

Random network (Erdős-Rényi model)

Adjacency matrix 작성과 degree distribution 확인

사회과학을 위한 행위자기반모형: 3주차 과제 (2026.9.19)

주혜신 (20265481)

목차

- 0단계. ER random network 이해하기
- 1단계. Network를 Adjacency Matrix로 작성하기
- 2단계. Degree (차수) 계산하기
- 3단계. Degree Distribution 그리기
- 4단계. 결과 해석 · Barabasi scale-free network와 비교

과제의 목적과 요건

ER random network

과제의 목적

1. Network를 그림이 아니라 adjacency matrix로 표현하는 방법
2. 그 배열에서 각 node의 degree를 계산하고 degree distribution을 그리는 방법
3. 시뮬레이션 결과를 이론적 예측과 비교하고 Barabasi scale-free network와 대조하여 해석하는 방법

과제의 요건

1. Network를 array (adjacency matrix)로 작성하기
2. Degree (차수) 계산하기
3. Degree Distribution (차수 분포) 그리기
4. 해석하고 Barabasi scale-free network와 비교하기

과제의 목적과 요건

ER random network

기호와 의미

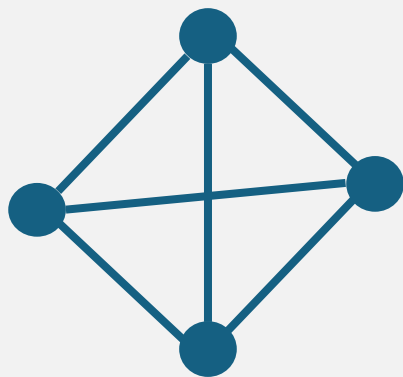
기호	의미
n	node 수
p	서로 다른 두 node 사이에 link가 생성될 확률
L	실제로 생성된 link 수 (link: 노드와 노드 사이의 연결선)
A	adjacency matrix (인접행렬)
$\bar{k}$	평균 degree (degree: 한 노드가 가지고 있는 link의 개수)

과제의 목적과 요건

ER random network

왜 평균 degree가 $(n - 1)p$ 인가

예를 들어, 노드가 4개라면 ($n=4$),



$$\text{만들 수 있는 링크 수는} \\ \frac{4(4-1)}{2} = 6\text{개} = \frac{n(n-1)}{2}$$

그런데, 모든 링크를 만드는 게 아니라 확률 p 로 만들 $\frac{n(n-1)}{2} p$

이때, 링크 1개는 2개 노드의 degree를 각각 1씩 증가 시킴
즉, link 1개이면 전체 degree의 합은 2 증가

→ 따라서, 모든 node의 degree 합 = $2L = 2 \times$ 전체 링크 수

그렇다면,

평균 degree $\langle k \rangle = \frac{\text{전체 degree의 합}}{n} = \frac{2L}{n}$ 이고,

기대 링크 수 $E[L] = \frac{n(n-1)}{2} p$ 이므로,

기대 평균 degree $E[\langle k \rangle] = \frac{2}{n} \cdot \frac{n(n-1)}{2} p = (n-1)p$

과제의 목적과 요건

ER random network

논문 Theorem 10의 표기와 본 실험의 표기를 대응해보면,

(논문) 노드 수 = n , 링크 수 N
(본 실험) 노드 수 = n , 링크 수 L

논문의 Theorem 10에 따르면,

링크 수 $L \approx cn$ 일 때,
한 노드의 degree가 j 일 확률은 평균이 $2c$ 인 Poisson distribution으로 수렴한다

$$P(k = j) \rightarrow \frac{(2c)^j e^{-2c}}{j!}$$

즉, 원 논문에서 네트워크의 평균 degree는 $2c$

과제의 목적과 요건

기대 평균 degree는 이렇게 표기함:

Theorem 10에서는 $2c$,
본 실험에서는 $(n-1)p$

본 실험 $G(n,p)$ 모형에서,
기대 링크 수 $E[L] = \frac{n(n-1)}{2} p$

따라서, 노드 1개당 링크 수 $c = \frac{E[L]}{n} = \frac{(n-1)}{2} p$
여기에, 양변에 2를 곱하면, $2c = (n-1)p$

- 즉, 원 논문의 평균 degree인 $2c$ 와
- 본 실험의 기대 평균 degree인 $(n-1)p$ 는
같은 양을 나타낸다

ER random network

그렇다면 어떻게 $(n-1)p \approx np$ 가 되나?

$(n-1)p = np - p$ 인데,
만약 n 이 엄청 커지면 n 에서 1을 제외한 영역이
엄청 작아짐

- $(n-1)p \approx np$ 가 된다
- $2c \approx np$ 가 된다

Simulation, Binomial, Poisson 그래프 비교해봐도,
Theorem 10와 일치하게 나옴

(ER network의 degree distribution이 평균 $2c$ 인 Poisson distribution으로 수렴한다)

1 단계.

Network를 Adjacency Matrix로 작성하기

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

Adjacency matrix A란? Node 사이의 연결 여부를 0과 1로 표현한 행렬

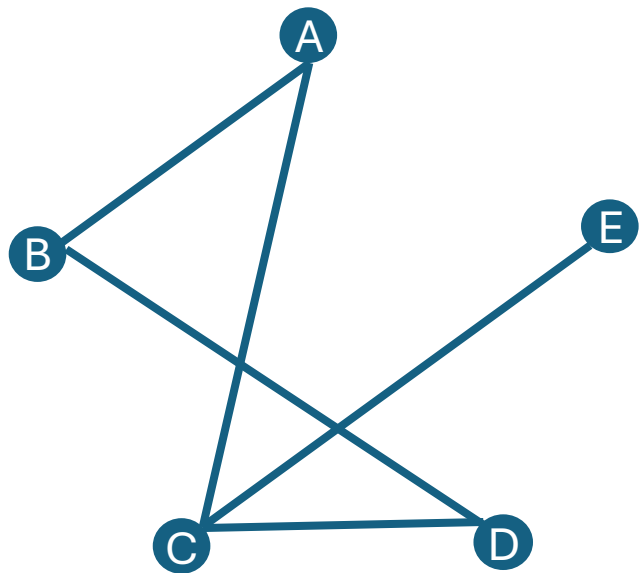
	의미
$A_{ij} = 1$	node i와 node j가 서로 연결됨
$A_{ij} = 0$	node i와 node j가 서로 연결되지 않음
Node는 자기 자신 node와는 연결하지 않으므로 대각선은 0	
방향은 고려하지 않는 네트워크이므로 $A_{ij} = A_{ji}$ 이며, 행렬은 대칭	

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

5개 노드로 구성된 작은 네트워크 만들어보기



```
import numpy as np

A_small = np.array([
    [0, 1, 1, 0, 0],
    [1, 0, 0, 1, 0],
    [1, 0, 0, 1, 1],
    [0, 1, 1, 0, 0],
    [0, 0, 1, 0, 0]
])

print(A_small)
```

```
[[0 1 1 0 0]
 [1 0 0 1 0]
 [1 0 0 1 1]
 [0 1 1 0 0]
 [0 0 1 0 0]]
```

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

① ER random network의 조건 설정하기

기호	의미	설정값
n	node 수	1000
p	서로 다른 두 node 사이에 link가 생성될 확률	0.01
seed	동일한 결과를 재현하기 위한 난수	42

```
n = 1000  
p = 0.01  
seed = 42
```

```
print("Node 수 n:", n)  
print("연결 확률 p:", p)  
print("난수 seed:", seed)  
print("기대 평균 degree:", (n - 1) * p)
```

```
Node 수 n: 1000  
연결 확률 p: 0.01  
난수 seed: 42  
기대 평균 degree: 9.99
```

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

② 무작위 숫자(난수)로 1000x1000 행렬 만들기

```
rng = np.random.default_rng(seed)

random_values = rng.random((n, n)) # nxn 행렬 만들고 각 칸에 0 이상 1 미만의 무작위 숫자를 넣음

print("행렬의 크기:", random_values.shape)
print(random_values[:5, :5])
```

```
행렬의 크기: (1000, 1000)
[[0.77395605 0.43887844 0.85859792 0.69736803 0.09417735]
 [0.06206311 0.45826204 0.12903006 0.15232671 0.63228281]
 [0.84273197 0.44638949 0.95297516 0.65079459 0.11589887]
 [0.82922114 0.59409599 0.24485475 0.745675 0.0844809 ]
 [0.71609997 0.3571156 0.85186786 0.24097716 0.53839349]]
```

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

③ 확률 p 를 활용해서, 그 행렬을 0과 1로 구성된 행렬로 변환하기

랜덤한 숫자로 구성된 행렬을
실제 네트워크의 '연결여부'로 바꾸는 단계

왜 $\text{random} < p$ 이면 연결 확률이 p 가 될까?

→ 난수생성 시 0~1 사이에서 숫자를 균등하게 뽑으니까,
랜덤한 숫자가 p (예를 들면 0.3) 보다 작을 확률을 보면,
30%가 됨.

→ 따라서, ' $\text{random_number} < 0.3$ 이면 1'로 만들면,
30%의 확률로 edge를 만드는 셈

```
connections = random_values < p  
print(connections[:5, :5])
```

```
[[False False False False False]  
 [False False False False False]  
 [False False False False False]  
 [False False False False False]  
 [False False False False False]]
```

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

④ 방향 없는 네트워크니까 대칭 행렬을 만들어야 함

행렬의 위쪽 절반만 사용하기

현재는 행렬의 각 칸마다 별개의 랜덤 숫자를 뽑았으므로 [0,1]과 [1,0]이 서로 다른 값일 수 있음.

