ECS 253 / MAE 253, Network Theory and Applications
Spring 2016
Solution to Advanced Problem Set # 5b
Topic: Generating Functions

1 Warming up

Suppose $f(x)$ is an ordinary generating function generating the sequence $(\phi_k)_{k=0}^{\infty}$, i.e.,

$$f(x) := \sum_{k=0}^{\infty} \phi_k x^k. \tag{1}$$

We use the notation $[x^n]f(x)$ to represent the coefficient multiplying x^n in the power series of $f(x)$ (in this case $[x^n]f(x) = \phi_n$). We say that $f(x)$ is a probability generating function (PGF) if the coefficients ϕ_k are to be interpreted as the probability $\mathbb{P}(K = k)$ that the random variable K takes the value k (and thus $0 \leq \phi_k \leq 1$ for all k). We say that a PGF $f(x)$ is normalized if $\sum_{k=0}^{\infty} \phi_k = 1$.

(a) Show that $[x^n]f(x) = \dfrac{1}{n!} \left. \dfrac{\mathrm{d}^n f(x)}{\mathrm{d}x^n} \right|_{x=0}$.

The case $n = 0$ holds: $[x^0]f(x) = \dfrac{1}{0!} \left(\phi_0 + x \sum_{k=1}^{\infty} \phi_k x^{k-1} \right) \Bigg|_{x=0} = \phi_0$.

Now suppose that the relation holds for n and consider the the right hand side in the case of $n+1$

$$\begin{aligned}
\frac{1}{(n+1)!} \left. \frac{\mathrm{d}^{n+1} f(x)}{\mathrm{d}x^{n+1}} \right|_{x=0} &= \frac{1}{(n+1)!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \frac{\mathrm{d}}{\mathrm{d}x} \sum_{k=0}^{\infty} \phi_k x^k \Bigg|_{x=0} = \frac{1}{(n+1)!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \sum_{k=0}^{\infty} k \phi_k x^{k-1} \Bigg|_{x=0} \\
&= \frac{1}{(n+1)} \frac{1}{n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \sum_{k=1}^{\infty} (k+1) \phi_{k+1} x^k \Bigg|_{x=0} = \frac{1}{(n+1)} \left([x^n] \sum_{k=1}^{\infty} (k+1) \phi_{k+1} x^k \right) \\
&= \frac{\cancel{(n+1)} \phi_{k+1}}{\cancel{(n+1)}} = [x^{n+1}] f(x).
\end{aligned}$$

The relation thus holds by mathematical induction.

(b) Assuming that $f(x)$ is a normalized PGF, show $f(1) = 1$.
Substituting $x = 1$ in Eq. (1) gives $f(1) = \sum_{k=0}^{\infty} \phi_k 1^k = \sum_{k=0}^{\infty} \phi_k = 1$. The last equality holds by the definition of a normalized PGF.

(c) Assuming that $f(x)$ is a normalized PGF, show $\mathbb{E}(K) := \sum_{k=0}^{\infty} k\mathbb{P}(K = k) = \left. \dfrac{\mathrm{d}\, f(x)}{\mathrm{d}x} \right|_{x=1}$.

Substitution gives

$$\left. \frac{\mathrm{d}\, f(x)}{\mathrm{d}x} \right|_{x=1} = \frac{\mathrm{d}}{\mathrm{d}x} \sum_{k=0}^{\infty} \phi_k x^k \Bigg|_{x=1} = \sum_{k=0}^{\infty} k \phi_k x^{k-1} \Bigg|_{x=1} = \sum_{k=0}^{\infty} k \phi_k = \sum_{k=0}^{\infty} k\mathbb{P}(K = k) = \mathbb{E}(K).$$

(d) Assuming that $f(x)$ is a normalized PGF, find an expression in terms of derivatives of $f(x)$ for $\mathbb{E}(K^2) := \sum_{k=0}^{\infty} k^2 \mathbb{P}(K = k)$.

