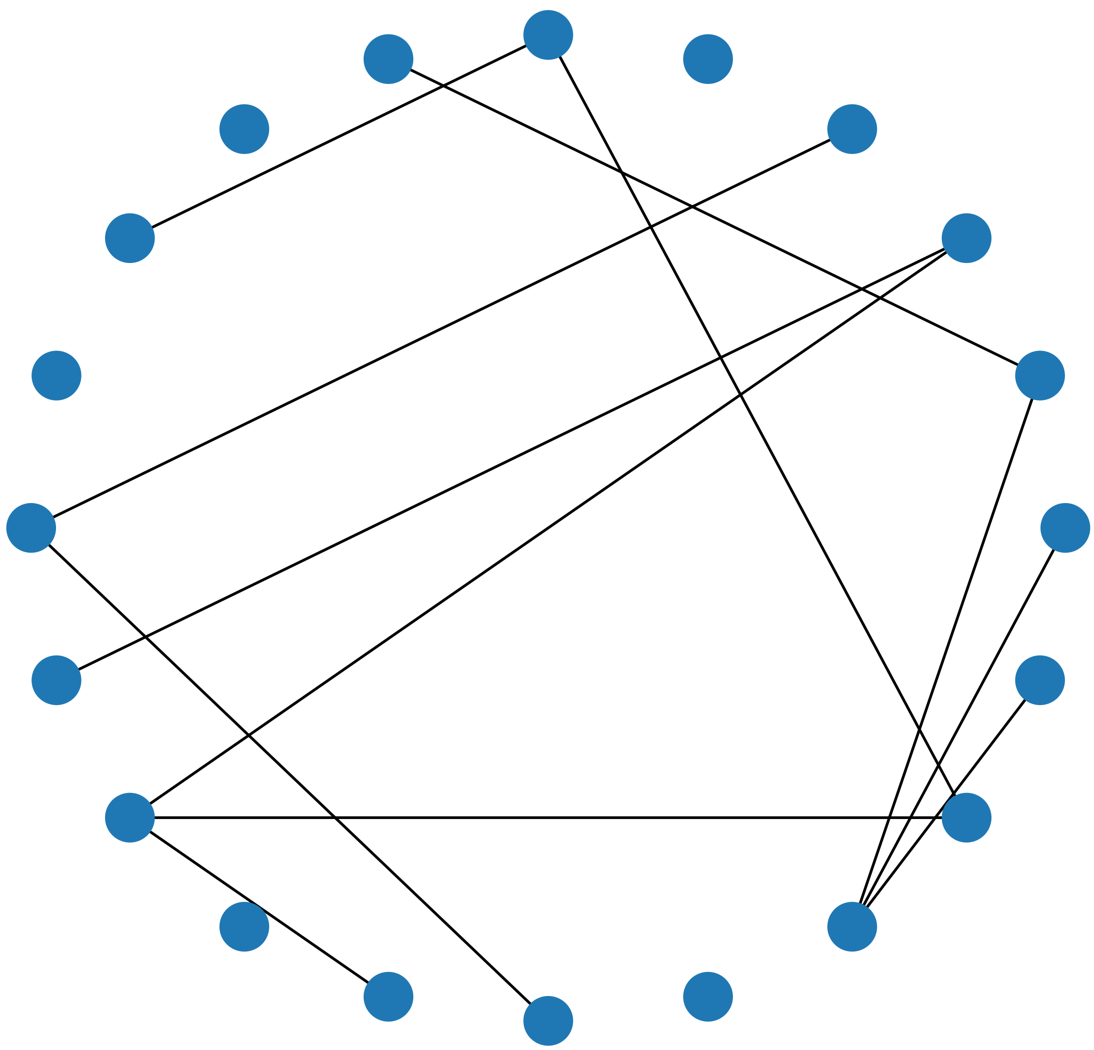
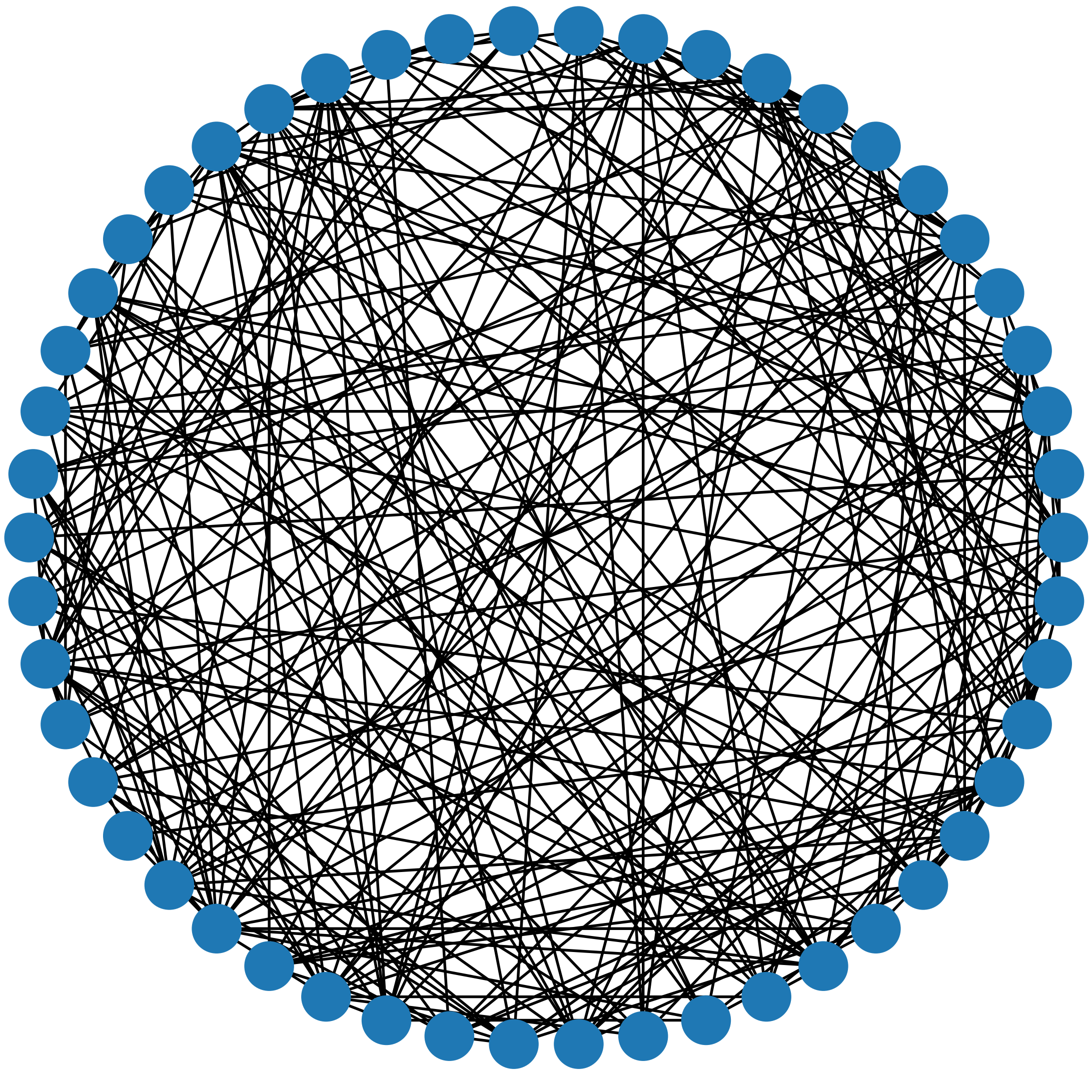
Lukas Fu Homework 3

