1 The fun with python continues

1.1

Equations:

- Weighted mean:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i \cdot w_i}{\sum_{i=1}^{n} w_i}$$

- Weighted standard deviation:

$$\sigma_w = \sqrt{\frac{\sum_{i=1}^{n} w_i \cdot (x_i - \bar{x})^2}{\sum_{i=1}^{n} w_i}}$$

1.2 og 1.3

```python
def check_weights_for_non_negativity(weights):
    """
    Checks if all elements in the weights array are non-negative.

    Parameters:
    - weights: An array containing weights.

    Returns:
    - bool: True if all weights are non-negative, False otherwise.
    """
    return all(weight >= 0 for weight in weights)

def normalize_weights(weights):
    """
    Normalizes the weights array such that the sum of weights equals 1.

    Parameters:
    - weights: An array containing weights.

    Returns:
    - list: A list of normalized weights.
    """
    if not check_weights_for_non_negativity(weights):
        raise ValueError("Weights must be non-negative.")
    total_weight = sum(weights)
    normalized_weights = [weight / total_weight for weight in weights]
    return normalized_weights

def compute_weighted_mean(data, weights=None):
    """
    Parameters:
    - data: An array containing data points.
    - weights: An array containing weights.
        If None, equal weights are assigned.

    Returns:
    - float: Weighted mean of the data.
    """
    if weights is None:
        weights = np.ones(len(data))
    else:
        if len(data) != len(weights):
            raise ValueError("Length of weights must be equal to length of data.")
        weights = normalize_weights(weights)
```

```python
    weighted_mean = np.sum(data * weights)
    return weighted_mean

def compute_weighted_std(data, weights=None, mean=None):
    """
    Parameters:
    - data: An array.
    - weights: An array containing weights.
        If None, equal weights are assigned.
    - mean: Mean of the data.

    Returns:
    - float: Weighted standard deviation of the data.
    """
    if weights is None:
        weights = np.ones(len(data))
    else:
        if len(data) != len(weights):
            raise ValueError("Length of weights must be equal to length of data.")
        weights = normalize_weights(weights)
    if mean is None:
        mean = compute_weighted_mean(data, weights)
    else:
        if len(data) != len(mean):
            raise ValueError("Length of mean must be equal to length of data.")
    weighted_variance = np.sum(weights * (data - mean)**2)
    weighted_std = np.sqrt(weighted_variance)
    return weighted_std

x = np.random.randn(100)
weights = np.random.exponential(1, 100)

numpy_weighted_mean = np.average(x, weights=weights)
numpy_weighted_std = np.sqrt(np.average((x - numpy_weighted_mean)**2, weights=weights))

print("Custom Weighted Mean:", compute_weighted_mean(x, weights))
print("NumPy Weighted Mean:", numpy_weighted_mean)
print("Custom Weighted Std:", compute_weighted_std(x, weights))
print("NumPy Weighted Std:", numpy_weighted_std)
```

```
Custom Weighted Mean: 0.005820889620768834
NumPy Weighted Mean: 0.005820889620768804
Custom Weighted Std: 1.026104398335031
NumPy Weighted Std: 1.026104398335031
```

2 The uniform Distribution

2.1

$$\int_{-\infty}^{\infty} pdf(x)\,dx = \int_a^b \frac{1}{b-a}\,dx = \frac{x}{b-a}\Big|_a^b = \frac{b-a}{b-a} = 1$$

2.2

$$Var(x) = E((x-\mu)^2) = E(x^2) - [\mu]^2$$

To find $E(x^2)$ we integrate $x^2 \cdot pdf(x)$:

$$E(x^2) = \int_a^b x^2 \cdot \frac{1}{b-a}\,dx = \frac{1}{b-a} \cdot \frac{x^3}{3}\Big|_a^b = \frac{1}{b-a} \cdot \frac{b^3-a^3}{3}$$

$$E(x^2) = \frac{b^3-a^3}{3(b-a)}$$

$$Var(X) = \frac{b^3 - a^3}{3(b-a)} - \left(\frac{b+a}{2}\right)^2$$

2.4

$$\text{Skew}(x) = \frac{E((x-\mu)^3)}{\sigma^3}$$

Have already computed $\mu = \frac{a+b}{2}$ and

$\sigma^2 = \frac{(b-a)^2}{12}$

We need to compute $E((x-\mu)^3)$. Notice that the integrand $(x-\mu)^3$ is an odd function. Since the uniform distribution is symmetric about its mean μ, the integral of any odd function over the symmetric interval $[a,b]$ will be zero.

