

# HW1

January 20, 2025

1 1

```
[4]: def fib(n):  
    if n == 0:  
        return "0"  
    elif n == 1:  
        return "01"  
  
    prevprev = "0"  
    prev = "01"  
    newstr = "".join([prev, prevprev])  
    for i in range(2, n):  
        prevprev = prev  
        prev = newstr  
        newstr = "".join([prev, prevprev])  
  
    return newstr
```

```
[5]: fib(1)
```

```
[5]: '01'
```

```
[6]: fib(3)
```

```
[6]: '01001'
```

```
[7]: for x in range(10):  
    print(f"S{x} = {fib(x)}")
```

```
S0 = 0  
S1 = 01  
S2 = 010  
S3 = 01001  
S4 = 01001010  
S5 = 0100101001001  
S6 = 010010100100101001010  
S7 = 01001010010010100101001001001001  
S8 = 01001010010010100101001001010010010100100101001010
```

S9 = 01001010010010100101001001010010010100101001010010100100101001001010010010100101  
00100101001001

**2 2**

```
[9]: def seq_count(seq, idx = None, prevElem = None, count = 0, highestCount = 0):
    if idx == len(seq):
        # take the highest of the current count and the global highest
        return max(count, highestCount)
    elif idx == 0:
        # start the recursion
        return seq_count(seq, 1, seq[0], 1, 1)

    if prevElem == seq[idx]:
        # add a count
        return seq_count(seq, idx + 1, seq[idx], count + 1, highestCount)
    else:
        # start a new count
        highestCount = max(count, highestCount)
        return seq_count(seq, idx + 1, seq[idx], 1, highestCount)
```

```
[10]: seq_count([1, 3, 1, 1, 3, 3, 4, 4, 4])
```

[10]: 3

```
[11]: seq_count((1, 3, 1, 1, 1, '1', 1, [3, 3, 3, 3], 3, 4, 0))
```

```
[11]: 3
```

```
[12]: seq_count([[1], [1], [1], 1, 3, 3, 2, 3, 3, 3, 3, 2, 4, 0])
```

[12] : 4

```
[13]: seq_count(('G', 'g', 'a', "a", "'a'", 2, 's', 's'))
```

[13] : 3

```
[14]: seq_count([3, 1, int(True), 1, 1, 2, 3, 3])
```

```
[14]: 4
```

```
[15]: seq_count((1, 3, None, 3, 3, 1, 3, 3, 4, 0))
```

[15]: 2

```
[23]: # brute force
def pattern_count(seq, pattern, idx = 0, maxCount = 0):
    if idx == len(seq):
```

```

    return maxCount

    # do calculation starting from this idx
    i, j = idx, 0
    count = 0
    while i < len(seq) and seq[i] == pattern[j]:
        i += 1
        j = (j + 1) % len(pattern)
        if j == 0:
            # we have cycled through all elements in the pattern
            count += 1

    maxCount = max(count, maxCount)
    return pattern_count(seq, pattern, idx + 1, maxCount)

```

```
[24]: pattern_count([0, 1, 2, 1, 2, 3, 1, 2, 1, 2, 1, 2, 4, 1, 2], (1, 2))
```

```
[24]: 3
```

```
[25]: pattern_count([], [2])
```

```
[25]: 0
```

```
[26]: pattern_count(['ab', 'ab', 'a', 'a', 'b'], 'ab')
```

```
[26]: 1
```

```
[27]: pattern_count('CGGACTACTAGACT', 'ACT')
```

```
[27]: 2
```

```
[28]: pattern_count((1, (1, 1, 1, 1), 2, 1, 1, 1), [1, 1])
```

```
[28]: 1
```

```
[29]: pattern_count(['ab', 'ab', 'a', 'a', 'b'], ('ab',))
```

```
[29]: 2
```

### 3 3

(a)

```

[1]: import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom

n, p = 10, 0.4

