SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis . . . in Informatics

Portfolio Optimization with Gaussian Process Regression

Xiyue ZHANG

SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY —
INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis … in Informatics

Portfolio Optimization with Gaussian Process Regression

Titel der Abschlussarbeit

Author:	Xiyue ZHANG
Supervisor:	Prof. Dr. Hans-Jochim Bungartz
Advisor:	Kislaya Ravi
Submission Date:	December 3rd

I confirm that this master's thesis . . . is my own work and I have documented all sources and material used.

Munich, December 3rd Xiyue ZHANG

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Background and Motivation

1.1.1 Evolution of portfolio optimization techniques

The field of portfolio optimization has undergone significant transformations since its inception in the mid-20th century, evolving from simple diversification principles to sophisticated mathematical models incorporating machine learning and artificial intelligence. This evolution reflects both the advancing computational capabilities and our deepening understanding of financial markets' complexity.

Classical Foundations (1950s-1960s) The modern era of portfolio optimization began with [Mar52]'s seminal paper "Portfolio Selection," which laid the groundwork for **MPT!** (**MPT!**). Markowitz introduced several revolutionary concepts:

- The mathematical formalization of diversification
- The mean-variance optimization framework
- The efficient frontier of optimal portfolios
- The fundamental relationship between risk and return

This work established the first rigorous mathematical framework for portfolio selection, earning Markowitz the Nobel Prize in Economics and fundamentally changing how practitioners approached portfolio management.

Early Developments (1960s-1980s) Building upon Markowitz's foundation, several crucial developments emerged:

1. **Capital Asset Pricing Model (CAPM)**
    - Developed by [Sha64], [Lin65], and [Mos66]
    - Introduced systematic and unsystematic risk concepts
    - Established the theoretical framework for asset pricing

- Created the foundation for risk-adjusted performance measures

2. **Single-Index Models**
   - Simplified the estimation of covariance matrices
   - Reduced computational complexity
   - Introduced market beta as a risk measure
   - Enhanced practical applicability of portfolio optimization

Advanced Optimization Era (1980s-2000s) The advent of increased computational power led to more sophisticated approaches:

1. **Black-Litterman Model (1990s)**
   - Incorporated investor views into the optimization process
   - Addressed estimation error issues in mean-variance optimization
   - Introduced Bayesian methods to portfolio optimization
   - Provided more stable and intuitive portfolio allocations
2. **Risk-Based Portfolio Optimization**
   - Development of risk parity strategies [Qia05]
   - Introduction of alternative risk measures (VaR, CVaR)
   - Focus on downside risk management
   - Enhanced robustness to estimation errors

Modern Approaches (2000s-Present) Recent developments have focused on addressing classical methods' limitations:

1. **Robust Optimization**
   - Accounts for parameter uncertainty
   - Provides protection against worst-case scenarios
   - Incorporates estimation error in the optimization process
   - Yields more stable portfolio allocations
2. **Dynamic Portfolio Optimization**
   - Considers time-varying investment opportunities
   - Incorporates transaction costs

- Accounts for changing market conditions
- Enables adaptive portfolio management

3. **Machine Learning Integration**
   - Neural networks for return prediction [HPW17]
   - Support vector machines for risk assessment
   - Reinforcement learning for portfolio management
   - **GP!** (**GP!**) for uncertainty quantification

Current Challenges and Future Directions Modern portfolio optimization faces several challenges:

1. **Data Quality and Quantity**
   - High-dimensional data processing
   - Non-stationary market conditions
   - Alternative data integration
   - Real-time data processing requirements
2. **Model Complexity**
   - Balance between model sophistication and robustness
   - Computational efficiency
   - Interpretability of results
   - Parameter stability
3. **Implementation Challenges**
   - Transaction costs
   - Market impact
   - Regulatory constraints
   - Operational considerations

The field continues to evolve with emerging technologies and methodologies:

1. **Artificial Intelligence Applications**
   - Deep learning for market prediction
   - Natural language processing for sentiment analysis

    - Alternative data processing
    - Automated portfolio rebalancing

2. **Advanced Risk Management**
    - Tail risk hedging
    - Dynamic risk allocation
    - Scenario analysis
    - Real-time risk monitoring

3. **Sustainability Integration**
    - ESG factors in optimization
    - Climate risk consideration
    - Impact investing metrics
    - Sustainable portfolio construction

This evolution of portfolio optimization techniques sets the stage for our research, which builds upon these foundations while incorporating modern machine learning approaches, specifically Gaussian Process Regression, to address current challenges in portfolio optimization.

1.1.2 Role of machine learning in financial forecasting

ML! (**ML!**) has gained significant traction in financial markets due to its ability to analyze vast amounts of data and identify complex patterns that traditional models may overlook. In particular, **GPR!** (**GPR!**) has emerged as a powerful tool for time-series forecasting, offering a flexible framework for capturing non-linear relationships and uncertainty in predictions.

1.1.3 Challenges in time-series forecasting and traditional portfolio optimization methods

Despite the advancements in **ML!** techniques, predicting asset returns remains a challenging task due to the inherent volatility and non-stationarity of financial markets. Moreover, traditional portfolio optimization methods, while theoretically elegant, often face significant practical challenges in implementation. These challenges primarily stem from the difficulty in accurately estimating input parameters and the inherent uncertainty in financial time-series forecasting. This section examines these challenges and introduces how **GPR!** provides a novel approach to addressing them.

Parameter Estimation Challenges in Modern Portfolio Theory Modern Portfolio Theory (MPT), despite its theoretical elegance, relies heavily on accurate estimation of key parameters: The practical implementation of MPT is fundamentally constrained by the difficulty in estimating volatility, arguably the most critical parameter in portfolio optimization. Traditional approaches to volatility estimation rely heavily on historical data, assuming that past patterns will persist into the future. However, financial markets are dynamic systems characterized by regime changes, varying volatility clusters, and complex interdependencies that make such assumptions problematic. Historical volatility estimates are inherently backward-looking and highly sensitive to the chosen estimation window, leading to potentially misleading inputs for portfolio optimization.

Moreover, the challenge extends beyond simple volatility estimation. The correlation structure between assets, another crucial input for MPT, exhibits time-varying properties that are difficult to capture using conventional methods. During periods of market stress, these correlations often shift dramatically, invalidating historical estimates precisely when accurate risk assessment is most critical. The dimensionality of this problem grows quadratically with the number of assets, making it particularly challenging for large, diversified portfolios.

- **Volatility Estimation**
  - Historical volatility may not reflect future risk
  - Sample estimates are sensitive to the chosen time window
  - Regime changes can invalidate historical estimates
  - Heteroskedasticity in financial time series complicates estimation
- **Expected Returns**
  - Notoriously difficult to estimate accurately
  - High sensitivity to estimation errors
  - Time-varying nature of expected returns
  - Impact of market regimes on return distributions
- **Correlation Structure**
  - Dynamic nature of asset correlations
  - Curse of dimensionality in large portfolios
  - Instability during market stress periods
  - Computational challenges in high dimensions

Limitations of Traditional Forecasting Methods Traditional forecasting approaches in finance have predominantly relied on methods that provide point estimates, failing to capture the inherent uncertainty in financial predictions. These methods often make strong assumptions about the underlying data distribution and struggle to adapt to the non-linear, non-stationary nature of financial time series. **ARIMA!** (**ARIMA!**) models, exponential smoothing, and other classical approaches, while mathematically tractable, often fall short in capturing the complex dynamics of financial markets.

A fundamental limitation of these traditional approaches is their rigidity in handling uncertainty. Point forecasts, even when accompanied by confidence intervals based on historical volatility, fail to capture the dynamic nature of prediction uncertainty. This limitation becomes particularly problematic in portfolio optimization, where understanding the reliability of forecasts is as important as the forecasts themselves.

Conventional approaches to financial time-series forecasting exhibit several limitations:

1. **Point Estimates**
    - Traditional methods often provide single-point forecasts
    - Lack of uncertainty quantification
    - Limited ability to capture prediction confidence
    - Insufficient information for risk management
2. **Model Rigidity**
    - Assumption of specific probability distributions
    - Difficulty in capturing non-linear relationships
    - Limited adaptation to changing market conditions
    - Oversimplification of complex market dynamics
3. **Data Requirements**
    - Need for large historical datasets
    - Sensitivity to outliers and noise
    - Challenge of incorporating multiple data sources
    - Difficulty in handling missing data

Advantages of Gaussian Process Regression Our research proposes **GPR!** as a solution to these challenges, offering several key advantages:

1. **Probabilistic Framework**
    - Natural uncertainty quantification
    - Automatic volatility estimation through posterior variance
    - Capture of prediction confidence intervals
    - Robust handling of noise in financial data
2. **Flexible Modeling**
    - Non-parametric approach avoiding distributional assumptions
    - Ability to capture complex non-linear relationships
    - Automatic complexity adjustment through kernel selection
    - Incorporation of prior knowledge through kernel design
3. **Parameter Estimation**
    - Direct modeling of volatility through posterior variance
    - Joint estimation of returns and risk
    - Principled handling of uncertainty
    - Adaptive to changing market conditions

Addressing Traditional Limitations Our **GPR!**-based approach specifically addresses the key limitations of **MPT!**:

$$\sigma^2_{GPR}(x_*) = k(x_*, x_*) - k(x_*, X)[K(X, X) + \sigma_n^2 I]^{-1} k(X, x_*) \tag{1.1}$$

Where $\sigma^2_{GPR}(x_*)$ represents the posterior variance at prediction point x_*, providing a direct estimate of volatility that:

- Naturally accounts for uncertainty in predictions
- Adapts to local data density and quality
- Provides time-varying volatility estimates
- Incorporates both local and global market information

The **GPR!** approach offers several fundamental advantages over traditional methods. Unlike historical volatility estimates that require arbitrary window selection, **GPR!**'s volatility estimates emerge naturally from the probabilistic learning process.

The method adapts automatically to different market regimes through its kernel function, which can capture both long-term trends and short-term fluctuations in market behavior.

Furthermore, **GPR!**'s non-parametric nature frees it from restrictive assumptions about return distributions. The method can capture complex, non-linear patterns in the data while maintaining the ability to quantify uncertainty in its predictions. This combination of flexibility and uncertainty awareness makes it particularly well-suited for financial applications where both accuracy and risk assessment are crucial.

Implications for Portfolio Optimization The **GPR!** framework transforms the traditional portfolio optimization problem by: The integration of **GPR!** into portfolio optimization transforms the traditional **MPT!** framework by providing more reliable and dynamic parameter estimates. By directly modeling the uncertainty in our predictions, we can make more informed portfolio allocation decisions that account for both expected returns and our confidence in those expectations. This approach naturally leads to more robust portfolios that adapt to changing market conditions while maintaining a principled approach to risk management.

The significance of this advancement cannot be overstated. By addressing one of the fundamental criticisms of **MPT!** – the difficulty of accurately estimating volatility – our **GPR!**-based approach bridges the gap between theoretical elegance and practical applicability. This enhancement makes **MPT!** more reliable and useful for real-world portfolio management, where accurate risk assessment is crucial for maintaining stable, long-term investment performance.

1.2 Research Objectives

The primary objective of this study is to develop a predictive portfolio optimization framework that leverages Gaussian Process Regression (GPR) for time-series forecasting in financial markets. By integrating advanced predictive modeling with strategic asset allocation, the study aims to enhance investment performance through informed decision-making. The specific objectives are as follows:

1. **Develop and Validate GPR! Models for Asset Return Prediction**
   - *Model Construction:* Build individual GPR models to forecast future returns of selected assets, including forex, gold, Bitcoin, and various stocks.
   - *Feature Engineering:* Utilize historical one-month returns and time as input features to capture both market dynamics and temporal patterns.

- *Model Updating:* Implement an iterative training process where models are updated daily with new market data to ensure predictions remain current and adaptive.

2. **Integrate GPR! Predictions into Portfolio Optimization Strategies**
   - *Expected Returns and Volatilities:* Extract predicted returns and associated volatilities from the GPR models for use in portfolio construction.
   - *Strategy Formulation:* Design multiple optimization strategies—Maximum Return, Minimum Volatility, Maximum Sharpe Ratio, and a Dynamic Strategy—based on the **GPR!** outputs.

3. **Develop a Dynamic Portfolio Optimization Strategy**
   - *Probabilistic Assessment:* Calculate the probability distribution of next-day cumulative portfolio returns using the predicted normal distribution of asset returns.
   - *Threshold-Based Decision Making:* Establish a threshold probability to decide when to reallocate the portfolio for maximizing returns versus holding the current positions.
   - *Adaptive Allocation:* Enable the portfolio to adapt dynamically to changing market conditions by selectively applying the Maximum Return Strategy based on probabilistic forecasts.

4. **Evaluate and Compare the Performance of Optimization Strategies**
   - *Backtesting Framework:* Conduct backtesting over a historical period using real market data to assess the strategies' performance.
   - *Incorporation of Transaction Costs:* Include realistic transaction fees in the evaluation to account for the costs associated with portfolio rebalancing.
   - *Performance Metrics:* Measure total returns, portfolio volatility, Sharpe ratios, and transaction costs to provide a comprehensive performance analysis.

