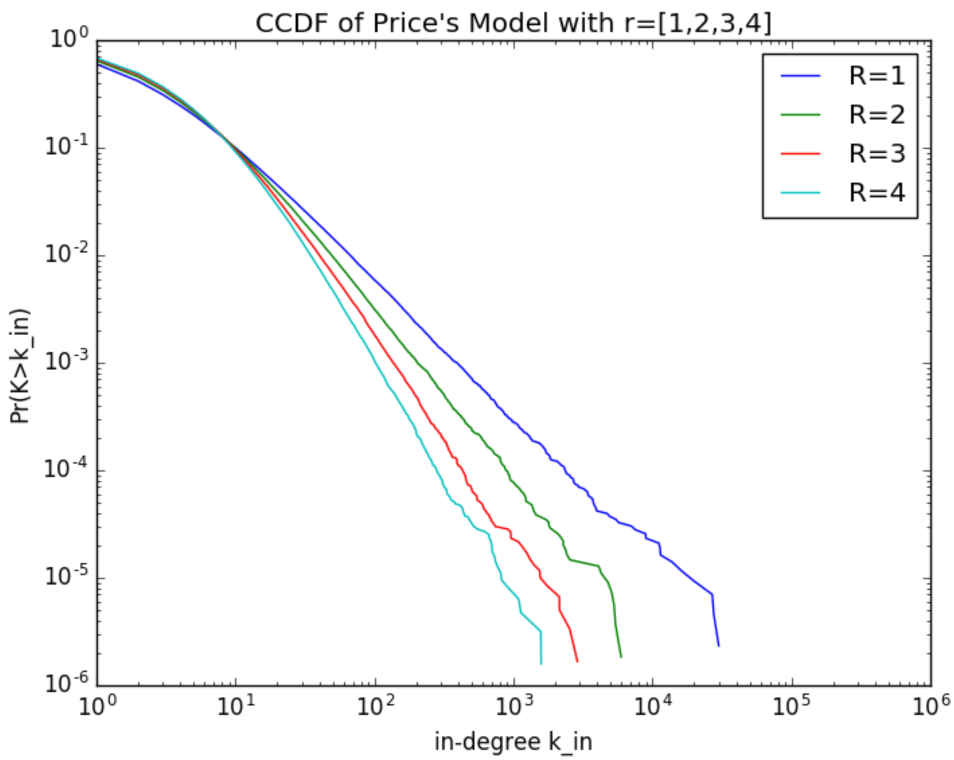
Donovan Guelde
CSCI 5352
PS 6

1. a.



At low k_in values less than 10, distributions of Price's model with varying R values are very similar. In this area of the distribution, in-degree is primarily driven by random attachment. At in-degree greater than 10, the R value becomes more important in determining in-degree, which can be seen in the separate distributions. As R increases, the affect of preferential attachment decreases, causing a narrower distribution in in-degree. After the 'shoulder' driven by random attachment, lower R values cause a more gradual drop-off in in-degree, reflecting a wider distribution. Without preferential attachment, degree distribution would be much narrower, with a steeper drop-off in the CCDF plot (as seen in question 1.e.)

The opposite effect can be seen, to a small degree, for Pr(K>k_in) where k_in is less than 10. At lower R values, the fraction of nodes with degree greater than 1 to 10 is higher than larger R values. This is caused by the larger effect of random attachment. Nodes that are added to the network towards the end of the growth period have an increased chance of being linked to by even later additions, as opposed to the preferential attachment model, which would give these 'late-comer' nodes a small chance of having an increased in-degree. This explains the small but consistent trend at in-degree =0. As the R value decreases, the fraction of nodes with in-degree = 0 also decreases.

1.b. Results were obtained using the simulation designed for 1.a. above, with c=12, R=5, and n=10^6. (i) Averaging over 100 iterations, the average in-degree of the first 10% (100,000) of papers published was determined to be 81.34. The range of in-degree for the first 10% of papers was quite large, with the largest average in-degree of 38,083.27, and the smallest in-degree of 17.11.
(ii). Average in-degree for the last 10% of nodes added to the network was 0.187. The largest average in-degree was 0.63, with many nodes having in-degree of zero.

