Introduction to High Performance Computing

Autumn, 2017

Lecture 5

Git/Bitbucket notes

- After modifying files in your local repo, remember to *add*, *commit*, and *push* your changes
- Before starting work which will modify files in your local repo, remember to sync your fork and *pull* any changes into your local repo.
- This will help you avoid merging problems which occur when your fork and local repo are both being updated independently
- There is a link to a nice, short online tutorial on using git in the supplementary reading section of the course webpage.

Python notes

Main differences between arrays and lists:

Lists are *flexible*: heterogeneous data, can grow or shrink, numerical calculations can be slow/cumbersome

Arrays: calculations are generally faster, but elements must be homogeneous, difficult to adjust size

Python notes

Getting comfortable with python:

- **Have command of *all* of the material in online lectures –use exercises for self-assessment (solutions are online)**
- **Understand structure and purpose of functions (lecture 3 slides)**
- **Choose an editor + terminal combination for developing code. Can be *spyder* (distributed w/ anaconda), *canopy*, or *atom* + *jupyter qtconsole*. Use python3.x (e.g. python3.6)**
- **Understand *mysqrt.py* and *brown.py* (provided in lecture 4 and 5 directories of course repo)**
- **Further help: list of supplementary material on course webpage, office hours**

Today

- Improving Brownian motion code
- Basics of network science
- Using *networkx* package in python

Code development

- **Use built-in functions whenever possible (*e.g.* sqrt instead of mysqrt.sqrt2)**