```

```

sample_size = 10000

# (i)
def inversion_method_binomial(n, p, size):
    u = np.random.uniform(0, 1, size)
    samples = []
    for prob in u:
        cumulative_prob = 0
        for k in range(n + 1):
            cumulative_prob += binom.pmf(k, n, p)
            if prob <= cumulative_prob:
                samples.append(k)
                break
    return np.array(samples)

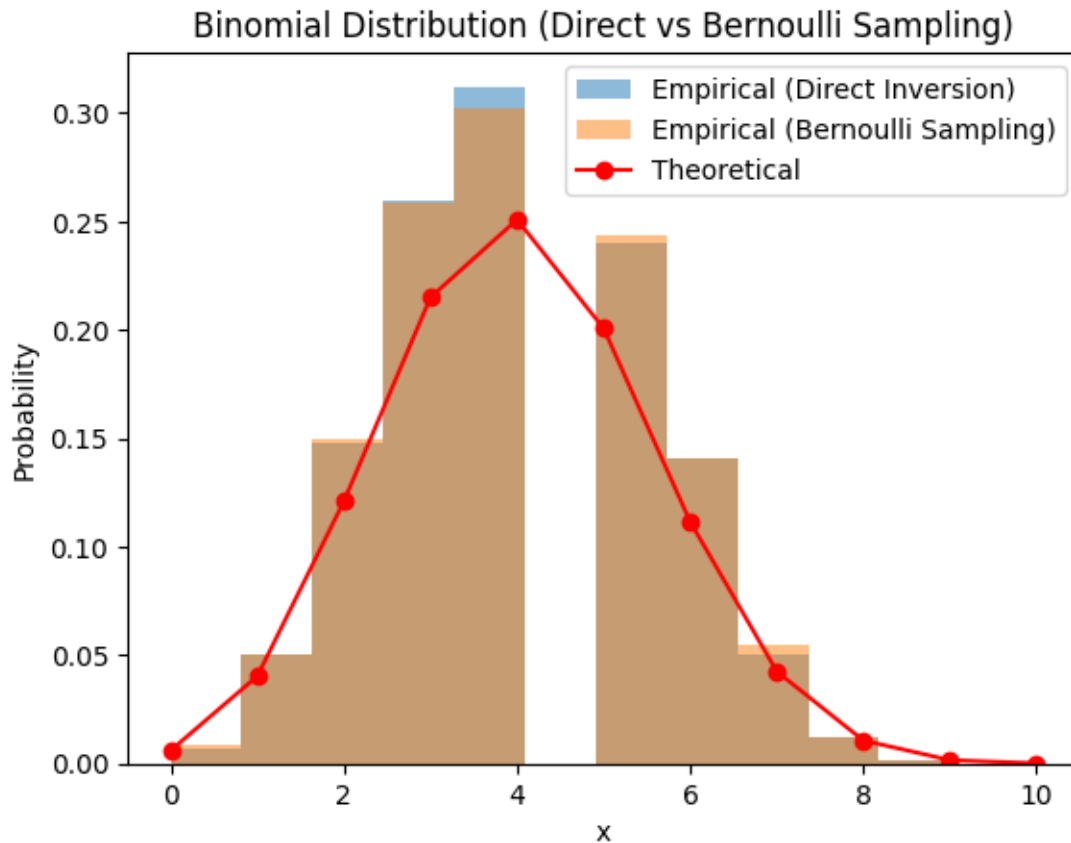
samples_direct = inversion_method_binomial(n, p, sample_size)

# (ii)
def bernoulli_to_binomial(n, p, size):
    samples = []
    for _ in range(size):
        bernoulli_trials = np.random.uniform(0, 1, n) < p
        samples.append(np.sum(bernoulli_trials))
    return np.array(samples)

samples_bernoulli = bernoulli_to_binomial(n, p, sample_size)

# (iii)
x = np.arange(n + 1)
plt.hist(samples_direct, bins=n+1, density=True, alpha=0.5, label="Empirical_
↳(Direct Inversion)")
plt.hist(samples_bernoulli, bins=n+1, density=True, alpha=0.5, label="Empirical_
↳(Bernoulli Sampling)")
plt.plot(x, binom.pmf(x, n, p), 'ro-', label="Theoretical")
plt.title("Binomial Distribution (Direct vs Bernoulli Sampling)")
plt.xlabel("x")
plt.ylabel("Probability")
plt.legend()
plt.show()