Exercise 12.1

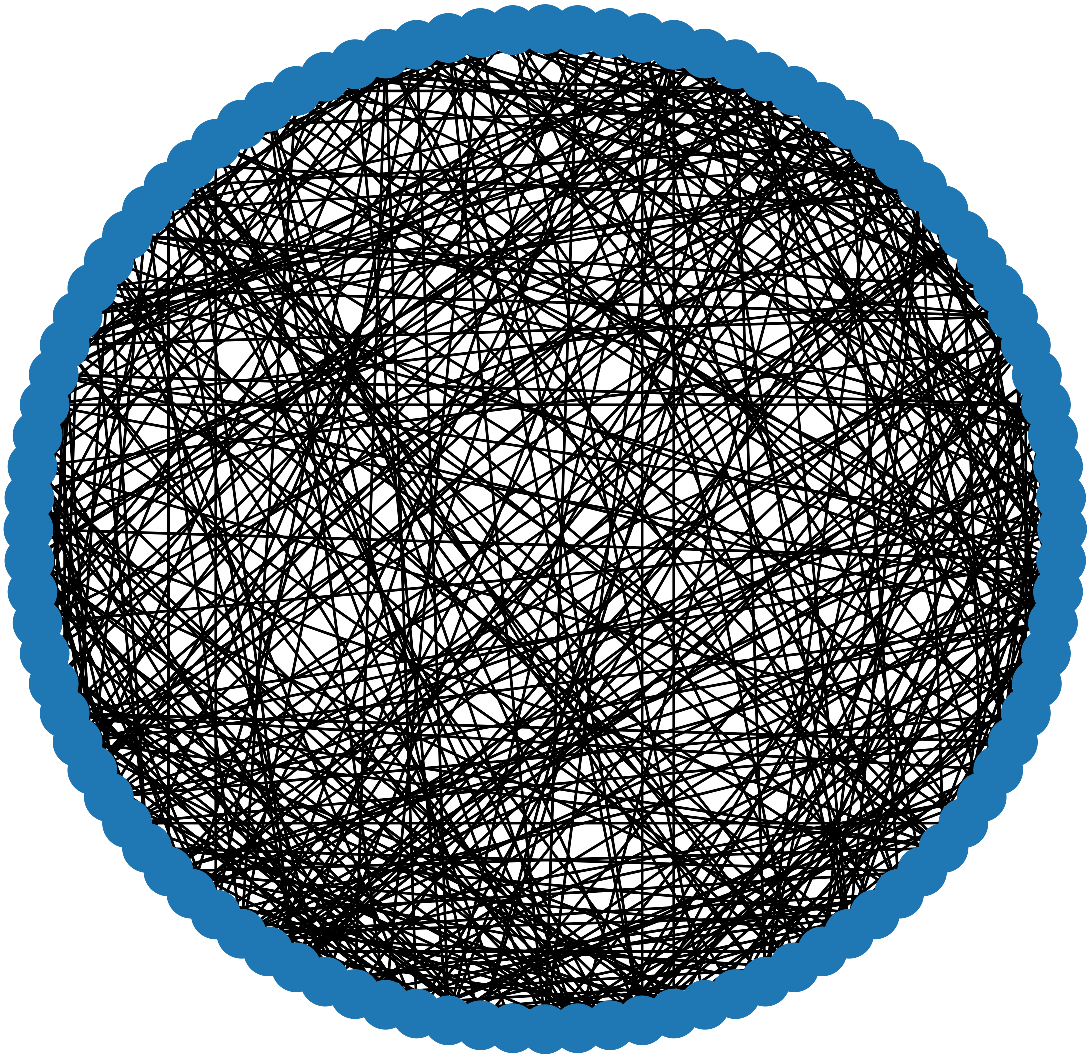
Subproblem (a)



Figur 1: With parameter values of $n = 20$ and $p = 0.1$.

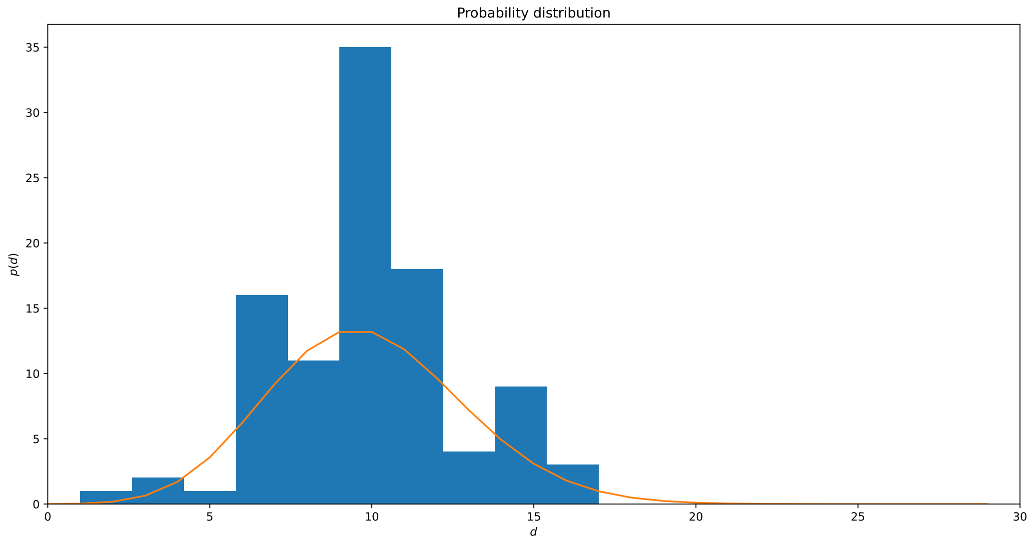


Figur 2: With parameter values of $n = 50$ and $p = 0.2$.

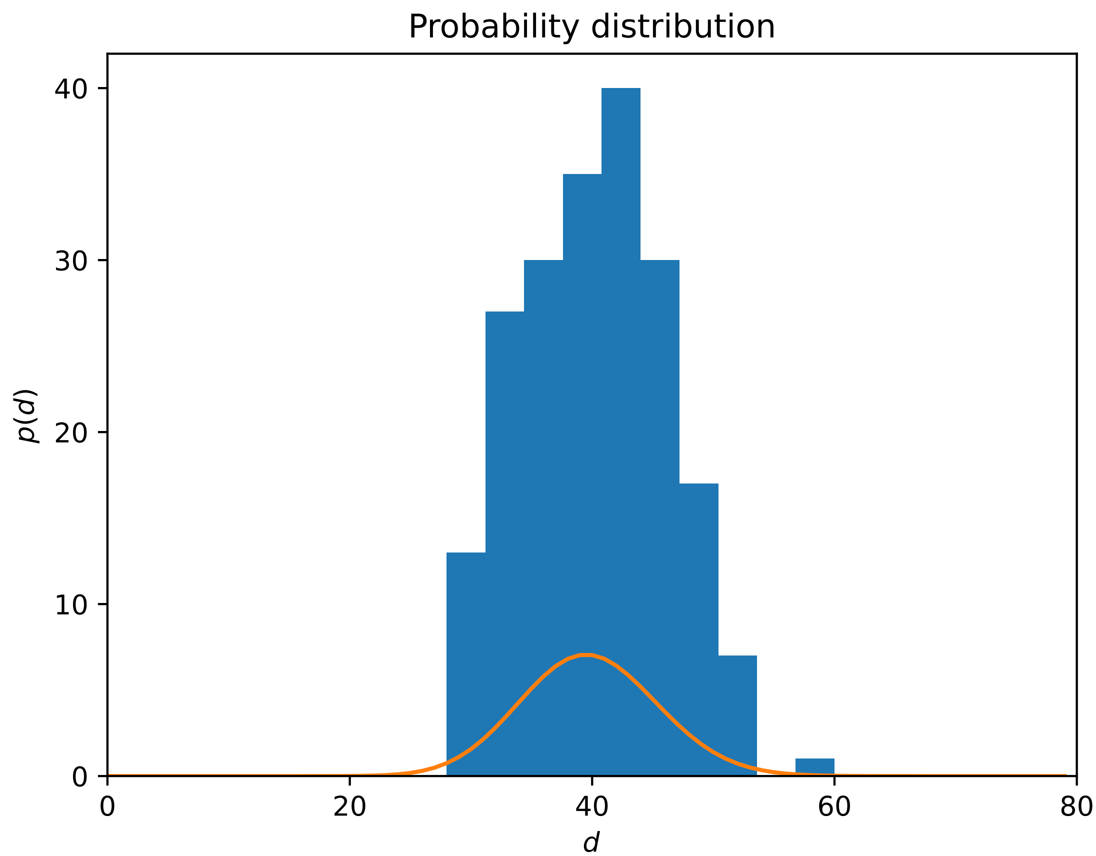


Figur 3: With parameter values of $n = 100$ and $p = 0.1$.

