

Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Lecture – 25
Handling Real-world Network Datasets
Programming Illustration: Emergence of Connectedness

We just now looked at a very surprising and interesting result which says that on a graph if we have n nodes then one just needs $n \log n$ number of edges to make this graph connected that is if we take n nodes and we randomly put $n \log n$ edges in this graph, the graph becomes connected. So, you can start at any node in this network and from this node you can find a path to every other node in the network.

Now, we will be doing a screen cast for it we will be writing a piece of code in which we will take random graphs of different sizes and then we will look at the number of edges required to make this graph connected. So, we will do this, we will look at the plot, let us get started.

(Refer Slide Time: 00:50)



```
In [149]: !notepad connectivity.py

In [150]: import connectivity

In [151]: connectivity.add_nodes(10)
Out[151]: <networkx.classes.graph.Graph at 0xa2cbb38>

In [152]: G=connectivity.add_nodes(10)
Out[152]: 10

In [153]: G.number_of_nodes()
Out[153]: 10

In [154]: nx.is_connected(G)
Out[154]: False

In [155]: !notepad connectivity.py

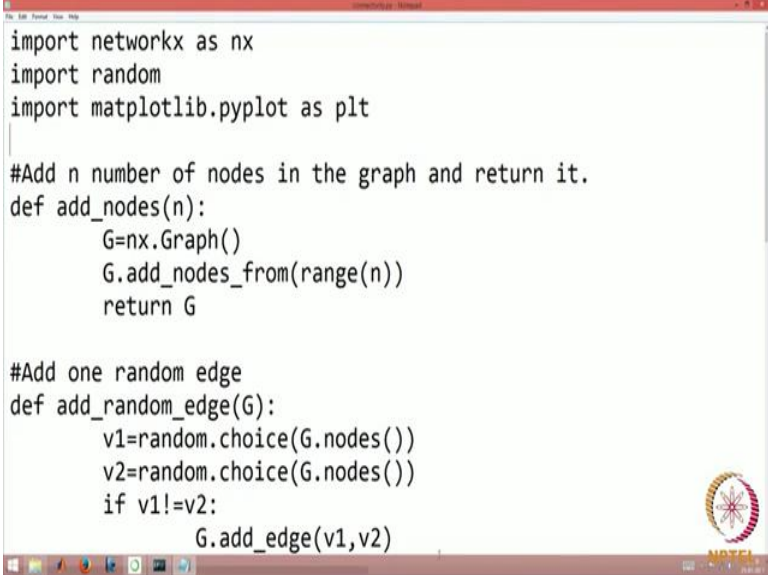
In [156]: reload(connectivity)
Out[156]: <module 'connectivity' from 'connectivity.py'>

In [157]: G=connectivity.add_random_edge(G)
```

Let us start from very basic coding. So, what we want to do is we want to have some nodes in the graph and then we add random edges in this network one at a time and our aim is to see after how after addition of how many edges does this network becomes

connected. So, we first create a file for this. So, let us open a new file in notepad let us name it as connectivity.py.

(Refer Slide Time: 01:26)

A screenshot of a Notepad window titled 'connectivity.py - Notepad'. The window contains Python code for creating a graph. The code imports 'networkx as nx', 'random', and 'matplotlib.pyplot as plt'. It defines a function 'add_nodes(n)' that creates a graph 'G' using 'nx.Graph()', adds nodes from 'range(n)' using 'G.add_nodes_from()', and returns 'G'. It also defines a function 'add_random_edge(G)' that selects two random nodes 'v1' and 'v2' from 'G.nodes()', checks if they are not equal, and adds an edge between them using 'G.add_edge(v1, v2)'. The Windows taskbar is visible at the bottom, and a small circular logo is in the bottom right corner of the Notepad window.

```
import networkx as nx
import random
import matplotlib.pyplot as plt

#Add n number of nodes in the graph and return it.
def add_nodes(n):
    G=nx.Graph()
    G.add_nodes_from(range(n))
    return G

#Add one random edge
def add_random_edge(G):
    v1=random.choice(G.nodes())
    v2=random.choice(G.nodes())
    if v1!=v2:
        G.add_edge(v1,v2)
```

So, here we have a file oh we first of all import our basic packages which we will need import networkx as nx see I think it is sufficient for the time being. So, what we do? We first define a function to add some specified number of nodes in the network. So, the function already exists in networkx you all know, but just for the sake of simplicity and clarification I will again make a function here and let us name this function as add nodes and you pass a number n as a parameter to this function. So, what this function is going to do is it will create a graph $G = nx.Graph$ and add n number of nodes in this graph. So, we can do $G.add_nodes_from$ I hope you remember the function add nodes from. So, this function adds the nodes in the graph from a list and the list we give here.

So, the list is from number 0 to number n - 1. So, n nodes get added to this graph and we finally, return the graph simple function we will just keep commenting everything side by side. So, we comment it add n number of nodes in the graph and return it perfect. So, we will save it and let us come back to our window.

So, what we need to do now is import this file we import connectivity. So, this gets imported let us call a function to call a function we have the syntax connectivity dot function name our function name is add underscore nodes and me and we had let us say we want to add 10 nodes in the graph you execute it. So, we get a graph here. So, our

function return the graph. So, let us collect the output in a graph name G. So, `G = connectivity.add_nodes(10)` now I want to see whether 10 nodes have been actually added in the graph or not. So, we can see that using the function `G.number_of_nodes` you see that there are 10 nodes in this graph next, we will write a function for adding one random edge to this network.

So, if you see currently in this network there are just 10 nodes there are no edges. So, this network is disconnected completely disconnected. So, to check that we have command `nx.is_connected` let say you want to see whether this graph is connected or not let us call. So, before using `nx.is_connected` parse graph G here which shows us false if the graph is not connected now one by one, we add edges to this network. So, first we write a code for adding one random edge to this network we go back to our file which was `connectivity.py` and now we add second function here and the function is for add one random edge in this network and one random edge.