```



```
[2]: from scipy.stats import norm, cauchy
import math

# (i)
c = math.sqrt(2 * math.e / math.pi)

# (ii)
def accept_reject_normal(size):
    samples = []
    total_attempts = 0

    while len(samples) < size:
        y = cauchy.rvs() # Cauchy distribution
        u = np.random.uniform(0, 1)
        total_attempts += 1

        #  $U * c * g(y) \leq f(y)$ 
        if u <= norm.pdf(y) / (c * cauchy.pdf(y)):
            samples.append(y)
```

```

    acceptance_prob = len(samples) / total_attempts
    return np.array(samples), acceptance_prob

size = 10000
samples_accept_reject, empirical_prob = accept_reject_normal(size)

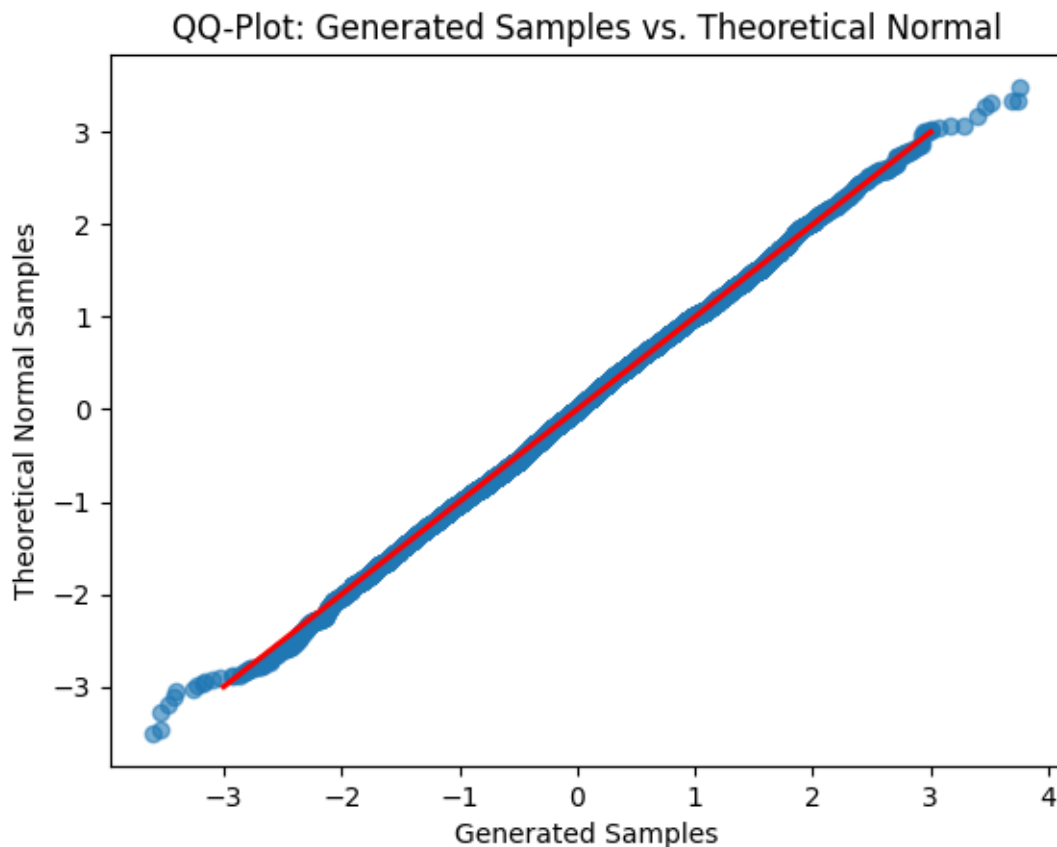
# (iii)
theoretical_prob = 1 / c
print(f"Theoretical Acceptance Probability: {theoretical_prob}")
print(f"Empirical Acceptance Probability: {empirical_prob}")

# (iv)
norm_samples = np.random.normal(size=size)
plt.figure()
plt.scatter(np.sort(samples_accept_reject), np.sort(norm_samples), alpha=0.6)
plt.plot([-3, 3], [-3, 3], color="red", lw=2)
plt.title("QQ-Plot: Generated Samples vs. Theoretical Normal")
plt.xlabel("Generated Samples")
plt.ylabel("Theoretical Normal Samples")
plt.show()

```

Theoretical Acceptance Probability: 0.7601734505331403

Empirical Acceptance Probability: 0.7211365111415591



4 4

```
[2]: import pandas as pd
```

```
[103]: county_age_dist = pd.read_csv("county_age_dist.csv")
fips_state = pd.read_csv("fips_state.csv", delimiter=';')
fips_county = pd.read_csv("fips_county.csv")
```

