SSE Composite Index Data Analysis

Introduction

The performance of the Shanghai Stock Exchange (SSE), one of the most important financial organizations in the world, can be used as a crucial indicator and offer insightful information about the state of China's economy. In our project, we are going to leverage statistical tools and the knowledge of programming to assess the financial risk by calculating the VaR (value at risk) and ES (expected shortfall) using python through the following steps led by these three interesting questions raised from this context.

Methodology

1. **Setting up distributions**

   First, we picked up the last 1000 records from the document, which are the most recent closing data, we need to compute the log returns, which can precisely approximate percentage changes over time and are typically more preferred in financial analysis (Saturn Cloud, n.d.), using the following formula:

   $$\text{LogarithmicReturn} = \ln(\text{PresentValue}/\text{PastValue})$$

   Realized through codes below:

```python
def drawDataset(data: pd.DataFrame):
    """
    Compute and visualize the prior distribution of the log returns
    """
    fig, ax = plt.subplots()

    # Create a histogram of the data
    n, bins, patches = ax.hist(data["log return"], 50, density=True, facecolor='C0', alpha=0.75)
    ax.set_ylabel('Probability mass/density')
    ax.set_xlabel('log return')

    # fit the data with a normal distribution
    mu, std = norm.fit(data["log return"])

    # Generate x values from min to max of your data for plotting the PDF
    xmin, xmax = plt.xlim()  # Get current x-axis limits
    x = np.linspace(xmin, xmax, 100)  # Generate evenly spaced values

    # Calculate the PDF using the fitted parameters
    p = norm.pdf(x, mu, std)

    # Plot the PDF on top of the histogram
    plt.plot(x, p, 'k', linewidth=2)  # 'k' is for black color line
    ax.text(-0.08, 50, f"mu={mu:.6f}, sigma={std:.6f}")

    # Display the plot
    plt.savefig("data/dataset.png")
```

   Its visualization lies below, which is approximately a normal distribution, and it can be demonstrated by the law of the large numbers:

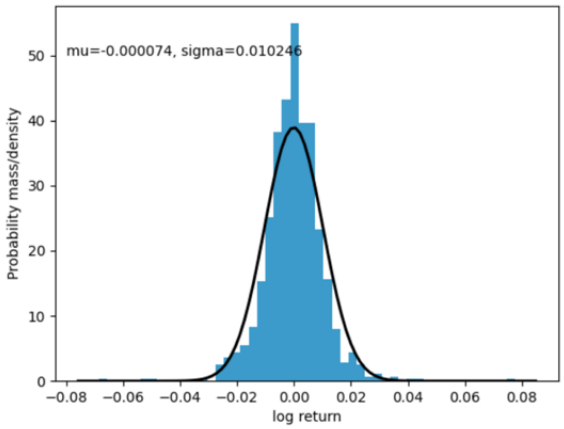


Figure 4: Original dataset of log returns collected

Using object-oriented programming, we can find the parameters of this normal distribution, which are μ_prior and τ_prior and let them fit into normal distribution and gamma distribution respectively:

```
class gamma_sampler(sampler):
    def __init__(self, alpha, beta):
        self.alpha = alpha
        self.beta = beta

    def sample(self):
        return gamma.rvs(self.alpha, self.beta, size=1)[0]

    def reset(self, alpha, beta):
        self.alpha = alpha
        self.beta = beta

class normal_sampler(sampler):
    def __init__(self, mean, std_dev):
        self.mean = mean
        self.std_dev = std_dev

    def sample(self):
        return norm.rvs(loc=self.mean, scale=self.std_dev, size=1)[0]

    def reset(self, mean, std_dev):
        self.mean = mean
        self.std_dev = std_dev
```

Here by using object-oriented programming, we can have clearer classes, to protect the data so that they can be easily invoked later as this is a long process of data analysis. Additionally, the computation complexity here is O(1) for initialization, sampling, and resetting for parameter updates.

Then, here comes the question: **How to find the right parameter that can better fit this graph?** By translating, that is with a vague prior:

Normal Distribution: μ_prior ~(0,10000)

Gamma Distribution: τ_prior(the inverse variance) ~(1,1000)

With the help of Bayesian inference, how to find the posterior distribution?

To solve this question, we need to know that observing the data from the posterior can be extremely complicated, but using programming, we can turn this problem into

something iterative, which aligns with the method of Gibbs sampling ((Peng, 2022), an easier way to estimate the posterior distribution of parameters by iteratively sampling from the conditional distributions of each variable.