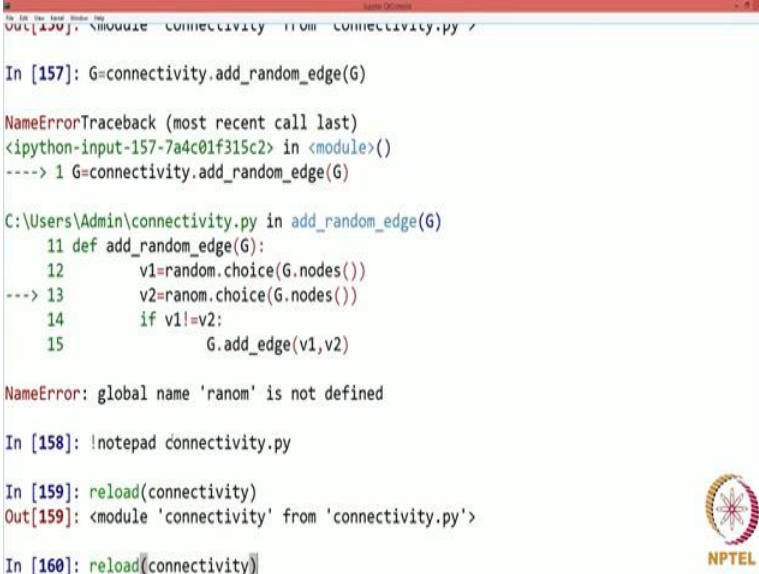
Let us name the function as add random edge and it takes us input the graph G graph G which has been already created with no edges. So, we pass that graph G actually you can pass a graph G with the edges also that is up to us. So, the function takes as input one in graph and adds one extra random edge to this graph define add underscore random edge G, now we need 2 random vertices from the network. So, how do we pick a random vertex? So, to put to pick a random entry we need one package name random. So, we have discussed about it before in the first week what does it do, we are going to use the function `random.choice` first vertex `v1`. So, this is the first random vertex which we want to pick it is `random.choice` function takes as the input one list and outputs as one random value from that list.

So, the list which we are going to pass here is a `G.nodes` that is the list of nodes in the network. So, `v1` gets first random node from this network let us choose another node `v2` `v2` equals to another random node `random.choice` and again we have here `G.nodes` we got 2 random vertices now and we want to create an edge between them, but there might be a case where `v1` and `v2` might be same nodes. So, we would have picked both the random nodes to be same although that is a very rare case that might happened, but we do not want that to happen because we do not want any self loops in our graph. So, we just put a condition here if `v1 != v2` only then we add an edge that is these 2 vertices

should be separate. So, if $v1 \neq v2$ we go ahead and add an edge `G.add_edge` from `v1` to `v2` after adding this edge we return a graph `G` save it we come back to our console.

Let us now we not need to reload our file. So, we reload connectivity since we have made changes to it reload connectivity now the spelling is correct reload connectivity. So, it is reloaded we already have a graph here which has 10 nodes. So, what do we do now is `G` equals to `connectivity` dot and we call our function add random edge on the graph `G` itself seems to be some problem?

(Refer Slide Time: 08:07)



```
Out[150]: <module 'connectivity' from 'connectivity.py'>

In [157]: G=connectivity.add_random_edge(G)

NameErrorTraceback (most recent call last)
<ipython-input-157-7a4c01f315c2> in <module>()
----> 1 G=connectivity.add_random_edge(G)


C:\Users\Admin\connectivity.py in add_random_edge(G)
     11 def add_random_edge(G):
     12     v1=random.choice(G.nodes())
--> 13     v2=ranom.choice(G.nodes())
     14     if v1!=v2:
     15         G.add_edge(v1,v2)

NameError: global name 'ranom' is not defined

In [158]: !notepad connectivity.py

In [159]: reload(connectivity)
Out[159]: <module 'connectivity' from 'connectivity.py'>

In [160]: reload(connectivity)
```



So, it is a name error which says that global name there is a spelling mistake. So, let us go back and correct it if we see here yeah just small spelling error, we correct it save it come back reload it and call the function again it works fine. So, we have we now one random edge should have been added in a graph.

(Refer Slide Time: 08:36)

```
15 G.add_edge(v1,v2)

NameError: global name 'ranom' is not defined

In [158]: !notepad connectivity.py

In [159]: reload(connectivity)
Out[159]: <module 'connectivity' from 'connectivity.py'>

In [160]: G=connectivity.add_random_edge(G)

In [161]: G.edges()
Out[161]: [(0, 7)]

In [162]: !notepad connectivity.py

In [163]: reload(connectivity)
Out[163]: <module 'connectivity' from 'connectivity.py'>

In [164]: G=connectivity.add_till_connectivity(G)

In [165]: G.edge
```

So, let us see it let us look at the list of edges in a graph. So, we do `G.edges` and we can see here that a random edge and edge has been added from 2 vertices 0 to 7. So, we have written the code till now for adding a specific number of vertices in the graph and adding one random edge what is our aim? Our aim is to keep adding these random edges in the network till our network becomes connected and then to look at how many edges were these which made this network connected for doing that we go back to our code.

(Refer Slide Time: 09:22)

```
G.add_nodes_from(range(n))
return G

#Add one random edge
def add_random_edge(G):
    v1=random.choice(G.nodes())
    v2=random.choice(G.nodes())
    if v1!=v2:
        G.add_edge(v1,v2)
    return G

#Keeps adding random edges in G till it becomes connected
def add_till_connectivity(G):
    while(nx.is_connected(G)==False):
        G=add_random_edge(G)
```

And let us create another function here what is function does is it add random edges let rather say keep keeps adding random edges in G till it becomes connected and let us name this function as add_till_connectivity add_till_connectivity and it takes the graph G as input. So, what it is going to do till the graph does not become sorry till the graphs become connected.

So, while nx.is_connected g. So, this is a function in networkx which returns true if the graph is connected false if it is not connected. So, while nx.is_connected(G) = false it keeps running and what does it do? It adds one random edge to the graph for graph G becomes add random edge G. So, one random edge is added to this graph and it keeps adding more random edges till the graph becomes connected and at the end, let us return the graph. So, let us look at what this code is doing again we reload connectivity and now what we are going to do is G = connectivity.add_till_connectivity. So, let us see how many how many edges it adds. So, we already had one edge which we have added previously, and we pass the graph G here and let us now look at G.edges.

