np, Nt, alright. So, so far so good and N; and so, we must also we have already initialized x = 0. I mean the fact that the simulation must start, so that all the points are initially at x = 0 that is already satisfied through this statement.

Now, what we must do? I am I am going to make one loop over here for i in np.arange(1,Nt). Meaning for i going from 1 to Nt -1, right because i = 0 is already the initial condition and that we already have.

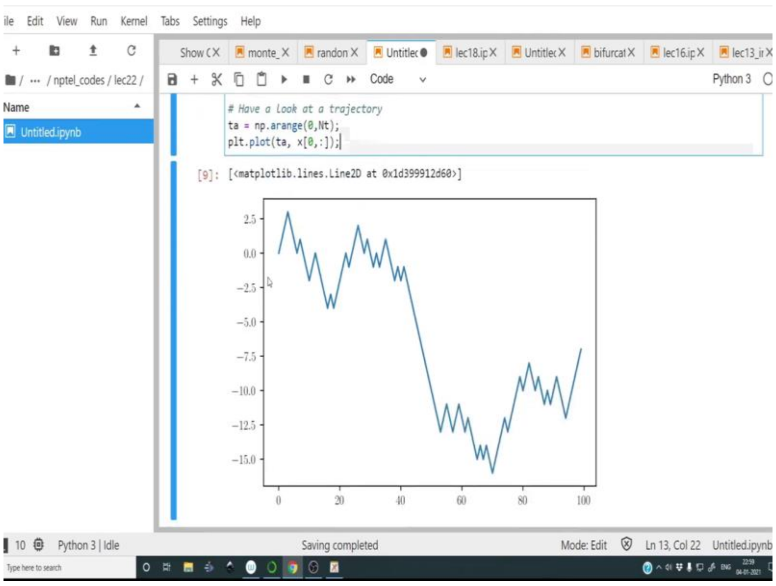
Inside this loop what I am going to do is x[:,i] = x[:,i-1] + steps, alright. So, essentially it is saying that all these rows will be all these previous locations plus some step, alright. And the step is simply sampled from the function that we have defined above, alright.

So, now, steps will be get steps and here it will be Np, alright. So, with this particular loop we what we can do is we can simply yeah; once this loop completely runs we can have a look at the trajectories.

So, how do we do that? We will do plt dot plot and we will simply plot. So, let me create a time loop or a time array ta, np.linspace not linspace, np.arange (0,Nt) and correspondingly we have all the locations.

So, you want to plot suppose I want to plot the trajectory of the particle with index equal to 0. That means, this first row all the points correspond to all the locations are successive times corresponding to particle with index equal to 0. So, let us do that; so we will do ta, x[0,:], alright. So, let us do this. So, let me just suppress this, ok.

(Refer Slide Time: 13:19)

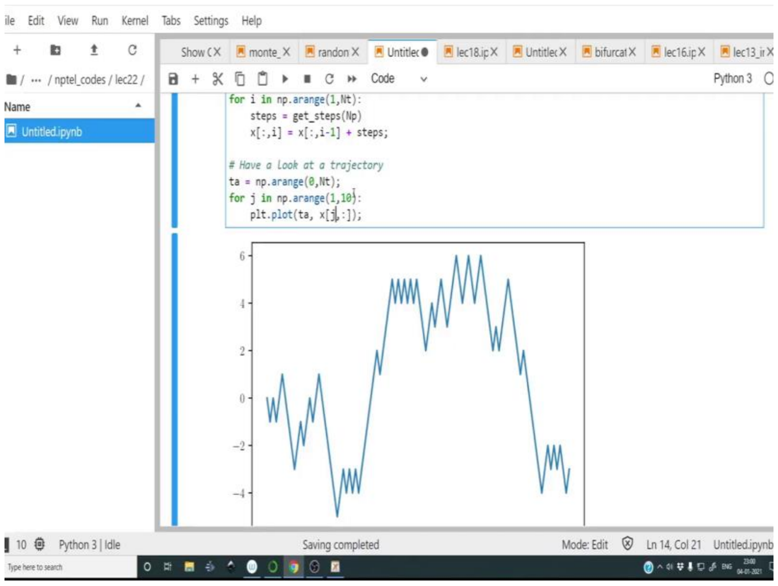

ile Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec22 /
Name
Untitled.ipynb
Show (monte_ randon Untitlec lec18.ip Untitlec bifurcat lec16.ip lec13_ir
Code
Python 3
Python 3 | Idle
Saving completed
Mode: Edit
Ln 13, Col 22
Untitled.ipynb

```
# Have a look at a trajectory
ta = np.arange(0,Nt);
plt.plot(ta, x[0,:]);
```

```
[9]: [<matplotlib.lines.Line2D at 0x1d399912d60>]
```

So, it starts with 0, then it does this work. So, some increments some decrements some decrement increment and it keeps on going like this, ok, no problem. In fact, if I run this again it will create another plot, the reason why it will create another plot because I would be re-sampling the random numbers, I would be re-sampling the steps the particle would take, ok. So, once I do this, I get another trajectory.

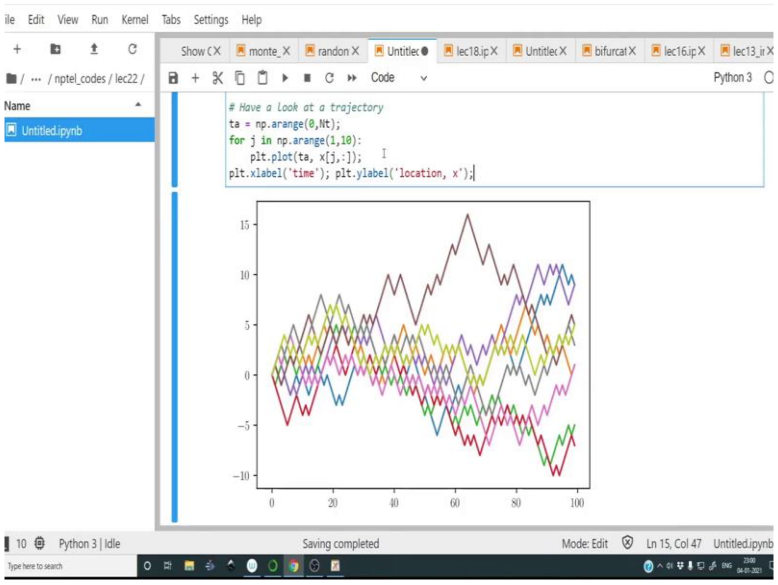
(Refer Slide Time: 13:44)



```
for i in np.arange(1,Nt):
    steps = get_steps(Np)
    x[:,i] = x[:,i-1] + steps;

# Have a Look at a trajectory
ta = np.arange(0,Nt);
for j in np.arange(1,10):
    plt.plot(ta, x[j,:]);
```

So, in fact, we can loop over some numbers and plot certain trajectories or sets of trajectories. So, for j in np. arrange let us do 1 to 10. Let us let us plot these trajectories and then I will put this inside the for loop and I will put j over here. So, this should give me all the trajectories. So, there you go. These are the various trajectories.

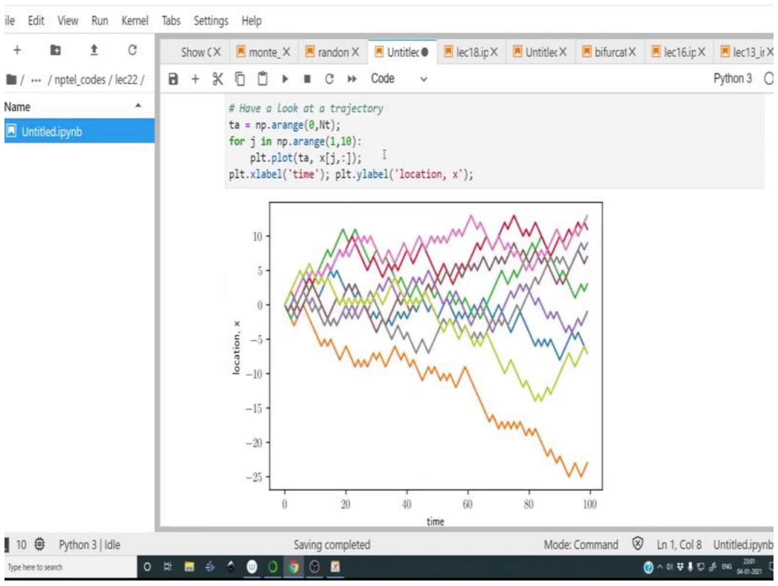
(Refer Slide Time: 14:11)



```
# Have a Look at a trajectory
ta = np.arange(0,Nt);
for j in np.arange(1,10):
    plt.plot(ta, x[j,:]);
plt.xlabel('time'); plt.ylabel('location, x');
```

So, what is the x axis? So, plt.x level. So, x axis is simply time whereas, the y axis which is nothing but the location, right, ok.

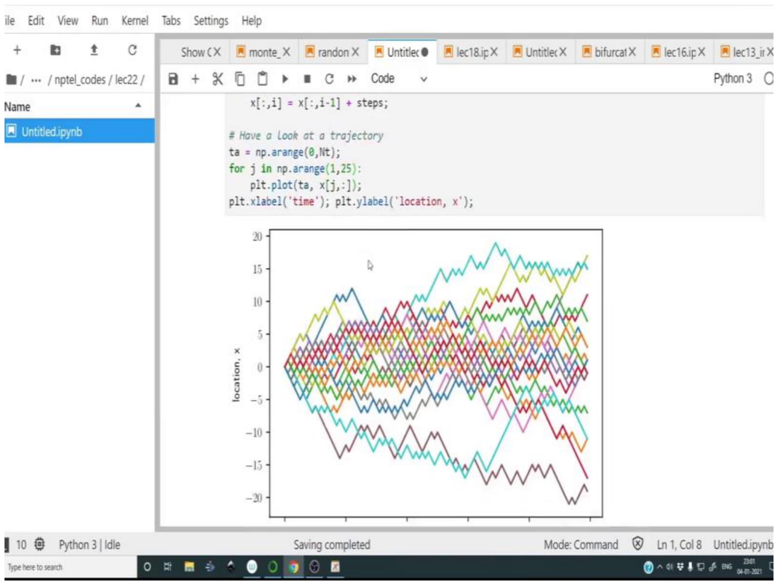
(Refer Slide Time: 14:38)



```
# Have a look at a trajectory
ta = np.arange(0,Nt);
for j in np.arange(1,10):
    plt.plot(ta, x[j,:]);
plt.xlabel('time'); plt.ylabel('location, x');
```

So, these are the evolution of the locations of the different particles and time, ok. So, what are we interested in? We are interested in finding out the distribution of locations for such a system. Meaning I will freeze time I will look at all the locations. And I will try to form a histogram.

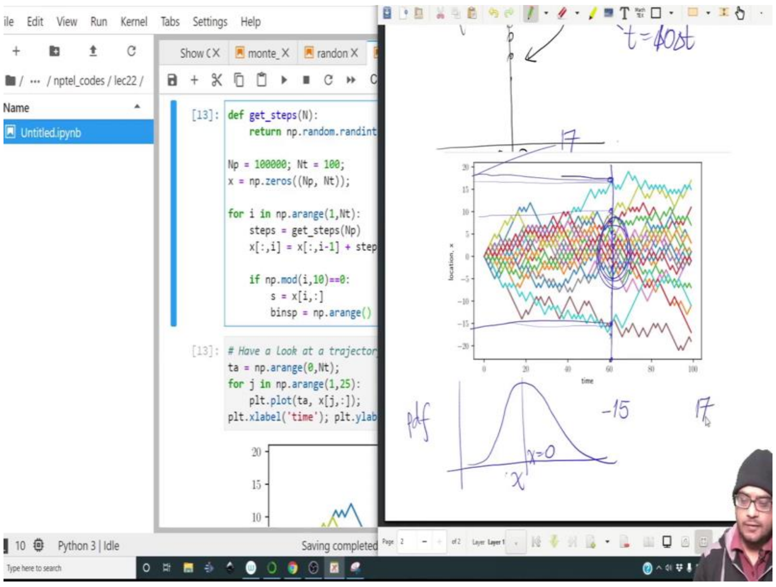
(Refer Slide Time: 15:08)



```
    x[:,i] = x[:,i-1] + steps;

# Have a look at a trajectory
ta = np.arange(0,Nt);
for j in np.arange(1,25):
    plt.plot(ta, x[j,:]);
plt.xlabel('time'); plt.ylabel('location, x');
```

In fact, let me put down a few more trajectories and we can copy it and source.

(Refer Slide Time: 15:27)



```
[13]: def get_steps(N):
          return np.random.randint

      Np = 100000; Nt = 100;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np)
          x[:,i] = x[:,i-1] + step

          if np.mod(i,10)==0:
              s = x[i,:]
              binsp = np.arange()
```

```
[13]: # Have a Look at a trajector
      ta = np.arange(0,Nt);
      for j in np.arange(1,25):
          plt.plot(ta, x[j,:]);
      plt.xlabel('time'); plt.ylab
```

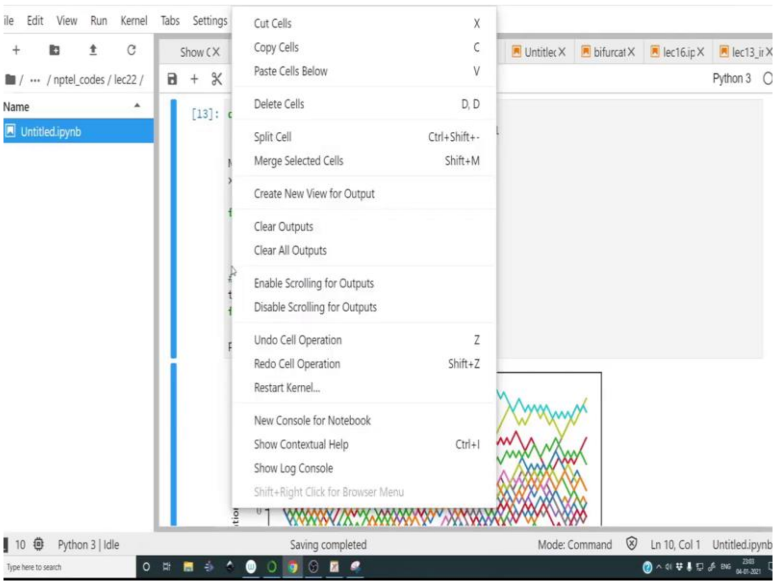
Let me take this to our notebook. So, we will fix time and look at the locations of all the particles, ok. We have a bunch of part particles. So, what do we have? We have some particles which are at 17 some which are so majority of them appear to lie near the origin, there are some outliers and so on.

Then, what we will do is we would like to find out the pdf, the probability density function of the location at some later time t because they are more clustered near the origin, we expect a distribution something like this, so where this is x = 0. And why do we expect them to be clustered?

Because you know you see the distribution from which the step is sampled its either +1 or its -1 meaning if it goes +1 in the positive x direction its equally likely in the next time instant to come back to -1, ok. So, that is why you will see more clustering near x = 0.

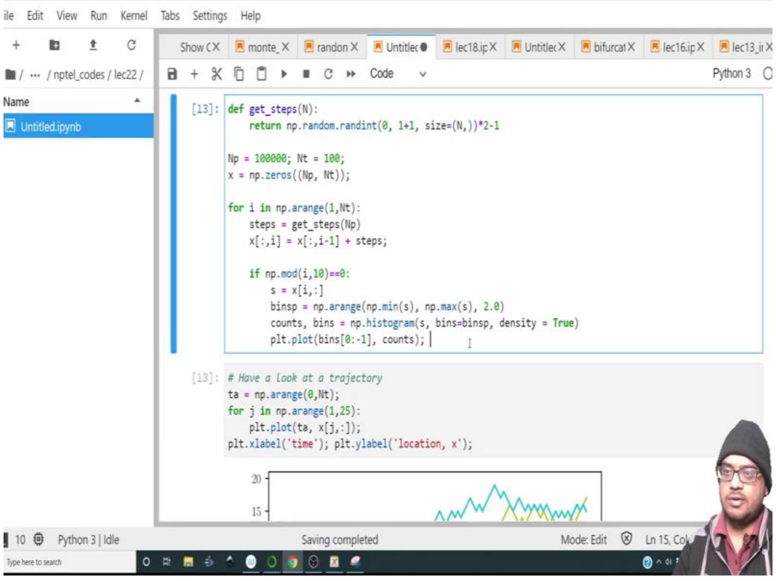
But what is the nature of the distribution that is what we are more interested in, ok. So, let us do the histogram. So, this basic program we have all the trajectories, but we are more interested in the histogram of x at a given time, ok. So, this is how the update happens so far so good.

(Refer Slide Time: 17:05)


Cut Cells X
Copy Cells C
Paste Cells Below V
Delete Cells D, D
Split Cell Ctrl+Shift+-
Merge Selected Cells Shift+M
Create New View for Output
Clear Outputs
Clear All Outputs
Enable Scrolling for Outputs
Disable Scrolling for Outputs
Undo Cell Operation Z
Redo Cell Operation Shift+Z
Restart Kernel...
New Console for Notebook
Show Contextual Help Ctrl+I
Show Log Console
Shift+Right Click for Browser Menu

Let me split this, so that it is in a different cell we do not need this at the moment. So, what should we do? So, suppose every tenth step I plot what the histograms are, ok, fine.