First notice that

$$\left.\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2}\right|_{x=1} = \left.\sum_{k=0}^{\infty} k(k-1)\phi_k x^{k-2}\right|_{x=1} = \sum_{k=0}^{\infty} k^2\mathbb{P}(K=k) - \sum_{k=0}^{\infty} k\mathbb{P}(K=k) = \mathbb{E}(K^2) - \mathbb{E}(K).$$

Hence, $\mathbb{E}(K^2) = \left.\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2}\right|_{x=1} + \mathbb{E}(K) = \left.\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2}\right|_{x=1} + \left.\frac{\mathrm{d}\, f(x)}{\mathrm{d}x}\right|_{x=1}$.

(e) Suppose that $\sum_{k=0}^{\infty}\mathbb{P}(K=k) = 1 - \mathbb{P}(K \text{ is infinite})$. If $\mathbb{P}(K \text{ is infinite}) > 0$, then the PGF $f(x)$ for the probability distribution $\mathbb{P}(K=k)$ is not normalized. Find the normalized PGF for the probability sequence $\mathbb{P}(K=k|K \text{ is finite})$. Also find the value for $\mathbb{P}(K \text{ is infinite})$.
From $\sum_{k=0}^{\infty}\mathbb{P}(K=k) = 1 - \mathbb{P}(K \textit{ is infinite})$, *we obtain*

$$\mathbb{P}(K \textit{ is infinite}) = 1 - \sum_{k=0}^{\infty}\mathbb{P}(K=k) = 1 - \sum_{k=0}^{\infty}\phi_k 1^k = 1 - f(1).$$

We know that K *is either finite or infinite, so* $\mathbb{P}(K \textit{ is finite}) + \mathbb{P}(K \textit{ is infinite}) = 1$ *and thus* $\mathbb{P}(K \textit{ is finite}) = f(1)$. *We also know that* $\mathbb{P}(K=k, K \textit{ is finite}) = \mathbb{P}(K=k)$ *for all finite values of* k, *which gives*

$$\mathbb{P}(K=k|K \textit{ is finite}) = \frac{\mathbb{P}(K=k, K \textit{ is finite})}{\mathbb{P}(K \textit{ is finite})} = \frac{\mathbb{P}(K=k)}{f(1)} \quad \textit{for finite } k.$$

The sought normalized PGF is thus $\dfrac{f(x)}{f(1)} = \displaystyle\sum_{k=0}^{\infty}\mathbb{P}(K=k|K \textit{ is finite})x^k$.

2 Percolation: some analytical results

An infinite configuration model random graph has its degree distribution specified by $(p_k)_{k=0}^{\infty}$ (i.e., a node sampled uniformly at random has probability p_k to have degree k). In the course, you have seen the following two generating functions

$$g_0(x) := \sum_{k=0}^{\infty} p_k x^k \tag{2a}$$

$$g_1(x) = \sum_{k=0}^{\infty} q_k x^k = \frac{g_0'(x)}{g_0'(1)} \qquad \left(\text{with } q_k := \frac{(k+1)p_{k+1}}{\sum_{k'=0}^{\infty} k' p_{k'}}\right), \tag{2b}$$

and you have seen that q_k is the probability that a node reached by following a random edge has k *other* edges than the one we followed (and thus a total degree $k+1$). You then obtained the following PGFs

$$h_1(x) := (1-T) + Txg_1\big(h_1(x)\big) \tag{2c}$$

$$h_0(x) := xg_0\big(h_1(x)\big) \tag{2d}$$

and saw that the probability of reaching r nodes by following a random edge is $[x^r]h_1(x)$, and that the probability of reaching s nodes by starting at a node selected uniformly at random (and including that node) is $[x^s]h_0(x)$. The equation you saw in the course had $T=1$. The parameter T here is the same as you saw in the last homework: it may be interpreted as either the probability for an edge of the original

configuration model to be present in the percolated one, or as the probability for a spreading process to spread along an edge when it encounters it. Equation (2c) may thus be interpreted as follows: with probability $1-T$, the edge is not followed and there should be no powers of x contributing here, and with probability T the edge is followed normally (hence the term $xg_1\big(h_1(x)\big)$).

In the case $T = 1$, you saw that the network contains a giant component when $g_1'(x) > 1$. In the more general case where $0 \leq T \leq 1$, there may be no giant component even if $g_1'(x) > 1$. In fact, there is a critical value of T, noted T_c, over which a giant component exists. Hence, for $T \leq T_c$, there are only small components and the PGF $g_0(x)$ should thus be normalized. Moreover, the average size of the small components should diverge when $T = T_c$.

