به نام خدا

گزارش کار تمرین ها

درس آمار احتمال مهندسی بهار 1404

ماهان بانشی

فصل 4:

سوال 2:

خواسته های سوال:

در این سوال ما به بررسی ارتفاع پیک های مختلف تابع گاوسی و ارتباط آن با انحراف معیار می پردازیم.

همچنین مفهوم انتگرال گاوسی و عوامل تاثیرگذار در آن مثل انحراف معیارو دامنه را بررسی می کنیم.

کد سوال:

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

# gaussian function
def gaussian(x, mu, sigma, normalize=True):
    factor = 1 / (np.sqrt(2 * np.pi) * sigma) if normalize else 1
    return factor * np.exp(-((x - mu) ** 2) / (2 * sigma ** 2))

x = np.linspace(-5, 5, 1000)
sigmas = [0.5, 1, 2]

# create gaussians
```

```python
plt.figure(figsize=(8, 5))
for sigma in sigmas:
    y = gaussian(x, mu=0, sigma=sigma)
    plt.plot(x, y, label=f'σ = {sigma}')

plt.xlabel('x')
plt.ylabel('Gaussian Distribution')
plt.title('Different Gaussians with Various σ')
plt.legend()
plt.grid(True)
plt.show()

# calculate Integral for 2 domains
domains = [(-3, 3), (-5, 5)]
for domain in domains:
    mask = (x >= domain[0]) & (x <= domain[1])
    x_domain = x[mask]

    print(f"\nDomain: {domain}")
    for sigma in sigmas:
        y = gaussian(x, mu=0, sigma=sigma)
        y_domain = y[mask]


        sum_gaussian = np.sum(y_domain)
        integral_gaussian = np.trapz(y_domain, x_domain)

        print(f"σ = {sigma}: Sum = {sum_gaussian:.4f}, Integral = 
{integral_gaussian:.4f}")

# without Normalization
plt.figure(figsize=(8, 5))
for sigma in sigmas:
    y = gaussian(x, mu=0, sigma=sigma, normalize=False)
    plt.plot(x, y, label=f'σ = {sigma} (No Normalization)')

plt.xlabel('x')
plt.ylabel('Gaussian Without Normalization')
plt.title('Gaussians Without Normalization Factor')
plt.legend()
plt.grid(True)
plt.show()
```

توضیحات کلی:

ابتدا یک تابع گاوسی را تعریف کرده ایم. سپس در دامنه و میانگین مشترک ولی با انحراف

معیار های مختلف 3 تا تابع گوسی کشیدیم. سپس مقدار انتگرال و sum گاوسی را برای هر کدام در دو دامنه ی متفاوت حساب کرده ایم. در آخر نیز توابع گاوسی را بدون normalize شدن plot کردیم تا ببینیم normalize شدن چه تاثیری روی آن ها می گذارد.

بخش های خاص کد:

```
def gaussian(x, mu, sigma, normalize=True):
    factor = 1 / (np.sqrt(2 * np.pi) * sigma) if normalize else 1
    return factor * np.exp(-((x - mu) ** 2) / (2 * sigma ** 2))
```

در اینجا تابع گاوسی را تعریف کرده ایم. خط اول مربوط به نرمال سازی تابع است و خط دوم هم همان فرمول گاوسی را استفاده می کند.

```
integral_gaussian = np.trapz(y_domain, x_domain)
```

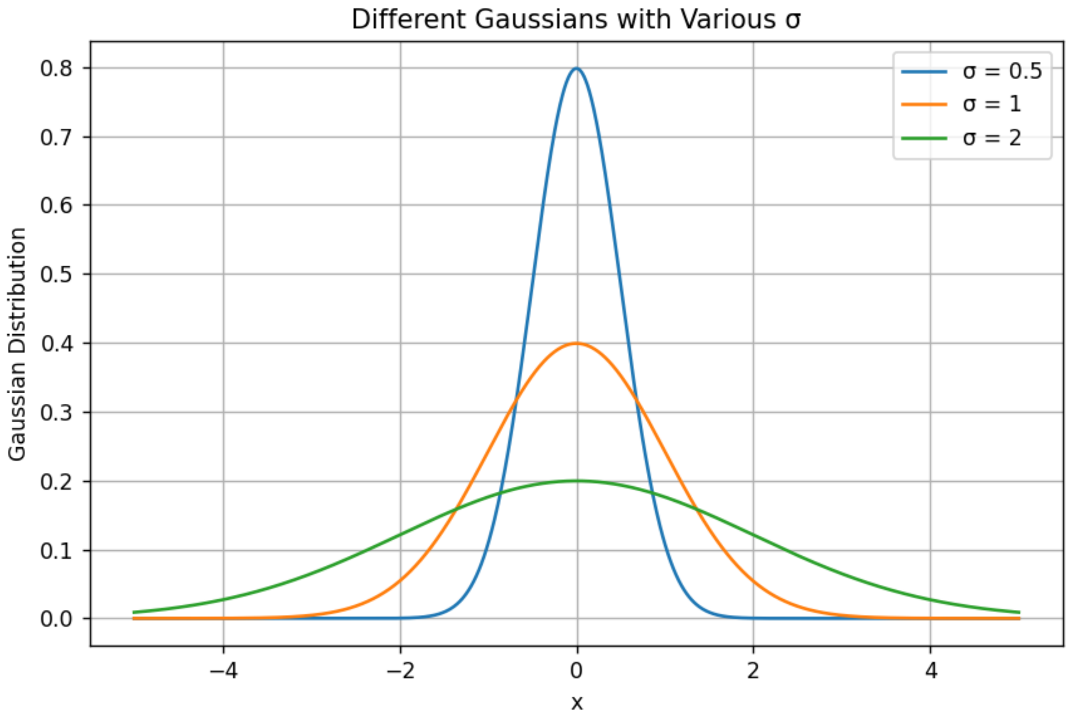
ما انتگرال گاوسی را به روش ذوزنقه ای حساب کردیم. نامپای خودش تابع آماده برای این کار را دارد.

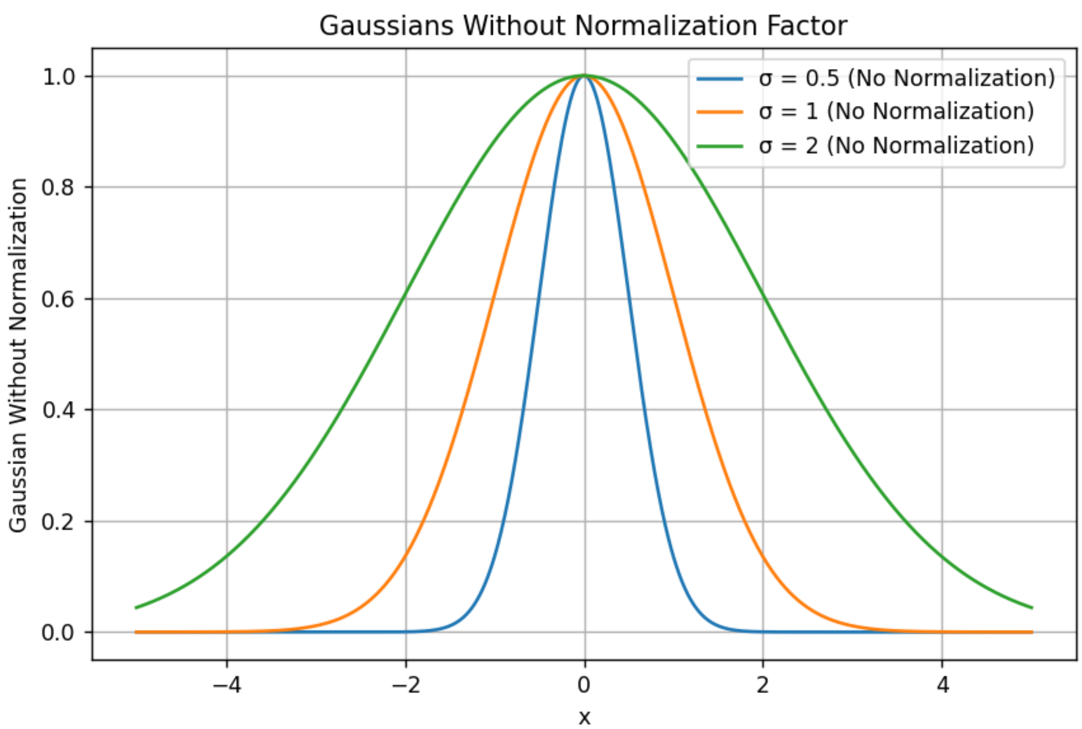
در آخر نیز نمودار ها را کشیدیم.

خروجی:

```
Domain: (-3, 3)
d:\University\basic\Amar\homeworks\HW3\codes\ch4q2
tead, or one of the numerical integration function
  integral_gaussian = np.trapz(y_domain, x_domain)
σ = 0.5: Sum = 99.9000, Integral = 1.0000
σ = 1: Sum = 99.6329, Integral = 0.9973
σ = 2: Sum = 86.5908, Integral = 0.8661

Domain: (-5, 5)
σ = 0.5: Sum = 99.9000, Integral = 1.0000
σ = 1: Sum = 99.8999, Integral = 1.0000
σ = 2: Sum = 98.6681, Integral = 0.9876
PS D:\University\basic\Amar\homeworks\HW3\codes>
```


Different Gaussians with Various σ
σ = 0.5
σ = 1
σ = 2
Gaussian Distribution
x
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
−4
−2
0
2
4


Gaussians Without Normalization Factor
σ = 0.5 (No Normalization)
σ = 1 (No Normalization)
σ = 2 (No Normalization)
Gaussian Without Normalization
x
0.0
0.2
0.4
0.6
0.8
1.0
−4
−2
0
2
4

نتیجه:

هر چه انحراف معیار بیشتر باشد تیزی نوک قله کاهش می یابد و تراکم نزدیک قله کمتر می شود. در ضمن نرمال سازی نیز خیلی تاثیر میگذارد روی پیک ها و می تواند پیک را خیلی ملایم کند.

مقدار جمع و انتگرال توابع گاوسی 1 است اگر دامنه بینهایت باشد. هر چه دامنه را کمتر کنیم این حاصل کمتر می شود.

سوال 4:

خواسته ی سوال:

هدف اصلی تحلیل واریانس نمونه‌ای و مقایسه آن با واریانس جامعه با استفاده از پارامترهای مختلف است. از داده‌های تصادفی در دامنه‌های مختلف استفاده می‌شود، و تأثیر تغییر اندازه نمونه و حذف ضریب نرمال‌سازی بررسی خواهد شد.

کد سوال:

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

def perform_experiment(range_start, range_end, sample_sizes, repeats):
    differences = []

    for _ in range(repeats):
        difference_for_sizes = []
        for size in sample_sizes:
            # create random data
            data = np.random.randint(range_start, range_end + 1, size=size)

            # calculate Variances
            variance_population = np.var(data, ddof=0)
```

```
            variance_sample = np.var(data, ddof=1)

            difference = abs(variance_sample - variance_population)
            difference_for_sizes.append(difference)

        differences.append(difference_for_sizes)

    mean_differences = np.mean(differences, axis=0)
    std_differences = np.std(differences, axis=0)
    return mean_differences, std_differences

sample_sizes = range(5, 101)
repeats = 25


mean_diff_large_range, std_diff_large_range = perform_experiment(-100, 100,
sample_sizes, repeats)
mean_diff_small_range, std_diff_small_range = perform_experiment(-10, 10,
sample_sizes, repeats)

# plot
plt.figure(figsize=(12, 6))

plt.errorbar(sample_sizes, mean_diff_large_range, yerr=std_diff_large_range,
fmt='o', label='Range: -100 to 100', ecolor='r', capsize=5)
plt.errorbar(sample_sizes, mean_diff_small_range, yerr=std_diff_small_range,
fmt='o', label='Range: -10 to 10', ecolor='b', capsize=5)

plt.xlabel('Sample Size')
plt.ylabel('Mean Difference in Variance')
plt.title('Impact of Sample Size and Data Range on Variance Differences')
plt.legend()
plt.show()
```

توضیح کلی:

ما یک آزمایش را برای دو مجموعه داده که یکی بزرگتر از دیگری است تکرار می کنیم. در هر آزمایش تفاوت میان واریانس نمونه و واریانس جامعه محاسبه می شود و میانگین و انحراف معیار تفاوت ها بررسی می شوند. نمودارها نشان می‌دهند که چگونه اندازه نمونه و بازه داده‌ها بر تفاوت‌ها تاثیر می‌گذارند.