That is, we need to compute two full conditional distributions (on the left) and generate the two Markov Chains iteratively given the selected initial values (on the right):

$$
\begin{aligned}
p(\mu \mid \tau, \mathbf{y}) &\propto f(\mathbf{y} \mid \mu, \tau) p(\mu) p(\tau) \\
&\propto f(\mathbf{y} \mid \mu, \tau) p(\mu) \\
&\propto \exp\left(-\frac{\tau}{2} \sum (y_i - \mu)^2\right) \exp\left(-\frac{w}{2} \mu^2\right) \\
&= \exp\left(-\frac{(n\tau + w)}{2} \mu^2 - \left(\sum y_i\right) \tau \mu\right) \\
&= \mathcal{N}\left(\frac{\tau}{n\tau + w} \sum y_i, \frac{1}{n\tau + w}\right)
\end{aligned}
$$

$$
\begin{aligned}
p(\tau \mid \mu, y) &\propto f(\mathbf{y} \mid \mu, \tau) p(\mu) p(\tau) \\
&\propto f(\mathbf{y} \mid \mu, \tau) p(\tau) \\
&\propto \tau^{\frac{n}{2}} \exp\left(-\frac{\tau}{2} \sum (y_i - \mu)^2\right) \tau^{\alpha - 1} \exp(-\tau \beta) \\
&= \tau^{\alpha - 1 + \frac{n}{2}} \exp\left(-\tau \left[\beta + \frac{1}{2} \sum (y_i - \mu)^2\right]\right) \\
&= \text{Gamma}\left(\alpha + \frac{n}{2}, \beta + \frac{1}{2} \sum (y_i - \mu)^2\right)
\end{aligned}
$$

$$
\tau_{n+1} = \text{Gamma}\left(\alpha + \frac{n}{2}, \beta + \frac{1}{2} \sum (y_i - \mu_n)^2\right)
$$

$$
\mu_{n+1} = \mathcal{N}\left(\frac{\tau_n}{n\tau_n + w} \sum y_i, \frac{1}{n\tau_n + w}\right)
$$

For this iteration: The for loop runs n-1 iterations, and within each iteration, it is O(1) per sample. Over the length m of y, complexity per iteration is O(m). Since the loop runs for n-1 iterations, and each iteration involves O(m) computations, the total complexity is O(n * m), a polynomial complexity. The data structure here is lists, to store μ and τ.

Codes as follows:

```python
def gibbs_sampling(n: int, # sample size
                   gamma_sampler: gamma_sampler, normal_sampler: normal_sampler, # gamma and normal samplers req'd
                   y: np.ndarray, # additionally observed data
                   tau_prior_alpha: int, tau_prior_beta: int, mu_prior_mean: int, mu_prior_tau: int): # The priors req'd
    # y: the observed data
    # Intialize the values for gibbs sampler
    # Here we use the mean of the prior
    tau = tau_prior_alpha * tau_prior_beta
    mean = mu_prior_mean

    # Two arrays to store the sampled values
    tau_samples = np.zeros(n)
    mean_samples = np.zeros(n)
    tau_samples[0] = tau
    mean_samples[0] = mean

    for i in range(1, n):

        gamma_sampler.reset(tau_prior_alpha + len(y)/2, tau_prior_beta + np.sum((y - mean_samples[i-1])**2)/2)
        tau_given_mean_and_data = gamma_sampler.sample() # The first full conditional distribution req'd
        tau_samples[i] = tau_given_mean_and_data

        # TODO: Check the formula, currently only work for mu_prior_mean = 0
        normal_sampler.reset(tau_samples[i-1]*np.sum(y)/(len(y)*tau_samples[i-1] + mu_prior_tau), 1/(len(y)*tau_samples[i-1] + mu_prior_tau))
        mean_given_tau_and_data = normal_sampler.sample() # The second full conditional distribution req'd
        mean_samples[i] = mean_given_tau_and_data

    return mean_samples, tau_samples
```

The visualization of μ and τ from the Gibbs Sampler:

```python
# Set up the prior distribution and additional data and pass into gibbs sampler
y = data['log return'].to_numpy()
mean_samples, tau_samples = modeller.gibbs_sampling(1000, modeller.gamma_sampler(0,0), modeller.normal_sampler(0,0),
                                                    y, # Additional data
                                                    0.1, 0.1, 0, 0.001) # Prior parameters

# Visualize the results of the gibbs sampler
visualizer.checkSamplerConvergence(mean_samples, "mean")
visualizer.checkSamplerConvergence(tau_samples, "tau")
```

