APPENDIX

APPENDIX A

APPLICATION OF PYTHON IN STATISTICAL ANALYSIS AND DISTRIBUTION MODELING

This chapter presents some Python code using in this thesis.

A.1 The Curves of the Posterior Probability Density Function

This section illustrates the visualization of the posterior probability density function (PDF) curves for different parameters, using Python code to generate plots for various distributions. The curves demonstrate how the posterior PDF varies with respect to different values of the parameters and help visualize the model's behavior for different data points.

A.1.1 The Curve of the Posterior PDF for the PWEU Distribution

This plot shows the curve of the posterior PDF for the PWEU model, where the value of θ is varied across a specific range. The graph is generated by plotting the value of $k(\lambda|\theta)$ for different values of λ and θ, allowing for a deeper understanding of the behavior of this distribution.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

lambda_value = np.linspace(0.001, 5, 200)
theta_value = np.linspace(np.pi/6, 2 * np.pi, 6)
len_theta = len(theta_value)

def g(lambda0, theta):
    numerator = 4 * np.pi ** 2 * lambda0 * np.exp(-lambda0 * theta)
    denominator = (1 - np.exp(-2 * np.pi * lambda0)) * ss.zeta(2, theta/(2 * np.pi))
    return numerator / denominator

for i in range(len_theta):
    y = g(lambda_value, theta_value[i])
    plt.plot(lambda_value, y, label=f'$\\theta$ = {theta_value[i]:.3f}')

plt.xlabel('$\\theta$')
plt.ylabel('$k(\lambda|\\theta)$')
#plt.title('PWEU$(\lambda|\\theta)$')
plt.legend()
plt.ylim(0,4)
plt.xlim(0,5)
plt.show()
```

A.1.2 The Curve of the Posterior PDF for the PWEG Distribution

The graph presents the posterior PDF curve for the PWEG model with given parameters, highlighting the relationship between λ and θ when $\alpha = 2$ and $\beta = 0.5$. By varying θ, the plot visually represents how the PWEG distribution behaves under different conditions, offering insights into the characteristics of this specific model.

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

lambda_value = np.linspace(0.001, 5, 200)
theta_value = np.linspace(np.pi/6, 2 * np.pi, 6)
len_theta = len(theta_value)

def gg(lambda0, theta, alpha, beta):
    numerator = ((2 * np.pi) ** (alpha + 1)) * (lambda0 ** alpha) * np.exp(-lambda0 * (
        theta + beta))
    denominator = ss.gamma(alpha + 1) * (1 - np.exp(-2 * np.pi * lambda0)) * ss.zeta(
        alpha + 1, (theta + beta) / (2 * np.pi))
    return numerator / denominator

for i in range(len_theta):
    y = gg(lambda_value, theta_value[i], 2, 0.5)
    plt.plot(lambda_value, y, label=f'$\\theta$ = {theta_value[i]:.3f}')

plt.xlabel('$\lambda$')
plt.ylabel('$k(\lambda|\\theta, 2, 0.5)$')
#plt.title('PWEG$(\lambda|\\theta, \\alpha=2, \\beta=0.5)$')
plt.ylim(0,2)
plt.xlim(0,5)
plt.legend()
plt.show()
```

A.2 Moment Generating Function, Statistical Measures

This section introduces several fundamental statistical measures for the PWEU and PWEG distributions, including the moment-generating function (MGF), mean, variance, skewness, and kurtosis. These measures are critical for understanding the distribution's properties and are essential for analyzing the behavior of the data modeled by these distributions.

A.2.1 The Statistical Measures for the PWEU Distribution

This section describes the calculation of the statistical measures for the PWEU distribution. It includes the computation of the moment-generating function (MGF), mean, variance, skewness, and kurtosis for this model. The corresponding Python code allows the user to calculate and visualize these statistics, helping to better understand the underlying distribution.

- MGF

```python
import numpy as np
import scipy.special as ss

def mgf(t, theta):
    numerator = ss.zeta(2, (theta - t) / (2 * np.pi))
    denominator = ss.zeta(2, theta / (2 * np.pi))
    return numerator / denominator
```