Subproblem (b)

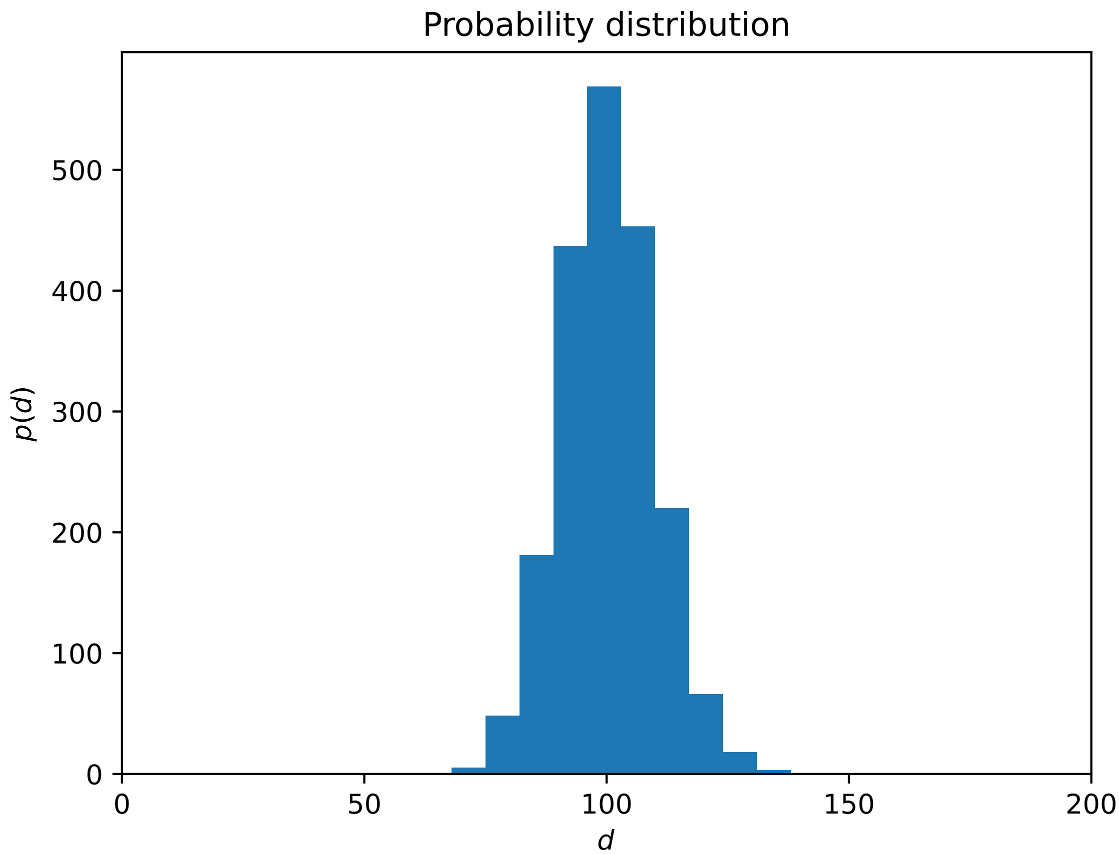


Figur 4: With parameter values of $n = 100$ and $p = 0.1$, comparing with a distribution formula 12.1 in the book.



Figur 5: With parameter values of $n = 200$ and $p = 0.2$, comparing with a distribution formula 12.1 in the book. The theoretical distribution also seem to approach a Gaussian distribution as n increases.

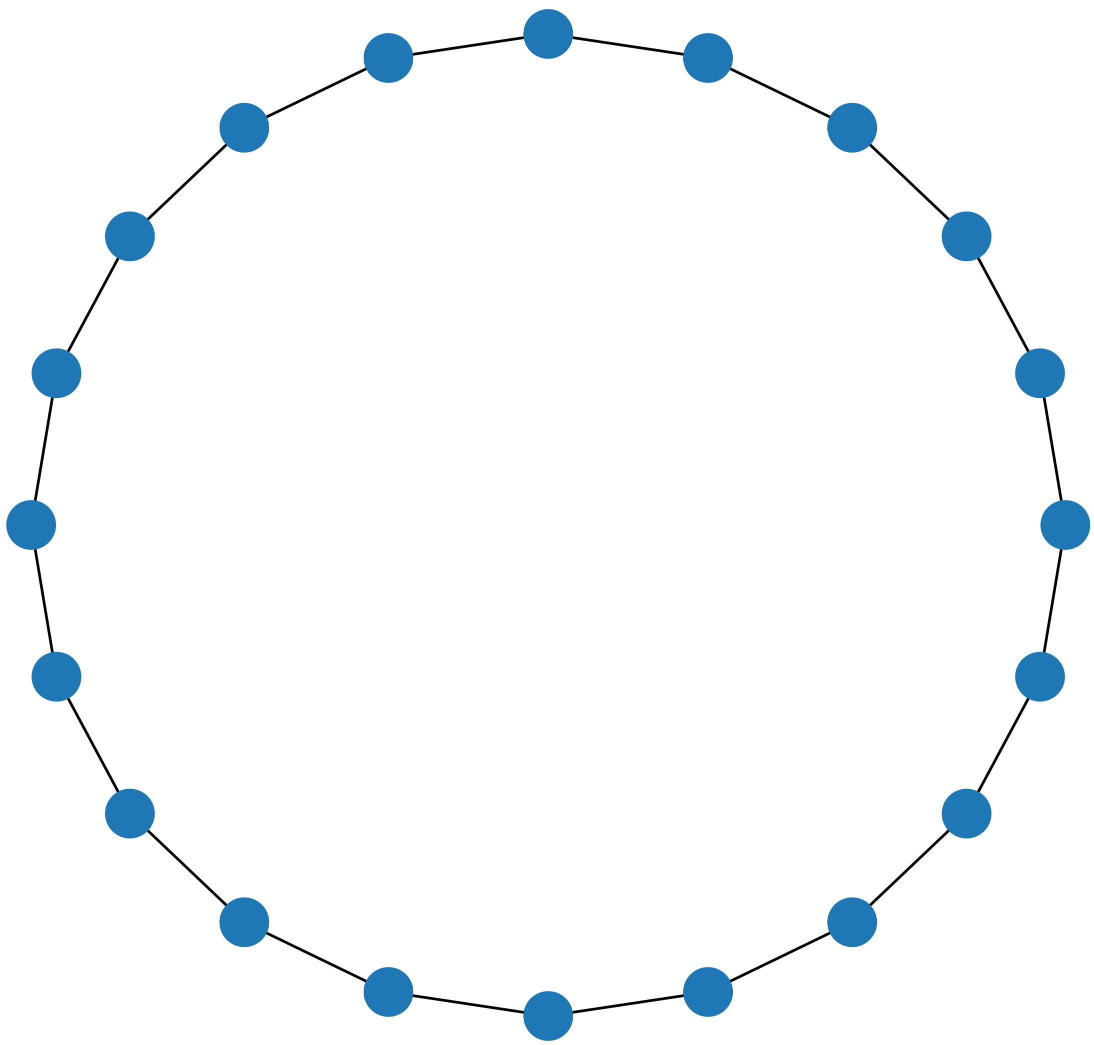
Subproblem (c)