- **When working with arrays, *vectorize* code whenever possible:**

```
In [8]: def sinx(x):
   ...: y = np.zeros_like(x)
   ...: for i,xi in enumerate(x):
   ...: y[i] = sin(xi)
   ...: return y
   ...:
In [9]: x = np.random.randn(10000)

In [10]: timeit y1=sin(x) #vectorized
159 µs ± 989 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)

In [11]: timeit y2 = sinx(x)
18.6 ms ± 140 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Networks

Examples of significant networks include:

- **Social networks**
- **World-wide web**
- **Internet**
- **Air transportation network**
- **Cellular network**

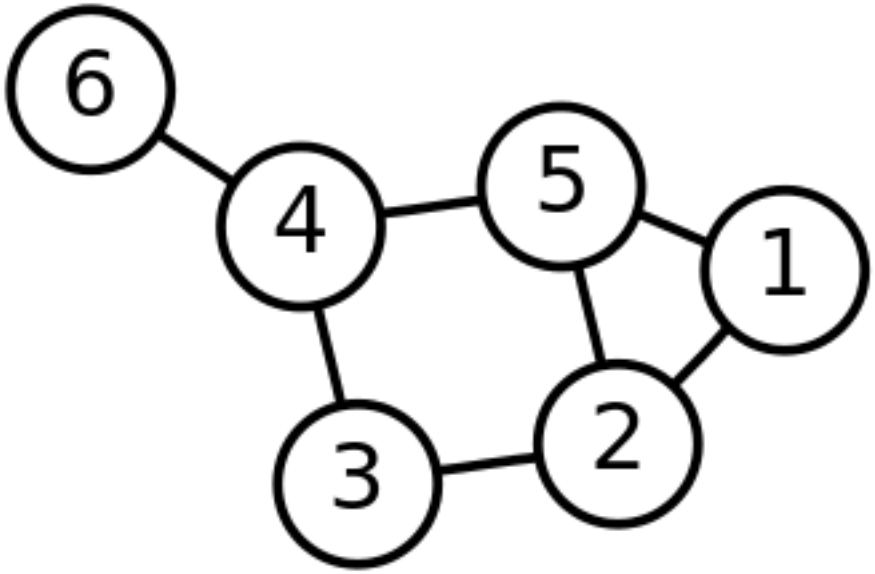
The science of networks is an important, rapidly growing field

Networks: basics

- **A network has** N ***nodes*** **and** L **links between nodes**
- **Each node has a label, e.g.** 1, 2, …, N
- **Then a link between node i and j can be represented simply as (i, j)**

Networks: basics

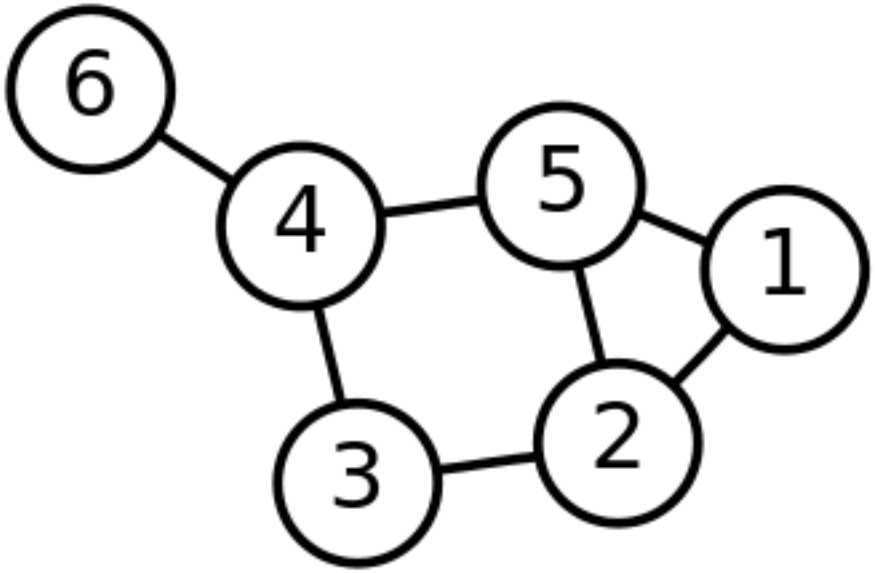
- **A network has** N ***nodes*** **and** L **links between nodes**
- **Each node has a label, e.g.** 1, 2, …, N
- **Then a link between node i and j can be represented simply as (i, j)**



Example: 6 nodes, 7 links
Node one has two edges: (1,2) and (1,5)

The graph can be represented by the *adjacency matrix*, A
A_{ij}=1 if there is link between nodes i and j

Networks: basics



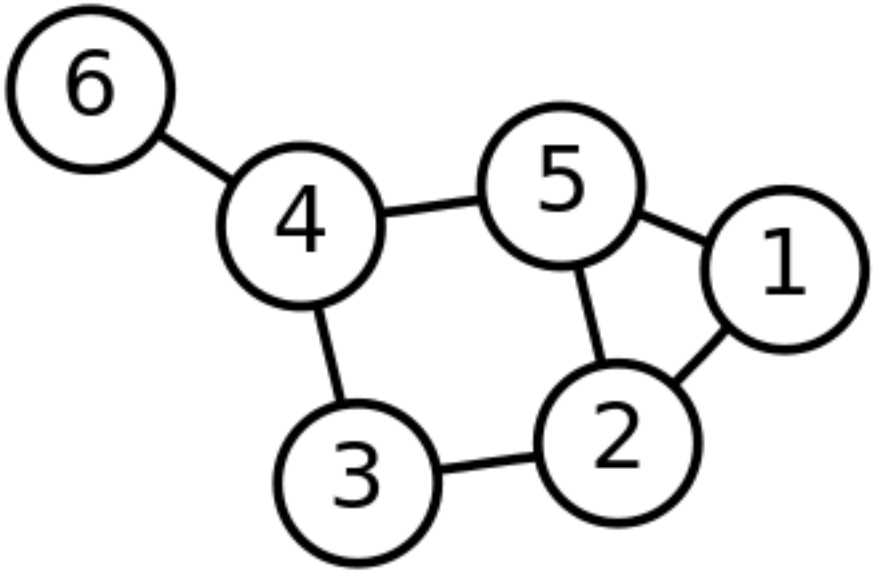
Example: 6 nodes, 7 links
Node one has two links: (1,2) and (1,5)

The graph can be represented by the *adjacency matrix*, A
A_{ij}=1 if there is link between nodes i and j

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

A is symmetric.

Networks: basics



Example: 6 nodes, 7 links
Node one has two links: (1,2) and (1,5)

The graph can be represented by the *adjacency matrix*, A
A_{ij}=1 if there is link between nodes i and j

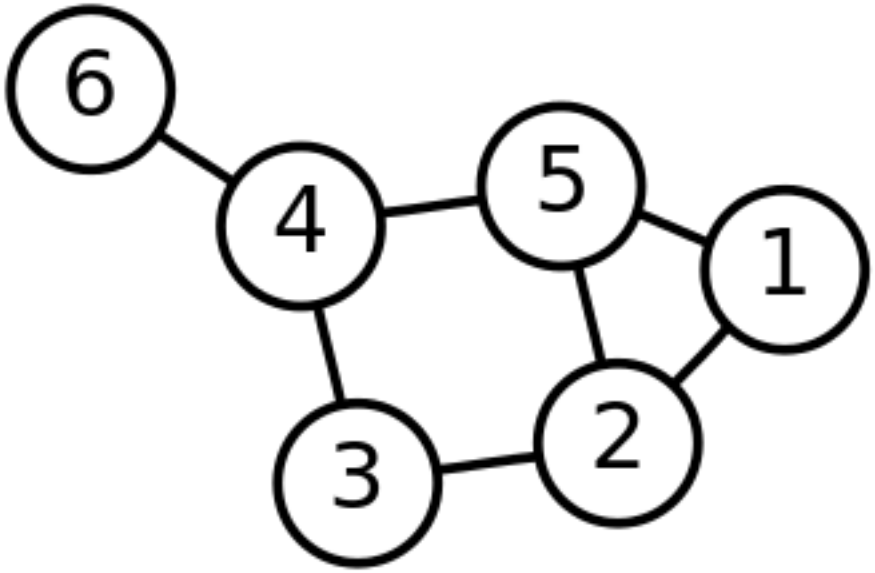
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Can also represent connected portions of graph with edge list:

$$\begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 4 & 4 \\ 2 & 5 & 3 & 5 & 4 & 5 & 6 \end{bmatrix}$$