→ $A_{ij} = A_{ji}$ 가 되게 해주기

```
upper_triangle = np.triu(connections, k=1) # 행렬의 위쪽 삼각형만 남겨라 (대각선과 아래쪽 부분은 모두 0으로)
```

```
print(upper_triangle[:5, :5])
```

```
[[False False False False False]
 [False False False False False]
 [False False False False False]
 [False False False False False]
 [False False False False False]]
```

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

④ 방향 없는 네트워크니까 대칭 행렬을 만들어야 함

행렬의 위쪽 절반을 아래쪽에 복사하기

```
A = upper_triangle + upper_triangle.T # 위쪽행렬 + 위쪽행렬의 전치행렬 -> 대칭인 인접행렬 만들어짐
A = A.astype(np.int8) # True/False 값을 각각 1/0으로 바꿈 -> A가 우리가 만든 adjacency matrix.

print("Adjacency matrix의 크기:", A.shape)
print(A[:10, :10])
```

Adjacency matrix의 크기: (1000, 1000)

```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

대칭 행렬이며, 대각이 0인가?

```
correct_shape = A.shape == (n, n)
is_symmetric = np.array_equal(A, A.T)
has_zero_diagonal = np.all(np.diag(A) == 0)
is_binary = np.all((A == 0) | (A == 1))

print("행렬의 크기가 n x n인가?", correct_shape)
print("대칭 행렬인가?", is_symmetric)
print("대각선이 모두 0인가?", has_zero_diagonal)
print("모든 원소가 0 또는 1인가?", is_binary)
```

행렬의 크기가 n x n인가? True

대칭 행렬인가? True

대각선이 모두 0인가? True

모든 원소가 0 또는 1인가? True

작은 네트워크 만들어서 확인하기

ER random network 만들기

만든 adjacency matrix 검증하기

실제 링크 수 L이 예상한 기대 link 수와 비슷한가?

기대 평균 degree: $(n-1)p \approx np$

$$\text{기대 링크 수 } E[L] = \frac{n(n-1)}{2} p$$

```
expected_L = n * (n - 1) / 2 * p # 기대 링크 수
L = A.sum() // 2 # 실제 링크 수

print("기대 link 수:", expected_L)
print("실제 link 수:", L)
```

기대 link 수: 4995.0

실제 link 수: 5045

2단계. Degree (차수) 계산하기

작은 네트워크 degree 계산해보기

ER 네트워크 degree 계산하기

계산한 degree 검증하기

- Degree란? node i에 연결된 link의 수 (한 노드가 가지고 있는 link의 개수)
 - Adjacency matrix에서 $A_{ij} = 1$ 이라면, node i와 node j가 서로 연결되어 있다
- 따라서, node i의 degree: adjacency matrix의 i번째 행에 있는 값을 모두 더한 값

$$k_i = \sum_{j=1}^n A_{ij}$$

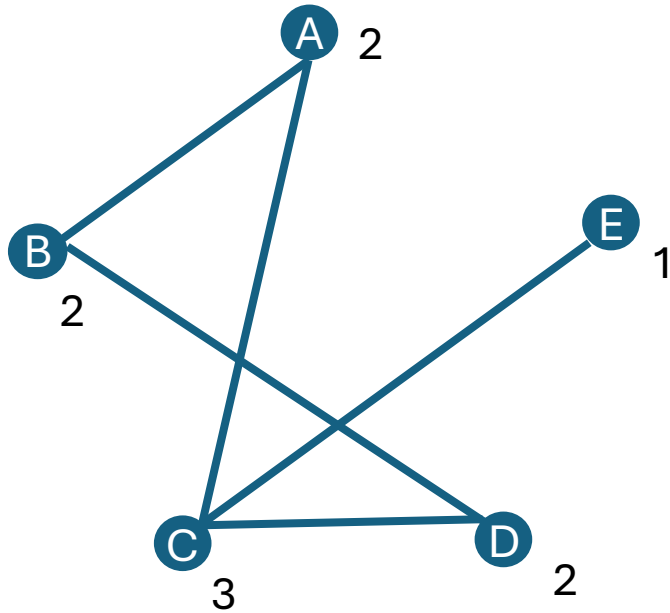
→ 모든 node의 degree를 계산하려면: adjacency matrix의 각 행을 합하기

작은 네트워크 degree 계산해보기

ER 네트워크 degree 계산하기

계산한 degree 검증하기

작은 예시에서 degree 계산해보기



```
degrees_small = A_small.sum(axis=1) # 각 행을 가로로 합함 (사실 무방향 네트워크니까 axis=1이나 axis=0이나 값은 동일함)

print("작은 네트워크의 adjacency matrix:")
print(A_small)

print("\n각 node의 degree:")
print(degrees_small)
```

작은 네트워크의 adjacency matrix:

```
[[0 1 1 0 0]
 [1 0 0 1 0]
 [1 0 0 1 1]
 [0 1 1 0 0]
 [0 0 1 0 0]]
```

각 node의 degree:

```
[2 2 3 2 1]
```

작은 네트워크 degree 계산해보기

ER 네트워크 degree 계산하기

계산한 degree 검증하기

① Adjacency matrix의 각 행을 합해서 모든 node의 degree 확인해보기

```
degrees = A.sum(axis=1)

print(degrees) # 1000개 node 각각의 degree를 담은 1차원 배열
print("Degree 배열의 크기:", degrees.shape)
print("앞의 20개 node의 degree:")
print(degrees[:20])

# node N의 degree 하나씩 뜯어보기
print(degrees[0]) # node 0의 degree
print(degrees[1]) # node 1의 degree
```

```
[10  7  6 11 10 11  9 13 15  9 12  8  7 12 11 15  8 15 12 15  9  6 15 13
13 12 12  5  7 14 10 12  8 15 12 13  6 11 10 11  7 10  6 12  7 10 11 12
11  7 12 12  7 16 15  9 11  9 10 11 14  9 17  8  5  8 12  9 11  8 10  9
16  7  8 15 16 10  9 10  5  9 14  3 17  7 15 15  9 10  8 12 12  9  5 14
14 12 11  7 17 10 14 10  8 13  9 10 13 10  9  8  9  7 11 16 11 18  9  6
11  4 13 15 10 11 10  9 15 11 14 13 10  9 11 13 10 11 16 10  9 15  8 12
15 12 13  8  8 12  7 11 13  9 14  9 11 11  8 13  7  9 19 14  6 11 14  8
11  6  8 12 12 11  6 10  7  9  8  8 10  6 11  9 10  5 10  3 10  8  7  6
 5 11 13 12  6  7 11  9 11 10 11 10  9 15 14  3  7  7 11 14  6  7  6  9
10 11 14  6 13 11 12  5 12 10 17 19  7 15  6  7  9 21 12  6 12  8 14 11
11  9  7  9  9  9 10 14 12 12  8 12  7  4 12 14 15 18  9 12  9 13  9  7
 9  9 11 10 11  6 10  9 11 13  6 12  5  6 10  7 10 13 14  7 13  8  5 11
 6 10  8  9  9  8  9 11 15  9  8 10 12  6  9 12  8  9  8 10  9 10  5  8
 5 11 10 13  8 11 10  9  8  9 13 11 13 12 13  7  9 13  7 12  9 13 14 10
14 11  5  8 13 13  8  8 14 11 13  4 16 19 15  8  8 14 11 11 13  7 10  7
12  7  5  9  5 13 10 10 11 10 13 11  9 10  8 10 10  6 13  3  9  6 12 11
16 10 11  9 11  7 13  6  6  7  9 12  7 13 10  7 16 11 11 15 11  8  4  8
 5  8 10  6  7 12  5 11 13 15  9  9 11 12 10 12 14 12 12 16  9 10  8 15
10  5 16  4 11 12  7  8  8  8 10  9  4 17  9  5  9  7 11  7 16  8  9 11
 9 14  8  7  8 15 15  9 12 13 11 10 12 11  7  9  4 12  8  9  9 11 10 12
13  8 10 18  8 11  8  5  5  6  2 12  9 13 12  9 16  9 14 11  7 15 10 14
 9 12  7  8  9 11 12 10 12 12  7  8  2 13  7  9 17 12 12  8 14 12  8 14
 9  6  7 15 10 12  7  6  8  7 10 17  8 11 11 12  8 17 10  8 16 12 10  8
 8 11 11  6 11 11 11 14 10 10 11 13 16  9 12  5  8 14 14 12  8 12 15  9
11 13  4 14 12 11  5  8 12  7  9  9 14 15 10 17  9 13  8  8  9  9 12  8
 6  8  8  7 16 12 12 12 10  7  7 12 13  9 10  6 11 12  8  9  6 10  8
11 17  8 11  7  8 13 12  8 11  7 14 19  6 10 12 14 11  7 16 12  8 10  6
 4  7  6  6  5 10 12  7  9  6  7 15 10 14 11 10 13  7  9 17 11 10  6 11
 5  6 13  6  6  8  9  8  5 14  7  5 12 10 10 10  5 12 13  8  9  3 13  8
 7  9 14 10 11 10  8  7  3 12 10 10  6 10  3 11  7  4 12  9 14  9  6  8
 7  8 11 13 16  7 11  8  8  6  7 13  5 10 14  4 14 15  7 12 13  9 14  8
 4 12 11 10  6  9  8 10 13 13  6 14  2  8 11 12 11 13  9  7 10 12  6 15
 6  8 11 14 12  9 11  9  9  7  7 12 11  7  7  8 12 15 16 16 16  9 11 13
12  7  9  9 15 11  6 11  7 14 13  7  5 10 13 14 10  9 12 10  6  8  8  9
11 11  6 12 11  6 10 10 11  8  9 17  8  8  9 13  9  7  6  8 17  9 12  9
 8  9 13 10 19  8 13 13 14 13  4 14  9  9  9  9 15  8 10 11  9  7  8 12
11  8 11  5 10 11  6  6  5 16  8 13 15 14 14 11 11  9  6  9 12 14 10 11
11 11  9  8  6 12 14 17 15 11 11 11  8 11 10  7  7  9 11  8  9 10 15  7
 7 15  8 11 10 10  6 10  9 15 14 10  6  7 13 12  9  7  9 11  9 11  8 10
 5 10 12  6 11  8  6  9  9  3 12 13 14  7  8  6 14 10 12 12  9  5 10 10
 8 15  9 10 15 11  8  9  9 10  7 11 10  9 10  9 10  8  7 13 15 12 10 14
12 12  9  8 10 19 22  7 14  9  8 19 14  6  6 11]
```

Degree 배열의 크기: (1000,)

앞의 20개 node의 degree:

```
[10  7  6 11 10 11  9 13 15  9 12  8  7 12 11 15  8 15 12 15]
```

10

7

작은 네트워크 degree 계산해보기

ER 네트워크 degree 계산하기

계산한 degree 검증하기

③ 개별 node의 degree를 확인했으니, 전체 네트워크의 최소, 최대, 평균 degree 확인하기

```
min_degree = degrees.min()
max_degree = degrees.max()
observed_mean_degree = degrees.mean()
expected_mean_degree = (n - 1) * p

print("최소 degree:", min_degree)
print("최대 degree:", max_degree)
print("관측 평균 degree:", observed_mean_degree)
print("기대 평균 degree:", expected_mean_degree)
```

```
최소 degree: 2
최대 degree: 22
관측 평균 degree: 10.09
기대 평균 degree: 9.99
```

작은 네트워크 degree 계산해보기

ER 네트워크 degree 계산하기

계산한 degree 검증하기

방법 ① degree와 link 수의 관계로 검증하기

	의미
link	노드와 노드 사이의 연결 선
degree	한 노드가 가지고 있는 link의 개수

```
degree_sum = degrees.sum()
two_times_L = 2 * L

print("모든 node의 degree 합:", degree_sum)
print("전체 link 수의 2배:", two_times_L)
print("두 값이 같은가?", degree_sum == two_times_L)
```

모든 node의 degree 합: 10090
전체 link 수의 2배: 10090
두 값이 같은가? True

이때, 무방향 네트워크에서 링크 1개는 2개 노드의 degree를 각각 1씩 증가 시킴
즉, link 1개이면 전체 degree의 합은 2 증가

→ 따라서, 모든 node의 degree 합 = 링크 수 x 2

→ 이를 node 개수(n)으로 나누면 = 실제 네트워크의 평균 degree

작은 네트워크 degree 계산해보기

ER 네트워크 degree 계산하기

계산한 degree 검증하기

방법 ② 평균 degree로 검증하기

모든 node의 degree 평균
링크 수로 계산한 평균 ($2L/n$)

```
mean_degree_from_degrees = degrees.mean()
mean_degree_from_links = 2 * L / n

print("Degree 배열에서 계산한 평균:", mean_degree_from_degrees)
print("Link 수로 계산한 평균:", mean_degree_from_links)
print(
    "두 평균이 같은가?",
    np.isclose(mean_degree_from_degrees, mean_degree_from_links)
)
```

Degree 배열에서 계산한 평균: 10.09
Link 수로 계산한 평균: 10.09
두 평균이 같은가? True

3단계. Degree Distribution 그리기

Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

① Degree 별 node 수 계산하기

- Degree distribution $P(k)$: 임의로 고른 node의 degree가 k 일 확률 (즉, degree가 k 인 node의 비율)
- 가로축에 k , 세로축에 $P(k)$ 를 놓은 막대그래프로 그리기

→ Degree가 몇인 노드가 몇 개씩 있는지 세보자

```
degree_values, degree_counts = np.unique(degrees, return_counts=True)

print("Degree 값:", degree_values) # Degree 배열에 어떤 값들이 존재하나
print("각 degree에 해당하는 node 수:", degree_counts) # 그 값들이 몇 개씩 있나
# 즉, Degree가 2인 node 수는 3개, degree가 3인 node 수는 8개, ...
```

```
Degree 값: [ 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 21 22]
각 degree에 해당하는 node 수: [ 3  8 13 33 65 87 113 130 115 121 101 65 57 42 21 14  3  7
 1  1]
```

Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

② Degree 별 확률 계산하기

- **degree_counts: node의 개수**
전체 node 중에서 이 degree 값을 갖는 node가 몇 개다
- **degree_counts/n: 확률**
전체 node 중에서 이 degree 값을 갖는 node가 차지하는 확률

```
degree_probabilities = degree_counts / n

for degree, count, probability in zip(
    degree_values,
    degree_counts,
    degree_probabilities
):
    print(
        f"Degree {degree:2d}: "
        f"{count:3d}개 node, "
        f"P(k) = {probability:.3f}"
    )

print("Degree 확률의 합:", degree_probabilities.sum())
```

```
Degree 2: 3개 node, P(k) = 0.003
Degree 3: 8개 node, P(k) = 0.008
Degree 4: 13개 node, P(k) = 0.013
Degree 5: 33개 node, P(k) = 0.033
Degree 6: 65개 node, P(k) = 0.065
Degree 7: 87개 node, P(k) = 0.087
Degree 8: 113개 node, P(k) = 0.113
Degree 9: 130개 node, P(k) = 0.130
Degree 10: 115개 node, P(k) = 0.115
Degree 11: 121개 node, P(k) = 0.121
Degree 12: 101개 node, P(k) = 0.101
Degree 13: 65개 node, P(k) = 0.065
Degree 14: 57개 node, P(k) = 0.057
Degree 15: 42개 node, P(k) = 0.042
Degree 16: 21개 node, P(k) = 0.021
Degree 17: 14개 node, P(k) = 0.014
Degree 18: 3개 node, P(k) = 0.003
Degree 19: 7개 node, P(k) = 0.007
Degree 21: 1개 node, P(k) = 0.001
Degree 22: 1개 node, P(k) = 0.001
Degree 확률의 합: 1.0
```

Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

③ Degree distribution 그리기

```
import matplotlib.pyplot as plt

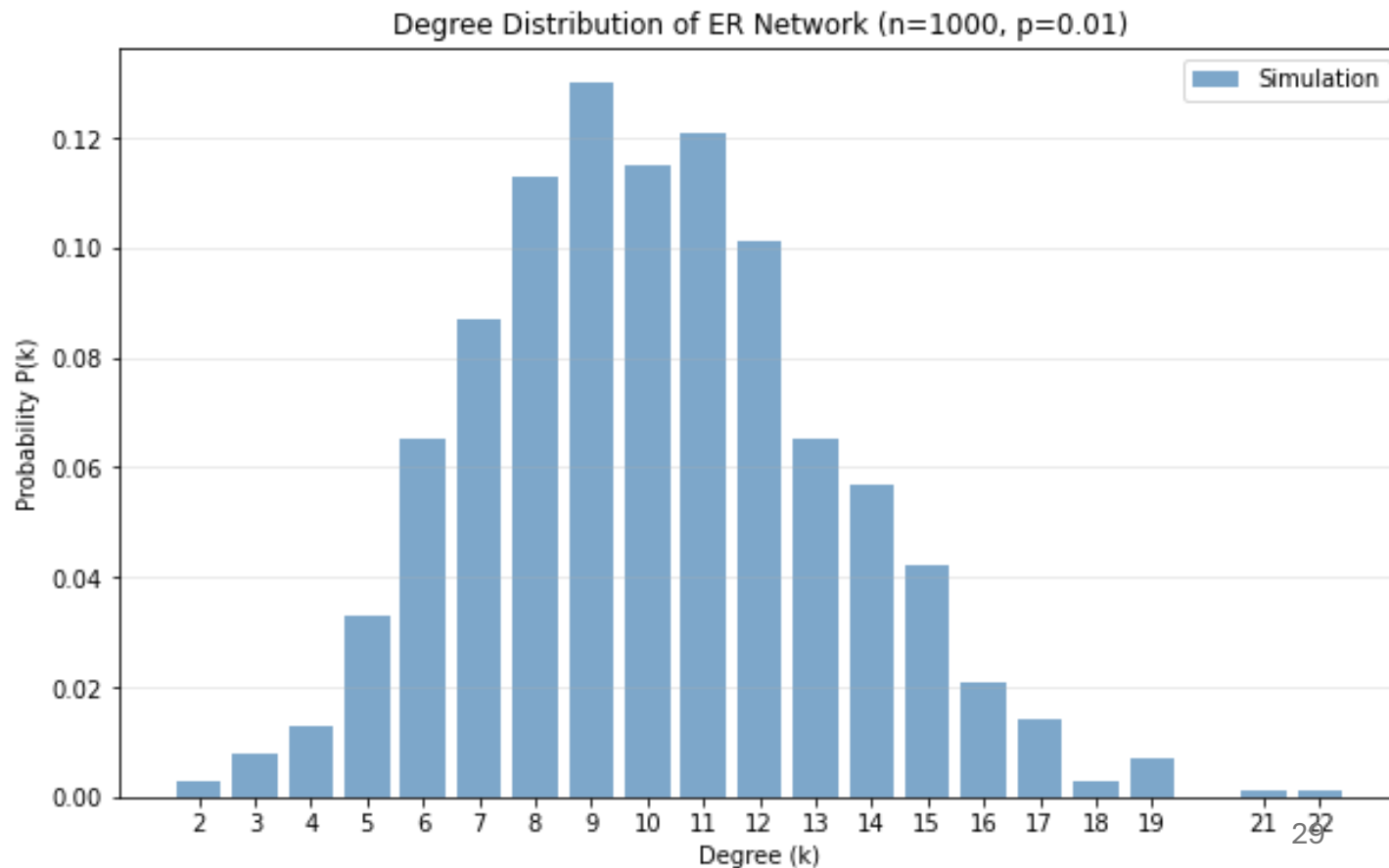
plt.figure(figsize=(10, 6))

plt.bar(
    degree_values,
    degree_probabilities,
    color="steelblue",
    alpha=0.7,
    label="Simulation"
)

plt.xlabel("Degree (k)") # degree k
plt.ylabel("Probability P(k)") # 해당 degree를 가진 node의 비율
plt.title(f"Degree Distribution of ER Network (n={n}, p={p})")
plt.xticks(degree_values)
plt.legend()
plt.grid(axis="y", alpha=0.3)
plt.show()
```

가로축: degree k

세로축: 전체 중 해당 degree를 가진 node의 비율



Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

① 왜 Binomial · Poisson distribution과 비교해봐야 할까?

- $G(n, p)$ 모형에서, 한 노드는 자신을 제외한 $n-1$ 개의 노드 각각과 독립적으로 확률 p 로 연결됨
→ 따라서, 한 노드의 degree K 는 binomial distribution을 따름
 - 또한, ER 네트워크의 degree는 binomial distribution을 따르지만, n 이 크고 p 가 작으면 binomial이 Poisson distribution과 거의 똑같아짐
→ Poisson으로 대신 표현할 수 있음
- 정말 그런지 그림으로 확인해보자!

Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

② Binomial과 Poisson 확률 계산하기