(Refer Slide Time: 17:23)



```
[13]: def get_steps(N):
          return np.random.randint(0, 1+1, size=(N,))*2-1

      Np = 100000; Nt = 100;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np)
          x[:,i] = x[:,i-1] + steps;

          if np.mod(i,10)==0:
              s = x[i,:]
              binsp = np.arange(np.min(s), np.max(s), 2.0)
              counts, bins = np.histogram(s, bins=binsp, density = True)
              plt.plot(bins[0:-1], counts);
```

```
[13]: # Have a Look at a trajectory
      ta = np.arange(0,Nt);
      for j in np.arange(1,25):
          plt.plot(ta, x[j,:]);
      plt.xlabel('time'); plt.ylabel('location, x');
```

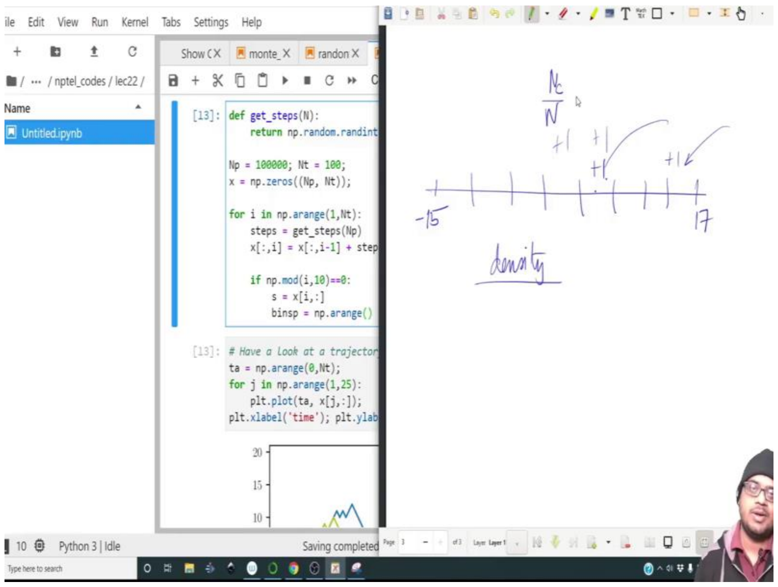
if np.mod(i,10)= =0:. Meaning each time I is a multiple of 10 we will execute something inside this. So, what will we do? So, we will first say all the samples from which we want to create the histogram. Look again I am going to repeat.

So, we are going to freeze time and look over all the locations that we have for different particles. What that means is we are going to look at a certain column at a given time this could be for example, t = 10 Δt . Then we do the same for t = 20 Δt . So, we would look at all the locations and try to plot the probability density function, right. So, s = x[:,i] this means it now contains all the locations. So, this I call as s for samples.

Now, I will create a bin space. So, in order to create the histogram, I must tell what the bins of the histogram should be. So, that the program can assign those many counts inside a given bin. So, bin space I mean I have I just want to going to it bin space you can create anything you want.

So, this will be np.arange and it will be going from. So, what should be the decision? How to find out the bin space? Look in at this particular time I know that the maximum value is something like 17, minimum value is something like I do not know at this point of time it is something like minus 15, but whatever.

(Refer Slide Time: 19:31)



```
[13]: def get_steps(N):
          return np.random.randint

      Np = 100000; Nt = 100;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np)
          x[:,i] = x[:,i-1] + step

          if np.mod(i,10)==0:
              s = x[i,:]
              binsp = np.arange()
```

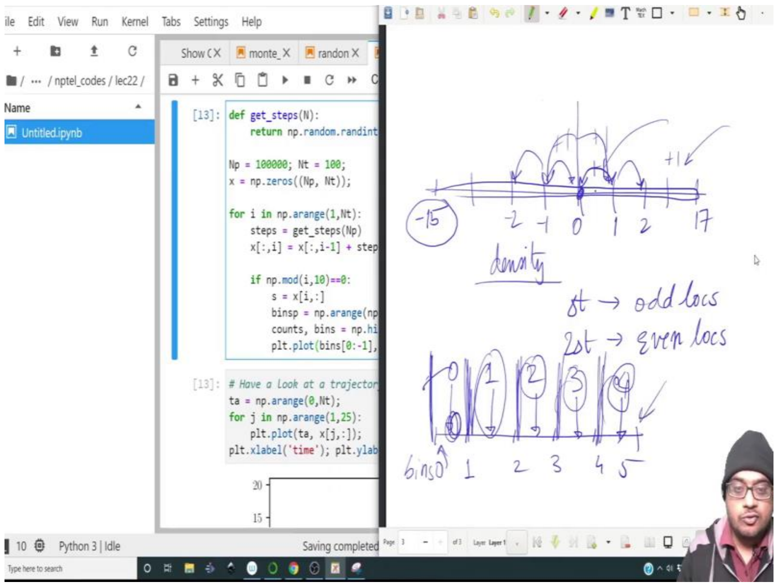
```
[13]: # Have a Look at a trajector
      ta = np.arange(0,Nt);
      for j in np.arange(1,25):
          plt.plot(ta, x[j,:]);
      plt.xlabel('time'); plt.ylab
```

So, we have to make bins going from 15 to 17, so it will look something like this. So, minus 15 to 17 and I must create a bunch of bins and each time a particular location lies in a bin I will make the counter +1, +1, +1, +1 and so on.

There will be fewer +1s at the edges because of the discussion we did a few moments earlier that the clustering will be near the origin. So, in that case if I ask the computer to instead of giving me the count give me the density, so because it is an important aspect instead of giving us the count it will give me the density.

Meaning it will normalize everything by the number of particles. So, if I have n counts over here, I will divide it simply by n to get the density. So, I will do this. So, the space over which I want to plot the histogram must be appropriately defined. If you do not define that Python will assume a certain number of bins and do the calculations. You can always override it, ok. So, I will have it going from np.min(s), np.max(s). And I will take the step size to be 2. Why have I taken the step size to be 2? Let us see.

(Refer Slide Time: 20:55)



```
[13]: def get_steps(N):
          return np.random.randint

      Np = 100000; Nt = 100;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np)
          x[:,i] = x[:,i-1] + step

          if np.mod(i,10)==0:
              s = x[i,:]
              binsp = np.arange(np
              counts, bins = np.hi
              plt.plot(bins[0:-1],

[13]: # Have a look at a trajector
      ta = np.arange(0,Nt);
      for j in np.arange(1,25):
          plt.plot(ta, x[j,:]);
      plt.xlabel('time'); plt.ylab
```

+1
-15
-2
-1
0
1
2
17
density
$\delta t \rightarrow$ odd locs
$2\delta t \rightarrow$ even locs
0
1
2
3
4
bins 0
1
2
3
4
5

So, this is 0, this is 1, this is -1, -2, 2. So, after one time step particle will be either at -1 or it will be at +1. So, at Δt the particles are at odd locations. At the next time step it will either go to 0, it will go to 2 or it will go to 0 or it will go to -2. So, at $2\Delta t$ the particles will be at even locations.

So, that is why this in order to avoid the seeming dichotomy occurring in this problem I will just simply take bins of size 2. I will simply do binning of size 2 rather than doing a bin of size 1; because if you I will I will show you if you do a bin of size one you will get empty peaks for every interval, either if you have odd time then you will have empty even bins if you have even time you will have empty odd bins because of this particular logic.

But this is not of great concern because I will choose the bin size to be 2.0, alright. So, now, that we have defined the bins now that we have the samples, we will call the histogram function. So, it will be counts, bins = np.histogram(s, bins=binsp, density = True), alright.

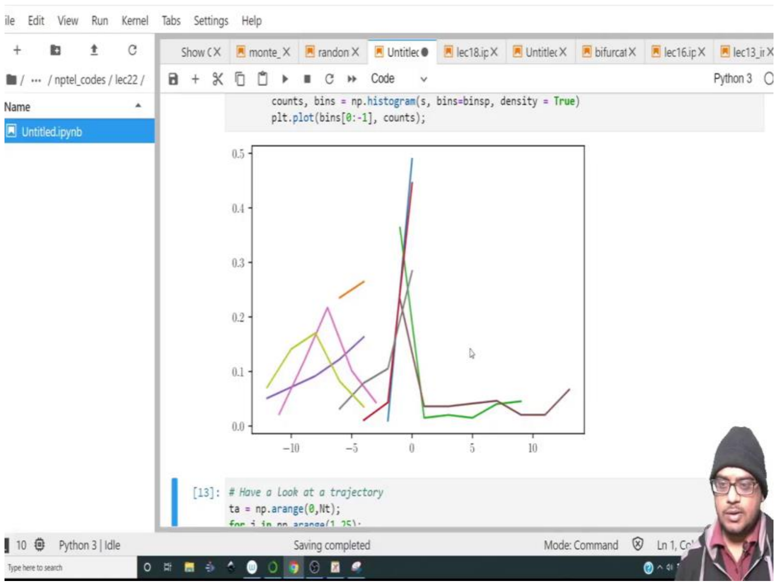
So, the np.histogram command takes as an input the sample the bin space and whether density has to be true or not. It will give us an output, the counts, so if density is false it will give you the actual integer count, if the density is true it will give you the fraction. So, essentially it will give us a normalized count. And bins will be the same as bin space. But no worries with that, alright. So, this is what we have.

Then what should we do? We have to plot this, so plt not plot. So, the thing about histogram is it will. So, these are the edges of bins, so this will be bins 0, this will be bins 1, bins 2, 3, 4, 5, but counts will be 1, 2, 3, 4, 5; rather this will be count 0, count 1, count 2, count 3, count 4. So, the number of elements in the array counts will be 1 less than the number of elements in the array bins.

And that is just something because of the edges and the center values. So, what we will do. We will disregard all that. We will simply do a plot of bins 0 to -1, counts. So, I am I am I am sort of shifting everything to the first element what I am doing is; whatever this is the value I will I will associated with this edge, this value I am associating with this edge, this value I am associating with this edge, this value associating with this edge, this value associating with this edge, this value remains empty.

And how do I choose 0 to second last element? It is through this command as simple as that we have done this iris splicing in the very beginning. So, if you forgotten just make a random array and try to do the splicing and see, alright.

(Refer Slide Time: 24:36)

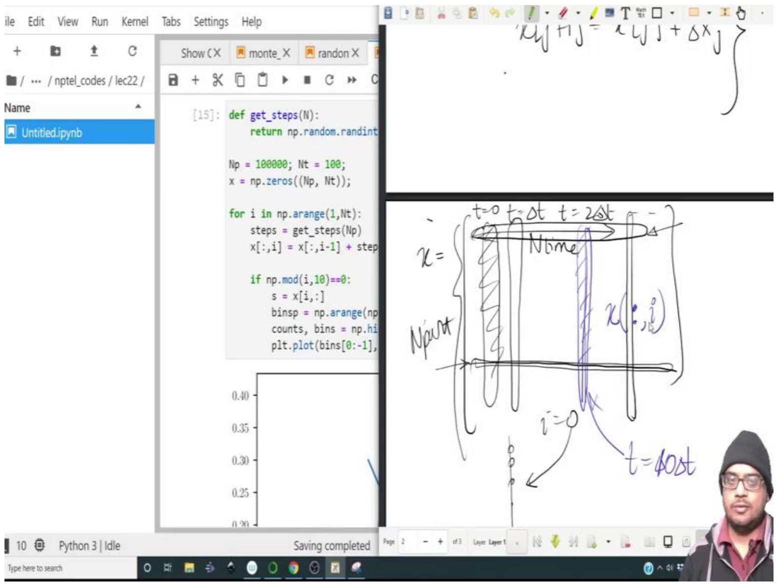


```
        counts, bins = np.histogram(s, bins=binsp, density = True)
        plt.plot(bins[0:-1], counts);
```

```
[13]: # Have a look at a trajectory
      ta = np.arange(0,Nt);
```

So, let us let us let us run this and see what happens, ok. We have something weird going on. So, why is that?

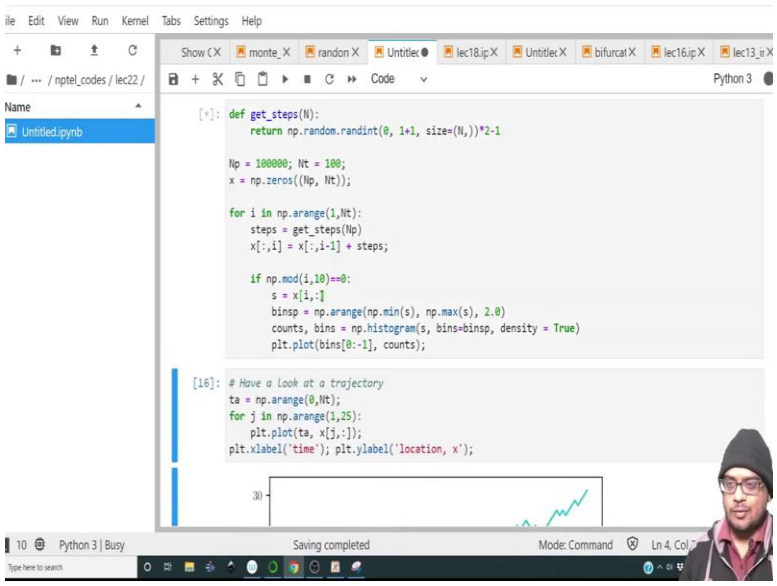
(Refer Slide Time: 24:55)



```
[15]: def get_steps(N):
          return np.random.randint

      Np = 100000; Nt = 100;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np)
          x[:,i] = x[:,i-1] + step

          if np.mod(i,10)==0:
              s = x[i,:]
              binsp = np.arange(np
              counts, bins = np.hi
              plt.plot(bins[0:-1],
```

So, a very critical mistake we have made is s = x[:,i] this should be, so essentially we are trying to get all of this and this is x all rows of the i-th column. So, rather than this it has to be well.

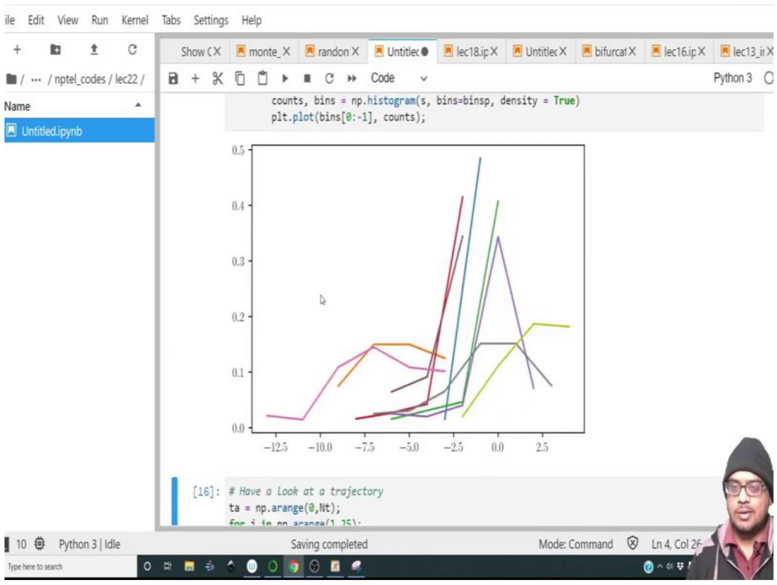
(Refer Slide Time: 25:05)



```
[*]: def get_steps(N):
         return np.random.randint(0, 1+1, size=(N,))*2-1

     Np = 100000; Nt = 100;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np)
         x[:,i] = x[:,i-1] + steps;

         if np.mod(i,10)==0:
             s = x[i,:]
             binsp = np.arange(np.min(s), np.max(s), 2.0)
             counts, bins = np.histogram(s, bins=binsp, density = True)
             plt.plot(bins[0:-1], counts);
```

```
[16]: # Have a Look at a trajectory
      ta = np.arange(0,Nt);
      for j in np.arange(1,25):
          plt.plot(ta, x[j,:]);
      plt.xlabel('time'); plt.ylabel('location, x');
```

(Refer Slide Time: 25:09)

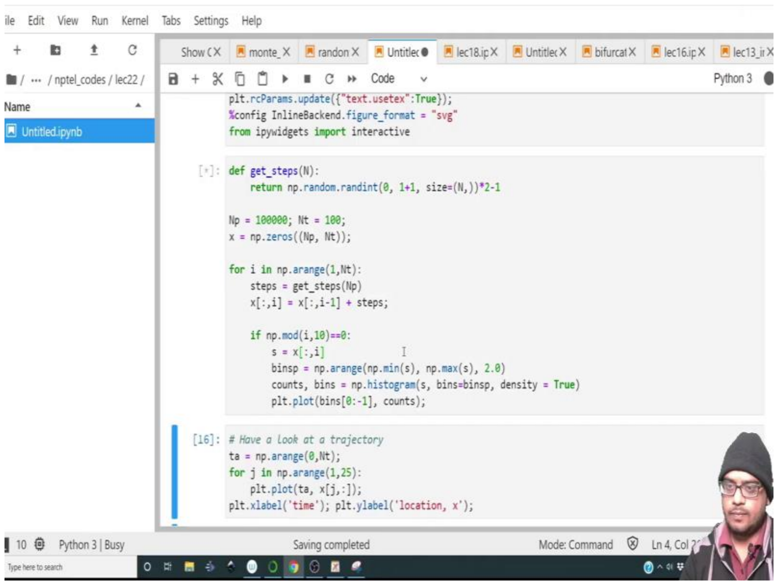


```
             counts, bins = np.histogram(s, bins=binsp, density = True)
             plt.plot(bins[0:-1], counts);
```

```
[16]: # Have a Look at a trajectory
      ta = np.arange(0,Nt);
```

In the heat of the moment, I might have written it reverse anyway no harm, done.