(Refer Slide Time: 11:19)



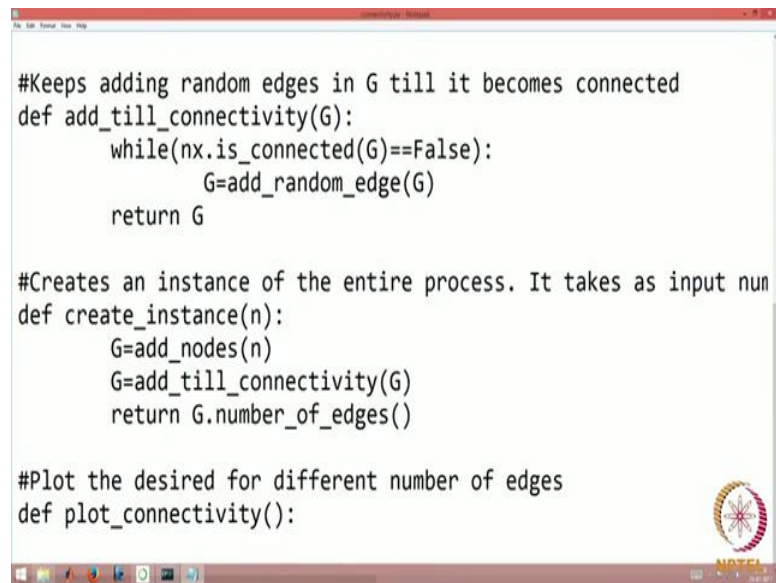
```
File Edit View Insert Window Help
In [165]: G.edges()
Out[165]:
[(0, 4),
 (0, 5),
 (0, 7),
 (1, 8),
 (1, 4),
 (1, 6),
 (1, 7),
 (2, 8),
 (2, 4),
 (2, 7),
 (3, 7),
 (4, 8),
 (4, 6),
 (5, 9),
 (5, 6),
 (6, 8)]

In [166]: G.number_of_edges()
Out[166]: 16

In [167]:
```

So, we see that many edges have been added to this network to be precise the number of edges added is 16.

(Refer Slide Time: 11:36)

A screenshot of a code editor window with a red title bar. The code is written in Python and defines three functions: `add_till_connectivity`, `create_instance`, and `plot_connectivity`. The `add_till_connectivity` function uses a while loop to add random edges until the graph is connected. The `create_instance` function creates a graph with `n` nodes, adds edges until it is connected, and returns the number of edges. The `plot_connectivity` function is partially visible. A small circular logo with a star-like pattern is in the bottom right corner of the code area.

```
#Keeps adding random edges in G till it becomes connected
def add_till_connectivity(G):
    while(nx.is_connected(G)==False):
        G=add_random_edge(G)
    return G

#Creates an instance of the entire process. It takes as input num
def create_instance(n):
    G=add_nodes(n)
    G=add_till_connectivity(G)
    return G.number_of_edges()

#Plot the desired for different number of edges
def plot_connectivity():
```

So, it seems like in a graph having sixteen nodes just sixteen edges are required to make this graph connected let us also quickly check whether its connected or not. So, `nx.is_connected G` we can say that it is we can see that it is connected it is true.

Till now what we have done is we have written a small piece of code which takes a graph as input keeps adding edges to this graph till the graph becomes connected next what we want to do is to repeat this procedure for different number of nodes. So, here for example, in this we did it for 10 nodes and we looked at that the number of edges required was 16, now want to do it for 20 look at the number want to do for 30 look at the number and so on.

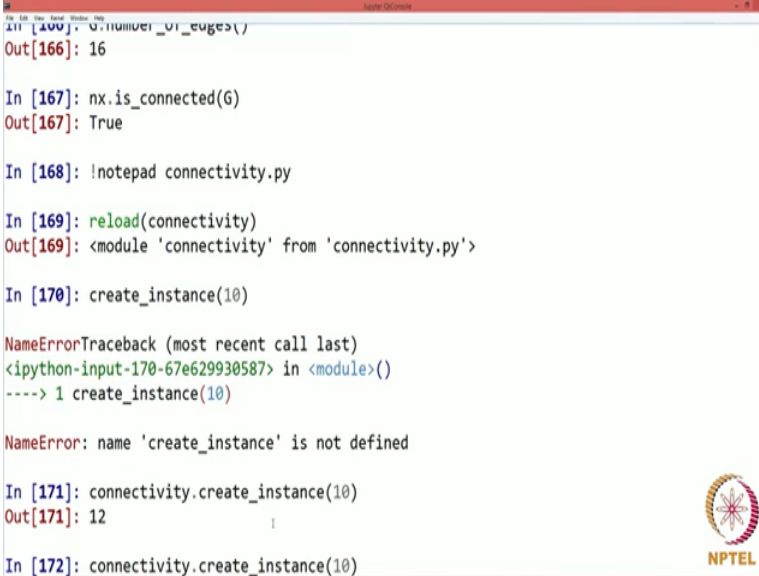
So, for doing that first of all let us create a new function here. So, this function is going to simplify what all we have written here let us name the function create creates an instance of the entire process. So, what do I mean here by instance of an entire process? It takes as input. So, the function will simply take as input number of nodes and returns the number of edges required for connectivity. So, it will make a task easier. So, we will just input a number and a number of nodes and it will output us it will tell us how many edges are required to connect a graph having these many nodes.

So, let us define this function let us name it as create instance and whatever is required inside the code for this function, we have already done that. So, define create underscore instance and let us pass a number and here what we do next is first of all we have to add

nodes. So, `G = add_underscore_nodes` and we add `n` nodes. So, we called the first function which we defined previously. So, whatever we have done till now in Ipython that only we are replicating here in the file.

So, we first call this function `add_underscore_nodes` and which returns us the graph and next we can simply call `G.equals` true keep adding till the graph gets connected. So, add till connectivity we again pass here `G` and it will return us what it will return us the number of edges return `G.number_of_edges`.

(Refer Slide Time: 14:50)



```

In [166]: G.number_of_edges()
Out[166]: 16

In [167]: nx.is_connected(G)
Out[167]: True

In [168]: !notepad connectivity.py

In [169]: reload(connectivity)
Out[169]: <module 'connectivity' from 'connectivity.py'>


In [170]: create_instance(10)

NameErrorTraceback (most recent call last)
<ipython-input-170-67e629930587> in <module>()
----> 1 create_instance(10)

NameError: name 'create_instance' is not defined

In [171]: connectivity.create_instance(10)
Out[171]: 12

In [172]: connectivity.create_instance(10)
```



Let us see how it does? how is it working reload the file and let us call `create_underscore_instance` and let us pass the number 10 here. So, it is in our file `connectivity.py`. So, `connectivity.create_instance(10)`. So, we see that if there are 10 nodes, we need 12 edges.