```
from scipy.stats import binom, poisson
```

```
# k_values: 확률을 계산해볼 degree 값들의 목록
k_values = np.arange(0, degrees.max() + 1)
```

```
# Binomial 확률 계산 (각 degree k가 나타날 Binomial 확률 계산)
```

```
binomial_probabilities = binom.pmf(
    k_values, # 알고싶은 degree: 0, 1, 2, ..., 22
    n - 1, # 한 node가 연결될 수 있는 다른 node 수
    p # 각각의 node와 연결될 확률
)
```

```
# Poisson Distribution
```

```
lambda_poisson = (n - 1) * p # lambda: 평균적으로 몇 번 발생하는가. (ER network의 기대 평균 degree 값)
```

```
poisson_probabilities = poisson.pmf(
    k_values, # 알고싶은 degree: 0, 1, 2, ..., 22
    lambda_poisson # ER network의 기대 평균 degree 값
)
```

```
# binomial 확률값 출력해보기 (degree 0의 확률, degree 1의 확률, ... 순서대로)
```

```
print("Binomial probabilities:")
print(binomial_probabilities)
```

```
# poisson 확률값 출력해보기 (degree 0의 확률, degree 1의 확률, ... 순서대로)
```

```
print("\nPoisson probabilities:")
print(poisson_probabilities)
```

```
print("Poisson distribution의 lambda:", lambda_poisson) # Poisson distribution 계산할 때 사용한 lambda 값 확인
```

```
# degree 별로 binomial 확률값과 poisson 확률값 출력해보기
print("Degree | Binomial | Poisson")
```

```
for k, b, pois in zip(k_values, binomial_probabilities, poisson_probabilities):
    print(k, b, pois)
```

```
Binomial probabilities:
```

```
[4.36073206e-05 4.40037508e-04 2.21796683e-03 7.44549809e-03
 1.87265558e-02 3.76422687e-02 6.29905979e-02 9.02592549e-02
 1.13051996e-01 1.25740211e-01 1.25740211e-01 1.14193819e-01
 9.49692703e-02 7.28319112e-02 5.18126006e-02 3.43672806e-02
 2.13493713e-02 1.24696565e-02 6.87160646e-03 3.58375648e-03
 1.77377846e-03 8.35271338e-04 3.75066744e-04]
```

```
Poisson probabilities:
```

```
[4.58562066e-05 4.58103504e-04 2.28822700e-03 7.61979592e-03
 1.90304403e-02 3.80228198e-02 6.33079949e-02 9.03495527e-02
 1.12824004e-01 1.25234644e-01 1.25109410e-01 1.13622091e-01
 9.45903909e-02 7.26890773e-02 5.18688488e-02 3.45446533e-02
 2.15688179e-02 1.26748524e-02 7.03454308e-03 3.69868870e-03
 1.84749501e-03 8.78879768e-04 3.99091313e-04]
```

```
Poisson distribution의 lambda: 9.99
```

```
Degree | Binomial | Poisson
```

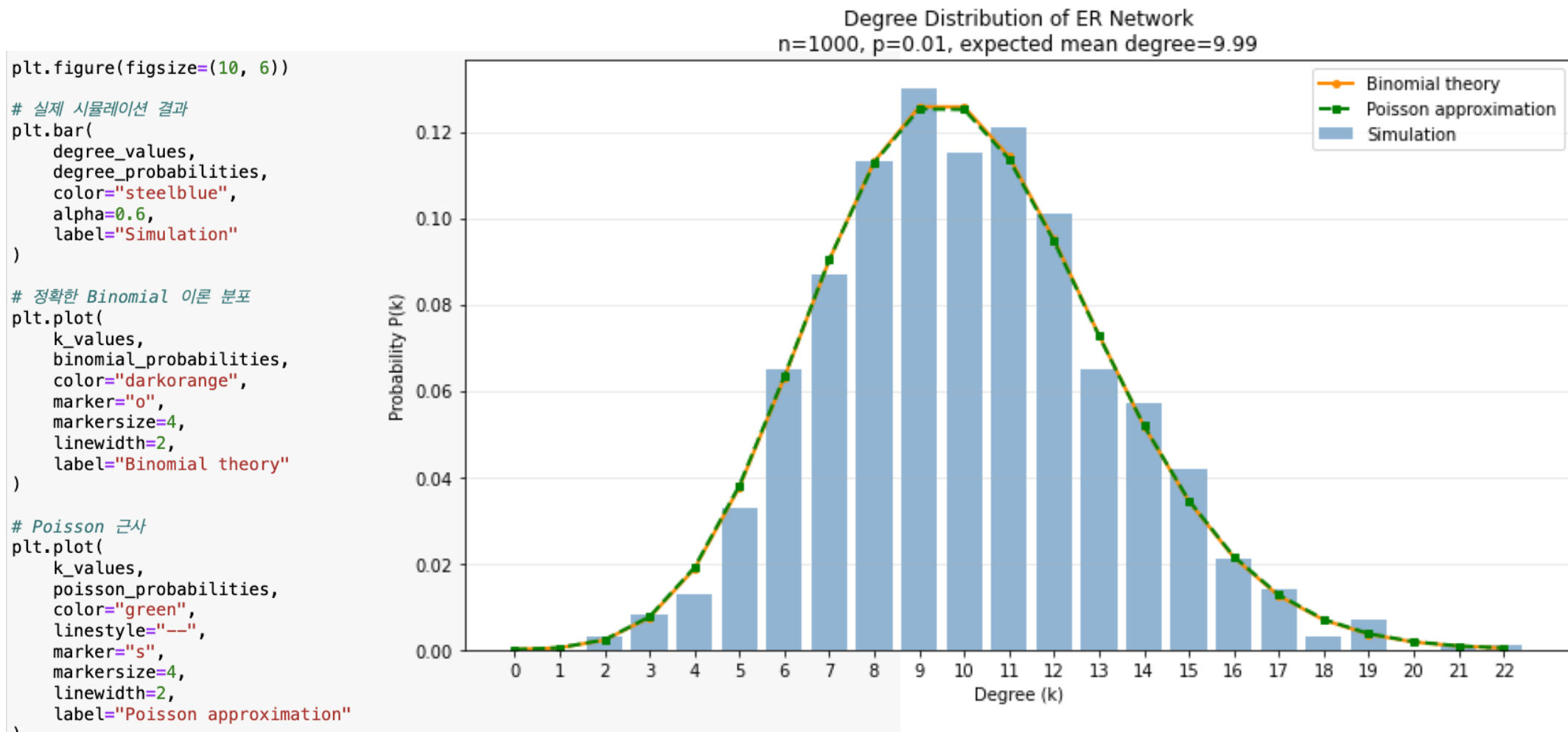
```
0 4.360732061682645e-05 4.585620664220731e-05
1 0.0004400375080419069 0.000458103504355651
2 0.0022179668334660683 0.0022882270042564766
3 0.007445498090792547 0.007619795924174064
4 0.018726555804109216 0.019030440320624738
5 0.0376422687375741 0.03802281976060819
6 0.06299059785377682 0.06330799490141266
7 0.09025925493329992 0.09034955272358755
8 0.11305199607806825 0.11282400396357983
9 0.12574021112602748 0.12523464439957363
10 0.125740211126109 0.12510940975517382
11 0.11419381891984526 0.11362209122310814
12 0.09496927028016251 0.09459039094323764
13 0.07283191124047236 0.07268907734791874
14 0.051812600637206295 0.05186884876469338
15 0.034367280557346384 0.034544653277285745
16 0.021349371255303524 0.021568817890005258
17 0.012469656532369719 0.012674852395361912
18 0.006871606461721008 0.007034543079425875
19 0.0035837564800322913 0.003698688703340227
20 0.0017737784598167248 0.0018474950073184427
21 0.0008352713382199463 0.0008788797677672078
22 0.0003750667441591504 0.0003990913127270166
```

Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

③ 우리의 degree distribution simulation · Binomial · Poisson 그려서 비교해보기



Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

① 평균 degree $(n-1)p$ 를 10으로 고정하면, node수가 커질 때 분포가 어떻게 달라질까?

조건	평균 degree= $(n-1)p$	n	$p=10/(n-1)$
A	10	500	0.02004
B	10	1000	0.01001
C	10	2000	0.00500

조건별 실험 함수 만들기