5. **Demonstrate the Effectiveness of the Dynamic Strategy**
   - *Performance Analysis:* Analyze the results to determine if the Dynamic Strategy achieves higher returns and lower transaction costs compared to traditional strategies.
   - *Risk Management:* Assess how the Dynamic Strategy balances the trade-off between pursuing higher returns and minimizing risks and costs.

- *Statistical Significance:* Use statistical methods to verify the significance of the observed performance differences among the strategies.

6. **Contribute to the Field of Predictive Portfolio Optimization**
   - *Innovative Approach:* Present a novel integration of GPR-based forecasting with adaptive portfolio optimization strategies.
   - *Practical Implications:* Provide insights and recommendations for practitioners on implementing dynamic, data-driven approaches in portfolio management.
   - *Foundation for Future Research:* Establish a basis for further exploration into advanced predictive models and adaptive strategies in financial optimization.

By achieving these objectives, the study seeks to demonstrate that incorporating sophisticated predictive models like **GPR!** into portfolio optimization can significantly enhance investment outcomes. The findings aim to contribute valuable knowledge to the field of quantitative finance, particularly in the areas of time-series forecasting and dynamic asset allocation.

1.2.1 Research Contributions

This study makes several significant contributions to the field of predictive portfolio optimization and quantitative finance. The key research contributions are outlined below:

1. **Novel Integration of Gaussian Process Regression with Dynamic Portfolio Optimization**

   This research presents a unique integration of **GPR!** models with dynamic portfolio optimization strategies. By employing **GPR!** for time-series forecasting, we capture complex, non-linear relationships in financial data, enhancing the accuracy of return predictions. The integration facilitates a more responsive and informed portfolio allocation process, adapting to market changes in real-time.

2. **Probabilistic Approach to Strategy Selection**

   We introduce a probabilistic framework for strategy selection within the portfolio optimization process. By calculating the probability distribution of future cumulative returns, the Dynamic Strategy makes informed decisions on whether to reallocate the portfolio based on a predefined threshold. This approach incorporates uncertainty and risk directly into the decision-making process, allowing for a more nuanced and adaptive investment strategy.

3. **Practical Implementation Considering Transaction Costs**

   The study emphasizes practical applicability by incorporating realistic transaction costs into the optimization and backtesting processes. By accounting for these costs, we provide a more accurate assessment of the strategies' net performance. This consideration is crucial for real-world portfolio management, where transaction fees can significantly impact returns, especially in high-frequency trading environments.

4. **Comparative Analysis of Different Optimization Strategies**

   We conduct a comprehensive comparative analysis of multiple portfolio optimization strategies, including Maximum Return, Minimum Volatility, Maximum Sharpe Ratio, and the proposed Dynamic Strategy. By evaluating these strategies under the same conditions and performance metrics, we provide valuable insights into their relative effectiveness. This analysis helps identify the strengths and limitations of each approach, guiding practitioners in selecting appropriate strategies based on their investment goals and risk tolerance.