(Refer Slide Time: 15:14)

```

NameErrorTraceback (most recent call last)
<ipython-input-170-67e629930587> in <module>()
----> 1 create_instance(10)

NameError: name 'create_instance' is not defined

In [171]: connectivity.create_instance(10)
Out[171]: 12

In [172]: connectivity.create_instance(20)
Out[172]: 27


In [173]: connectivity.create_instance(30)
Out[173]: 50

In [174]: !notepad connectivity.py

In [175]: reload(connectivity)
Out[175]: <module 'connectivity' from 'connectivity.py'>

In [176]: connectivity.plot_connectivity()

```



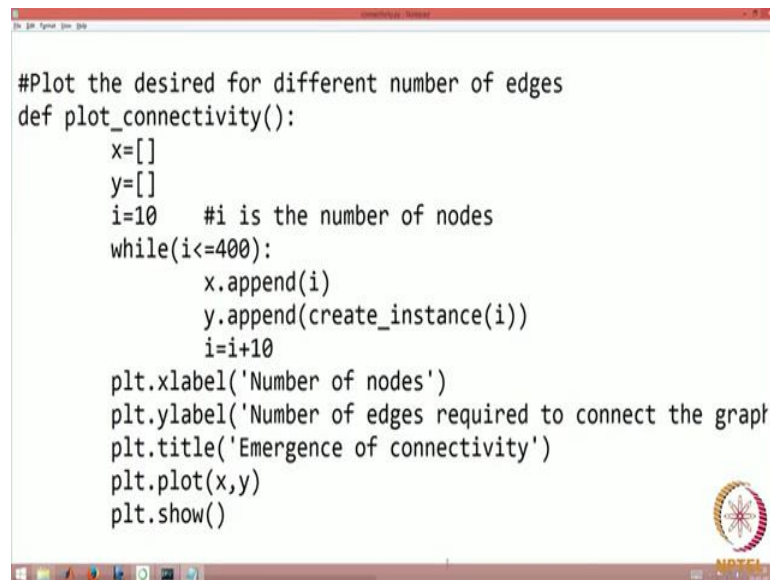
Let us make it twenty nodes we need 27 edges we have thirty nodes it is 50 edges and so on and what we are interested in I want to plot this and visualize how does this plot looks like. So, we just need to we just need to plot what we have done. So, for plotting it we go back to the file, we can define a function here. So, what is this function do plot the desired let say we know what is desired. So, desired is the number of edges required for connectivity plot the desired for different number of edges and let us define a function and let us call it plot and let us call it plot connectivity.

Let us define this function plot connectivity and we it does not require as input anything we are not taking any input for simply for different number of nodes will be plot it. So, for plotting always we need the arrays corresponding to a 2 axis; x axis and y axis.

(Refer Slide Time: 16:36)

```
#Plot the desired for different number of edges
def plot_connectivity():
    x=[]
    y=[]
    i=10    #i is the number of nodes
    while(i<=400):
        x.append(i)
        y.append(create_instance(i))
        i=i+10

    plt.xlabel('Number of nodes')
    plt.ylabel('Number of edges required to connect the graph')
    plt.title('Emergence of connectivity')
    plt.plot(x,y)
    plt.show()
```

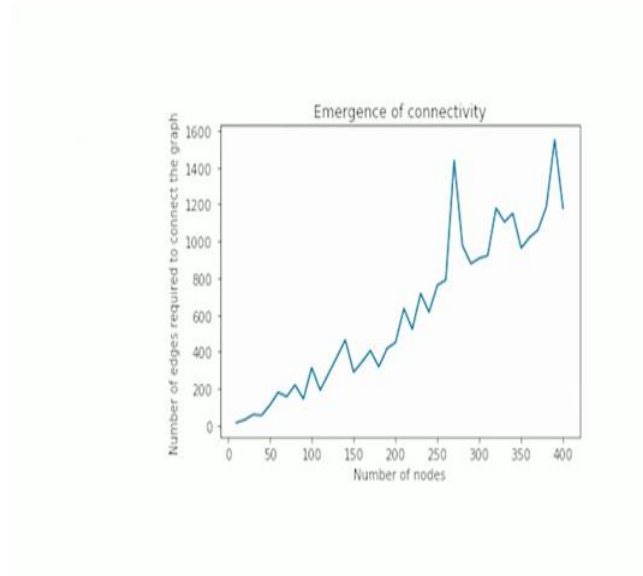


So, let us the array associated with x axis be x and the array associated with y axis be y. So, we have here x and y then we take a number $i = 10$. So, i what does i tells us i is the number of nodes for which we are experimenting. So, i is the number of nodes while and let us take the number of nodes till 100 or let us rather make it 400. So, while $i \leq 400$ what is loop will do in x axis, we append the number of nodes which is nothing, but i and in y axis we can append or do we append the number of edges required to connect these many nodes which we can get from our function create instance. So, $y.append$ create instance and we pass i here and we need let us increment i with 10 every time $i = i + 10$.

Now, we need to plot this for plotting this we need the package matplotlib, we import matplotlib.pyplot as plt come back. So, what do we do $plt.plot$ what we want to plot is x against y and $plt.show$. So, let us just do a little bit of decoration here before plotting it let us add some labels and the title for the curve let us say $plt.xlabel$ which is the label for the x axis is nothing, but the number of nodes $plt.ylabel$ label for the y axis is the number of edges required to connect the graph and $plt.title$ let us give a title to this curve and let us title it as emergence of connectivity it done.