- Mean

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)

def zeta(k, theta):
    return ss.gamma(2 + k) * ss.zeta(2 + k, theta / (2 * np.pi)) / np.power(2 *
        np.pi, k)
def mu(theta):
    numerator = zeta(1, theta)
    denominator = zeta(0, theta)
    return numerator / denominator

plt.plot(theta_value2, mu(theta_value2))
plt.xlabel('$\\theta$')
plt.ylabel('$\mu(\\theta)$')
#plt.title('Graph of $\mu(\\theta)$')
plt.ylim(0, 80)
plt.xlim(0, 2 * np.pi)
plt.show()
```

- Variance

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)


def zeta(k, theta):
    return ss.gamma(2 + k) * ss.zeta(2 + k, theta / (2 * np.pi)) / np.power(2 *
        np.pi, k)
def mu(theta):
    numerator = zeta(1, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def raw_2m(theta):
    numerator = zeta(2, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def sigma_s(theta):
    return raw_2m(theta) - np.power(mu(theta), 2)

plt.plot(theta_value2, sigma_s(theta_value2))
plt.xlabel('$\\theta$')
plt.ylabel('$\sigma^{2}(\\theta)$')
#plt.title('Graph of $\sigma^{2}(\\theta)$')
plt.ylim(0, 80)
plt.xlim(0, 2 * np.pi)
plt.show()
```

- Skewness

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)

def zeta(k, theta):
    return ss.gamma(2 + k) * ss.zeta(2 + k, theta / (2 * np.pi)) / np.power(2 *
        np.pi, k)
def mu(theta):
    numerator = zeta(1, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def raw_2m(theta):
    numerator = zeta(2, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def sigma_s(theta):
    return raw_2m(theta) - np.power(mu(theta), 2)
def raw_3m(theta):
    numerator = zeta(3, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def skew(theta):
    numerator = raw_3m(theta) - 3 * mu(theta) * sigma_s(theta) - np.power(mu(
        theta), 3)
    denominator = np.power(np.sqrt(sigma_s(theta)), 3)
    return numerator / denominator

plt.plot(theta_value2, skew(theta_value2))
plt.xlabel('$\\theta$')
plt.ylabel('$\gamma(\\theta)$')
#plt.title('Graph of $\gamma_{1}(\\theta)$')
plt.ylim(0, 2)
plt.xlim(0, 2 * np.pi)
plt.show()
```

- Kurtosis

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)

def zeta(k, theta):
    return ss.gamma(2 + k) * ss.zeta(2 + k, theta / (2 * np.pi)) / np.power(2 *
        np.pi, k)
def mu(theta):
    numerator = zeta(1, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def raw_2m(theta):
    numerator = zeta(2, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def sigma_s(theta):
    return raw_2m(theta) - np.power(mu(theta), 2)
def raw_3m(theta):
    numerator = zeta(3, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def raw_4m(theta):
    numerator = zeta(4, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def Kurt(theta):
    numerator = raw_4m(theta) - (4 * mu(theta) * raw_3m(theta)) + (6 * np.power(
        mu(theta), 2) * sigma_s(theta)) + (3 * np.power(mu(theta), 4))
    denominator = np.power(sigma_s(theta), 2)
    return numerator / denominator

plt.plot(theta_value2, Kurt(theta_value2))
plt.xlabel('$\\theta$')
plt.ylabel('$\kappa(\\theta)$')
#plt.title('Graph of $\\beta_{2}(\\theta)$')
plt.ylim(0, 10)
plt.xlim(0, 2 * np.pi)
plt.show()
```

A.2.2 The Statistical Measures for the PWEG Distribution

In this subsection, the statistical measures for the PWEG distribution are computed and visualized. The code provided allows for the calculation of the MGF, mean, variance, skewness, and kurtosis, with visualizations to help interpret the effects of varying parameters such as α and β. These measures offer critical insights into the shape and characteristics of the distribution.

- MGF

```
import numpy as np
import scipy.special as ss

def mgf_g(t, theta, alpha, beta):
    numerator = ss.zeta(alpha + 1, (theta + beta - t) / (2 * np.pi))
    denominator = ss.zeta(alpha + 1, (theta + beta) / (2 * np.pi))
    return numerator / denominator
```

- Mean

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)
alpha_value = np.linspace(1, 9, 9)
len_alpha = len(alpha_value)

def zeta_g(k, theta, alpha, beta):
    return ss.gamma(alpha + 1 + k) * ss.zeta(alpha + 1 + k, (theta + beta) / (2
        * np.pi)) / np.power(2 * np.pi, k)
def mu_g(theta, alpha, beta):
    numerator = zeta_g(1, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator

for i in range(len_alpha):
    y = mu_g(theta_value2, alpha_value[i], 0.5)
    plt.plot(theta_value2, y, label=f'$\\alpha$ = {alpha_value[i]:.0f}')

plt.xlabel('$\\theta$')
plt.ylabel('$\mu(\\theta|\\alpha, 0.5)$')
#plt.title('Graph of $\mu(\\theta|\\alpha, \\beta = 0.5)$')
plt.ylim(0, 20)
plt.xlim(0, 2 * np.pi)
plt.legend()
plt.show()
```

- Variance

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

