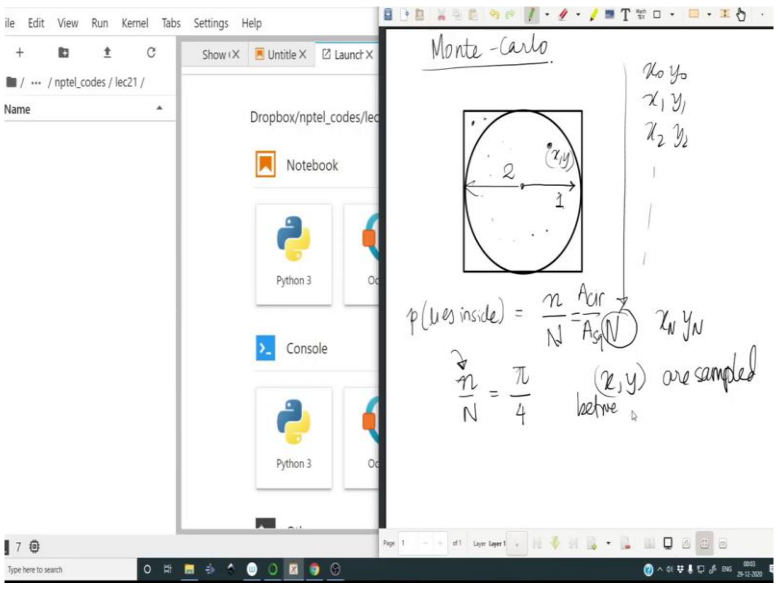
Tools in Scientific Computing

Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 21
Monte-Carlo simulation: Darts and Buffon's needle

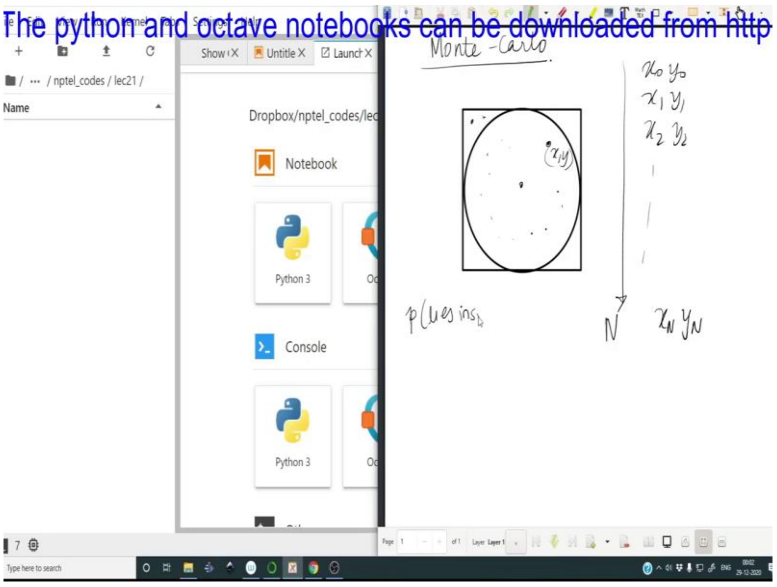
(Refer Slide Time: 00:36)


Monte-Carlo
$x_0\ y_0$
$x_1\ y_1$
$x_2\ y_2$
(x,y)
2
1
$p(\text{lies inside}) = \frac{n}{N} = \frac{A_{cir}}{A_{sq}}$
$x_N\ y_N$
$\frac{n}{N} = \frac{\pi}{4}$
(x,y) are sampled before

Hi everyone. We continue our endeavor and it will proceed with the Monte-Carlo simulations for finding out the value of π. So, the very primitive or the very simple way of doing it is as follows. So, you have a square and a circle which is inscribed in it right. Well, I can draw them properly. So, let me do that alright, you get the point. So, you have something like this, yeah.

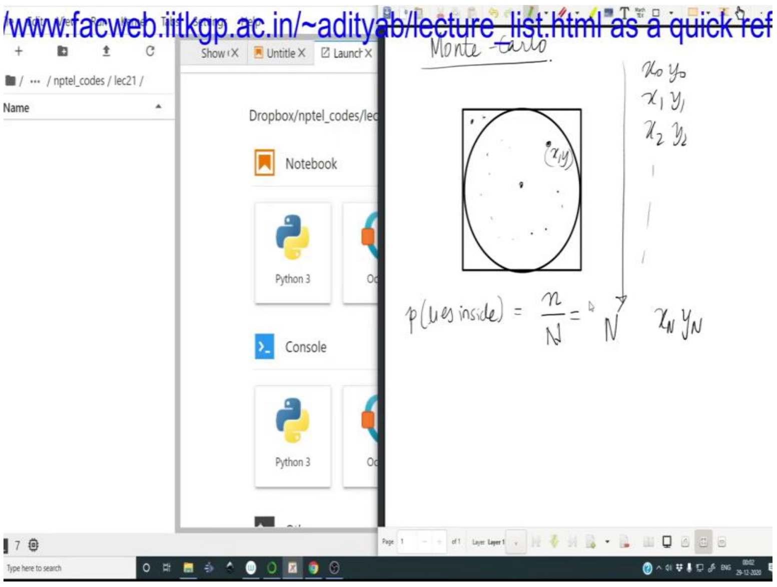
So, then the point is you sort of choose a point randomly in you choose a pair of points (x, y) in the square ok. So, suppose, you choose this point, you choose this point, you randomly sample a whole bunch of points. So, you have $x_0, y_0, x_1, y_1, x_2, y_2$ and then, you have may be n such samples x_n, y_n alright.

(Refer Slide Time: 02:05)


The python and octave notebooks can be downloaded from http

So, then for each point, what we would want to know is whether that point lies inside the circle or not.

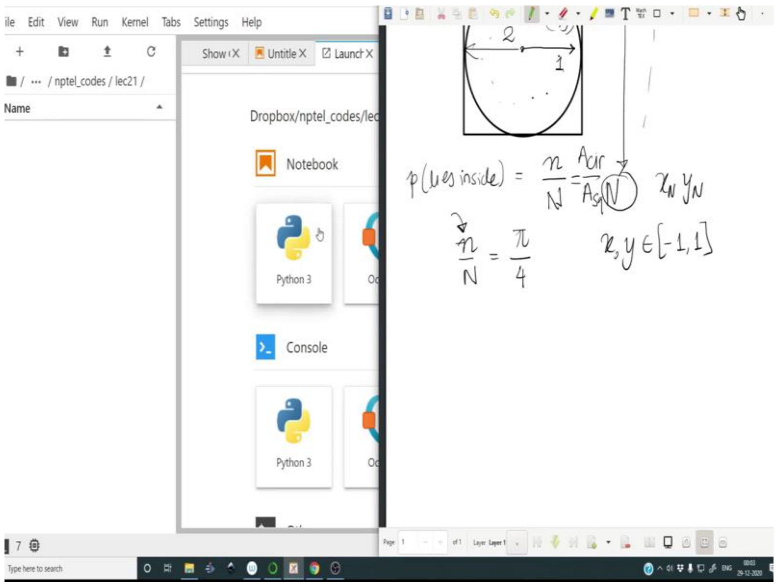
(Refer Slide Time: 02:24)


/www.facweb.iitkgp.ac.in/~adityab/lecture_list.html as a quick ref

So, the probability that a point lies inside is equal to the number of points inside the circle. So, if I call that a small n divided by the total number of points and this probability will be equal to the ratio of the area of circle to the area of this square.

So, if this radius is equal to 1, then n/N becomes equal to π upon; so, if this radius is 1, this diameter becomes equal to 2. So, the square area is 4 alright. So, if we choose a large number of points alright, so the number of samples are large and the choice of x and y are sampled from a uniform distribution ok.

(Refer Slide Time: 03:35)

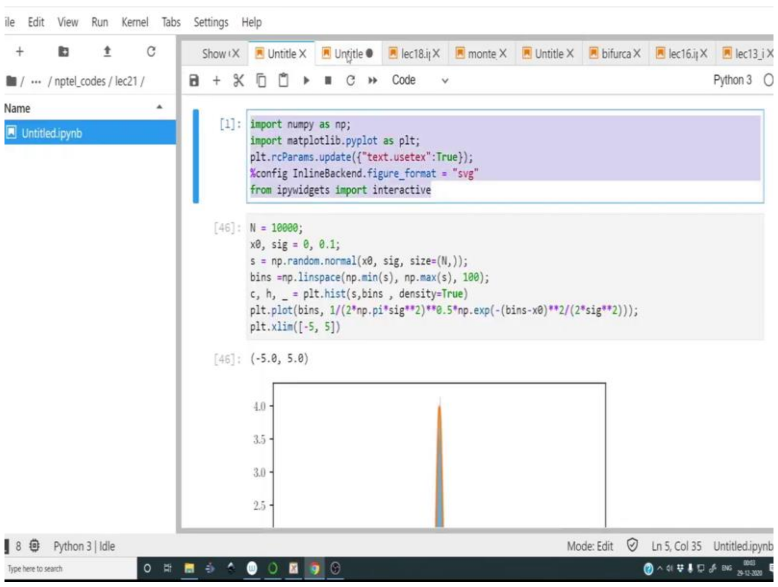


$$p(\text{lies inside}) = \frac{n}{N} = \frac{A_{cir}}{A_{sq}} \quad N \quad x_N\, y_N$$

$$\frac{n}{N} = \frac{\pi}{4} \qquad x, y \in [-1, 1]$$

So, x and y are sampled between or rather we can write it like this, (x, y) and belongs to -1 to 1 ok. So, let us go to our computer and do this.

(Refer Slide Time: 03:51)

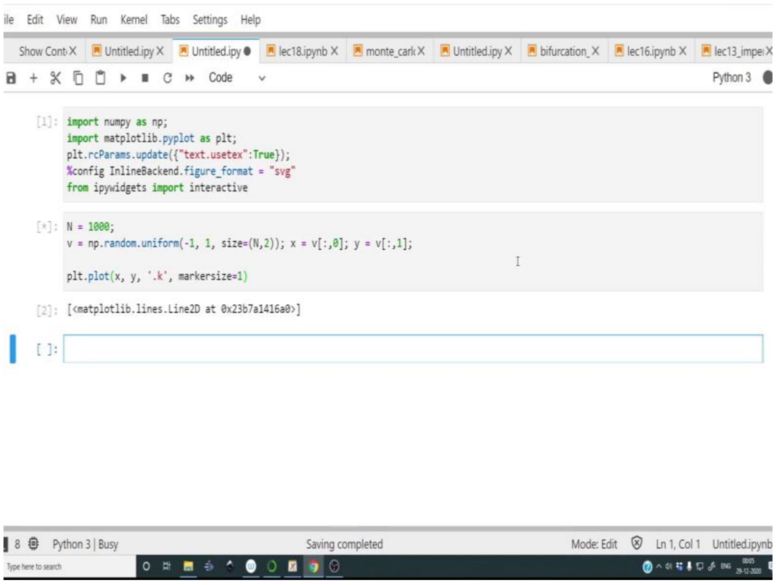


```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive
```

```
[46]: N = 10000;
      x0, sig = 0, 0.1;
      s = np.random.normal(x0, sig, size=(N,));
      bins =np.linspace(np.min(s), np.max(s), 100);
      c, h, _ = plt.hist(s,bins , density=True)
      plt.plot(bins, 1/(2*np.pi*sig**2)**0.5*np.exp(-(bins-x0)**2/(2*sig**2)));
      plt.xlim([-5, 5])
```

```
[46]: (-5.0, 5.0)
```

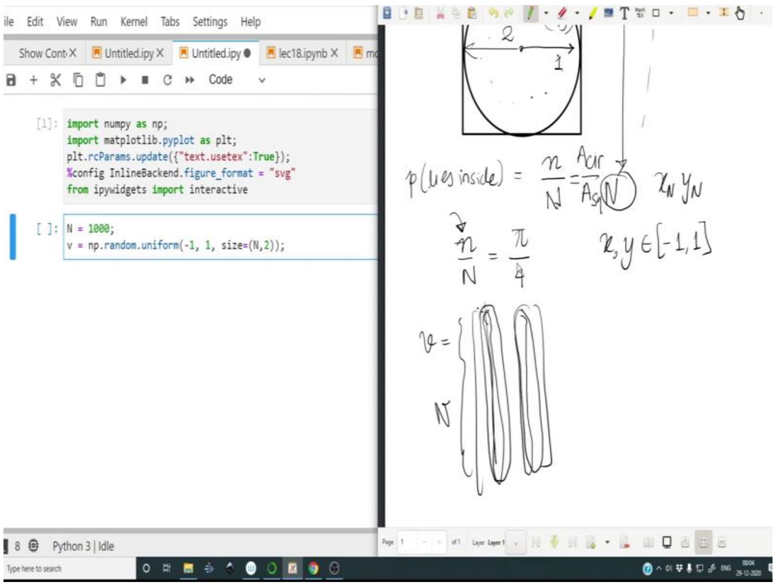
(Refer Slide Time: 03:56)



```
[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

[*]: N = 1000;
     v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

     plt.plot(x, y, '.k', markersize=1)

[2]: [<matplotlib.lines.Line2D at 0x23b7a1416a0>]
```

So, let me copy this snippet. We will require it as usual. So, let me run it, then I will have a large number of points sampled between -1 to 1 ok. So, I will do; I will do the following. I will say v=np.random.uniform and I will sample it from -1 to 1 and I will declare the size equal to N rows and 2 columns, where I will define what N is alright.