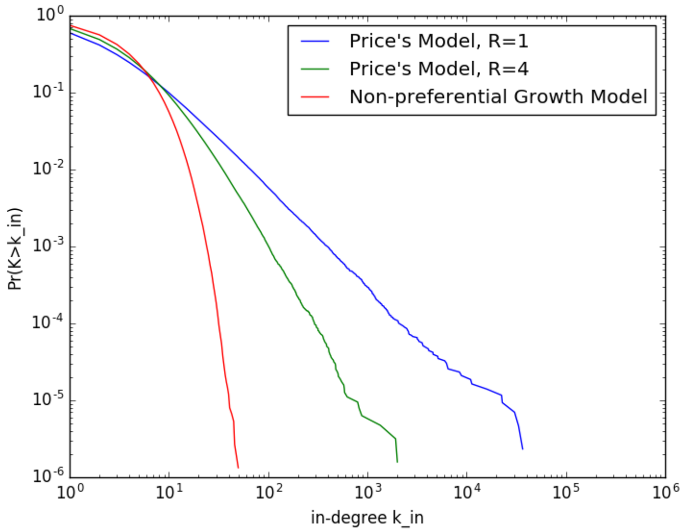
To examine the first-mover advantage, we can examine the results for the first 10% of papers more carefully. The highest average in-degree is over 38,000, but moving down the sorted list of average in-degree just 13 places gives an average in-degree of 20,080. By position 587, average in-degree is less than 1000. The first-mover advantage definitely exists, and is limited to the small fraction of nodes added at the beginning of graph construction. A similar examination of results for the last 10% of papers published reveals a very small range, as previously mentioned. A paper published in the last 90th percentile has very little advantage over even the very last paper published.

1.c. Average in-degree for the first 10% of papers published in the "arXiv citation networks (1993-2003)" was determined to be 23.62, while the average in-degree for the last 10% of papers published was 14.36. These results do not agree with results from 1.b. While there are more papers with relatively high in-degree in the first 10% (over 100, for example), there are also many papers with low (less than 10) in-degree. The extreme first-mover advantage seen in 1.b. is not present in this real network. Results for the last 10% of papers published also disagree with 1.b. There are very few papers with 0 in-degree in the last 10% of this network, in contrast to the many found in the simulation conducted in 1.b.

A possible explanation for absence of the extreme first-mover advantage seen in 1.b. could be that authors were not forced to rely solely on information and ideas presented in the early papers. For example, the 10th paper published could be drawing on outside sources and independently formulating conclusions from the authors of the earlier papers. In a new field, particularly, this may be common. Many authors, while working in the same new field, draw from various sources. This would reduce the first-mover advantage as other authors continue to publish. Subsequent authors can also choose among the previously-published papers of highest quality. This would give a small range of papers a high in-degree, rather than just a few early-movers.

1.d. Ways that the preferential attachment mechanism is unrealistic are the extreme advantage given to first-movers, the extreme penalty given to late-comers, and the disregard for quality of work in a given paper, regardless of time published. To analyze the first weakness of the preferential attachment model, the extreme first-mover advantage, we can perform analysis similar to 1.c. above. A preferential attachment simulation shows a very strong first-mover advantage, while analysis of a real network does not bear this out. There is definitely higher in-degree in earlier-published papers in the real network, but nowhere near as large as that predicted by simulation. The extreme penalty for late-comers can be examined in a similar manner. Simulations predict average in-degree of late-comers to be much less than 1, while real network examination in 1.c. does not agree. To determine if quality of work, regardless of paper publishing date, one could look for outliers in both large and small expected in-degree. The existence of a paper published in the very early stages of network growth, but with very low in-degree, could be an indicator of poor methodology or experiment design, among other things. Similarly, a paper with very high in-degree among the last papers published could be an indicator of a breakthrough, interesting conclusions, etc. that allows the paper to gather many more citations than the preferential model would predict.