```python
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)
alpha_value = np.linspace(1, 9, 9)
len_alpha = len(alpha_value)

def zeta_g(k, theta, alpha, beta):
    return ss.gamma(alpha + 1 + k) * ss.zeta(alpha + 1 + k, (theta + beta) / (2
        * np.pi)) / np.power(2 * np.pi, k)
def raw_2m_g(theta, alpha, beta):
    numerator = zeta_g(2, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def mu_g(theta, alpha, beta):
    numerator = zeta_g(1, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def sigma_s_g(theta, alpha, beta):
    return raw_2m_g(theta, alpha, beta) - np.power(mu_g(theta, alpha, beta), 2)

for i in range(len_alpha):
    y = sigma_s_g(theta_value2, alpha_value[i], 0.5)
    plt.plot(theta_value2, y, label=f'$\\alpha$ = {alpha_value[i]:.0f}')

plt.xlabel('$\\theta$')
plt.ylabel('$\sigma^{2}(\\theta| \\alpha, 0.5)$')
#plt.title('Graph of $\sigma^{2}(\\theta| \\alpha, \\beta = 0.5)$')
plt.ylim(0, 40)
plt.xlim(0, 2 * np.pi)
plt.legend()
plt.show()
```

- Skewness

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)
alpha_value = np.linspace(1, 9, 9)
len_alpha = len(alpha_value)

def zeta_g(k, theta, alpha, beta):
    return ss.gamma(alpha + 1 + k) * ss.zeta(alpha + 1 + k, (theta + beta) / (2
        * np.pi)) / np.power(2 * np.pi, k)
def raw_2m_g(theta, alpha, beta):
    numerator = zeta_g(2, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def mu_g(theta, alpha, beta):
    numerator = zeta_g(1, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def sigma_s_g(theta, alpha, beta):
    return raw_2m_g(theta, alpha, beta) - np.power(mu_g(theta, alpha, beta), 2)
def raw_3m_g(theta, alpha, beta):
    numerator = zeta_g(3, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def skew_g(theta, alpha, beta):
    numerator = raw_3m_g(theta, alpha, beta) - (3 * mu_g(theta, alpha, beta) *
        sigma_s_g(theta, alpha, beta)) - np.power(mu_g(theta, alpha, beta), 3)
    denominator = np.power(np.sqrt(sigma_s_g(theta, alpha, beta)), 3)
    return numerator / denominator

for i in range(len_alpha):
    y = skew_g(theta_value2, alpha_value[i], 0.5)
    plt.plot(theta_value2, y, label=f'$\\alpha$ = {alpha_value[i]:.0f}')

plt.xlabel('$\\theta$')
plt.ylabel('$\gamma(\\theta | \\alpha, 0.5)$')
#plt.title('Graph of $\gamma{1}(\\theta, \\alpha, \\beta = 0.5)$')
```

```
plt.ylim()
plt.xlim(0, 2 * np.pi)
plt.legend()
plt.show()
```

- Kurtosis

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)

def zeta_g(k, theta, alpha, beta):
    return ss.gamma(alpha + 1 + k) * ss.zeta(alpha + 1 + k, (theta + beta) / (2
        * np.pi)) / np.power(2 * np.pi, k)
def raw_2m_g(theta, alpha, beta):
    numerator = zeta_g(2, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def mu_g(theta, alpha, beta):
    numerator = zeta_g(1, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def sigma_s_g(theta, alpha, beta):
    return raw_2m_g(theta, alpha, beta) - np.power(mu_g(theta, alpha, beta), 2)
def raw_3m_g(theta, alpha, beta):
    numerator = zeta_g(3, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def raw_4m_g(theta, alpha, beta):
    numerator = zeta_g(4, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def Kurt_g(theta, alpha, beta):
    numerator = raw_4m_g(theta, alpha, beta) - (4 * mu_g(theta, alpha, beta) *
        raw_3m_g(theta, alpha, beta)) + (6 * np.power(mu_g(theta, alpha, beta),
        2) * sigma_s_g(theta, alpha, beta)) + (3 * np.power(mu_g(theta, alpha,
        beta), 4))
    denominator = np.power(sigma_s_g(theta, alpha, beta), 2)
    return numerator / denominator
def Kurt_ex_g(theta, alpha, beta):
    return Kurt_g(theta, alpha, beta) - 3