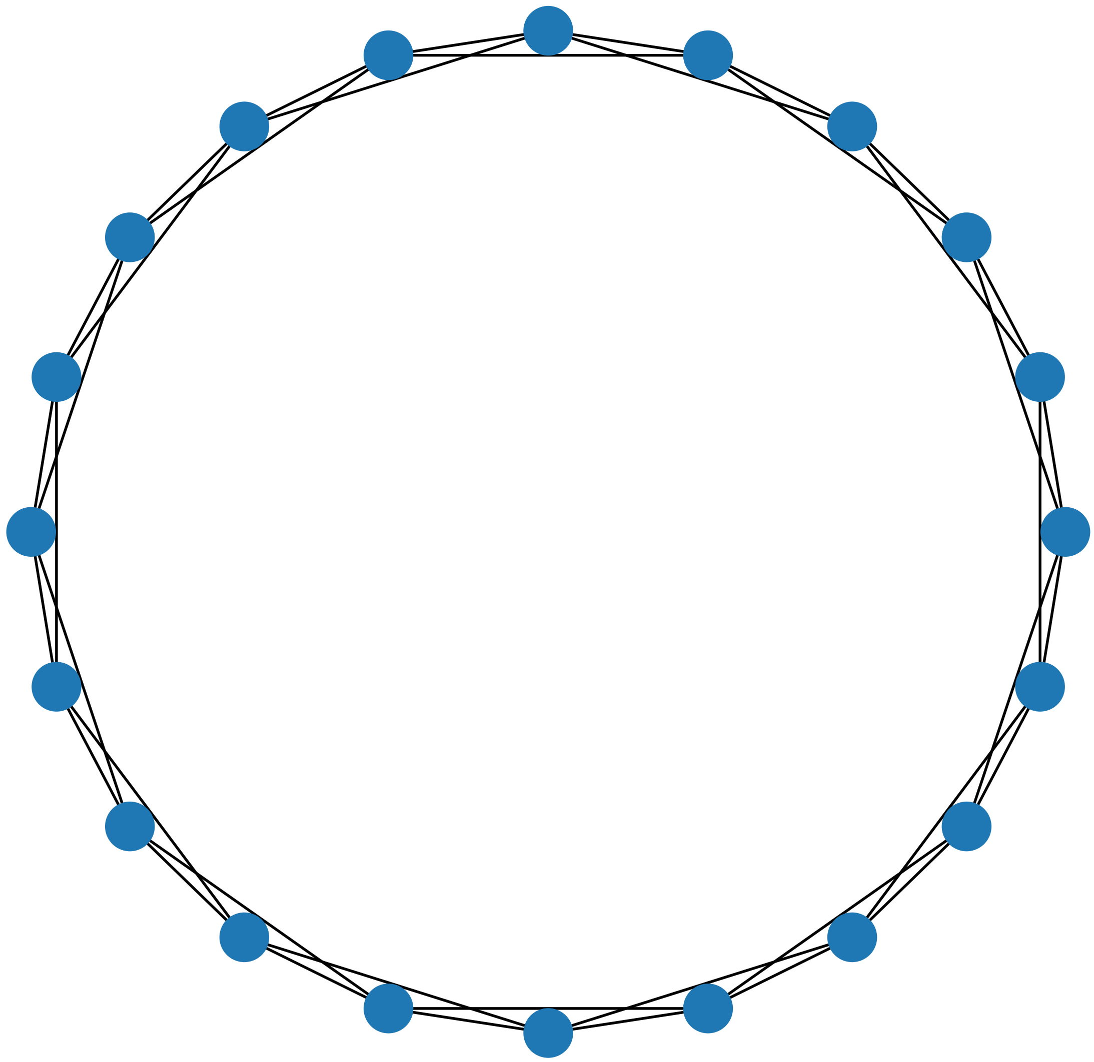
Figur 6: With parameter values of $n = 2000$ and $p = 0.05$. The histogram shows a similar distribution to a normal Gaussian distribution, and also similar to figure 5 in terms of shape.

Exercise 12.2

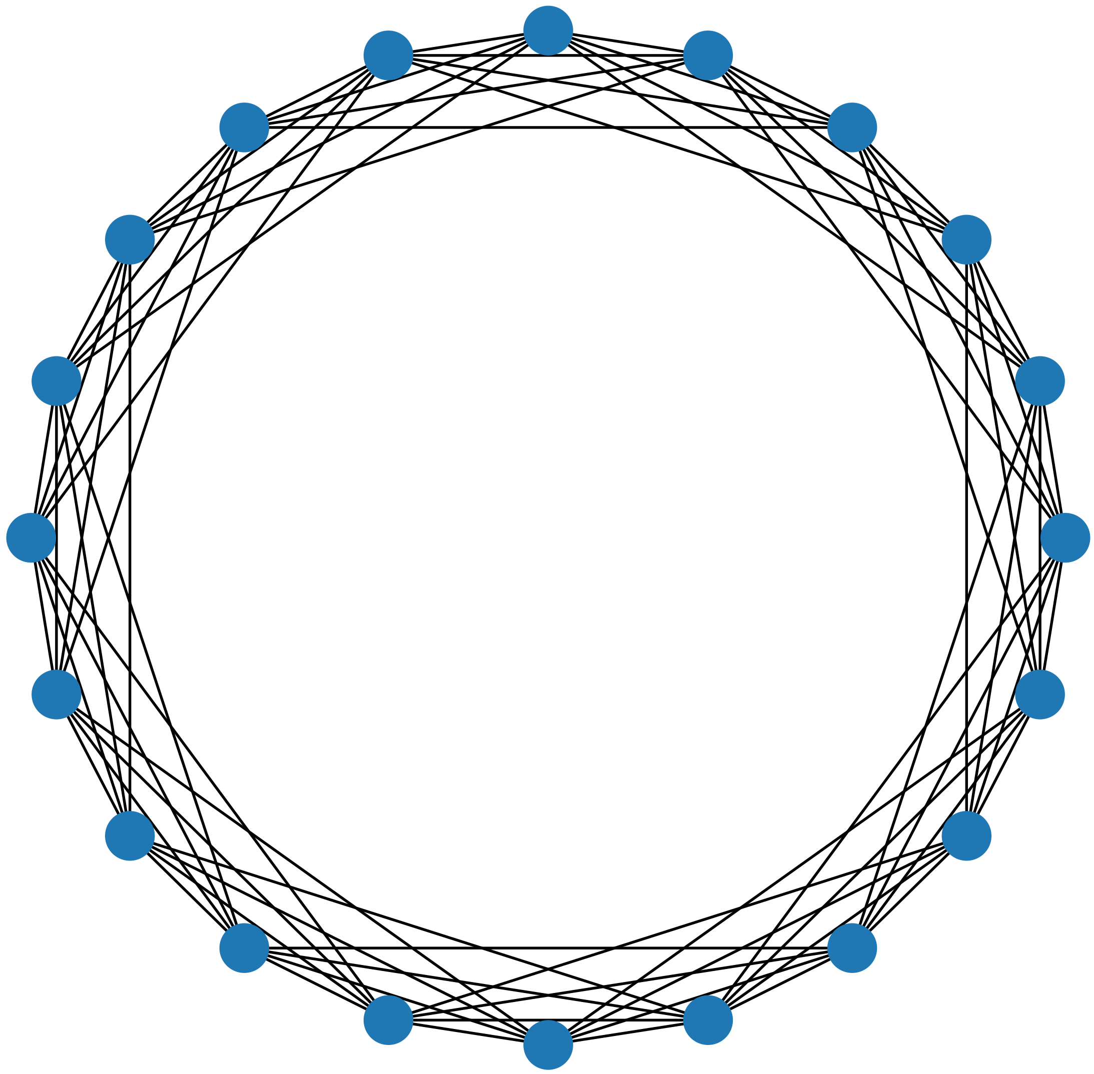
Subproblem (a)



Figur 7: With parameter values of $n = 20$, $c = 2$ and $p = 0$.