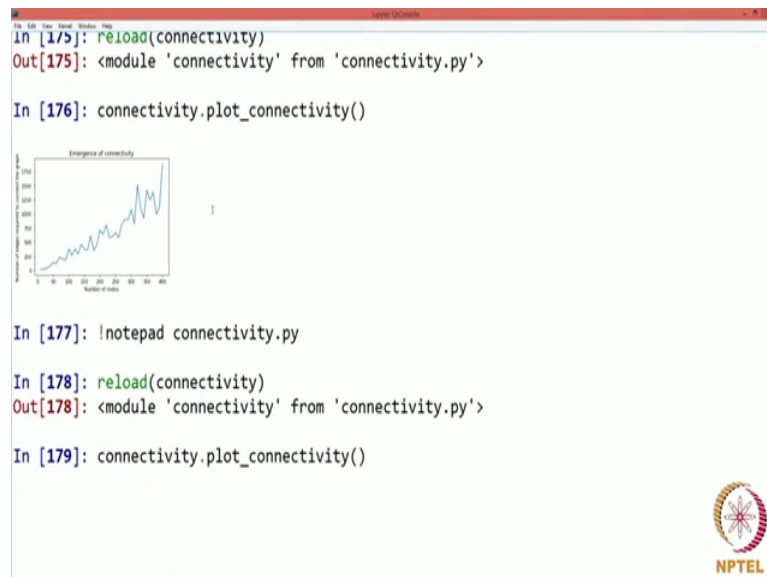
Let us look at how does this plot look like save the file we will reload it and then we call connectivity dot plot connectivity I will take some time to execute yes.

(Refer Slide Time: 19:33)



So, here we can see the curve here we have the number of nodes here we have the number of edges and we can see that as the number of nodes increases the number of edges required to connect it also increases although we see that although we see this plot where number of nodes with number of edges we see a lot of spikes in this curve.

(Refer Slide Time: 19:49)



So, why are these spike occurring it is because our experiment is a random experiment which gives a different output every time you execute it whenever you execute a random random experiment it will give you a slightly different outputs. So, one time it can throw

a value three other time 5 other time 4, but if you take the expected value it gives you a better estimate. So, to remove these spikes in the plot instead of taking one value corresponding to one instance we might want to average it out. So, let us how can we do that we go back. So, what are we doing here is corresponding to one value of I that is corresponding to one particular instance given a particular number of nodes stand here we execute this code only one time?

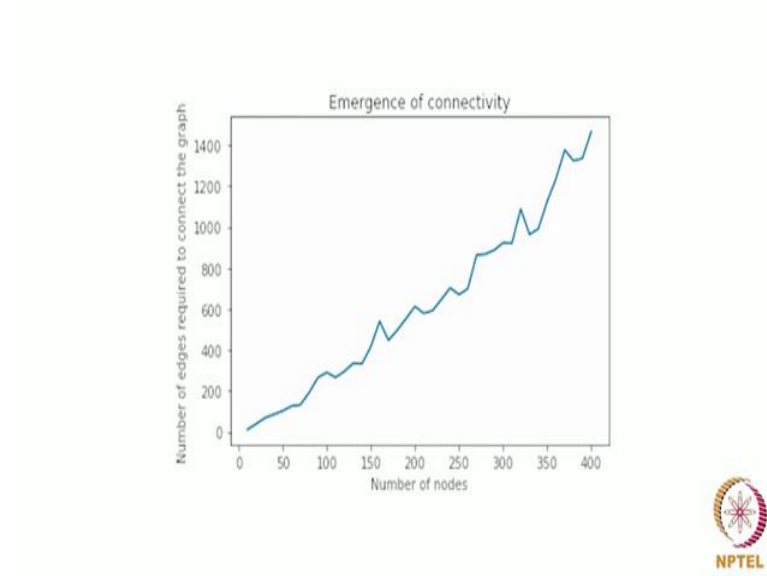
So, we take a graph having 10 nodes we will look at how many edges are required to connect it and then we return the number of edges what will give us a better result is if we do it multiple times. So, we take a graph having 10 nodes then look at the number of edges we again do this again do this do it 10 times and take an average over all those experiments that will give us a better estimate. So, let us try doing that. So, doing that is very easy. So, in this create instance. So, create underscore instances for one instance you give it one number of nodes and it gives you number of edges.

Let us average it over some particular number of instances let us average it over 4 instances. So, n see what I am going to do here is we define a new function n and let us name it create average instance n and what it does is it call the function create underscore instance 4 times and takes an average over them. So, we create a list here list one and for i in range 0 to 4. So, its repeated 4 times what does it do is list one dot append what does it append to list one is create underscore instance n. So, it appends it to list and what it returns is the average. So, it return numpy.average of list one.

So, we have yet node imported numpy. So, let us import numpy here import numpy. So, its returns numpy dot average list one. So, what we are going to do is while plotting in plotting connectivity instead of using creating underscore instance now we create we use create underscore average underscore instance. So, the value is averaged over 4 particular instances now let us look at the change in the curve it creates we reload connectivity and we again call this function.

So, since it is doing it 4 times it takes some time and you can see you can say that the difference in 2 curves is visible this curve is more smoother because we have averaged out the values over 4 instances where it was only one instance.

(Refer Slide Time: 23:21)



In this case let us play around with it a little bit further and let us change this 4 to let say 10 we do an average over 10 instances we make this 4 as 10.

(Refer Slide Time: 23:45)

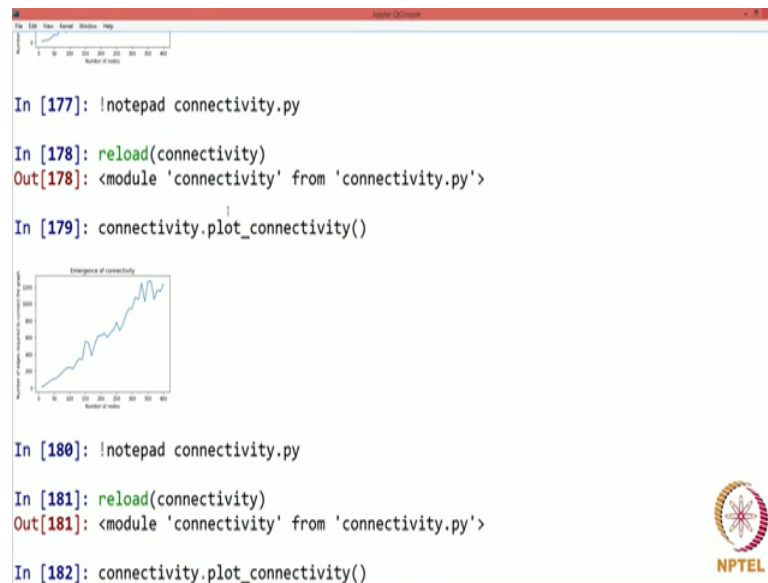
```
def create_instance(n):
    G=add_nodes(n)
    G=add_till_connectivity(G)
    return G.number_of_edges()

#Average it over 10 instances
def create_avg_instance(n):
    list1=[]
    for i in range(0,10):
        list1.append(create_instance(n))
    return numpy.average(list1)

#Plot the desired for different number of edges
def plot_connectivity():
    x=[]
    y=[]
```

