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

Link Analysis (Continued)
Lecture – 110
Convergence in Repeated Matrix Multiplication
(Pre-requisite 1)

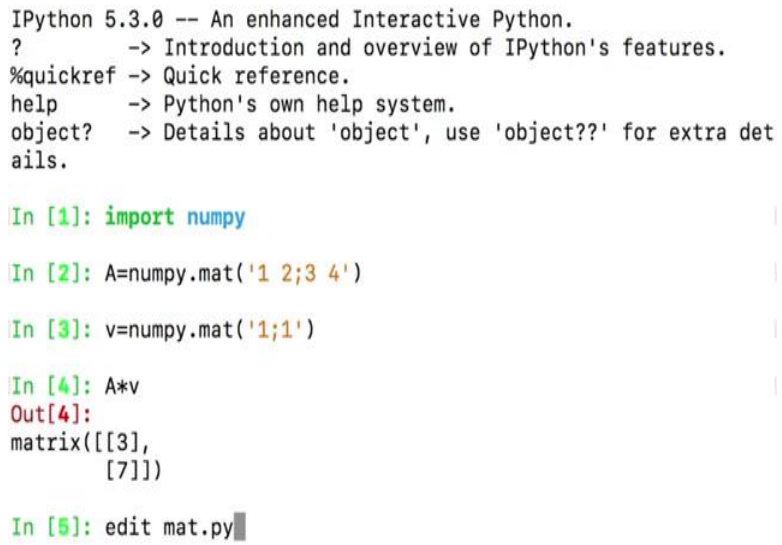
(Refer Slide Time: 00:05)

```
Sudarshans-Air:~ sudarshaniyengar$ ipython
Python 2.7.13 |Anaconda 4.4.0 (x86_64)| (default, Dec 20 2016, 23
:05:08)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra det
ails.

In [1]: import numpy

In [2]: A=numpy.mat('1 2;3 4')

In [3]: v=numpy.mat('1;1')

In [4]: A*
```

Let me now open our ipython shell and let me try coding whatever I observed just now . So, before which I will try to help you understand a few commands, there is something called the numpy library which is a storehouse of a lot of matrix operations functions which help you meddle with matrices. So, how do you declare a matrix? You simply say A equals numpy.mat and then you write down your matrix in the form of the 1st row followed by a semicolon and then the 2nd row.

If you remember it right from a previous lecture we are taking this matrix $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$ and then there is a new vector v that I defined which is if you remember [1, 1], that is defined in python as 1 space 1, no that is not right 1 semicolon and then 1 isn't that right. So, what is it denote, I am sorry what is it denote? 1 is the first row, 1 is the second row right. So, when I say A times v now A is a matrix $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$ and v is the matrix $(1\ 1)$.

(Refer Slide Time: 01:16)



I get [[3], [7]] which is 1 + 2 and then 3 + 4 correct that is the way in which you multiply a matrix with the vector. So, let me now open a new shell ipython shell and then type a piece of code, to observe what happens when you keep repeatedly applying A on the vector v of course, you keep normalizing it every now and then ok.

(Refer Slide Time: 01:50)

So, let me edit mat.py and what I do is I say import numpy and then I will declare my matrix which is A = numpy.mat (‘1;2 ; 3 4’)the close and then my initial vector is going to be numpy.mat (‘1; 1’). And now what I am doing is, for i in range 10 you know what I

am doing here; I say z = A * v right and then I need to normalize this vector right. Do you remember $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$supplication on [1, 1] gave us some value and we pulled that to the unit circle? That pulling to the unit circle is done by the following operation, what you do is z = z/numpy.linalg this is a linear algebra package that is available within numpy you say norm(z).

So, this one is going to be a real number which is basically that what you get. So, 1 plus 2 is 3 3 + 4 is 7 this becomes [[3], [7]] when you apply this matrix on [1, 1]. And, [[3], [7]] you normalize it what do you do 3 divided by 3, 7 divided by square root of 3 square + 7 squared right. Remember in the previous lecture I am just doing the same thing here.

After doing this what I do is, I print my z and I print some stars here. So, that it looks neat it separates two things and then I go back here, and I simply display my v for clarity sake; just to tell you from where we started from correct. And, this is just beautification do not worry much this is just for us to read the output clearly nothing else ok.