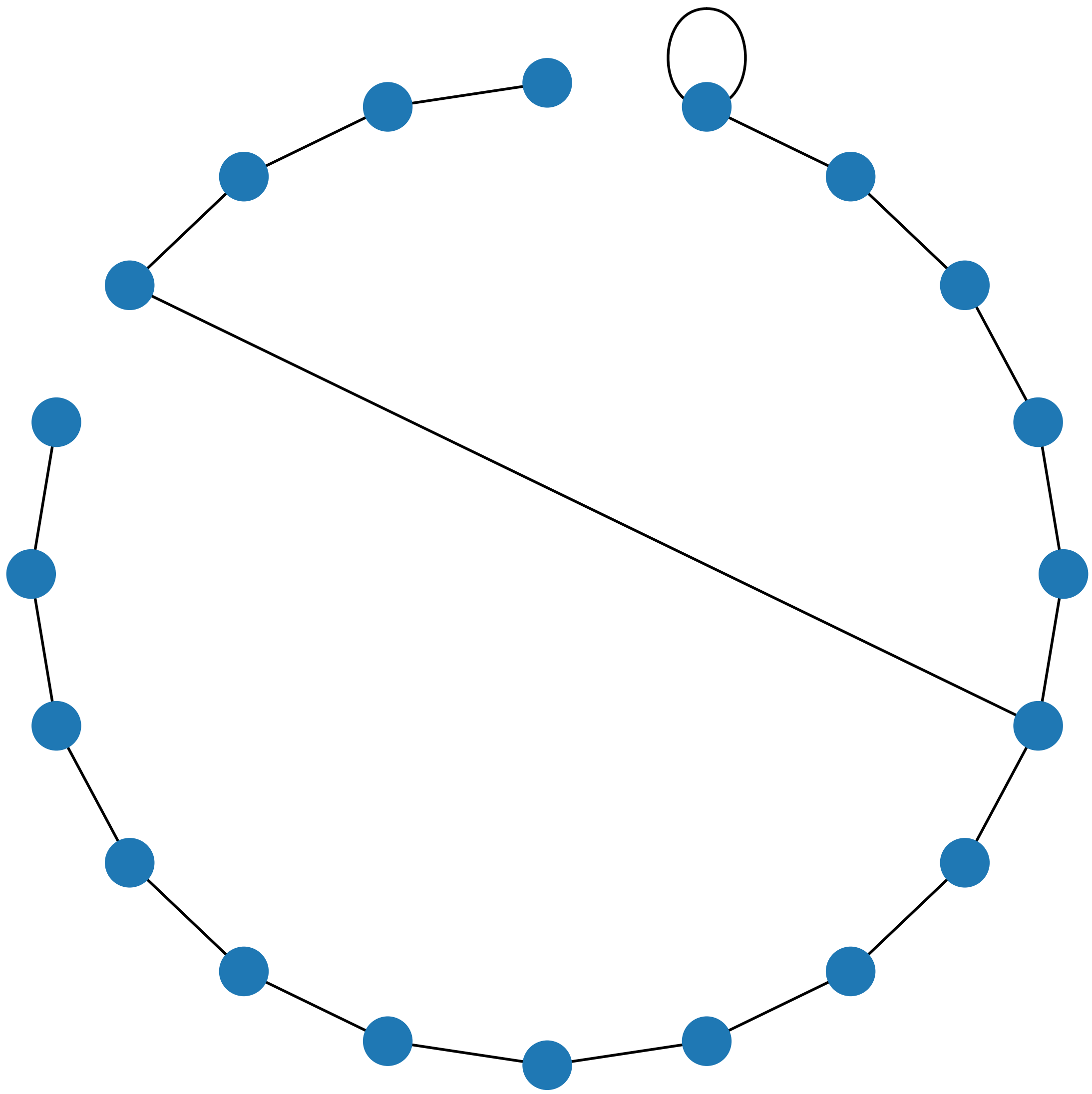


Figur 8: With parameter values of $n = 20$, $c = 4$ and $p = 0$.

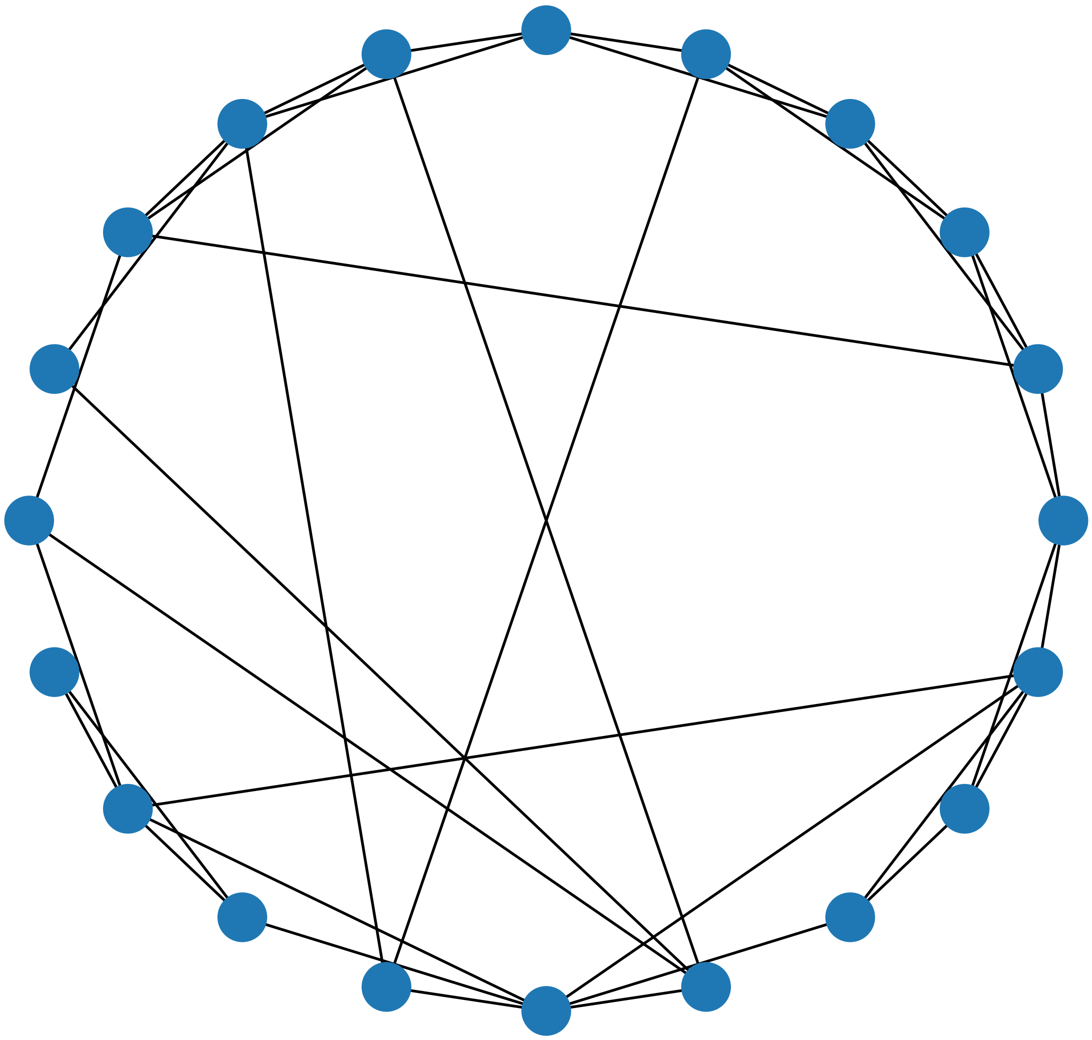


Figur 9: With parameter values of $n = 20$, $c = 8$ and $p = 0$.