These contributions collectively advance the understanding of how advanced predictive models and adaptive strategies can be effectively combined to enhance portfolio performance. The novel methodologies and findings offer practical benefits for portfolio managers and lay the groundwork for future research in predictive asset allocation.

```sql
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 1.1: An example for a source code listing.

!TeX root = ../main.tex

2 Previous Literature

2.1 Theoretical Framework

This chapter discusses the fundamental theories underpinning the study, focusing on portfolio optimization and **GPR!**. Understanding these theories is essential for developing the predictive portfolio optimization framework proposed in this research.

2.1.1 Portfolio Optimization Theory

Portfolio optimization is a cornerstone of modern finance, aiming to allocate assets in a way that balances expected returns against risk. The foundational theory in this domain is the **MPT!**, introduced by Harry Markowitz in 1952 [Mar52].

Modern Portfolio Theory (MPT)

MPT! posits that investors can construct an optimal portfolio that offers the maximum expected return for a given level of risk or, equivalently, the minimum risk for a given level of expected return. The key assumptions of MPT are:

- Investors are rational and risk-averse, preferring higher returns and lower risk.
- Markets are efficient, and all investors have access to the same information.
- Asset returns are normally distributed and can be described by their mean (expected return) and variance (risk).

Expected Return and Risk The expected return of a portfolio, $E[R_p]$, is the weighted sum of the expected returns of the individual assets:

$$E[R_p] = \sum_{i=1}^{n} w_i E[R_i], \tag{2.1}$$

where w_i is the weight of asset i in the portfolio, $E[R_i]$ is the expected return of asset i, and n is the total number of assets.

The portfolio variance, σ_p^2, representing risk, is given by:

$$\sigma_p^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \sigma_{ij}, \tag{2.2}$$

where σ_{ij} is the covariance between asset i and asset j. The standard deviation σ_p is the square root of the variance.

Efficient Frontier The set of optimal portfolios that offer the highest expected return for a given level of risk forms the *Efficient Frontier*. Portfolios on the efficient frontier are considered optimal, as no other portfolios offer higher returns for the same risk level.

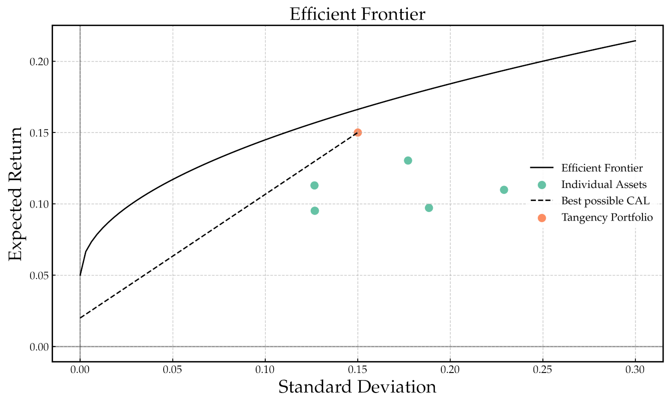

Figure 2.1: Efficient Frontier in Mean-Variance Space

Mean-Variance Optimization Mean-variance optimization involves solving for the portfolio weights that minimize the portfolio variance for a given expected return. The optimization problem can be formulated as:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} \\ \text{subject to} \quad & \mathbf{w}^T \mathbf{E} = E_p, \\ & \sum_{i=1}^{n} w_i = 1, \\ & w_i \geq 0, \quad i = 1, \ldots, n, \end{aligned} \tag{2.3}$$

where:

- $\mathbf{w} \in \mathbb{R}^n$ is the vector of asset weights

- $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$ is the covariance matrix of asset returns
- $\mathbf{E} \in \mathbb{R}^n$ is the vector of expected asset returns
- $E_p \in \mathbb{R}$ is the desired expected portfolio return

Risk-Return Trade-off and the Sharpe Ratio

Investors seek to maximize returns while minimizing risk. The *Sharpe Ratio*, introduced by William F. Sharpe [**sharpe1966mutual**], measures the risk-adjusted return of a portfolio:

$$S = \frac{E[R_p] - R_f}{\sigma_p}, \tag{2.4}$$

where R_f is the risk-free rate. A higher Sharpe Ratio indicates a more favorable risk-return trade-off.

Portfolio Optimization Strategies

Various strategies exist for portfolio optimization, each with different objectives and constraints:

- **Maximum Return Strategy**: Focuses on maximizing expected returns, often leading to higher risk.
- **Minimum Volatility Strategy**: Aims to minimize risk while achieving a minimum acceptable return.
- **Maximum Sharpe Ratio Strategy**: Seeks the optimal balance between return and risk by maximizing the Sharpe Ratio.
- **Dynamic Strategies**: Adjust portfolio allocations based on changing market conditions and predictive insights.

2.1.2 Gaussian Process Regression Theory and Applications

Gaussian Process Regression (GPR) is a non-parametric, Bayesian approach to regression that is particularly powerful for modeling complex, non-linear relationships. GPR provides not only predictions but also uncertainty estimates, which are valuable in risk-sensitive applications like finance.

Gaussian Processes

A *Gaussian Process* (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution **[rasmussen2006gaussian]**. A GP is fully specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$:

$$f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{2.5}$$

Mean Function The mean function $m(\mathbf{x})$ represents the expected value of the function at point $\mathbf{x}$:

$$m(\mathbf{x}) = E[f(\mathbf{x})]. \tag{2.6}$$

Covariance Function The covariance function $k(\mathbf{x}, \mathbf{x}')$ defines the covariance between function values at points $\mathbf{x}$ and $\mathbf{x}'$:

$$k(\mathbf{x}, \mathbf{x}') = E\left[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))\right]. \tag{2.7}$$

Common choices for the covariance function include the squared exponential kernel and the Matérn kernel.

Gaussian Process Regression for Time-Series Forecasting

GPR models the underlying function mapping inputs to outputs, capturing uncertainty in predictions. For time-series forecasting:

- **Inputs**: Historical data points, such as lagged returns and time indices.
- **Outputs**: Future values of the time series, such as asset returns.

Given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i$ are inputs and y_i are observations, the goal is to predict the output f_* at a new input $\mathbf{x}_*$.

Predictive Distribution The predictive distribution of f_* given the training data is Gaussian:

$$p(f_* | \mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}\left(f_* | \mu_*, \sigma_*^2\right), \tag{2.8}$$

where

$$\mu_* = k_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}, \sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) - k_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} k_*. \tag{2.9}$$

Here, $k_* = [k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_N)]^\top$, $\mathbf{K}$ is the covariance matrix with entries $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, and σ_n^2 is the noise variance.

Covariance Functions (Kernels)

The choice of kernel function $k(\mathbf{x}, \mathbf{x}')$ is critical in GPR as it encodes assumptions about the function being learned.

- *Squared Exponential Kernel*:

$$k_{\text{SE}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2\ell^2}||\mathbf{x} - \mathbf{x}'||^2\right), \tag{2.10}$$

  where σ_f^2 is the signal variance and ℓ is the length-scale parameter.

- *Matérn Kernel*:

$$k_{\text{Matérn}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}||\mathbf{x} - \mathbf{x}'||}{\ell}\right)^\nu K_\nu \left(\frac{\sqrt{2\nu}||\mathbf{x} - \mathbf{x}'||}{\ell}\right), \tag{2.11}$$

  where ν controls the smoothness, Γ is the gamma function, and K_ν is the modified Bessel function.

Gaussian Process Regression for Time-Series Forecasting

In GPR, given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the goal is to predict the output y_* for a new input $\mathbf{x}_*$. The predictive distribution is Gaussian with mean and variance:

$$E[y_*] = \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}, \tag{2.12}$$

$$\text{Var}[y_*] = k_{**} - \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*, \tag{2.13}$$

where:

- $\mathbf{K}$ is the $n \times n$ covariance matrix evaluated at the training inputs.
- $\mathbf{k}_*$ is the covariance vector between the test point and the training inputs.
- k_{**} is the covariance between the test point and itself.
- σ_n^2 is the variance of the observation noise.
- $\mathbf{y}$ is the vector of training targets.

Advantages of GPR in Financial Modeling GPR offers several benefits for financial time-series forecasting:

- *Non-parametric Flexibility*: Does not assume a specific functional form, allowing for modeling of complex, non-linear relationships.
- *Uncertainty Quantification*: Provides probabilistic predictions with associated confidence intervals, which are crucial for risk management.
- *Bayesian Framework*: Naturally incorporates prior knowledge and updates beliefs with new data.

Challenges in Applying GPR to Finance

Despite its advantages, GPR faces challenges in financial applications:

- *Computational Complexity*: Involves inverting an $n \times n$ matrix, which can be computationally intensive for large datasets.
- *Non-Stationarity*: Financial time-series often exhibit non-stationary behavior, violating assumptions of stationarity in standard GPR.
- *Noise Characteristics*: Financial data can be noisy and volatile, affecting model performance.
- *Hyperparameter Tuning*: Selection of kernel functions and hyperparameters significantly impacts model performance and requires careful tuning.

2.1.3 Integrating Portfolio Optimization and GPR

Combining portfolio optimization with GPR-based forecasting aims to leverage accurate predictions of asset returns and associated uncertainties to make informed allocation decisions. The integration involves:

- Using GPR to predict expected returns and volatilities for assets.
- Incorporating these predictions into the optimization models to determine optimal portfolio weights.
- Adjusting for uncertainties by considering the confidence intervals provided by GPR in the optimization process.

Dynamic Portfolio Optimization Dynamic optimization involves updating the portfolio allocation as new information becomes available. By retraining the GPR models with new data and adjusting the portfolio accordingly, the strategy adapts to changing market conditions, potentially enhancing performance.

2.1.4 Conclusion

The theoretical foundation provided by portfolio optimization theory and Gaussian Process Regression is critical for developing the predictive portfolio optimization framework. Understanding the principles, advantages, and limitations of these theories allows for effective integration and application in financial modeling and asset allocation.

2.2 Risk measures and portfolio optimization

2.3 Machine Learning in Financial Markets

2.3.1 Overview of Machine Learning Applications in Finance

The financial markets generate vast amounts of data daily, encompassing stock prices, trading volumes, economic indicators, and news articles. Machine learning algorithms are uniquely positioned to process and analyze this data, uncovering patterns and insights that traditional statistical methods may overlook. Key applications of machine learning in finance include:

Time-Series Forecasting

Predicting future asset prices and market trends is a fundamental objective in finance. Machine learning models, such as neural networks, support vector machines, and ensemble methods, are employed to forecast time-series data by capturing non-linear relationships and complex temporal dependencies [**sezer2020financial**].

Algorithmic Trading

Machine learning algorithms facilitate the development of automated trading systems that execute trades based on predefined strategies and real-time data analysis. Techniques like reinforcement learning enable the optimization of trading strategies through continuous learning from market interactions [**nevmyvaka2006reinforcement**].

Risk Management

Accurate risk assessment is crucial for financial institutions. Machine learning models help in predicting credit risk, market risk, and operational risk by analyzing historical data and identifying factors that contribute to potential losses [**lessmann2015benchmarking**].

Portfolio Optimization

Machine learning enhances portfolio optimization by providing more accurate estimates of expected returns and covariances between assets. Advanced models can adapt to changing market conditions and incorporate a broader set of predictive features [HPW17].

Fraud Detection and Anomaly Detection

Detecting fraudulent activities and anomalies is vital for maintaining the integrity of financial systems. Machine learning algorithms, particularly unsupervised learning techniques, are used to identify unusual patterns in transaction data that may indicate fraud [**phua2010comprehensive**].

Sentiment Analysis and Natural Language Processing

Analyzing news articles, social media, and financial reports using natural language processing (NLP) helps investors gauge market sentiment and its potential impact on asset prices [**hagenau2013automated**].

2.3.2 Conclusion

Machine learning plays a pivotal role in modern financial analysis, offering sophisticated tools for modeling and decision-making. Gaussian Process Regression, with its probabilistic nature and flexibility, is particularly well-suited for financial applications that require modeling uncertainty and non-linear relationships. Understanding the theoretical foundations and practical considerations of GPR is essential for its effective integration into financial modeling and portfolio optimization.

2.4 Dynamic Portfolio Management

Dynamic portfolio management involves continuously adjusting asset allocations in response to changing market conditions, forecasts, and investment objectives. Unlike static strategies, dynamic approaches aim to optimize the portfolio over time by incorporating new information and adapting to market dynamics. This section reviews dynamic optimization strategies, discusses the impact of transaction costs on portfolio rebalancing, and examines existing approaches to strategy switching.

2.4.1 Review of Dynamic Optimization Strategies

Dynamic optimization strategies are designed to adapt portfolio allocations over time, taking into account the stochastic nature of asset returns and changing investment opportunities. Key concepts and methods in dynamic portfolio optimization include:

Dynamic Programming

Dynamic programming is a mathematical optimization approach that solves complex problems by breaking them down into simpler subproblems. In the context of portfolio optimization, dynamic programming can be used to determine the optimal investment policy over multiple periods [**bellman1957dynamic**].

Bellman's Principle of Optimality Bellman's principle states that an optimal policy has the property that, whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. This principle underpins dynamic programming methods in portfolio optimization.

Stochastic Control Theory

Stochastic control theory deals with decision-making in systems that evolve over time under uncertainty. In portfolio optimization, stochastic control models can determine optimal asset allocations by considering the stochastic processes governing asset returns and investor wealth [**merton1969lifetime**].

Merton's Portfolio Problem Robert C. Merton extended the continuous-time portfolio optimization framework by incorporating stochastic control techniques. Merton's model determines the optimal consumption and portfolio allocation strategies for an investor with a finite or infinite investment horizon, maximizing expected utility [**merton1971optimum**].

Reinforcement Learning

Reinforcement learning (RL) is a machine learning paradigm where agents learn optimal policies through interactions with the environment by maximizing cumulative rewards. In finance, RL can be applied to portfolio management by treating asset allocation as a sequential decision-making problem [**moody1998performance**].

Applications in Portfolio Management RL algorithms, such as Q-learning and policy gradients, have been used to develop adaptive trading strategies that learn from market data and adjust allocations dynamically [**almahdi2019adaptive**].