بخش های خاص کد:

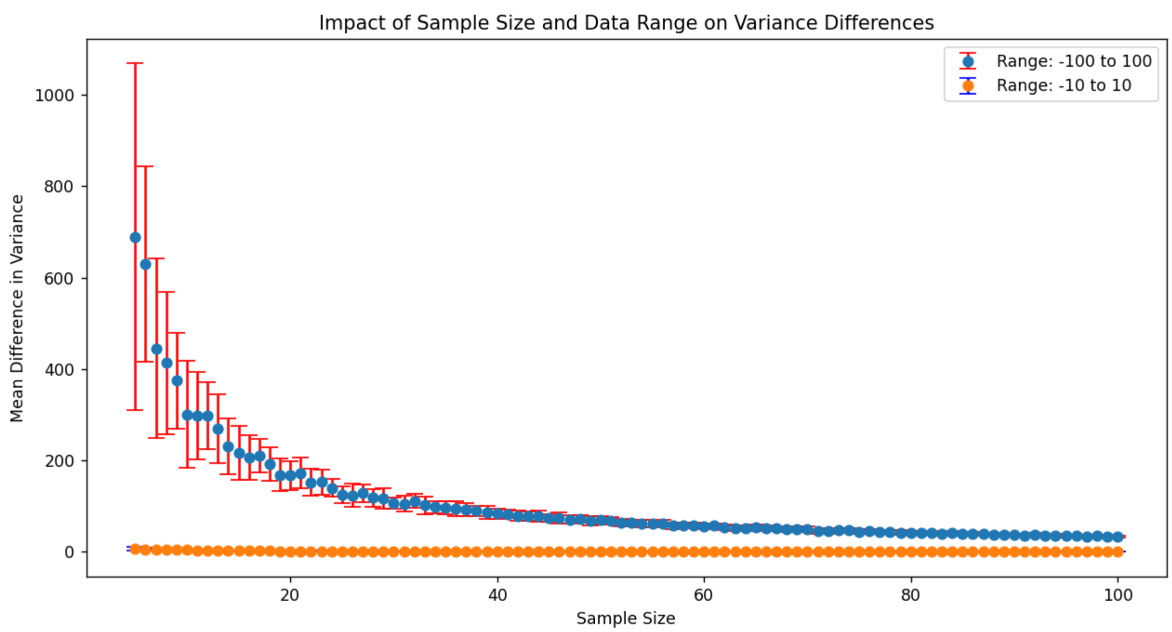
```
plt.errorbar(sample_sizes, mean_diff_large_range, yerr=std_diff_large_range,
fmt='o', label='Range: -100 to 100', ecolor='r', capsize=5)
plt.errorbar(sample_sizes, mean_diff_small_range, yerr=std_diff_small_range,
fmt='o', label='Range: -10 to 10', ecolor='b', capsize=5)
```

این تابع در Matplotlib برای رسم نمودار نقاط همراه با میله‌های خطا (error bars) استفاده می‌شود. این میله‌های خطا معمولاً برای نشان دادن میزان عدم قطعیت (uncertainty) یا تغییرپذیری (variability) در داده‌ها به کار می‌روند.

پارامتر هایش به ترتیب داده‌های محور افقی، داده‌های محور عمودی،

مقدار خطا یا عدم قطعیت برای هر نقطه در محور y، فرمت نقاط داده روی نمودار، برچسب داده، رنگ نوارهای خطا و اندازه خطوط افقی کوچک هستند.

خروجی:



نتیجه:

با افزایش اندازه نمونه (از ۵ تا ۱۰۰)، میانگین تفاوت بین واریانس نمونه‌ای و واریانس جامعه کاهش می‌یابد.

بازه داده ها در میزان این تفاوت تاثیر دارد. وقتی داده‌ها در بازه کوچکتری مانند (‎-10 تا ‎+10) هستند، تفاوت بین واریانس نمونه‌ای و واریانس جامعه نیز کوچک‌تر است. این به دلیل کاهش مقیاس داده‌ها است که مقدار کلی واریانس را کاهش می‌دهد.

نوارهای خطا (Error Bars) نشان‌دهنده تغییرات یا عدم قطعیت نتایج هستند. این تغییرات در نمونه‌های کوچک‌تر به دلیل تنوع بیشتر در داده‌های تصادفی بزرگ‌تر است، اما در نمونه‌های بزرگ تر کاهش می‌یابد.

سوال 9:

خواسته های سوال:

این سوال درباره ی رابطه ی بین دامنه ی بین چارکی (اختلاف چارک اول و سوم) با مقدار انحراف معیار است. یکبار برای توزیع نرمال داده ها و یکبار برای توزیع نمایی. همچنین می خواهیم به رابطه ی

$IQR \approx 1.35\sigma$ نیز برسیم.

کد سوال:

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

# plot histogram
def plot_histogram(data, title, panel):

    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    iqr = q3 - q1

    # calculate mean and standard deviation
    mean = np.mean(data)
    std = np.std(data)

    # Plot histogram
    plt.hist(data, bins='fd', alpha=0.7, color='blue', edgecolor='black')
    plt.axvline(q1, color='black', linestyle='solid', label='Q1 (25th
Percentile)')
    plt.axvline(q3, color='black', linestyle='solid', label='Q3 (75th
Percentile)')
```

```
    plt.axvline(mean - 1.35 * std, color='red', linestyle='dashed', label='-
1.35σ')
    plt.axvline(mean + 1.35 * std, color='red', linestyle='dashed',
label='+1.35σ')
    plt.title(f"{title} (Panel {panel})")
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.legend()
    plt.grid(True)

    return iqr, std

np.random.seed(42)
normal_data = np.random.normal(loc=0, scale=1, size=10000)

# plot
plt.figure(figsize=(10, 10))
plt.subplot(2, 1, 1)
iqr_normal, std_normal = plot_histogram(normal_data, "Normal Distribution", "A")

exponential_data = np.exp(normal_data)

plt.subplot(2, 1, 2)
iqr_exponential, std_exponential = plot_histogram(exponential_data, "Exponential
Distribution", "B")

plt.tight_layout()
plt.show()

# print results
print("Results for Normal Distribution:")
print(f"Standard Deviation (σ): {std_normal:.4f}")
print(f"Interquartile Range (IQR): {iqr_normal:.4f}")
print(f"IQR / σ ≈ {iqr_normal / std_normal:.4f}")

print("\nResults for Exponential Distribution:")
print(f"Standard Deviation (σ): {std_exponential:.4f}")
print(f"Interquartile Range (IQR): {iqr_exponential:.4f}")
print(f"IQR / σ ≈ {iqr_exponential / std_exponential:.4f}")
```

توضیح کلی:

ابتدا تابع کشیدن هیستوگرام را تعریف کرده ایم و همانجا چارک ها، میانگین، انحراف معیار و IQR را حساب کرده ایم. سپس دو مجموعه داده یکی بصورت عادی و یکی بصورت نمایی ساخته ایم و به آن تابع پاس داده ایم. در آخر هم مطلوب های سوال را بررسی و پرینت کرده ایم.

بخش های خاص کد:

```
q1 = np.percentile(data, 25)
```

این تابع نامپای صدک ها را محاسبه می کند. صدک 25 همان چارک اول است.