(a) Differentiate both sides of Eq. (2c) and solve for $h_1'(1)$. Find T_c in terms of the average degree $\langle k\rangle := \mathbb{E}(K)$ and the second moment $\langle k^2\rangle := \mathbb{E}(K^2)$.
Differentiation gives

$$h_1'(x) = Tg_1\big(h_1(x)\big) + Txg_1'\big(h_1(x)\big)h_1'(x).$$

Under the assumption that $T \leq T_c$, we have $h_1(1) = 1$ because the probability that a finite number of nodes is reached is 1. Hence, we have

$$h_1'(1) = Tg_1\big(h_1(1)\big) + Tg_1'\big(h_1(1)\big)h_1'(1) = Tg_1(1) + Tg_1'(1)h_1'(1) \quad \text{for } T \leq T_c,$$

which gives

$$h_1'(1) = \frac{Tg_1(1)}{1 - Tg_1'(1)} \quad \text{for } T \leq T_c.$$

We can see that $h_1'(1)$ diverges when $T = \frac{1}{g_1'(1)}$. Notice that the expected number of reached nodes for $T \leq T_c$ is $h_0'(1) = 1 + g_0'(1)h_1'(1)$, which diverges if $h_1'(1)$ diverges. Hence, this divergence point is the threshold $T_c = \frac{1}{g_1'(1)} = \frac{g_0'(1)}{g_0''(1)} = \frac{\langle k\rangle}{\langle k^2\rangle - \langle k\rangle}$ over which a giant component exists.

(b) What is T_c if the degree distribution follows a power law $p_k \propto k^{-\gamma}$ with $\gamma = 2.5$?
In this case, $\langle k^2\rangle$ diverges to infinity, so $T_c = 0$. This means that as long as T is nonzero, a giant component exists.

(c) Suppose that the highest degree present in the network is 3 (i.e., only p_0, p_1, p_2 and p_3 may be nonzero). Obtain a closed form for $h_0(x)$. You will need the quadratic formula to obtain $h_1(x)$.
We have

$$\begin{aligned} h_1(x) &= (1-T) + Tx\left(q_0 + q_1h_1(x) + q_2[h_1(x)]^2\right) \\ 0 &= \underbrace{Txq_2}_{a(x)}[h_1(x)]^2 + \underbrace{(Txq_1 - 1)}_{b(x)}h_1(x) + \underbrace{(1 - T + Txq_0)}_{c(x)} = a(x)[h_1(x)]^2 + b(x)h_1(x) + c(x), \end{aligned}$$

and thus

$$h_1(x) = \frac{-b(x) - \sqrt{b(x)^2 - 4a(x)c(x)}}{2a(x)}.$$

The negative root has been selected because we know that $h_1(1) = 1$ for $T \leq T_c$. Hence, $h_0(x)$ is

$$h_0(x) = xp_0 + xp_1h_1(x) + xp_2[h_1(x)]^2 + xp_3[h_1(x)]^3,$$

where $h_1(x)$ is defined above.

(d) Obtain T_c in the case $p_0 = 0$, $p_1 = 0.2$, $p_2 = 0.5$, $p_3 = 0.3$, and $p_k = 0$ for $k > 3$. Obtain $h_0(1)$ for $T = 0.70$, $T = 0.75$ and $T = 0.80$. What is the size of the giant component (if any) in each of these cases?

For these values, we have

$$q_0 = \frac{2}{21} \qquad q_1 = \frac{10}{21} \qquad q_3 = \frac{9}{21}$$

and thus

$$T_c = \frac{1}{h_1'(1)} = \frac{1}{0 \cdot \frac{2}{21} + 1 \cdot \frac{10}{21} + 2 \cdot \frac{9}{21}} = \frac{3}{4} = 0.75.$$

Substitution in the equations above gives

$$h_0(1)|_{T=0.70} = 1 \qquad h_0(1)|_{T=0.75} = 1 \qquad h_0(1)|_{T=0.80} = 0.80\overline{5}.$$

There is no giant component in the cases $T = 0.70$ and $T = 0.75$. There is a giant component for $T = 0.80$, and it encompasses a fraction $1 - h_0(1)|_{T=0.80} = 0.19\overline{4}$ of the nodes.