(Refer Slide Time: 04:53)



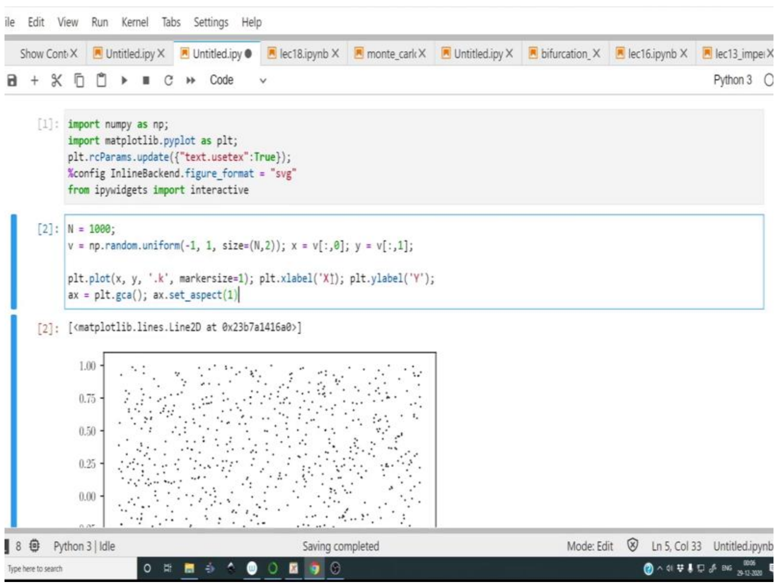
```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive

[ ]: N = 1000;
     v = np.random.uniform(-1, 1, size=(N,2));
```

2
1
$p(\text{lies inside}) = \frac{n}{N} = \frac{A_{cir}}{A_{sq}}$
$x_N\ y_N$
$\frac{n}{N} = \frac{\pi}{4}$
$x, y \in [-1, 1]$
$v =$
N

So, I will define N = 1000 ok. No problem. Once I have v, so what is v? So, it has N rows and 2 columns ok. So, these will be the x values and these will be the y value. So, let me splice the array v to assign them to x and y. So, x equal to v all rows of the first column and y equal to v all rows of the second column alright.

So, now, that we have all these x and y's; what we will do is we will try to plot these pairs of points ok. So, plt.plot (x, y) and we will put them as markers and we will reduce the markersize. We will make it 1 alright.

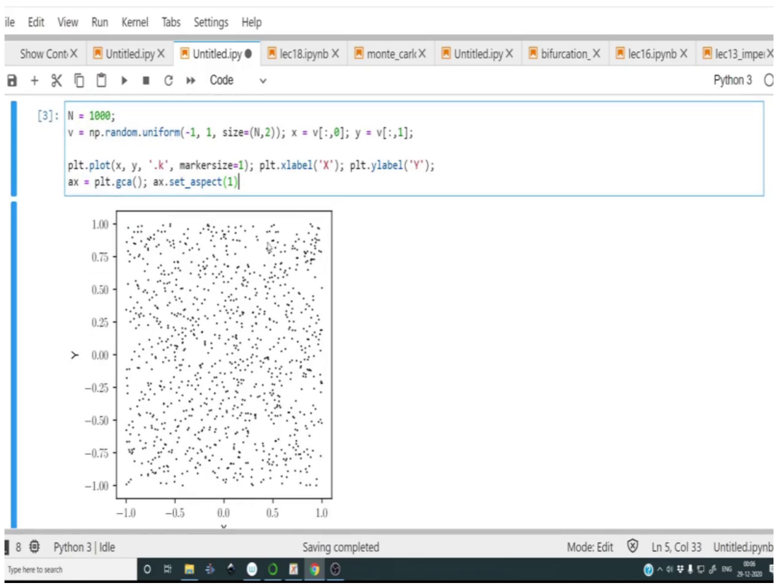
(Refer Slide Time: 05:58)



```
[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

[2]: N = 1000;
     v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

     plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
     ax = plt.gca(); ax.set_aspect(1)

[2]: [<matplotlib.lines.Line2D at 0x23b7a1416a0>]
```

Let us see what we have. We will set the aspect ratio to 1 because we would like the aspect ratio of the plot to be correct alright ok.

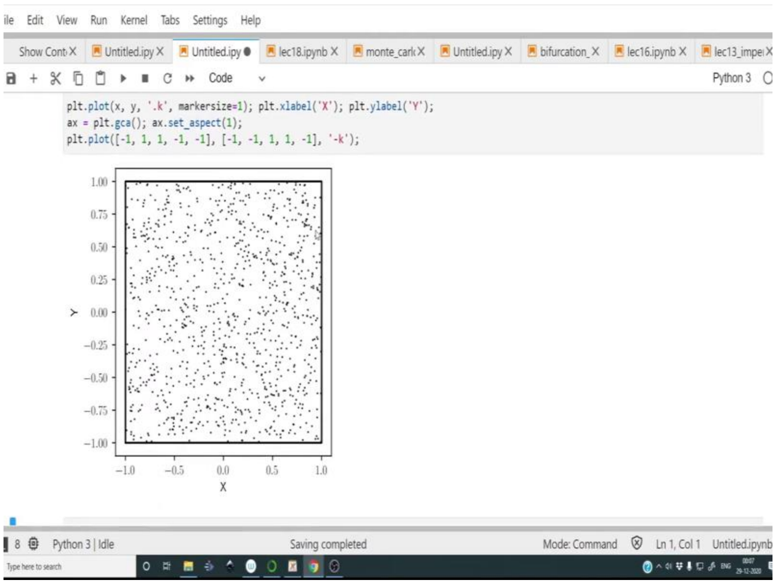
(Refer Slide Time: 06:31)



```
[3]: N = 1000;
     v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

     plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
     ax = plt.gca(); ax.set_aspect(1)
```

So, this is what we have. So, these are all the points that we have sampled randomly, but uniformly between -1 and 1 alright.

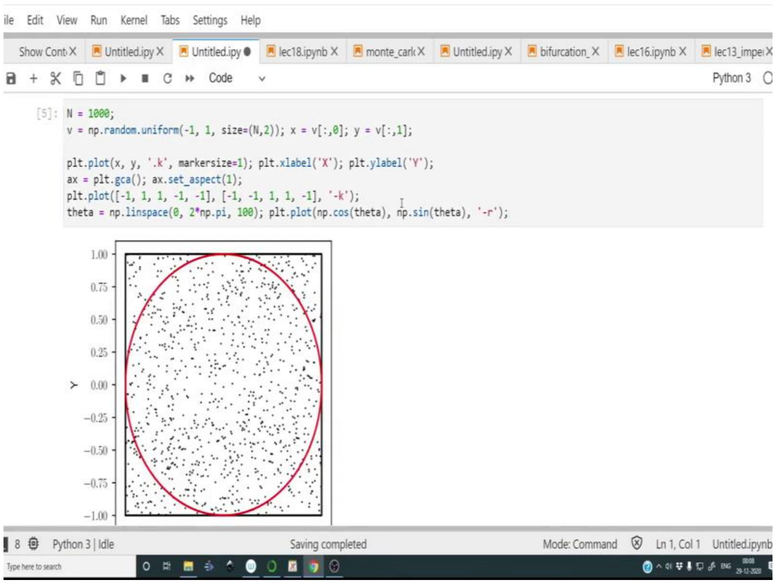
(Refer Slide Time: 06:55)



```
plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
ax = plt.gca(); ax.set_aspect(1);
plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
```

So, let me draw the square; let me draw the square. So, the x coordinates will be starting from -1, 1, 1, -1, -1. The y coordinates will be -1, -1, 1, 1, -1 and we will use a black line. So, that is the square. We will also plot the circle; so, in order to plot the circle with the radius 1 ok.

(Refer Slide Time: 07:48)

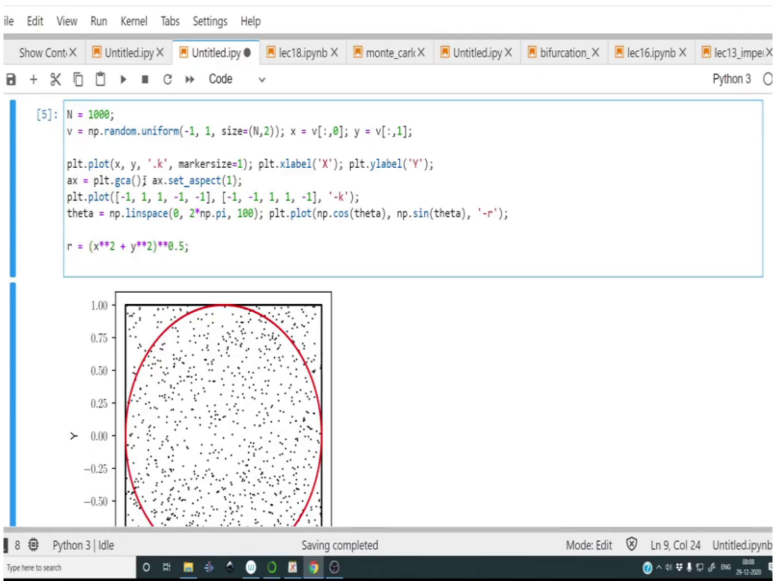


```
[5]: N = 1000;
     v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

     plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
     ax = plt.gca(); ax.set_aspect(1);
     plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
     theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');
```

So, we can do the following. We can define theta=np.linspace (0, 2*np.pi). Let me take 100 values. Then, we will do plt.plot. So, the radius is 1. So, the x coordinates will be r cos(theta) that is cos(theta). So, np.cos(theta) np.sin(theta) and we will make it as a red circle alright.

So, now, the question is whether or we will be want to find out how many points that we have randomly sampled or rather uniformly sampled between -1 and 1 for both x and y? How many of those points lie inside the circle out of the total number of points sampled ok?

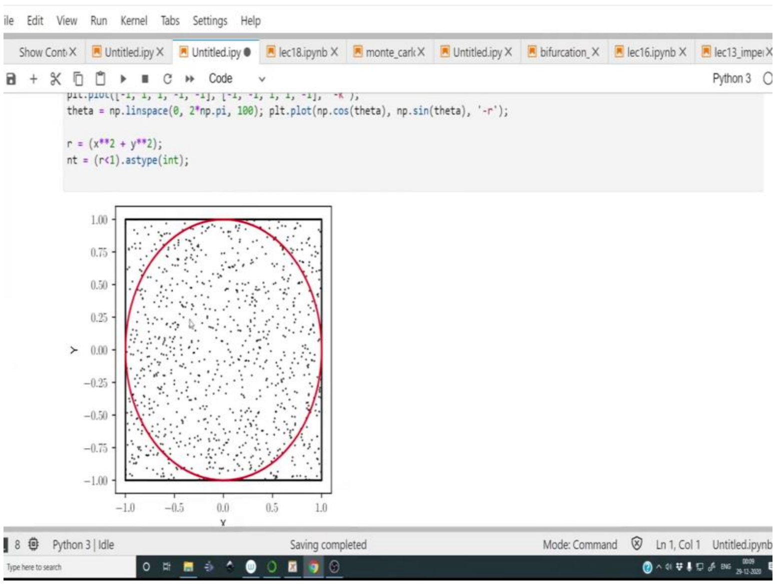
(Refer Slide Time: 08:51)



```
[5]: N = 1000;
     v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

     plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
     ax = plt.gca(); ax.set_aspect(1);
     plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
     theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

     r = (x**2 + y**2)**0.5;
```

So, let us do that. So, once we have plotted this, we can create an array r. So, r will be equal to simply $x^2 + y^2$. Now, ideally r will be $x^2 + y^2$ and this is to 0.5, but because we are going to check it against 1, the radius going to be 1. We can drop this to save on one set of calculation. We can drop the square root. So, now r is this and we are concerned with all those sort of index of r which are less than 1 ok.