for i in range(len_alpha):
    y = Kurt_ex_g(theta_value2, alpha_value[i], 0.5)
    plt.plot(theta_value2, y)

plt.xlabel('$\\theta$')
plt.ylabel('$\gamma_{2}(\\theta, \\alpha, \\beta = 0.5)$')
plt.title('Graph of $\gamma_{2}(\\theta, \\alpha, \\beta = 0.5)$')
plt.ylim()
plt.xlim(0, 2 * np.pi)
plt.show
```

A.3 Bayesian Estimators and Risk Functions

This section delves into the concept of Bayesian estimators and their associated risk functions. Using the PWEU distribution, the Bayesian Estimators (BE) and Risk Minimization Functions (RMF) are derived and visualized. The plots generated here help to understand the efficiency and effectiveness of the Bayesian approach in estimating model parameters under different loss functions.

A.3.1 Bayesian Estimators for the PWEU Distribution

This part explains the process of deriving and visualizing Bayesian Estimators for the PWEU model using a squared error loss function (SEL). The corresponding Python code demonstrates how to compute the Bayesian estimator and visualize the results, helping to assess the performance of the estimator under varying values of θ.

- Squared Error Loss Function

  - Bayesian Estimator (BE)

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)

def zeta(k, theta):
    return ss.gamma(2 + k) * ss.zeta(2 + k, theta / (2 * np.pi)) / np.power
        (2 * np.pi, k)
def mu(theta):
    numerator = zeta(1, theta)
    denominator = zeta(0, theta)
    return numerator / denominator

plt.plot(theta_value2, mu(theta_value2))
plt.xlabel('$\\theta$')
plt.ylabel('$\hat\lambda_{SEL}$')
#plt.title('Graph of $\hat{\lambda}(\\theta)$')
plt.ylim(0, 80)
plt.xlim(0, 2 * np.pi)
plt.show()
```

  - Risk Minimization Function (RMF)

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)

```

```

def zeta(k, theta):
    return ss.gamma(2 + k) * ss.zeta(2 + k, theta / (2 * np.pi)) / np.power
        (2 * np.pi, k)
def mu(theta):
    numerator = zeta(1, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def raw_2m(theta):
    numerator = zeta(2, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def sigma_s(theta):
    return raw_2m(theta) - np.power(mu(theta), 2)

plt.plot(theta_value2, sigma_s(theta_value2))
#plt.title('Graph of $R(\lambda,\hat{\lambda})$')
plt.xlabel('$\\theta$')
plt.ylabel('$R(\lambda,\hat{\lambda}_{SEL})$')
plt.ylim(0, 80)
plt.xlim(0, 2 * np.pi)
plt.show()
```

- Precautionary Loss Function

  - Bayesian Estimator (BE)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)

def zeta(k, theta):
    return ss.gamma(2 + k) * ss.zeta(2 + k, theta / (2 * np.pi)) / np.power
        (2 * np.pi, k)
def raw_2m(theta):
    numerator = zeta(2, theta)
    denominator = zeta(0, theta)
    return numerator / denominator

plt.plot(theta_value2, np.power(raw_2m(theta_value2),0.5))
plt.xlabel('$\\theta$')
plt.ylabel('$\hat\lambda_{PL}$')
plt.ylim(0, 80)
plt.xlim(0, 2 * np.pi)
plt.show()
```

– Risk Minimization Function (RMF)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)

def zeta(k, theta):
    return ss.gamma(2 + k) * ss.zeta(2 + k, theta / (2 * np.pi)) / np.power
        (2 * np.pi, k)
def mu(theta):
    numerator = zeta(1, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def raw_2m(theta):
    numerator = zeta(2, theta)
    denominator = zeta(0, theta)
    return numerator / denominator
def sigma_s(theta):
    return 2 * (np.power(raw_2m(theta),3/2) - mu(theta))
```