```
def generate_er_degrees(n, p, seed=42): # n=node개수, p=두 node가 연결될 확률, seed=난수 생성 시작점
    rng = np.random.default_rng(seed) # 난수 생성기

    random_values = rng.random((n, n)) # n x n 행렬 만들고 각 칸에 0~1 사이의 랜덤 숫자 넣고
    connections = random_values < p # 랜덤 숫자가 p보다 작은지 확인

    upper_triangle = np.triu(connections, k=1) # 무방향 네트워크니까 행렬 위쪽 절반만 사용

    A = upper_triangle + upper_triangle.T # 위쪽행렬+전치행렬 붙여서 대칭적인 adjacency matrix 만들고
    A = A.astype(np.int8) # True,False를 1,0으로 바꾸고

    degrees = A.sum(axis=1) # 각 node의 degree 계산

    return degrees # degree 배열 제공
```

조건 A, B, C 그래프 그리기

```
## 실험할 조건 3개 저장
conditions = [
    {"n": 500, "p": 10 / 499, "seed": 42},
    {"n": 1000, "p": 10 / 999, "seed": 42},
    {"n": 2000, "p": 10 / 1999, "seed": 42}
]

## 그래프 그릴 공간 만들기
fig, axes = plt.subplots(
    1,
    3,
    figsize=(18, 5),
    sharey=True
)

## 조건 A, B, C 하나씩(condition), 공간(ax)에 그리기
for ax, condition in zip(axes, conditions):
    current_n = condition["n"] # 현재 조건에서 n 가져오기
    current_p = condition["p"] # 현재 조건에서 p 가져오기
    current_seed = condition["seed"] # 현재 조건에서 seed 가져오기

    current_degrees = generate_er_degrees( # ER 네트워크 생성하기 (위에서 만들 함수 호출)
        current_n,
```

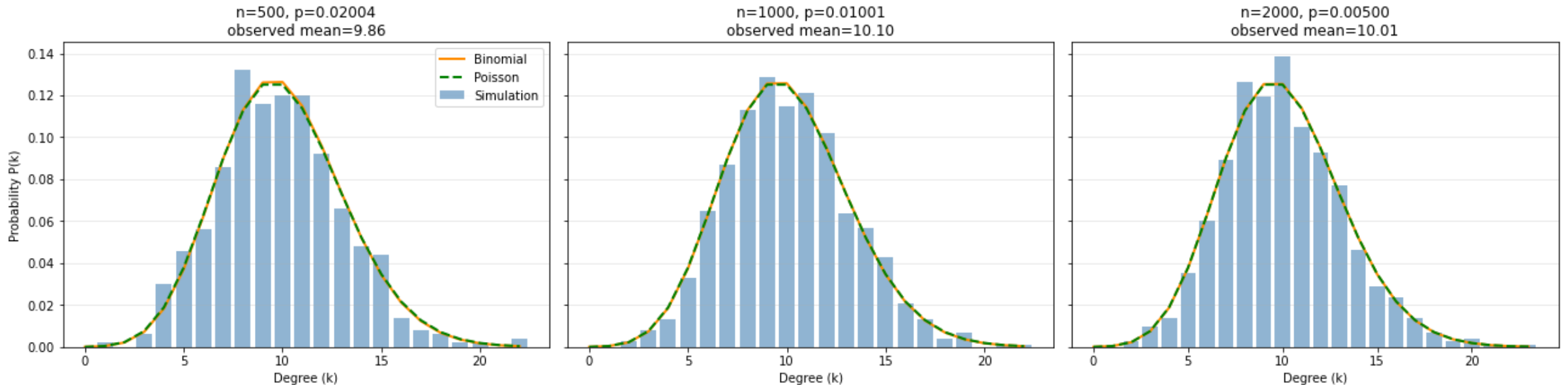
Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

① 평균 degree $(n-1)p$ 를 10으로 고정하면,
분포의 모양(봉우리)은 유지됨 & n 이 커지고 p 가 작아질수록 simulation 결과가 poisson 분포를 더 안정적으로 따름

Degree Distributions under Different n and p



조건	평균 degree= $(n-1)p$	n	$p=10/(n-1)$
A	10	500	0.02004
B	10	1000	0.01001
C	10	2000	0.00500

Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

② n을 1000으로 고정하고 p를 바꾸면, 분포가 어떻게 달라질까?

조건	n	p	기대평균 degree = (n-1)p
A	1000	0.005	4.995
B	1000	0.010	9.990
C	1000	0.020	19.980

- 노드 수 n을 유지하면, p가 2배가 될 때 평균 degree도 거의 2배로 커짐
→ p가 커지면서 degree distribution의 중심과 봉우리가 오른쪽으로 이동할까?

```
# 실험 조건 설정하기
fixed_n = 1000
p_values = [0.005, 0.01, 0.02]
fixed_seed = 42

# 조건 A, B, C의 degree distribution 그리기
## 앞에서 만든 함수 불러오기
def generate_er_degrees(n, p, seed=42):
    rng = np.random.default_rng(seed)

    random_values = rng.random((n, n))
    connections = random_values < p

    upper_triangle = np.triu(connections, k=1)

    A = upper_triangle + upper_triangle.T
    A = A.astype(np.int8)