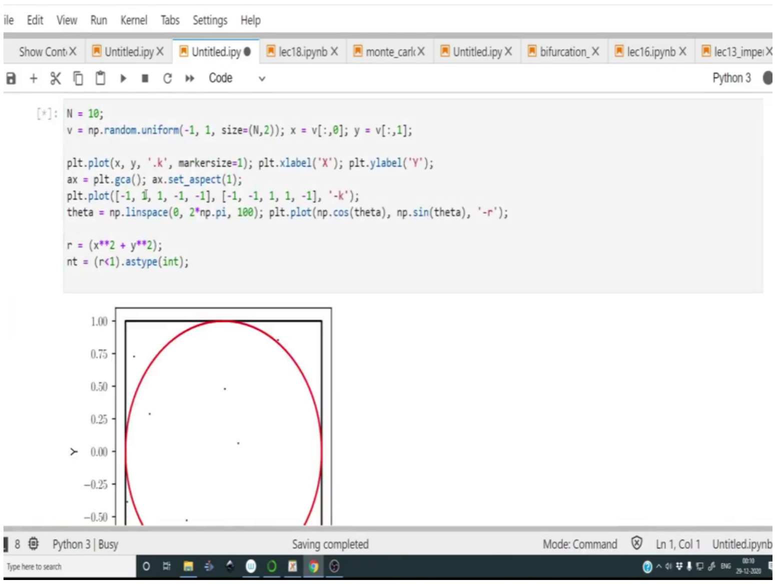
(Refer Slide Time: 09:41)



```
plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

r = (x**2 + y**2);
nt = (r<1).astype(int);
```

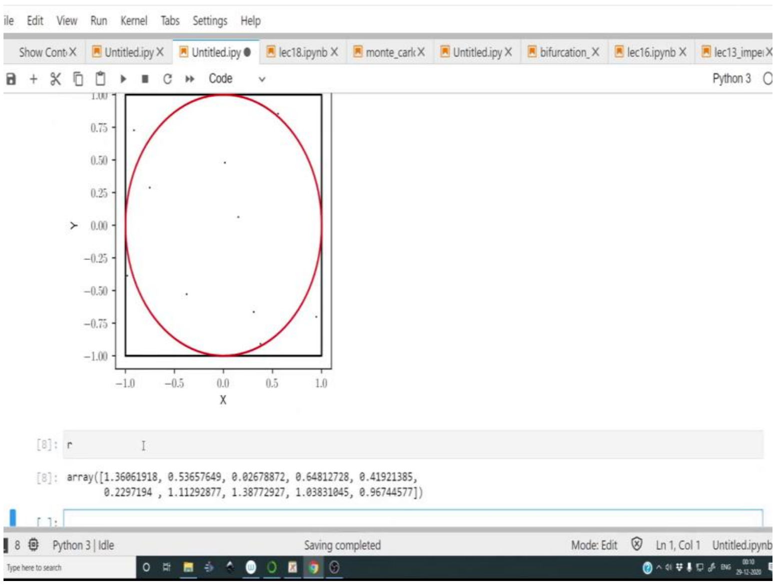
So, n true will be r less than 1 and we will type est it as int; meaning, so the meaning of this line is well, I will print it and show you. I think it is much easier, if I print it and show you.

(Refer Slide Time: 10:12)



```
[*]: N = 10;
     v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

     plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
     ax = plt.gca(); ax.set_aspect(1);
     plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
     theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

     r = (x**2 + y**2);
     nt = (r<1).astype(int);
```

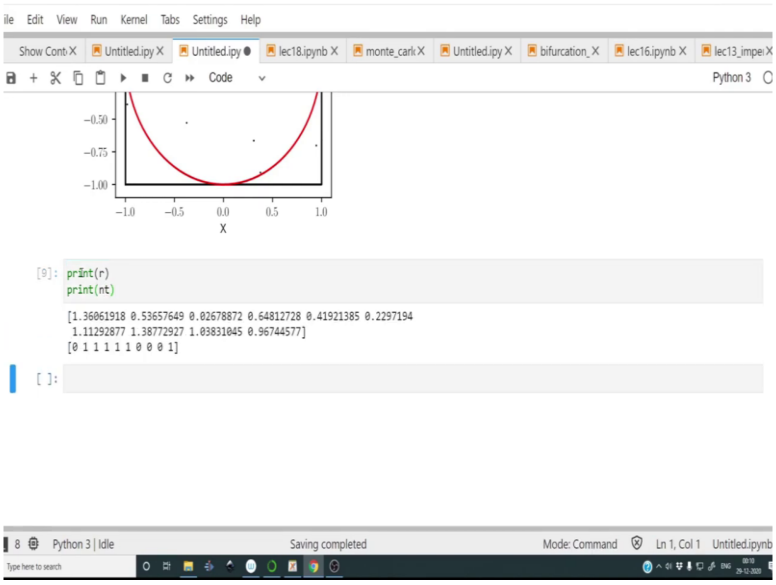
(Refer Slide Time: 10:14)



```
[8]: r

[8]: array([1.36061918, 0.53657649, 0.02678872, 0.64812728, 0.41921385,
            0.2297194 , 1.11292877, 1.38772927, 1.03831045, 0.96744577])
```

So, let me run this. Let me show you what r is. In fact, let me reduce the number of samples to 10. So, then showing it is much easy.

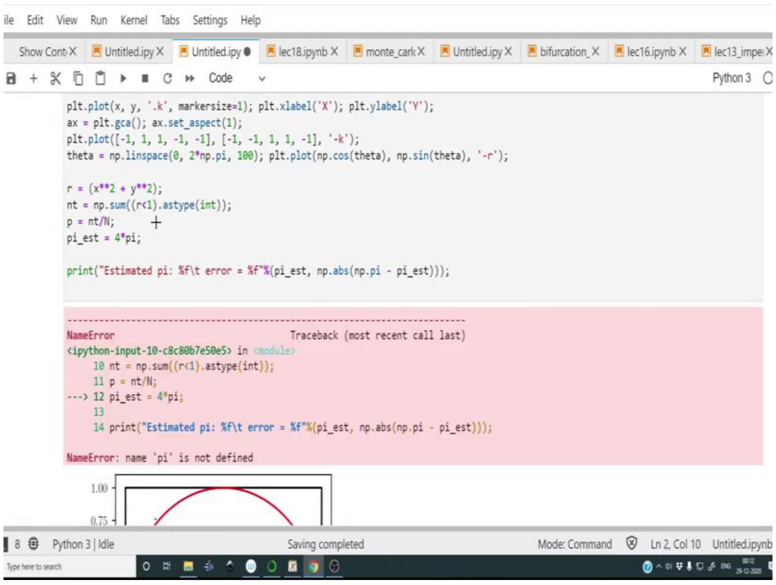
(Refer Slide Time: 10:25)



```
[9]: print(r)
     print(nt)

     [1.36061918 0.53657649 0.02678872 0.64812728 0.41921385 0.2297194
      1.11292877 1.38772927 1.03831045 0.96744577]
     [0 1 1 1 1 1 0 0 0 1]
```

So, r is this. Now, let me show you what nt is. So, let me print r print nt. So, now look. It is greater than 1; so, it is 0. It is less than 1; so, it is 1 less than 1, 1 and so on. So, I now have an array which gives me the Boolean whether something is greater than 1 or less than 1; that is something is inside the circle or outside the circle. Now, how many occurrences of that particular conditional do I have? This particular conditional, how many occurrences do I have? I will simply sum all the indexes of nt.

(Refer Slide Time: 11:08)



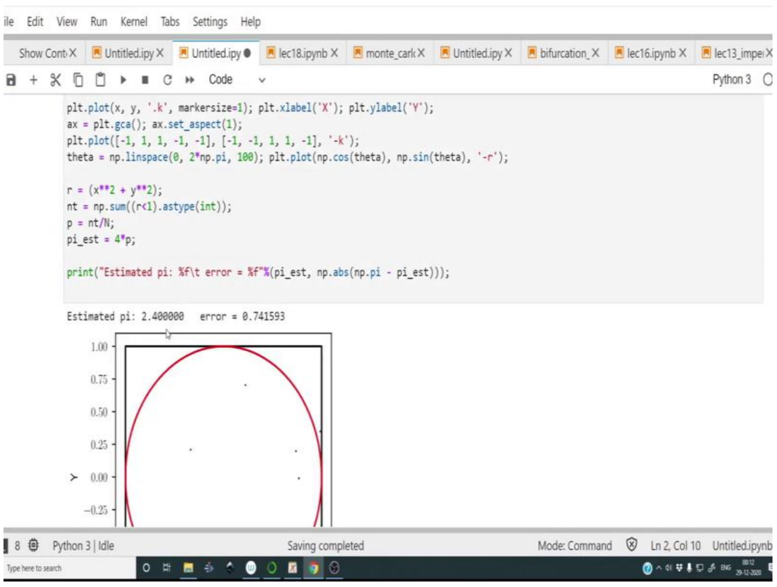
```
plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
ax = plt.gca(); ax.set_aspect(1);
plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

r = (x**2 + y**2);
nt = np.sum((r<1).astype(int));
p = nt/N;
pi_est = 4*pi;

print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-c8c80b7e50e5> in <module>
     10 nt = np.sum((r<1).astype(int));
     11 p = nt/N;
---> 12 pi_est = 4*pi;
     13
     14 print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));

NameError: name 'pi' is not defined
```

So, this will be np.sum of this. So, that will give me all the instances, where the radius is less than 1 and so, now, the probability this particular probability is equal to n/N. So, let me define p is nt upon number of samples. So, now, I have an Estimate of π. So, π estimate is equal to 4 times this probability and then, I will simply print pi_est or in fact, I can do the following.

Pi estimate %f\t error = %f and I will print over here pi_est. In fact, let me make this Estimated π and I will do this ,np.abs(np.pi minus pi est) ok. So, let me run this. So, π is this has to be p ok.

(Refer Slide Time: 12:36)

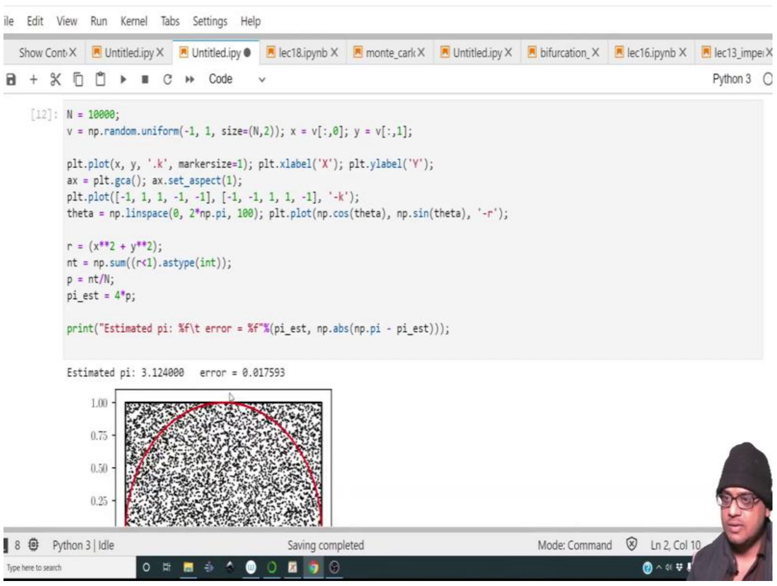


```
plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
ax = plt.gca(); ax.set_aspect(1);
plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

r = (x**2 + y**2);
nt = np.sum((r<1).astype(int));
p = nt/N;
pi_est = 4*p;

print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));
```

```
Estimated pi: 2.400000   error = 0.741593
```

So, obviously when the number of samples are quite low, there will be a very large error.