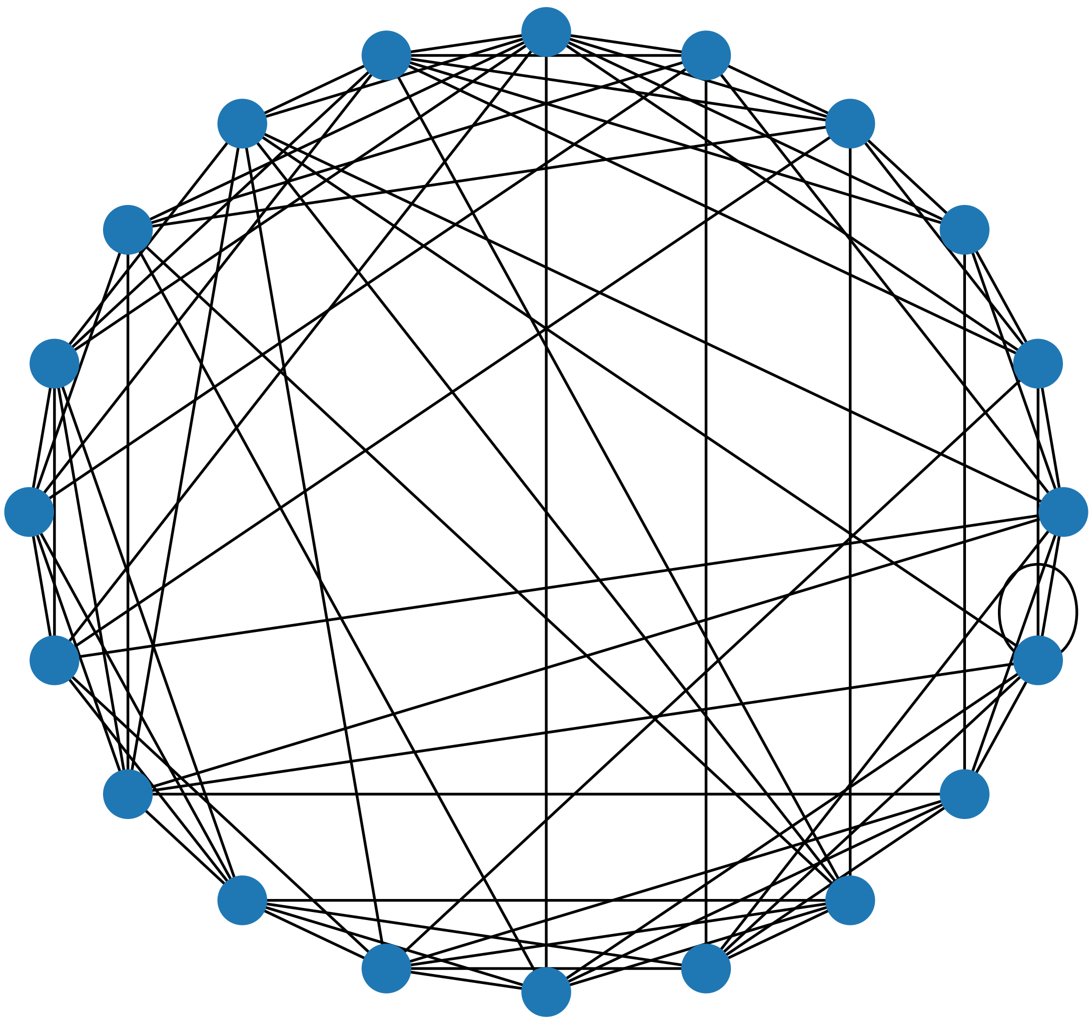
Subproblem (b)



Figur 10: With parameter values of $n = 20$, $c = 2$ and $p = 0.1$.



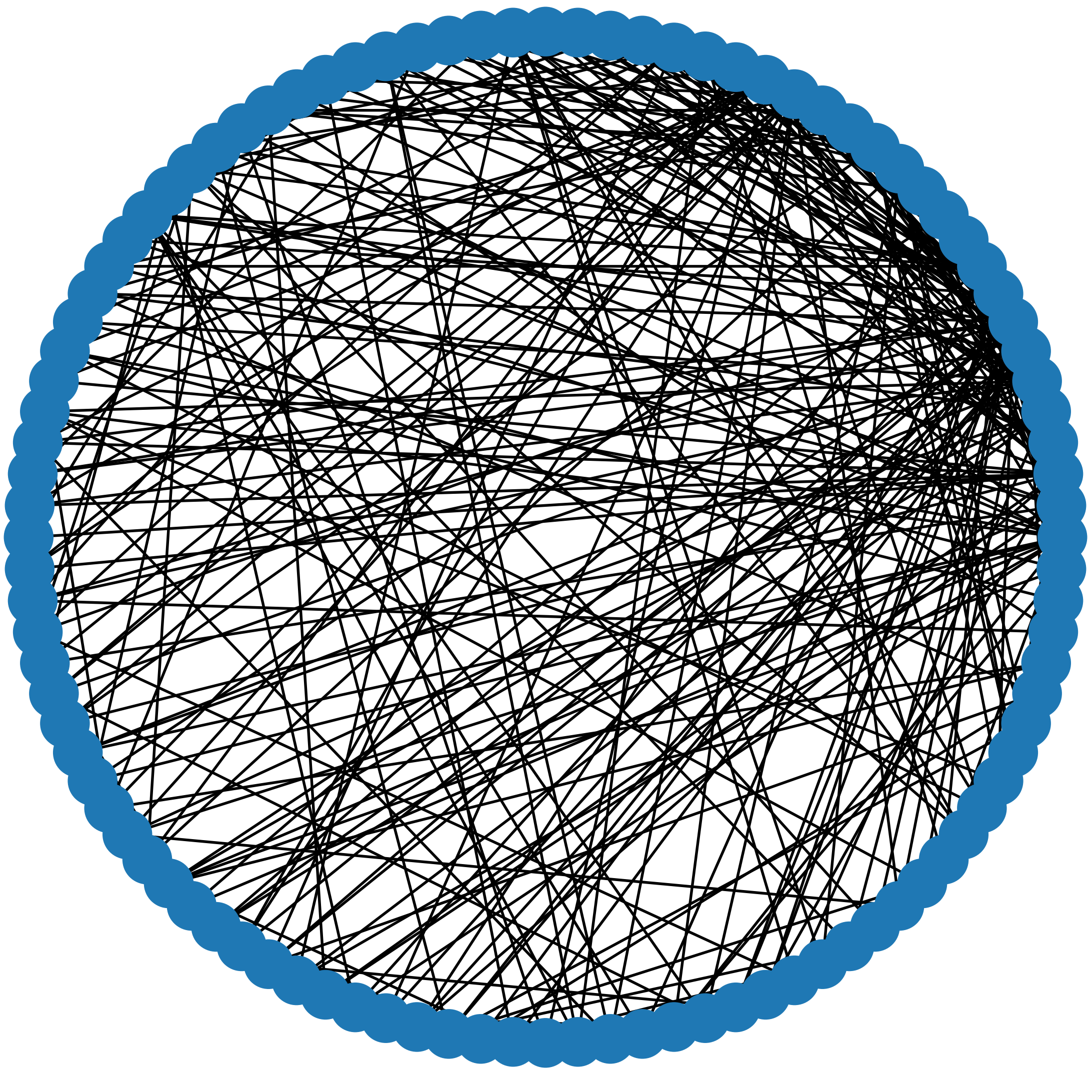
Figur 11: With parameter values of $n = 20$, $c = 4$ and $p = 0.3$.



