Erdös-Rényi Networks (ER)

G(N,K): random graph with N nodes and K edges.

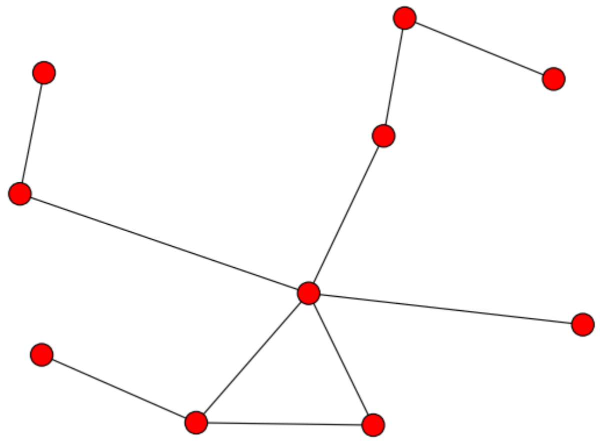
Implemented in the file ER1.py.

Usage: python ER1 n-nodes M-edges.

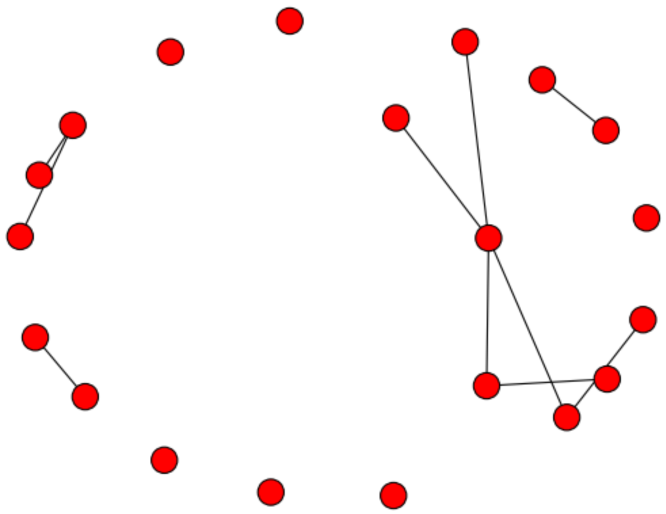
The main algorithm for this method is:

```
//while the actual number of edges < desired number
while (G.number_of_edges() < numedges):

   //calculate two random numbers – nsource and ndst
   nsource = np.random.choice(numnodes)
   ndst = np.random.choice(numnodes)

   //trace an edge between them
   if nsource <> ndst:
        G.add_edge(nsource, ndst)
```

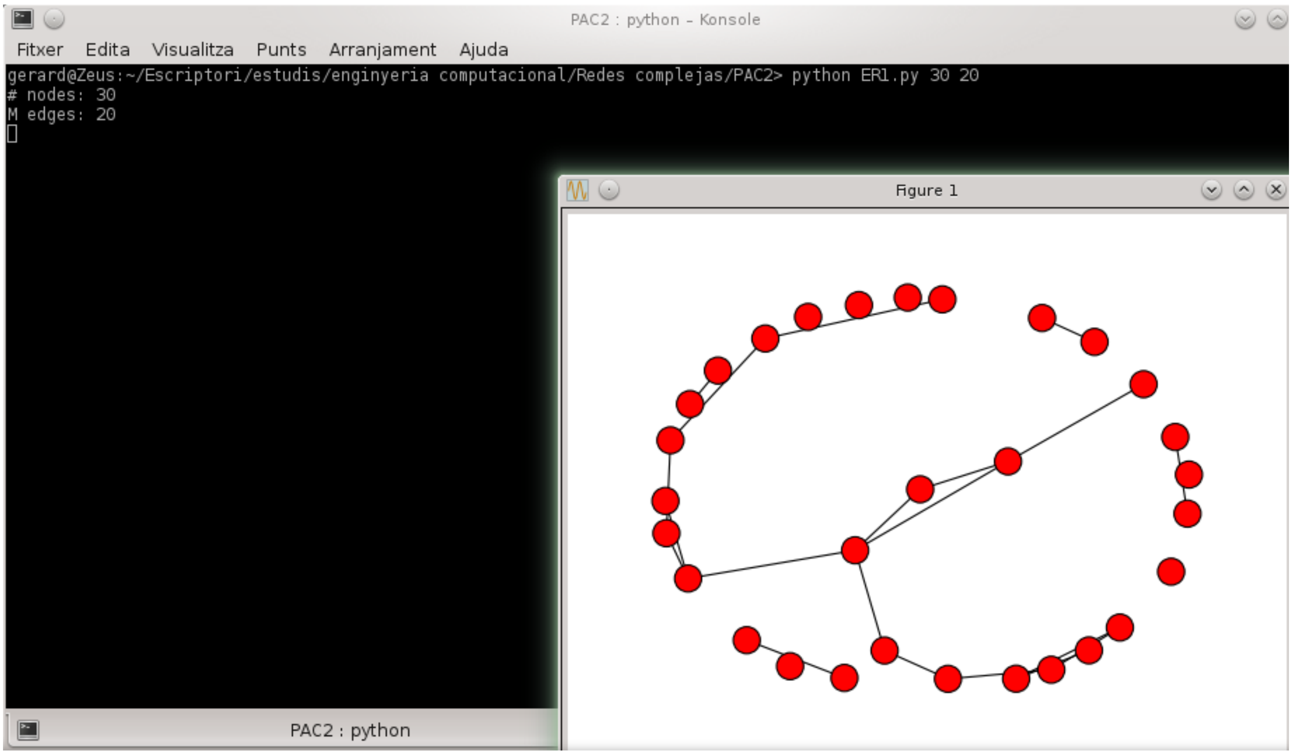
Here we have some samples:



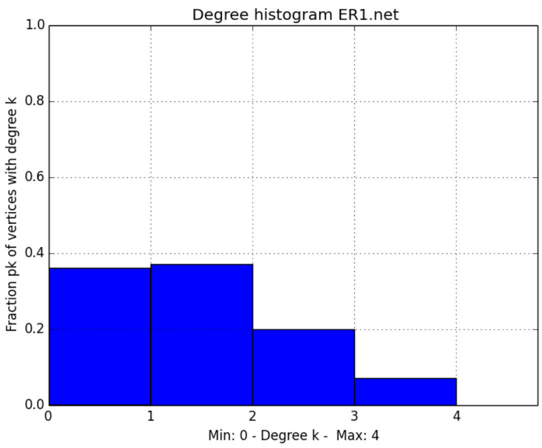
n=10 and k=10



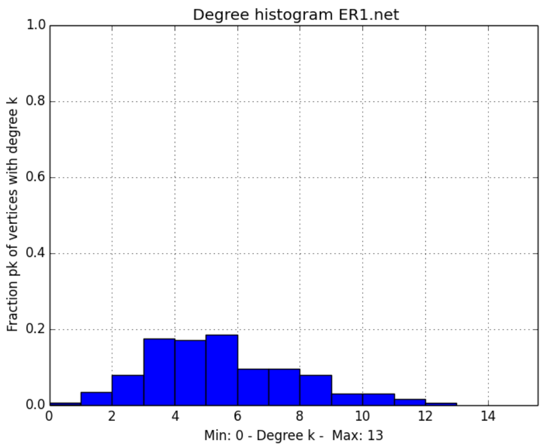
n=20 and k=10



n=30 and k=20. In this picture, you can see the python execution interface



Degree distribution of an ER network with 200 nodes and 100 edges



Degree distribution of an ER network with 200 nodes and 500 edges

G(N,p): random graph with N nodes and p probability to have an edge between two edges.

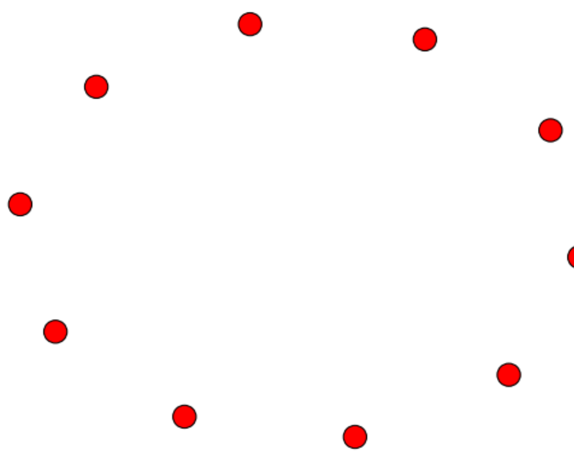
Implemented in the file ER2.py.

Usage: python ER2 n-nodes p-probability.

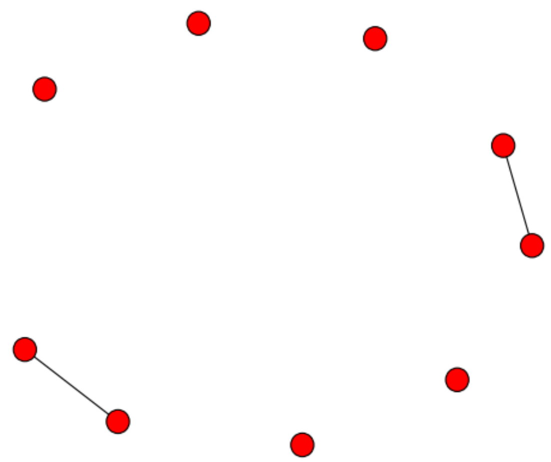
The main algorithm for this method is:

```
for j in range(0, numnodes):
   //The algorithm evaluates every posible edge
   print 'Evaluating edges on node', j
   for k in range(j+1, numnodes):

        //We calculate 0 and 1 randomly with an asociated probability of prob
        r=np.random.choice([0,1],1,p=[1-prob,prob])[0]
        if r==1:
          G.add_edge(j,k)
```

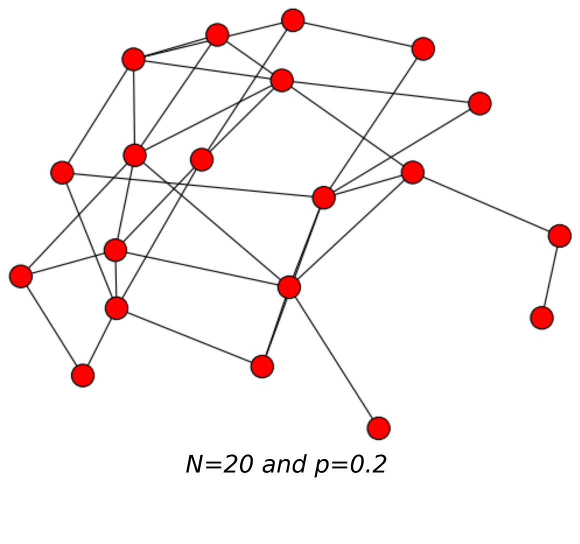
Here we have some samples:



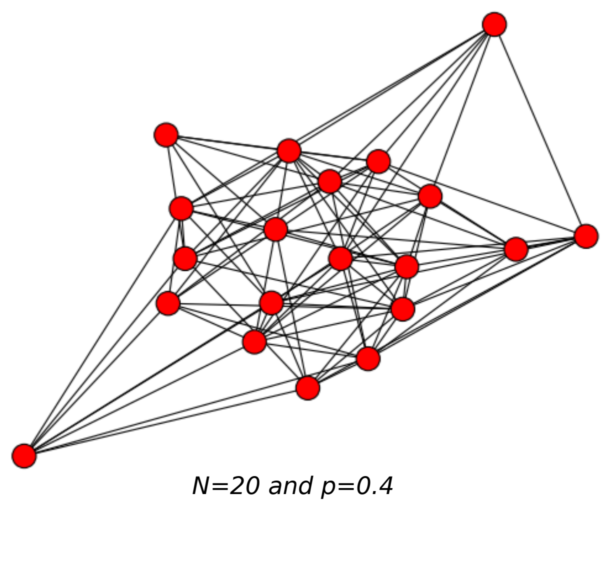
N=10 and p=0.00. No probability to have any edge



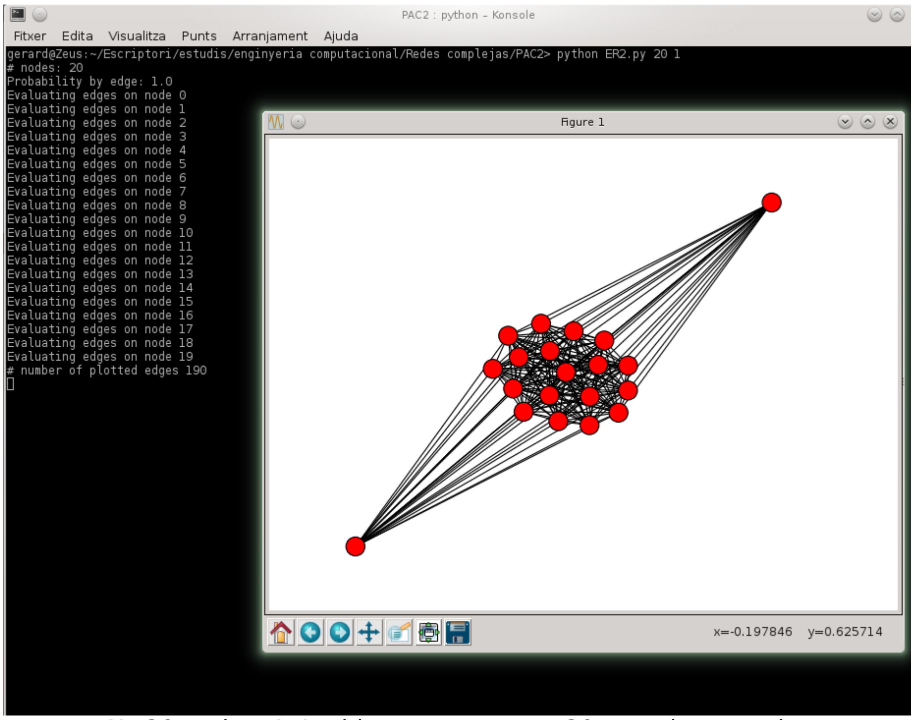
N=10 and p=0.1



N=20 and p=0.2

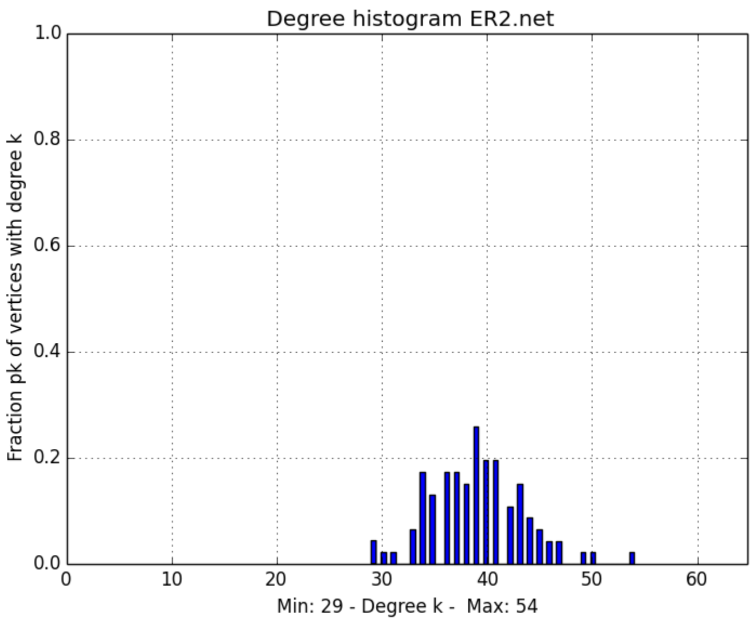


N=20 and p=0.4

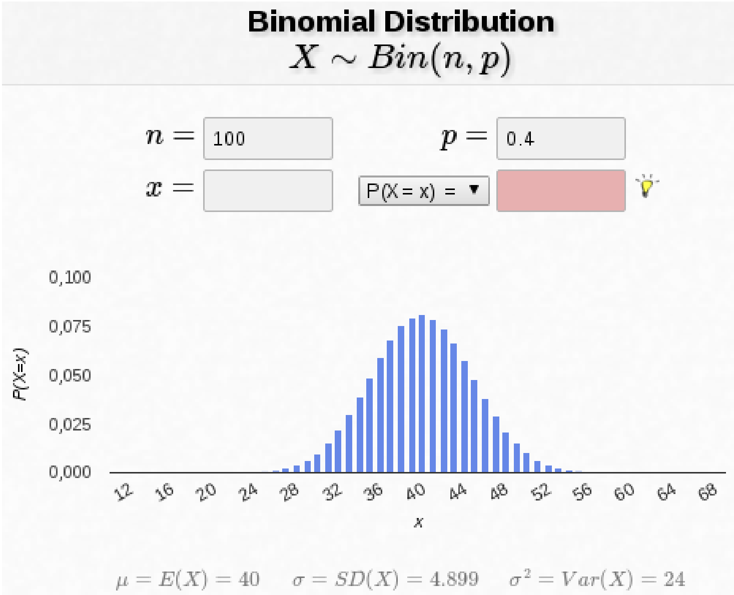


N=20 and p=1. In this case, we get a 20 complete graph.
*Total number of edges: 190 (20*19/2)*

The next figure shows the normalized empirical degree histogram and the theorical one. This model seeks a binomial probability distribution.



Normalized histogram of an ER2 network with n=100 and p=0.4.
This histogram has been generated using the Python script histogram.py



Theoretical binomial distribution with n=100 and p=0.4. Graph obtained via http://homepage.stat.uiowa.edu/~mbognar/applets/bin.html

Watts-Strogatz (WS) model

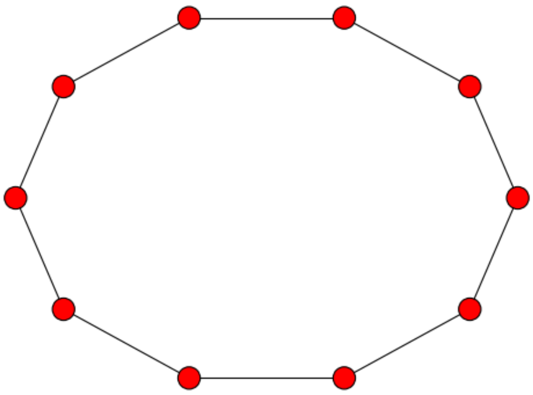
Implemented in the file WS.py.

Usage: python watts-strogatz n-nodes k-edges p-probability.