1.e. Removing the preferential attachment mechanism from our simulation used for 1.a. above acts to drastically reduce the distribution of node in-degree. As seen in 1.a, the distribution of in-degrees narrows as R increases. When preferential attachment is removed completely, we see the same 'shoulder' present in Price's model, but after that, the fraction of nodes with larger in-degree drops off very quickly to zero. This plot completely lacks the power-log curve shown by Price's model. The fraction of nodes with in-degree zero is smaller, since even the very last papers published have a higher chance of being cited without the preferential attachment mechanism.



Code for 1.a.

```
# Donovan Guelde
# CSCI-5352
# PS6 Q.1.a

import numpy as np
import matplotlib.pyplot as plt



def growNetwork(r,n,pr,c):

        vertexLabelList=np.full(n*c,-1)
        adjacencyList=np.zeros(shape=(n,c))
        adjacencyList.fill(-1)
        numberVertexes=0
        numberEdges=0
        iterations=1
        #make a 'seed' graph, c+1 nodes to simulate growth from 0 nodes
        #node 0 has no outgoing edges, node 1 points to 0, node 2 points to 0 and 1,
        #node 3 points to nodes 0,1,2, etc. Node[c+1] now has c outgoing edges
        for index in range(0,c):
                for index2 in range(index+1,c+1):
                        adjacencyList[index2][index]=index
                        vertexLabelList[numberEdges]=index
                        numberEdges+=1
        numberVertexes=c+1 #we preseeded |c+1| vertexts
        for index in range(numberVertexes,n): #add remaining vertexes to network
                chosen=[]
```

```
                tempList=vertexLabelList[0:numberEdges]
                coinFlips=np.random.random(c)
                for index2 in range(0,c): #each new vertex hac c out-degree
                        if coinFlips[index2]<pr: #choose in proportion to in-degree
                                edgePointsTo=np.random.choice(tempList)
                                while (edgePointsTo in chosen or edgePointsTo==-1):
                                        edgePointsTo=np.random.choice(tempList)
                        else: #randomly select
                                edgePointsTo=np.random.randint(0,high=numberVertexes-1)
                                while(edgePointsTo in chosen):
                                        edgePointsTo=np.random.randint(0,high=numberVertexes-1)
                        chosen.append(edgePointsTo)
                counter=0
                numberVertexes+=1
                for item in chosen: #add new edges to list
                        adjacencyList[index][counter]=item
                        vertexLabelList[numberEdges]=item
                        counter+=1
                        numberEdges+=1
        return adjacencyList[1:]




def main():
        #parameters per assignment
        c=3
        N=1000000
        R=[1.,2.,3.,4.]
        XaxisResults=[]
        YaxisResults=[]
        #bigger r --> lower pr --> higher chance of random assignment of new edges --> more uniform distribution
        for r in R:
                pr=c/(c+r) #probability of attaching to node in proportion of in-degree
                n=N
                fileName = '1a_plot.png'.format(str(r))

                adjacencyList=growNetwork(r,n,pr,c)
                nodes,indegree=np.unique(adjacencyList,return_counts=True) #
                uniqueValues=np.unique(indegree)
                indegreeList=np.zeros(n)
                counter=0
                for item in nodes:
                        if(item!=-1):
                                indegreeList[item]=indegree[counter]
                        counter+=1

                xAxis=np.unique(indegreeList) #observed degree on x axis
                XaxisResults.append(xAxis)
                yAxis=np.zeros(len(xAxis))
                counter=0
                for item in uniqueValues:
                        yAxis[counter]=np.count_nonzero(indegreeList>=item) #fraction (nodes indegree >= indegree k)
                        counter+=1
                yAxis=yAxis/float(np.amax(yAxis)) #regularize as a fraction < 1
                YaxisResults.append(yAxis)

        labels=['R=1','R=2','R=3','R=4']
```

```
        plt.title('CCDF of Price\'s Model with r=[1,2,3,4]')
        plt.xlabel('in-degree k_in')
        plt.ylabel('Pr(K>k_in)')
        plt.xlim(1,n)
        for index in range(0,len(R)):
                plt.loglog(XaxisResults[index],YaxisResults[index],label=labels[index])
        plt.legend(loc='upper right')
        plt.savefig(fileName)
        plt.close()

main()
```