(Refer Slide Time: 25:15)



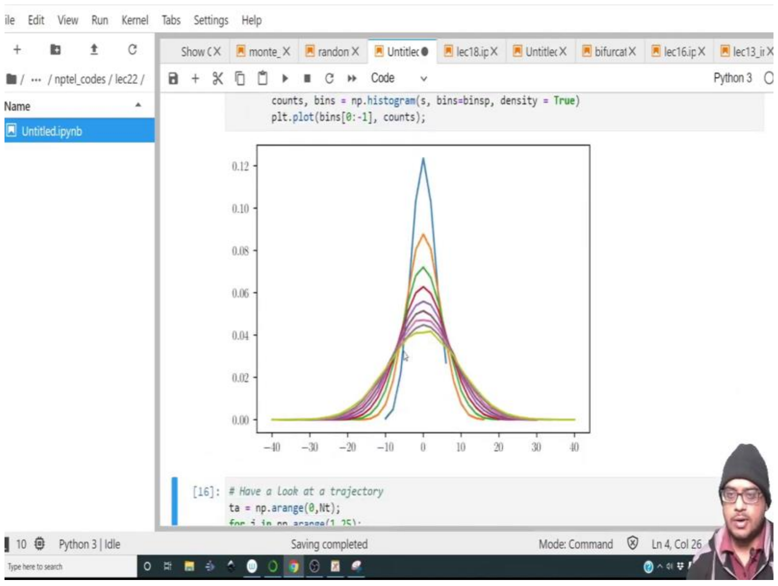
```
plt.rcParams.update({"text.usetex":True});
%config InlineBackend.figure_format = "svg"
from ipywidgets import interactive
```

```
[*]: def get_steps(N):
         return np.random.randint(0, 1+1, size=(N,))*2-1

     Np = 100000; Nt = 100;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np)
         x[:,i] = x[:,i-1] + steps;

         if np.mod(i,10)==0:
             s = x[:,i]
             binsp = np.arange(np.min(s), np.max(s), 2.0)
             counts, bins = np.histogram(s, bins=binsp, density = True)
             plt.plot(bins[0:-1], counts);
```

```
[16]: # Have a look at a trajectory
      ta = np.arange(0,Nt);
      for j in np.arange(1,25):
          plt.plot(ta, x[j,:]);
      plt.xlabel('time'); plt.ylabel('location, x');
```

File Edit View Run Kernel Tabs Settings Help
Show (monte_ randon Untitled lec18.ip Untitled bifurcat lec16.ip lec13_ir
/ … / nptel_codes / lec22 /
Name
Untitled.ipynb
Code
Python 3
10 Python 3 | Busy
Saving completed
Mode: Command
Ln 4, Col 2
Type here to search

(Refer Slide Time: 25:22)

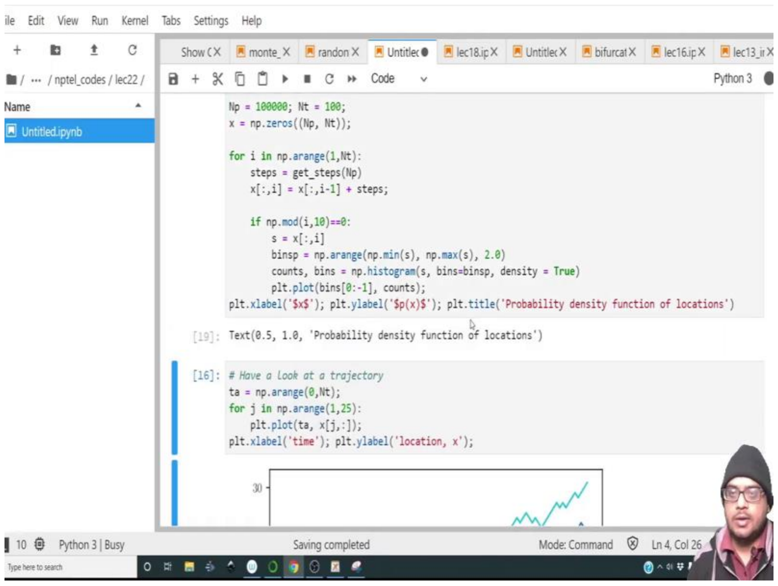


```
        counts, bins = np.histogram(s, bins=binsp, density = True)
        plt.plot(bins[0:-1], counts);
```

```
[16]: # Have a look at a trajectory
      ta = np.arange(0,Nt);
```

What did I do yeah, ok, great. So, this is how the evolution of the probability density function looks like.

(Refer Slide Time: 25:30)



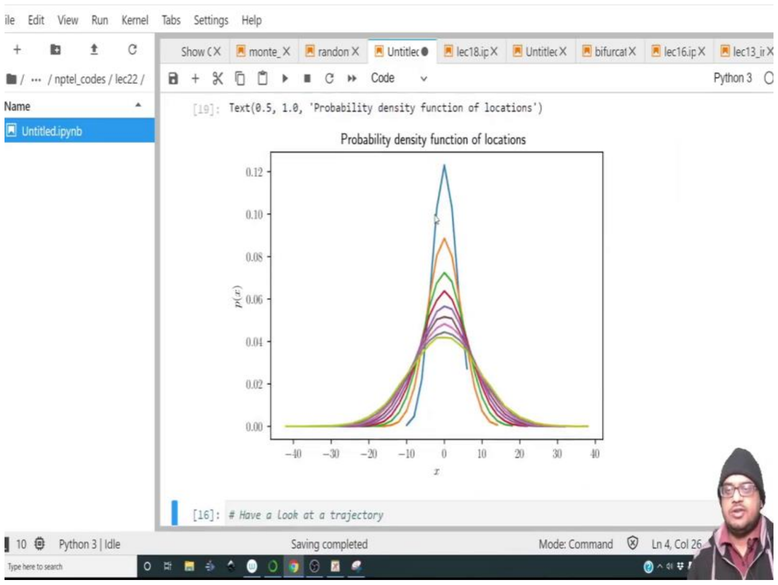
```
Np = 100000; Nt = 100;
x = np.zeros((Np, Nt));

for i in np.arange(1,Nt):
    steps = get_steps(Np)
    x[:,i] = x[:,i-1] + steps;

    if np.mod(i,10)==0:
        s = x[:,i]
        binsp = np.arange(np.min(s), np.max(s), 2.0)
        counts, bins = np.histogram(s, bins=binsp, density = True)
        plt.plot(bins[0:-1], counts);
plt.xlabel('$x$'); plt.ylabel('$p(x)$'); plt.title('Probability density function of locations')
```

```
[19]: Text(0.5, 1.0, 'Probability density function of locations')
```

```
[16]: # Have a look at a trajectory
ta = np.arange(0,Nt);
for j in np.arange(1,25):
    plt.plot(ta, x[j,:]);
plt.xlabel('time'); plt.ylabel('location, x');
```

In fact, let me give it the levels over here. So, plt.xlabel this is simply going to be plt.xlabel('x'); plt.ylabel('$p(x)$'); plt.title('Probability density function of locations'), alright.

(Refer Slide Time: 26:03)

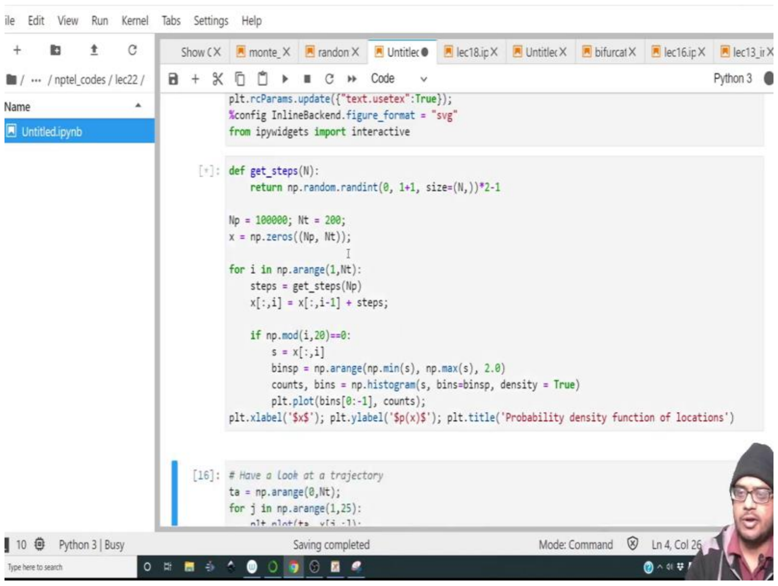


```
[19]: Text(0.5, 1.0, 'Probability density function of locations')
```

```
[16]: # Have a look at a trajectory
```

So, what do we see? We see that initially everything has peaked about 0 and still everything is still picked about 0, but as time progresses, the pdf sort of flattens out, it widens out.

(Refer Slide Time: 26:18)

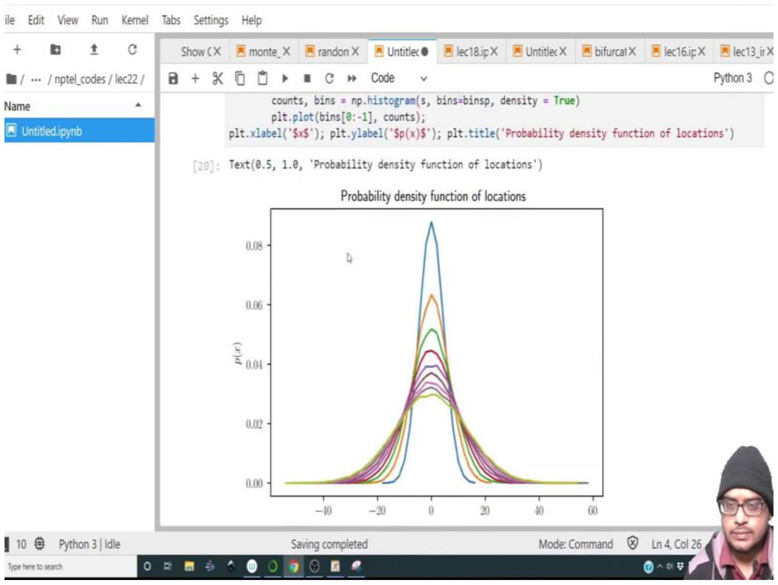


```
plt.rcParams.update({"text.usetex":True});
%config InlineBackend.figure_format = "svg"
from ipywidgets import interactive
```

```
[*]: def get_steps(N):
         return np.random.randint(0, 1+1, size=(N,))*2-1

     Np = 100000; Nt = 200;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np)
         x[:,i] = x[:,i-1] + steps;

         if np.mod(i,20)==0:
             s = x[:,i]
             binsp = np.arange(np.min(s), np.max(s), 2.0)
             counts, bins = np.histogram(s, bins=binsp, density = True)
             plt.plot(bins[0:-1], counts);
     plt.xlabel('$x$'); plt.ylabel('$p(x)$'); plt.title('Probability density function of locations')
```

```
[16]: # Have a look at a trajectory
      ta = np.arange(0,Nt);
      for j in np.arange(1,25):
```

(Refer Slide Time: 26:25)

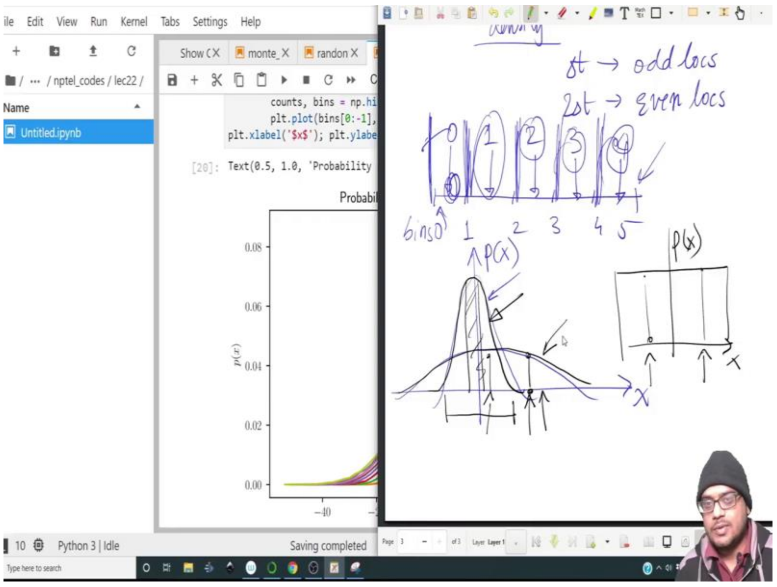


```
            counts, bins = np.histogram(s, bins=binsp, density = True)
            plt.plot(bins[0:-1], counts);
plt.xlabel('$x$'); plt.ylabel('$p(x)$'); plt.title('Probability density function of locations')
```

```
[20]: Text(0.5, 1.0, 'Probability density function of locations')
```

And let me increase the number of time or let me do this and let me take a plot every, yeah. So, this is how the evolution happens the pdf peak becomes smaller. So, what is the meaning of the fact that the pdf peak becomes smaller?

(Refer Slide Time: 26:38)



```
        counts, bins = np.hi
        plt.plot(bins[0:-1],
plt.xlabel('$x$'); plt.ylabe
```

```
[20]: Text(0.5, 1.0, 'Probability
```

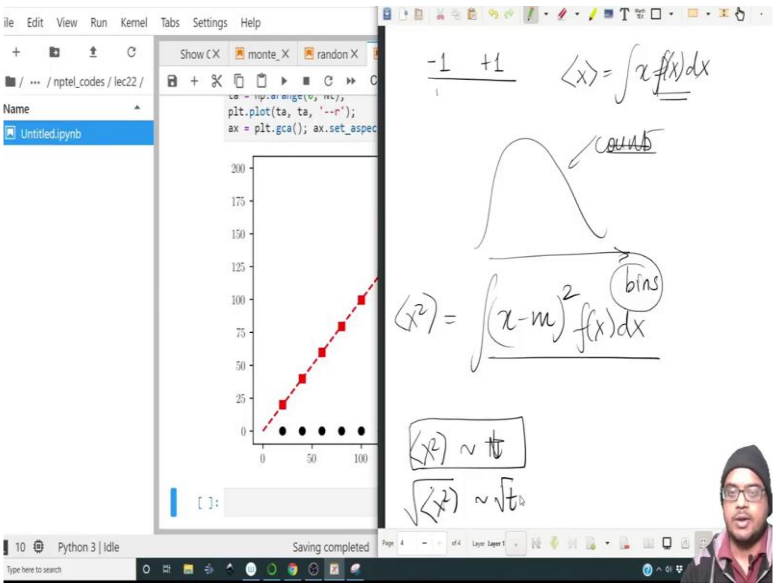
So, let us plot this over here. So, what is the difference between this pdf and this pdf? This pdf means; so, on the x axis we have x, on the y axis we have P(x) that is the probability density function of x.

So, in this pdf what it implies is the these are the most likely values that you are going to encounter because of this distribution and these are the highly probable values, but when the probability becomes something like this, so in this particular pdf you would not expect values of x over here because the probability is going to 0 over there.

But in this particular flat pdf the probability of finding a value over here and over here is not that different because it is a very flat pdf. In fact, a uniform pdf looks something like this, this is x this is P(x). So, this is how the pdf for uniform distribution looks like it is equally likely for this value and this value to occur in fact, for any value to occur, ok. So, the pointier the pdf it is more likely to fetch you values near the point location. The flatter the pdf is its more likely to get you a range of values in that flat location, ok.

So, but still regardless of that it is it is seen that the locations near the origin are more likely to come than locations away from the origin, ok. And all this is a consequence of the central limit theorem which means which is says that if you have a process in which you are sampling steps from a distribution which has a finite variance, then you are going to end up with the distribution of locations which is described by the normal distribution, ok.

(Refer Slide Time: 28:42)



```
plt.plot(ta, ta, '--r');
ax = plt.gca(); ax.set_aspec
```

$-1 \quad +1$

$\langle x \rangle = \int x f(x)\,dx$

counts

bins

$\langle x^2 \rangle = \int (x-m)^2 f(x)\,dx$

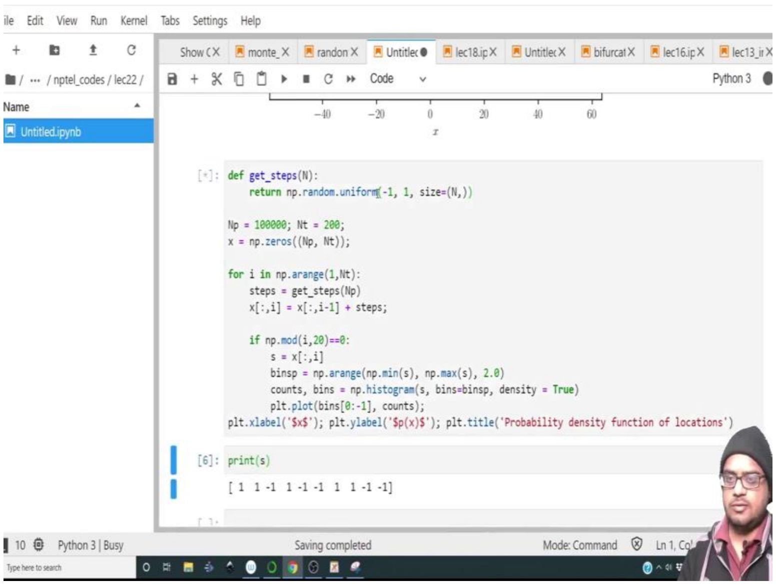
$\boxed{\langle x^2 \rangle \sim t}$

$\sqrt{\langle x^2 \rangle} \sim \sqrt{t}$

So, in this particular case, we are sampling the steps from a very simple distribution it is a binary distribution its either 1 or -1. So, the variance of this distribution is bounded meaning in this particular distribution you do not you have a well-defined mean the mean is 0 and you have a variance as well you can easily find it out.

So, given this distribution which has a finite variance you are going to end up with a distribution which is a normal distribution. A normal distribution is also the name of a Gaussian. So, now if we chose any other distribution which has a finite variance would we still get this that is the question. In order to find that out let us copy this code.