```
plt.hist(data, bins='fd', alpha=0.7, color='blue', edgecolor='black')
```

از قانون Freedman–Diaconis برای تعیین تعداد binها استفاده می کنیم. این قانون یک روش استاندار برای تعیین تعداد bin ها است که که بین وضوح . هموار بودن داده ها تعادل برقرار می کند.

```
    plt.axvline(mean - 1.35 * std, color='red', linestyle='dashed', label='-1.35σ')
    plt.axvline(mean + 1.35 * std, color='red', linestyle='dashed', label='+1.35σ')
```

دو خط قرمز خطچین رسم می کنیم که مقدار ۱.۳۵ انحراف معیار از میانگین را نشان می‌دهد.

```
normal_data = np.random.normal(loc=0, scale=1, size=10000)
```

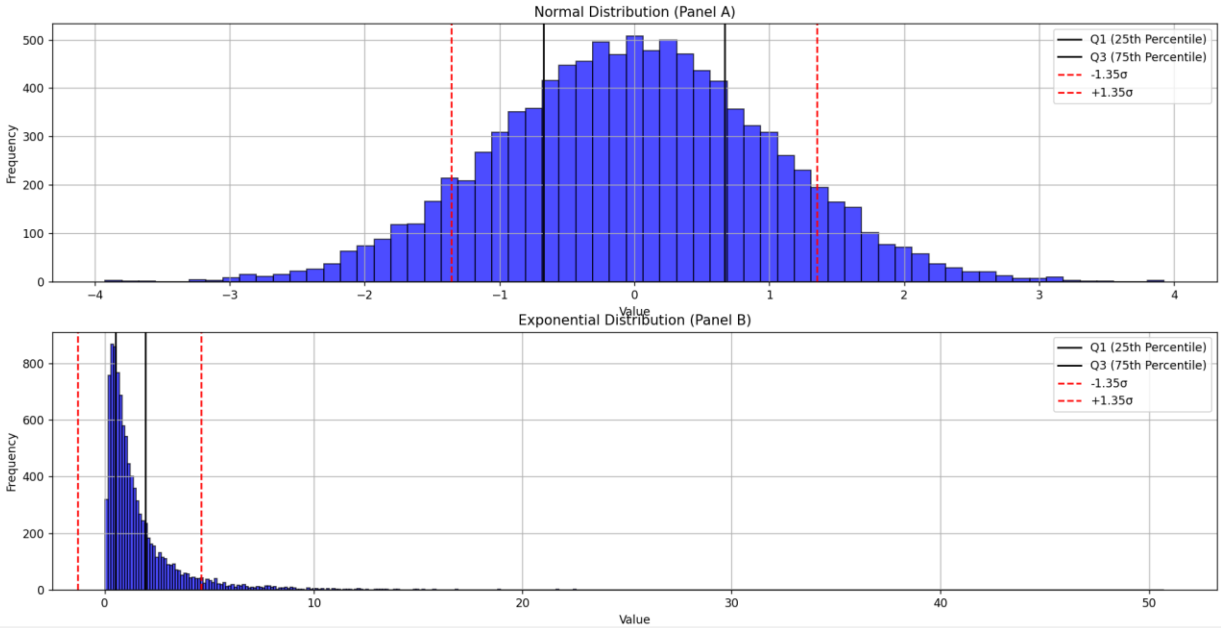
توزیع نرمال 10000 داده با میانگین 0 و انحراف معیار 1.

```
exponential_data = np.exp(normal_data)
```

توضیح داده ب صورت نمایی.

در آخر نیز یک سری از نتایج را پرینت می کنیم.

خروجی:




```
PS D:\University\basic\Amar\homeworks\HW3\codes> python -u "d:
Results for Normal Distribution:
Standard Deviation (σ): 1.0034
Interquartile Range (IQR): 1.3437
IQR / σ ≈ 1.3391

Results for Exponential Distribution:
Standard Deviation (σ): 2.1917
Interquartile Range (IQR): 1.4460
IQR / σ ≈ 0.6597
PS D:\University\basic\Amar\homeworks\HW3\codes>
```

نتایج:

همانطور که در نمودار ها مشخص است، در توزیع نرمال، نسبت IQR/σ تقریباً مقدار ثابتی دارد، در حالی که در توزیع نمایی به علت نا متقارن بودن، این نسبت متفاوت است.

در نتیجه رابطه ی IQR/σ = 1.35 نیز همیشه واقعی نیست و ممکن است این نسبت تغییر کند.

سوال 10:

باید یک تابع برای محاسبه ی FWHM تعریف کنیم. این تابع:

مقادیر را نرمال‌سازی کند (یعنی مقادیر را بین ۰ و ۱ تبدیل کند).

موقعیت پیک را مشخص می کند.

نقاط نیمه‌ماکزیمم (نقاطی که تقریباً برابر 0.5 است) را در محور x قبل و بعد از پیک پیدا کند.

مقدار FWHM را محاسبه کند و بازگرداند.

بعد از آن تابع را روی توزیع گاوسی و توزیع تصادفی تست و نتایج را با هم مقایسه می کنیم.

کد سوال:

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

# create FWHM
def empFWHM(x, y):

    normalized_y = (y - np.min(y)) / (np.max(y) - np.min(y))
    peak_index = np.argmax(normalized_y)

    # find halfMaxes
    pre_half_max_x = x[np.where(normalized_y[:peak_index] >= 0.5)[-1][0]]
    post_half_max_x = x[np.where(normalized_y[peak_index:] >= 0.5)[0][0] +
peak_index]

    # calculate FWHM
    fwhm = post_half_max_x - pre_half_max_x
    return fwhm, pre_half_max_x, post_half_max_x

# experiment
sigma_values = np.linspace(1, 5, 50)
fwhm_empirical = []
fwhm_analytical = []

for sigma in sigma_values:
    x = np.linspace(-8, 8, 1001)
```

```python
    y = np.exp(-x**2 / (2 * sigma**2))

    fwhm_exp, _, _ = empFWHM(x, y)
    fwhm_empirical.append(fwhm_exp)
    fwhm_analytical.append(2.35482 * sigma)

# plot
plt.figure(figsize=(8, 6))
plt.plot(sigma_values, fwhm_empirical, 'ks-', label='Empirical FWHM')
plt.plot(sigma_values, fwhm_analytical, 'd-', color='gray', label='Analytical
FWHM')
plt.xlabel('σ (Standard Deviation)')
plt.ylabel('FWHM')
plt.title('Comparison of Empirical and Analytical FWHM')
plt.legend()
plt.show()

# FWHM and histogram
data = np.random.normal(0, 1, 12345)
hist_values, bin_edges = np.histogram(data, bins=100, density=True)

# middles
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# calculate FWHM
fwhm_histogram, pre_half_max, post_half_max = empFWHM(bin_centers, hist_values)

# plot histogram
plt.figure(figsize=(8, 6))
plt.bar(bin_centers, hist_values, width=(bin_edges[1] - bin_edges[0]),
color='lightblue', label='Histogram')
plt.axvline(pre_half_max, color='r', linestyle='--', label='Half-Max Points')
plt.axvline(post_half_max, color='r', linestyle='--')
plt.xlabel('Value')
plt.ylabel('Density')
plt.title(f'Empirical FWHM = {fwhm_histogram:.2f}')
plt.legend()
plt.show()
```