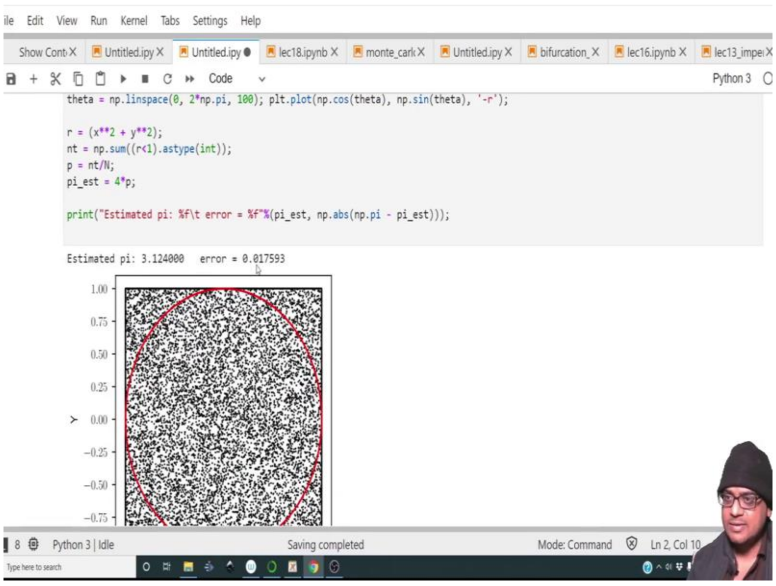
(Refer Slide Time: 12:46)



```
[12]: N = 10000;
      v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

      plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
      ax = plt.gca(); ax.set_aspect(1);
      plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
      theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

      r = (x**2 + y**2);
      nt = np.sum((r<1).astype(int));
      p = nt/N;
      pi_est = 4*p;

      print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));
```

```
Estimated pi: 3.124000   error = 0.017593
```

(Refer Slide Time: 12:49)

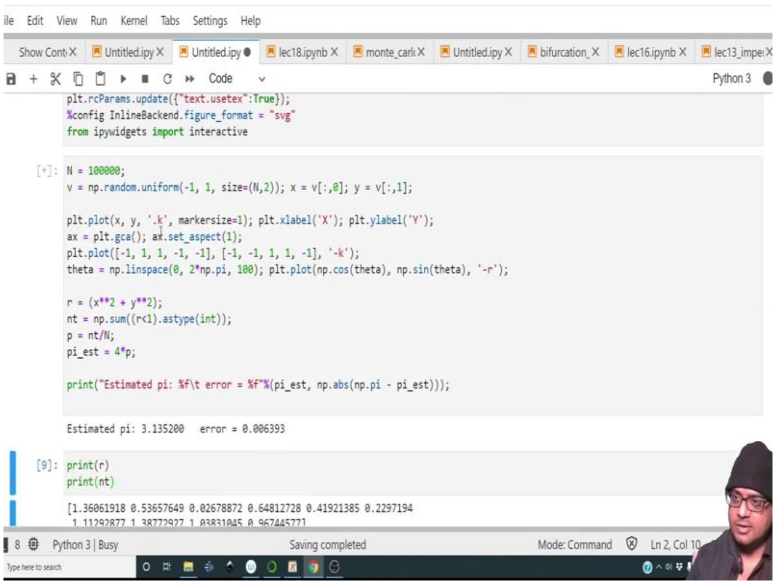


```
theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

r = (x**2 + y**2);
nt = np.sum((r<1).astype(int));
p = nt/N;
pi_est = 4*p;

print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));
```

```
Estimated pi: 3.124000   error = 0.017593
```

Now, what we need to do is increase the number of samples; we see that the error is decreasing.

(Refer Slide Time: 12:56)



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

```
[*]: N = 100000;
     v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

     plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
     ax = plt.gca(); ax.set_aspect(1);
     plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
     theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

     r = (x**2 + y**2);
     nt = np.sum((r<1).astype(int));
     p = nt/N;
     pi_est = 4*p;

     print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));
```

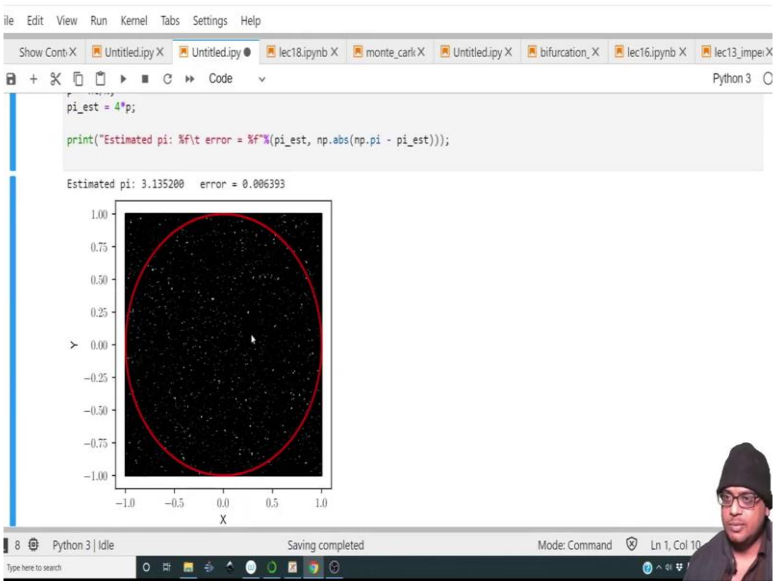
```
Estimated pi: 3.135200   error = 0.006393
```

```
[9]: print(r)
     print(nt)
```

```
[1.36061918 0.53657649 0.02678872 0.64812728 0.41921385 0.2297194
 1.11292877 1.38772927 1.03831045 0.96744577]
```

So, now let me increase this further. So, the error becomes even smaller. So, let us do one thing. Let us alright.

(Refer Slide Time: 13:06)

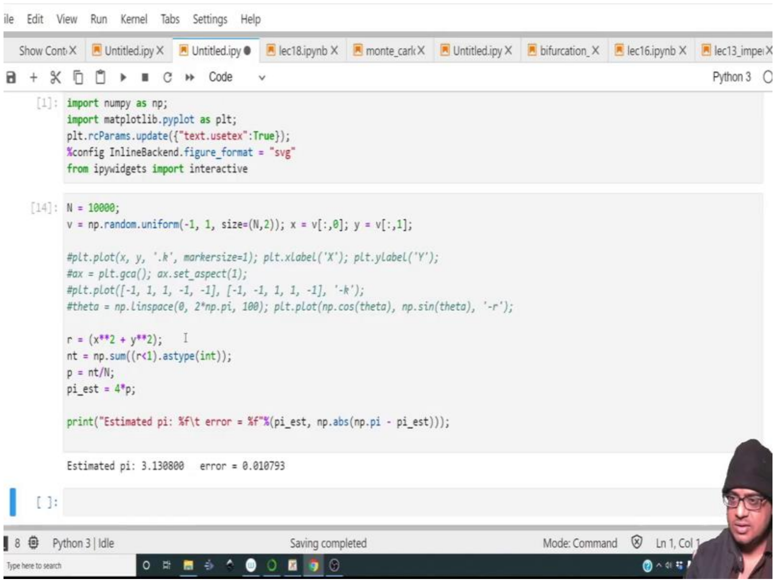


```
pi_est = 4*p;

print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));
```

```
Estimated pi: 3.135200    error = 0.006393
```

It is still running alright, takes a while ok. So, the plotting took a bit of bit of time. But the fact that the whole rectangle is almost black means you have uniformly sampled a lot of points over this entire space ok.

(Refer Slide Time: 13:27)



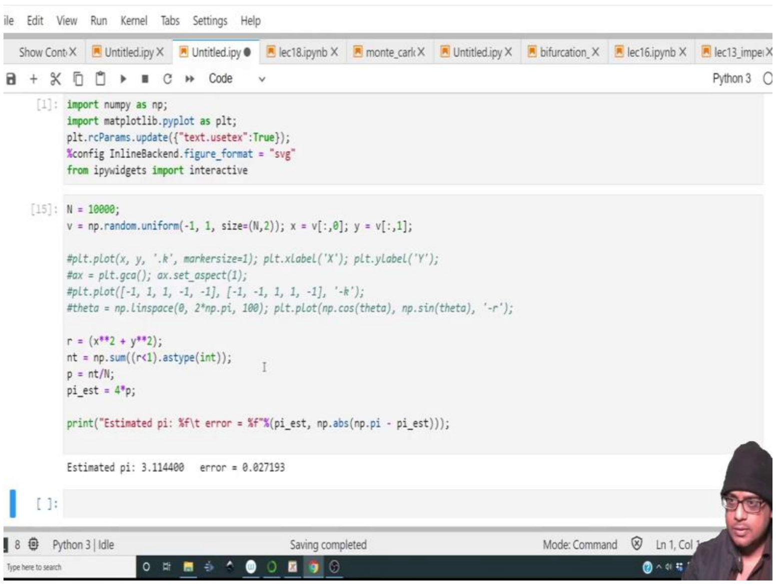
```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive
```

```
[14]: N = 10000;
      v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

      #plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
      #ax = plt.gca(); ax.set_aspect(1);
      #plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
      #theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

      r = (x**2 + y**2);
      nt = np.sum((r<1).astype(int));
      p = nt/N;
      pi_est = 4*p;

      print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));
```

```
Estimated pi: 3.130800    error = 0.010793
```

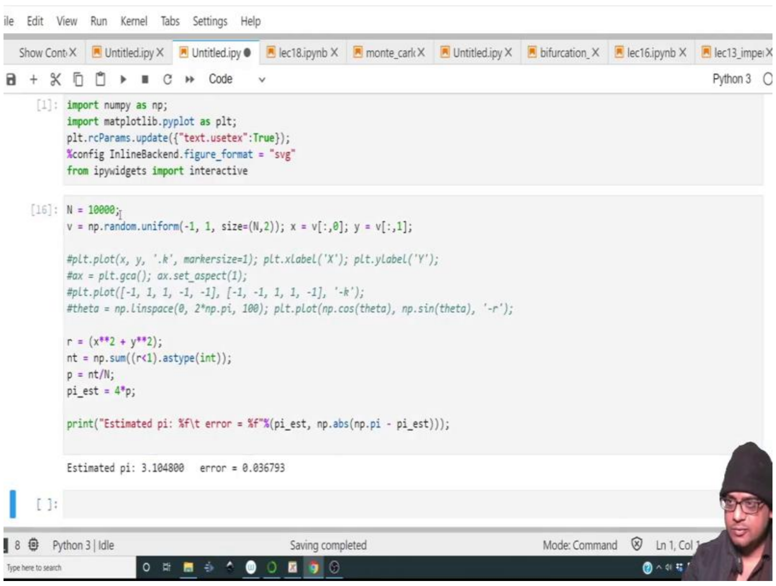
So, in fact, let me reduce this and so, the plotting takes a bit of time. So, in fact, let me ditch the plots. We do not need to really plot all these because we are convinced that it works ok. So, we do not need this theta business also. So, this is the entirety of the code the and the code otherwise runs quite fast.