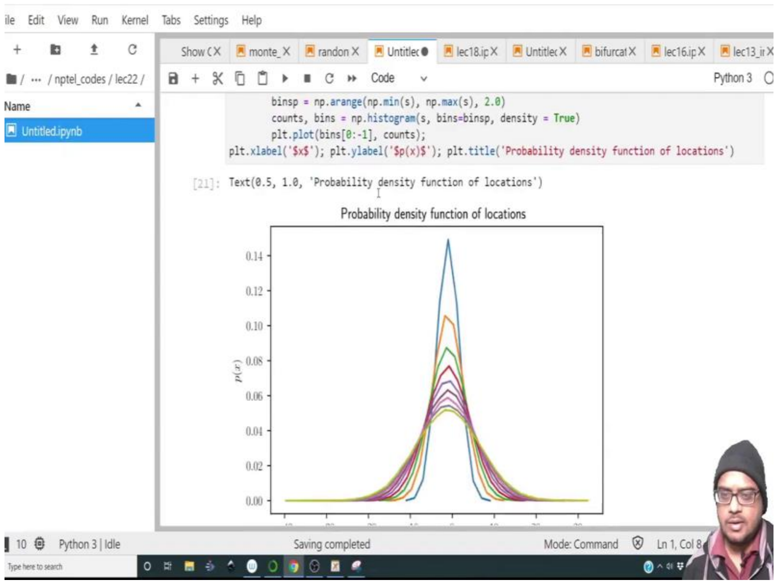
(Refer Slide Time: 29:33)



```
[*]: def get_steps(N):
         return np.random.uniform(-1, 1, size=(N,))

     Np = 100000; Nt = 200;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np)
         x[:,i] = x[:,i-1] + steps;

         if np.mod(i,20)==0:
             s = x[:,i]
             binsp = np.arange(np.min(s), np.max(s), 2.0)
             counts, bins = np.histogram(s, bins=binsp, density = True)
             plt.plot(bins[0:-1], counts);
     plt.xlabel('$x$'); plt.ylabel('$p(x)$'); plt.title('Probability density function of locations')

[6]: print(s)

     [ 1  1 -1  1 -1 -1  1  1 -1 -1]
```

Let us get rid of these trajectories. We do not need the trajectories of now as of now. Let me sample from a uniform distribution and the uniform distribution goes from -1 to 1, right.

(Refer Slide Time: 29:46)



```
        binsp = np.arange(np.min(s), np.max(s), 2.0)
        counts, bins = np.histogram(s, bins=binsp, density = True)
        plt.plot(bins[0:-1], counts);
plt.xlabel('$x$'); plt.ylabel('$p(x)$'); plt.title('Probability density function of locations')
```

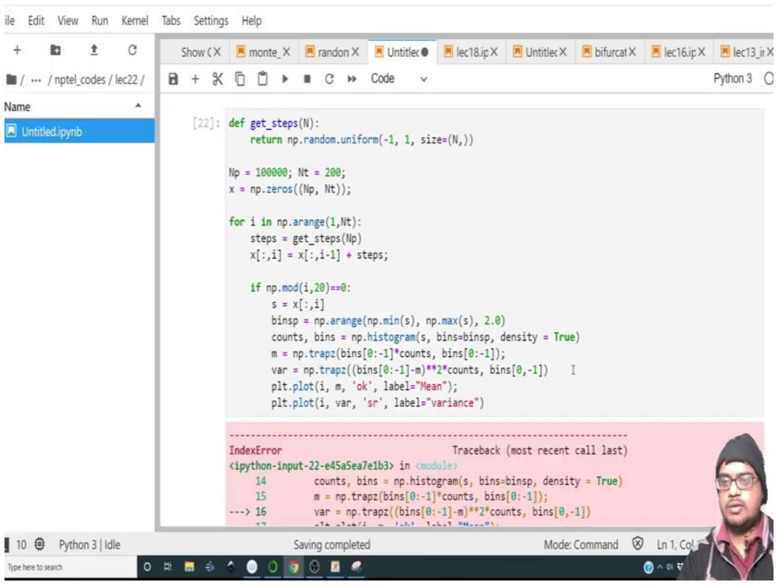
```
[21]: Text(0.5, 1.0, 'Probability density function of locations')
```

And let us run this also. Let us see what happens. And again, we do get a normal distribution. As of I am saying a normal distribution, but how do you know it is a normal distribution, for all you know it is not a normal distribution, ok. We will we will show how we can figure out it is a normal distribution or not. For now, you just take my word it is a normal distribution.

So, what can we do now? We have shown that given any distribution it converges to the normal distribution the key point is that the distribution from which we sample the step size it has to be bounded meaning it has to have a finite variance.

If it does not have a finite variance then it will not converge to the central converge to the normal distribution it will not follow the central limit theorem. In that case, we have to generalize the central limit theorem to find out what it does converge to. But through this very simple program I have shown you how you can find out such kinds of simple random walks. Now, let us take this step forward and let us try to find out the mean and the variance for all the different look, all the different distributions that we have.

(Refer Slide Time: 31:09)



```
[22]: def get_steps(N):
          return np.random.uniform(-1, 1, size=(N,))

      Np = 100000; Nt = 200;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np)
          x[:,i] = x[:,i-1] + steps;

          if np.mod(i,20)==0:
              s = x[:,i]
              binsp = np.arange(np.min(s), np.max(s), 2.0)
              counts, bins = np.histogram(s, bins=binsp, density = True)
              m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
              var = np.trapz((bins[0:-1]-m)**2*counts, bins[0,-1])
              plt.plot(i, m, 'ok', label="Mean");
              plt.plot(i, var, 'sr', label="variance")
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-22-e45a5ea7e1b3> in <module>
     14             counts, bins = np.histogram(s, bins=binsp, density = True)
     15             m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
---> 16             var = np.trapz((bins[0:-1]-m)**2*counts, bins[0,-1])
```

So, now given; so, we do not want to, ok. Let me copy this function again we let us go to a new cell you know in this cell we will try to find out the variance and the mean of this evolving distribution.

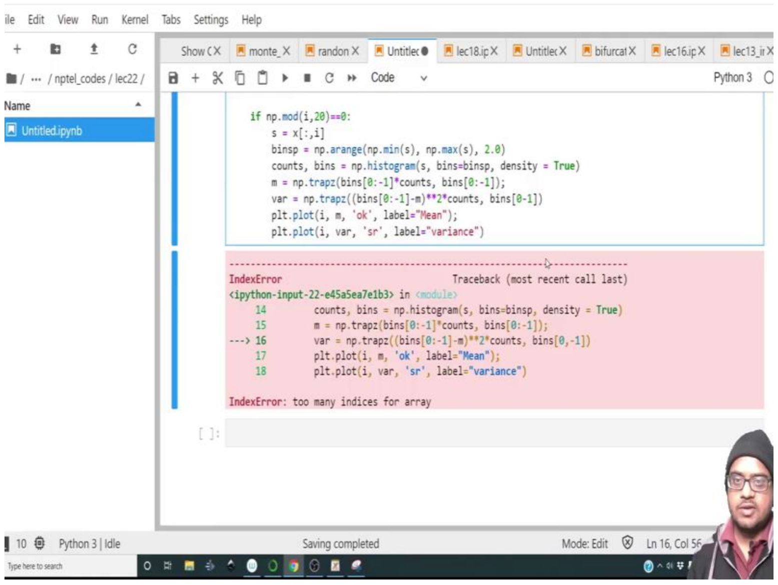
So, let me get rid of these plots. So, what we have is the distribution. What is the distribution? It is a function of bins and counts, ok. So, in order to find out the mean, so let me call the mean as m it will be simply the integral of. So, what is the mean? Let us let me write down the definition.

So, the mean of $<x>=\int xf(x)dx$ or P(x) dx. So, the x over here is bins and the f (x) is the counts. So, np.trapz over here it will be; so, we are trying to find out bins[0:-1]*counts, bins[0:-1] because the syntax for finding out the integration by trapezoidal rule is that you must have y first and then x. So, this gives us the evolution of the mean.

Now, once we have the evolution of the mean, what let us have the evolution of the variance. So, the variance will be np.trapz. So, now, this will be something comma bins 0 to -1 that something will be multiplied by the pdf. The pdf is the counts in this case and over here we will have bins 0 to -1 - $<x^2>$. So, essentially what we are doing it is the variance will be $<x^2>=\int(x-m)^2 f(x)dx$. So, just verify this is the same as this, alright.

So, now, that we have expressions for the mean and the variance let us do a plot of these. So, plt.plot; so, we will simply do a plt.plot(i, m, 'ok', label="Mean"), this is going to be mean. We will do plt.plot(i, var, 'sr', label="variance").

(Refer Slide Time: 34:00)

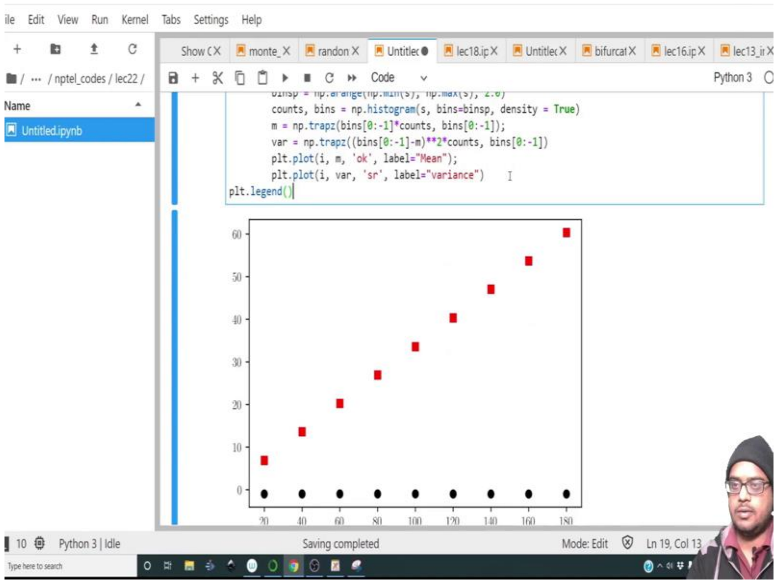


```
    if np.mod(i,20)==0:
        s = x[:,i]
        binsp = np.arange(np.min(s), np.max(s), 2.0)
        counts, bins = np.histogram(s, bins=binsp, density = True)
        m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
        var = np.trapz((bins[0:-1]-m)**2*counts, bins[0-1])
        plt.plot(i, m, 'ok', label="Mean");
        plt.plot(i, var, 'sr', label="variance")
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-22-e45a5ea7e1b3> in <module>
     14         counts, bins = np.histogram(s, bins=binsp, density = True)
     15         m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
---> 16         var = np.trapz((bins[0:-1]-m)**2*counts, bins[0,-1])
     17         plt.plot(i, m, 'ok', label="Mean");
     18         plt.plot(i, var, 'sr', label="variance")

IndexError: too many indices for array
```

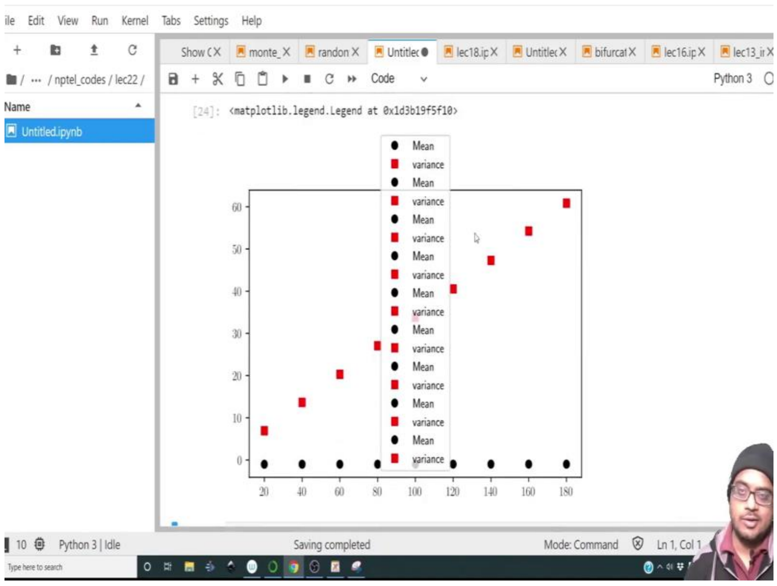
So, let me run this. There appears to be an error; too many, this has to be comma, this has to be a column, ok.

(Refer Slide Time: 34:10)



```
        counts, bins = np.histogram(s, bins=binsp, density = True)
        m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
        var = np.trapz((bins[0:-1]-m)**2*counts, bins[0:-1])
        plt.plot(i, m, 'ok', label="Mean");
        plt.plot(i, var, 'sr', label="variance")
plt.legend()
```

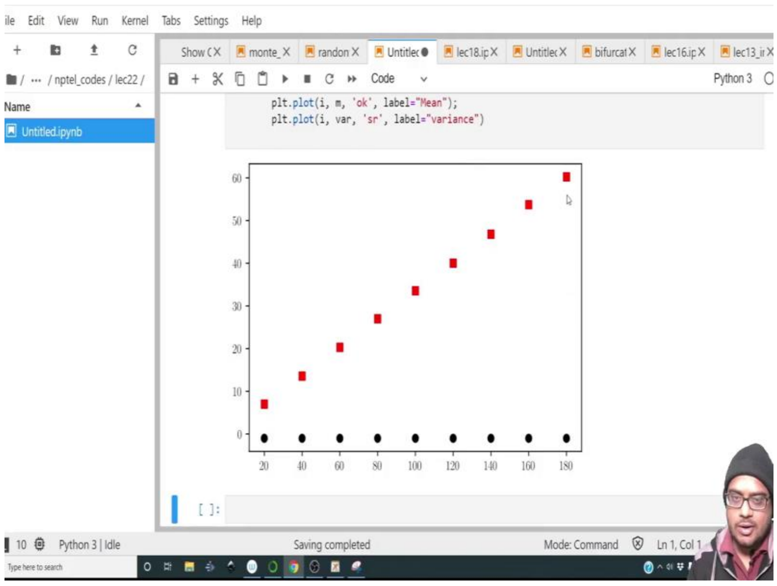
(Refer Slide Time: 34:23)



```
[24]: <matplotlib.legend.Legend at 0x1d3b19f5f10>
```

So, what do we see? Let me do plt.legend. Oops we have too many legends, alright. Anyway, it is using each level every time, but anyway.

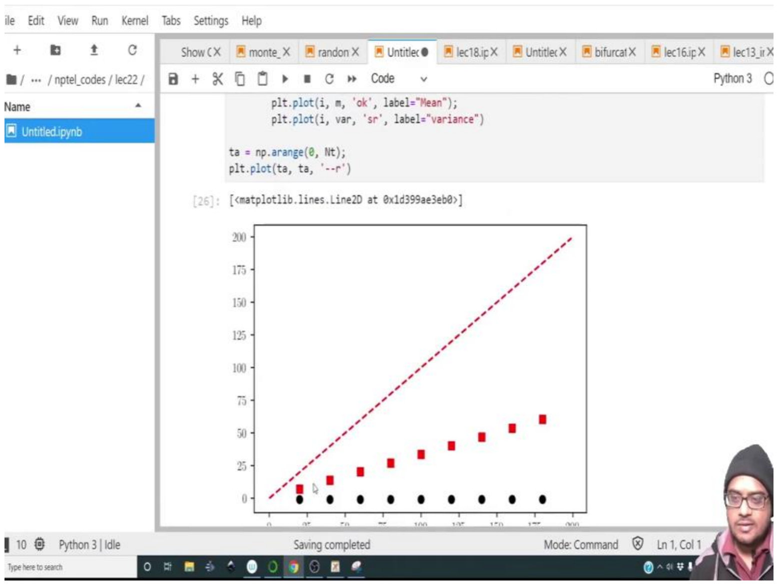
(Refer Slide Time: 34:32)



```
    plt.plot(i, m, 'ok', label="Mean");
    plt.plot(i, var, 'sr', label="variance")
```

So, the variance is the square red marker it increases linearly in time. So, how do we know it is linearly in time? We will plot the linear function as well whereas; the mean is always hovering near 0, ok. So, on top of this let us plot a linear trend line as well, ok.

(Refer Slide Time: 34:56)

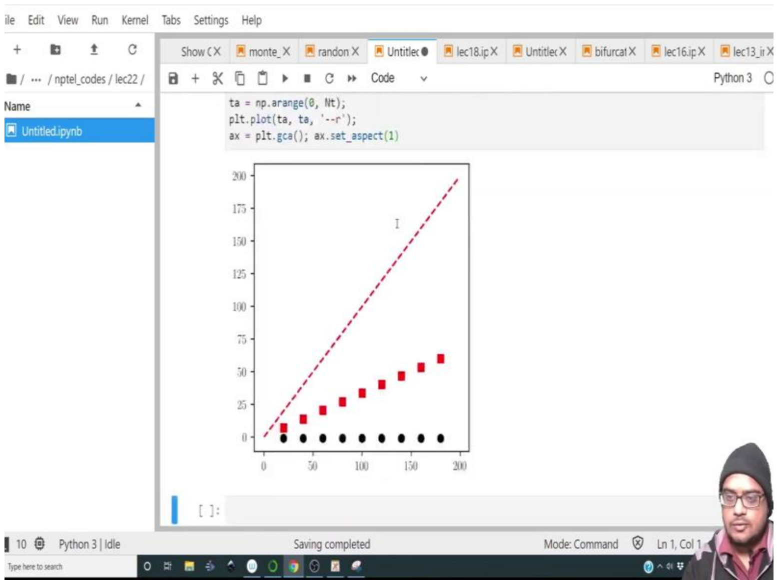


```
    plt.plot(i, m, 'ok', label="Mean");
    plt.plot(i, var, 'sr', label="variance")

ta = np.arange(0, Nt);
plt.plot(ta, ta, '--r')
```

```
[26]: [<matplotlib.lines.Line2D at 0x1d399ae3eb0>]
```

So, let me do ta again is going to be ta = np.arange(0, Nt); plt.plot(ta, ta, '--r') . Let me plot this as; so, this is just to give a trend line. So, this is a linear trend line, ok. So, the variance is having a different slope than y = x line.