3 Percolation: semi-analytical results

(a) Dust-off the code you created for exercise 2(e) of the last homework. Using the parameters `number_simulatio` $= 10000$, $\mathbf{n} = (0, 200, 500, 300)$, and $\mathcal{A}$, run your code to estimate the probability distribution for the number s of reached nodes for $T = 0.70$, $T = 0.75$ and $T = 0.80$. In each case, create a log-log graph showing the probability as a function of the number of reached nodes s. Use small markers without lines joining them.

(b) Suppose $p_0 = 0$, $p_1 = 0.2$, $p_2 = 0.5$, $p_3 = 0.3$, and $p_k = 0$ for $k > 3$. Use the DFT method (with $M = 1001$) to extract the coefficients $[x^s]h_0(x)$ from the solution you obtained in exercise 3 in the following three cases: $T = 0.70$, $T = 0.75$ and $T = 0.80$. Display your results on the same three log-log plots as in exercise 1, this time using a plain thin line without markers.

(c) (This part was not required, just FYI) Usually, you will not have access to such an analytical solution for $h_0(z)$. Fortunately, you can understand the recurrence equation (2c) as if it were a dynamical system. Indeed, for a given z, you can estimate $h_1(z)$ as using $h_1^{(L)}(z)$ defined as follows: $h_1^{(0)}(z) = 0$, and $h_1^{(L+1)}(z) = (1 - T) + Tzg_1\big(h_1^{(L)}(z)\big)$. You can then estimate $h_0(z)$ using $h_0^{(L)}(z) = zg_0\big(h_1^{(L)}(z)\big)$.

- Create a function receiving T, $(p_k)_{k=0}^K$ and L and returning $h_0^{(L)}(z)$.
- Use the DFT method to extract the coefficients $[z^n]h_0^{(10)}$ for $p_0 = 0$, $p_1 = 0.2$, $p_2 = 0.5$, $p_3 = 0.3$, and $p_k = 0$ for $k > 3$, and in the three cases $T = 0.70$, $T = 0.75$ and $T = 0.80$. Report these results on the same 3 plots as before, this time using a doted line.
- Do the same for $[z^n]h_0^{(100)}$, this time using a wide dashed line.

See Figs. 1–3 and the code at the end of this document.