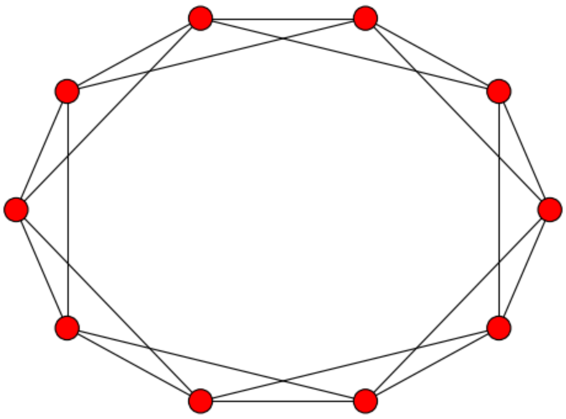
The main algorithm for this method is:

```
//the algorithm constructs a graph with the desired number of edges
//(with the same degree per node)
//each node gets connected with a determinated number of neighbours
for j in range(0, numnodes):
        for k in range(0, numnodes):
                l = abs(j-k) % (numnodes -1 - indeg/2 )
                if l > 0 and l<=indeg/2:
                        G.add_edge(j, k)

//asociated probability to have an edge between two nodes
for j in range(0, numnodes):
        for k in range(j+1, numnodes):
                r=np.random.choice([0,1],1,p=[1-prob,prob])[0]
                if r==1:
                        G.add_edge(j,k)
```

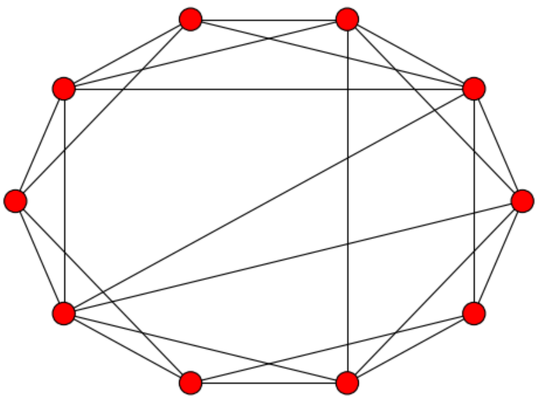
Here we have some samples:



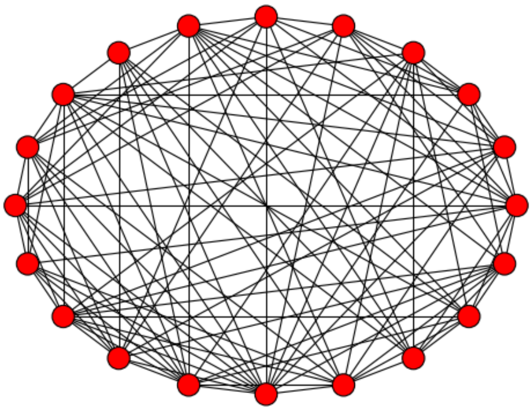
Parameters: # nodes = 10, #edges=10, probability=0