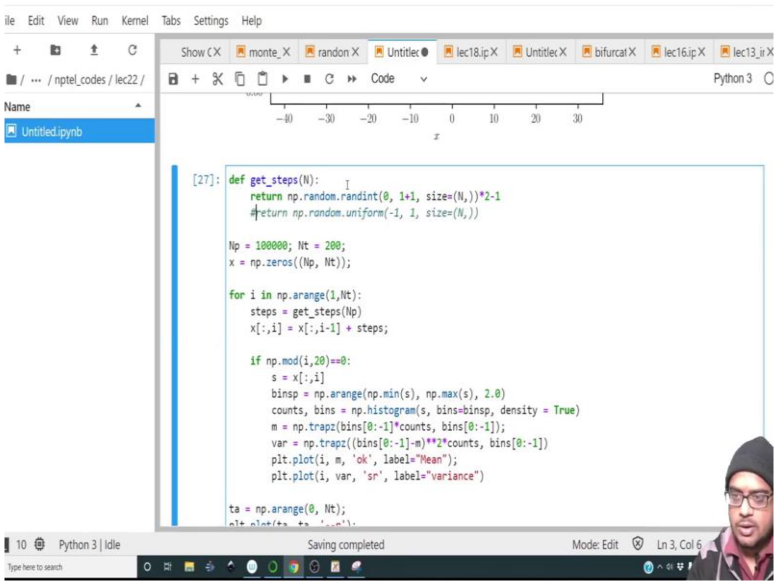
(Refer Slide Time: 35:28)



```
ta = np.arange(0, Nt);
plt.plot(ta, ta, '--r');
ax = plt.gca(); ax.set_aspect(1)
```

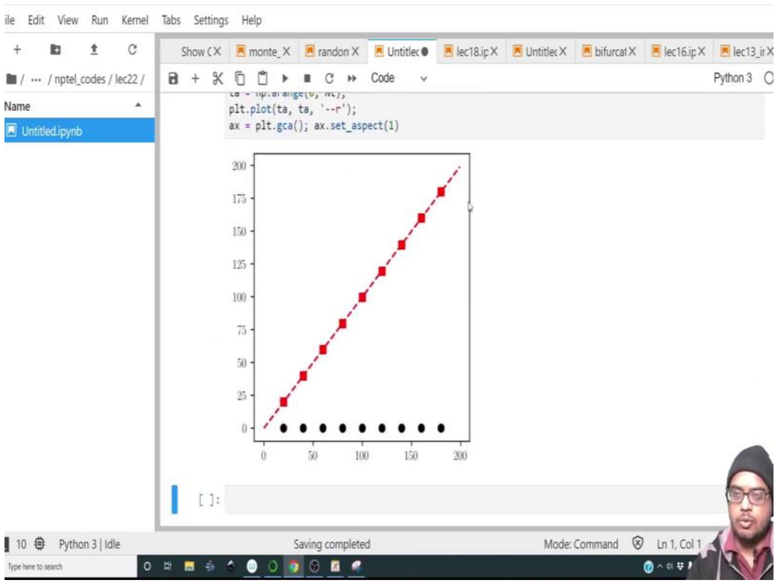
Let me put the aspect ratio of the plot proper. So, ax = plt.gca(); ax.set_aspect(1), ok. So, with the help of this now we can ascertain what the slope has to be. Well, the variance of the uniform distribution, ok. Let me first show you what happens when the distribution is same Bernoulli trial as the first problem that we saw.

(Refer Slide Time: 36:04)



```
[27]: def get_steps(N):
          return np.random.randint(0, 1+1, size=(N,))*2-1
          #return np.random.uniform(-1, 1, size=(N,))

      Np = 100000; Nt = 200;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np)
          x[:,i] = x[:,i-1] + steps;

          if np.mod(i,20)==0:
              s = x[:,i]
              binsp = np.arange(np.min(s), np.max(s), 2.0)
              counts, bins = np.histogram(s, bins=binsp, density = True)
              m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
              var = np.trapz((bins[0:-1]-m)**2*counts, bins[0:-1])
              plt.plot(i, m, 'ok', label="Mean");
              plt.plot(i, var, 'sr', label="variance")

      ta = np.arange(0, Nt);
```

(Refer Slide Time: 36:09)

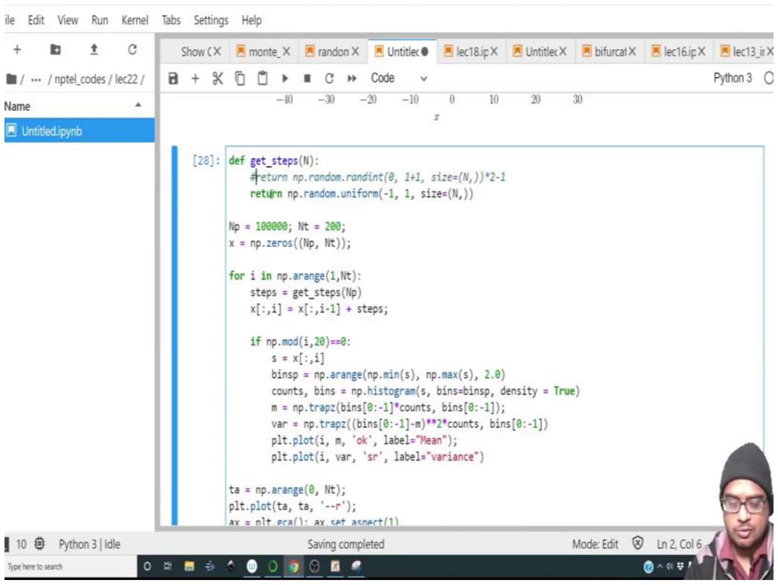


```
plt.plot(ta, ta, '--r');
ax = plt.gca(); ax.set_aspect(1)
```

Then we will look at the uniform distribution and there is a small subtlety which gives rise to the difference in the slope. Let me run this, boom. Everything lies on the straight line and this does mean that the variance goes as the time or the number of time steps. So, this is called as a diffusive process. So, in a diffusion process the rms distance goes as time. Well, the rms goes distance goes a square root of time, but anyway. If you take a square root of this it is like an rms its going as square root of time.

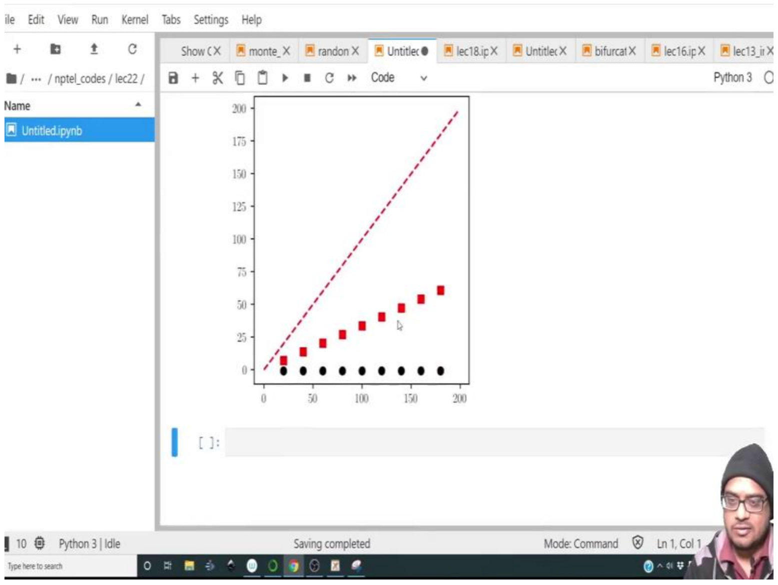
So, the point I am trying to make is in this particular case the variance is 1, but in the uniform distribution the variance is not 1 and that gave rise to a different slope. So, now, you may ask the question wait what is the then variance of this uniform distribution. What is the variance of this uniform distribution?

(Refer Slide Time: 36:57)

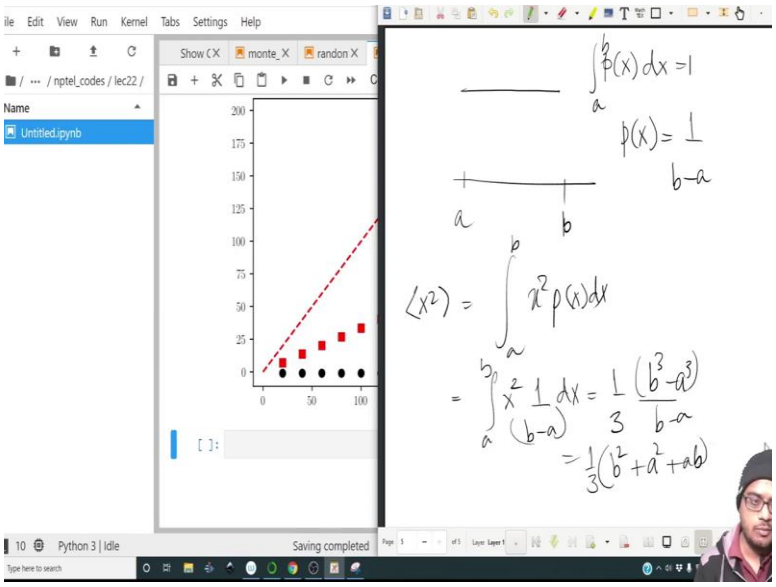


```
[28]: def get_steps(N):
          #return np.random.randint(0, 1+1, size=(N,))*2-1
          return np.random.uniform(-1, 1, size=(N,))

      Np = 100000; Nt = 200;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np)
          x[:,i] = x[:,i-1] + steps;

          if np.mod(i,20)==0:
              s = x[:,i]
              binsp = np.arange(np.min(s), np.max(s), 2.0)
              counts, bins = np.histogram(s, bins=binsp, density = True)
              m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
              var = np.trapz((bins[0:-1]-m)**2*counts, bins[0:-1])
              plt.plot(i, m, 'ok', label="Mean");
              plt.plot(i, var, 'sr', label="variance")

      ta = np.arange(0, Nt);
      plt.plot(ta, ta, '--r');
      ax = plt.gca(); ax.set_aspect(1)
```

(Refer Slide Time: 37:00)



(Refer Slide Time: 37:05)

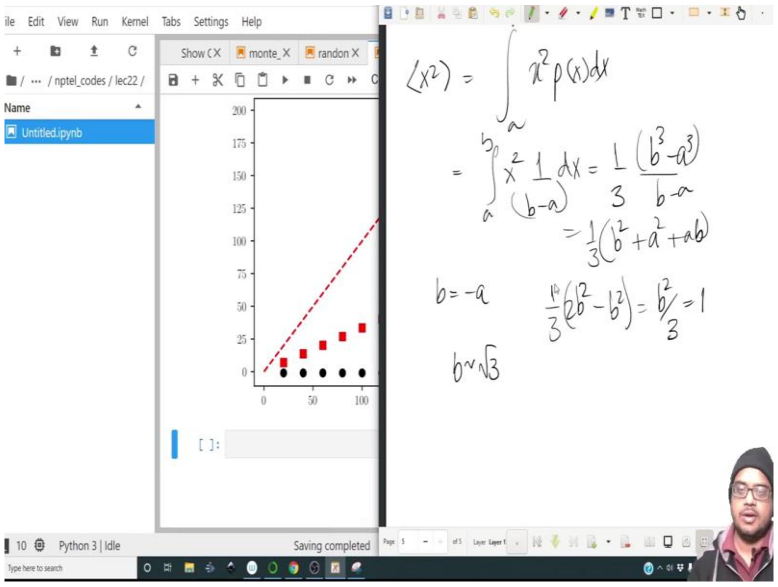


$\int_a^b p(x)\,dx = 1$
$p(x) = \frac{1}{b-a}$
a
b
$\langle x^2\rangle = \int_a^b x^2 p(x)\,dx$
$= \int_a^b x^2 \frac{1}{(b-a)}\,dx = \frac{1}{3}\frac{(b^3-a^3)}{b-a}$
$= \frac{1}{3}(b^2+a^2+ab)$

So, let us find it out. So, if you have a uniform distribution between a and b, the distribution will look something like this. So, we know that $\int_a^b p(x)dx = 1$ and this implies that $p(\text{x}) = \frac{1}{b-a}$. So, this is how we define a uniform distribution.

So, what is the variance of the uniform distribution? It is going to be $< x^2 >= \int_a^b x^2 p(x)dx$. So, this is going to be $\int_a^b x^2 \frac{1}{b-a}dx$. So, it is going to be $\frac{1}{3}\frac{(\text{b}^3 - \text{a}^3)}{b-a}$. This is going to be $\frac{1}{3}(\text{b}^2 + a^2 + ab)$. So, b-a cancels out from the numerator and denominator.

(Refer Slide Time: 37:58)



$$\langle x^2 \rangle = \int_a x^2 p(x)\,dx$$

$$= \int_a^b x^2 \frac{1}{(b-a)}\,dx = \frac{1}{3}\frac{(b^3-a^3)}{b-a}$$

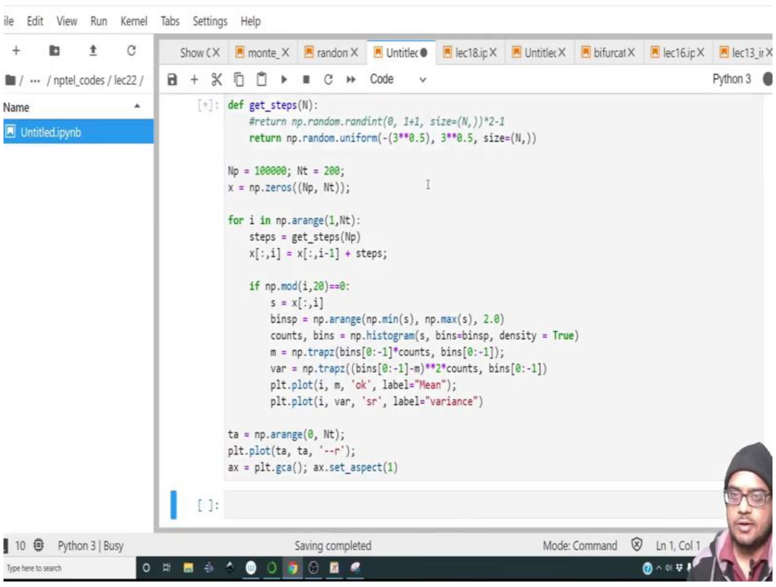
$$= \frac{1}{3}(b^2+a^2+ab)$$

$$b=-a \qquad \frac{1}{3}(2b^2-b^2) = b^2/3 = 1$$

$$b \sim \sqrt{3}$$

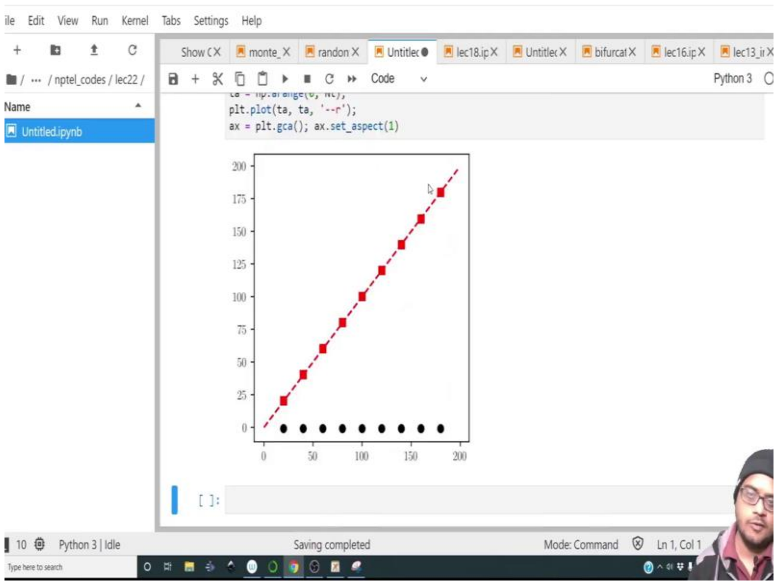
So, now if b = -a that is for a symmetric distribution, what do we have? $\frac{1}{3}(2b^2 - b^2)$. So, it is essentially $\frac{b^2}{3}$. If we require the variance to be 1, then what do we have? Over here we will have $b \sim \sqrt{3}$. Rather the distribution the uniform distribution has to sample from $\pm\sqrt{3}$, alright.

(Refer Slide Time: 38:33)



```
def get_steps(N):
    #return np.random.randint(0, 1+1, size=(N,))*2-1
    return np.random.uniform(-(3**0.5), 3**0.5, size=(N,))

Np = 100000; Nt = 200;
x = np.zeros((Np, Nt));

for i in np.arange(1,Nt):
    steps = get_steps(Np)
    x[:,i] = x[:,i-1] + steps;

    if np.mod(i,20)==0:
        s = x[:,i]
        binsp = np.arange(np.min(s), np.max(s), 2.0)
        counts, bins = np.histogram(s, bins=binsp, density = True)
        m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
        var = np.trapz((bins[0:-1]-m)**2*counts, bins[0:-1])
        plt.plot(i, m, 'ok', label="Mean");
        plt.plot(i, var, 'sr', label="variance")

ta = np.arange(0, Nt);
plt.plot(ta, ta, '--r');
ax = plt.gca(); ax.set_aspect(1)
```

(Refer Slide Time: 38:49)

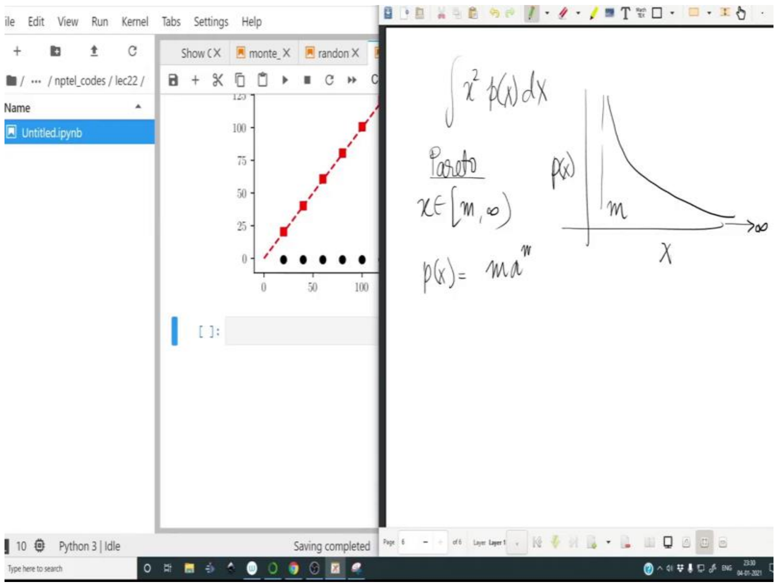


```
plt.plot(ta, ta, '--r');
ax = plt.gca(); ax.set_aspect(1)
```

So, let me now run this and see whether we get something, boom, we get the same trend line. If I choose this to be something else I will have a different slope, right. So, now this is how you can now do numerical experiments you can choose a wide range of different step size distributions. So, so far we have focused on step sizes which are not step sizes, but steps which are bounded.