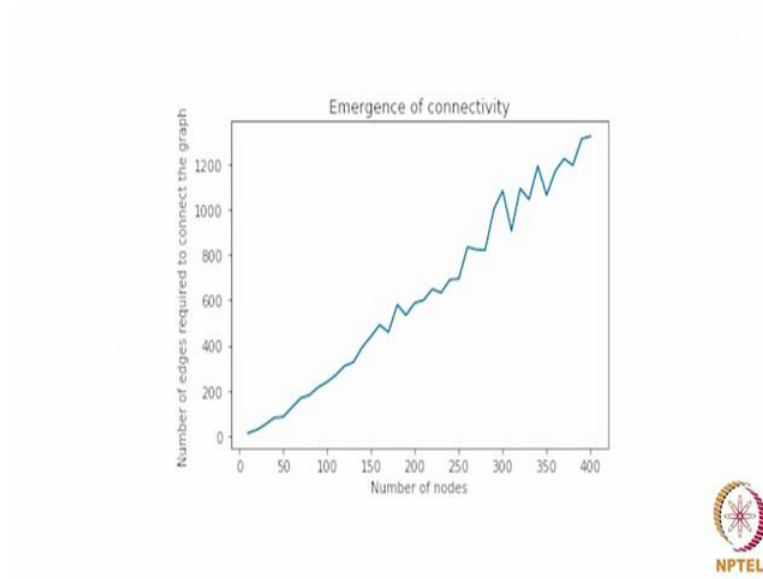
The image shows a code editor window with Python code for graph connectivity. The code defines three functions: `create_instance(n)` which adds nodes and edges until connectivity is achieved and returns the number of edges; `create_avg_instance(n)` which averages the results of 10 instances of `create_instance`; and `plot_connectivity()` which initializes lists `x` and `y` for plotting. The NPTEL logo is visible in the bottom right corner of the slide.

(Refer Slide Time: 23:52)



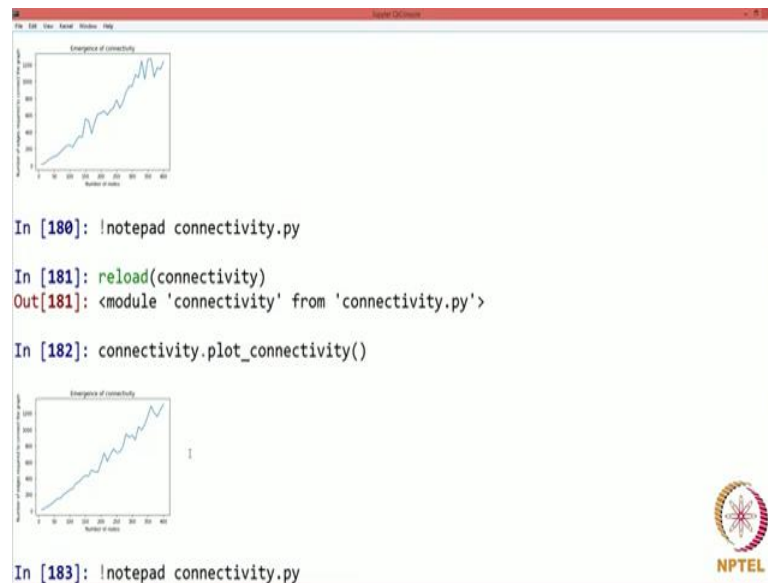
We reload it and we call the function again. So, let us wait for the output.

(Refer Slide Time: 24:01)



So, again you can see a visible change in the curve it gets all the more smooth. So, you can change this value from 10 to 20 to 3200 and the curve will become more and more smooth.

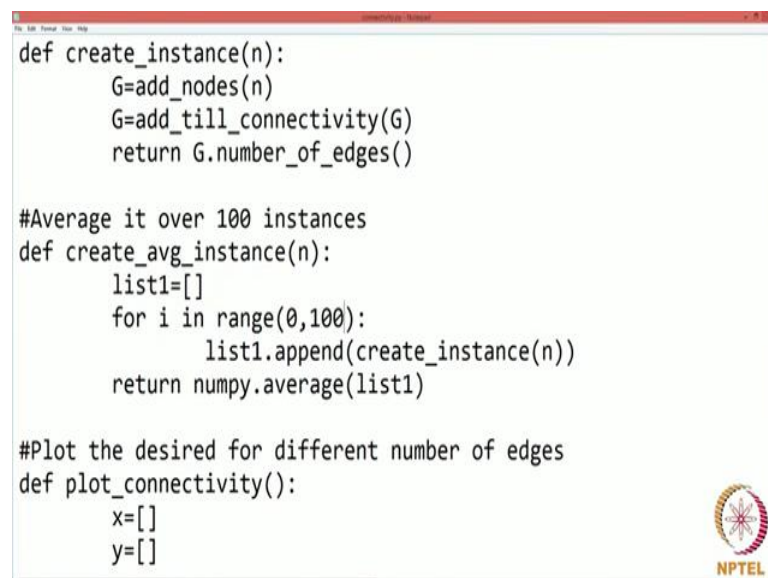
(Refer Slide Time: 24:12)



So, we can try it for higher values now the final and the biggest question what was our claim was that if there are n nodes in the network it takes just $n \log n$ number of edges to connect this network the question is are these number of edges $n \log n$. So, if I look at this curve is it the curves of $n \log n$ oh let us write a piece of code and check that as well.

So, what I am going to do is I am going to draw this plot. So, I am going to have 2 plots in the same figure one plot is the plot which we have achieved from our code which is the actual data and let us plot it against $n \log n$ and look at what is the result.