    degrees = A.sum(axis=1)
```

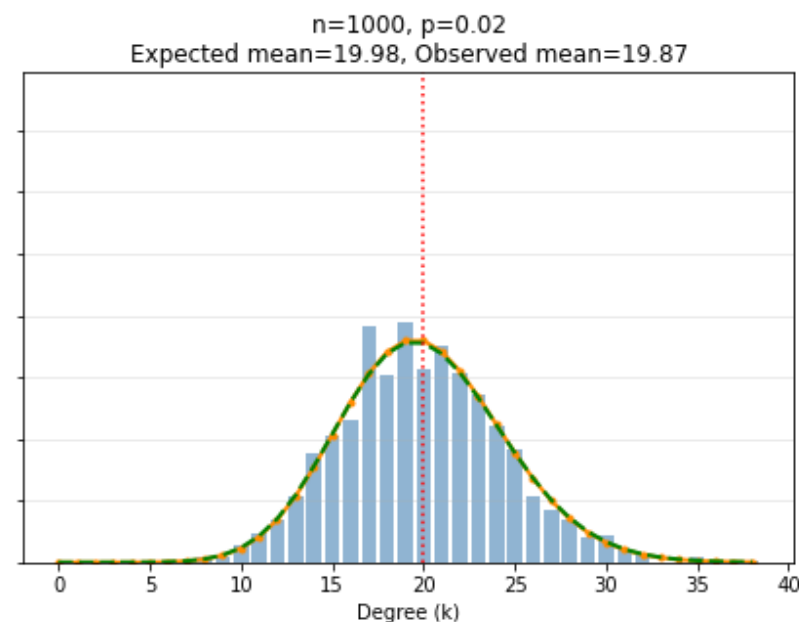
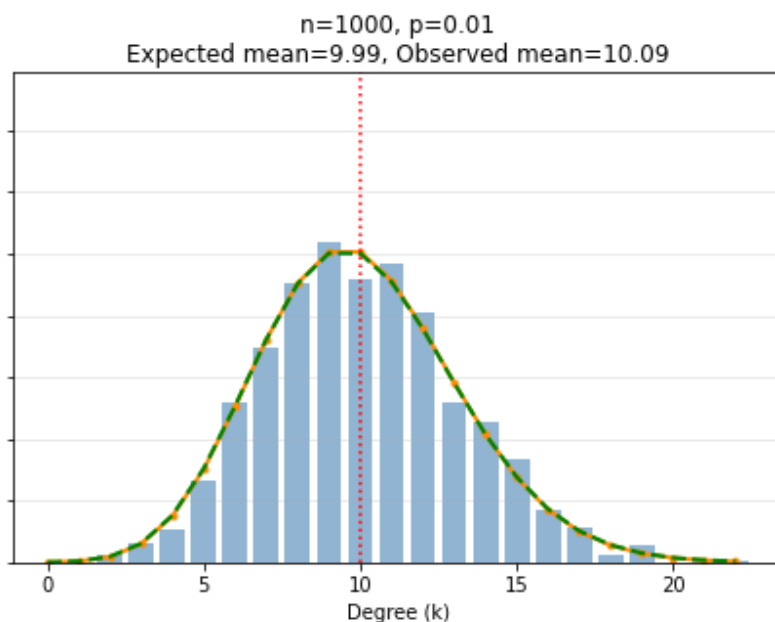
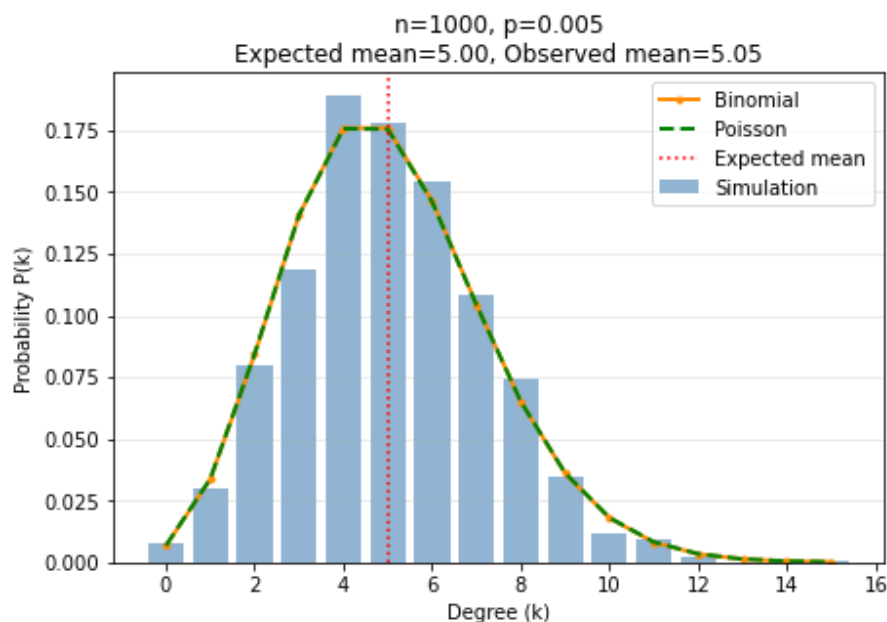
Degree Distribution 그리기

Binomial · Poisson과 비교하기

n, p 조건 바꿔 비교하기

② n을 1000으로 고정하고 p를 바꾸면,
degree distribution의 중심과 봉우리가 오른쪽으로 이동함

Degree Distributions with Fixed n and Different p



조건	n	p	기대평균 degree = (n-1)p
A	1000	0.005	4.995
B	1000	0.010	9.990
C	1000	0.020	19.980

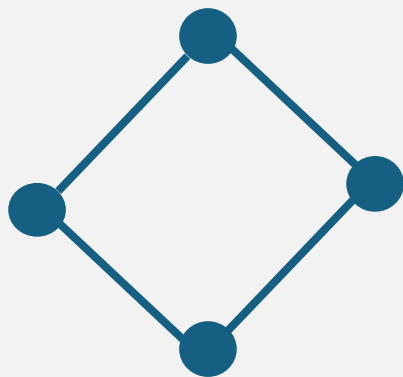
4단계. 결과 해석 · Barabasi scale-free network와 비교

ER random network 결과 해석

Barabasi scale-free network와 비교

왜 평균 degree가 $(n - 1)p$ 인가

예를 들어, 노드가 4개라면 ($n=4$),



$$\text{만들 수 있는 링크 수는} \\ \frac{4(4-1)}{2} = 6\text{개} = \frac{n(n-1)}{2}$$

그런데, 모든 링크를 만드는 게 아니라 확률 p 로 만들 $\frac{n(n-1)}{2} p$

이때, 링크 1개는 2개 노드의 degree를 각각 1씩 증가 시킴
즉, link 1개이면 전체 degree의 합은 2 증가

→ 따라서, 모든 node의 degree 합 = $2L$

그렇다면,

평균 degree $\langle k \rangle = \frac{\text{전체 degree의 합}}{n} = \frac{2L}{n}$ 이고,

기대 링크 수 $E[L] = \frac{n(n-1)}{2} p$ 이므로,

기대 평균 degree $E[\langle k \rangle] = \frac{2}{n} \cdot \frac{n(n-1)}{2} p = (n-1)p$

ER random network 결과 해석

Barabasi scale-free network와 비교

논문 Theorem 10의 표기와 본 실험의 표기를 대응해보면,

(논문) 노드 수 = n , 링크 수 N
(본 실험) 노드 수 = n , 링크 수 L

논문의 Theorem 10에 따르면,

링크 수 $L \approx cn$ 일 때,
한 노드의 degree가 j 일 확률은 평균이 $2c$ 인 Poisson distribution으로 수렴한다

$$P(k = j) \rightarrow \frac{(2c)^j e^{-2c}}{j!}$$

즉, 원 논문에서 네트워크의 평균 degree는 $2c$

ER random network 결과 해석

기대 평균 degree는 이렇게 표기함:

Theorem 10에서는 $2c$,
본 실험에서는 $(n-1)p$

본 실험 $G(n,p)$ 모형에서,
기대 링크 수 $E[L] = \frac{n(n-1)}{2} p$

따라서, 노드 1개당 링크 수 $c = \frac{E[L]}{n} = \frac{(n-1)}{2} p$
여기에, 양변에 2를 곱하면, $2c = (n-1)p$

- 즉, 원 논문의 평균 degree인 $2c$ 와
- 본 실험의 기대 평균 degree인 $(n-1)p$ 는
같은 양을 나타낸다

Barabasi scale-free network와 비교

그렇다면 어떻게 $(n-1)p \approx np$ 가 되나?

$(n-1)p = np - p$ 인데,
만약 n 이 엄청 커지면 n 에서 1을 제외한 영역이
엄청 작아짐

- $(n-1)p \approx np$ 가 된다
- $2c \approx np$ 가 된다

Simulation, Binomial, Poisson 그래프 비교해봐도,
Theorem 10와 일치하게 나옴

(ER network의 degree distribution이 평균 $2c$ 인 Poisson distribution으로 수렴한다)

ER random network 결과 해석

Barabasi scale-free network와 비교

① Barabasi scale-free network 만들어서 node, link, degree 확인해보기

- 노드 수 $n = 1000$
- 새로운 노드는 기존 노드 5개와 연결된다고 하자 ($m=5$)

만약,

- 노드 1개 추가 → link는 $1 \times 5 = 5$ 개 더 생기고, 전체 degree 합 10 증가
(link 1개는 node 2개의 degree를 각각 1씩 올리니까)
- 노드 2개 추가 → link는 $2 \times 5 = 10$ 개 더 생기고, 전체 degree 합 20 증가
- 노드 3개 추가 → link는 $3 \times 5 = 15$ 개 더 생기고, 전체 degree 합 30 증가

즉, 네트워크가 성장하면,

- ‘추가된 노드 수 $\times m$ ’만큼 링크가 더 생기고 (링크수 $L = nm$)
- ‘추가된 노드 수 $\times m \times 2$ ’만큼 전체 degree 합이 증가함 (전체 degree 총합 = $2nm$)

$$\text{결과적으로, 평균 degree} = \frac{\text{전체 노드들의 degree 총합}}{\text{node 수}} = \frac{2 \times \text{링크 수}}{\text{node 수}} = \frac{2L}{n} = 2m$$

```
ba_n = 1000
ba_m = 5 # 새로운 node는 기존 node 중 5개와 연결된다
ba_seed = 42

BA = nx.barabasi_albert_graph( # BA network 생성
    ba_n,
    ba_m,
    seed=ba_seed
)

ba_degrees = np.array([ # BA network에서 모든 node의 degree
    degree
    for node, degree in BA.degree()
])

print("BA network의 node 수:", BA.number_of_nodes())
print("BA network의 link 수:", BA.number_of_edges()) # L=
print("BA network의 평균 degree:", ba_degrees.mean()) # 실
print("BA network의 최소 degree:", ba_degrees.min())
print("BA network의 최대 degree:", ba_degrees.max()) # pre
print(ba_degrees)
```

```
BA network의 node 수: 1000
BA network의 link 수: 4975
BA network의 평균 degree: 9.95
BA network의 최소 degree: 5
BA network의 최대 degree: 123
[ 86  50  82  30  42 122  88  87 123  95  66  56  28  10  48  53  6
  53  36  34  30  27  24  16  41  49  23  15  25  33  26  35  21  1
  18  16  19  18  22  15  22  13  37  52  15  13  15  24  11  29
  32  25  25  11  17  15  25  35  22  17  23  18  20  27  31  31  1
  31  26  29  16  18  11  25  20  21  12  30  13  17  24  21  16  2
   9  21  20  13  15  18  12  22  8  15  16  22  18  11  17  18  1
  11  16  12  10  6  21  20  10  24  15  12  17  28  14  24  11  1
  11  16  21  13  8  17  8  0  7  12  8  16  8  17  15  18]
```

ER random network 결과 해석

Barabasi scale-free network와 비교

② ER과 BA 네트워크 수치 비교하기

네트워크	노드별 평균 degree	최소 degree	최대 degree
ER	10.09	2	22
BA	9.95	5	123

ER의 최대 degree는 22로 비교적 작는데, BA의 최대 degree는 123으로 훨씬 큼
→ 즉, BA scale free network에는 소수의 hub가 존재한다.

ER random network 결과 해석

Barabasi scale-free network와 비교

③ ER과 BA 네트워크 degree distribution 그리기

ER, BA의 Degree distribution 계산하기

```
# ER degree distribution
er_degree_values, er_degree_counts = np.unique(
    degrees,
    return_counts=True
)

er_degree_probabilities = (
    er_degree_counts / len(degrees)
)

# BA degree distribution
ba_degree_values, ba_degree_counts = np.unique(
    ba_degrees,
    return_counts=True
)

ba_degree_probabilities = (
    ba_degree_counts / len(ba_degrees)
)
```

ER, BA의 Degree distribution 그리기

```
fig, axes = plt.subplots(
    1,
    2,
    figsize=(15, 5),
    sharey=True
)

# ER network
axes[0].bar(
    er_degree_values,
    er_degree_probabilities,
    color="steelblue",
    alpha=0.7
)

axes[0].set_title(
    f"ER Random Network\n"
    f"Mean={degrees.mean():.2f}, Max={degrees.max()}"
)
axes[0].set_xlabel("Degree (k)")
axes[0].set_ylabel("Probability P(k)")
axes[0].grid(axis="y", alpha=0.3)

# BA network
axes[1].bar(
    ba_degree_values,
    ba_degree_probabilities,
```

ER, BA의 Degree distribution log-log plot 그리기

```
# log-log plot으로 그리기

# ER degree distribution
er_degree_values, er_degree_counts = np.unique(
    degrees,
    return_counts=True
)
er_degree_probabilities = er_degree_counts / len(degrees)

# BA degree distribution
ba_degree_values, ba_degree_counts = np.unique(
    ba_degrees,
    return_counts=True
)
ba_degree_probabilities = ba_degree_counts / len(ba_degrees)