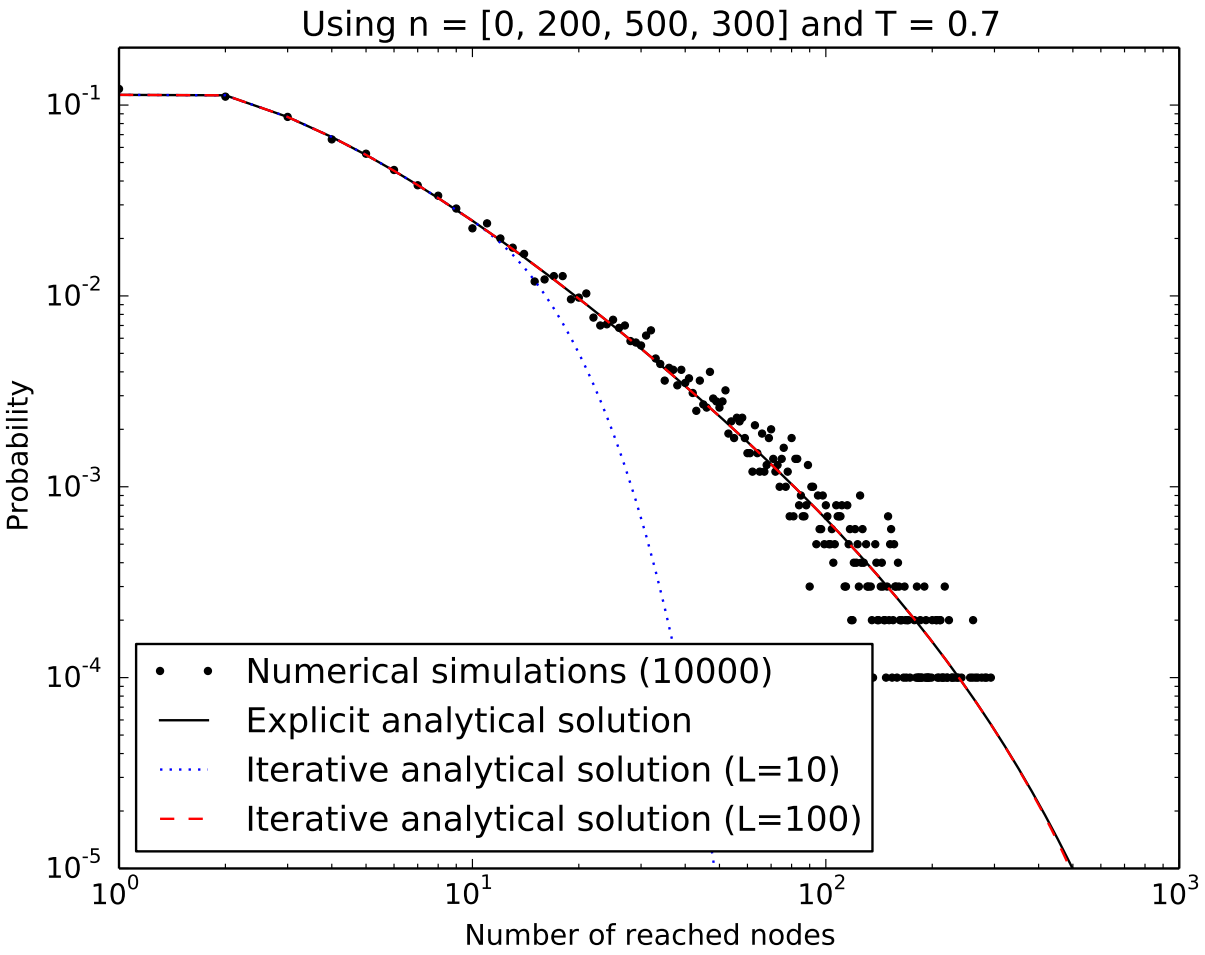

Figure 1: Solution for Problem 4 in the case $T = 0.70$. In this case, $T < T_c$, so the numerical and analytical results agree pretty well, with the exception of the iterative method using $L = 10$. In this last case, the iterative process amounts to say that the spreading stops after 10 jumps, hence the cutoff tail. The beginning of the distribution is fine, though, and in fact the first 10 points are "exact" up to numerical precision.