(Refer Slide Time: 13:54)



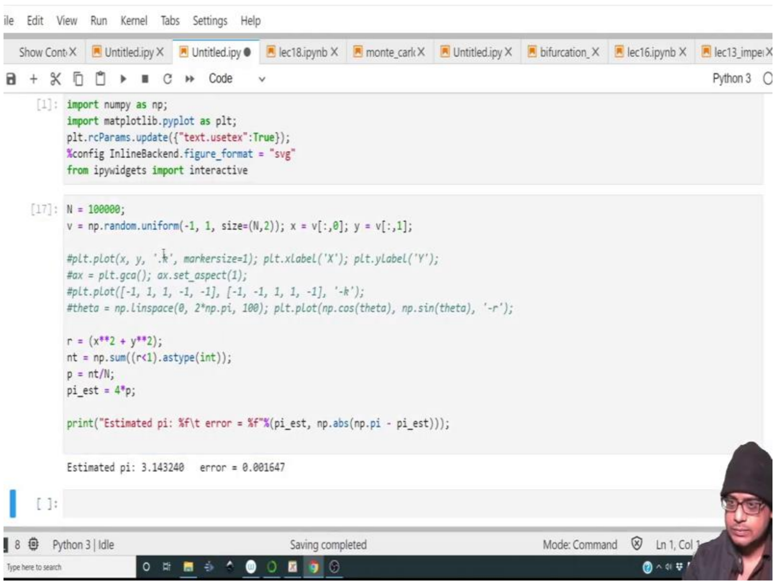
```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive

[15]: N = 10000;
      v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

      #plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
      #ax = plt.gca(); ax.set_aspect(1);
      #plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
      #theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

      r = (x**2 + y**2);
      nt = np.sum((r<1).astype(int));
      p = nt/N;
      pi_est = 4*p;

      print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));

      Estimated pi: 3.114400   error = 0.027193
```

(Refer Slide Time: 13:55)



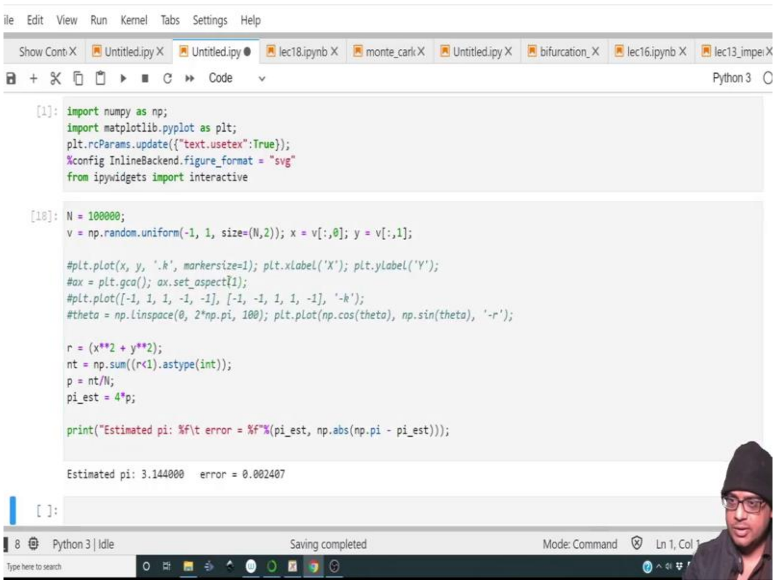
```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive

[16]: N = 10000;
      v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

      #plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
      #ax = plt.gca(); ax.set_aspect(1);
      #plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
      #theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

      r = (x**2 + y**2);
      nt = np.sum((r<1).astype(int));
      p = nt/N;
      pi_est = 4*p;

      print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));

      Estimated pi: 3.104800   error = 0.036793
```

(Refer Slide Time: 13:56)



```
[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

[17]: N = 100000;
      v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

      #plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
      #ax = plt.gca(); ax.set_aspect(1);
      #plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
      #theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

      r = (x**2 + y**2);
      nt = np.sum((r<1).astype(int));
      p = nt/N;
      pi_est = 4*p;

      print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));

      Estimated pi: 3.143240   error = 0.001647
```

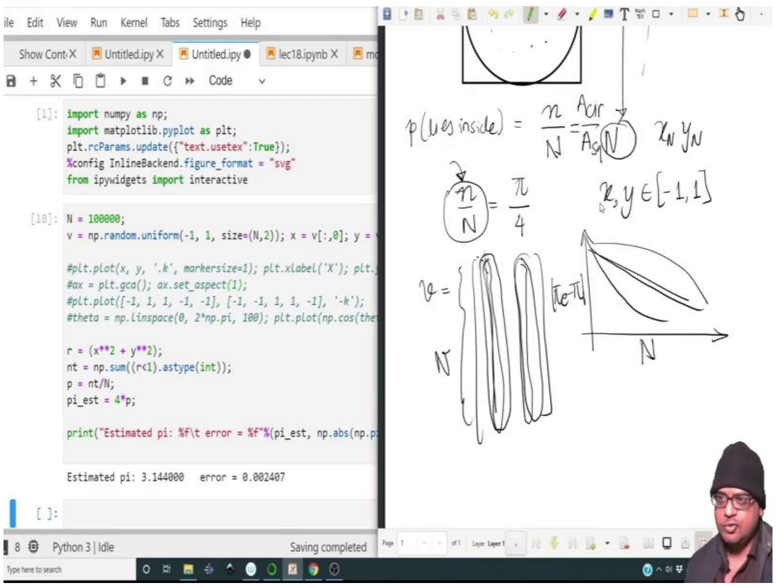
(Refer Slide Time: 14:00)



```
[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

[18]: N = 100000;
      v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];

      #plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.ylabel('Y');
      #ax = plt.gca(); ax.set_aspect(1);
      #plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
      #theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(theta), np.sin(theta), '-r');

      r = (x**2 + y**2);
      nt = np.sum((r<1).astype(int));
      p = nt/N;
      pi_est = 4*p;

      print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.pi - pi_est)));

      Estimated pi: 3.144000   error = 0.002407
```

Plotting seems to create a bit of time delay. So, what we want to do is.

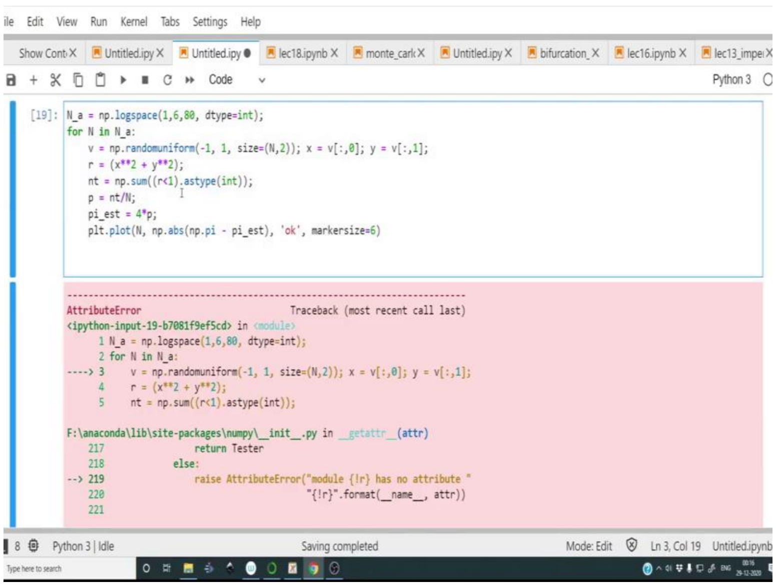
(Refer Slide Time: 14:06)



```
[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

[18]: N = 100000;
      v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y =

      #plt.plot(x, y, '.k', markersize=1); plt.xlabel('X'); plt.
      #ax = plt.gca(); ax.set_aspect(1);
      #plt.plot([-1, 1, 1, -1, -1], [-1, -1, 1, 1, -1], '-k');
      #theta = np.linspace(0, 2*np.pi, 100); plt.plot(np.cos(the

      r = (x**2 + y**2);
      nt = np.sum((r<1).astype(int));
      p = nt/N;
      pi_est = 4*p;

      print("Estimated pi: %f\t error = %f"%(pi_est, np.abs(np.p

      Estimated pi: 3.144000    error = 0.002407

[ ]:
```

So, we know that the more we sample points in space, the better this estimate will be alright. So, against N, how does that π estimate converge or rather π estimate - π? This error, how does it converge? Does it converge like this or like this or like this? So, what is the nature of convergence as we increase N? Alright.

(Refer Slide Time: 14:50)

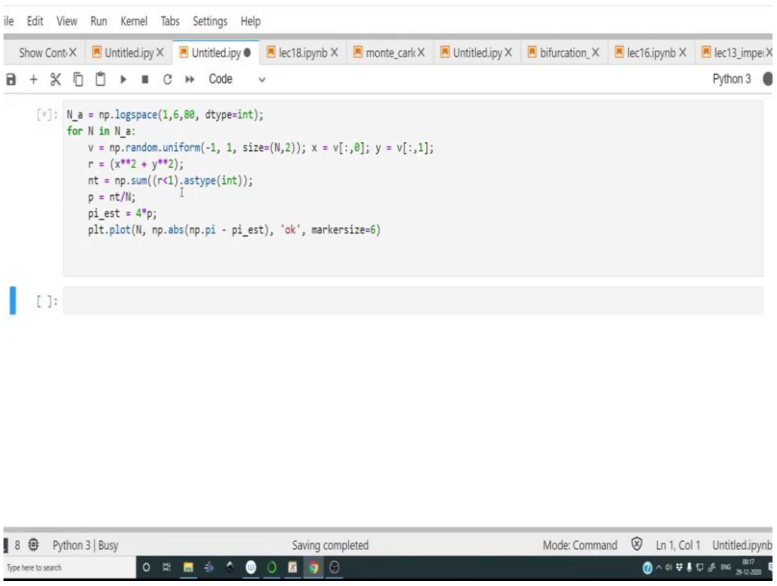


```
[19]: N_a = np.logspace(1,6,80, dtype=int);
      for N in N_a:
          v = np.randomuniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];
          r = (x**2 + y**2);
          nt = np.sum((r<1).astype(int));
          p = nt/N;
          pi_est = 4*p;
          plt.plot(N, np.abs(np.pi - pi_est), 'ok', markersize=6)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-19-b7081f9ef5cd> in <module>
      1 N_a = np.logspace(1,6,80, dtype=int);
      2 for N in N_a:
----> 3     v = np.randomuniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];
      4     r = (x**2 + y**2);
      5     nt = np.sum((r<1).astype(int));

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
```

So, let us do that. Let us let me remove this. In fact, let me take, let me put it in a different cell. Let me remove this alright. So, let me define N_a as np. So, I have created a log space between 1 and 6. I have taken 80 points in that. I have declared the type of the log space as int because when we declare a log space, it is 10 to the power of something right. So, its 10^1, all the way to 10^6 and we are taking 80 points.

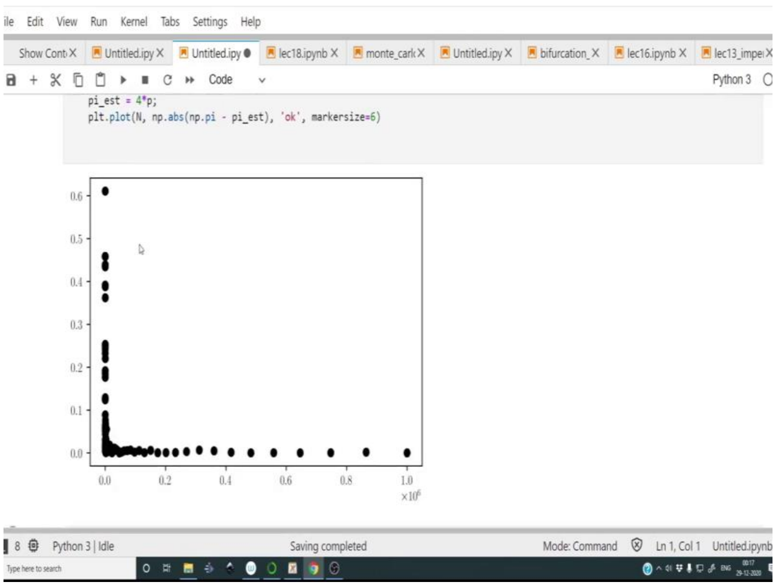
And we are casting it to a type int. So, then we will wrap everything inside this for N in N_a and then we will indent all of this because supposed to be inside the for loop and what we will do is we will do a plot plt.plot. So, on the x axis, we will put N and on the y axis, we will put this error ok. So, and we will put it down as this alright. So, let me get rid of this print statement as well. So, let us see ok; there is an error, let us see ok.