Scenario Analysis and Model Predictive Control

Scenario analysis involves evaluating portfolio performance under different hypothetical future states of the world. Model Predictive Control (MPC) uses forecasts to optimize current decisions while considering future trajectories, adjusting strategies as new information becomes available [**primbs2009dynamic**].

Advantages of MPC MPC allows for the incorporation of predictive models (e.g., GPR forecasts) into the optimization process, enabling the portfolio to adapt dynamically to anticipated market changes.

2.4.2 Transaction Costs and Portfolio Rebalancing

Transaction costs play a significant role in dynamic portfolio management, as frequent rebalancing can erode returns. Understanding and modeling transaction costs are essential for effective portfolio optimization.

Types of Transaction Costs

Transaction costs can be broadly categorized into:

- *Fixed Costs*: Costs that are constant per transaction, such as brokerage fees.
- *Variable Costs*: Costs that depend on the transaction size, including bid-ask spreads and market impact.
- *Slippage*: The difference between the expected transaction price and the actual execution price.

Impact on Portfolio Performance

High transaction costs can negate the benefits of frequent rebalancing. Incorporating transaction costs into the optimization model encourages more conservative adjustments, potentially leading to better net performance [**garleanu2009dynamic**].

Optimal Rebalancing Frequency

Determining the optimal frequency of portfolio rebalancing involves a trade-off between maintaining the desired asset allocation and minimizing transaction costs. Strategies to address this include:

- *Threshold-Based Rebalancing*: Rebalancing only when asset weights deviate beyond predefined thresholds.
- *Periodic Rebalancing*: Adjusting the portfolio at regular intervals (e.g., monthly, quarterly).
- *Cost-Aware Optimization*: Incorporating transaction costs directly into the optimization problem to balance expected returns against costs.

2.4.3 Existing Approaches to Strategy Switching

Strategy switching involves changing between different portfolio optimization strategies based on market conditions, forecasts, or performance metrics. This adaptive approach seeks to capitalize on the strengths of various strategies under different scenarios.

Regime-Switching Models

Regime-switching models assume that financial markets alternate between different states or regimes (e.g., bull and bear markets). By identifying the current regime, investors can switch to strategies that are better suited to prevailing conditions [**ang2002regime**].

Markov Switching Models These models use Markov processes to model transitions between regimes, allowing for probabilistic predictions of future states and informing strategy selection [**hamilton1989new**].

Meta-Learning and Ensemble Methods

Meta-learning approaches involve training a higher-level model to select the best-performing strategy from a set of candidates. Ensemble methods combine multiple strategies to create a more robust overall strategy [**huang2019building**].

Performance-Based Selection Strategies can be evaluated based on historical or real-time performance metrics, such as Sharpe ratios or drawdowns, with the best-performing strategy selected for implementation [**poterba2000portfolio**].

Rule-Based Switching

Rule-based systems use predefined criteria or indicators to trigger strategy changes. Examples include:

- *Technical Indicators*: Switching strategies based on signals from moving averages, momentum indicators, or relative strength indexes.
- *Economic Indicators*: Adjusting strategies in response to macroeconomic data releases or changes in interest rates.
- *Forecast Confidence*: Modifying the strategy based on the confidence level of predictive models, such as forecast variances from GPR.

Machine Learning-Based Switching

Machine learning models can be trained to predict the optimal strategy based on market data and indicators. Classification algorithms, such as support vector machines or neural networks, can identify patterns associated with the outperformance of specific strategies [**fernandez2018machine**].

2.4.4 Conclusion

Dynamic portfolio management seeks to enhance investment performance by adapting to market conditions and new information. Incorporating transaction costs into the optimization process is crucial for realistic strategy implementation. Existing approaches to strategy switching provide valuable frameworks for developing adaptive strategies, leveraging techniques such as regime-switching models, meta-learning, rule-based systems, and machine learning. This study builds upon these concepts by introducing a probabilistic, threshold-based strategy switching mechanism informed by Gaussian Process Regression forecasts.

3 Methodology

3.1 Data Collection and Preprocessing

3.1.1 Asset Selection and Justification

In constructing the portfolio for this study, we selected five stocks representing five major sectors out of the eleven sectors defined by the Global Industry Classification Standard (GICS): Information Technology, Health Care, Industrials, Consumer Discretionary, and Financials. These sectors were chosen due to their significant contributions to the S&P 500 Index, collectively accounting for the majority of the index's market capitalization. The selected stocks are as follows: **MSFT!** (**MSFT!**) from the Information Technology sector, **LLY!** (**LLY!**) from the Health Care sector, **ETN!** (**ETN!**) from the Industrials sector, **HLT!** (**HLT!**) from the Consumer Discretionary sector, and **JPM!** (**JPM!**) from the Financials sector. And these five individual stocks are selected based on entropy analysis and comparison.

Stock Market Sectors The rationale behind this selection is multifaceted. Firstly, by including stocks from these predominant sectors, the portfolio captures the performance dynamics of the key drivers of the U.S. equity market. This approach ensures that the portfolio is both diversified and representative of broader market movements. Diversification across different sectors helps mitigate unsystematic risk associated with individual industries or companies, allowing the portfolio to benefit from growth opportunities while reducing the impact of sector-specific downturns.

Secondly, focusing on sectors that contribute most significantly to the S&P 500 enhances the relevance of the study's findings to investors who benchmark their performance against this widely recognized index. The Information Technology sector, for example, encompasses leading companies in innovation and technology, significantly influencing market trends. The Health Care sector includes firms pivotal in advancing medical technology and pharmaceuticals, demonstrating resilience in various economic conditions.

Industrials and Consumer Discretionary sectors represent core components of the economy, reflecting business cycles and consumer spending patterns, respectively. The Financials sector plays a critical role in economic infrastructure, encompassing banking, insurance, and investment services. By selecting assets from these sectors, the portfolio encompasses areas crucial to economic growth and market capitalization.

This strategic selection aligns with the study's objective of developing a predictive portfolio optimization framework applicable to a realistic and impactful set of assets. It allows for testing the effectiveness of the proposed models and strategies in a context that is meaningful to investors and relevant to overall market performance. Moreover, by focusing on sectors that significantly influence the S&P 500, the study ensures that its results have practical implications for portfolio management and investment decision-making.

Entropy Methods for Time Series Analysis To assess the complexity of the time-series data for these assets, we utilized entropy-based methods. Specifically, we employed the `OrdianlEntropy` package in Python, which provides time-efficient, ordinal pattern-based entropy algorithms for computing the complexity of one-dimensional time-series. This analysis informed our feature engineering process by highlighting the inherent unpredictability and dynamic behavior of the asset prices.

Permutation Entropy (PE) Permutation Entropy (PE), introduced by Bandt and Pompe in 2002, is a measure of complexity that quantifies the diversity of ordinal patterns in a time series. It is based on the relative ordering of the values within embedding vectors derived from the time series. Given a time series $x_{t_{t=1}}^{N}$, the calculation of PE involves the following steps:

1. **Embedding the Time Series:** Choose an embedding dimension m and time delay τ, and construct embedding vectors:
$$s_i = \left(x_i, x_{i+\tau}, x_{i+2\tau}, \dots, x_{i+(m-1)\tau}\right), \quad i = 1, 2, \dots, N-(m-1)\tau. \tag{3.1}$$
2. **Forming Ordinal Patterns:**
For each embedding vector s_i, determine the permutation π_i that sorts its components in ascending order:
$$s_i \to \left(x_{i+\tau(k_0)} \leq x_{i+\tau(k_1)} \leq \cdots \leq x_{i+\tau(k_{m-1})}\right), \tag{3.2}$$
where $(k_0, k_1, \dots, k_{m-1})$ represents the indices of the sorted components.
3. **Calculating Probabilities:**
Count the frequency of each of the $m!$ possible ordinal patterns π_j, and estimate their probabilities:
$$p(\pi_j) = \frac{\text{Number of times pattern } \pi_j \text{ occurs}}{N-(m-1)\tau}. \tag{3.3}$$

4. **Calculating Permutation Entropy:**

   Compute the permutation entropy using Shannon's entropy formula:

$$\mathrm{PE} = -\sum_{j=1}^{m!} p(\pi_j) \ln p(\pi_j). \tag{3.4}$$

   Normalize PE by dividing by $\ln(m!)$ to ensure it ranges between 0 and 1:

$$\mathrm{PE}_{\mathrm{norm}} = \frac{\mathrm{PE}}{\ln(m!)}. \tag{3.5}$$

- A **low PE** value indicates that the time series has a predictable or regular structure, with fewer unique ordinal patterns.
- A **high PE** value suggests that the time series is more complex or random, exhibiting a higher diversity of ordinal patterns.

Weighted Permutation Entropy (WPE) Weighted Permutation Entropy (WPE) enhances PE by incorporating the amplitude information of the time series. It assigns weights to ordinal patterns based on the values within the embedding vectors, thus accounting for both the order and magnitude of the data. The calculation of WPE involves the following steps:

1. **Embedding and Ordinal Patterns:** Follow steps 1 and 2 as in the calculation of PE to obtain the embedding vectors and their corresponding ordinal patterns.

2. **Assigning Weights:**

   For each embedding vector s_i, calculate a weight w_i based on the amplitude of its components. A common choice is the variance or absolute differences:

$$w_i = \mathrm{Var}(s_i) = \frac{1}{m} \sum_{k=0}^{m-1} \left(x_{i+\tau k} - \bar{x}_i\right)^2, \tag{3.6}$$

   where $\bar{x}_i$ is the mean of the components of s_i.

3. **Calculating Weighted Probabilities:**

   For each ordinal pattern π_j, compute the weighted probability:

$$p_w(\pi_j) = \frac{\sum_{s_i \in \pi_j} w_i}{\sum_{i=1}^{N-(m-1)\tau} w_i}. \tag{3.7}$$

4. **Calculating Weighted Permutation Entropy:**

   Compute WPE using the weighted probabilities:

   $$\mathrm{WPE} = -\sum_{j=1}^{m!} p_w(\pi_j) \ln p_w(\pi_j). \tag{3.8}$$

   Normalize WPE similarly to PE:

   $$\mathrm{WPE}_{\mathrm{norm}} = \frac{\mathrm{WPE}}{\ln(m!)}. \tag{3.9}$$

- **WPE** reflects both the diversity of ordinal patterns and the magnitude of fluctuations in the time series.
- It is more sensitive to significant changes in amplitude, making it useful for detecting volatility and abrupt transitions.

Dispersion Entropy (DE) Dispersion Entropy (DE), introduced by Rostaghi and Azami in 2016, is an entropy measure that accounts for both the ordering and dispersion of data by mapping the time series into a sequence of classes or symbols. To calculate DE, follow these steps:

1. **Mapping Data to Classes:** Define c classes and map each data point x_t to a class label k_t using a mapping function, such as:

   $$k_t = \left\lfloor c \cdot \frac{x_t - x_{\min} + \epsilon}{x_{\max} - x_{\min} + 2\epsilon} \right\rfloor + 1, \quad k_t \in \{1, 2, \ldots, c\}, \tag{3.10}$$

   where $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the time series, and ϵ is a small constant to prevent division by zero.

2. **Embedding:**

   Form embedding vectors of class labels:

   $$s_i = \left(k_i, k_{i+\tau}, k_{i+2\tau}, \ldots, k_{i+(m-1)\tau}\right). \tag{3.11}$$

3. **Creating Dispersion Patterns:**

   Each embedding vector s_i represents a dispersion pattern d_j, where j indexes the possible patterns.

4. **Calculating Probabilities:**

   Count the frequency of each possible dispersion pattern and estimate their probabilities:
   $$p(d_j) = \frac{\text{Number of times pattern } d_j \text{ occurs}}{N - (m-1)\tau}. \tag{3.12}$$

5. **Calculating Dispersion Entropy:**

   Compute DE using:
   $$\mathrm{DE} = -\sum_{j=1}^{c^m} p(d_j) \ln p(d_j). \tag{3.13}$$

   Normalize DE by dividing by $\ln(c^m)$:
   $$\mathrm{DE}_{\mathrm{norm}} = \frac{\mathrm{DE}}{\ln(c^m)}. \tag{3.14}$$

- **DE** considers both the frequency and dispersion of data, providing a robust measure of complexity for non-stationary time series.
- It effectively captures changes in amplitude and is less sensitive to noise and outliers.

Choosing WPE and DE for Stock Price Analysis Analyzing stock prices requires accounting for both the order and magnitude of price changes due to the following reasons:

Advantages of WPE

- **Amplitude Incorporation:** By weighting ordinal patterns, WPE accounts for the magnitude of price changes, capturing volatility.
- **Volatility Sensitivity:** It is effective in detecting periods of high market volatility, which are critical for risk management.
- **Balanced Complexity:** WPE balances computational efficiency with the need for detailed analysis.

Advantages of DE

- **Non-Stationarity Adaptation:** DE effectively handles non-stationary time series by mapping data into classes.
- **Noise Robustness:** It is less sensitive to noise and outliers, providing reliable complexity measures in the presence of market anomalies.
- **Comprehensive Analysis:** DE considers both the frequency and amplitude of patterns, offering a holistic view of market dynamics.
- **Amplitude Sensitivity:** The size of price movements affects returns and risk; thus, methods that incorporate amplitude information are essential.
- **Volatility Detection:** Financial markets are characterized by varying volatility, which influences investment decisions.
- **Non-Stationarity Handling:** Stock prices exhibit trends and abrupt changes, necessitating entropy measures that handle non-stationary data effectively.

Conclusion Weighted Permutation Entropy and Dispersion Entropy are more suitable than Permutation Entropy for analyzing stock prices because they:

- Capture essential information about both the direction and magnitude of price movements.
- Are better equipped to handle the inherent non-stationarity and volatility of financial time series.
- Provide more informative and sensitive complexity measures that can enhance asset selection and risk assessment strategies.

By utilizing WPE and DE, analysts and investors can gain deeper insights into market behavior, aiding in the selection of assets across different sectors based on the complexity and predictability of their stock prices.

3.1.2 Data Sources and Time Period

To acquire the historical market data necessary for this study, we employed the EOD Historical Data API, a reputable and comprehensive source for end-of-day and historical financial data across various asset classes. The data retrieval process was automated using a Python script that constructs API requests based on the asset ticker, desired data period (e.g., daily, weekly), and specified date range.

For U.S.-listed assets, the script utilizes the endpoint format:

```
https://eodhd.com/api/eod/{ticker}.US?period={period}&api_token={api_token}
```

where `{ticker}` represents the stock symbol, `{period}` denotes the data frequency, `{api_token}` is the authentication token, and `{start_date}` and `{end_date}` define the data range. For Bitcoin (BTC), which is categorized differently in the API, the script accesses data using the endpoint:

```
https://eodhd.com/api/eod/BTC-USD.CC?period={period}&api_token={api_token}
```

An API token, securely stored using environment variables to maintain confidentiality, is included in the requests for authentication. The script handles HTTP responses by checking for successful status codes and raising exceptions in case of errors, ensuring robust data retrieval.

Upon receiving a valid response, the JSON data is parsed into a pandas DataFrame for efficient data manipulation and analysis. The DataFrame includes essential financial indicators such as open, high, low, close prices, and trading volumes. The data is then saved as a CSV file in a structured directory hierarchy corresponding to each asset, following the path:

```
../Stocks/{ticker}/{ticker}_us_{period}.csv
```

By automating the data fetching and saving process, we ensured consistency and repeatability in data collection of the whole pipeline. This method allowed us to systematically gather historical market data for all selected assets over the specified time periods, providing a reliable dataset for training the Gaussian Process Regression models and conducting backtesting for the portfolio optimization strategies. The use of the EOD Historical Data API ensured that the data was up-to-date and accurate, reflecting real market conditions essential for the validity of our analysis. Also, for future work, the script can be easily adapted to retrieve data for additional assets or extend the time period, enabling further research and analysis in the financial domain.

3.1.3 Data Preprocessing and Log Returns

Data preprocessing and feature engineering are critical steps in preparing the dataset for modeling. They ensure that the data fed into the Gaussian Process Regression models are clean, consistent, and informative.

Use of Log Returns In this project, log returns are utilized for modeling asset price movements due to several compelling reasons that align with both theoretical and practical considerations in financial analysis.

Theoretical Foundation in Finance Log returns are integral to many foundational financial models, such as the Black-Scholes option pricing model, which assume that asset prices follow a log-normal distribution. By using log returns, we align our modeling approach with these theoretical frameworks, facilitating more accurate and consistent analyses. **GPR!** models assume normally distributed outputs, and since log returns of log-normally distributed prices are normally distributed, this compatibility enhances the effectiveness of our predictive modeling.

Stability Over Time Log returns exhibit greater stability compared to simple arithmetic returns, particularly in the presence of extreme outliers or during periods of high market volatility. They tend to smooth out spikes and reduce the impact of short-term noise, making the models less sensitive to sudden market anomalies. This stability is crucial for developing robust predictive models that can perform reliably under various market conditions.

Time Consistency (Additivity) One of the key mathematical properties of log returns is their additive nature over time. The total log return over a period is the sum of the log returns over sub-periods:

$$\log\left(\frac{S_t}{S_0}\right) = \log\left(\frac{S_t}{S_{t-1}}\right) + \log\left(\frac{S_{t-1}}{S_{t-2}}\right) + \cdots + \log\left(\frac{S_1}{S_0}\right), \tag{3.15}$$

where S_t is the asset price at time t. This additive property simplifies the computation of returns over arbitrary time horizons, such as weekly or monthly periods, by allowing us to sum daily log returns. It is particularly beneficial for forecasting and portfolio optimization over multi-day horizons, as it facilitates the aggregation of returns without the need for complex compounding calculations.

Normalization of Price Scale Log returns are scale-invariant, meaning they standardize returns across assets regardless of their price levels. Whether an asset is priced at \$1 or \$1,000, the log return brings their percentage changes onto a consistent scale. This normalization simplifies comparisons across assets with vastly different price levels and reduces the need for additional data scaling or normalization procedures. It ensures that no single asset disproportionately influences the model due to its absolute price, allowing for a more balanced and equitable analysis within the portfolio.

Conclusion By incorporating log returns into our modeling framework, we leverage their theoretical compatibility with financial models, enhance stability against market volatility, benefit from their time-additive properties, and achieve scale normalization across diverse assets. These advantages contribute to the robustness and accuracy of our Gaussian Process Regression models and improve the effectiveness of our dynamic portfolio optimization strategies.

Data Normalization and Scaling To bring all features onto a similar scale and improve the numerical stability of the models, we applied data normalization techniques. Specifically, we used min-max scaling to normalize the historical return features and the time index:

$$X_{\text{normalized}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}, \tag{3.16}$$

where X represents the original feature values, and X_{min} and X_{max} are the minimum and maximum values of the feature, respectively. This scaling transforms the data to a [0, 1] range, facilitating efficient model training.

Treatment of Missing Data and Outliers Financial time-series data often contain missing values and outliers due to market closures, data recording errors, or extreme market events. To address missing data, we employed interpolation methods appropriate for time-series, such as linear interpolation and forward/backward filling, ensuring temporal continuity in the data.

Outliers were identified using the Interquartile Range (IQR) method:

$$\text{IQR} = Q_3 - Q_1, \tag{3.17}$$

where Q_1 and Q_3 are the first and third quartiles, respectively. Data points lying outside 1.5 times the IQR from the quartiles were considered outliers. We assessed these outliers to determine whether they were due to data errors or genuine market anomalies. Genuine outliers representing significant market movements were retained to preserve the dataset's integrity, while erroneous data points were corrected or removed.

Handling Missing Data with GPR **GPR!** is particularly effective in addressing missing data and managing outliers due to its probabilistic framework and ability to model uncertainties. As demonstrated in the figure, GPR leverages the observed data points to predict missing values by constructing a distribution over possible functions that fit the data. As shown in **??** for each prediction, GPR provides both a mean estimate and a confidence interval that quantifies the uncertainty of the prediction. This feature allows it to interpolate missing data seamlessly while considering the underlying patterns and relationships in the time-series.

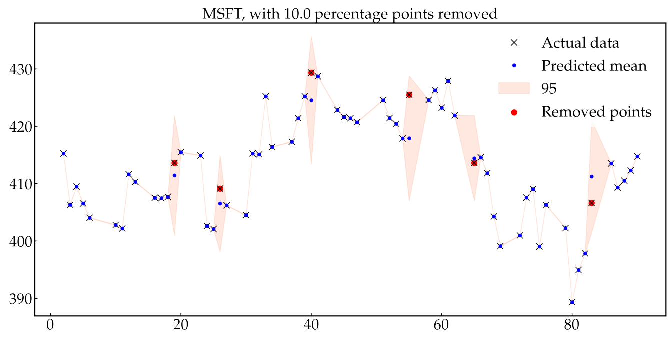

Figure 3.1: GPR Prediction with Missing Data and Outliers

Furthermore, GPR handles outliers by naturally weighting data points according to their fit within the modeled distribution. Outliers, which deviate significantly from the predicted mean and exhibit higher uncertainty, are less influential in the model's predictions. This property prevents extreme values from distorting the regression output, making GPR robust in noisy and incomplete datasets. The ability to quantify uncertainty not only aids in filling missing data but also provides a valuable tool for identifying potential anomalies in the dataset, as seen in the highlighted regions of the plot. By incorporating these capabilities, GPR ensures that the dataset remains coherent and usable for subsequent analysis, such as predictive modeling and portfolio optimization.

Data Splitting and Cross-Validation The dataset was divided into training and testing sets to evaluate the model's predictive performance. The training set consisted of the first 80% of the time period, while the remaining 20% was reserved for testing. This chronological split respects the temporal order of the data, avoiding look-ahead bias.

To further validate the models, we used time-series cross-validation with a rolling window approach. In each iteration, the model was trained on a window of consecutive data points and tested on the subsequent period. This method provides a more robust assessment of the model's performance over time and simulates real-world forecasting conditions.

Sliding Window Approach To denoise the time-series data and reduce short-term fluctuations, we employed a sliding window approach using a centered rolling window mechanism. For each data point in the series, a window of a specified size w was centered around it, and a statistical function was applied to the data within this window to compute a denoised value. The primary function used was the mean, though other functions could be applied as needed.

Formally, let $\{x_t\}_{t=1}^T$ represent the original time-series data, and $\{\tilde{x}_t\}_{t=1}^T$ denote the denoised series. The denoised value at time t, $\tilde{x}_t$, is calculated as:

$$\tilde{x}_t = \frac{1}{n_t} \sum_{i=t-k}^{t+k} x_i, \tag{3.18}$$

where $k = \left\lfloor \frac{w}{2} \right\rfloor$, and n_t is the number of data points within the window centered at time t. The window size w is an odd integer to ensure symmetry around the central point. At the edges of the time-series (when $t-k < 1$ or $t+k > T$), the window is adjusted by including available data points, and the minimum number of periods is set to 1 to allow computation even with incomplete windows.

To handle any missing values that may arise at the edges due to insufficient data points, we applied forward and backward filling methods. Forward filling propagates the last observed non-missing value forward to fill subsequent missing positions, while backward filling fills missing values by propagating the next observed non-missing value backward. These steps ensure that the denoised series is complete and free from missing values.

This sliding window denoising process effectively smooths the data by averaging over the local neighborhood of each data point, reducing random noise while preserving significant trends and patterns. By enhancing the signal-to-noise ratio in the time-series, this approach improves the quality of the input data for the Gaussian Process Regression models, leading to better predictive performance and more reliable portfolio optimization decisions.

Gaussian Filter Denoising Method To further enhance the quality of the time-series data and reduce high-frequency noise, we employed the Gaussian filter denoising method. The Gaussian filter is a convolutional filter that applies a Gaussian kernel to smooth data by averaging neighboring points with weights determined by the Gaussian function. This technique preserves significant trends and patterns while effectively attenuating random fluctuations and noise.

The Gaussian kernel is defined by the Gaussian (normal) distribution function:

$$G(i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{i^2}{2\sigma^2}\right), \tag{3.19}$$

where i is the index distance from the central data point, and σ is the standard deviation of the Gaussian distribution, controlling the degree of smoothing. A larger σ results in a wider kernel and more extensive smoothing.

The denoised value at time t, $\tilde{x}_t$, is computed by convolving the original time-series data x_t with the Gaussian kernel:

$$\tilde{x}_t = \sum_{i=-k}^{k} G(i) \cdot x_{t+i}, \tag{3.20}$$

where k is the window size parameter determining the range of data points considered around time t.

In our implementation, we applied the Gaussian filter to the closing prices of the assets using a standard deviation $\sigma = 1$, which provides a balanced smoothing effect without overly distorting the data. The following code snippet illustrates the application of the Gaussian filter using the `gaussian_filter` function from the SciPy library in Python:

```python
if isFiltered:
    df['filtered_close'] = gaussian_filter(df['close'], sigma=1)
```

In this code, `df['close']` represents the original closing price data, and `df['filtered_close']` stores the denoised data after applying the Gaussian filter. The `isFiltered` flag allows for conditional application of the filtering process.

By using the Gaussian filter denoising method, we effectively reduced the impact of noise and short-term fluctuations in the financial time-series data. This preprocessing step enhances the signal-to-noise ratio, allowing the Gaussian Process Regression models to focus on underlying market trends and improving the accuracy of the return predictions. The combination of the sliding window approach and Gaussian filtering provides a robust methodology for data smoothing, contributing significantly to the reliability and performance of the predictive modeling and subsequent portfolio optimization.

The filtered data retained essential market movements while reducing random fluctuations, allowing the Gaussian Process Regression models to focus on meaningful signals.

3.1.4 Summary

By carefully selecting assets, sourcing reliable data, and meticulously preprocessing the dataset, we established a solid foundation for our predictive modeling. The combination of normalization, outlier treatment, strategic data splitting, and denoising ensured that the inputs to our models were of high quality. These steps are crucial for enhancing the performance of the Gaussian Process Regression models and, ultimately, for developing effective portfolio optimization strategies.

3.2 Model Development

In this section, we outline the development of the predictive framework and evaluation methodology for forecasting asset returns and optimizing portfolio allocations, namely trading strategies. The focus is on the performance metrics used to assess model effectiveness, as well as the selection of a baseline model for comparison with the Gaussian Process Regression (GPR) approach.

3.2.1 Baseline Models

To establish a reference point for the performance of the Gaussian Process Regression model, we selected the ARIMA (AutoRegressive Integrated Moving Average) model as the baseline.

ARIMA! Model

The **ARIMA!** model is a widely used time-series forecasting technique that combines autoregressive (AR), differencing (I), and moving average (MA) components. It is defined by three parameters: p, d, and q, where:

- p is the order of the autoregressive term.
- d is the degree of differencing required to achieve stationarity.
- q is the order of the moving average term.

The **ARIMA!** model forecasts the value at time t as:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}, \tag{3.21}$$

where ϕ_i are the AR coefficients, θ_i are the MA coefficients, and ϵ_t is the white noise term.

Rationale for Baseline Selection

The **ARIMA!** model serves as an appropriate baseline because it is a well-established method for modeling time-series data and capturing trends and seasonality. While it is linear in nature, ARIMA provides a benchmark for assessing the added value of GPR, which can capture complex non-linear relationships and quantify uncertainty in predictions.

3.2.2 Comparative Evaluation

The performance of the **GPR!** model is compared against the **ARIMA!** baseline using the **MSE!** (**MSE!**) for predictive accuracy and the Sharpe Ratio for portfolio optimization outcomes. This dual evaluation approach ensures that the models are assessed comprehensively, both in terms of their forecasting ability and their impact on investment decisions. By demonstrating improvements over the **ARIMA!** model, the study highlights the effectiveness of **GPR!** in dynamic portfolio management.

3.2.3 Performance Metrics

To evaluate the performance of the predictive models and portfolio strategies, we employ the following metrics:

Mean Squared Error (MSE)

The Mean Squared Error (MSE) measures the accuracy of the model's predictions compared to the actual observed values. It is calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.22}$$

where y_i represents the actual value, $\hat{y}_i$ is the predicted value, and n is the total number of predictions. A lower MSE indicates better predictive accuracy. This metric is particularly useful for evaluating the performance of GPR and baseline models in forecasting asset returns.

Sharpe Ratio

The Sharpe Ratio assesses the risk-adjusted performance of a portfolio, measuring the excess return per unit of risk. It is defined as:

$$\text{Sharpe Ratio} = \frac{E[R_p] - R_f}{\sigma_p}, \tag{3.23}$$

where $E[R_p]$ is the expected portfolio return, R_f is the risk-free rate, and σ_p is the portfolio's standard deviation of returns. A higher Sharpe Ratio indicates a more favorable trade-off between risk and return, making it a crucial metric for evaluating the effectiveness of the portfolio optimization strategies derived from the predicted returns.

3.2.4 Multi-Input Gaussian Process Regression Model

In this study, we developed a multi-input Gaussian Process Regression (GPR) model that incorporates multiple economic factors and time as input dimensions to predict the returns of individual stocks. The model combines features from a diverse set of assets and economic indicators, including Brent Oil prices, the US Dollar Index (DXY), the S&P 500 Index, the Nasdaq 100 Index, Bitcoin (BTC), and gold prices (XAU/USD). This multi-input framework allows the model to capture intricate relationships between the target stock and broader market dynamics, enhancing its predictive power.

Input Data and Feature Selection

The input features for the GPR model were selected based on their correlation with the target stock. For each feature, we calculated the correlation between the feature's historical returns and the target stock's returns. Only features with an absolute correlation exceeding a predefined threshold were included in the model to ensure relevance and reduce noise in the predictions. This approach prioritizes the most influential economic factors while discarding irrelevant or weakly correlated inputs.

The final input matrix was constructed by concatenating the selected features along with time as an additional dimension. By including time, the model accounts for temporal trends and seasonality in the data, further improving its predictive capabilities.

Composite Kernel for Multi-Dimensional Inputs

To effectively model the multi-dimensional input space, we employed a composite kernel approach that assigns separate kernels to different input dimensions. Specifically:

- One kernel was applied to the economic factors, allowing the model to learn the specific relationships between the target stock and each factor. This kernel captures the varying scales and complexities of the economic data.
- Another kernel was applied to the time dimension, enabling the model to account for temporal dependencies and trends.

The composite kernel combines these two components multiplicatively, ensuring that the model can flexibly adjust to different input dimensions and set appropriate length scales for each. This design allows the GPR model to capture the unique characteristics of each feature while modeling their joint effects on the target stock's returns.

Training and Testing Framework

The training and testing process involved splitting the data into separate periods. The training set included data from the beginning of the time series up to a specified date, while the testing set encompassed the subsequent period. This split ensured that the model was evaluated on unseen data, simulating real-world forecasting conditions.

To further enhance the model's robustness, the noise variance in the GPR model was either fixed to a small value or optimized as a trainable parameter. This dual approach ensured flexibility in handling data with varying noise levels while preventing overfitting.

Prediction and Evaluation

Once the model was trained, it was used to predict the returns of the target stock over the testing period. The predictive mean and variance were computed for each test point, providing both point estimates and uncertainty bounds. The model's performance was evaluated using the Mean Squared Error (MSE), calculated for both normalized and denormalized data, to assess its accuracy in both relative and absolute terms.

By leveraging multi-input GPR with composite kernels, this approach captures the interplay between diverse economic factors, time, and stock returns. The flexibility in kernel design and inclusion of multi-dimensional inputs ensures that the model is both robust and adaptive, providing accurate predictions for individual stock performance in dynamic market environments.

3.2.5 Hyperparameter Tuning

Effective hyperparameter tuning is crucial for optimizing the performance of **GPR!** models, as it directly impacts the model's ability to fit the data and generalize to unseen samples. In this study, two distinct approaches were implemented for training **GPR!** models, differing in how the likelihood variance (noise variance) was treated during the optimization process. This subsection describes the methodologies and their respective advantages.

Fixed Likelihood Variance

In the first approach, the likelihood variance was fixed to a predefined value ($\sigma^2 = 10^{-3}$) and excluded from the training process. By keeping the noise variance constant, the optimization focuses solely on tuning the parameters of the kernel function, ensuring that the model does not overfit to noise in the data. This approach is computationally efficient and is particularly useful when the level of noise in the data is well-understood or consistent across the dataset. The training process involved minimizing the model's negative log marginal likelihood using the Scipy optimizer provided by `gpflow`:

$$\text{Negative Log Marginal Likelihood (NLML)} = -\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}), \tag{3.24}$$

where $\boldsymbol{\theta}$ represents the trainable kernel parameters.

Trainable Likelihood Variance

In the second approach, the likelihood variance was set as a trainable parameter, allowing the model to adaptively learn the optimal noise level from the data. This method is advantageous when the data contains varying levels of noise or when the noise characteristics are not well-understood. To enhance the robustness of this approach, multiple starting values for the likelihood variance were used (10^{-5}, 10^{-3}, 10^{-1}, 1.0), ensuring that the optimization process explored a diverse range of initial configurations. For each starting variance, the model was trained using the Scipy optimizer, and the final negative log marginal likelihood was evaluated.

- The model with the lowest final loss was selected as the best-performing model.
- This approach balances the trade-off between model flexibility and the risk of overfitting by optimizing both the kernel parameters and the likelihood variance.

Comparison of Approaches

Both approaches were evaluated based on their ability to minimize the negative log marginal likelihood and the predictive performance on the test dataset. The fixed likelihood variance method provided computational simplicity and reduced the risk of overfitting in scenarios with stable noise levels. Conversely, the trainable likelihood variance approach demonstrated higher adaptability, allowing the model to account for varying noise levels and capture the data's inherent uncertainty more effectively.

Best Model Selection

The trainable likelihood variance approach, with multiple initializations, identified the model configuration that minimized the negative log marginal likelihood. This model provided a balance between accurately capturing the signal and accounting for noise, making it the preferred choice for the Gaussian Process Regression framework used in this study.

3.3 Forecasting Approach

Describe the iterative forecasting method and how predictions are updated daily.

3.3.1 Iterative Retraining for Sequential Predictions

In addition to batch training and prediction, we implemented an iterative retraining approach for the Gaussian Process Regression (GPR) model. This method reflects a real-world scenario where new market data becomes available daily, requiring the model to be updated frequently to make accurate short-term predictions. In this approach, the GPR model is retrained daily using all available data up to the current day, and predictions are made for the next day. This process is repeated iteratively for five consecutive days to generate predictions for a five-day horizon.

Methodology

The iterative retraining approach involves the following steps:

1. **Daily Data Integration:** At the start of each iteration, the most recent daily market data is fetched and added to the training dataset. This includes the latest values of the target stock and selected economic factors.
2. **Training with Updated Data:** The GPR model is retrained using the expanded dataset, incorporating all historical data up to the current day. Both fixed and trainable likelihood variance configurations are evaluated to ensure robust performance.
3. **Prediction for the Next Day:** Using the retrained model, predictions are made for the target stock's return for the next day. The predictive mean and variance are recorded for each iteration.
4. **Iterative Execution:** Steps 1 through 3 are repeated iteratively, with each new day's data being integrated into the model, until predictions are generated for all five future days.

Advantages of Iterative Retraining

The iterative retraining approach offers several advantages:

- **Real-Time Adaptation:** By retraining the model daily, it continuously adapts to the latest market conditions, ensuring that predictions remain relevant and responsive to recent trends.

- **Enhanced Short-Term Accuracy:** The use of updated data allows the model to account for the most recent market dynamics, improving the accuracy of short-term forecasts.
- **Uncertainty Quantification:** With each iteration, the GPR model provides a predictive mean and variance for the next day, enabling the assessment of uncertainty in the predictions. This is particularly useful for risk-aware decision-making in portfolio management.

Performance Evaluation

To evaluate the effectiveness of this approach, the Mean Squared Error (MSE) was calculated for both normalized and denormalized predicted returns across the five days. The iterative predictions were also compared against actual returns to assess the model's ability to track short-term market movements. Additionally, the cumulative performance over the five-day horizon was analyzed to understand the overall impact of daily updates on predictive accuracy.

Practical Application

This iterative retraining approach closely mimics real-world trading scenarios where investors rely on daily market updates to make decisions. By retraining the model with the latest data and predicting only one day ahead, this method ensures that the predictions are both timely and aligned with current market conditions. This sequential prediction framework provides a robust foundation for dynamic portfolio management, enabling more accurate and adaptive asset allocation strategies.

3.4 Portfolio Optimization Strategies

3.4.1 Traditional Strategies

Maximum Return Strategy Formulation

The maximum return strategy aims to maximize the expected return of a portfolio while considering risk constraints and incorporating regularization penalties such as L1/L2 norms and transaction costs. The optimization problem is formulated based on the given code snippet and can be expressed mathematically as follows.