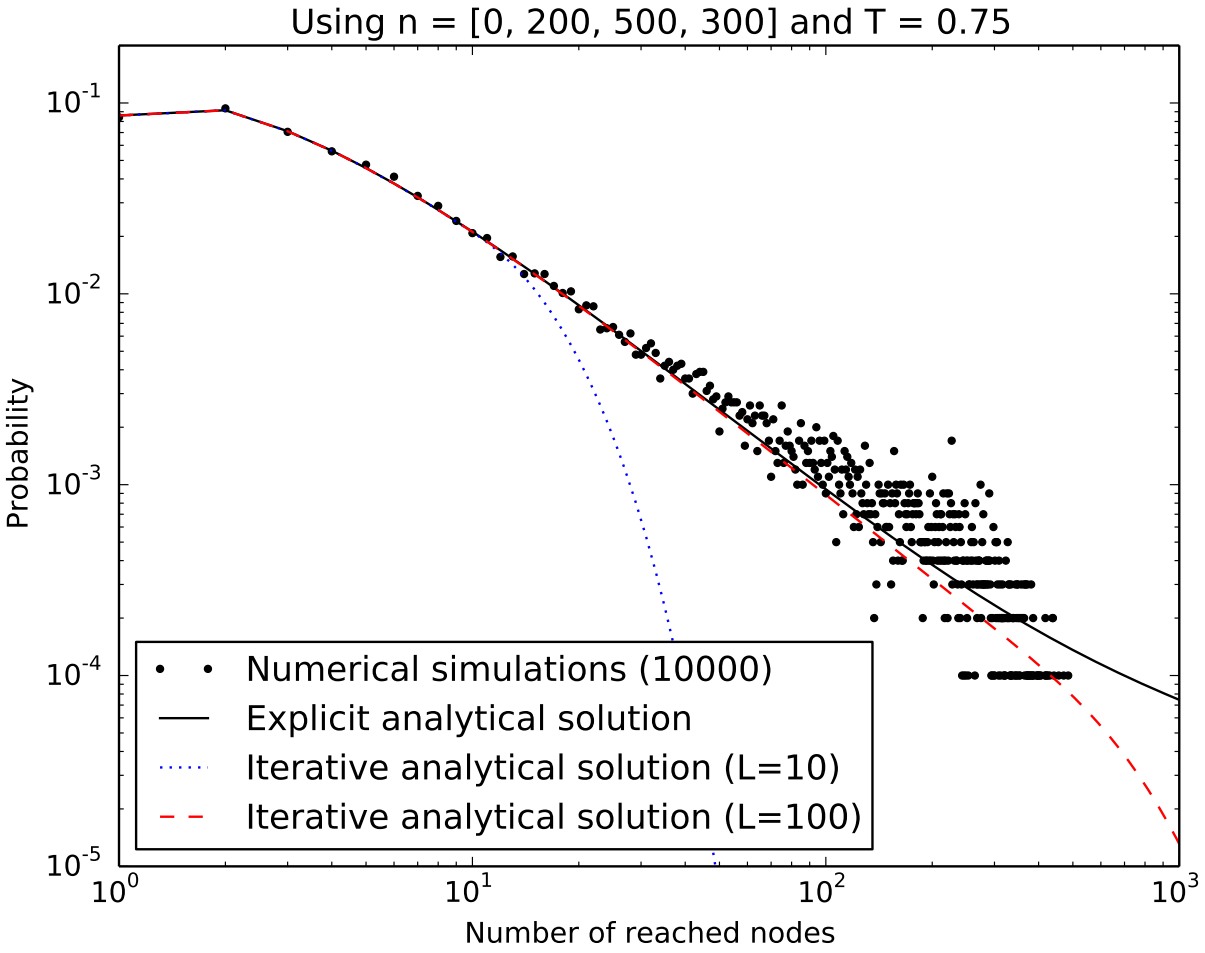

Figure 2: Solution for Problem 4 in the case $T = 0.75$. This is the critical case $T = T_c$, and the four curves all show different behavior. First, as expected, the iterative approach with $L = 10$ shows a similar cutoff as in Fig. 1. More surprisingly, the iterative approach with $L = 100$ shows a similar cutoff, although appearing later on: this is due to a critical slowing down of the convergence of this iterative process. The explicit analytical solution shows the expected power-law behavior typical of such critical cases, except that the curve appears to curve up at the end. This is due to aliasing introduced by the FFT algorithm: the "actual" analytical probability distribution falls according to a power law, but the values for sizes higher than 1000 get summed to those in the range $[0, 1000]$. This could be mitigated by using an M greater than 1000 (say 10000), then by discarding all values past 1000. Finally, the fact that the simulations appear to "bunch" around 300 is a finite size effect due to the "exhaustion" of degree 3 nodes. This may be better understood by pushing the phenomenon to the extreme: there are only 1000 nodes in the network, so there cannot be more than 1000 reached nodes. Our analytical results do not predict such finite size effects because they assume an infinite network.