(Refer Slide Time: 16:31)



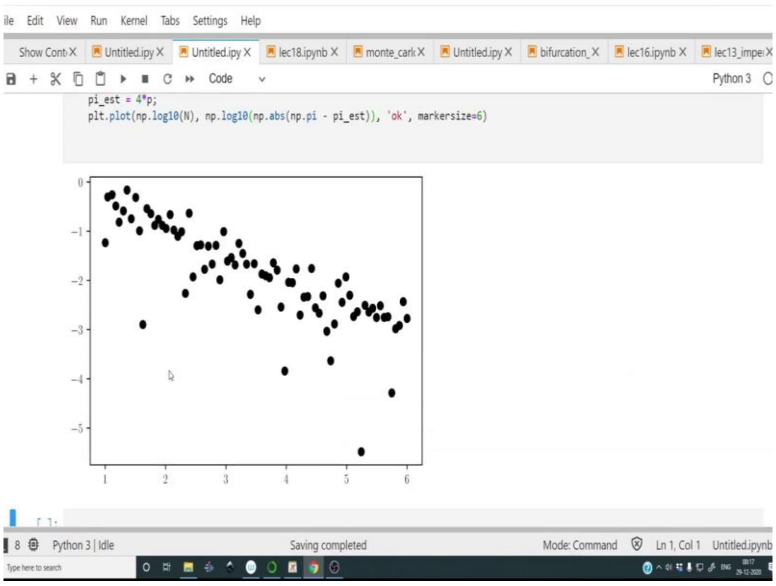
```
[*]: N_a = np.logspace(1,6,80, dtype=int);
     for N in N_a:
         v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];
         r = (x**2 + y**2);
         nt = np.sum((r<1).astype(int));
         p = nt/N;
         pi_est = 4*p;
         plt.plot(N, np.abs(np.pi - pi_est), 'ok', markersize=6)

[ ]:
```

(Refer Slide Time: 16:35)



```
    pi_est = 4*p;
    plt.plot(N, np.abs(np.pi - pi_est), 'ok', markersize=6)
```

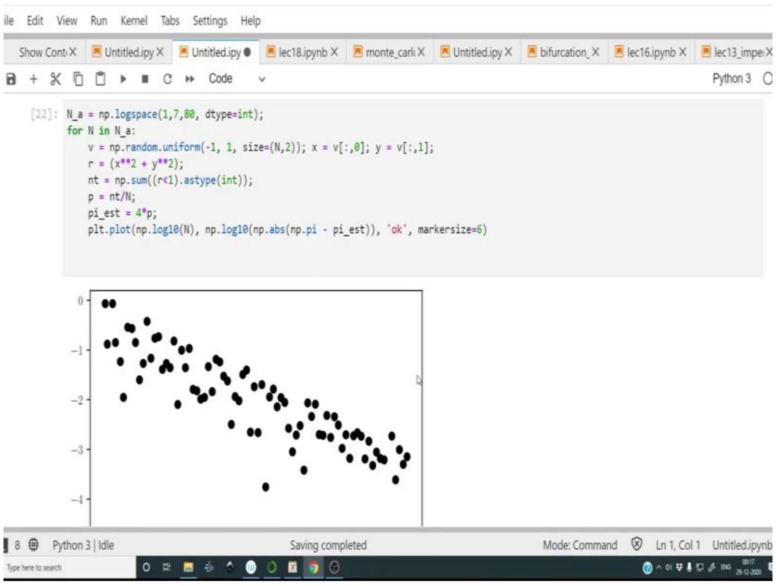
(Refer Slide Time: 16:47)



```
    pi_est = 4*p;
    plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
```

So, obviously, I need to take a log of both the access. So, I will just do that np.log 10; np log 10 and the thing is because it is going across such a large range, it is going from 10^1 all the way to 10^6. So, I need to sort of take a log and represent it in a log scale ok. So, let me run this. It looks something nice. So, it does seem that there is some sort of convergence as N increases.

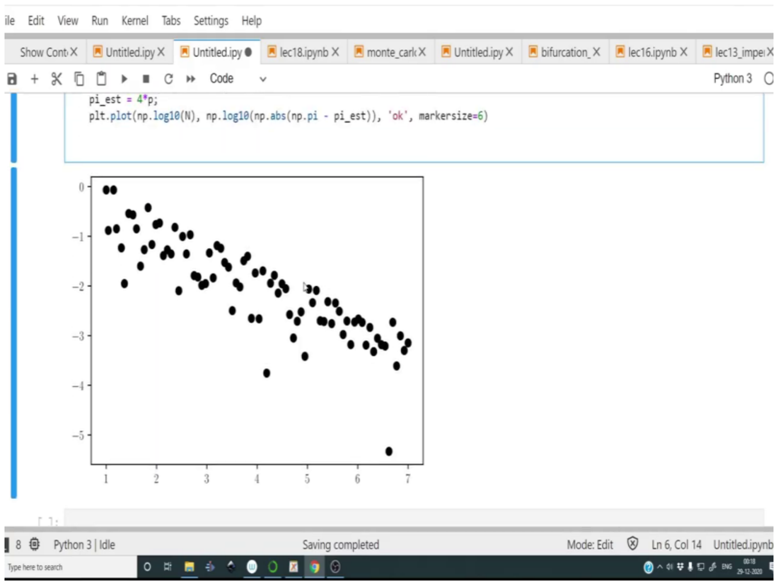
(Refer Slide Time: 17:24)



```
[22]: N_a = np.logspace(1,7,80, dtype=int);
      for N in N_a:
          v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];
          r = (x**2 + y**2);
          nt = np.sum((r<1).astype(int));
          p = nt/N;
          pi_est = 4*p;
          plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
```

So, I do not want to push this further; maybe I can go to 10^7 as well on my machine. I do not want to go beyond that ok, there might be some issues that may occur.

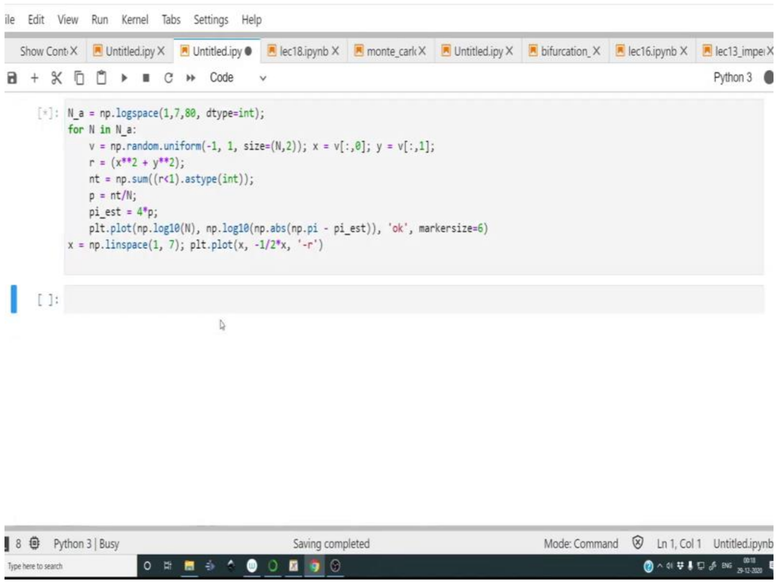
(Refer Slide Time: 17:28)



```
    pi_est = 4*p;
    plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
```

So, let us see what kind of curve, we can fit over here. So, let me do this. It is going from 1 to 7 right.

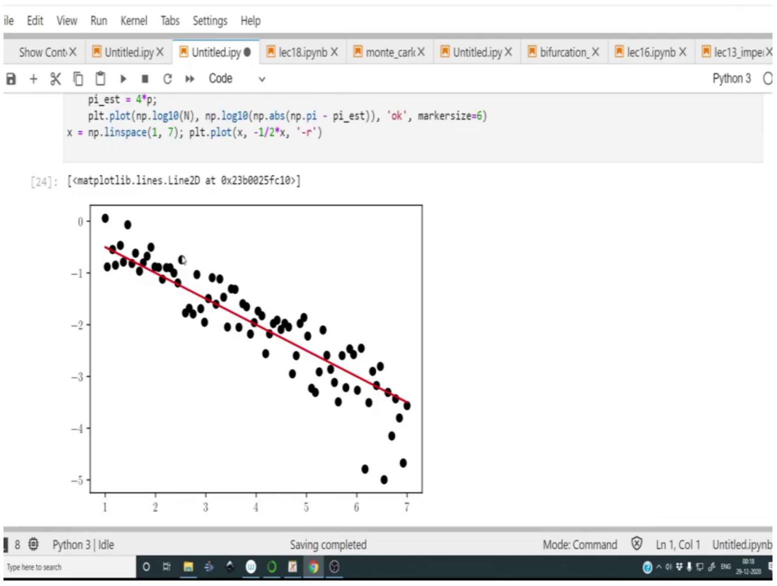
(Refer Slide Time: 17:52)



```
[*]: N_a = np.logspace(1,7,80, dtype=int);
     for N in N_a:
         v = np.random.uniform(-1, 1, size=(N,2)); x = v[:,0]; y = v[:,1];
         r = (x**2 + y**2);
         nt = np.sum((r<1).astype(int));
         p = nt/N;
         pi_est = 4*p;
         plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
     x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')

[ ]:
```

So, let me do this x=np.linspace (1, 7), I will do a plot (x, -1/2×x). We will put down red lined. So, let me run this. This is plt.plot alright.

(Refer Slide Time: 18:19)

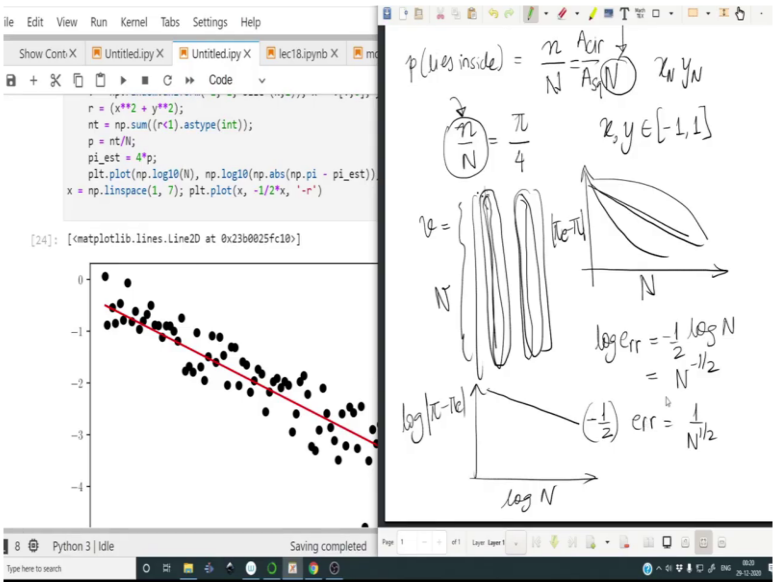


```
    pi_est = 4*p;
    plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')
```

```
[24]: [<matplotlib.lines.Line2D at 0x23b0025fc10>]
```

So, I have fitted this against x, -1/2×x meaning. So, let us just analyze what is going on.

(Refer Slide Time: 18:31)



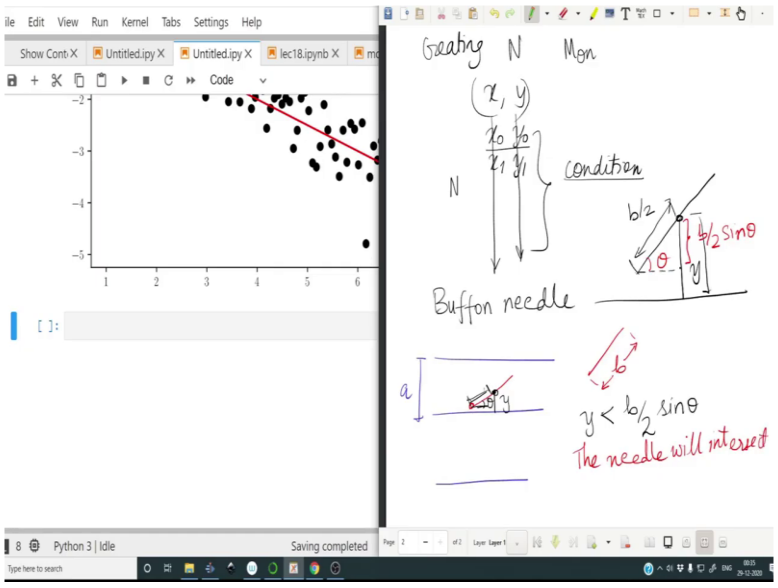
```
    r = (x**2 + y**2);
    nt = np.sum((r<1).astype(int));
    p = nt/N;
    pi_est = 4*p;
    plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est))
x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')
```

```
[24]: [<matplotlib.lines.Line2D at 0x23b0025fc10>]
```

So, on the x axis, you have log N and the y axis you have $\log|\pi - \pi_e|$ and it appears to fit quite well to this particular line which has a slope of -1/2. So, slope of -1/2 means the log of the error, the absolute error is -1/2×log N and this is $N^{-\frac{1}{2}}$ ok.