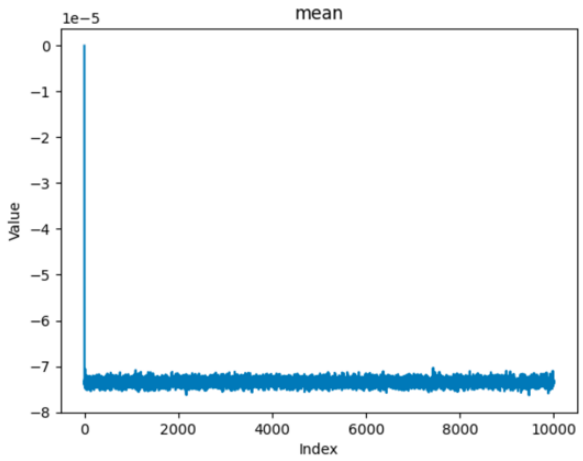


Figure 9: Samples of μ from the Gibbs Sampler

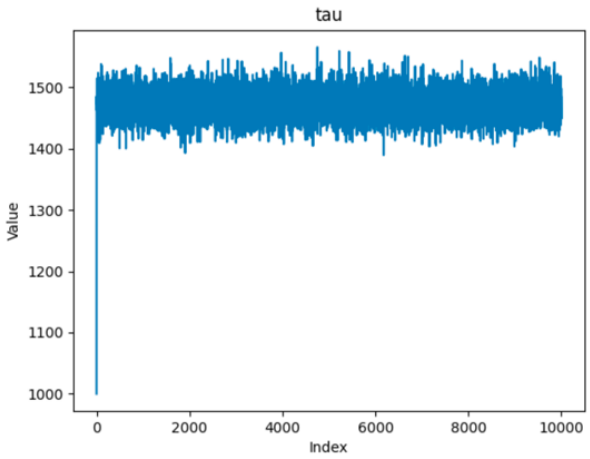


Figure 10: Samples of τ from the Gibbs Sampler

Hence, we have the posterior distribution by throwing new data into the prior distribution and update the distribution iteratively.

2. Validation of this distribution

Given the posterior distribution above, the second question arises: **Before assessing the financial risks by calculating the VaR and ES, how can we make sure that the distribution we found is a right distribution?**

By utilizing scipy.stats, we can have the functions of the auto-correlation function to check the convergence of our parameters, and here we exclude the first 100 samples using slicing, because during early iterations, the chain may still be "finding its way" toward the stationary distribution, so they are not representative of the true target distribution and can be biased.

```
# Pick the from the 100th sample for convergence
mean_samples = mean_samples[100:]
tau_samples = tau_samples[100:]

visualizer.checkSamplerACF(mean_samples, "mean ACF")
visualizer.checkSamplerACF(tau_samples, "tau ACF")
```

Hence, we plot the ACF for the mean and the inverse variance now.

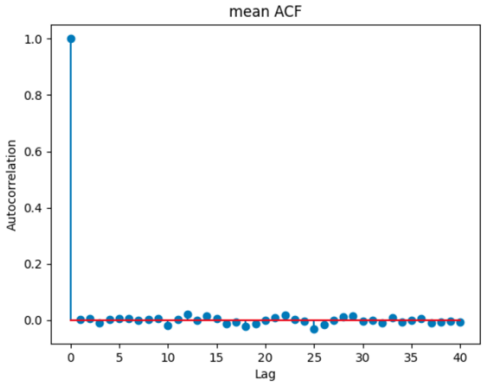


Figure 7: The ACF test on μ samples

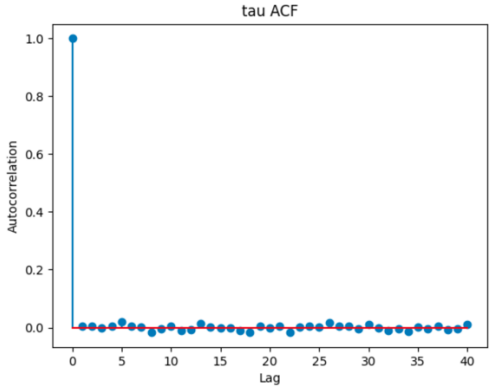


Figure 8: The ACF test on τ samples

As ACF values decay to zero at higher lags, it indicates the time series becomes statistically independent, meaning the chain efficiently explores the target distribution. Therefore, as demonstrated in the results, our samples points can be seen as independent with a correlation near 0, meaning this can be seen as a quite reliable distribution to be the basis of VaR and ES calculation.