(a)

```
[104]: county_age_dist.head()
```

```
[104]:
```

	fips	0-17	18-24	25-34	35-44	45-54	55-64	65-74	75-84	85+
0	1001	25941	11422	12315	13828	14000	12697	9594	5430	1945
1	1003	86587	37568	44133	46730	49675	52405	43252	23262	8854
2	1005	11057	6162	6603	5907	6490	6377	5255	2795	1074
3	1007	9671	5241	5788	5472	6707	5563	4270	2555	638
4	1009	25671	11360	12635	13570	14737	14123	12106	6560	2022

```
[105]: fips_county.head()
```

```
[105]:      fips      name  info
0  01000      Alabama  NaN
1  01001  Autauga County  NaN
2  01003  Baldwin County  NaN
3  01005  Barbour County  NaN
4  01007    Bibb County  NaN
```

```
[106]: fips_county['fips'] = pd.to_numeric(fips_county['fips'], errors='coerce')
```

```
[107]: fips_county = fips_county.dropna(subset=['fips'])
```

```
[108]: fips_county['fips'] = fips_county['fips'].astype('int64')
```

```
[109]: fips_county.head()
```

```
[109]:      fips      name  info
0   1000      Alabama  NaN
1   1001  Autauga County  NaN
2   1003  Baldwin County  NaN
3   1005  Barbour County  NaN
4   1007    Bibb County  NaN
```

```
[110]: data = pd.merge(county_age_dist, fips_county, on="fips")
```

```
[111]: data.head()
```

```
[111]:      fips  0-17  18-24  25-34  35-44  45-54  55-64  65-74  75-84  85+  \
0   1001  25941  11422  12315  13828  14000  12697   9594   5430  1945
1   1003  86587  37568  44133  46730  49675  52405  43252  23262  8854
2   1005  11057   6162   6603   5907   6490   6377   5255   2795  1074
3   1007   9671   5241   5788   5472   6707   5563   4270   2555   638
4   1009  25671  11360  12635  13570  14737  14123  12106   6560  2022
```

```
      name  info
0  Autauga County  NaN
1  Baldwin County  NaN
2  Barbour County  NaN
3    Bibb County  NaN
4  Blount County  NaN
```

```
[112]: fips_state.head()
```

```
[112]:      FIPS      STATE
0      1      ALABAMA
1      2      ALASKA
2      4      ARIZONA
3      5      ARKANSAS
4      6  CALIFORNIA
```



```
[113]: data['state_fips'] = data['fips'] // 1000
```

```
[114]: data.head()
```

```
[114]:
```

	fips	0-17	18-24	25-34	35-44	45-54	55-64	65-74	75-84	85+	\
0	1001	25941	11422	12315	13828	14000	12697	9594	5430	1945	
1	1003	86587	37568	44133	46730	49675	52405	43252	23262	8854	
2	1005	11057	6162	6603	5907	6490	6377	5255	2795	1074	
3	1007	9671	5241	5788	5472	6707	5563	4270	2555	638	
4	1009	25671	11360	12635	13570	14737	14123	12106	6560	2022	

	name	info	state_fips
0	Autauga County	NaN	1
1	Baldwin County	NaN	1
2	Barbour County	NaN	1
3	Bibb County	NaN	1
4	Blount County	NaN	1

```
[115]: data = pd.merge(data, fips_state, left_on="state_fips", right_on="FIPS",
    ↪how="inner")
```

```
[116]: data.head()
```

```
[116]:
```

	fips	0-17	18-24	25-34	35-44	45-54	55-64	65-74	75-84	85+	\
0	1001	25941	11422	12315	13828	14000	12697	9594	5430	1945	
1	1003	86587	37568	44133	46730	49675	52405	43252	23262	8854	
2	1005	11057	6162	6603	5907	6490	6377	5255	2795	1074	
3	1007	9671	5241	5788	5472	6707	5563	4270	2555	638	
4	1009	25671	11360	12635	13570	14737	14123	12106	6560	2022	

	name	info	state_fips	FIPS	STATE
0	Autauga County	NaN	1	1	ALABAMA
1	Baldwin County	NaN	1	1	ALABAMA
2	Barbour County	NaN	1	1	ALABAMA
3	Bibb County	NaN	1	1	ALABAMA
4	Blount County	NaN	1	1	ALABAMA

```
[117]: data = data.rename(columns=lambda x: x.strip().capitalize())
data.head()
```

```
[117]:
```