A is symmetric.

Networks: basics



The *degree* of a node is the the total number of links connected to it:
$q_1 = 2$, $q_5 = 3$, …

The *degree distribution*, P(q) is particularly important. P(q) is the fraction of nodes in the graph with degree = q

P(1) = 1/6, P(2) = 2/6, P(3) = 3/6

Networks: basics

Network	Nodes	Links	Directed / Undirected	N	L	‹K›
Internet	Routers	Internet connections	Undirected	192,244	609,066	6.34
WWW	Webpages	Links	Directed	325,729	1,497,134	4.60
Power Grid	Power plants, transformers	Cables	Undirected	4,941	6,594	2.67
Mobile-Phone Calls	Subscribers	Calls	Directed	36,595	91,826	2.51
Email	Email addresses	Emails	Directed	57,194	103,731	1.81
Science Collaboration	Scientists	Co-authorships	Undirected	23,133	93,437	8.08
Actor Network	Actors	Co-acting	Undirected	702,388	29,397,908	83.71
Citation Network	Papers	Citations	Directed	449,673	4,689,479	10.43
E. Coli Metabolism	Metabolites	Chemical reactions	Directed	1,039	5,802	5.58
Protein Interactions	Proteins	Binding interactions	Undirected	2,018	2,930	2.90

Table 2.1

Canonical Network Maps

The basic characteristics of ten networks used throughout this book to illustrate the tools of network science. The table lists the nature of their nodes and links, indicating if links are directed or undirected, the number of nodes (N) and links (L), and the average degree for each network. For directed networks the average degree shown is the average in- or out-degrees $\langle k \rangle = \langle k_{in} \rangle = \langle k_{out} \rangle$ (see Equation (2.5)).

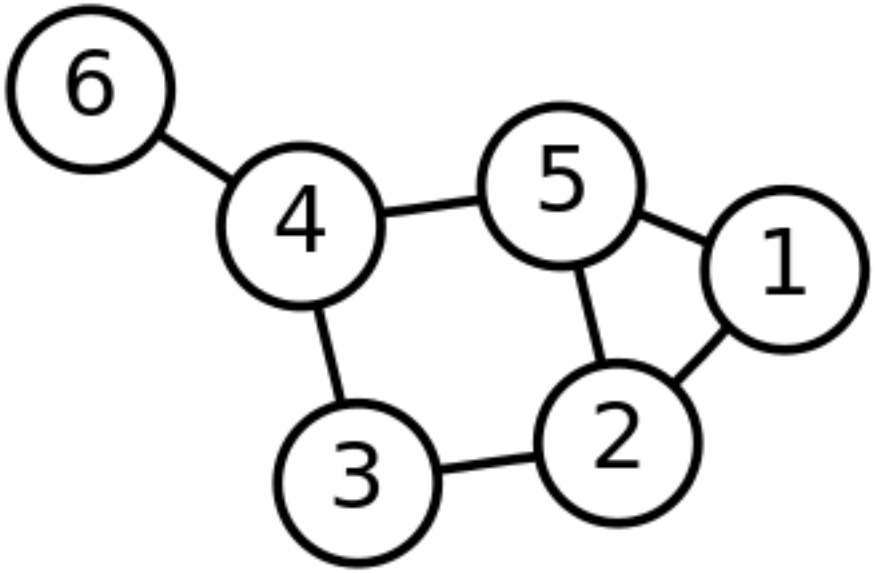
Generally interested in large *complex* networks

Analysis can be complicated and expensive (classical example: computing shortest path between nodes)

Networkx package provides a suite of tools for working with complex networks

More generally: avoid writing own code whever possible! Many powerful highly-efficient libraries are available

Networkx: basics

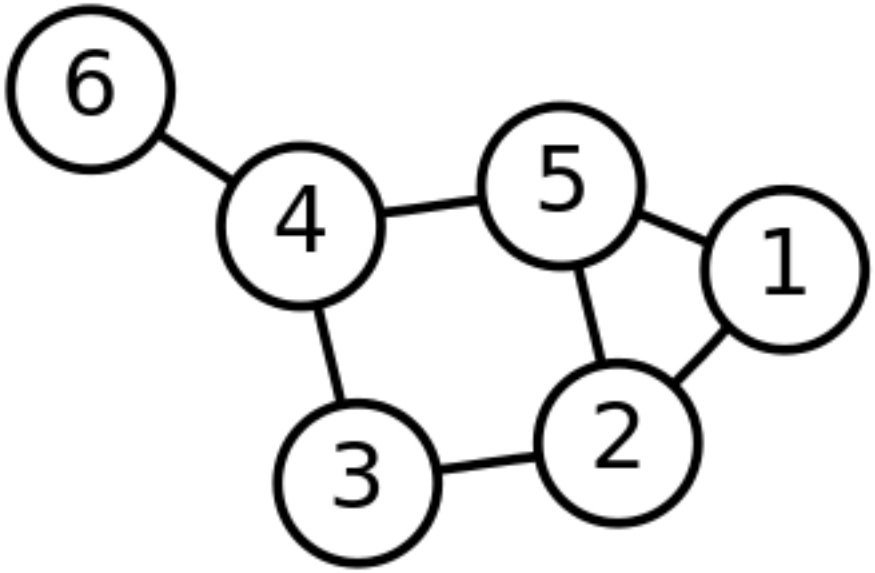


- **Let's work with this network in networkx**

First, import the module, and initialize a graph:

```
In [55]: import networkx as nx

In [56]: G = nx.Graph()
```

Networkx: basics



- **Let's work with this network in networkx**

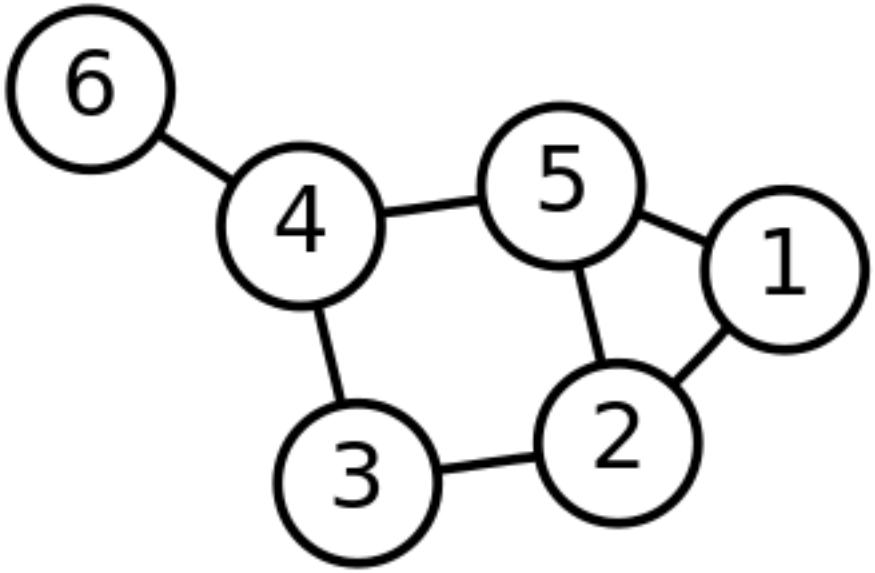
- **First, import the module, and initialize a graph:**

```
In [55]: import networkx as nx

In [56]: G = nx.Graph()
```

- **There are numerous methods for building a graph**

```
In [57]: G.add_edge(1,2)