# Log-log plot
plt.figure(figsize=(8, 6))

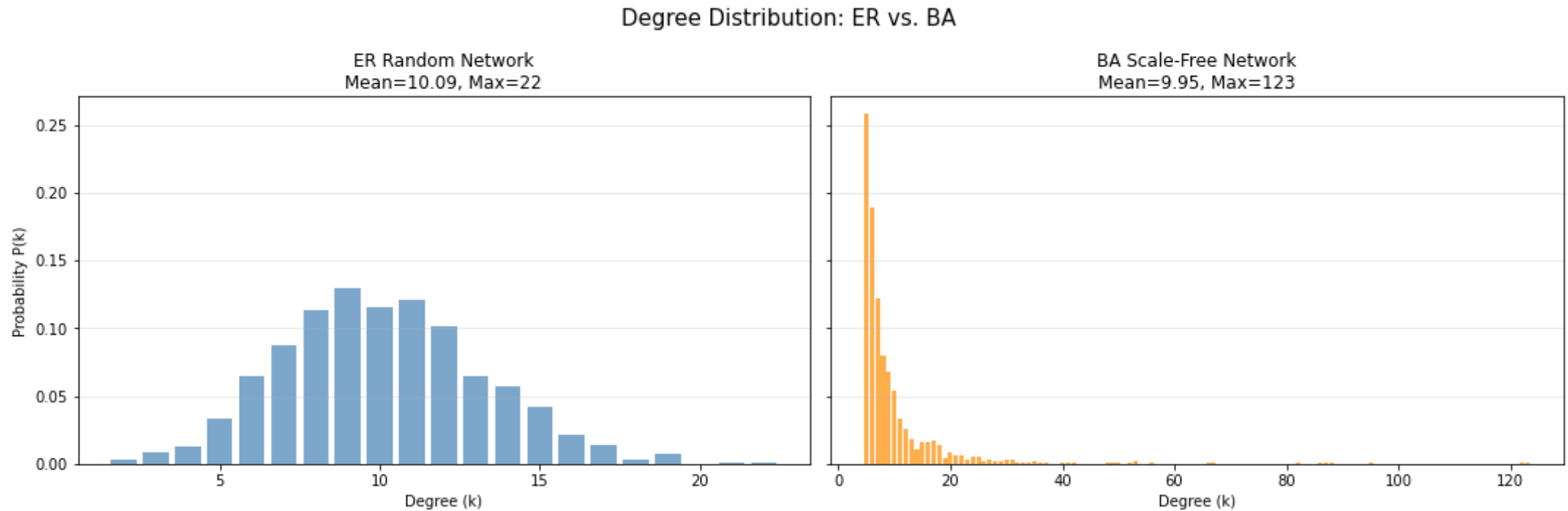
plt.loglog(
    er_degree_values,
    er_degree_probabilities,
    marker="o",
    linestyle="--",
    label="ER Random Network"
)