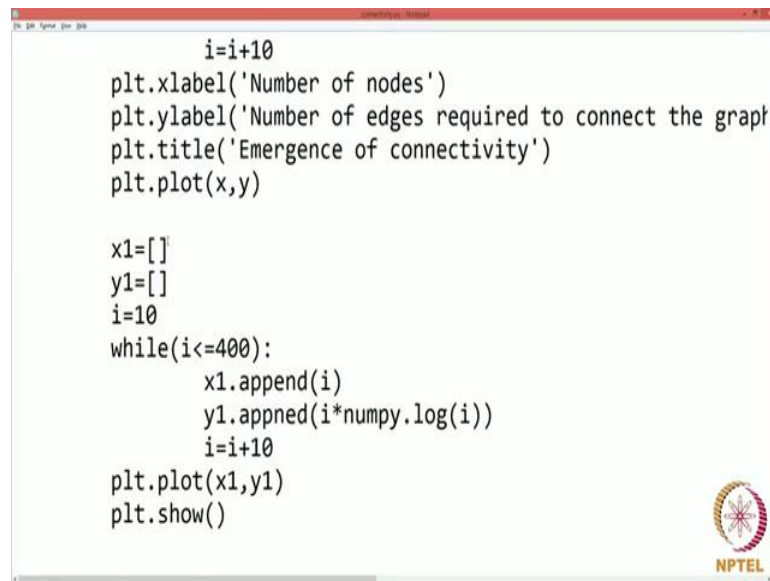
(Refer Slide Time: 24:58)



So, I want a very smooth curve. So, for a very smooth curve what I do is I average it to hundred instances. So, I average it 400 instances and now I want to plot it against the $n \log n$ curve. So, what I do is after doing this before plotting I do exactly the same procedure, which is written above, but here I am going to plot $n \log n$.

So, whatever I am going to do here is simply a replication of the above code with a little bit of change and you can write it as a separate function as well and I am writing it here.

(Refer Slide Time: 25:38)



```
i=i+10
plt.xlabel('Number of nodes')
plt.ylabel('Number of edges required to connect the graph')
plt.title('Emergence of connectivity')
plt.plot(x,y)

x1=[]
y1=[]
i=10
while(i<=400):
    x1.append(i)
    y1.append(i*numpy.log(i))
    i=i+10
plt.plot(x1,y1)
plt.show()
```

So, I take a new array $x1$ and a new array $y1$ and then again I set $i = 10$ which is the number of nodes and while $i \leq 400$ what I do is $x1.append(i)$. $y1.append$ is the curve I want and the curve which I want is $i * \log(i)$, sorry, it is $x1.append$ and it is $y1.append(i * \text{numpy.log}(i))$ and then I increment i with 10 and I do $\text{plt.plot}(x1, y1)$. So, it is going to show me 2 plots in the same figure 1 is for xy which is the actual data number of nodes and the number of edges required to connect them and second is the $x1 y1$ which tells me a number on the x axis and if the number is i , it tells me $i \log i$ on the y axis and again reload the file and let us run it.

(Refer Slide Time: 27:08)

```
In [183]: !notepad connectivity.py

In [184]: reload(connectivity)
Out[184]: <module 'connectivity' from 'connectivity.py'>

In [185]: connectivity.plot_connectivity()

AttributeErrorTraceback (most recent call last)
<ipython-input-185-3205f2213366> in <module>()
----> 1 connectivity.plot_connectivity()

C:\Users\Admin\connectivity.py in plot_connectivity()
     56     while(i<=400):
     57         x1.append(i)
--> 58         y1.appned(i*numpy.log(i))
     59         i=i+10
     60     plt.plot(x1,y1)

AttributeError: 'list' object has no attribute 'appned'

In [186]: reload(connectivity)
```

And let us give it some time to execute since it is averaging over hundred instances it will take some decent amount of time the code here ends within a error and next again a spelling mistake let us go back and correct it. So, just correct it here and what we do here is I have added an extra statement here print i which keeps showing us how many iterations has the code complete it.

(Refer Slide Time: 27:34)

```
#Plot the desired for different number of edges
def plot_connectivity():
    x=[]
    y=[]
    i=10    #i is the number of nodes
    while(i<=200):
        print i
        x.append(i)
        y.append(create_avg_instance(i))
        i=i+10

    plt.xlabel('Number of nodes')
    plt.ylabel('Number of edges required to connect the graph')
    plt.title('Emergence of connectivity')
    plt.plot(x,y)

    x1=[]
```


So, for the time being let us make this I 2 hundred since for 400 it takes a lot of time. So, let us see how it works for 200, but if we see a code below.

(Refer Slide Time: 27:59)

```
print i
x.append(i)
y.append(create_avg_instance(i))
i=i+10

plt.xlabel('Number of nodes')
plt.ylabel('Number of edges required to connect the graph')
plt.title('Emergence of connectivity')
plt.plot(x,y)

x1=[]
y1=[]
i=10
while(i<=200):
    x1.append(i)
    y1.append(i*numpy.log(i))
    i=i+10
```




So, for the second plot it is still plotting till the till 400 nodes. So, we also change this and make it here 200 and now we just save this code and let us run it.

(Refer Slide Time: 28:09)

```
In [125]: !notepad connectivity.py

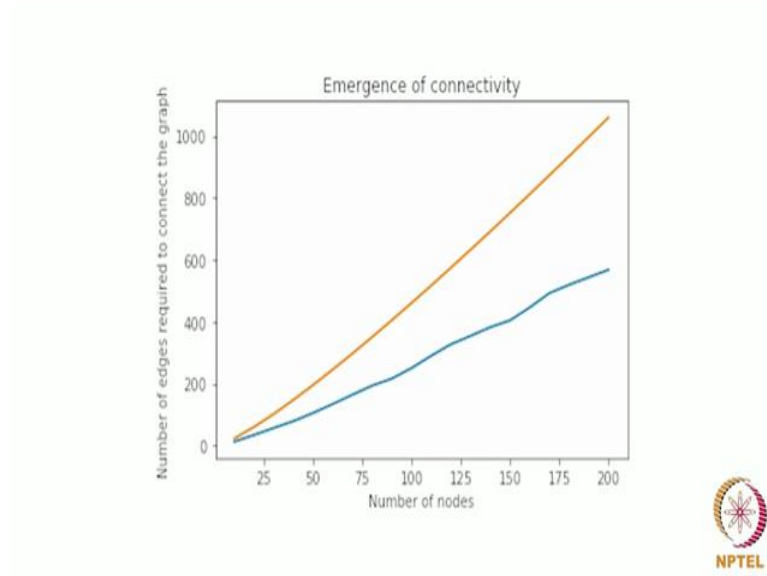
In [126]: reload(connectivity)
Out[126]: <module 'connectivity' from 'connectivity.py'>

In [127]: connectivity.plot_connectivity()
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
```



So, let us reload our file we call function connectivity.plot_connectivity. So, here you can see 2 lines in this plot one is the blue line which corresponds to our data where we took different number of nodes and looked at the number of edges required to connect the corresponding network and the red line it corresponds to the $n \log n$ plot. So, the lines are not totally overlapping in this case.