توضیح کلی:

در این کد به بررسی عرض کامل در نیمه بیشینه (FWHM) برای توزیع گاوسی می‌پردازیم. ابتدا یک تابع برای محاسبه‌ی FWHM به‌صورت تجربی از داده‌های عددی

تعریف می کنیم. سپس این تابع برای توزیع گاوسی با انحراف معیارهای مختلف اعمال و با مقدار تحلیلی مقایسه می‌کنیم. در نهایت، یک توزیع نرمال تصادفی شبیه‌سازی شده و FWHM آن از طریق هیستوگرام محاسبه و نمایش داده می‌شود.

بخش های خاص کد:

```
normalized_y = (y - np.min(y)) / (np.max(y) - np.min(y))
```

در اینجا نرمال سازی کرده ایم. داده ی ماکسیمم 1 و داده ی مینیمم 0 می شود.

```
    pre_half_max_x = x[np.where(normalized_y[:peak_index] >= 0.5)[-1][0]]
    post_half_max_x = x[np.where(normalized_y[peak_index:] >= 0.5)[0][0] +
peak_index]
```

اینجا نیمه ماکسیمم ها را پیدا میکنیم که می شوند آخرین index قبل از پیک و اولین index بعد از پیک به شرطی که مقدارشان از 0.5 بیشتر باشد.

```
    fwhm_empirical.append(fwhm_exp)
    fwhm_analytical.append(2.35482 * sigma)
```

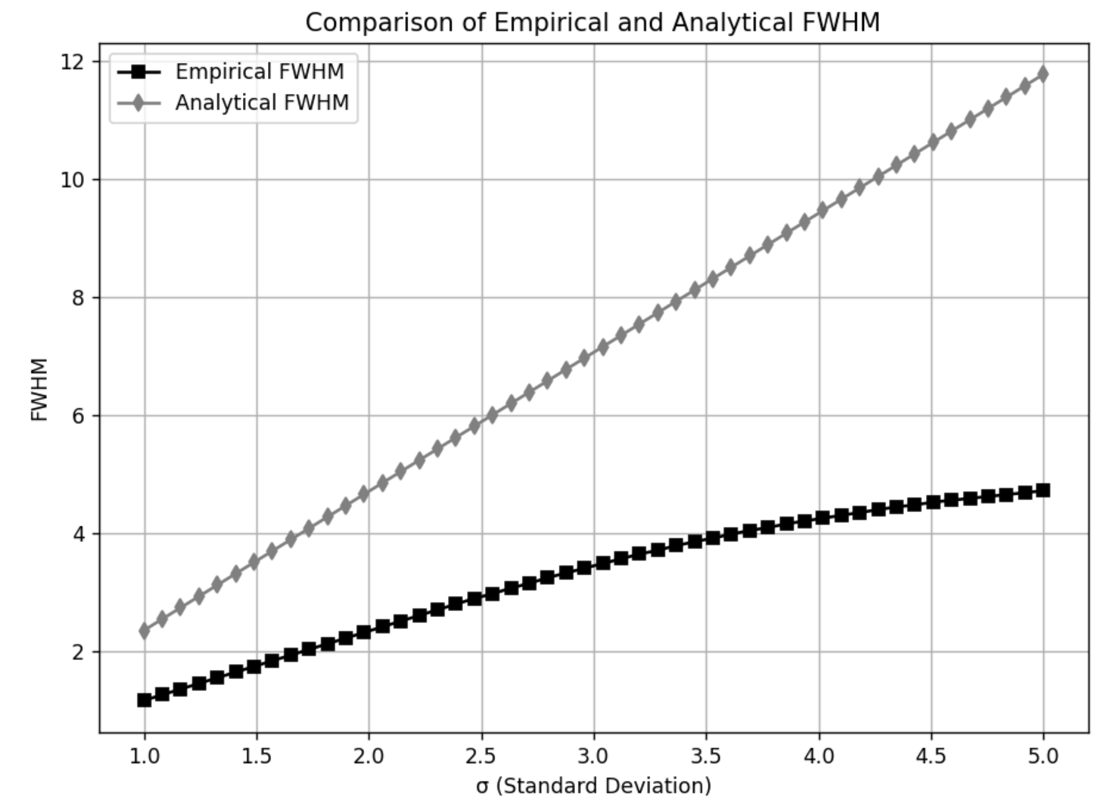
در خط بالایی مقادیر تجربی به لیست اضافه می شود و در خط پایینی کقادیر تحلیلی (بر اساس فرمول گاوس) به لیست اضافه می شوند.

```
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
```

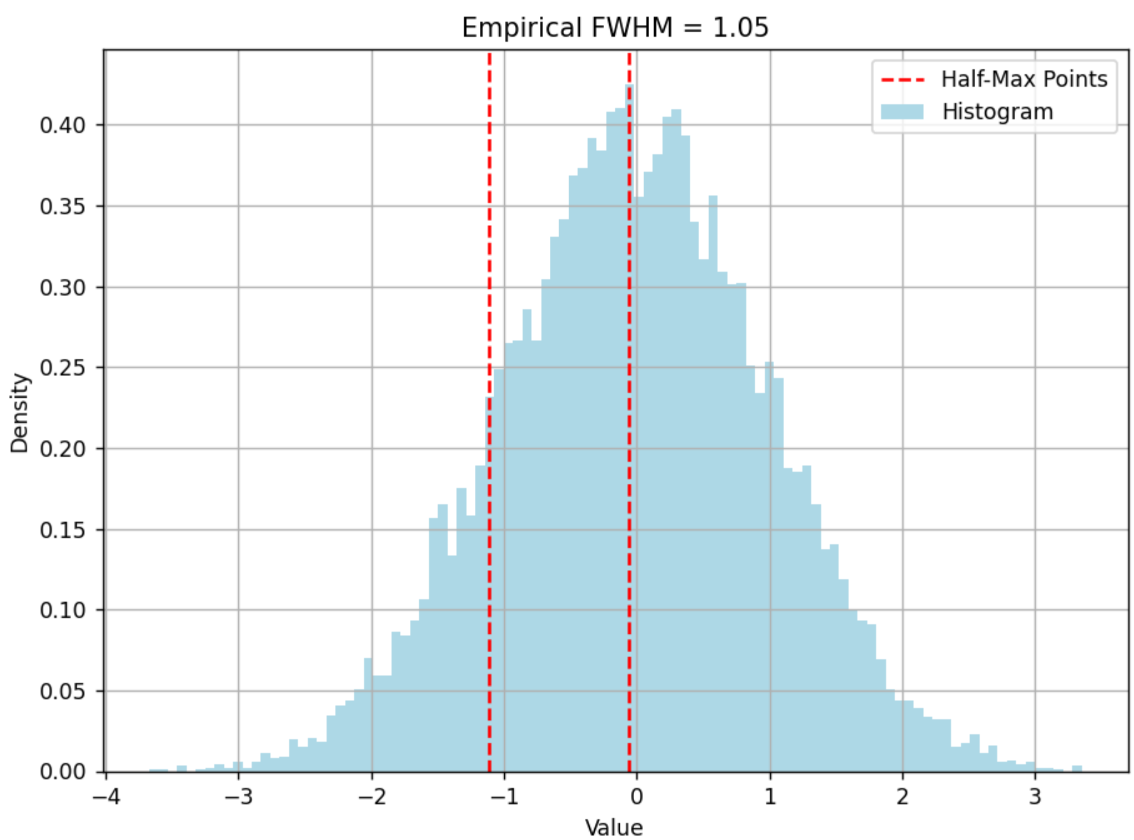
اینجا مرکز هر ستون را در هیستوگرام محاسبه می کنیم.