So, I am going to say this if you want let us go through the code once import numpy that imports all the library functions, that numpy has and I declare a matrix $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$. And, then the next line is I declare this column matrix [1, 1]. I simply display what is this matrix [1, 1] here. And, then I repeat this loop 10 times what do I do, I apply my matrix A on v and call it z and then I normalize z. What do you mean by normalizing z? You pull it to the unit circle right, that is what this does linear algebra norm precisely does that square root of sum of squares nothing right. So, now let us say you print z.

(Refer Slide Time: 04:47)

```
ails.

In [1]: import numpy

In [2]: A=numpy.mat('1 2;3 4')

In [3]: v=numpy.mat('1;1')

In [4]: A*v
Out[4]:
matrix([[3],
        [7]])

In [5]: edit mat.py
Editing... /Users/sudarshaniyengar/anaconda/lib/python2.7/site-pa
ckages/IPython/core/magics/code.py:714: UserWarning: Could not op
en editor
  warn('Could not open editor')

In [6]: run mat.py
```

Let us see what the output of this is, forget the warning message that you keep getting like this that generally happens because of the it is relation problems. And, you would not output the write editor and stuff like that anyway let us ignore it.

(Refer Slide Time: 05:03)

```
In [6]: run mat.py
[[1]
 [1]]
####################
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************
```

So, what I do is run mat.py and I get some answer here yes. So, I start with [1, 1], it is application whose application? The matrix $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$application on [1, 1] gives me [[3], [7]]. When I normalize [[3], [7]], if you remember from my previous lecture, I indeed got

0.39 and 0.91 there. The question there was if I repeatedly apply what do I get? I am getting the same values as you can see.

(Refer Slide Time: 05:32)

```
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************
[[ 0.3939193 ]
 [ 0.91914503]]
***************

In [7]:
```

It is simply 0.3939193 and 0.91914503 I hope there is no error with the code, let me just check the code.

(Refer Slide Time: 05:38)

```
 1 import numpy
 2
 3 A=numpy.mat('1 2;3 4')
 4 v=numpy.mat('1;1')
 5 print v
 6 print "####################"
 7 for i in range(10):
 8     z=A*v
 9     z=z/numpy.linalg.norm(z)
10     print z
11     v=z
12     print "***************"
~
~
~
~
~
~
~
```

So, there is a problem here indeed this is the reason why I think it is good for us to code together and if there is an error I will not redo the coding I will correct it here; so, that you will realize what kind of mistakes one can do. So, I have made a very silly mistake here, I

am simply taking z and then applying assigning A of v we do it right. What I should do is, if I want repeated application after displaying z I should say v = z.

So, that when I come here, I am sorry when I come here the next z will be A applied to my previous z which was v right. So, the output that I got previously was incorrect let us redo this. So, let me say run mat.py and I get some values here, let me see what this is 0.39 0.91 as expected and then the next one is something else. You see 4175 something the next one is 4158 something, 4159 something, 415973, but this was 415980 so on and so forth you will observe that oh my goodness it is converging. 0.41597356 here the next iteration the 10th iteration was 41597356 precisely the same as the previous iteration, you will even observe it with this.

So, these two column vectors sort of are the same, maybe this was this converged. What if, let me do something what if what was that mat.py and so, edit mat.py and I will go here and I will try to vary this [1, 1] to let us say 1 comma let us say let us say some 3 comma sometime 11 alright. Now, let us see what happens to this, I come out the code remains the same I come out and I execute this; you see what happens looks like I am getting the same values how is that.

(Refer Slide Time: 07:45)

```
In [9]: edit mul.py
Editing...
In [10]: edit mat.py
Editing...
In [11]: run mat.py
[[ 3]
 [11]]
####################
[[ 0.42661868]
 [ 0.90443159]]
***************
[[ 0.41523533]
 [ 0.90971403]]
***************
[[ 0.41602471]
 [ 0.90935331]]
***************
[[ 0.41597001]
 [ 0.90937833]]
***************
```

So, [[3], [11]] gave me this value after normalizing and application of the matrix A on this gives me this. An application of my matrix A on this gives me this right you see that finally; I am getting 41597356 and 90937671 this looks familiar. Why? Let me scroll up

and see do you see that this is precisely the value that we got before, 41599093 come down 41599093 it is exactly the same.

What do we conclude here? No, matter where you start from what vector you start from, when you start applying the same matrix 1 2 3 4 on any vector, you get a constant vector. It converges to a constant vector.