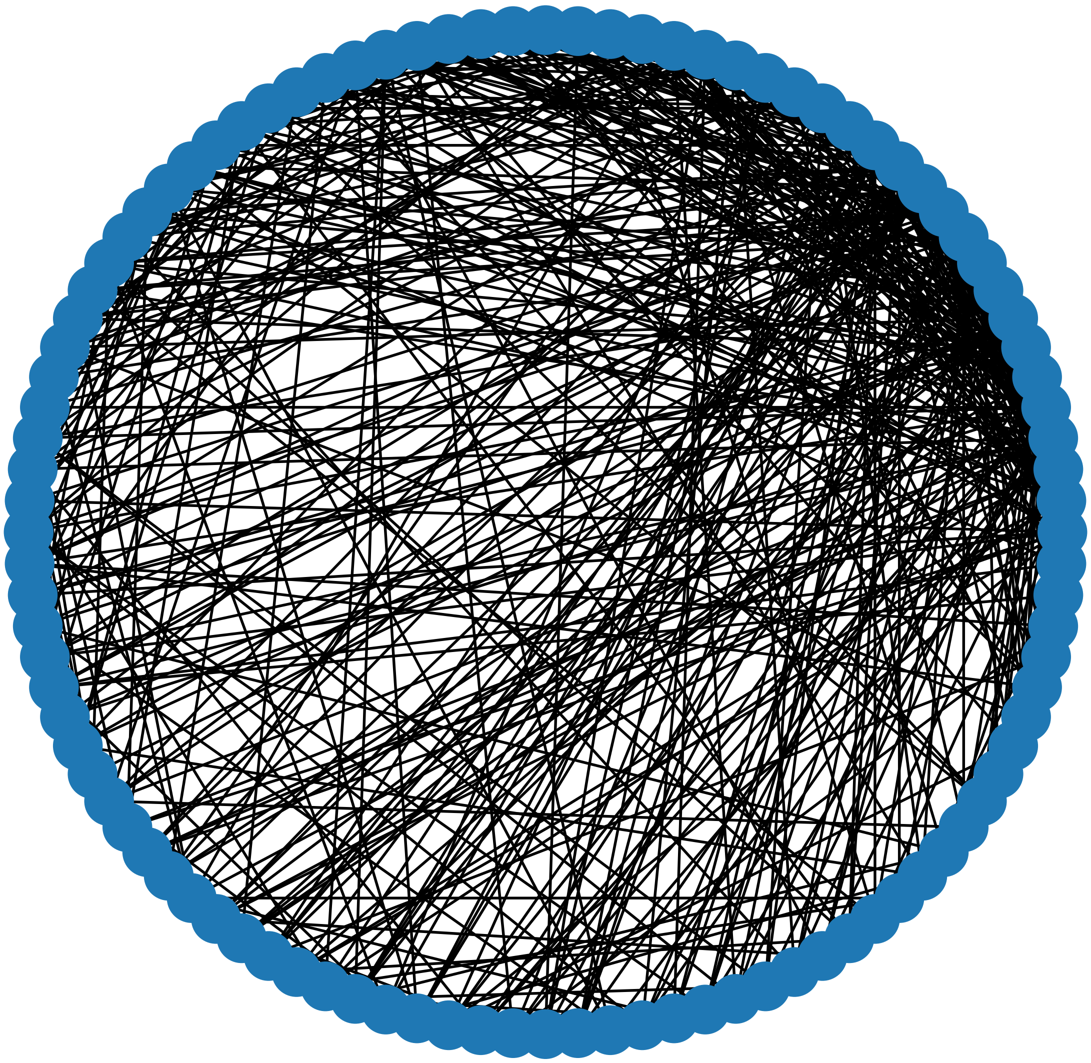
Figur 12: With parameter values of $n = 20$, $c = 8$ and $p = 0.4$.

Exercise 12.3

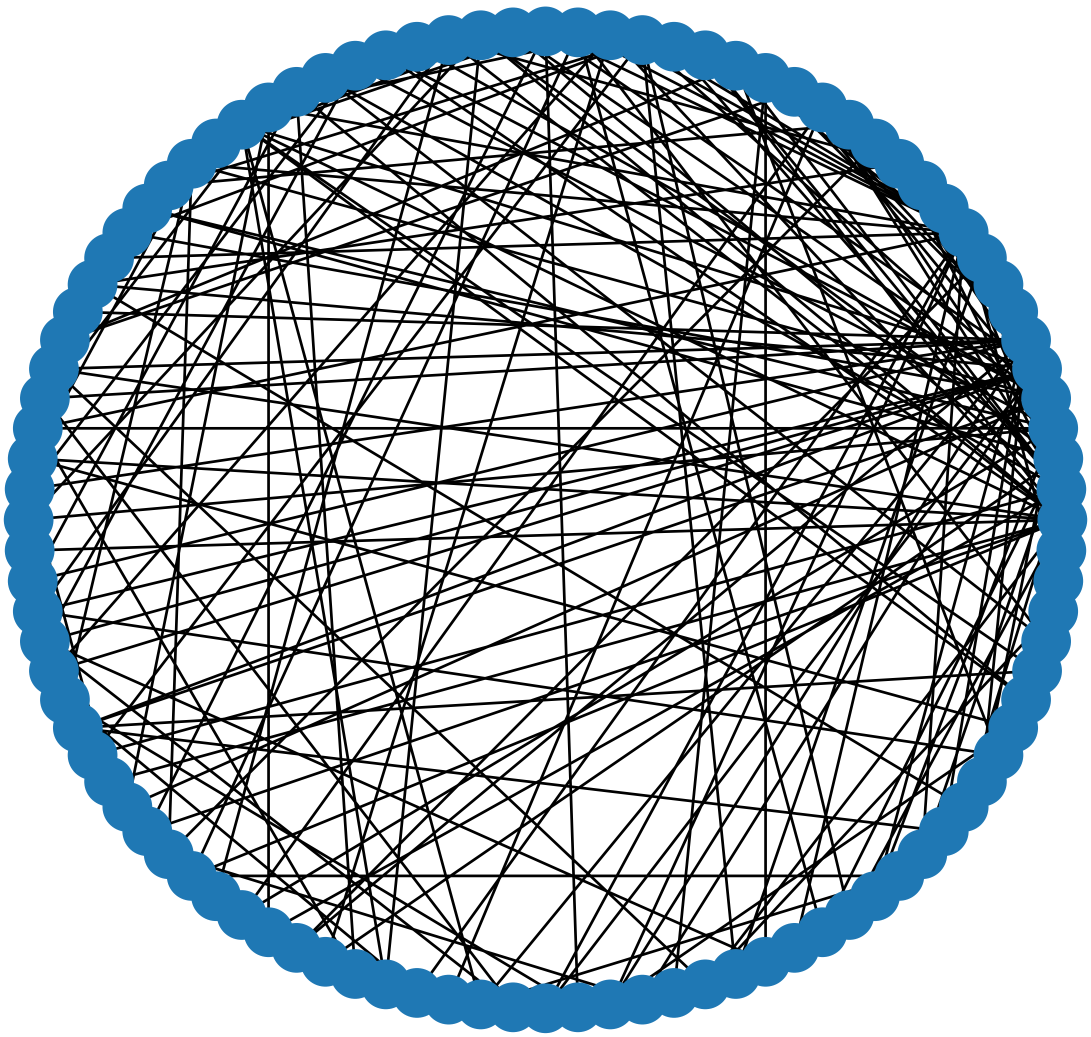
Subproblem (a)



Figur 13: With parameter values of $n = 100$, $n_0 = 10$ and $m = 3$.