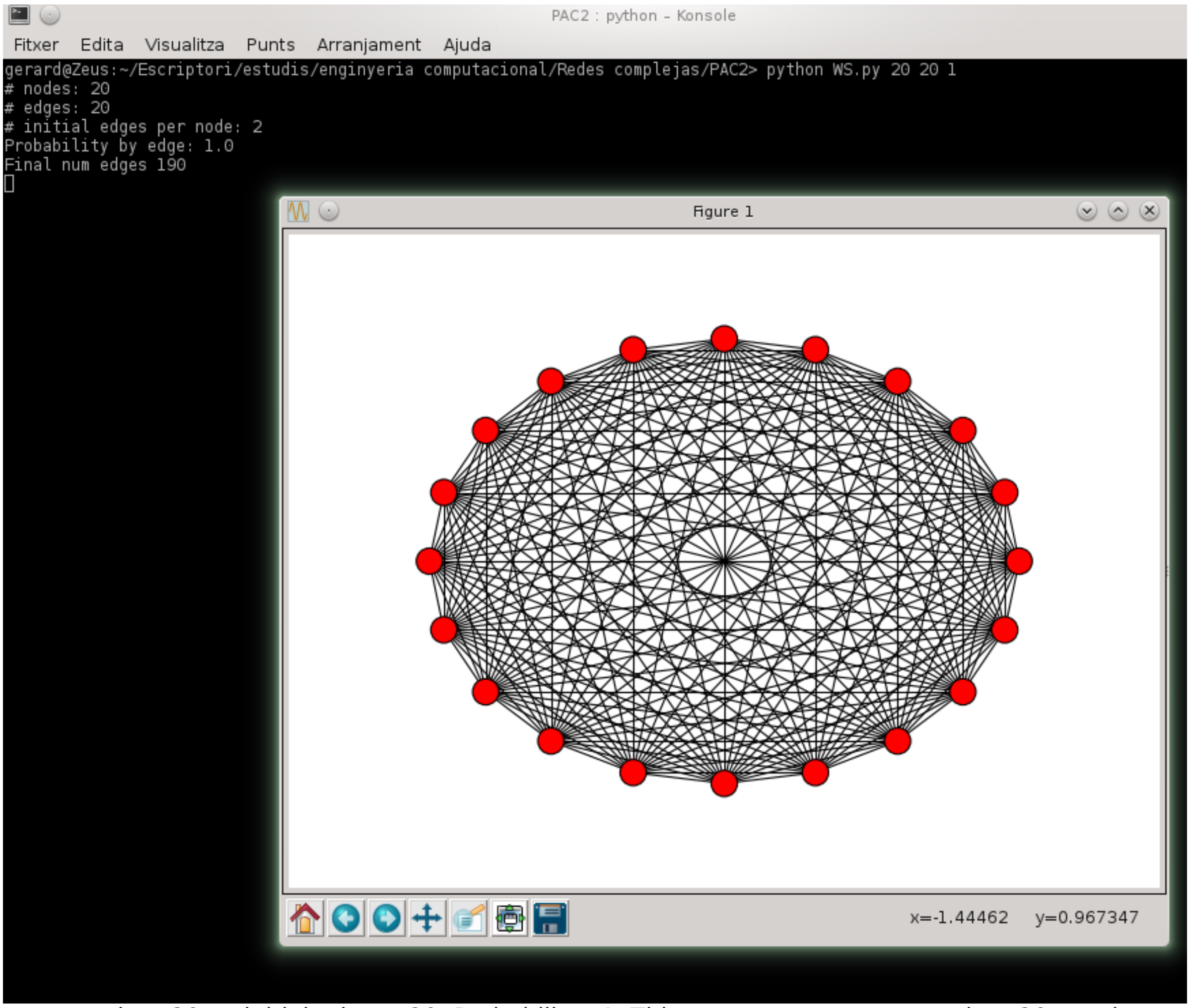
#nodes=10, #edges=20, probability=0



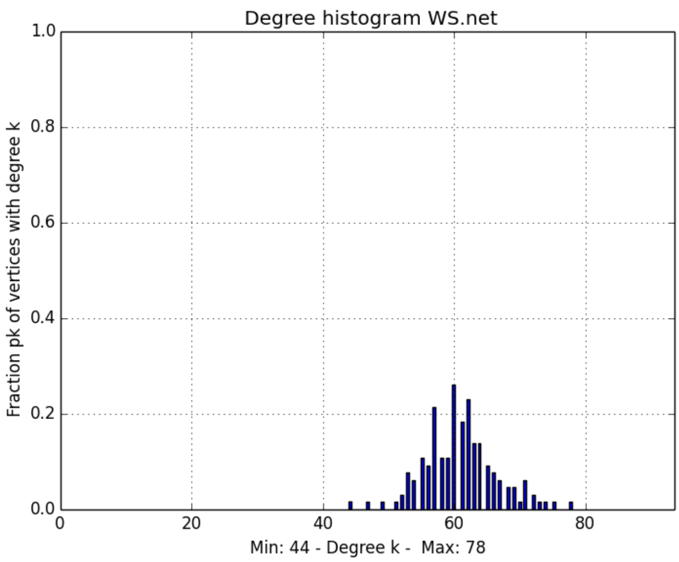
#niodes=10, initial edges=20, probability=0.1. Final number of edges: 24



#nodes=20, initial edges=20. Probability=0.5. Final number of edges: 108



*#nodes=20, # initial edges=20. Probability=1. This case generates a complete 20-graph. Total number of edges: 20*19/2=190 (as prints the algorithm)*