Code for 1.b.

```
# Donovan Guelde
# CSCI-5352
# PS6 Q.1.b

import numpy as np
import matplotlib.pyplot as plt
import time



def growNetwork(r,n,pr,c):

        vertexLabelList=np.full(n*c,-1)
        adjacencyList=np.zeros(shape=(n,c))
        adjacencyList.fill(-1) #-1 signifies no edge
        numberVertexes=0
        numberEdges=0

        #make a 'seed' graph, c+1 nodes to simulate growth from 0 nodes
        #node 0 has no outgoing edges, node 1 points to 0, node 2 points to 0 and 1,
        #node 3 points to nodes 0,1,2, etc. Node[c+1] now has c outgoing edges
        for index in range(0,c):
                for index2 in range(index+1,c+1):
                        adjacencyList[index2][index]=index
                        vertexLabelList[numberEdges]=index
                        numberEdges+=1
        numberVertexes=c+1 #we preseeded |c+1| vertexts
        for index in range(numberVertexes,n): #add remaining vertexes to network
                chosen=[]
                tempList=vertexLabelList[0:numberEdges]
                coinFlips=np.random.random(c)
                for index2 in range(0,c): #each new vertex hac c out-degree
                        if coinFlips[index2]<pr: #choose in proportion to in-degree
                                edgePointsTo=np.random.choice(tempList)
                                while (edgePointsTo in chosen or edgePointsTo==-1):
                                        edgePointsTo=np.random.choice(tempList)
                        else: #randomly select
                                edgePointsTo=np.random.randint(0,high=numberVertexes-1)
                                while(edgePointsTo in chosen):
                                        edgePointsTo=np.random.randint(0,high=numberVertexes-1)
                        chosen.append(edgePointsTo)
                counter=0
                numberVertexes+=1
                for item in chosen: #add new edges to list
```

```
                        adjacencyList[index][counter]=item
                        vertexLabelList[numberEdges]=item
                        counter+=1
                        numberEdges+=1

        return adjacencyList




def main():
        #parameters per assignment
        c=12
        n=1000000
        r=5.
        iterations=100 #?
        fileName = '1b_{}.txt'.format(str(iterations))
        pr=c/(c+r) #probability of attaching to node in proportion of in-degree
        results=np.zeros((iterations,n*.2))




        for iteration in xrange(iterations):

                start=time.time()
                adjacencyList=growNetwork(r,n,pr,c).astype(int)
                adjacencyList=adjacencyList[adjacencyList>=0]
                adjacencyList=np.sort(adjacencyList,axis=None)

                for index in xrange(len(adjacencyList)):
                        v = adjacencyList[index]

                        if ( v < int(n*.1)):
                                results[iteration][v]+=1

                        if (v >= int(n*.9)):
                                results[iteration][v-n]+=1


                print time.time()-start

        average=np.average(results,axis=0)
        np.savetxt(fileName,average,fmt='%1.4f')






main()
```