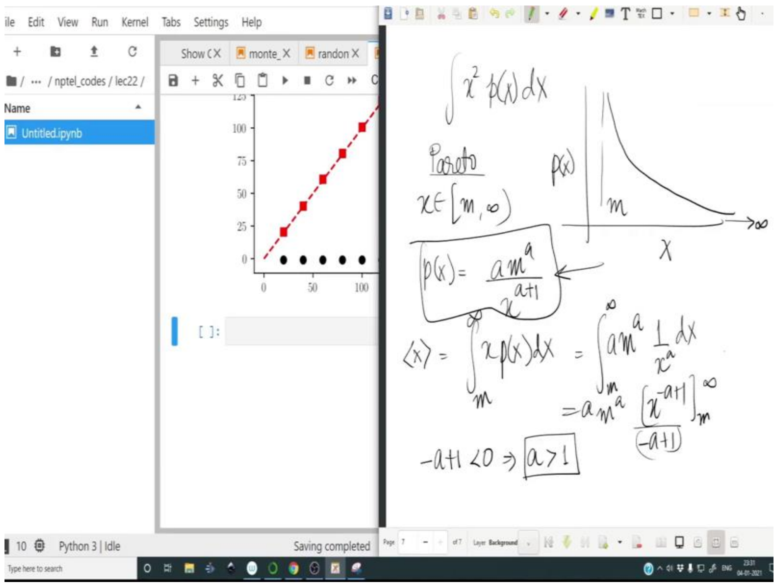
But what happens when in; by bounded I mean when the variance exists it is not undefined, but we can have the case where the sampling of the step size is from a distribution which does not have a variance when can that happen. Let us look at the definition of variance in that case.

(Refer Slide Time: 39:54)



So, we have something which is something like this. Now, the variance may not exist for a wide variety of distributions and in general a power law distribution. The most common distribution that is used is the pareto distribution. So, pareto distribution has a power law or tail something like this and it ranges from m to infinity. So, pareto distribution the variable lies between $[m,\infty)$. And it has a form $p(x) = \frac{am^a}{x^{a+1}}$, yeah.

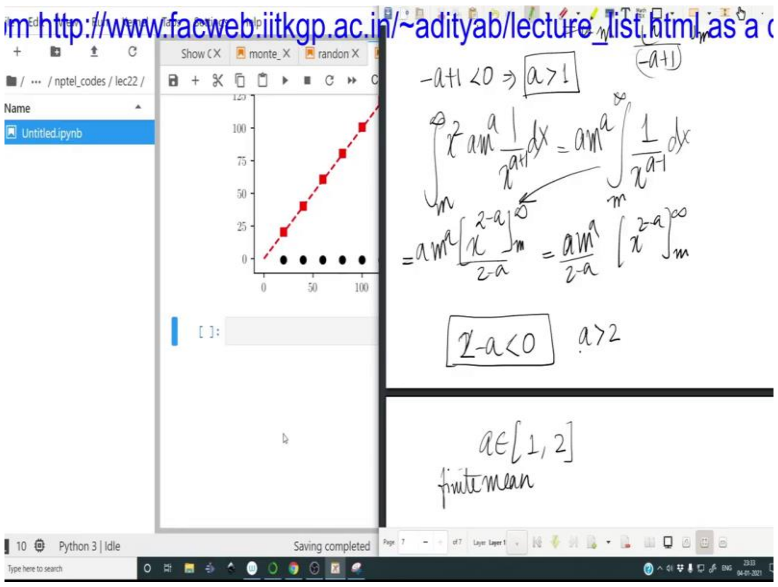
(Refer Slide Time: 40:34)


$\int x^2 p(x)\,dx$
Pareto
$x \in [m, \infty)$
$\boxed{p(x) = \frac{am^a}{x^{a+1}}}$
$p(x)$
m
x
$\to \infty$
$\langle x \rangle = \int_m^\infty x\,p(x)\,dx = \int_m^\infty am^a \frac{1}{x^a}\,dx$
$= am^a \left[\frac{x^{-a+1}}{(-a+1)}\right]_m^\infty$
$-a+1 < 0 \Rightarrow \boxed{a > 1}$

So, now let us see what is the condition with the mean. So, the $< x >= \int_m^\infty xp(x)dx$, and so this is going to be $\int_m^\infty am^a \frac{1}{x^a}dx$. So, this is a; so this goes from m to ∞, am^a, so $\frac{[x^{-a+1}]_m^\infty}{-a+1}$.

So, now, if the limit at ∞ has to exist then $-a+1<0$ which implies that $a>1$. So, if given this distribution if $a>1$, then the mean exists. Now, with the similar logic you can clearly see where we are going with this when does the variance not exist, ok. So, let us do that.

(Refer Slide Time: 42:03)


http://www.facweb.iitkgp.ac.in/~adityab/lecture_list.html
$-a+1<0 \Rightarrow \boxed{a>1}$
$\int_m^\infty x^2 am^a \frac{1}{x^{a+1}}dx = am^a \int_m^\infty \frac{1}{x^{a-1}}dx$
$= am^a \left[\frac{x^{2-a}}{2-a}\right]_m^\infty = \frac{am^a}{2-a}\left[x^{2-a}\right]_m^\infty$
$\boxed{2-a<0}$ $a>2$
$a\in[1,2]$
finite mean

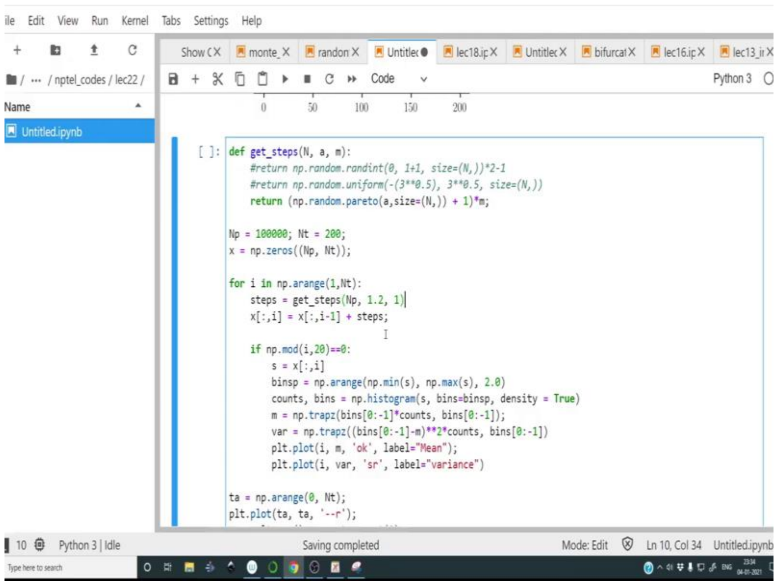
So, $\int_m^\infty x^2 am^a \frac{1}{x^{a+1}}dx$. So, what is this going to be? $am^a \int_m^\infty \frac{1}{x^{a-1}}dx$, so this. So, now, this is going to be x^{1-a}. So, this is going to be $am^a \frac{[x^{2-a}]_m^\infty}{2-a}$.

So, this expression simplifies to this now this $\frac{am^a}{2-a}[x^{2-a}]_m^\infty$, now this is going to exist when $2-a<0$ and the reason I am saying that that $2-a<0$ is because the limited infinity has to be non-divergent if $2-a>0$ then x to the power something greater than 0 will be ∞ at ∞.

But if x if $2-a<0$ then the large number raised to this exponent which is less than 0 that will decay that will become 0. So, then bound at infinity will be well behaved and that gives rise to this kind of constraint. This implies a > 2.

So, we can say that if $a \in [1,2]$ then the distribution has a finite mean, but the variance is not defined in that particular case does the distribution converge to you know normal distribution or not that is the question. And the answer is obviously, not. But, how does the distribution look like?

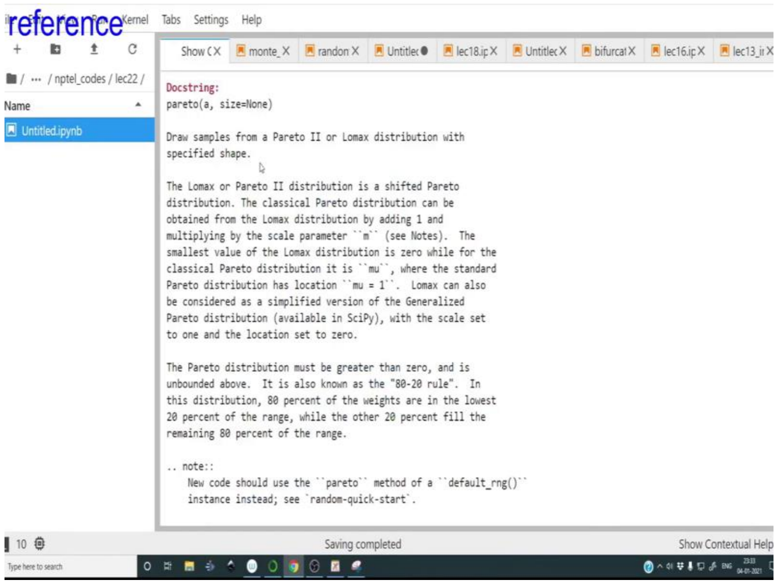
(Refer Slide Time: 44:01)



```
[ ]: def get_steps(N, a, m):
         #return np.random.randint(0, 1+1, size=(N,))*2-1
         #return np.random.uniform(-(3**0.5), 3**0.5, size=(N,))
         return (np.random.pareto(a,size=(N,)) + 1)*m;

     Np = 100000; Nt = 200;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np, 1.2, 1)
         x[:,i] = x[:,i-1] + steps;

         if np.mod(i,20)==0:
             s = x[:,i]
             binsp = np.arange(np.min(s), np.max(s), 2.0)
             counts, bins = np.histogram(s, bins=binsp, density = True)
             m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
             var = np.trapz((bins[0:-1]-m)**2*counts, bins[0:-1])
             plt.plot(i, m, 'ok', label="Mean");
             plt.plot(i, var, 'sr', label="variance")

     ta = np.arange(0, Nt);
     plt.plot(ta, ta, '--r');
```

So, let me grab this real quick. Let me go over here and instead of this let us get a pareto distribution.

(Refer Slide Time: 44:13)

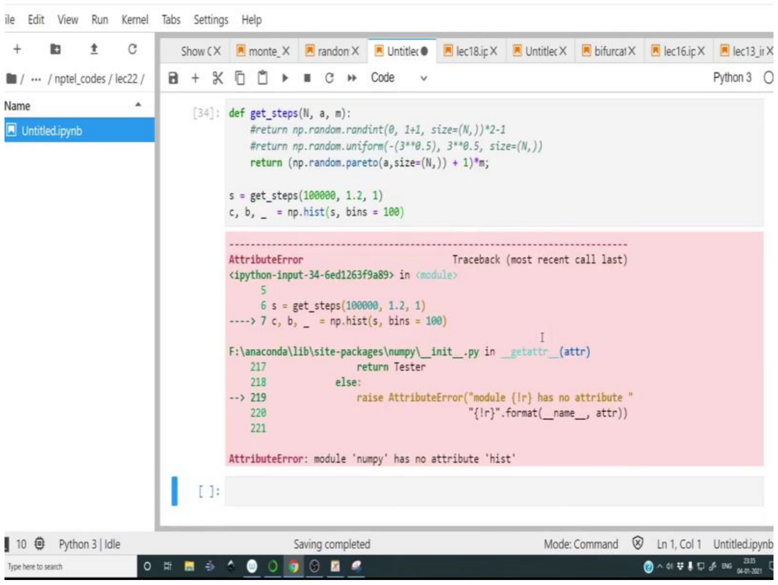


```
Docstring:
pareto(a, size=None)

Draw samples from a Pareto II or Lomax distribution with
specified shape.

The Lomax or Pareto II distribution is a shifted Pareto
distribution. The classical Pareto distribution can be
obtained from the Lomax distribution by adding 1 and
multiplying by the scale parameter ``m`` (see Notes).  The
smallest value of the Lomax distribution is zero while for the
classical Pareto distribution it is ``mu``, where the standard
Pareto distribution has location ``mu = 1``.  Lomax can also
be considered as a simplified version of the Generalized
Pareto distribution (available in SciPy), with the scale set
to one and the location set to zero.

The Pareto distribution must be greater than zero, and is
unbounded above.  It is also known as the "80-20 rule".  In
this distribution, 80 percent of the weights are in the lowest
20 percent of the range, while the other 20 percent fill the
remaining 80 percent of the range.

.. note::
    New code should use the ``pareto`` method of a ``default_rng()``
    instance instead; see `random-quick-start`.
```

So, you must return np.random.pareto and the parameters for the pareto distribution are a comma size, ok. So, it will be a,size=(N,), and it has to we have to add one for giving the scale factor and multiply everything by m, alright. So, this must now also accept a, m, right, no problem.

So, now, while calling the steps I must also pass, so let a be 1.2 and let m be 1. But now what is going on? It is a one-sided distribution meaning that, ok. Before all that let me show you how a pareto distribution looks like.

(Refer Slide Time: 45:05)

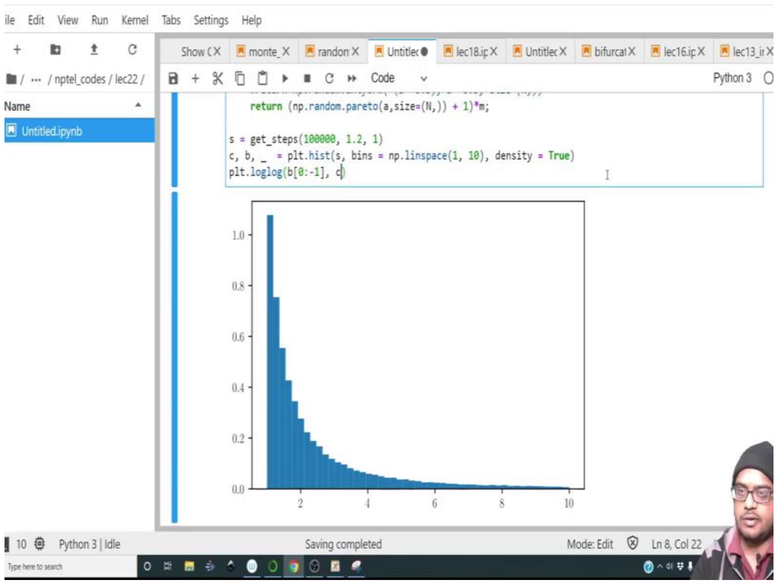


```
[34]: def get_steps(N, a, m):
          #return np.random.randint(0, 1+1, size=(N,))*2-1
          #return np.random.uniform(-(3**0.5), 3**0.5, size=(N,))
          return (np.random.pareto(a,size=(N,)) + 1)*m;

      s = get_steps(100000, 1.2, 1)
      c, b, _  = np.hist(s, bins = 100)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-34-6ed1263f9a89> in <module>
      5
      6 s = get_steps(100000, 1.2, 1)
----> 7 c, b, _  = np.hist(s, bins = 100)

F:\anaconda\lib\site-packages\numpy\__init__.py in __getattr__(attr)
    217                 return Tester
    218             else:
--> 219                 raise AttributeError("module {!r} has no attribute "
    220                                      "{!r}".format(__name__, attr))
    221

AttributeError: module 'numpy' has no attribute 'hist'
```

Then 100000, 1.2 , 1. So, now, let me do a histogram of s. So, ccounts, bins = np.histogram(s, bins=binsp, density = True), ok.

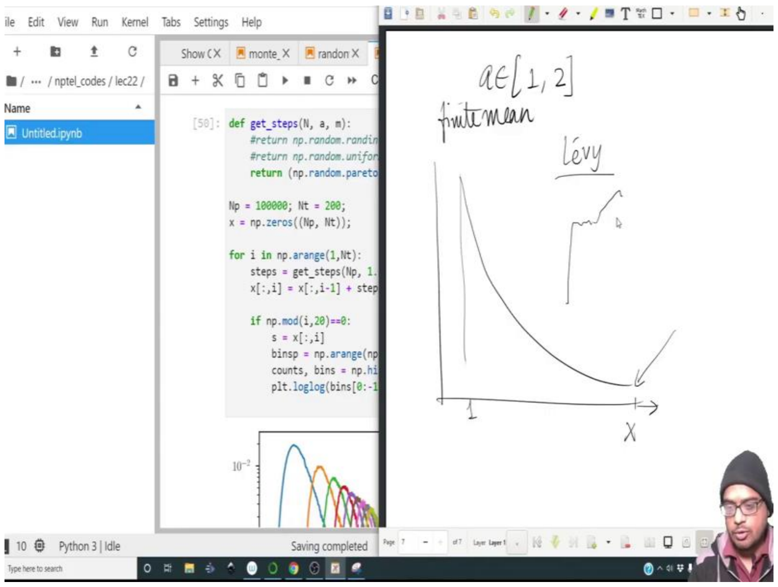
(Refer Slide Time: 45:38)



```
    return (np.random.pareto(a,size=(N,)) + 1)*m;

s = get_steps(100000, 1.2, 1)
c, b, _  = plt.hist(s, bins = np.linspace(1, 10), density = True)
plt.loglog(b[0:-1], c)
```

So, I have taken only a linspace between 1 to 10, just to show you how the density looks like.