So, the error goes down as $\frac{1}{N^{\frac{1}{2}}}$ as N increases, the error reduces according to this power law ok. So, this gives us a way of estimating the value of π by sort of creating a bunch of states ok.

(Refer Slide Time: 19:37)



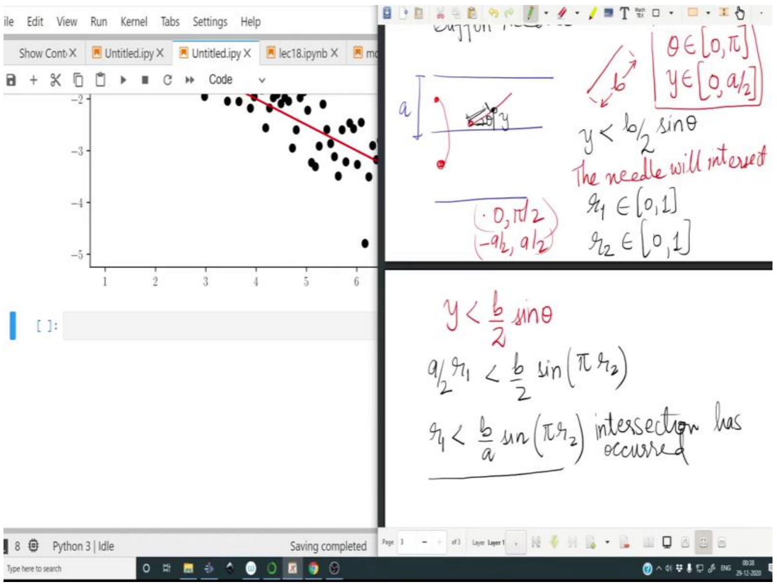
What is really happening is we are creating a whole large number of states. So, the next problem that I want to discuss is that of the Buffon needle. Imagine you have bunch of parallel lines and the distance between each line is a and you have a needle of length b right. Then, you throw this needle randomly on this floor. So, then what is the probability that the needle crosses the line? That is the question.

So, if a needle is something like this, it is not crossing the line; whereas, if the needle is something like this, it is crossing them. It could be it could have any orientation, it need not be like this, it would be something like this and so on. So, that is the basic question.

So, now, the idea is to write down the distance between the midpoint of this needle and the closest it is to this parallel line. So, if I call this distance as y. So, if y is larger than or probably if y is less than the projection of this length. So, this is the midpoint; so, if $y < \frac{b}{2}\sin\theta$.

So, this angel is. So, let me enlarge the figure ok. This is the midpoint; this distance is y. This distance is $\frac{b}{2}$. This distance is going to be $\frac{b}{2}\sin\theta$. So, if $y < \frac{b}{2}\sin\theta$, then the needle will intersect alright. So, now, what can be the prospective values of y and θ?

(Refer Slide Time: 22:38)



$\theta \in [0, \pi]$

$y \in [0, a/2]$

$y < b/2 \sin\theta$

The needle will intersect

$(0, \pi/2)$

$(-a/2, a/2)$

$r_1 \in [0, 1]$

$r_2 \in [0, 1]$

$y < \frac{b}{2}\sin\theta$

$a/2\, r_1 < \frac{b}{2}\sin(\pi r_2)$

$r_1 < \frac{b}{a}\sin(\pi r_2)$ intersection has occurred

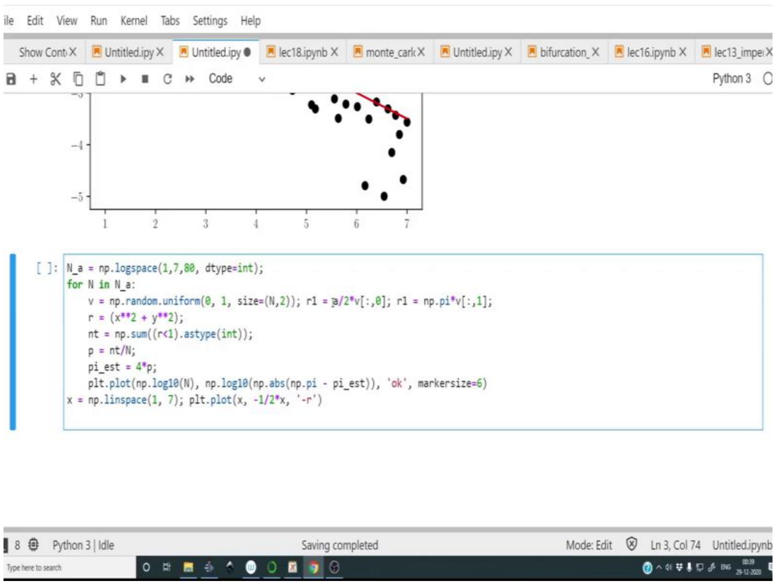
So, obviously, θ can go all the way from; so, it can go all the way from 0 to π and it can take all orientations from 0 to π; whereas, y can take all values from, so this midpoint can be directly on the line, so it can be 0 to the first case, it can be at the midpoint between these 2 lines ok. Now, it is symmetric in the y direction.

So, it can go all the way from 0 to a by 2. So, if you just think about it, this is what you will get. In fact, you can also do the following; if you can go from 0 to pi by 2, but then you have to sample y from -$\frac{a}{2}$ to $\frac{a}{2}$. But anyway, we do not want to discuss this too much and you we want to get to the numeric software.

So, then finally, the condition that we have obtained is this. So, now, let us focus on this condition. So, let us sample r1 between 0 and 1 so that y can be $\frac{a}{2} \times r1$, similarly θ can be sampled between; so, let r2 be also another sample drawn uniformly between 0 and 1.

So, this θ will be will be simply $\pi \times r2$. So, now, if $r1 < \frac{b}{a}\sin(\pi \times r2)$, then we say that intersection has occurred; otherwise, intersection has not occurred alright. So, let us encode this; it is not at all difficult to encode.

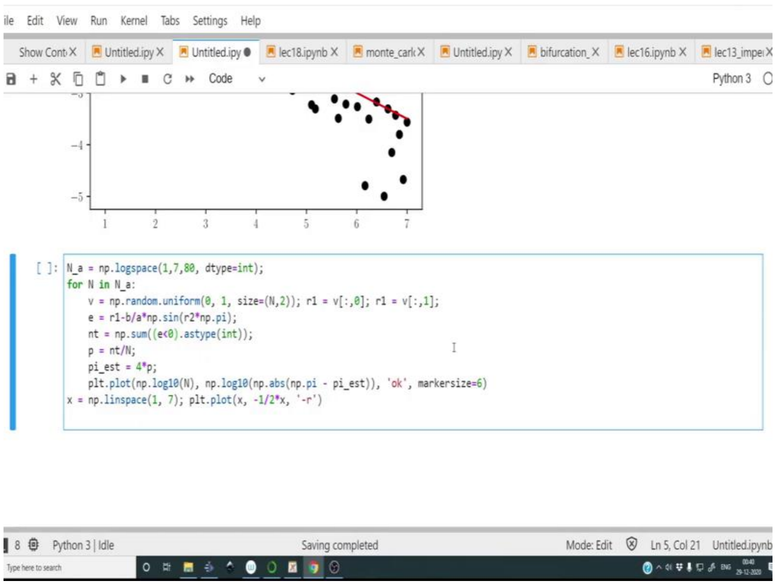
(Refer Slide Time: 25:28)



```
[ ]: N_a = np.logspace(1,7,80, dtype=int);
     for N in N_a:
         v = np.random.uniform(0, 1, size=(N,2)); r1 = a/2*v[:,0]; r1 = np.pi*v[:,1];
         r = (x**2 + y**2);
         nt = np.sum((r<1).astype(int));
         p = nt/N;
         pi_est = 4*p;
         plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
     x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')
```

And in fact, we can reuse this program. So, let me yes instead of sampling from -1 to 1, we will sample from 0 to 1 and we will make this as r1, we make this as r2 and we will multiply it by $\frac{a}{2}$ and we will multiply this by π alright. So, then we will check. In fact, r1 and r2 are just sampled between 0 and 1, they are not multiplied by this because we have obtained the inequality in terms of the random numbers r1 and r2 rather than something else. So, let me fix this ok.

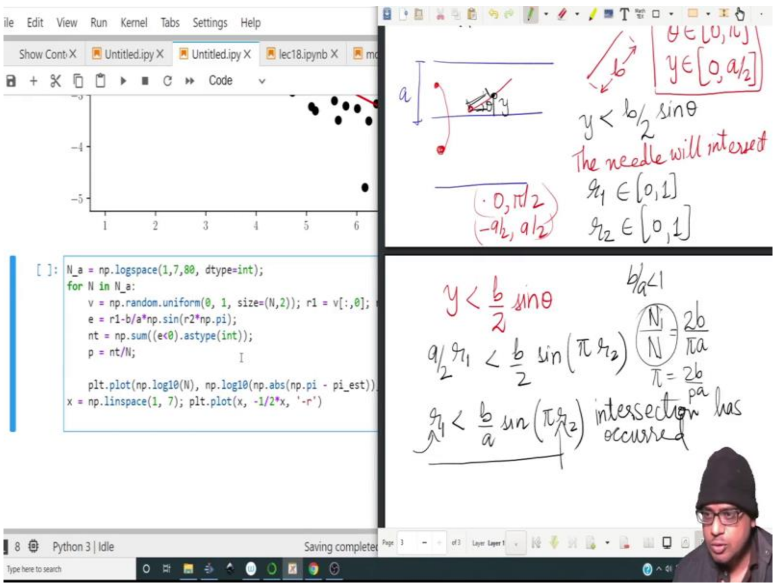
(Refer Slide Time: 26:24)



```
N_a = np.logspace(1,7,80, dtype=int);
for N in N_a:
    v = np.random.uniform(0, 1, size=(N,2)); r1 = v[:,0]; r1 = v[:,1];
    e = r1-b/a*np.sin(r2*np.pi);
    nt = np.sum((e<0).astype(int));
    p = nt/N;
    pi_est = 4*p;
    plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')
```

So, now, that we have this we can write down the inequality. So, we can write $e = r1 - \frac{b}{a} np.\sin(\pi \times r2)$ and then, we will find out when e < 0, then find out how all the values and find out the sum of all the values and then, the probability is this. Well, the estimate for π_est for now we will simply plot well.

(Refer Slide Time: 27:28)

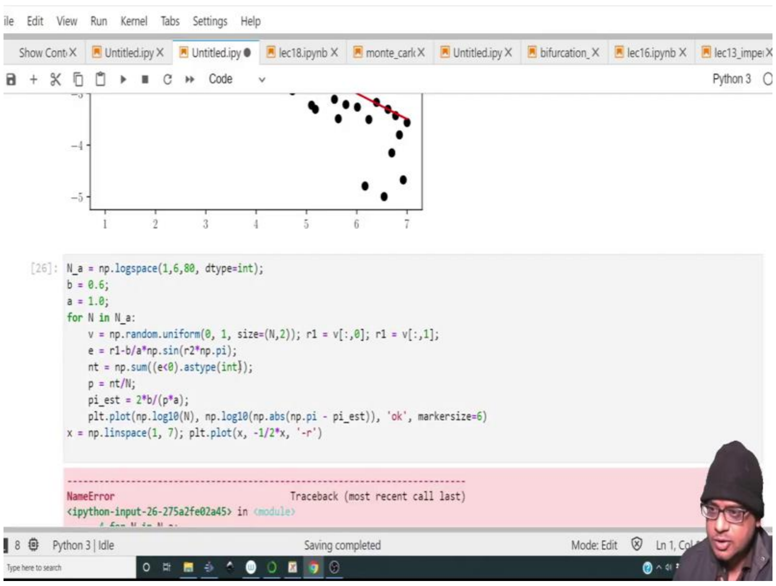


```
N_a = np.logspace(1,7,80, dtype=int);
for N in N_a:
    v = np.random.uniform(0, 1, size=(N,2)); r1 = v[:,0];
    e = r1-b/a*np.sin(r2*np.pi);
    nt = np.sum((e<0).astype(int));
    p = nt/N;

    plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est))
x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')
```

a
y
b
$\theta \in [0,\pi]$
$y \in [0, a/2]$
$y < \frac{b}{2}\sin\theta$
The needle will intersect
$(\cdot\, 0, \pi/2)$
$(-a/2, a/2)$
$r_1 \in [0,1]$
$r_2 \in [0,1]$
$y < \frac{b}{2}\sin\theta$
$b/a < 1$
$\frac{a}{2} r_1 < \frac{b}{2}\sin(\pi r_2)$
$\frac{N_1}{N} = \frac{2b}{\pi a}$
$\pi = \frac{2b}{pa}$
$r_1 < \frac{b}{a}\sin(\pi r_2)$ intersection has occurred

In this particular problem, we can show analytically that when $\frac{b}{a} < 1$, then the probability that this number of samples which intersect upon total number of samples that will be something like or something like it will be $\frac{2b}{\pi}$. So, the estimate for π has to be $\frac{2b}{p \times a}$; where, $p \times a$ is this probability. So, let us encode that information.