در آخر نیز تمام نمودار های تولید شده را با plot می کشیم.

خروجی:



Comparison of Empirical and Analytical FWHM
Empirical FWHM
Analytical FWHM
FWHM
σ (Standard Deviation)



Empirical FWHM = 1.05
Half-Max Points
Histogram
Density
Value

نتیجه:

FWHM تجربی با با مقدار تحلیلی تا حدودی همانگی دارد. هر چند که رشد تابع تحلیلی بیشتر است.

در هیستوگرام هم مقدار FWHM را بدست آورده ایم.

سوال 11:

مطلوب سوال:

سوال از ما خواسته است که با 4 منطق مختلف بین بندی 4 هیستوگرام تولید کنیم و آن ها را با هم مقایسه کنیم. برای راحت تر شدن مقایسه هم بجای اینکه نمودار ها ستونی باشند از خط شکسته استفاده می کنیم و همه را در یک مختصات نمایش می دهیم.

کد سوال:

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
data = np.random.normal(loc=0, scale=1, size=5000)

bin_rules = {
    "Fixed (40 bins)": 40,
    "FD Rule": 'fd',    # fridman
    "Sturges Rule": 'sturges',  # sturges
    "Scott Rule": 'scott'  # scott
}

# create historgam as a graph
plt.figure(figsize=(10, 6))

for rule_name, bins in bin_rules.items():
    counts, bin_edges = np.histogram(data, bins=bins, density=True)
    bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
    plt.plot(bin_centers, counts, marker='o', linestyle='-', label=rule_name)

plt.title("Histogram Comparison with Line Plot")
```

```python
plt.xlabel("Value")
plt.ylabel("Density")
plt.legend()
plt.grid(True)
plt.show()
```

توضیح کلی:

ابتدا یک داده تصادفی درست کردیم و سپس با 4 منطق مختلف 4 هیستوگرام درست کردیم. هیستوگرام ها بصورت نمودار خط شکسته هستند.

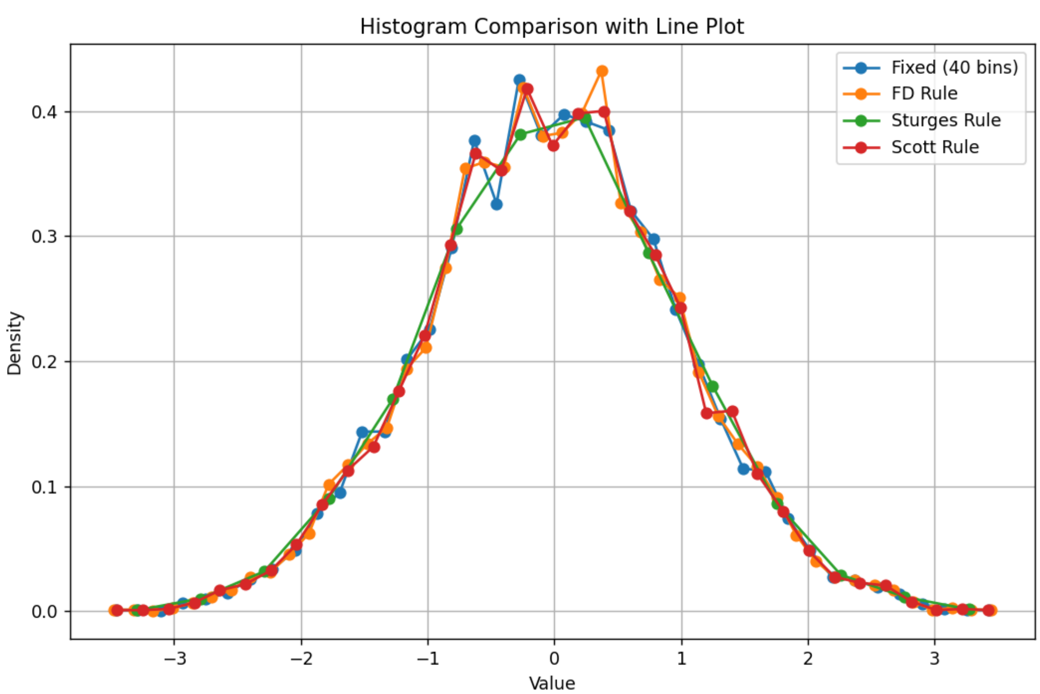
بخش های خاص کد:

```python
for rule_name, bins in bin_rules.items():

    counts, bin_edges = np.histogram(data, bins=bins, density=True)
    bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
    plt.plot(bin_centers, counts, marker='o', linestyle='-', label=rule_name)
```

در اینجا یک حلقه زدیم و همه ی نمودار ها را کشیدیم.

خروجی:



نتیجه:

روش Fixed (40 bins) دارای تعداد زیادی باین ثابت است و ممکن است جزئیات ریز را نشان دهد اما می‌تواند بیش از حد شلوغ باشد.

FD Rule (فریدمن-دیاکونیس) باین‌ها را بر اساس IQR (فاصله بین‌چارکی) تنظیم می‌کند، بنابراین برای داده‌هایی با توزیع‌های متفاوت انعطاف‌پذیرتر است.

Sturges Rule برای توزیع‌های نرمال و داده‌های کم‌حجم بهتر عمل می‌کند و تعداد باین‌های کمتری دارد.

Scott Rule برای کاهش خطای تخمین چگالی طراحی شده و معمولاً نتایج نرم‌تر و روان‌تری ارائه می‌دهد.

هر روش باین‌بندی منحنی متفاوتی از توزیع داده‌ها ارائه می‌دهد، اما همگی دارای قله در مرکز و دنباله‌هایی در دو طرف هستند که نشان‌دهنده توزیع نرمال است.

فصل 8:

سوال 6:

خواسته های سوال:

در این سوال عملیات نرمال سازی را برای cdf ها و pdf ها بررسی می کنیم.

کد سوال:

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

x = np.linspace(-6, 6, 1001)
dx = x[1] - x[0]

def gaussian(x, mean, std_dev):
    return (1 / (std_dev * np.sqrt(2 * np.pi))) * np.exp(-((x - mean) ** 2) / (2
* std_dev**2))

# create 2 pdfs
pdf1 = gaussian(x, mean=-2.7, std_dev=1)
pdf2 = gaussian(x, mean=2.7, std_dev=1)
```

```python
pdf = pdf1 + pdf2

# create cdf by pdf
cdf = np.cumsum(pdf) * dx

# normalized cdf and pdf
pdf_normalized = pdf / np.sum(pdf * dx)
cdf_normalized = np.cumsum(pdf_normalized) * dx

# plot them
plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)
plt.plot(x, pdf, label="PDF (Unnormalized)", color='blue')
plt.plot(x, pdf_normalized, label="PDF (Normalized)", color='green',
linestyle='dashed')
plt.title("PDF (Original and Normalized)")
plt.xlabel("x")
plt.ylabel("Density")
plt.legend()
plt.grid(True)

plt.subplot(3, 1, 2)
plt.plot(x, cdf, label="CDF (Unnormalized)", color='red')
plt.title("CDF (Unnormalized)")
plt.xlabel("x")
plt.ylabel("Cumulative Probability")
plt.legend()
plt.grid(True)

plt.subplot(3, 1, 3)
plt.plot(x, cdf_normalized, label="CDF (Normalized)", color='purple')
plt.title("CDF (Normalized)")
plt.xlabel("x")
plt.ylabel("Cumulative Probability")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

print("Unnormalized CDF:")
print(f"Final value of unnormalized CDF: {cdf[-1]:.4f}")
```

```
print("\nNormalized CDF:")
print(f"Final value of normalized CDF: {cdf_normalized[-1]:.4f}")
```

توضیح کلی:

ابتدا با استفاده از توضیح گاوسی دو تا pdf با میانگین های متفاوت ایجاد کرده ایم و سپس آن ها را جمع کرده ایم و به pdf جدیدی رسیدیم. حال از این pdf یک cdf تولید کردیم.

سپس هر دو را normalize کردیم و نمودار های جدید را با قبلی ها مقایسه کردیم.

بخش های خاص کد:

```
def gaussian(x, mean, std_dev):
    return (1 / (std_dev * np.sqrt(2 * np.pi))) * np.exp(-((x - mean) ** 2) / (2
* std_dev**2))
```

در اینجا توضیح گاوسی را تعریف کرده ایم. آرگومان های ورودی اش به ترتیب مقادیر، میانگین و انحراف معیار هستند.

در بخش اول نرمال سازی می کنیم که تضمین شود که مجموع احتمالات ما 1 است. در بهش بعدی هم مجموع توان 2 های اختلافات از میانگین را به توان 2 می رسانیم و بر 2 برابر واریانس تقسیم میکنیم. اینگونه همان مقدار نمایی ایجاد می شود.

```
cdf = np.cumsum(pdf) * dx
```

در این بخش cdf را از روی pdf بدست آوردیم.cumsum همان کار سیگما را می کند.

```
pdf_normalized = pdf / np.sum(pdf * dx)
```

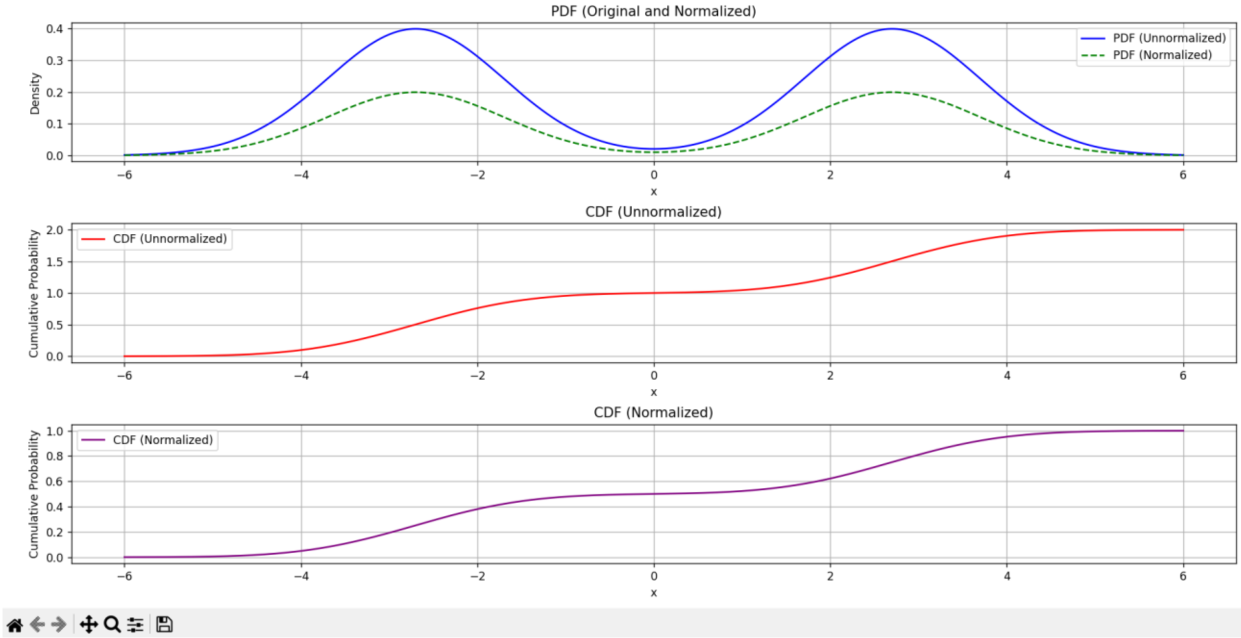
برای نرمال سازی pdf را بر مجموع کل انتگرال ها تقسیم کرده ایم.

و در ادامه نیز تمام نمودار هایمان را plot کردیم.