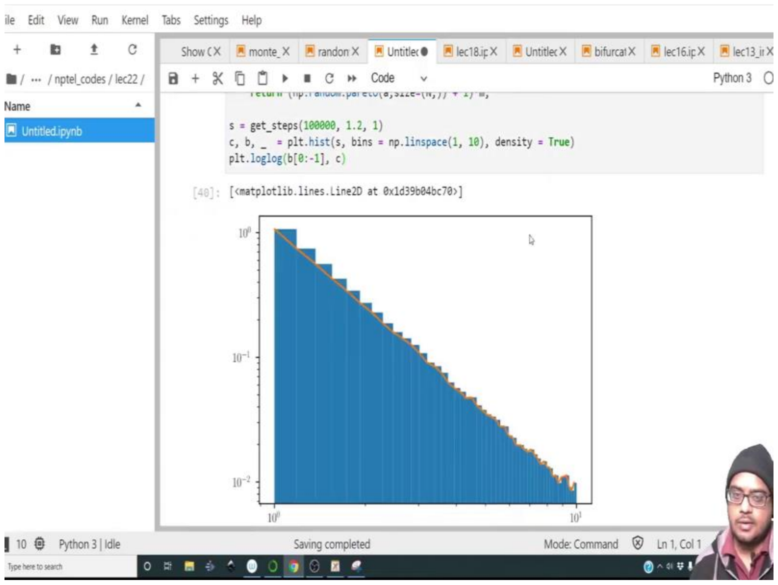
(Refer Slide Time: 45:58)



```
[50]: def get_steps(N, a, m):
          #return np.random.randin
          #return np.random.unifor
          return (np.random.pareto

      Np = 100000; Nt = 200;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np, 1.
          x[:,i] = x[:,i-1] + step

          if np.mod(i,20)==0:
              s = x[:,i]
              binsp = np.arange(np
              counts, bins = np.hi
              plt.loglog(bins[0:-1
```

So, it is it looks something like this. Meaning its more likely for you to sample around values near 1 and the probability for higher values of x goes down, but it is not really 0, ok. It lacks a variance meaning it is not bounded to some number it will always sort of falling down. So, in a log plot; in a log plot it will it will look like a straight curve. Let us see. There you go.

(Refer Slide Time: 46:33)



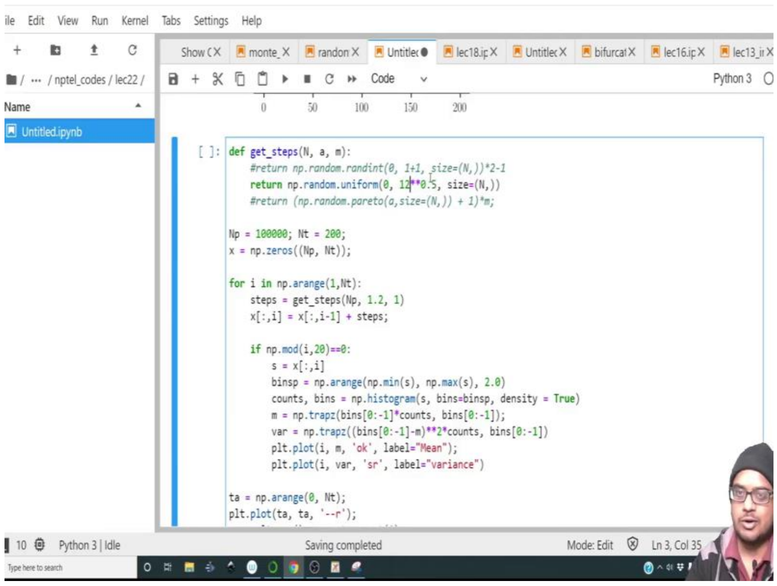
```
    return (np.random.pareto(a,size=(N,)) + 1)*m,

s = get_steps(100000, 1.2, 1)
c, b, _  = plt.hist(s, bins = np.linspace(1, 10), density = True)
plt.loglog(b[0:-1], c)
```

```
[40]: [<matplotlib.lines.Line2D at 0x1d39b04bc70>]
```

So, it is continuously falling now. It is not really falling off very sharply. This is in contrast to a normal distribution or any bounded distribution where you do this we will have a sharp fall and this will be quite apparent at the end of this particular exercise. So, this is how pareto distribution looks like. So, it is more likely to sample this, but it will also sample very large values, but with the lower probability. But nevertheless, the probability is not 0, the probability is still exists, ok.

So, what we will be looking at let me delete this cell we do not require the cell. So, what we will be looking at is we are going to sample from the pareto distribution and we are going to find the mean and variance. So, I have set the pareto distribution like this.

(Refer Slide Time: 47:24)

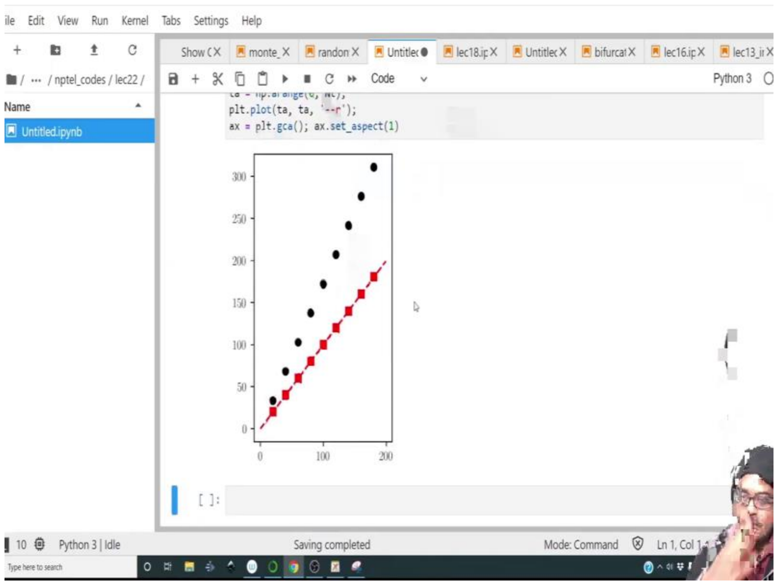


```
[ ]: def get_steps(N, a, m):
         #return np.random.randint(0, 1+1, size=(N,))*2-1
         return np.random.uniform(0, 12**0.5, size=(N,))
         #return (np.random.pareto(a,size=(N,)) + 1)*m;

     Np = 100000; Nt = 200;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np, 1.2, 1)
         x[:,i] = x[:,i-1] + steps;

         if np.mod(i,20)==0:
             s = x[:,i]
             binsp = np.arange(np.min(s), np.max(s), 2.0)
             counts, bins = np.histogram(s, bins=binsp, density = True)
             m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
             var = np.trapz((bins[0:-1]-m)**2*counts, bins[0:-1])
             plt.plot(i, m, 'ok', label="Mean");
             plt.plot(i, var, 'sr', label="variance")

     ta = np.arange(0, Nt);
     plt.plot(ta, ta, '--r');
```

But before that let us just look at how the increasing distribution looks like. Meaning earlier distributions were two-sided, they were symmetric about the origin or half probability to go to the right half probability go to the left, but in the pareto distribution, it is obviously, going from m to infinity.

So, either we can reflect the distribution and multiply the -1. That is one way. But we can alternately define the random walk to only go in one direction in space. So, let me make it from 0 to $\sqrt{12}$ and find out why this $\sqrt{12}$ comes into the picture.

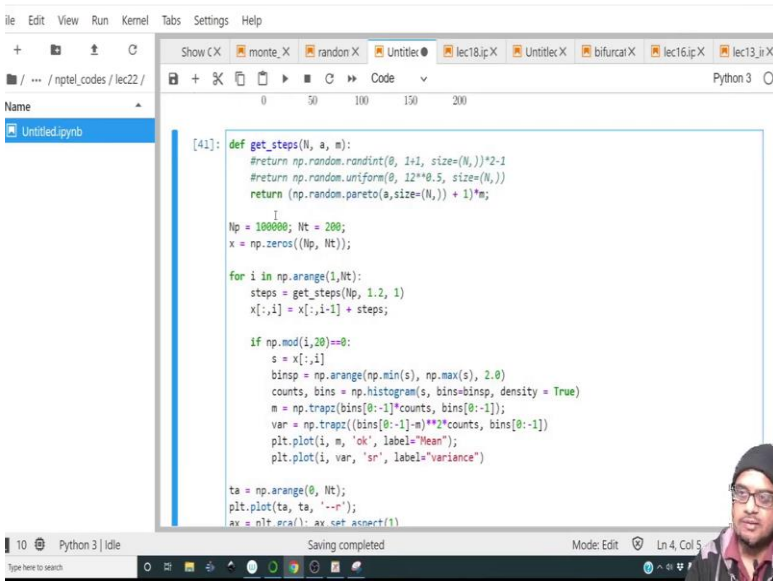
(Refer Slide Time: 48:03)



```
plt.plot(ta, ta, '--r');
ax = plt.gca(); ax.set_aspect(1)
```

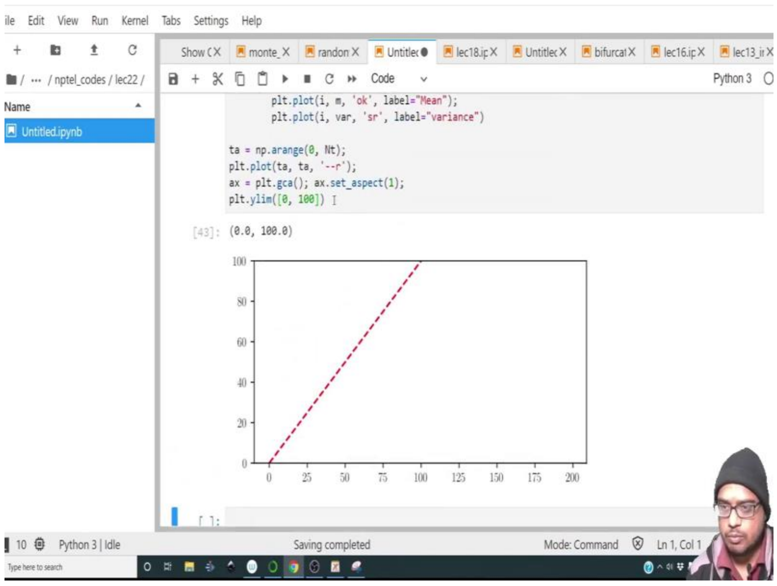
Let us run this and see what goes on. So, the variance increases linearly in time and the mean also increases linear in time. Why does the mean increase linearly in time? Because obviously, we have steps which are only positive and hence the mean also increases linear in time, fine.

(Refer Slide Time: 48:21)



```
[41]: def get_steps(N, a, m):
          #return np.random.randint(0, 1+1, size=(N,))*2-1
          #return np.random.uniform(0, 12**0.5, size=(N,))
          return (np.random.pareto(a,size=(N,)) + 1)*m;

      Np = 100000; Nt = 200;
      x = np.zeros((Np, Nt));

      for i in np.arange(1,Nt):
          steps = get_steps(Np, 1.2, 1)
          x[:,i] = x[:,i-1] + steps;

          if np.mod(i,20)==0:
              s = x[:,i]
              binsp = np.arange(np.min(s), np.max(s), 2.0)
              counts, bins = np.histogram(s, bins=binsp, density = True)
              m = np.trapz(bins[0:-1]*counts, bins[0:-1]);
              var = np.trapz((bins[0:-1]-m)**2*counts, bins[0:-1])
              plt.plot(i, m, 'ok', label="Mean");
              plt.plot(i, var, 'sr', label="variance")

      ta = np.arange(0, Nt);
      plt.plot(ta, ta, '--r');
```

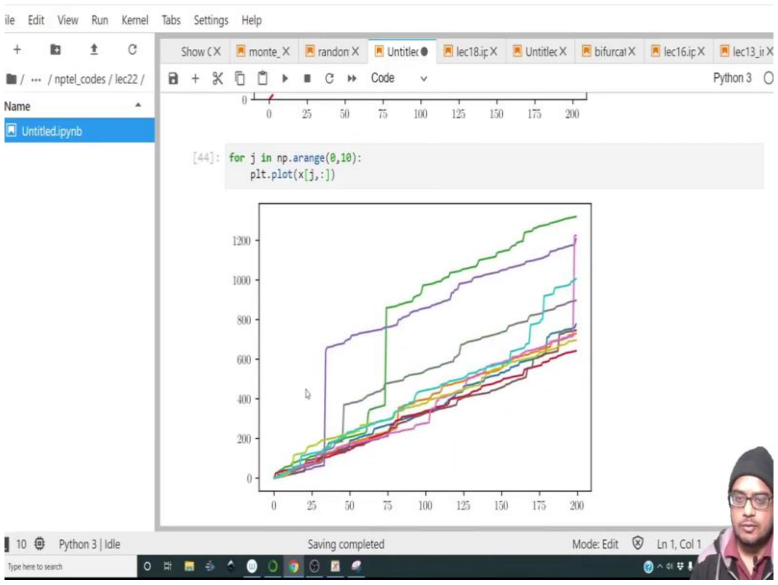
(Refer Slide Time: 48:23)



```
              plt.plot(i, m, 'ok', label="Mean");
              plt.plot(i, var, 'sr', label="variance")

      ta = np.arange(0, Nt);
      plt.plot(ta, ta, '--r');
      ax = plt.gca(); ax.set_aspect(1);
      plt.ylim([0, 100])

[43]: (0.0, 100.0)
```

Now, what happens with the pareto distribution. So, let me get rid of this. Let me do this. Now, something very weird happens. We have very large values of x. Let me set the limit, ok. So, everything blows off. Why does everything blow off? Let us have a look at a few trajectories. So, did we delete the trajectory record? Ok. We deleted the trajectory record, but no never mind we can again plot that.

(Refer Slide Time: 48:55)



```
[44]: for j in np.arange(0,10):
          plt.plot(x[j,:])
```

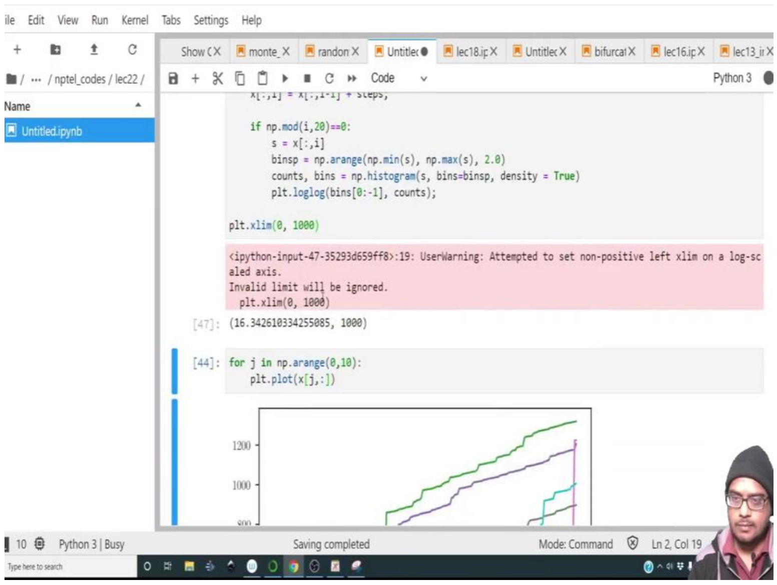
So, we have x all values. So, for j in np.arange(0,10), we are going to plt.plot(x[j,:]) and this will give me the trajectories of the different particles. Let us see. So, what do we see over here? We see and, we have small increments small increment, but boom suddenly we have a large jump.

It is because while most of the steps are quite small, because of the nature of the choice of step from the pareto distribution there will be a few jumps now and then, which will make a large increase in the number and you can imagine that because of the presence of this far off jumps, ok. These jumps happen now and then the jumps are not frequent because we are sampling it from a distribution in which the large jump is not probable.

But it is probable, but it is not likely to come very frequently and that is why we will have jumps like this and because of this once you try to find out the mean it will start blowing up because there will be a many trajectories which have such a large jump because they might have happened sometime earlier in the in the simulation or they may happen sometime in the future.

But such jumps do exist and hence it is not easy to define. But nevertheless what we can do is find out the pdf of the locations that may give us some insight. So, let me grab this particular program.