Normalized empirical degree distribution from a WS graph with 150 nodes, 2 initial edges per node and probability=0.4

Barabási & Albert model (BA)

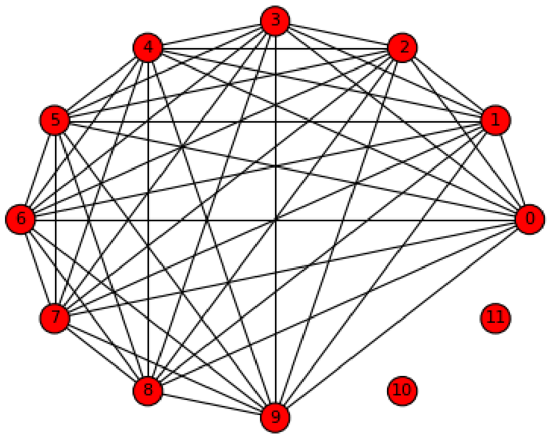
Model implemented in the file BA.py.

Usage: python BA nodes-initial nodes-final degree-new-nodes

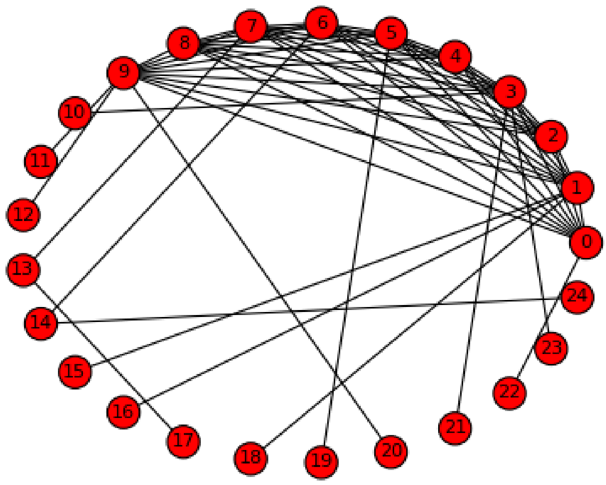
The main algorithm for this method is:

```
# The algorithm first constructs an all-to-all network.

# until we get the number of desired nodes, asign probability to form new edges by
looking the target degree.

while ( G.number_of_nodes() < nnodfi ):
        G.add_node(i)
        while (G.degree(i) < m):
                s=float( sum(nx.degree(G).values()))
                prob_sequence=[float(x / s) for x in nx.degree(G).values()]
                r=np.random.choice(range(0,i+1),1,p=prob_sequence)[0]
                if r<>i:
                       G.add_edge(i, r)

        i = i + 1
        print " Adding new node..."
```

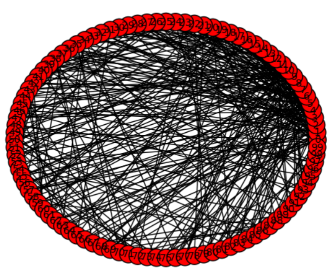
Here we have some samples:



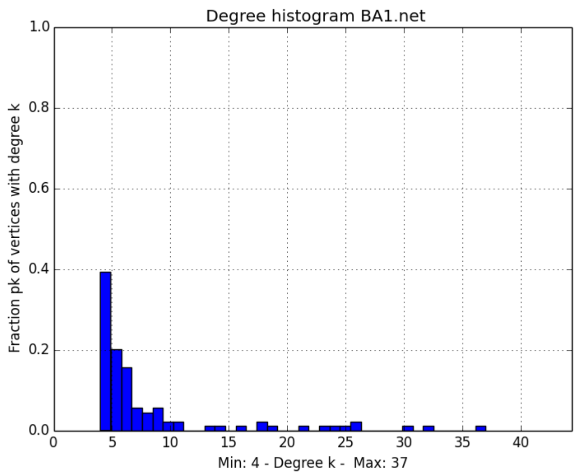
Input parameters: BA.py 10 12 0. 10 initial nodes forming an all-to-all connections. 12 final nodes. 0 degree for the new ones.



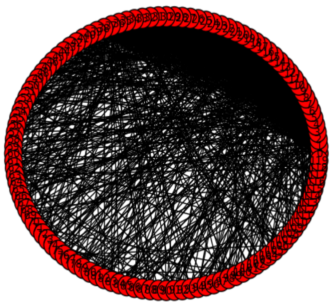
Input parameters: BA.py 10 25 1. 10 initial nodes, 25 final nodes and degree=1 for the new nodes



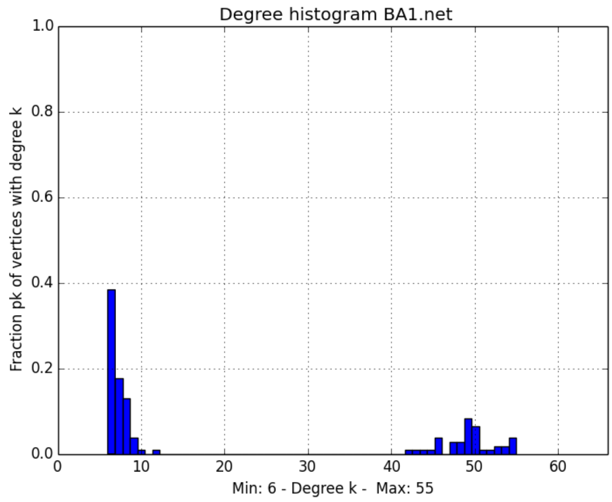
BA with 10 initial nodes, 100 final nodes and degree=4



Degree histogram for the graph in the left. This histogram seeks a power-law distribution.