3. Linear correlation between VaR an ES

After validating the distribution, we can get reliable data for calculating VaR and ES. Using multiple groups of μ and τ, thus getting multiple groups of VaR and ES using the given formula:

$$VaR\alpha=\mu+z\alpha\cdot\sigma$$

$$ES\alpha=\mu+\varphi(z\alpha)\cdot\sigma$$

The third question is: **What's the correlation between VaR and ES?**

According to their definitions, we assume the correlation between VaR and ES is linear. Thus, we use loops to calculate the linear regression equation with least squares method by the given formula:

$$\hat{y} = \hat{a} + \hat{b}x$$

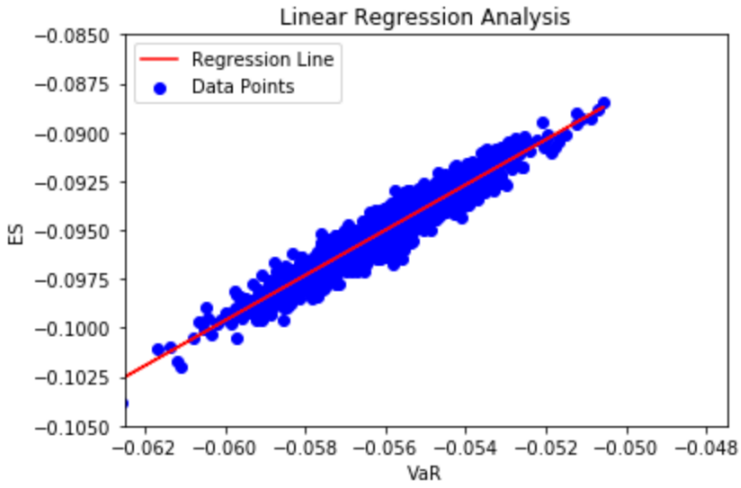
$$\hat{b} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i{}^2 - n\bar{x}^2}$$

$$\hat{a} = \bar{y} - \hat{b}\bar{x}$$

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The scatterplot and regression line can be drawn by plotting.

```python
import matplotlib.pyplot as plt

list1 = VaR_array
list2 = ES_array

n = len(list1)
x_mean = sum(list1) / n
y_mean = sum(list2) / n

# Calculate slope
numerator = sum((list1[i] - x_mean) * (list2[i] - y_mean) for i in range(n))
denominator = sum((list1[i] - x_mean) ** 2 for i in range(n))
slope = numerator / denominator

# Calculate intercept
intercept = y_mean - slope * x_mean

print(f"Slope: {slope}")
print(f"Intercept: {intercept}")

# Visualization of results
plt.scatter(list1, list2, color='blue', label='Data Points')
plt.plot(list1, [slope * x + intercept for x in list1], color='red', label='Regression Line')

# Set the limits for x and y axes as specified
plt.xlim(-0.0625, -0.0475)  # X-axis limits
plt.ylim(-0.105, -0.085)  # Y-axis limits

plt.xlabel('VaR')
plt.ylabel('ES')
plt.title('Linear Regression Analysis')
plt.legend()
plt.show()
```



By calculating the value of correlation coefficient: 0.946, we can find VaR and ES are highly linearly correlated, with the middle denser than the ends. It is valid thinking back of their definitions, with VaR representing maximum loss at a given confidence level and ES representing the expected loss in the tail of the distribution beyond the VaR threshold. Therefore, with high positive correlations between asset returns, tail risks are amplified.

Conclusion

This project exemplifies the efficiency of using programming in solving complex statistical problems, emphasizing the importance of computational efficiency, data structure organization, and algorithmic rigor in financial data analysis. Moreover, the methodology presented here may be applicable to many other contexts requiring risk assessment and inference modeling.

References

Saturn Cloud. (n.d.). What are logarithmic returns and how to calculate them in pandas dataframe? Retrieved December 1, 2024, from https://saturncloud.io/blog/what-are-logarithmic- returns-and-how-to-calculate-them-in-pandas-dataframe/

Peng, R. D. (2022). 7.3 Gibbs Sampler — Advanced Statistical Computing. Retrieved December 7, 2024, from https://bookdown.org/rdpeng/advstatcomp/gibbs-sampler.html

Notes

Our dataset is available here:

https://www.kaggle.com/datasets/jmselina/sse-shanghai-stock-exchange?resource=download

Questions:

1. Given the knowledge of Bayesian inference (Gibbs Sampling), how to find the posterior distribution?

2. Before assessing the financial risks by calculating the VaR and ES, how can we make sure that the distribution we found is a right distribution?

3. After obtaining reliable data of VaR and ES, what's the correlation between VaR and ES?