(Refer Slide Time: 50:36)



```
    x[:,i] = x[:,i-1] + steps;

    if np.mod(i,20)==0:
        s = x[:,i]
        binsp = np.arange(np.min(s), np.max(s), 2.0)
        counts, bins = np.histogram(s, bins=binsp, density = True)
        plt.loglog(bins[0:-1], counts);

plt.xlim(0, 1000)
```

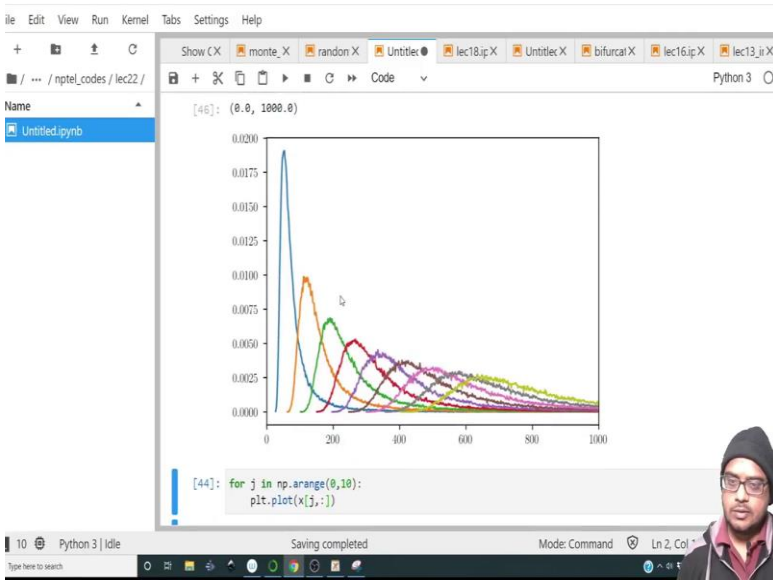
```
<ipython-input-47-35293d659ff8>:19: UserWarning: Attempted to set non-positive left xlim on a log-sc
aled axis.
Invalid limit will be ignored.
  plt.xlim(0, 1000)
```

```
[47]: (16.342610334255085, 1000)
```

```
[44]: for j in np.arange(0,10):
          plt.plot(x[j,:])
```

And let me see how let us forget about the mean and variance because for such cases it does not really make sense. Let me plot the pdf. Let me get rid of this as well. Let me run this.

(Refer Slide Time: 51:18)

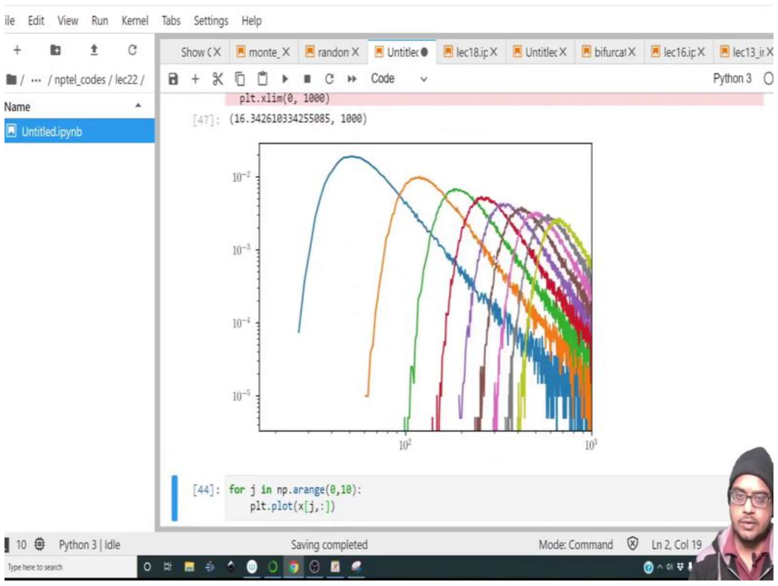


```
[46]: (0.0, 1000.0)
```

```
[44]: for j in np.arange(0,10):
          plt.plot(x[j,:])
```

Let me set, ok. This is how the distributions look like. They are increasing in, so the mean is going shifting towards the right that is fine, but at the same time we have a flat tail. So, the tail is not really coming down like a Gaussian it is its sort of extending all the way to infinity well and that is called as a fat tail. And it should be clearer if we do a log log plot. So, if we double log everything, we should be able to see the existence of the power law, ok.

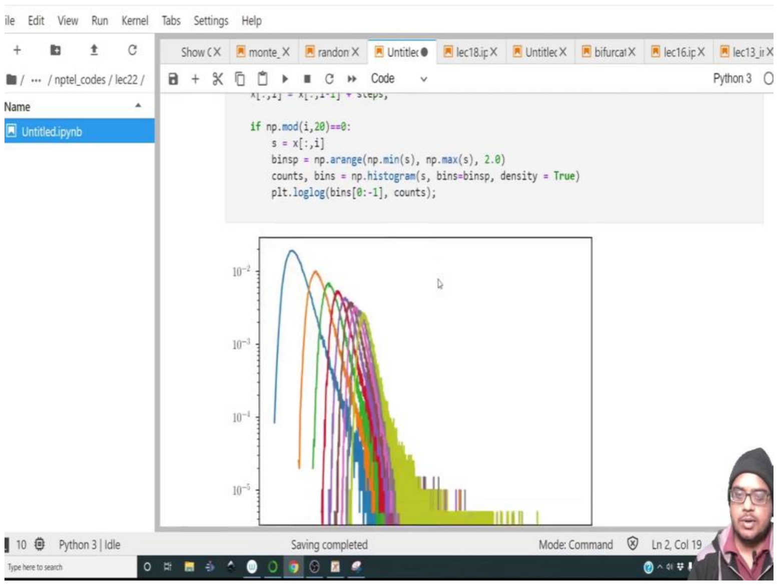
(Refer Slide Time: 51:53)



```
plt.xlim(0, 1000)
[47]: (16.342610334255085, 1000)
```

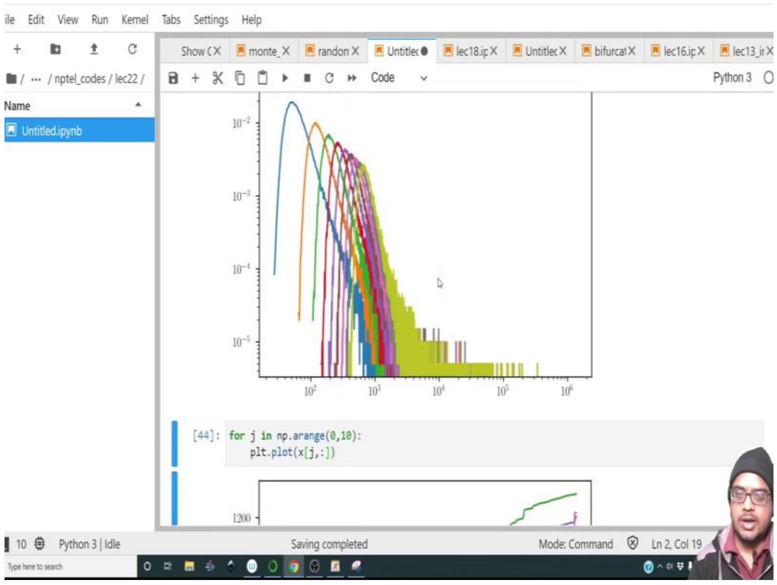
```
[44]: for j in np.arange(0,10):
          plt.plot(x[j,:])
```

(Refer Slide Time: 51:56)



```
    x[:,i] = x[:,i-1] + steps;

    if np.mod(i,20)==0:
        s = x[:,i]
        binsp = np.arange(np.min(s), np.max(s), 2.0)
        counts, bins = np.histogram(s, bins=binsp, density = True)
        plt.loglog(bins[0:-1], counts);
```

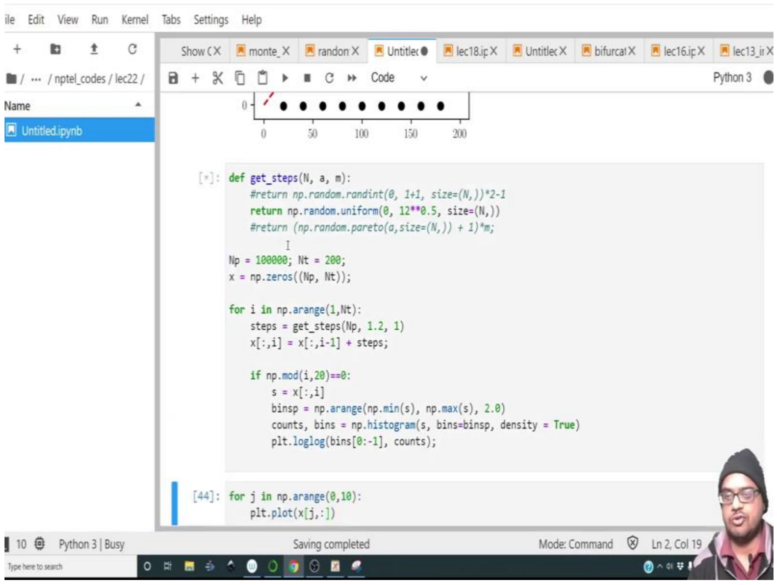
(Refer Slide Time: 51:59)



```
[44]: for j in np.arange(0,10):
          plt.plot(x[j,:])
```

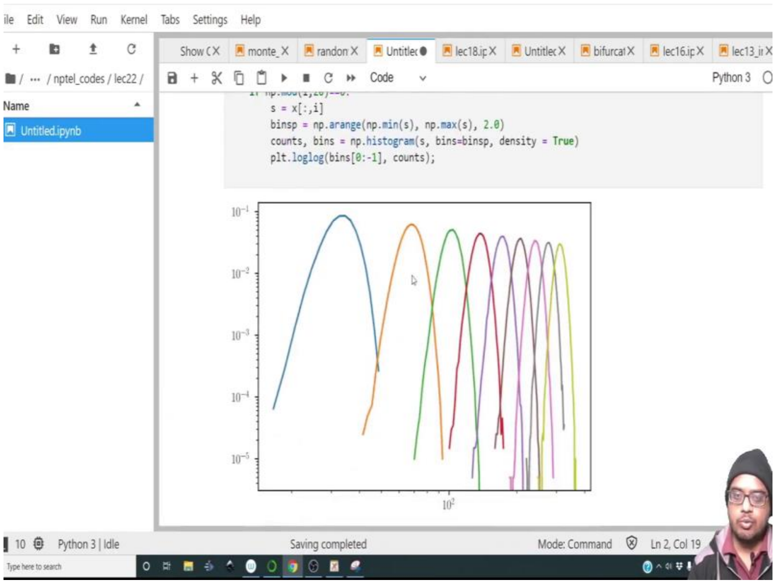
So, that power law does exist, in fact, let me get rid of this limit to highlight that power law. So, this is that power law, ok. So, this long tail that exists, it never goes to 0. But now who said that, this is that fat tail I mean I am not shown you how the Gaussian distribution looks like in the log log. So, let us let us do that the normal distribution how it looks in log log.

(Refer Slide Time: 52:20)



```
[*]: def get_steps(N, a, m):
         #return np.random.randint(0, 1+1, size=(N,))*2-1
         return np.random.uniform(0, 12**0.5, size=(N,))
         #return (np.random.pareto(a,size=(N,)) + 1)*m;

     Np = 100000; Nt = 200;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np, 1.2, 1)
         x[:,i] = x[:,i-1] + steps;

         if np.mod(i,20)==0:
             s = x[:,i]
             binsp = np.arange(np.min(s), np.max(s), 2.0)
             counts, bins = np.histogram(s, bins=binsp, density = True)
             plt.loglog(bins[0:-1], counts);

[44]: for j in np.arange(0,10):
          plt.plot(x[j,:])
```

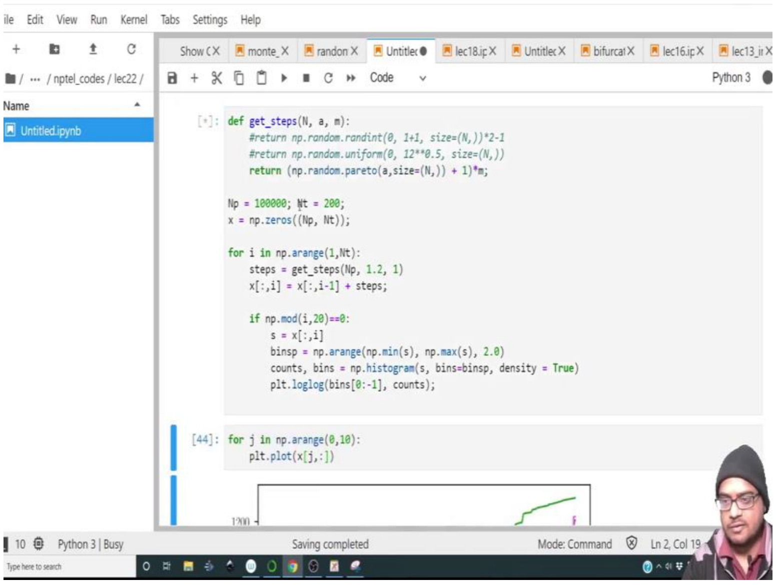
(Refer Slide Time: 52:33)



```
        s = x[:,i]
        binsp = np.arange(np.min(s), np.max(s), 2.0)
        counts, bins = np.histogram(s, bins=binsp, density = True)
        plt.loglog(bins[0:-1], counts);
```

So, the beauty of user using a code is you can just change the return value and you can get this distribution you want and boom, for a Gaussian always comes down quickly. It does not have that power law decay, alright.

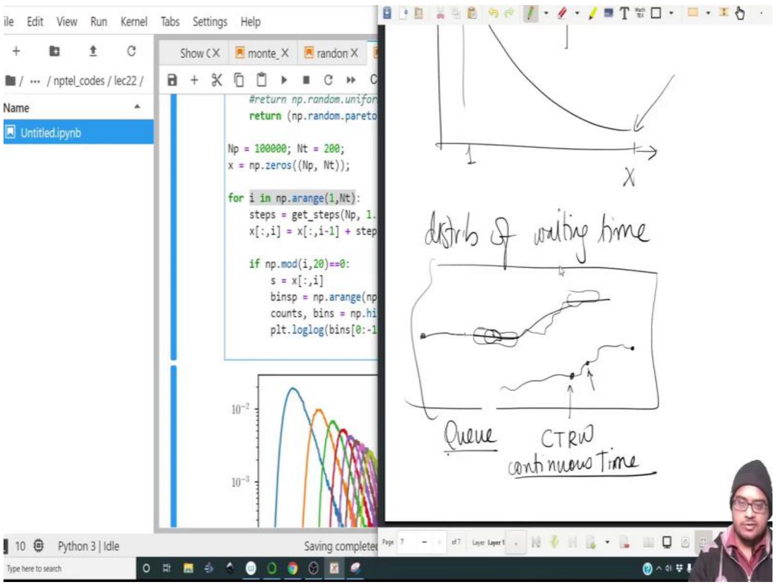
(Refer Slide Time: 52:43)



```
[*]: def get_steps(N, a, m):
         #return np.random.randint(0, 1+1, size=(N,))*2-1
         #return np.random.uniform(0, 12**0.5, size=(N,))
         return (np.random.pareto(a,size=(N,)) + 1)*m;

     Np = 100000; Nt = 200;
     x = np.zeros((Np, Nt));

     for i in np.arange(1,Nt):
         steps = get_steps(Np, 1.2, 1)
         x[:,i] = x[:,i-1] + steps;

         if np.mod(i,20)==0:
             s = x[:,i]
             binsp = np.arange(np.min(s), np.max(s), 2.0)
             counts, bins = np.histogram(s, bins=binsp, density = True)
             plt.loglog(bins[0:-1], counts);

[44]: for j in np.arange(0,10):
          plt.plot(x[j,:])
```

So, yeah, these kind of walks are called as levy walks it. It is named after French mathematician, Levy, whose work on all these kinds of fat tales gave rise to distributions called as Levy stable distributions and such kind of catastrophic jumps are quite common in groundwater transport.

(Refer Slide Time: 53:12)



```
    #return np.random.unifor
    return (np.random.pareto

Np = 100000; Nt = 200;
x = np.zeros((Np, Nt));

for i in np.arange(1,Nt):
    steps = get_steps(Np, 1.
    x[:,i] = x[:,i-1] + step

    if np.mod(i,20)==0:
        s = x[:,i]
        binsp = np.arange(np
        counts, bins = np.hi
        plt.loglog(bins[0:-1
```

So, for example, if you have a porous medium, in the porous medium you have regions of high hydraulic conductivity low hydraulic conductivity. So, if a particle is going like this if the conductivity is high over here its position will suddenly increase, but if you have some it keeps on going if it encounters a region of high conductivity again it will suddenly increase, ok.

So, such kind of paradigms are quite useful to model various kinds of physical processes, it is not just for fun that people do this and all these fall within the domain of stochastic modeling, it has also ramifications in finance and choices a lot of economic planning and things like that.

But I have shown you some of the tools with which you can get started already. Well, what we have done is we have only discussed with the step size distribution whereas, we have kept the time distribution to be fixed each time we are doing a loop over delta t alternately you could have a distribution of times as well. So, we could have a distribution of waiting times.

It is meaning that if a particle is going like this it will wait over here for a long time before making another step. It will wait over here for a long time before going to somewhere else. So, such kinds of waiting time distributions are quite useful for queuing theory, que whatever. It is quite a called doozy spelling.

So, queuing theory its quite important because in order to understand how your industrial process should be or how a queue should function, you must understand what waiting times distribution will you will have if you have so many processes downstream or upstream. And such kinds of stochastic processes in which there is a distribution of waiting times they are called as CTRWs that is continuous time random walks, ok.

And continuous time random walks also very important from the point of view of modeling hydrological flows, so how a solute will wait in a region of low conductivity or it will not wait in a region of high conductivity. So, the distribution of velocities and over the entire domain gives rise to such kinds of wait times which eventually allow us to look deeply into how the dispersion of a solvent or a solute in a in a given porous medium will happen.

So, with this I end this session. I hope you enjoyed this. I hope you will take away many wonderful things from this particular set of examples that we did. You can try doing this, this kind of random work for a bunch of different distributions of step size, time, or even do it for a host of step distribution functions which have finite variances which do not have finite variances and then just see what kind of things you get.

For those of you who are mathematically inclined you can have a look at a variety of books which deal with first passage times and so on, central limit theorems, and you can peer deeply into the mathematics of it as well.

With this I am going to end this particular lecture and see you next time. Bye.