```

plt.plot(theta_value2, sigma_s(theta_value2))
plt.xlabel('$\\theta$')
plt.ylabel('$R(\lambda,\hat\lambda_{PL})$')
plt.ylim(0, 80)
plt.xlim(0, 2 * np.pi)
plt.show()
```

A.3.2 Bayesian Estimators for the PWEG Distribution

In this section, Bayesian Estimators for the PWEG distribution are computed and visualized, using a squared error loss function (SEL). The provided Python code demonstrates the calculation of the Bayesian Estimator and shows the resulting plots. These visualizations help in understanding how the estimator behaves as the parameters α and β change and the model's sensitivity to different loss functions.

- Squared Error Loss Function

  - Bayesian Estimator (BE)

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)
alpha_value = np.linspace(1, 9, 9)
len_alpha = len(alpha_value)

def zeta_g(k, theta, alpha, beta):
    return ss.gamma(alpha + 1 + k) * ss.zeta(alpha + 1 + k, (theta + beta)
        / (2 * np.pi)) / np.power(2 * np.pi, k)
def mu_g(theta, alpha, beta):
    numerator = zeta_g(1, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator

for i in range(len_alpha):
    y = mu_g(theta_value2, alpha_value[i], 0.5)
    plt.plot(theta_value2, y, label=f'$\\alpha$ = {alpha_value[i]:.0f}')

plt.xlabel('$\\theta$')
plt.ylabel('$\hat\lambda_{SEL}(\\theta | \\alpha, 0.5)$')
plt.ylim(0, 20)
plt.xlim(0, 2 * np.pi)
plt.legend()
plt.show()
```

- Risk Minimization Function (RMF)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)
alpha_value = np.linspace(1, 9, 9)
len_alpha = len(alpha_value)

def zeta_g(k, theta, alpha, beta):
    return ss.gamma(alpha + 1 + k) * ss.zeta(alpha + 1 + k, (theta + beta)
        / (2 * np.pi)) / np.power(2 * np.pi, k)
def raw_2m_g(theta, alpha, beta):
    numerator = zeta_g(2, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def mu_g(theta, alpha, beta):
    numerator = zeta_g(1, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
```

```
def sigma_s_g(theta, alpha, beta):
    return raw_2m_g(theta, alpha, beta) - np.power(mu_g(theta, alpha, beta)
        , 2)

for i in range(len_alpha):
    y = sigma_s_g(theta_value2, alpha_value[i], 0.5)
    plt.plot(theta_value2, y, label=f'$\\alpha$ = {alpha_value[i]:.0f}')

plt.xlabel('$\\theta$')
plt.ylabel('$R(\lambda,\hat\lambda_{SEL} | \\alpha, 0.5)$')
plt.ylim(0, 40)
plt.xlim(0, 2 * np.pi)
plt.legend()
plt.show()
```

- Precautionary Loss Function

  – Bayesian Estimator (BE)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)
alpha_value = np.linspace(1, 9, 9)
len_alpha = len(alpha_value)

def zeta_g(k, theta, alpha, beta):
    return ss.gamma(alpha + 1 + k) * ss.zeta(alpha + 1 + k, (theta + beta)
        / (2 * np.pi)) / np.power(2 * np.pi, k)
def raw_2m_g(theta, alpha, beta):
    numerator = zeta_g(2, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
```

```

for i in range(len_alpha):
    y = raw_2m_g(theta_value2, alpha_value[i], 0.5)
    plt.plot(theta_value2, y, label=f'$\\alpha$ = {alpha_value[i]:.0f}')

plt.xlabel('$\\theta$')
plt.ylabel('$\hat\lambda_{PL}(\\theta | \\alpha, 0.5)$')
plt.ylim(0, 20)
plt.xlim(0, 2 * np.pi)
plt.legend()
plt.show()
```

– Risk Minimization Function (RMF)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as ss

theta_value2 = np.linspace(0.001, 2 * np.pi, 360)
alpha_value = np.linspace(1, 9, 9)
len_alpha = len(alpha_value)

def zeta_g(k, theta, alpha, beta):
    return ss.gamma(alpha + 1 + k) * ss.zeta(alpha + 1 + k, (theta + beta)
        / (2 * np.pi)) / np.power(2 * np.pi, k)
def raw_2m_g(theta, alpha, beta):
    numerator = zeta_g(2, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def mu_g(theta, alpha, beta):
    numerator = zeta_g(1, theta, alpha, beta)
    denominator = zeta_g(0, theta, alpha, beta)
    return numerator / denominator
def sigma_s_g(theta, alpha, beta):
    return 2*(np.power(raw_2m_g(theta, alpha, beta),3/2) - mu_g(theta,
        alpha, beta))

for i in range(len_alpha):
    y = sigma_s_g(theta_value2, alpha_value[i], 0.5)
    plt.plot(theta_value2, y, label=f'$\\alpha$ = {alpha_value[i]:.0f}')

plt.xlabel('$\\theta$')
plt.ylabel('$R(\lambda,\hat\lambda_{PL} | 2, \\beta)$')
plt.ylim(0, 40)
plt.xlim(0, 2 * np.pi)
plt.legend()
plt.show()
```