In [58]: G.edges()
Out[58]: [(1, 2)]

In [59]: G.nodes()
Out[59]: [1, 2]
```

Networkx: basics



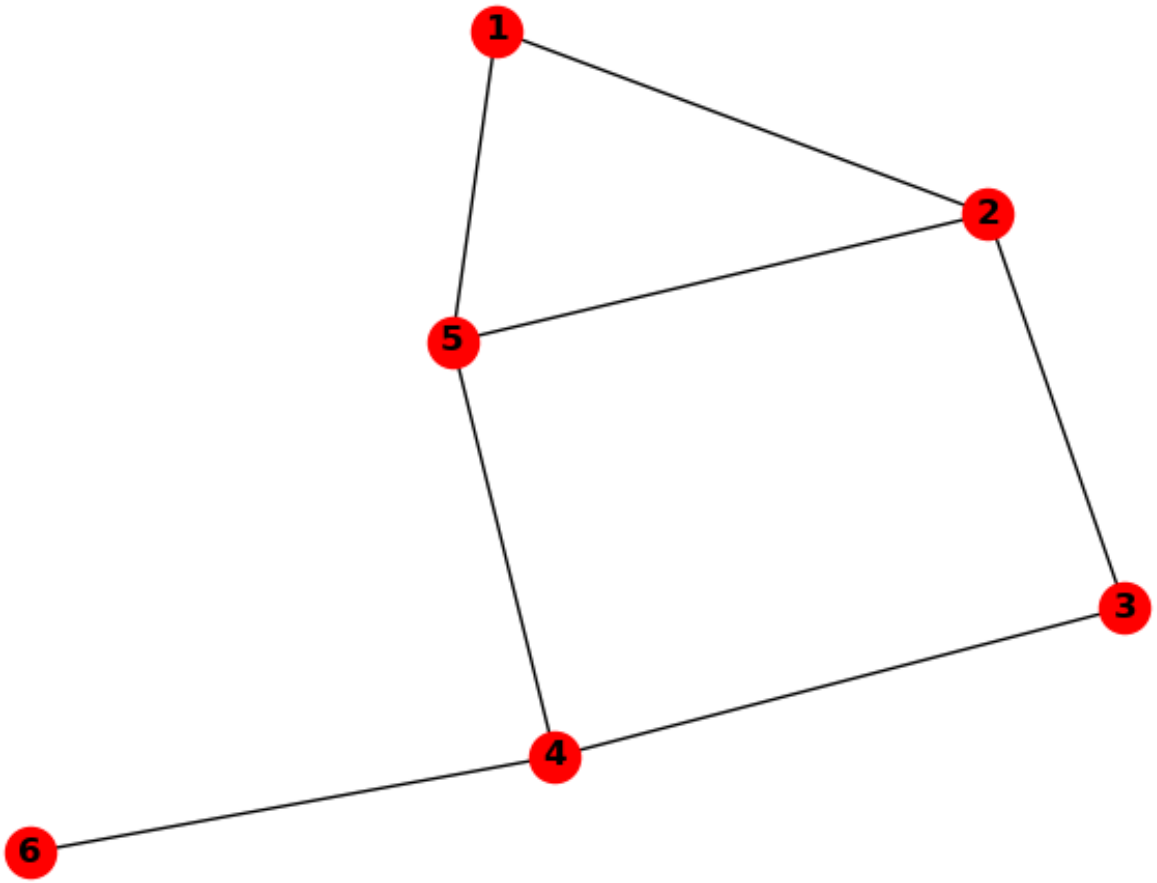
Can add several edges (or nodes) at once:

```
In [65]: e = [(1,5),(2,5),(2,3),(3,4),(4,5),(4,6)]

In [66]: G.add_edges_from(e)

In [67]: G.edges()
Out[67]: [(1, 2), (1, 5), (2, 3), (2, 5), (5, 4), (3, 4), (4, 6)]

In [68]: G.nodes()
Out[68]: [1, 2, 5, 3, 4, 6]
```

Networkx: basics

Use nx.draw **to visualize the network:**

```
In [69]: figure()
Out[69]: <matplotlib.figure.Figure at 0x1515e3fef0>

In [70]: nx.draw(G, with_labels=True, font_weight='bold')
```



Networkx: basics

Can now analyze our graph:

```
In [74]: A = nx.adjacency_matrix(G)