(Refer Slide Time: 28:25)



So, here I would like to remind you that according to the analysis that we have done the number of edges required to connect the graph was not exactly $n \log n$ it is of the order of $n \log n$.

So, let us do a small; a little bit of tweak in our code let us go back to a notepad file.

(Refer Slide Time: 29:06)

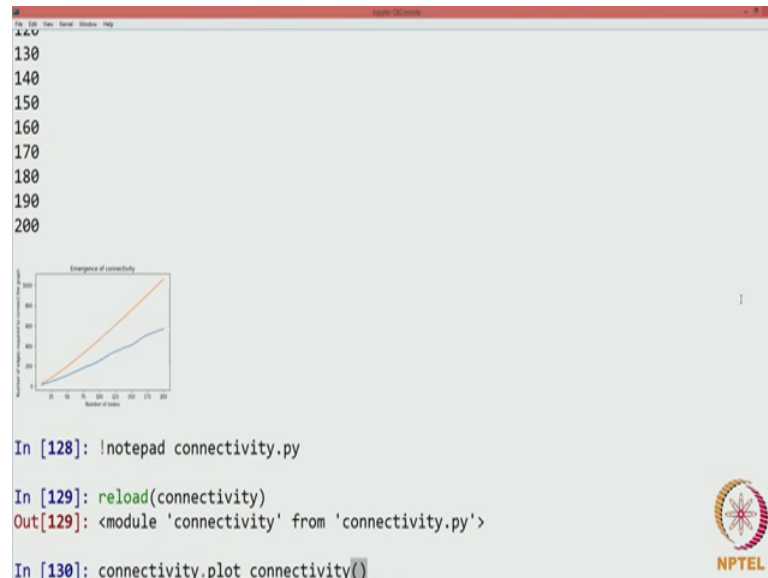
```
print i
x.append(i)
y.append(create_avg_instance(i))
i=i+10
plt.xlabel('Number of nodes')
plt.ylabel('Number of edges required to connect the graph')
plt.title('Emergence of connectivity')
plt.plot(x,y)

x1=[]
y1=[]
i=10
while(i<=200):
    x1.append(i)
    y1.append(i*float(numpy.log(i))/2)
    i=i+10
```

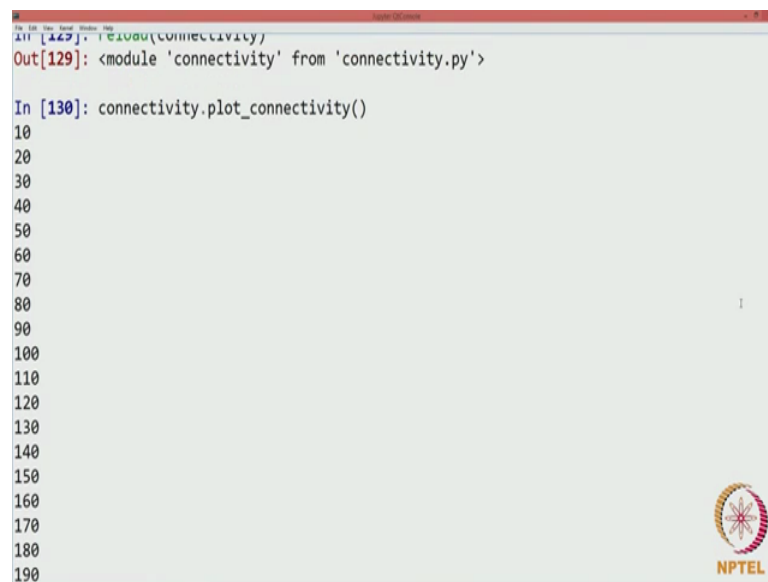
And what I am going to do here is instead of plotting $n \log n$ what I plot here is $n \log n$ divided by two. So, please note that even if I divide this term $n \log n$ by 2 in the final

analysis the number of edges required to connect the network still remains of the order of $n \log n$. So, let us try doing that. So, what I do here is I divide the term $\log n$ by 2.

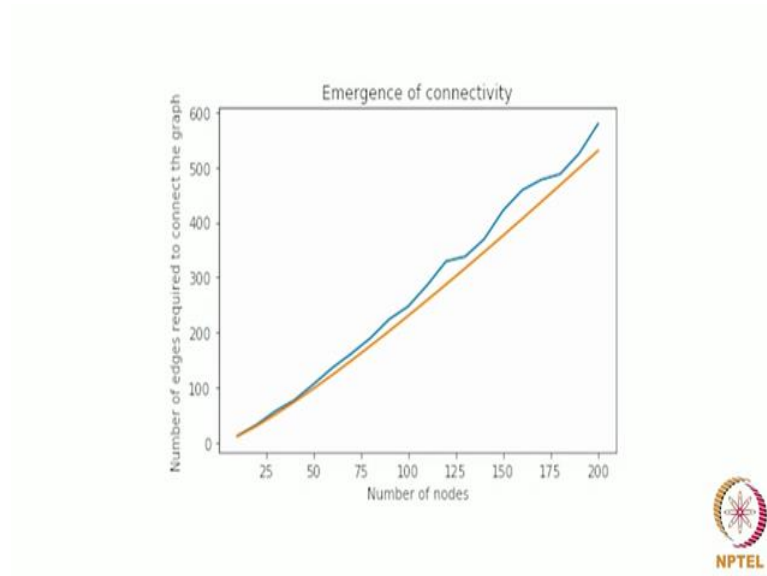
(Refer Slide Time: 29:35)



(Refer Slide Time: 29:38)



(Refer Slide Time: 29:46)



And let us see how our plot changes let us plot it. So, here you can see that both the curves become very close to each other establishing the fact that it is essentially $\frac{n \log n}{2}$ number of edges almost of the order of $n \log n$ which are required to make our graph connecting hence satisfying a previous claim.