	Fips	0-17	18-24	25-34	35-44	45-54	55-64	65-74	75-84	85+	\
0	1001	25941	11422	12315	13828	14000	12697	9594	5430	1945	
1	1003	86587	37568	44133	46730	49675	52405	43252	23262	8854	
2	1005	11057	6162	6603	5907	6490	6377	5255	2795	1074	
3	1007	9671	5241	5788	5472	6707	5563	4270	2555	638	
4	1009	25671	11360	12635	13570	14737	14123	12106	6560	2022	

	Name	Info	State_fips	Fips	State
0	Autauga County	NaN	1	1	ALABAMA
1	Baldwin County	NaN	1	1	ALABAMA
2	Barbour County	NaN	1	1	ALABAMA
3	Bibb County	NaN	1	1	ALABAMA
4	Blount County	NaN	1	1	ALABAMA

0	Autauga County	NaN	1	1	ALABAMA
1	Baldwin County	NaN	1	1	ALABAMA
2	Barbour County	NaN	1	1	ALABAMA
3	Bibb County	NaN	1	1	ALABAMA
4	Blount County	NaN	1	1	ALABAMA

```
[118]: data = data.drop(columns=['Info'])
data.head()
```

```
[118]:
```

	Fips	0-17	18-24	25-34	35-44	45-54	55-64	65-74	75-84	85+	\
0	1001	25941	11422	12315	13828	14000	12697	9594	5430	1945	
1	1003	86587	37568	44133	46730	49675	52405	43252	23262	8854	
2	1005	11057	6162	6603	5907	6490	6377	5255	2795	1074	
3	1007	9671	5241	5788	5472	6707	5563	4270	2555	638	
4	1009	25671	11360	12635	13570	14737	14123	12106	6560	2022	

	Name	State_fips	Fips	State
0	Autauga County		1	ALABAMA
1	Baldwin County		1	ALABAMA
2	Barbour County		1	ALABAMA
3	Bibb County		1	ALABAMA
4	Blount County		1	ALABAMA

```
[119]: for col in data.columns:
data[col] = data[col].map(lambda x: x.strip() if isinstance(x, str) else x)
data.head(4)
```

```
[119]:
```

	Fips	0-17	18-24	25-34	35-44	45-54	55-64	65-74	75-84	85+	\
0	1001	25941	11422	12315	13828	14000	12697	9594	5430	1945	
1	1003	86587	37568	44133	46730	49675	52405	43252	23262	8854	
2	1005	11057	6162	6603	5907	6490	6377	5255	2795	1074	
3	1007	9671	5241	5788	5472	6707	5563	4270	2555	638	

	Name	State_fips	Fips	State
0	Autauga County		1	ALABAMA
1	Baldwin County		1	ALABAMA
2	Barbour County		1	ALABAMA
3	Bibb County		1	ALABAMA

(b)

```
[120]: def calculate_proportions(row):
total_pop = sum([row[f"{age_group}"] for age_group in ["0-17", "18-24",
↵ "25-34", "35-44", "45-54", "55-64", "65-74", "75-84", "85+"]])
cpe = sum([row["65-74"], row["75-84"], row["85+"]]) / total_pop
cpy = sum([row["0-17"], row["18-24"]]) / total_pop
return pd.Series({"CPY": cpy, "CPE": cpe})
```

```

proportions = data.apply(calculate_proportions, axis=1)
data = pd.concat([data, proportions], axis=1)

# View first 4 rows as per the test
data.head(4)

```

```

[120]:
   Fips  0-17  18-24  25-34  35-44  45-54  55-64  65-74  75-84  85+  \
0  1001  25941  11422  12315  13828  14000  12697   9594   5430  1945
1  1003  86587  37568  44133  46730  49675  52405  43252  23262  8854
2  1005  11057   6162   6603   5907   6490   6377   5255   2795  1074
3  1007   9671   5241   5788   5472   6707   5563   4270   2555   638

```

	Name	State_fips	Fips	State	CPY	CPE
0	Autauga County	1	1	ALABAMA	0.348627	0.158334
1	Baldwin County	1	1	ALABAMA	0.316346	0.192037
2	Barbour County	1	1	ALABAMA	0.332927	0.176411
3	Bibb County	1	1	ALABAMA	0.324845	0.162575