BA network with 40 initial nodes , 120 final nodes and degree=6



Degree histogram for the graph in the left. Also, a power-law distribution.

I've also implemented a script to calculate the exponent from a graph.

This is the main code:

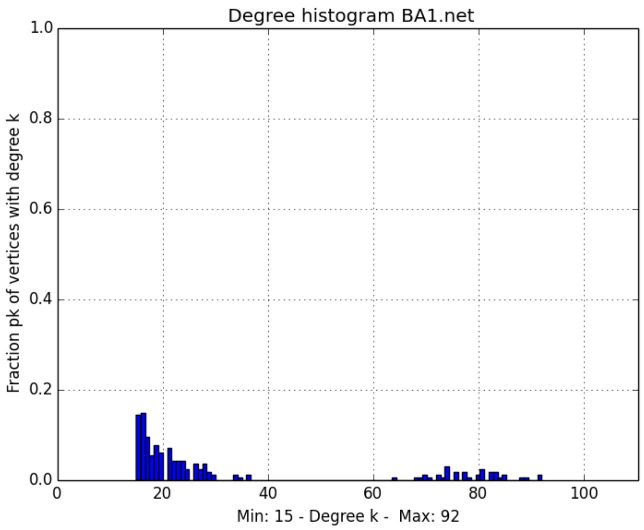
```
G=nx.read_pajek("BA1.net")
d=nx.degree(G)

maxdegree=max ( d.iteritems(), key=operator.itemgetter(1))[1]
mindegree=min ( d.iteritems(), key=operator.itemgetter(1))[1]
sum=0
i=0
for key, val in d.items():
        sum = sum + log ( val / (mindegree - 0.5) )
        i=i+1

al = 1 + i * pow(sum,-1)
```

This is an example:

```
python BA.py 40 200 15
```

Right now, I can calculate the estimated exponent:

```
python exponent.py
2.71820027813
```



BA with 40 initial nodes, 200 final nodes and degree=15

Configuration model (CM)

Implemented in the file CM.py. I've implemented only Poisson and Gamma distributions (NOT power-law).

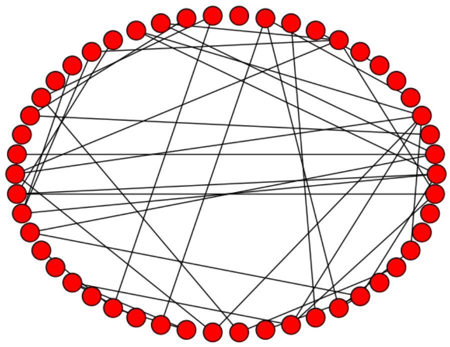
Usage: python CM n-nodes (ER poisson | GM Gamma | SF power-law) value. We will generate a random graph using CM

Example: python CM.py 100 ER 3 (100 nodes with Poisson(3)).
Example: python CM.py 500 GM 2 (500 nodes with Gamma(2,1)).

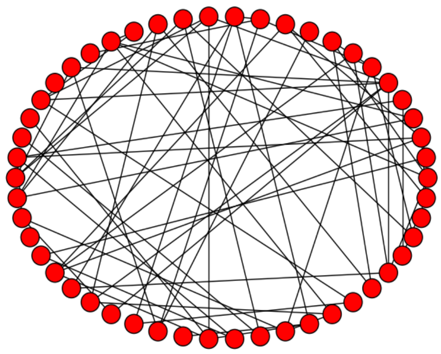
The main algorithm is something like this:

```
if dist=="ER":
        deg_dist=np.random.poisson(param, numnodes)

if dist=="GM":
        deg_dist=np.random.gamma(param, 1, numnodes)

deg_dist[deg_dist > numnodes-1] = numnodes-1

while sum(deg_dist)>0:
        o = random.randint(0,numnodes-1)
        if (deg_dist[o] > 0):
                d = random.randint(0,numnodes-1)
                if (( deg_dist[d] > 0 ) and (o <> d) and (G.has_edge(o,d)==False)):
                        G.add_edge(o,d)
                        deg_dist[o]=deg_dist[o]-1
                        deg_dist[d]=deg_dist[d]-1
```

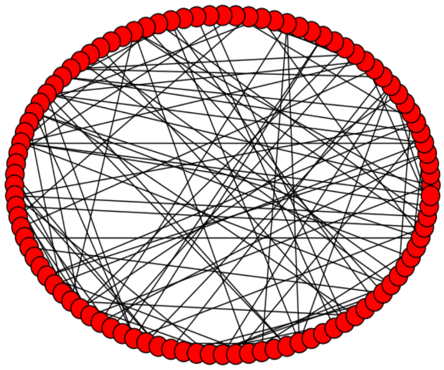
Here we have some samples:



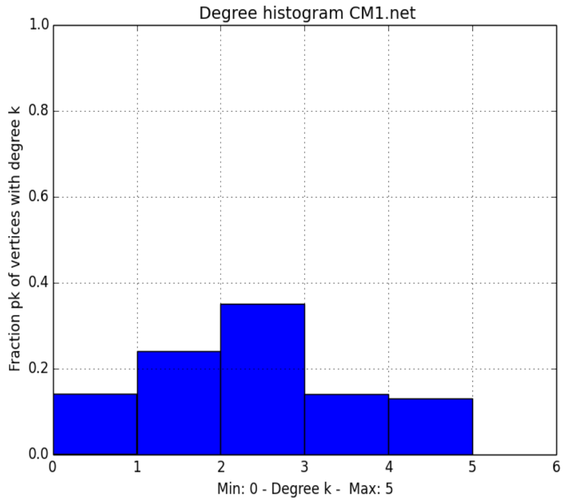
CM with 50 nodes and a Poisson(2)