(Refer Slide Time: 27:59)

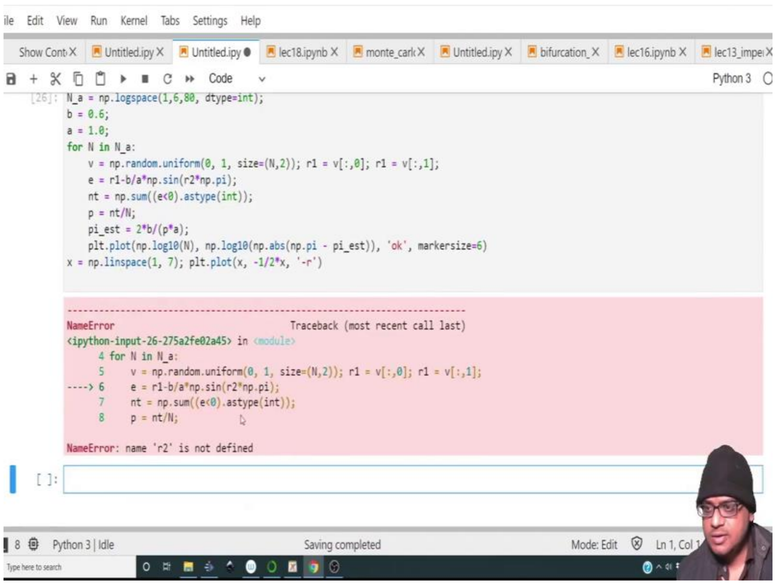


```
[26]: N_a = np.logspace(1,6,80, dtype=int);
b = 0.6;
a = 1.0;
for N in N_a:
    v = np.random.uniform(0, 1, size=(N,2)); r1 = v[:,0]; r1 = v[:,1];
    e = r1-b/a*np.sin(r2*np.pi);
    nt = np.sum((e<0).astype(int));
    p = nt/N;
    pi_est = 2*b/(p*a);
    plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-275a2fe02a45> in <module>
```

So, let me write down π_est = $\frac{2b}{p \times a}$ alright. So, let me reduce this. Let me define what b (Refer Time: 28:18). So, let b be 0.6, let a be 1. So, let me run this ok. There is another r2 is not required (Refer Time: 28:29) ok.

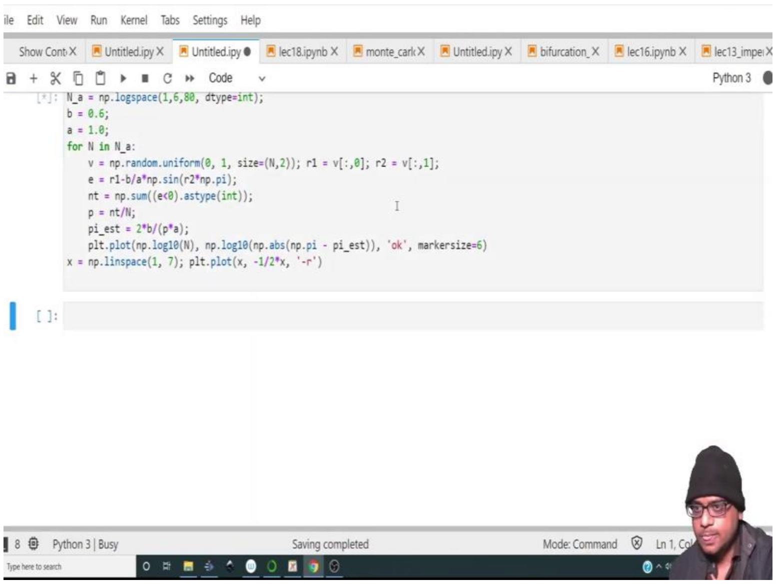
(Refer Slide Time: 28:23)



```
[26]: N_a = np.logspace(1,6,80, dtype=int);
      b = 0.6;
      a = 1.0;
      for N in N_a:
          v = np.random.uniform(0, 1, size=(N,2)); r1 = v[:,0]; r1 = v[:,1];
          e = r1-b/a*np.sin(r2*np.pi);
          nt = np.sum((e<0).astype(int));
          p = nt/N;
          pi_est = 2*b/(p*a);
          plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
      x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')
```

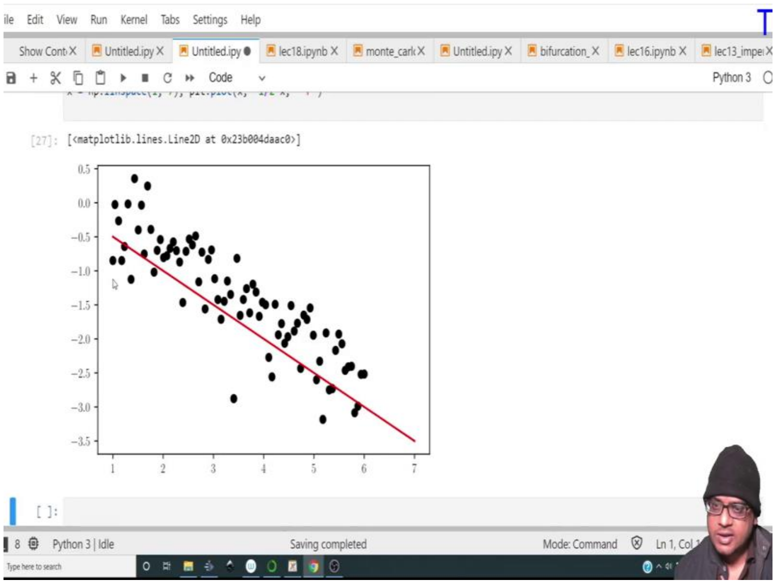
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-275a2fe02a45> in <module>
      4 for N in N_a:
      5     v = np.random.uniform(0, 1, size=(N,2)); r1 = v[:,0]; r1 = v[:,1];
----> 6     e = r1-b/a*np.sin(r2*np.pi);
      7     nt = np.sum((e<0).astype(int));
      8     p = nt/N;

NameError: name 'r2' is not defined
```

(Refer Slide Time: 28:28)



```
[*]: N_a = np.logspace(1,6,80, dtype=int);
     b = 0.6;
     a = 1.0;
     for N in N_a:
         v = np.random.uniform(0, 1, size=(N,2)); r1 = v[:,0]; r2 = v[:,1];
         e = r1-b/a*np.sin(r2*np.pi);
         nt = np.sum((e<0).astype(int));
         p = nt/N;
         pi_est = 2*b/(p*a);
         plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est)), 'ok', markersize=6)
     x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')
```

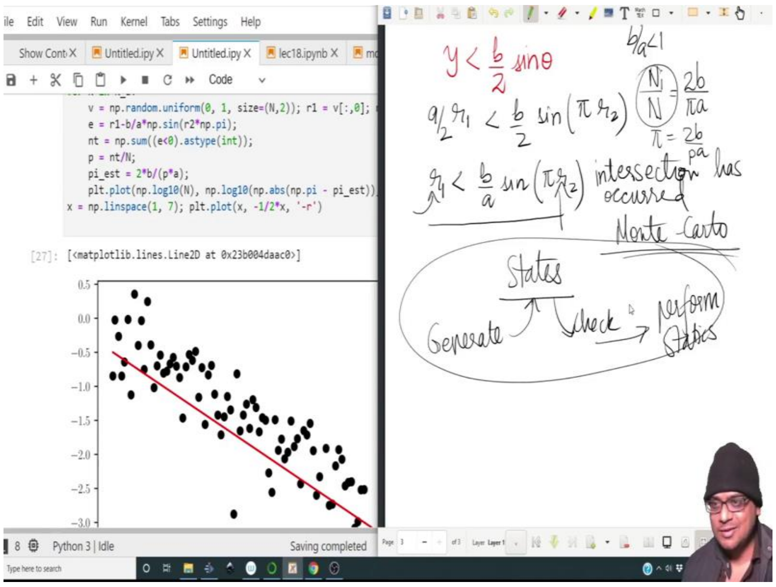
(Refer Slide Time: 28:29)



```
[27]: [<matplotlib.lines.Line2D at 0x23b004daac0>]
```

Let me run this, we have two convergence, once again as they are increases through Buffon needle experiment as well. So, we have done two Monte-Carlo simulations. So, why is this again? Why is this also a Monte-Carlo simulation? Because you are generating states.

(Refer Slide Time: 28:52)



```
    v = np.random.uniform(0, 1, size=(N,2)); r1 = v[:,0];
    e = r1-b/a*np.sin(r2*np.pi);
    nt = np.sum((e<0).astype(int));
    p = nt/N;
    pi_est = 2*b/(p*a);
    plt.plot(np.log10(N), np.log10(np.abs(np.pi - pi_est))
x = np.linspace(1, 7); plt.plot(x, -1/2*x, '-r')

[27]: [<matplotlib.lines.Line2D at 0x23b004daac0>]
```

$$y < \frac{b}{2}\sin\theta \qquad b/a < 1$$

$$\frac{a}{2} r_1 < \frac{b}{2}\sin(\pi r_2) \qquad \frac{N_i}{N} = \frac{2b}{\pi a}$$

$$\pi = \frac{2b}{pa}$$

$$r_1 < \frac{b}{a}\sin(\pi r_2)$$ intersection has occurred

Monte-Carlo

States

Generate → check → perform Statics

You are not going through the evolution of a needle as it is bouncing around, you are generating a state for the needle; we are checking a certain condition against that state ok. So, you generate states, you check the states and then, you perform the statistics and that is pretty much Monte-Carlo interaction. And put those (Refer Time: 29:30) Monte-Carlo is seen it.

Well, region to the south of firms and it is known for its casinos and gambling and so, the fact that you are using a technique which draws random which makes random states, it sort of resembles sort of a gambling process going on; not a gambling process, but draw off cards that you make ok, you are drawing states just like you would do in a casino and that is why it is called as Monte-Carlo.

This is just a quick online history of this, but you know I mean this can be a very powerful method of achieving a lot of things. In fact, Monte-Carlo simulation are often used to minimize energy of various molecular systems because you can generate a whole bunch of state space like positions and you can generate a whole lot of velocities and you can check whether that particular state that you have generated whether that has a certain energy threshold.

It is lower than threshold, then you accept it. You do, you generate a new state based on that. So, there can be a whole lot of physical relevant things, you can do with it and I have shown you two ways how you can estimate π to 2 processes and this by known in the trivial task to understand how you should sample something because in the value involving Buffon needle, we were sampling between 0 and 1 uniform way (Refer Time: 31:21) for pi problem, we were on the darts problem.

These also called as darts problem. We were sampling uniformly between minus 1 and 1 and for any other problem, you have to really figure out as to how you will go about generating these or how you will describe the system in terms of a few random samples with sets of random samples and as to how you would set a condition to check; so, all that depends on the experience and your intuition.

Well, this and this particular lecture and next time, we will look at some dynamical simulations like random box. Until then, it is good bye; bye.