Code for 1.c.

```
# Donovan Guelde
# CSCI 5352 PS6 q1c

import numpy as np
from igraph import *
```

```
from sets import Set

def readFile():
	g = Graph(directed=True)

	with (open('cit-HepPh-dates.txt','r')) as f:
		next(f) #skip header row
		nodesByDate=[]
		for line in f:
			line=line.split()
			node,date=line[0],line[1]
			nodesByDate.append(node)




	edgeList=[]
	with (open('cit-HepPh.txt','r')) as f:

		nodes=Set()
		f.seek(0,0)
		next(f)
		next(f)
		next(f)
		next(f)
		for line in f:
			line=line.split()
			node,neighbor=line[0],line[1]
			if node not in nodes:
				nodes.add(node)
			if neighbor not in nodes:
				nodes.add(neighbor)

			edgeList.append((node,neighbor))
	for item in nodes:
		g.add_vertex(item)
	g.add_edges(edgeList)
	#how many vertices have date info?
	count=0.
	for item in nodes:
		if item in nodesByDate:
			count+=1
	return g,nodesByDate,nodes,count



def main():
	g,nodesByDate,nodes,haveDateInfo=readFile()
	interval=int(.1*haveDateInfo)
	print interval
	print nodesByDate[0]
	firstTenPercent=[]
	count=0
	counter=0
	firstTenTotal=0.
	while(count<interval):
		try:
			firstTenTotal+= Graph.degree(g,nodesByDate[counter],mode='in')
```

```
            node= nodesByDate[counter]
            degree= Graph.degree(g,nodesByDate[counter],mode='in')
            firstTenPercent.append((node,degree))
            count+=1
        except ValueError:
            pass
        counter+=1
    print "first 10% average:",firstTenTotal/interval

    counter=int(haveDateInfo)
    sum=0.
    count=0
    lastTenPercent=[]
    lastTenTotal=0.
    while(count<interval):
        try:
            lastTenTotal+= Graph.degree(g,nodesByDate[counter],mode='in')
            node= nodesByDate[counter]
            degree= Graph.degree(g,nodesByDate[counter],mode='in')
            lastTenPercent.append((node,degree))
            count+=1
        except ValueError:
            pass
        counter-=1
    print "last 10% average:",lastTenTotal/interval

    with(open('1cResults.txt','w')) as f:
        f.write('first ten percent:\n')
        f.write('\n'.join('%s %s' % x for x in firstTenPercent))

        f.write("total: "+str(firstTenTotal)+'\n')
        f.write("Average: "+str(firstTenTotal/interval))
        f.write('\n\n')
        f.write('last ten percent:\n')
        f.write('\n'.join('%s %s' % x for x in lastTenPercent))
        f.write('total: '+str(lastTenTotal)+'\n')
        f.write("Average: "+str(lastTenTotal/interval))






main()
```

Code for 1.e.

```
# Donovan Guelde
# CSCI-5352
# PS6 Q.1.e

import numpy as np
import matplotlib.pyplot as plt



def growNetwork(r,n,pr,c):