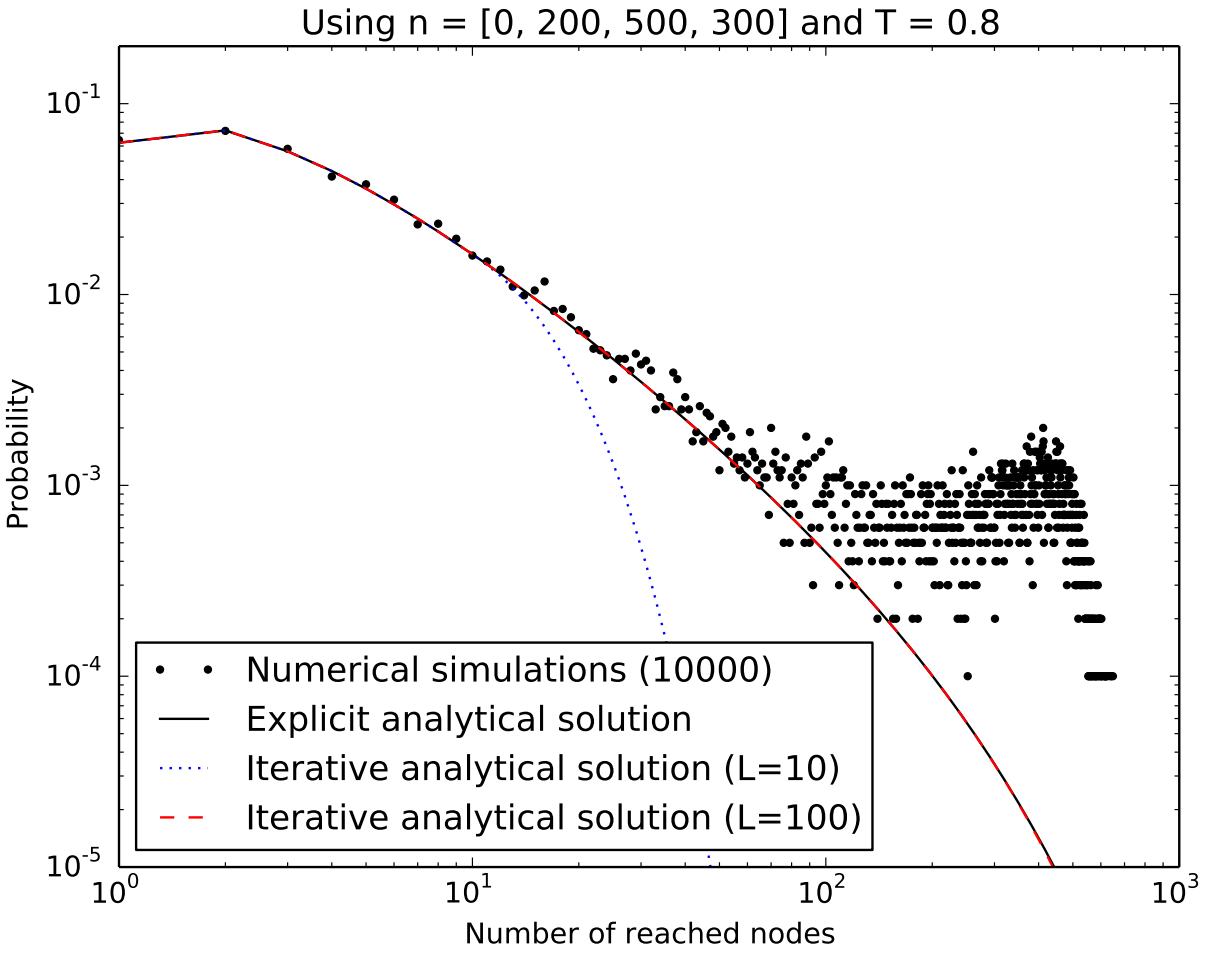

Figure 3: Solution for Problem 4 in the case $T = 0.80$. Here we are in the supercritical regime $T > T_c$. The iterative approach with $L = 100$ now agrees with the explicit analytical solution since we are past the "critical slowing down" regime. The numerical simulations agree with these two curves for smaller sizes, but they strongly disagree for large sizes: these are giant components. If we were to repeat the experiment with a network 10 times larger (i.e., $\mathbf{n} = (0, 2000, 5000, 3000)$), then the beginning of the distribution would stay unchanged, and the giant components would move by one decade to the right (hence making a crisper distinction between small and large components).