خروجی:

```
Unnormalized CDF:
Final value of unnormalized CDF: 1.9991

Normalized CDF:
Final value of normalized CDF: 1.0000
PS D:\University\basic\Amar\homeworks\HW3\codes>
```



نتیجه گیری:

وقتی نرمال سازی می کنیم شکل نمودار عوض می شود. (چه pdf باشد و چه cdf) . در بین نمودار ها نمودار نرمال سازی شده درست است. زیرا در نمودارهای نرمال سازی شده است که جمع چگالی احتمالات 1 می شود.

همچنین یک نتیجه ی دیگر که میتوانیم بگیریم این است که عملیات های ریاضی مثل جمع pdf های نرمال را غیر نرمال می کنندو هر گاه از ایم عملیات ها استفاده می کنیم باید بعدش نمودار را نرمال سازی کنیم.

سوال 7:

خواسته های سوال:

خواسته این سوال این است که روابط بین pdf و cdf را بررسی کنیم. یعنی اینکه آیا می توان pdf را از روی cdf کشید یا نه.

برای این کار ما ابتدا یک cdf با استفاده از توضیح lognormal رسم می کنیم. سپس دو تا pdf ایجاد می کنیم. یکی با استفاده از همان توضیح lognormal و دیگری با گرفتن مشتق گسسته از cdf. سپس این دو نمودار را با هم بررسی می کنیم که ببینیم تا چه حد با هم منطبق هستند.

کدی که برای سوال زده ایم:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import lognorm

x = np.linspace(0, 10, 200)
shape = 0.5
scale = np.exp(1)

# create cdf
cdf = lognorm.cdf(x, shape, scale=scale)

# 1:create pdf by lognorm
pdf_direct = lognorm.pdf(x, shape, scale=scale)

# 2: create pdf by cdf
dx = x[1] - x[0]
pdf_derivative = np.gradient(cdf, dx)

pdf_derivative_normalized = pdf_derivative / np.trapz(pdf_derivative, x)

# plot cdf
plt.figure(figsize=(12, 6))
```

```python
plt.subplot(2, 1, 1)
plt.plot(x, cdf, label="CDF", color="blue")
plt.title("CDF of Lognormal Distribution")
plt.xlabel("x")
plt.ylabel("Cumulative Probability")
plt.grid(True)
plt.legend()

# plot pdf
plt.subplot(2, 1, 2)
plt.plot(x, pdf_direct, label="PDF (Direct from lognorm.pdf)", color="green")
plt.plot(x, pdf_derivative_normalized, label="PDF (Derived from CDF)",
color="red", linestyle="dashed")
plt.title("PDFs of Lognormal Distribution")
plt.xlabel("x")
plt.ylabel("Probability Density")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

print(pdf_direct == pdf_derivative_normalized)
```

بخش های خاص کد:

```python
cdf = lognorm.cdf(x, shape, scale=scale)
```

این تابع cdf را رسم می کند. x همان مقادیرش است. shape مقدار انحراف معیار است. scale هم همان مکان است که در اینجا توضیح ما حول 2.7 است.

```python
pdf_direct = lognorm.pdf(x, shape, scale=scale)
```

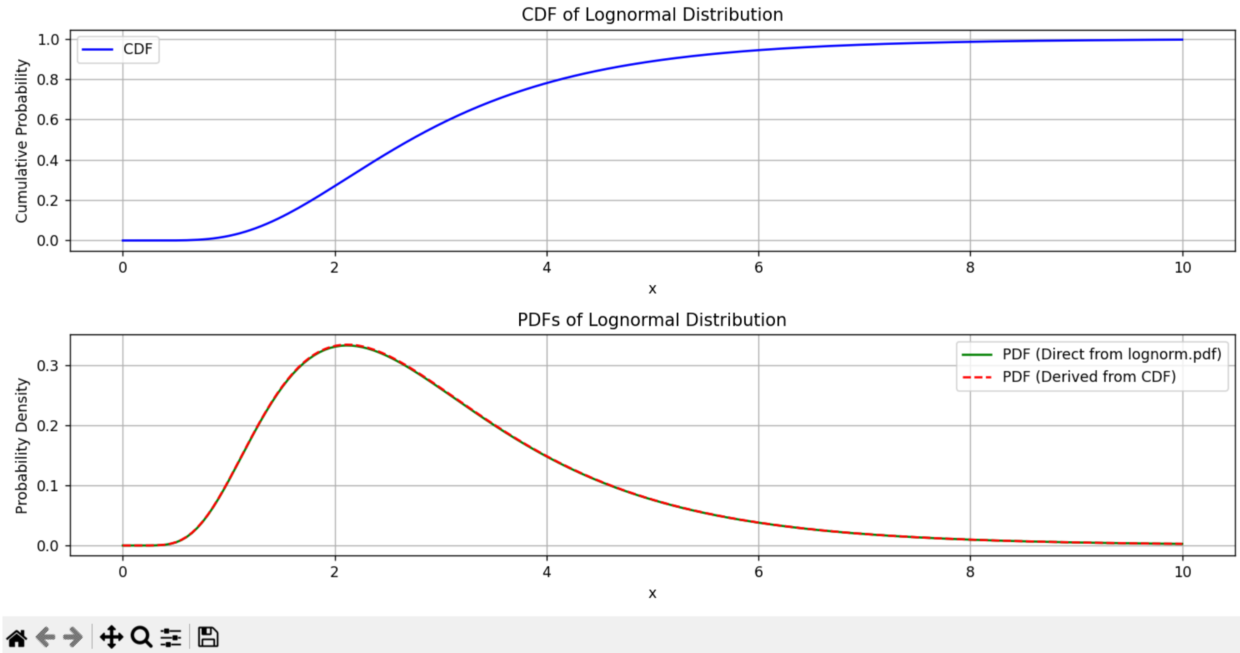
این تابع pdf را رسم می کند. آرگومان هایش مثل تابع قبلی است.

```python
dx = x[1] - x[0]
pdf_derivative = np.gradient(cdf, dx)
pdf_derivative_normalized = pdf_derivative / np.trapz(pdf_derivative, x)
```

در اینجا pdf را با استفاده از cdf می کشیم. ابتدا از cdf مشتق گسسته می گیریم. در آخر نیز برای اینکه مطمئن شویم مجموع احتمالات 1 میشود تابع را نرمال سازی کردیم.

در ادامه ی کد هم نمودار های را کشیده ایم.

خروجی:



نتیجه گیری:

همانطور که میبینیم دو pdf که به روش های مختلف کشیدیم تا حد زیادی به روی هم منطبق هستند. پس در کل ما همانطور که می توانیم cdf را از روی pdf بکشیم می توانیم pdf را نیز از روی cdf بکشیم.