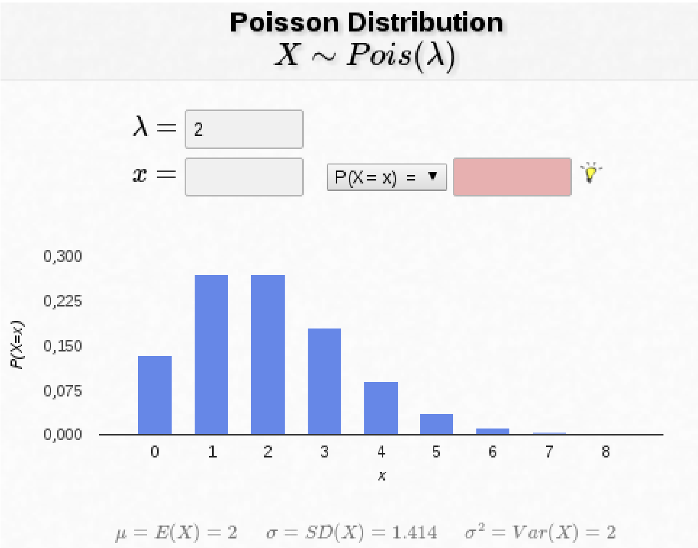
CM with 50 nodes and Gamma(3,1)



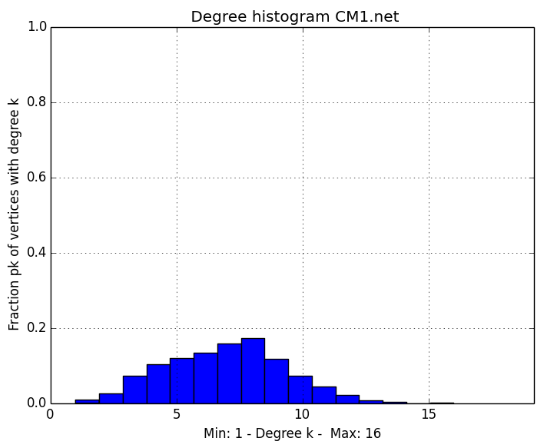
CM with 100 nodes with Poisson(2)



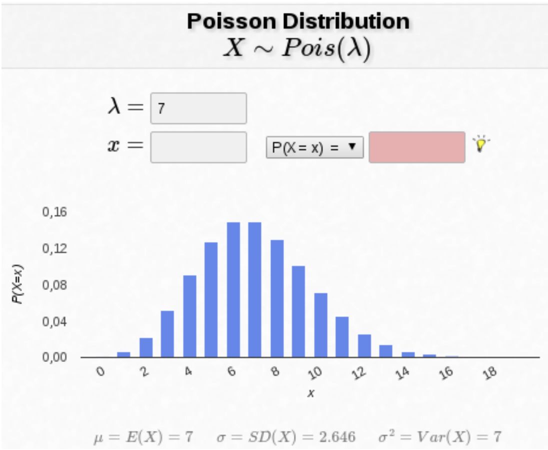
Empirical degree distribution from the last graph



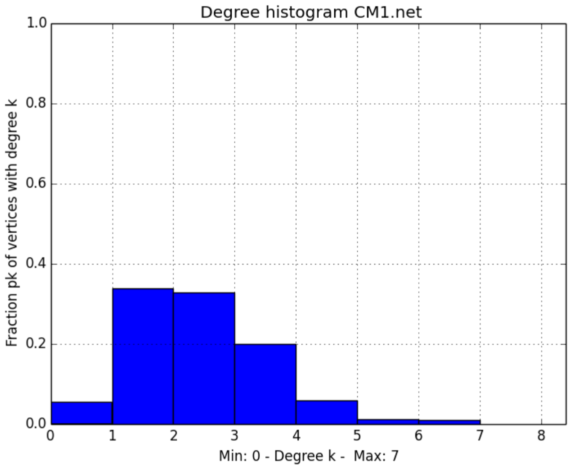
Theorical degree distribution network



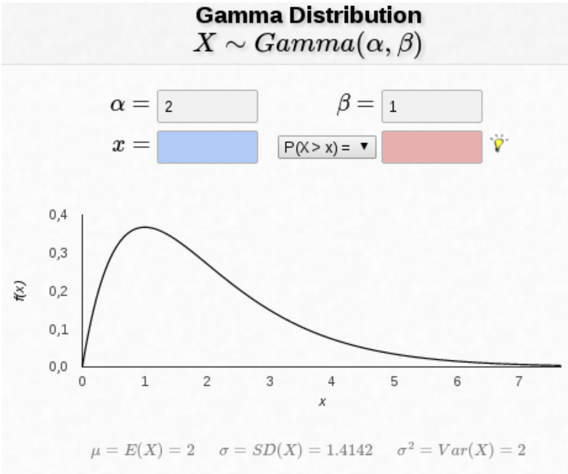
Degree distribution histogram from a network with 500 nodes seeking a Poisson(7)



Theorical Poisson(7) distribution



Degree histogram from a network with 500 nodes and Gamma(2,1) distribution



Theorical gamma(2,1) distribution