plt.loglog(
    ba_degree_values,
    ba_degree_probabilities,
```

ER random network 결과 해석

Barabasi scale-free network와 비교

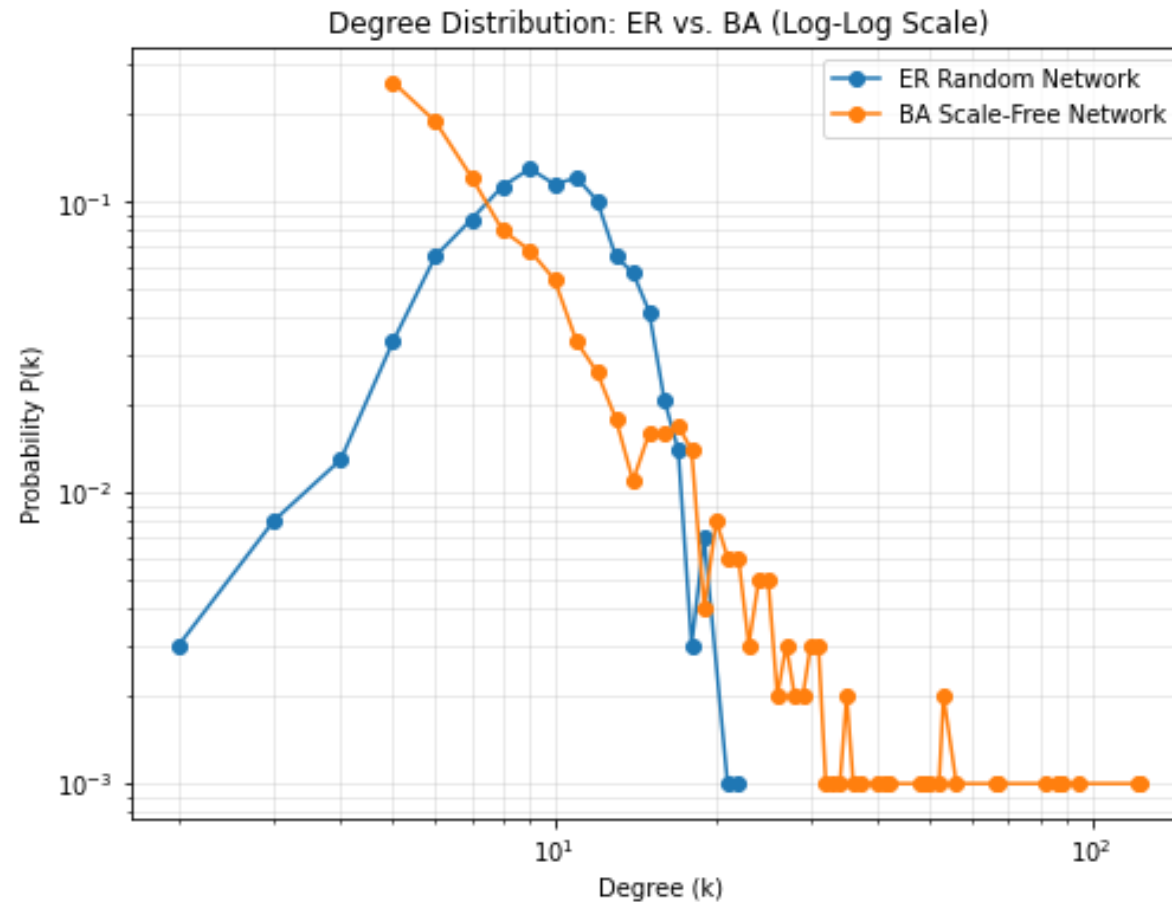
③ ER과 BA 네트워크 degree distribution 그리기



ER random network 결과 해석

Barabasi scale-free network와 비교

③ ER과 BA 네트워크 degree distribution 그리기

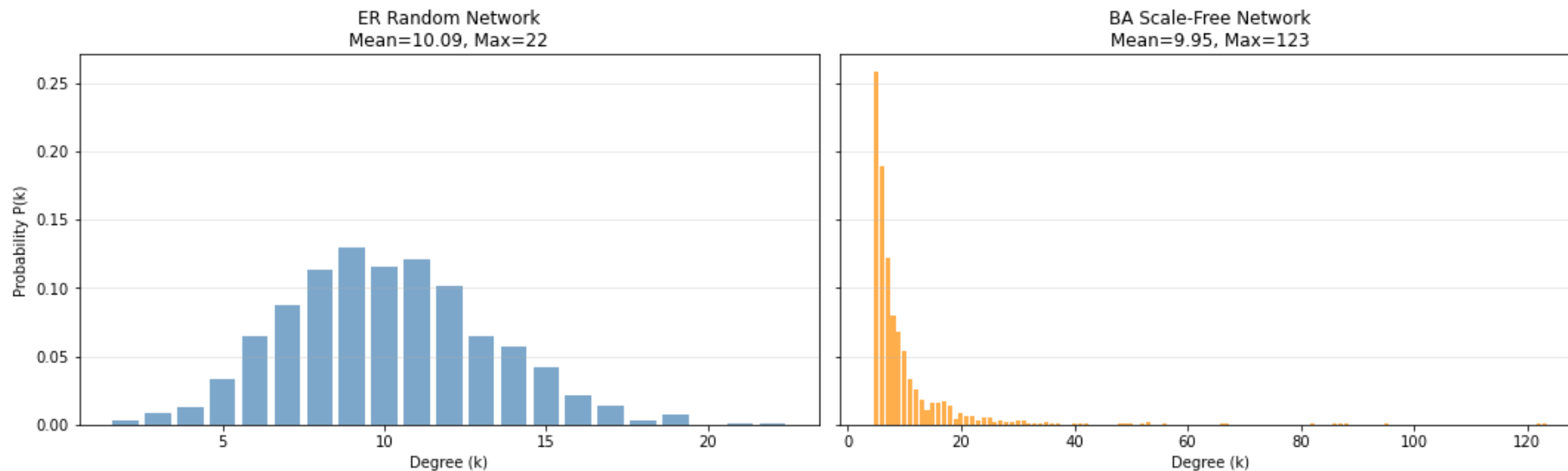


ER random network 결과 해석

Barabasi scale-free network와 비교

④ ER과 BA 네트워크 결과 해석하기

Degree Distribution: ER vs. BA



- 평균 degree인 10 주변에 봉우리가 있는 종 모양
- 최대 degree는 22로, 평균 degree 10에서 크게 벗어나지 않음
- 즉, n 이 충분히 크고 평균 degree가 일정할때, ER network의 degree distribution이 Poisson distribution에 가까워진다

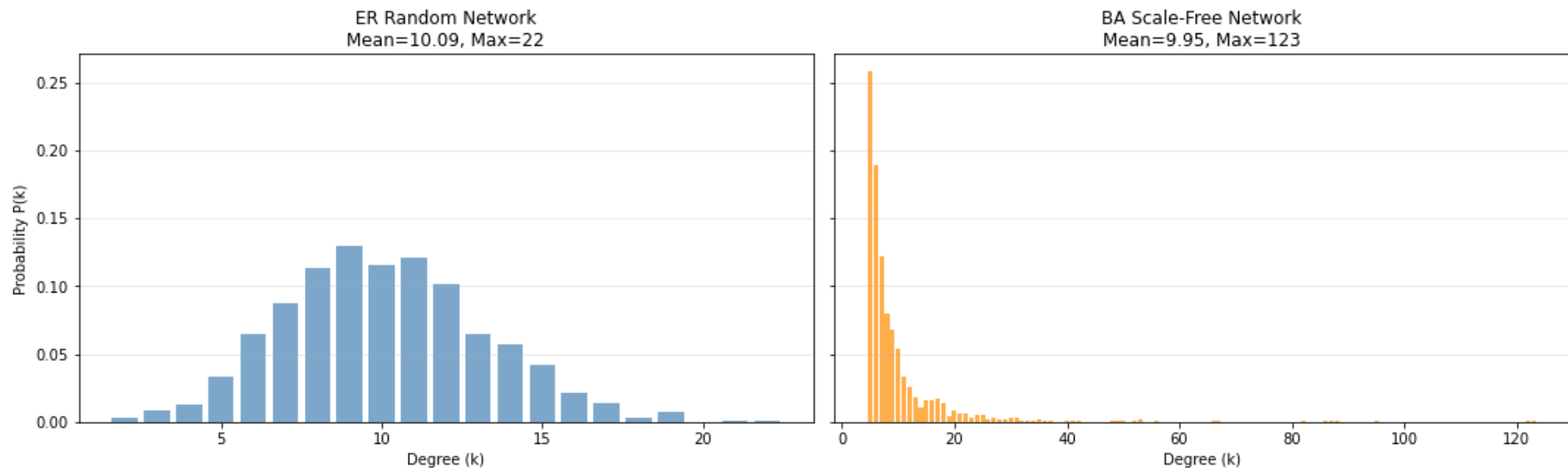
- 낮은 degree를 가진 node가 많이 나타났지만, 동시에 매우 높은 degree를 가진 소수의 node도 존재함
- BA network의 degree distribution은 오른쪽으로 긴 꼬리 보임 (최대 degree도 ER network(22)보다 훨씬 큼)

ER random network 결과 해석

Barabasi scale-free network와 비교

④ ER과 BA 네트워크 결과 해석하기

Degree Distribution: ER vs. BA



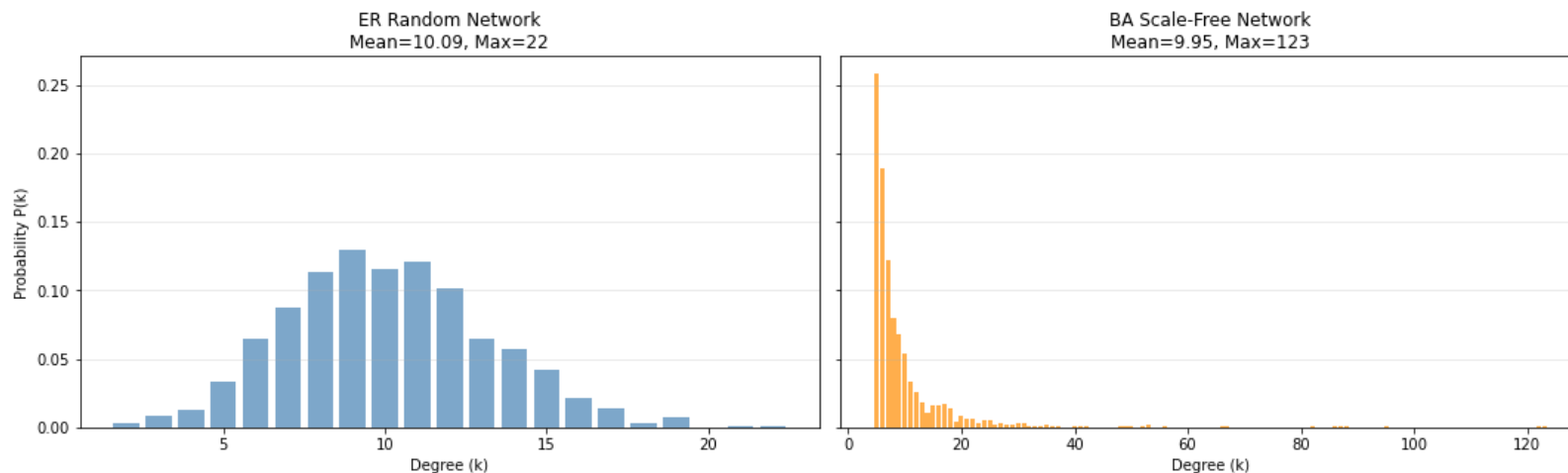
- (공통점) 두 네트워크의 평균 degree는 약 10으로 유사하지만
- (차이점) ER에서는 link가 node들 사이에 비교적 고르게 분배되는 반면, BA에서는 link가 소수의 hub에 집중됨

ER random network 결과 해석

Barabasi scale-free network와 비교

⑤ 두 분포의 모양이 왜 다를까?

Degree Distribution: ER vs. BA



- 모든 node 쌍이 동일한 확률 p 로 독립적으로 연결됨 (즉, 연결이 많은 node라고 해서 다음 link를 얻을 확률이 높아지지 않음)
- 따라서 특정 node에 link가 계속 몰릴 이유가 X, 대부분의 node가 기대 평균 degree 주변에 분포

→ 평균 주변에 node가 집중된 종 모양의 degree distribution, 특별한 hub 없음

- Growth: 네트워크가 성장한다 (네트워크에 새로운 node가 계속 추가된다)
- Preferential attachment: 새로운 node는 이미 degree가 높은 node와 연결될 가능성이 더 높다 (Rich get richer)

→ 대부분의 node는 적은 수의 연결만 가지고, 소수의 node만 매우 많은 연결을 가진다.

ER random network 결과 해석

Barabasi scale-free network와 비교

⑥ 실제 사회 네트워크는 어느 쪽에 더 가까울까?

ER보다는 BA의 특징에 더 가까운 경우가 많음

ER은 모든 노드가 다른 노드와 동일한 확률로 연결된다고 가정하지만,
실제 사회에서는 이미 많은 연결을 가진 사람이 새로운 연결을 만들 기회도 더 많을 수 있음
(예시: 팔로워 많은 계정)

그러나 실제 사회의 연결을 사회적 요인에도 영향을 받기 때문에, 완전히 BA와 같지는 않음

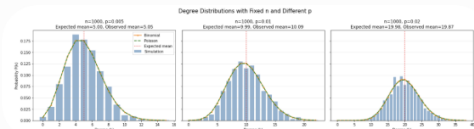
친구의 친구, 같은 학교/직장, 공통 관심사 등도 새로운 관계 형성에 영향을 미침
(예시: 단순히 팔로워 많은 계정이라고 나도 다 팔로우하지는 않음. 동문이나 같은 회사 사람끼리 팔로우 하기도 하고.)

감사합니다

Appendix

LLM 활용 방법

결과 해석 및 검증



이렇게 나왔는데 정상이야?

Worked for 9s

응, 정상적으로 잘 나온 결과야. 오히려 과제에서 기대하는 현상이 아주 선명하게 나타났어.

1. 평균 degree가 예상대로 증가함

p	기대 평균 degree	관측 평균 degree
0.005	5.00	5.05
0.010	9.99	10.09
0.020	19.98	19.87

관측 평균이 기대 평균과 매우 가까워.

예를 들어 $p = 0.02$ 에서는:

$$E[K] = (n - 1)p = 999 \times 0.02 = 19.98$$

이고 실제 관측 평균은 19.87이야. 난수로 생성한 하나의 네트워크이기 때문에 완전히 동일하지 않은 것이 정상이고, 이 정도 차이는 매우 작아.

2. p 가 증가하면서 봉우리가 오른쪽으로 이동함

그래프의 중심이 대략 다음과 같이 이동했어.

체크리스트 확인

HW3_Random Network.ipynb
File

이게 지금까지 작성한 파일이야

Worked for 38s >

확인해보니 (1단계)는 제대로 완료됐어. 코드와 실행 결과 모두 정상이고, 과제 평가 기준인 "대칭, 대각선 0, 작은 예시와 코드의 일치"도 충족해.

확인 결과

점검 항목	결과
작은 adjacency matrix 예시	정상
$n, p, seed$ 명시	정상
$G(n, p)$ 방식으로 연결 생성	정상
self-loop 제거	정상
대칭행렬 생성	정상
행렬 크기 1000×1000	정상

Appendix LLM 활용 방법

답변 구성

4.1. ER random network와 Barabasi scale-free network가 각각 어떻게 생겼나

- ER network 분포 그리기 & 결과 해석
- Barabasi scale-free network 분포 그리기 & 결과 해석

4.2. 두 분포의 모양이 왜 다른가

4.3. 실제 사회 네트워크는 어느 쪽에 가까운가? (근거와 함께)

아니 BA 네트워크를 생성해서 그래프 그리는 건 들어갔으면 좋겠어. 이렇게 4.1, 4.2, 4.3 구성으로 답안 작성하고 싶어

Worked for 1m 10s >

응, 그 구성이라면 범위를 적절하게 잡을 수 있어.

- 4.1: ER과 BA 네트워크를 실제로 생성하고 degree distribution 비교
- 4.2: 생성 규칙 때문에 분포가 왜 달라지는지 설명
- 4.3: 실제 사회 네트워크에 대한 본인의 판단

BA 코드는 넣되, 앞에서처럼 이론을 지나치게 확장하지 않고 과제에서 요구한 내용에만 맞추자.

코드 질문

아까는 조건별 실험 함수 만들고 -> 그리고 순서로 했었는데, 이번에는 조건별 실험 함수 만들 필요는 없나?

이번에도 사실 조건별 실험 함수를 사용하고 있어. 새로 만들 필요가 없었던 이유는 앞에서 만든 함수를 그대로 재사용했기 때문이야.

앞에서 만든 함수가 이것이었지.

</> Python



Run

```
def generate_er_degrees(n, p, seed=42):
    rng = np.random.default_rng(seed)

    random_values = rng.random((n, n))
    connections = random_values < p

    upper_triangle = np.triu(connections, k=1)

    A = upper_triangle + upper_triangle.T
    A = A.astype(np.int8)

    degrees = A.sum(axis=1)
```