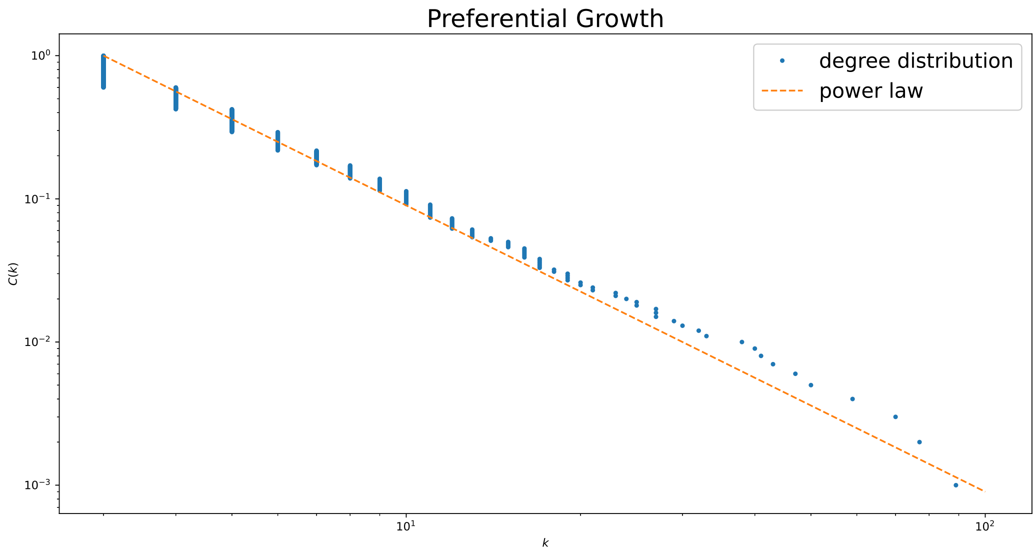


Figur 14: With parameter values of $n = 100$, $n_0 = 15$ and $m = 5$.



Figur 15: With parameter values of $n = 100$, $n_0 = 7$ and $m = 2$.

Subproblem (b)



Figur 16: With parameter values of $n = 1000$, $n_0 = 5$ and $m = 3$.

Code

12.1a

```
import networkx as nx
import numpy as np
from matplotlib import pyplot as plt


def random_graph(n,p):
    A = np.zeros((n,n))
    for i in range(n):
        for j in range(i+1,n):
            A[i,j] = np.random.rand()<p
            A[j,i] = A[i,j]

    return A


n = 100
p = 0.1
A = random_graph(n,p)
G = nx.from_numpy_matrix(A)
nx.draw_circular(G)
plt.show()
```

12.1b

```
import numpy as np
from matplotlib import pyplot as plt
import scipy.special


def random_graph(n,p):
    A = np.zeros((n,n))
    for i in range(n):
        for j in range(i+1,n):
            A[i,j] = np.random.rand()<p
            A[j,i] = A[i,j]
    return A


def probability(n,p,k):
    P = scipy.special.binom(n-1,k) * p ** k * (1-p)**(n-1-k)
    return P


n = 200
p = 0.2
xmax = 80
A = random_graph(n,p)
degrees = np.sum(A,1)
prob=np.zeros(xmax)
for k in range(xmax):
    prob[k]=probability(n,p,k)*100

plt.hist(degrees)
plt.plot(prob)
plt.xlim([0,xmax])
```

```
plt.title('Probability distribution')
plt.ylabel('$p(d)$')
plt.xlabel('$d$')
plt.rcParams.update({'font.size': 18})
plt.show()
```

12.1c

```
import numpy as np
from matplotlib import pyplot as plt
import scipy.special


def random_graph(n,p):
    A = np.zeros((n,n))
    for i in range(n):
        for j in range(i+1,n):
            A[i,j] = np.random.rand()<p
            A[j,i] = A[i,j]
    return A


n = 2000
p = 0.05
xmax = 200
A = random_graph(n,p)
degrees = np.sum(A,1)
#prob=np.random.normal(n)

plt.hist(degrees)
#plt.plot(prob)
plt.xlim([0,xmax])
plt.title('Probability distribution')
plt.ylabel('$p(d)$')
plt.xlabel('$d$')
plt.rcParams.update({'font.size': 18})
plt.show()
```

12.2

```
import networkx as nx
import numpy as np
from matplotlib import pyplot as plt
from random import sample


def smallworld_graph(n,c,p):
    A = np.zeros((n,n))
    for i in range(n):
        for j in range(int(c/2)):
            A[i,(i+j+1)%n] = 1  # periodic boundary condition, if it goes past the boundary n,
            if np.random.rand()<p:   # With probabilty p, do a rewire
                rewire_index = sample(list(np.where(np.logical_not(A[i,:])&np.logical_not(rang
                # rewire the edge to another node with no connection     and     not to itself
                # only condition for np.where, so it returns an array that shows which nodes t
                # nodes that can not be rewired to are nodes that are currently wired and the 
                # list puts these nodes in a list, sample choose a node that becomes the rewir
                A[i,(i+j+1)%n] = 0              # remove the existing edge
                A[i,rewire_index] = 1           # rewire the edge
```

```
    A = A + np.transpose(A)                      # make the adjacency matrix symmetric
    return A


n = 20
c = 4  # c is how many connections each node can have
p = 0.3
A = smallworld_graph(n,c,p)
G = nx.from_numpy_matrix(A)
nx.draw_circular(G)

plt.show()
```

12.3a

```
import networkx as nx
import numpy as np
from matplotlib import pyplot as plt


def preferentialgrowth(n, n0, m):
    A = np.zeros((n, n))
    for i in range(n0):  #
        for j in range(i + 1, n0):
            A[i, j] = 1
    A = A + np.transpose(A)
    for t in range(n - n0):
        D = np.sum(A, axis=0) / np.sum(A)
        edges = np.random.choice(np.arange(n), m, replace=False, p=D)
        for i in range(m):
            A[t + n0, edges[i]] = 1
            A[edges[i], t + n0] = 1
    return A


n = 100
n0 = 7
m = 2
A = preferentialgrowth(n,n0,m)
G = nx.from_numpy_matrix(A)
nx.draw_circular(G)
plt.show()
```

12.3b

```
import numpy as np
from matplotlib import pyplot as plt


def preferentialgrowth_graph(n, n0, m):
    A = np.zeros((n, n))
    for i in range(n0):  # start with n0 sized network, preallocating the full n sized adjacen
        for j in range(i + 1, n0):  # note that this avoids the diagonal i=j since j=i+1
            A[i, j] = 1
    A = A + np.transpose(A)  # symmetric
    for t in range(n - n0):  # creating n-n0 nodes and connect the node with m nodes (that may
        D = np.sum(A, axis=0) / np.sum(A)  # current degrees of the existing nodes???
        edges = np.random.choice(np.arange(n), m, replace=False, p=D)  # chooses m edges from
        # once a node is chosen, it can not be chosen again
```

```
        # the choice is based on the probability determined by the degree D
        for i in range(m):  # wires the randomly chosen edges at the specified nodes
            A[t + n0, edges[i]] = 1
            A[edges[i], t + n0] = 1
    return A


n = 1000
n0 = 5
m = 3  # m is the minimum number of nodes
A = preferentialgrowth_graph(n,n0,m)
degrees = np.sum(A,1)
plt.loglog(np.sort(degrees)[::-1],np.arange(n)/n,'.')
plt.loglog(np.arange(m,np.max(degrees)),m**2*np.arange(m,np.max(degrees))**(-2),'--')

plt.rcParams.update({'font.size': 18})
plt.title('Preferential Growth')
plt.legend(['degree distribution', 'power law'])
plt.ylabel('$C(k)$')
plt.xlabel('$k$')
plt.show()
```