```

```python
    vertexLabelList=np.full(n*c,-1)
    adjacencyList=np.zeros(shape=(n,c))
    adjacencyList.fill(-1)
    numberVertexes=0
    numberEdges=0
    iterations=1
    #make a 'seed' graph, c+1 nodes to simulate growth from 0 nodes
    #node 0 has no outgoing edges, node 1 points to 0, node 2 points to 0 and 1,
    #node 3 points to nodes 0,1,2, etc. Node[c+1] now has c outgoing edges
    for index in range(0,c):
        for index2 in range(index+1,c+1):
            adjacencyList[index2][index]=index
            vertexLabelList[numberEdges]=index
            numberEdges+=1
    numberVertexes=c+1 #we preseeded |c+1| vertexts
    for index in range(numberVertexes,n): #add remaining vertexes to network
        chosen=[]
        coinFlips=np.random.random(c)
        tempList=vertexLabelList[0:numberEdges]
        for index2 in range(0,c): #each new vertex hac c out-degree
            if coinFlips[index2]<pr: #choose in proportion to in-degree
                edgePointsTo=np.random.choice(tempList)
                while (edgePointsTo in chosen or edgePointsTo==-1):
                    edgePointsTo=np.random.choice(tempList)
            else: #randomly select
                edgePointsTo=np.random.randint(0,high=numberVertexes-1)
                while(edgePointsTo in chosen):
                    edgePointsTo=np.random.randint(0,high=numberVertexes-1)
            chosen.append(edgePointsTo)
        counter=0
        numberVertexes+=1
        for item in chosen: #add new edges to list
            adjacencyList[index][counter]=item
            vertexLabelList[numberEdges]=item
            counter+=1
            numberEdges+=1
    return adjacencyList[1:]


def growNetworkNonpreferential(n,c):
    vertexLabelList=np.full(n*c,-1)
    adjacencyList=np.zeros(shape=(n,c))
    adjacencyList.fill(-1)
    numberVertexes=0
    numberEdges=0
    iterations=1
    #make a 'seed' graph, c+1 nodes to simulate growth from 0 nodes
    #node 0 has no outgoing edges, node 1 points to 0, node 2 points to 0 and 1,
    #node 3 points to nodes 0,1,2, etc. Node[c+1] now has c outgoing edges
    for index in range(0,c):
        for index2 in range(index+1,c+1):
            adjacencyList[index2][index]=index
            vertexLabelList[numberEdges]=index
            numberEdges+=1
    numberVertexes=c+1 #we preseeded |c+1| vertexts
    for index in range(numberVertexes,n): #add remaining vertexes to network
        chosen=[]
        tempList=vertexLabelList[0:numberEdges]
```

```
            for index2 in range(0,c): #each new vertex hac c out-degree
                edgePointsTo=np.random.randint(0,high=numberVertexes-1)
                while(edgePointsTo in chosen):
                    edgePointsTo=np.random.randint(0,high=numberVertexes-1)
                chosen.append(edgePointsTo)
            counter=0
            numberVertexes+=1
            for item in chosen: #add new edges to list
                adjacencyList[index][counter]=item
                vertexLabelList[numberEdges]=item
                counter+=1
                numberEdges+=1
        return adjacencyList[1:]


def main():
        #parameters per assignment
        c=3
        N=1000000
        XaxisResults=[]
        YaxisResults=[]

        R=[1.,4.]
        #bigger r --> lower pr --> higher chance of random assignment of new edges --> more uniform distribution
        for r in R:
            pr=c/(c+r) #probability of attaching to node in proportion of in-degree
            n=N
            fileName = '1e_plot_r={}.png'.format(str(r))

            adjacencyList=growNetwork(r,n,pr,c)
            nodes,indegree=np.unique(adjacencyList,return_counts=True) #
            uniqueValues=np.unique(indegree)
            indegreeList=np.zeros(n)
            counter=0
            for item in nodes:
                if(item!=-1):
                    indegreeList[item]=indegree[counter]
                counter+=1

            xAxis=np.unique(indegreeList) #observed degree on x axis
            XaxisResults.append(xAxis)
            yAxis=np.zeros(len(xAxis))
            counter=0
            for item in uniqueValues:
                yAxis[counter]=np.count_nonzero(indegreeList>=item) #fraction (nodes indegree >= indegree k)
                counter+=1
            yAxis=yAxis/float(np.amax(yAxis)) #regularize as a fraction < 1
            YaxisResults.append(yAxis)

        adjacencyList=growNetworkNonpreferential(n,c) #non-preferential attachment graph
        nodes,indegree=np.unique(adjacencyList,return_counts=True) #
        uniqueValues=np.unique(indegree)
        indegreeList=np.zeros(n)
        counter=0
        for item in nodes:
            if(item!=-1):
                indegreeList[item]=indegree[counter]
            counter+=1
```

```

        xAxis=np.unique(indegreeList) #observed degree on x axis
        XaxisResults.append(xAxis)
        yAxis=np.zeros(len(xAxis))
        counter=0
        for item in uniqueValues:
                yAxis[counter]=np.count_nonzero(indegreeList>=item) #fraction (nodes indegree >= indegree k)
                counter+=1
        yAxis=yAxis/float(np.amax(yAxis)) #regularize as a fraction < 1
        YaxisResults.append(yAxis)

        labels=['Price\'s Model, R=1','Price\'s Model, R=4','Non-preferential Growth Model']
        plt.xlabel('in-degree k_in')
        plt.ylabel('Pr(K>k_in)')
        plt.xlim(1,n)
        for index in range(0,len(R)+1):
                plt.loglog(XaxisResults[index],YaxisResults[index],label=labels[index])

        plt.legend(loc='upper right')
        plt.savefig(fileName)
        plt.close()

main()
```