$$E((x-\mu)^3) = 0$$

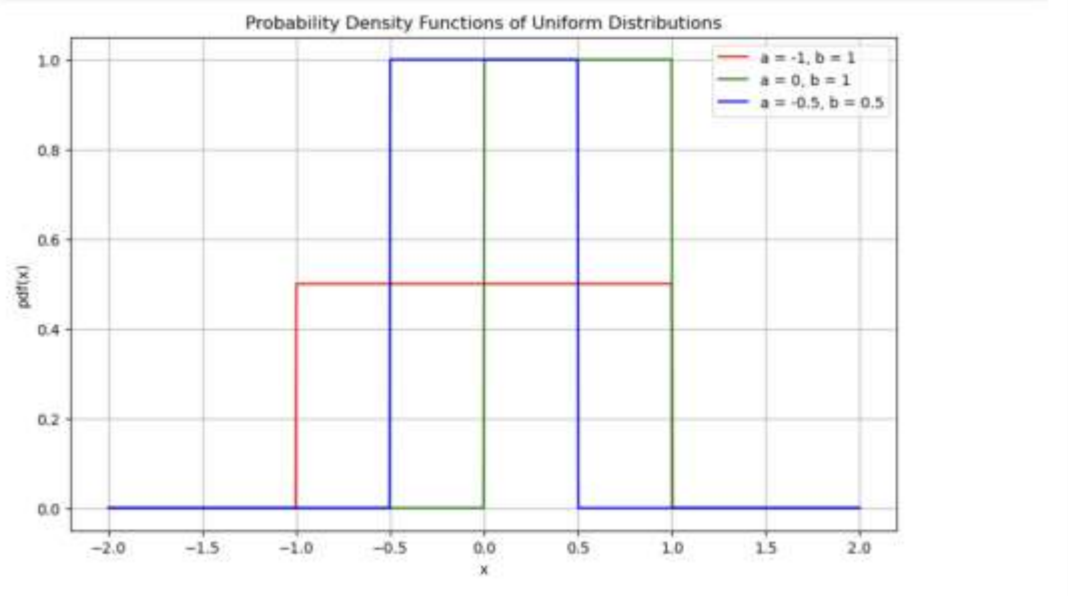
```
import matplotlib.pyplot as plt

parameters = [(-1, 1), (0, 1), (-0.5, 0.5)]
colors = ['r', 'g', 'b']
labels = ['a = -1, b = 1', 'a = 0, b = 1', 'a = -0.5, b = 0.5']

x = np.linspace(-2, 2, 1000)

plt.figure(figsize=(10, 6))
for i, (a, b) in enumerate(parameters):
    pdf = np.where((x >= a) & (x <= b), 1/(b-a), 0)
    plt.plot(x, pdf, color=colors[i], label=labels[i])

plt.title('Probability Density Functions of Uniform Distributions')
plt.xlabel('x')
plt.ylabel('pdf(x)')
plt.legend()
plt.grid(True)
plt.show()
```



4.1

$$E(x) = \int_0^\infty \exp(-\lambda x)\, dx$$

$$= \lim_{b\to\infty}\left(-\frac{1}{\lambda}\exp(-\lambda b)+\frac{1}{\lambda}\right)$$

$$= -\frac{1}{\lambda}(0) + \frac{1}{\lambda}$$

$$= \underline{\underline{\frac{1}{\lambda}}}$$

4.2

$$E(x^2) = \int_0^\infty x^2\cdot\exp(-\lambda x)\,dx$$

$$= \left[-\frac{1}{\lambda}x^2\exp(-\lambda x)\right]_0^\infty - \int_0^\infty 2x\left(-\frac{1}{\lambda}x\cdot\exp(-\lambda x)\right) dx$$

$$= \left[-\frac{1}{\lambda}x^2\cdot\exp(-\lambda x)\right]_0^\infty - \left[\frac{1}{\lambda^2}2x\cdot\exp(-\lambda x)\right]_0^\infty - \int_0 2\left(\frac{1}{\lambda^2}\cdot\exp(-\lambda x)\right)dx$$

$$= \left[-\frac{x^2}{\lambda}\cdot\exp(-\lambda x)\right]_0^\infty - \left[2\frac{x}{\lambda^2}\cdot\exp(-\lambda x)\right]_0^\infty - \left[-2\frac{\lambda^3}{\lambda^3}\cdot\exp(-\lambda x)\right]_0^\infty$$

$$= \left[(-x^2/\lambda - 2x/\lambda^2 - 2\lambda^3)\exp(-\lambda x)\right]_0^\infty$$

$$= 0$$

$$Var(x) = E(x^2) - [E(x)]^2$$

$$= 0 - \left(\frac{1}{\lambda}\right)^2$$

$$= \frac{1}{\lambda^2}$$

```python
x = np.linspace(0,5,100)
lambdas = [1,2,10]

plt.figure(figsize=(7,5))
for lam in lambdas:
    pdf = lam * np.exp(-lam*x)
    plt.plot(x,pdf,label=f'lambda = {lam}')

plt.title('Probability Density Functions of Exponential Distributions')
plt.xlabel('x')
plt.ylabel('pdf(x)')
plt.legend()
plt.grid(True)
plt.show();
```