Content of homework_5b.py

```
"""
This is part of the answer to Homework 5b.
"""

import scipy as sp
import matplotlib.pyplot as plt
from itertools import repeat
from solution_3b import spreading, bin_many_simulations

def extract_coefficients(f, M, r=1):
    """
    Extract the coefficients of a function f(x).

    'M' is the number of points used in the FFT, and 'r' is the radius of the
    circular path in the complex plane.

    This is an implementation of the method developed in Problem 4.
    """
    if r == 1: #Special case for r=1 (faster and less numerical error).
        return sp.fft(f(sp.exp(2*sp.pi*1j*sp.arange(M)/M)))/M
    else: #Generic case.
        return sp.array([r])**(-sp.arange(M))*sp.fft(
            f(r*sp.exp(2*sp.pi*1j*sp.arange(M)/M)))/M

class ConfModPGF:
    """
    Provides PGFs for the configuration model random graph.
    """
    def __init__(self,p):
        assert( abs(1.0-sum(p)) < 1e-14 )
        #To simplify the explicit solution, we want 'p' have degree 3 or more.
        if len(p) < 4:
            p = list(p) + [0]*(4 - len(p))
        #Store p and q.
        self.p = sp.array(p)
        self.q = sp.arange(1,len(p))*self.p[1:] #Not done with q yet!
        self.q = self.q/sum(self.q) #OK, done with q now.
        #For polynomials, we want a version with the highest power first.
        self.p_poly = self.p[::-1]
        self.q_poly = self.q[::-1]
    def g0(self,x):
        """Evaluate g_0(x)."""
        return sp.polyval(self.p_poly,x)
    def g1(self,x):
```

```
        """Evaluate g_1(x)."""
        return sp.polyval(self.q_poly,x)
    #***For highest degree 3.***
    def h1_explicit(self,x,T=1.0):
        """Evaluate h_1(x) explicitly (part of Problem 4c)."""
        assert(len(self.p) == 4) #Only work if highest degree is 3.
        a = T*x*self.q[2]
        b = T*x*self.q[1]-1.0
        c = 1.0 - T + T*x*self.q[0]
        return (-b - sp.sqrt(b**2-4*a*c))/(2*a)
    def h0_explicit(self,x,T=1.0):
        """Evaluate h_0(x) explicitly (part of Problem 4c)."""
        h1 = self.h1_explicit(x,T)
        return x*self.g0(h1)
    #***For arbitrary highest degree.***
    def h1_iterative(self,x,T=1.0,L=100):
        """Evaluate h_1(x) iteratively (see Problem 5c)."""
        h1_tmp = 0.0*x
        for _ in repeat(None,L):
            h1_tmp = 1-T + T*x*self.g1(h1_tmp)
        return h1_tmp
    def h0_iterative(self,x,T=1.0,L=100):
        """Evaluate h_0(x) iteratively (see Problem 5c)."""
        h1 = self.h1_iterative(x,T,L)
        return x*self.g0(h1)

if __name__ == '__main__':

    #Set some parameters.
    n = [0,200,500,300]
    number_simulations = 10000
    T_to_try = [0.70, 0.75, 0.80]

    for T in T_to_try:
        #Run the simulations.
        simulation = bin_many_simulations(n,T,spreading,number_simulations)
        #Get analytical results
        conf_mod_pgf = ConfModPGF(sp.float_(n)/sum(n))
        analytic_explicit = extract_coefficients(
            lambda x:conf_mod_pgf.h0_explicit(x,T), sum(n)+1)
        analytic_iterative10 = extract_coefficients(
            lambda x:conf_mod_pgf.h0_iterative(x,T,10), sum(n)+1)
        analytic_iterative100 = extract_coefficients(
            lambda x:conf_mod_pgf.h0_iterative(x,T,100), sum(n)+1)
        #Make a plot of the results.
        x_axis_values = list(range(sum(n)+1))
        plt.loglog(x_axis_values,simulation,'k.',
```

```
            x_axis_values, analytic_explicit, 'k-',
            x_axis_values, analytic_iterative10, 'b:',
            x_axis_values, analytic_iterative100, 'r--')
plt.xlabel('Number of reached nodes')
plt.ylabel('Probability')
plt.ylim([1e-5,.2])
plt.legend(['Numerical simulations ('+str(number_simulations)+')',
             'Explicit analytical solution',
             'Iterative analytical solution (L=10)',
             'Iterative analytical solution (L=100)'], loc=3)
plt.title("Using n = " + str(n) + " and T = " + str(T))
plt.savefig("probabilities_spread_with_fft_T"+str(int(T*100+1e-14))
            +".pdf",format='pdf')
plt.clf()
```