In [75]: type(A)
Out[75]: scipy.sparse.csr.csr_matrix


In [76]: A.todense()
Out[76]:
matrix([[0, 1, 1, 0, 0, 0],
[1, 0, 1, 1, 0, 0],
[1, 1, 0, 0, 1, 0],
[0, 1, 0, 0, 1, 0],
[0, 0, 1, 1, 0, 1],
[0, 0, 0, 0, 1, 0]], dtype=int64)
```

Networkx: basics

Can now analyze our graph:

```
In [78]: G.adjacency_list()
Out[78]: [[2, 5], [1, 3, 5], [1, 4, 2], [2, 4], [3, 5, 6], [4]]

In [79]: G.nodes()
Out[79]: [1, 2, 5, 3, 4, 6]
```

- **Adjacency list representation is much more efficient for sparse networks!**
- **Most complex networks are sparse**

Networkx: basics

Can now analyze our graph:

```
In [83]: nx.degree_histogram?
Signature: nx.degree_histogram(G)
Docstring:
Return a list of the frequency of each degree value.
Returns
-------
hist : list
A list of frequencies of degrees.
The degree values are the index in the list.

In [84]: h = nx.degree_histogram(G)

In [85]: h
Out[85]: [0, 1, 2, 3]
```

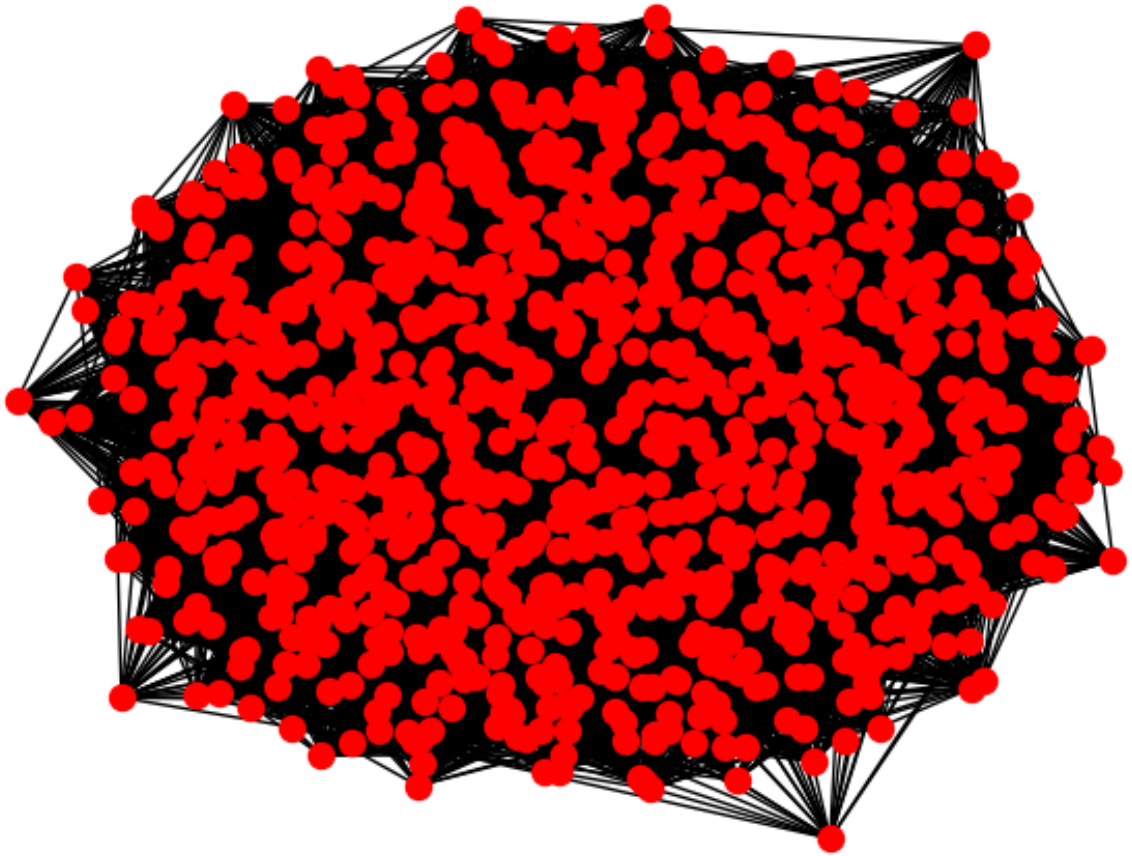
- **Graph has one degree-1 node, two degree-2 nodes, and three degree-3 nodes**

- **Degree distribution is more interesting for large networks:**

Networkx: basics

Erods-Renyi network N nodes, a link is placed between a pair of nodes with probability p:

```
In [119]: Grandom = nx.gnp_random_graph(1000,0.05)

In [120]: nx.draw(Grandom,node_shape='.')
```

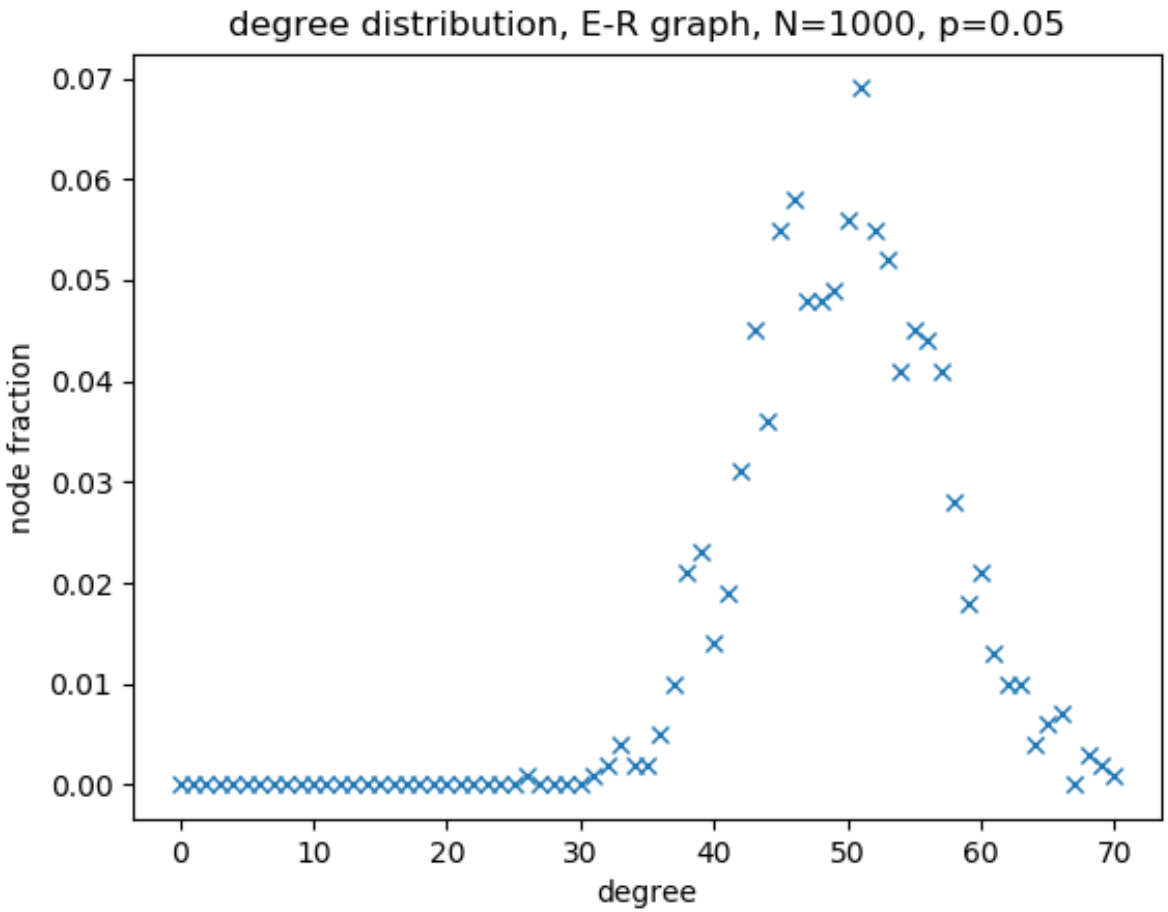


Networkx: basics

Erods-Renyi network N nodes, a link is placed between a pair of nodes with probability p

Degree distribution follows the binomial distribution

```
In [119]: Grandom = nx.gnp_random_graph(1000,0.05)

In [120]: nx.draw(Grandom,node_shape='.')
```



Networkx: getting started

- Read the online tutorial: https://networkx.github.io/documentation/stable/tutorial.html
- Browse through the online reference section: https://networkx.github.io/documentation/stable/reference/index.html
- Try out one or two graph generators