Let:

- $\mathbf{w} \in \mathbb{R}^n$ be the vector of portfolio weights for n assets.

- $\boldsymbol{\mu} \in \mathbb{R}^n$ be the vector of expected returns for each asset.
- $\Sigma \in \mathbb{R}^{n \times n}$ be the covariance matrix of asset returns.
- $\sigma_{\max}$ be the maximum allowable portfolio volatility.
- $\mathcal{R}(\mathbf{w})$ be the regularization penalty function, which may include L1/L2 penalties and transaction costs.

The optimization problem is:

$$\begin{aligned} \min_{\mathbf{w}} \quad & -\mathbf{w}^\top \boldsymbol{\mu} + \mathcal{R}(\mathbf{w}) \\ \text{subject to} \quad & \sqrt{\mathbf{w}^\top \Sigma \mathbf{w}} \leq \sigma_{\max} \\ & \sum_{i=1}^{n} w_i = 1 \\ & w_i \geq 0, \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

Objective Function The objective function consists of two components:

1. **Negative Expected Return**: $-\mathbf{w}^\top \boldsymbol{\mu}$
2. **Regularization Penalty**: $\mathcal{R}(\mathbf{w})$

By minimizing the negative expected return, we effectively maximize the portfolio's expected return. The regularization penalty $\mathcal{R}(\mathbf{w})$ accounts for additional considerations such as promoting sparsity (L1 regularization), preventing overfitting (L2 regularization), and minimizing transaction costs.

Constraints The constraints ensure that:

- **Volatility Constraint**: The portfolio's volatility does not exceed $\sigma_{\max}$:

$$\sqrt{\mathbf{w}^\top \Sigma \mathbf{w}} \leq \sigma_{\max}$$

- **Full Investment Constraint**: The sum of the portfolio weights equals 1:

$$\sum_{i=1}^{n} w_i = 1$$

- **No Short-Selling Constraint**: All portfolio weights are non-negative:

$$w_i \geq 0, \quad \forall i$$

Regularization Penalty The regularization penalty $\mathcal{R}(\mathbf{w})$ can be detailed as:

$$\mathcal{R}(\mathbf{w}) = \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^{n} |w_i - w_i^{\text{prev}}|$$

where:

- $\|\mathbf{w}\|_1 = \sum_{i=1}^{n} |w_i|$ is the L1 norm, promoting sparsity in the portfolio weights.
- $\|\mathbf{w}\|_2^2 = \sum_{i=1}^{n} w_i^2$ is the squared L2 norm, preventing extreme weight allocations.
- w_i^{prev} is the previous weight of asset i, accounting for transaction costs when adjusting the portfolio.
- $\lambda_1, \lambda_2, \gamma \geq 0$ are hyperparameters controlling the influence of each penalty component.

Optimization Method The optimization problem is solved using the Sequential Least Squares Programming (SLSQP) algorithm, which is suitable for constrained nonlinear optimization. The method minimizes the objective function while satisfying the specified constraints.

$$\text{Result} = \arg\min_{\mathbf{w}} \left(-\mathbf{w}^\top \boldsymbol{\mu} + \mathcal{R}(\mathbf{w}) \right)$$

$$\text{subject to constraints}$$

Implementation Notes In our implementation:

- The function `returns_objective` computes the objective function.
- The `maximize_returns` method sets up the optimization problem, including the volatility constraint and any additional constraints.
- Regularization and transaction costs are incorporated through the `total_penalty` function.

By formulating the strategy in this manner, the model seeks the optimal balance between maximizing returns and controlling for risk and costs, resulting in a robust and practical portfolio optimization solution.

Maximum Sharpe Ratio Optimization

The maximum Sharpe ratio optimization aims to maximize the portfolio's Sharpe ratio, which is a measure of risk-adjusted return. This involves finding the optimal portfolio weights that maximize the expected return minus the risk-free rate, divided by the portfolio's volatility, while also incorporating regularization penalties such as L1/L2 norms and transaction costs.

Let:

- $\mathbf{w} \in \mathbb{R}^n$ be the vector of portfolio weights for n assets.
- $\boldsymbol{\mu} \in \mathbb{R}^n$ be the vector of expected returns for each asset.
- $\Sigma \in \mathbb{R}^{n \times n}$ be the covariance matrix of asset returns.
- r_f be the risk-free rate.
- $\mathcal{R}(\mathbf{w})$ be the regularization penalty function, which may include L1/L2 penalties and transaction costs.

The Sharpe ratio $S(\mathbf{w})$ is defined as:

$$S(\mathbf{w}) = \frac{\mathbf{w}^\top \boldsymbol{\mu} - r_f}{\sqrt{\mathbf{w}^\top \Sigma \mathbf{w}}}$$

The optimization problem is:

$$\begin{aligned} \max_{\mathbf{w}} \quad & S(\mathbf{w}) - \mathcal{R}(\mathbf{w}) \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i = 1 \\ & w_i \geq 0, \quad \forall i \in \{1, \ldots, n\} \end{aligned}$$

Since optimization algorithms typically perform minimization, we can reformulate the problem as minimizing the negative Sharpe ratio plus the regularization penalty:

$$\begin{aligned} \min_{\mathbf{w}} \quad & -\frac{\mathbf{w}^\top \boldsymbol{\mu} - r_f}{\sqrt{\mathbf{w}^\top \Sigma \mathbf{w}}} + \mathcal{R}(\mathbf{w}) \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i = 1 \\ & w_i \geq 0, \quad \forall i \end{aligned}$$

Objective Function The objective function consists of:

1. **Negative Sharpe Ratio**: $-\frac{\mathbf{w}^\top \boldsymbol{\mu} - r_f}{\sqrt{\mathbf{w}^\top \Sigma \mathbf{w}}}$
2. **Regularization Penalty**: $\mathcal{R}(\mathbf{w})$

By minimizing the negative Sharpe ratio, we effectively maximize the Sharpe ratio, thus maximizing the portfolio's risk-adjusted return.

Constraints The constraints ensure that:

- **Full Investment Constraint**: The sum of the portfolio weights equals 1:

$$\sum_{i=1}^{n} w_i = 1$$

- **No Short-Selling Constraint**: All portfolio weights are non-negative:

$$w_i \geq 0, \quad \forall i$$

Regularization Penalty The regularization penalty $\mathcal{R}(\mathbf{w})$ includes both L1/L2 penalties and transaction costs, and can be expressed as:

$$\mathcal{R}(\mathbf{w}) = \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^{n} |w_i - w_i^{\text{prev}}|$$

where:

- $\|\mathbf{w}\|_1 = \sum_{i=1}^{n} |w_i|$ is the L1 norm, promoting sparsity in the portfolio weights.
- $\|\mathbf{w}\|_2^2 = \sum_{i=1}^{n} w_i^2$ is the squared L2 norm, preventing extreme weight allocations.
- w_i^{prev} is the previous weight of asset i, accounting for transaction costs when adjusting the portfolio.
- $\lambda_1, \lambda_2, \gamma \geq 0$ are hyperparameters controlling the influence of each penalty component.

Optimization Method The optimization problem is solved using the Sequential Least Squares Programming (SLSQP) algorithm, which is suitable for constrained nonlinear optimization. The method minimizes the objective function while satisfying the specified constraints:

$$\text{Result} = \arg\min_{\mathbf{w}} \left(-\frac{\mathbf{w}^\top \boldsymbol{\mu} - r_f}{\sqrt{\mathbf{w}^\top \Sigma \mathbf{w}}} + \mathcal{R}(\mathbf{w}) \right)$$

$$\text{subject to constraints}$$

Implementation Notes In our implementation:

- The function `objective` computes the negative Sharpe ratio plus regularization penalties.
- The `optimize_portfolio` method sets up the optimization problem, including the necessary constraints.
- Regularization and transaction costs are incorporated through the `total_penalty` function.

By formulating the strategy in this manner, the model seeks to maximize the portfolio's risk-adjusted return while controlling for risk and costs, resulting in an efficient and practical portfolio optimization solution.

3.4.2 Dynamic Strategy

In this project, we specifically designed a dynamic strategy, that is to decide at each time point which portfolio optimization strategy to use based on the predicted market conditions. The dynamic strategy involves two main components: the regime detection model and the strategy selection mechanism. For the regime detection model, we employed a Probability Estimation Model and a Expected Value Comparison Method to identify distinct regimes based on the predicted returns and uncertainties.

For the strategy selection mechanism, we developed a Decision Rule that determines the optimal portfolio optimization strategy based on the detected regime. The dynamic strategy aims to adapt to changing market conditions and optimize portfolio allocations accordingly, enhancing the robustness and performance of the investment approach.

Probability Estimation: $P(S_1 > S_2)$

When estimating the probability $P(S_1 > S_2)$ for two random variables S_1 and S_2, the methodology depends on the nature of their distributions and their dependence structure. This section outlines three approaches: numerical integration, Monte Carlo simulation, and the use of copulas for dependent variables.

Numerical Integration If the probability density functions (PDFs) of S_1 and S_2 are known, the probability $P(S_1 > S_2)$ can be expressed as:

$$P(S_1 > S_2) = \int_{-\infty}^{\infty} \int_{y}^{\infty} f_{S_1}(x) f_{S_2}(y) \, dx \, dy, \tag{3.25}$$

where:

- $f_{S_1}(x)$ is the PDF of S_1,
- $f_{S_2}(y)$ is the PDF of S_2.

This double integral represents the joint probability over the region where $S_1 > S_2$, and it requires numerical methods for evaluation when closed-form solutions are unavailable.

Copulas for Dependence Especially, When S_1 and S_2 are dependent, the joint distribution can be modeled using a copula. A copula is a function that describes the dependence structure between random variables, linking their marginal distributions. Let $F_{S_1}(x)$ and $F_{S_2}(y)$ represent the cumulative distribution functions (CDFs) of S_1 and S_2, respectively. The joint CDF can be expressed as:

$$F_{S_1,S_2}(x,y) = C(F_{S_1}(x), F_{S_2}(y)), \tag{3.26}$$

where $C(u,v)$ is the copula function.

The probability $P(S_1 > S_2)$ can then be computed as:

$$P(S_1 > S_2) = \int_{-\infty}^{\infty} \int_{y}^{\infty} \frac{\partial^2 C(F_{S_1}(x), F_{S_2}(y))}{\partial u \partial v} \, dx \, dy. \tag{3.27}$$

The steps to compute this are:

1. Determine the marginal distributions $F_{S_1}(x)$ and $F_{S_2}(y)$ for S_1 and S_2.
2. Select an appropriate copula function $C(u,v)$ based on the dependence structure (e.g., Gaussian, Clayton, or Gumbel copulas).

3. Use numerical methods to evaluate the double integral above.

Copulas are particularly effective when the marginal distributions are non-normal or when the dependence structure is non-linear and cannot be captured by simple correlation measures.

Monte Carlo Methods If the distributions of S_1 and S_2 are not explicitly known but sampling from these distributions is possible, a Monte Carlo simulation can be used to estimate $P(S_1 > S_2)$. The steps are as follows:

1. Generate N independent samples $S_1^{(i)}$ and $S_2^{(i)}$ from the respective distributions of S_1 and S_2.
2. Count the number of instances where $S_1^{(i)} > S_2^{(i)}$. Denote this count by n.
3. Estimate the probability as:

$$P(S_1 > S_2) \approx \frac{n}{N}, \tag{3.28}$$

where

$$n = \sum_{i=1}^{N} \mathbb{I}(S_1^{(i)} > S_2^{(i)}), \tag{3.29}$$

and $\mathbb{I}(\cdot)$ is the indicator function, which equals 1 if the condition is true and 0 otherwise.

Monte Carlo simulation is particularly useful for complex distributions or dependent variables, where analytical integration is impractical. In our case, we have multiple assets with potentially non-normal distributions and complex dependencies, making Monte Carlo methods a valuable tool for estimating $P(S_1 > S_2)$. Specifically, we will use Monte Carlo simulation to estimate the probability of one asset outperforming another in our portfolio optimization strategies. And we chose a sample size of $N = 10,000$ to ensure accurate probability estimates.

Comparison of Methods

- **Numerical Integration:** Provides an exact solution given the PDFs of S_1 and S_2, but computationally intensive for high-dimensional problems or non-standard distributions.
- **Copulas:** Allows modeling of complex dependence structures, particularly useful for non-normal distributions or asymmetric dependencies.

- **Monte Carlo Simulation:** Flexible and practical alternative when sampling is straightforward, though its accuracy depends on the number of samples N.

Each method has its strengths and limitations, given the availability of distributional information and computational resources of our case, we chose to use Monte Carlo simulation for estimating $P(S_1 > S_2)$.

Strategy switching criteria Describe the criteria for switching between portfolio optimization strategies based on the estimated probability $P(S_1 > S_2)$. We set a threshold probability τ such that if $P(S_1 > S_2) > \tau$, the strategy with the higher expected return is selected, and if $P(S_1 > S_2) \leq \tau$, the strategy with the lower volatility is chosen. This threshold ensures that the strategy selection is based on a balance between return and risk, incorporating the estimated probability of one asset outperforming the other.

Expected Value Comparison

The Expected Value Comparison method involves comparing the expected values of two time points distribution to determine the optimal choice. This approach is based on the principle of maximizing the expected return or Sharpe ratio while considering the estimated probabilities of each strategy's performance. The Expected Value Comparison method is particularly useful for dynamic portfolio optimization, where the selection of strategies is based on their expected outcomes under different market conditions.

The dynamic strategy employs a decision-making process based on the comparison of expected portfolio returns using previous and current predictions. The core idea is to adaptively switch between maximizing returns and minimizing volatility (uncertainty) depending on the anticipated changes in the market. The strategy aims to optimize the portfolio by considering not only the expected returns but also transaction costs associated with rebalancing.

Let:

- $\boldsymbol{\mu}_A \in \mathbb{R}^n$ be the vector of expected returns at time $t-1$.
- $\boldsymbol{\mu}_B \in \mathbb{R}^n$ be the vector of expected returns at time t.
- $\mathbf{w}_{\text{prev}} \in \mathbb{R}^n$ be the portfolio weights optimized at time $t-1$.
- λ_{tx} be the transaction cost rate.

Decision Logic The strategy proceeds as follows:

1. **Initialization**: For the first time step ($t = 0$), where no previous weights exist, the strategy allocates weights by maximizing returns under a volatility constraint:

   $$\mathbf{w}_0 = \arg\min_{\mathbf{w}} \left(-\boldsymbol{\mu}_B^\top \mathbf{w} + \mathcal{R}(\mathbf{w}) \right)$$

   subject to the constraints specified in the maximum return strategy.

2. **Expected Returns Calculation**: For $t \geq 1$, calculate the expected portfolio returns at times $t-1$ and t using the previous optimal weights $\mathbf{w}_{\text{prev}}$:

   $$R_A = \boldsymbol{\mu}_A^\top \mathbf{w}_{\text{prev}}$$
   $$R_B = \boldsymbol{\mu}_B^\top \mathbf{w}_{\text{prev}}$$

3. **Comparison of Expected Returns**:
   - If $R_A > R_B$ (the expected return is decreasing), it suggests that the current allocation may yield lower returns in the next period. Therefore, the strategy opts to **maximize returns** by solving:

     $$\mathbf{w}_t = \arg\min_{\mathbf{w}} \left(-\boldsymbol{\mu}_B^\top \mathbf{w} + \mathcal{R}(\mathbf{w}) \right)$$

     subject to the constraints specified in the maximum return strategy.
   - If $R_A \leq R_B$ (the expected return is increasing), the strategy becomes more conservative and chooses to **minimize volatility** by solving:

     $$\mathbf{w}_t = \arg\min_{\mathbf{w}} \left(\sqrt{\mathbf{w}^\top \Sigma_B \mathbf{w}} + \mathcal{R}(\mathbf{w}) \right)$$

     subject to the constraints specified in the minimum volatility approach, where Σ_B is the covariance matrix at time t.

4. **Transaction Cost Adjustment**: Compute the transaction costs incurred due to rebalancing:

   $$\text{TransactionCost} = \lambda_{\text{tx}} \sum_{i=1}^{n} |w_{i,t} - w_{i,t-1}|$$

5. **Realized Return Calculation**: Calculate the realized change in expected return after accounting for transaction costs:

$$\Delta R_{\text{realized}} = (R_B - R_A) - \text{TransactionCost}$$

6. **Strategy Adjustment**:
   - If $\Delta R_{\text{realized}} > 0$, the expected improvement in return outweighs the transaction costs, and the new weights $\mathbf{w}_t$ are adopted.
   - If $\Delta R_{\text{realized}} \leq 0$, the transaction costs negate the benefits of rebalancing, and the strategy **reverts to the previous weights**:

$$\mathbf{w}_t = \mathbf{w}_{\text{prev}}$$

Rationale The strategy's decision criteria are designed to balance the trade-off between expected returns and transaction costs:

- By comparing R_A and R_B, the strategy assesses whether the expected return is improving or deteriorating.
- Maximizing returns when the expected return is decreasing attempts to counteract the anticipated decline by seeking higher-return allocations.
- Minimizing volatility when the expected return is increasing adopts a conservative stance to preserve gains while benefiting from favorable market conditions.
- Incorporating transaction costs ensures that rebalancing decisions are economically justified, preventing unnecessary trading that could erode returns.

Implementation Notes

- **Covariance Matrices**: The covariance matrices Σ_A and Σ_B are used in volatility calculations and are estimated based on predicted volatilities and correlations at times $t-1$ and t, respectively.
- **Regularization Penalty**: The regularization function $\mathcal{R}(\mathbf{w})$ includes L1/L2 penalties and transaction costs as detailed in previous sections.
- **Constraints**: The optimization problems are subject to standard portfolio constraints such as full investment and no short-selling.

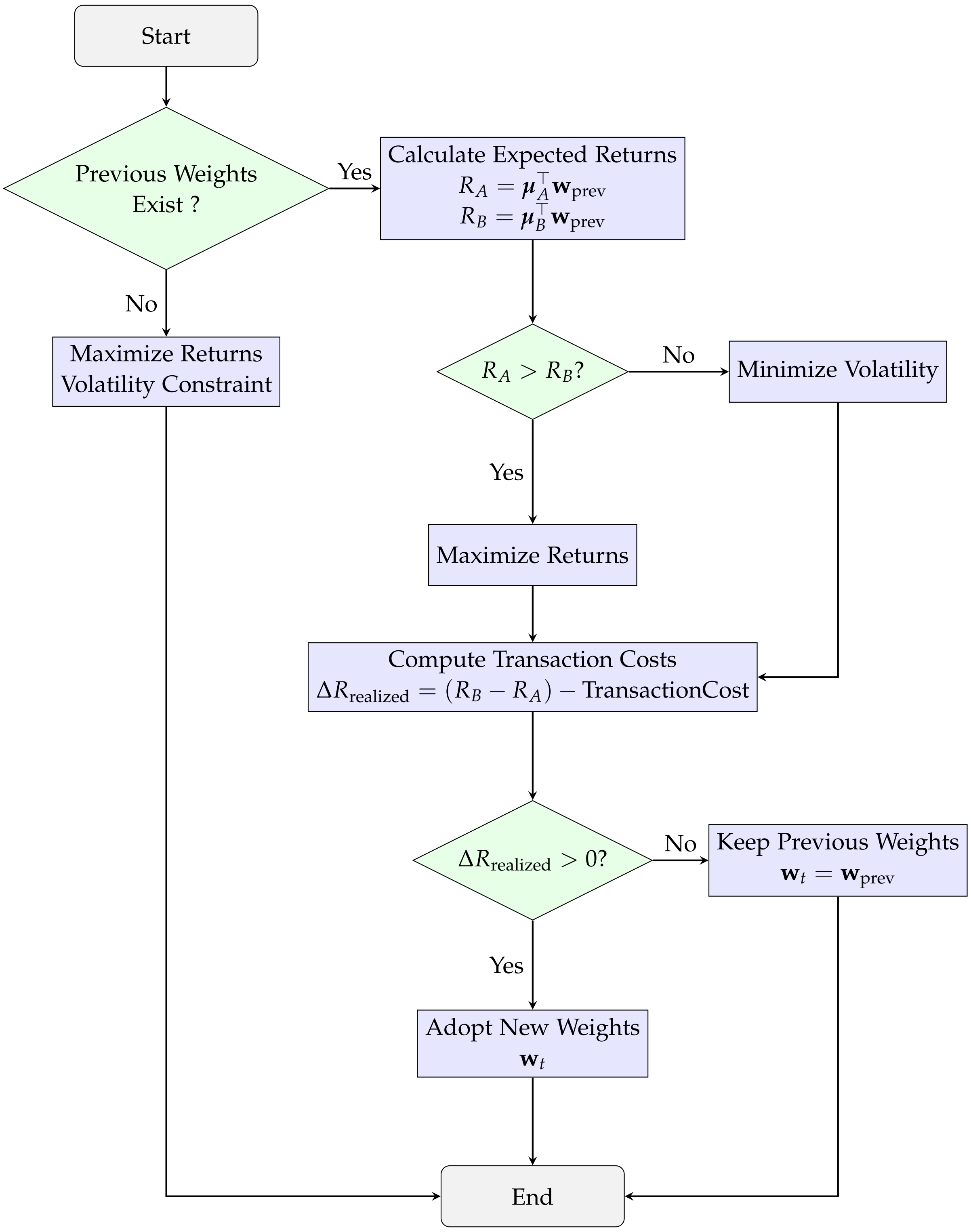
Start
Previous Weights Exist ?
Yes
No
Calculate Expected Returns
$R_A = \boldsymbol{\mu}_A^\top \mathbf{w}_{\text{prev}}$
$R_B = \boldsymbol{\mu}_B^\top \mathbf{w}_{\text{prev}}$
Maximize Returns Volatility Constraint
$R_A > R_B$?
No
Minimize Volatility
Yes
Maximize Returns
Compute Transaction Costs
$\Delta R_{\text{realized}} = (R_B - R_A) - \text{TransactionCost}$
$\Delta R_{\text{realized}} > 0$?
No
Keep Previous Weights
$\mathbf{w}_t = \mathbf{w}_{\text{prev}}$
Yes
Adopt New Weights
$\mathbf{w}_t$
End

Figure 3.2: Decision Flowchart for the Dynamic Strategy

- **Adaptability**: This strategy dynamically adjusts its optimization objective based on market expectations, aiming to enhance returns while controlling risk and costs.

Conclusion The dynamic strategy effectively combines elements of both the maximum return and minimum volatility strategies, switching between them based on expected changes in returns and considering transaction costs. By doing so, it aims to optimize the portfolio's performance in varying market conditions while maintaining a balance between risk and return.

3.5 Backtesting Framework

In this section, we describe the backtesting framework used to evaluate the performance of the portfolio optimization strategies. The backtesting process involves simulating the optimized portfolios over historical data to assess their effectiveness in real-world scenarios. The framework utilizes two primary classes, `Return` and `Volatility`, to calculate portfolio returns and volatilities, respectively.

3.5.1 Overview of the Backtesting Process

The backtesting process involves the following steps:

1. **Data Preparation**: Historical asset returns and predicted volatilities are collected for the assets under consideration.
2. **Portfolio Optimization**: The optimization strategies (e.g., maximum Sharpe ratio, maximum return, minimum volatility) are applied to generate optimal portfolio weights.
3. **Return and Volatility Calculation**: The `Return` and `Volatility` classes compute the daily portfolio returns, cumulative returns, transaction costs, and portfolio volatilities.
4. **Performance Evaluation**: Key performance metrics such as cumulative return, cumulative transaction costs, portfolio variance, and Sharpe ratio are calculated to evaluate the strategies.

3.5.2 Return Calculation

Return Class Implementation

The `Return` class is designed to calculate portfolio returns based on asset returns and portfolio weights, accounting for transaction costs. It is initialized with:

- `asset_returns`: A NumPy array of asset returns.
- `weights`: A NumPy array of portfolio weights, where each row corresponds to a specific time period.
- `transaction_cost_rate`: A scalar representing the transaction cost rate (default is 0).

The class provides methods to calculate:

- **Daily Portfolio Returns**: Computes the net portfolio returns for each day, considering transaction costs.
- **Cumulative Return**: Calculates the cumulative return of the portfolio over the backtesting period.
- **Cumulative Transaction Costs**: Sums up the transaction costs incurred over the backtesting period.
- **Daily Transaction Costs**: Provides the transaction costs for each day.

Daily Portfolio Return Calculation The daily portfolio return $r_{\text{net},t}$ on day t is calculated as:

$$r_{\text{net},t} = \mathbf{w}_t^\top \mathbf{r}_t - \text{Transaction Cost}_t$$

where:

- $\mathbf{w}_t$ is the portfolio weight vector at time t.
- $\mathbf{r}_t$ is the asset return vector at time t.
- $\text{Transaction Cost}_t$ is the transaction cost incurred on day t, calculated as:

$$\text{Transaction Cost}_t = \lambda_{\text{tx}} \sum_{i=1}^{n} |w_{i,t} - w_{i,t-1}|$$

  with λ_{tx} being the transaction cost rate, and $w_{i,t}$ being the weight of asset i at time t.

Cumulative Return Calculation The cumulative return over T days is calculated as:

$$\text{Cumulative Return} = \prod_{t=1}^{T}(1 + r_{\text{net},t}) - 1$$

3.5.3 Volatility Calculation

Volatility Class Implementation

The `Volatility` class calculates the portfolio volatility using predicted asset volatilities and portfolio weights. It is initialized with:

- `predicted_volatilities`: A DataFrame of predicted volatilities for each asset.
- `weights_df`: A DataFrame of portfolio weights for each time period.

Portfolio Volatility Calculation The portfolio volatility $\sigma_{\text{portfolio},t}$ at time t is calculated as:

$$\sigma_{\text{portfolio},t} = \sqrt{\sum_{i=1}^{n} w_{i,t}^2 \sigma_{i,t}^2}$$

where:

- $w_{i,t}$ is the weight of asset i at time t.
- $\sigma_{i,t}$ is the predicted volatility of asset i at time t.

3.5.4 Backtesting Procedure

The backtesting is implemented in the `backtest_portfolio` method, which performs the following steps:

1. **Initialization**: The method takes historical returns (`historical_returns`), predicted volatilities (`predicted_volatilities`), and the optimized portfolio weights (`optimal_weights`) as inputs.
2. **Return and Volatility Calculators**: Instances of the `Return` and `Volatility` classes are created using the historical data and optimized weights.
3. **Daily Calculations**:

- **Portfolio Returns**: The `calculate_portfolio_returns` method computes the daily net portfolio returns and transaction costs.
- **Portfolio Volatility**: The `calculate_portfolio_volatility` method computes the daily portfolio volatility.
- **Sharpe Ratio**: The daily Sharpe ratio is calculated as:

$$\text{Sharpe Ratio}_t = \frac{r_{\text{net},t} - r_f}{\sigma_{\text{portfolio},t}}$$

  where r_f is the risk-free rate.

4. **Cumulative Metrics**:
   - **Cumulative Return**: Calculated using the `calculate_cumulative_return` method.
   - **Cumulative Transaction Costs**: Summed over the backtesting period using the `calculate_cumulative_transaction_costs` method.
   - **Cumulative Variance**: Sum of daily portfolio variances.
   - **Overall Sharpe Ratio**: Calculated using cumulative return and cumulative variance:

$$\text{Overall Sharpe Ratio} = \frac{\text{Cumulative Return} - r_f}{\text{Cumulative Variance}}$$

5. **Results Presentation**: The method prints out daily and cumulative metrics for analysis.

3.5.5 Performance Evaluation

The strategies are evaluated based on the following key performance metrics:

- **Cumulative Return**: Indicates the total return generated by the portfolio over the backtesting period.
- **Cumulative Transaction Costs**: Reflects the total costs incurred due to portfolio rebalancing.
- **Cumulative Variance**: Measures the total risk taken over the period.
- **Sharpe Ratio**: Provides a risk-adjusted return metric, indicating how much excess return is received for the extra volatility endured.

Cumulative Return

Calculated as:

$$\text{Cumulative Return} = \prod_{t=1}^{T}(1 + r_{\text{net},t}) - 1$$

This metric helps in comparing the total profitability of different strategies over the same period.

Cumulative Transaction Costs

Calculated as:

$$\text{Cumulative Transaction Costs} = \sum_{t=1}^{T} \text{Transaction Cost}_t$$

This metric is crucial for understanding the impact of trading activity on net returns.

Sharpe Ratio

Calculated as:

$$\text{Sharpe Ratio} = \frac{\text{Cumulative Return} - r_f}{\sqrt{\sum_{t=1}^{T} \sigma^2_{\text{portfolio},t}}}$$

A higher Sharpe ratio indicates a more attractive risk-adjusted return.

3.5.6 Backtesting Results Interpretation

By analyzing the cumulative returns, transaction costs, and Sharpe ratios, we can assess the effectiveness of each optimization strategy. Strategies that yield higher cumulative returns with lower transaction costs and higher Sharpe ratios are considered more efficient.

The backtesting framework allows us to simulate real-world trading conditions, including the impact of transaction costs, and provides a comprehensive evaluation of the portfolio optimization strategies.

3.5.7 Implementation Notes

- **Transaction Costs**: Incorporated in the return calculations to simulate realistic trading scenarios.

- **Time-Varying Weights**: The framework supports dynamic rebalancing, with portfolio weights potentially changing each day.
- **Debugging and Logging**: Optional print statements in the code provide detailed insights into the daily performance metrics, aiding in debugging and analysis.

3.5.8 Conclusion

The backtesting framework is a critical component in validating the portfolio optimization strategies. By accurately simulating the investment process over historical data, accounting for transaction costs, and computing relevant performance metrics, the framework provides valuable insights into the potential real-world performance of the strategies under consideration.

4 Results and Analysis

4.1 Model Performance Analysis

4.1.1 Comparison with Benchmark Models

In this section, we compare the performance of our Gaussian Process Regression (GPR) models with a benchmark model, specifically the Autoregressive Integrated Moving Average (ARIMA) model. The comparison is based on the Mean Squared Error (MSE) of predictions over different forecast horizons.

- **ARIMA Model**: A traditional time series forecasting model that uses past values and past errors to predict future values.
- **Market Efficiency**: Assumes that markets are efficient and all investors have access to the same information.
- **Normal Distribution of Asset Returns**: Assumes that asset returns are normally distributed and can be described by their mean (expected return) and variance (risk).

Comparison with ARIMA We evaluated the predictive performance of the GPR and ARIMA models over different forecast horizons (5, 10, 13, 20, and 26 days). The Mean Squared Error (MSE) was used as the evaluation metric. The results are summarized in Table **??**.

Table 4.1: Mean Squared Error (MSE) Comparison between GPR and ARIMA Models

Forecast Horizon (Days)	ARIMA MSE	GPR MSE	Best Model
5	34.8847	15.8594	GPR
10	45.7229	39.1529	GPR
13	47.0780	58.2299	ARIMA
20	78.4795	98.9632	ARIMA
26	267.8519	195.5078	GPR

As shown in Table **??**, the GPR model outperforms the ARIMA model for shorter forecast horizons (5 and 10 days), exhibiting lower MSE values. However, for medium forecast horizons (13 and 20 days), the ARIMA model shows better performance. For the longest horizon (26 days), the GPR model again demonstrates a lower MSE compared to ARIMA.

Visual Comparison Figure **??** illustrates the predicted values from both GPR and ARIMA models against the actual values over the forecast horizons.

`prediction_comparison.png`

Figure 4.1: Comparison of Predicted Values from GPR and ARIMA Models

In the figure, the actual values are plotted alongside the predictions from both models. The visual comparison allows us to observe the accuracy of each model over different time horizons.

Discussion

The GPR model, with its non-parametric nature and ability to capture complex patterns, performs better for shorter forecast horizons. This suggests that GPR can effectively model short-term dependencies in the data. On the other hand, the ARIMA model, being a parametric model that relies on past values and errors, shows better performance for medium-term forecasts.

The fluctuation in performance across different forecast horizons indicates that no single model consistently outperforms the other. Therefore, the choice of model may depend on the specific forecasting requirements, such as the desired forecast horizon and the nature of the asset's return dynamics.

MSE Analysis

The Mean Squared Error (MSE) is calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where:

- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.
- n is the number of observations.

Lower MSE values indicate better predictive accuracy. The results show that for short-term predictions (up to 10 days), the GPR model significantly outperforms the ARIMA model. However, as the forecast horizon extends, the performance gap narrows, and ARIMA performs better at certain points.

Conclusion

The comparison highlights the strengths and limitations of both models. GPR excels in capturing short-term fluctuations due to its flexibility and non-parametric nature, making it suitable for short-term forecasting. ARIMA, with its reliance on historical patterns and residuals, may be more stable for medium-term forecasts. Selecting the appropriate model depends on the specific investment horizon and the characteristics of the asset being analyzed.

4.1.2 Analysis of prediction intervals

4.1.3 Model robustness and generalization

4.2 Portfolio Optimization Outcomes

4.2.1 Strategy Performance Comparison

Return analysis

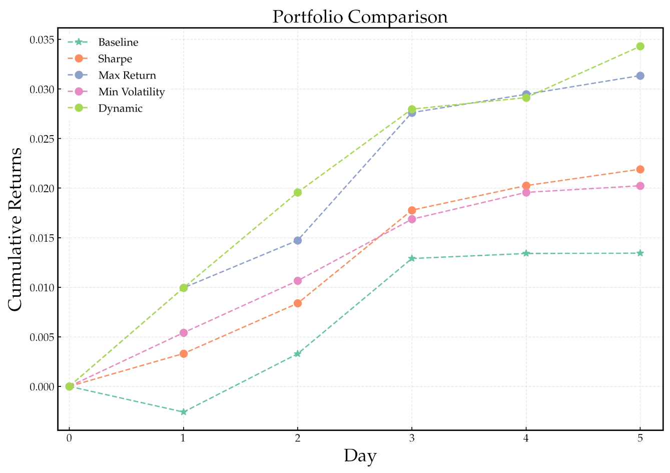

Figure 4.2: Portfolio Comparison

Risk metrics Transaction costs impact

Strategy switching frequency analysis

Comparative Analysis: Compare the performance of all strategies, highlighting the strengths and weaknesses of each.

4.2.2 Dynamic Strategy Evaluation

Probability threshold sensitivity Strategy switching effectiveness Portfolio turnover analysis Risk-adjusted performance metrics

4.2.3 Strategy Performance Calculation

In addition to backtesting, we calculate the expected performance of the portfolio optimization strategies based on the predicted returns and the optimal weights obtained from the optimization process. This involves estimating the cumulative portfolio return

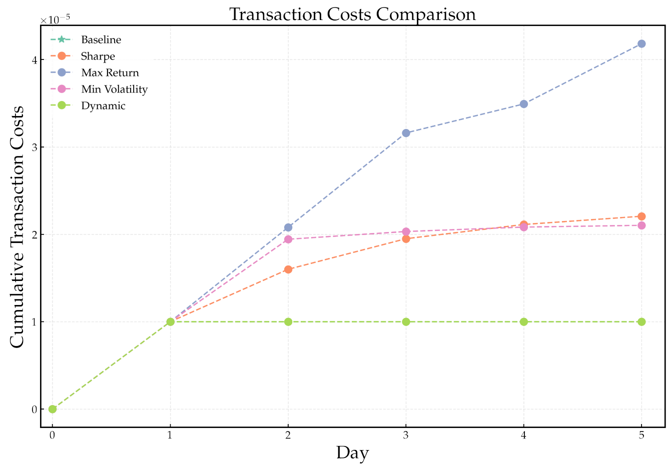

Figure 4.3: Transaction Costs Comparison

and the predicted portfolio volatility, which can later be compared with the actual returns from backtesting to assess the accuracy and effectiveness of the strategies.

Portfolio Class Implementation

The `Portfolio` class is designed to manage the portfolio optimization and performance evaluation process. It integrates the optimizer, various strategies, and performance calculation methods. The key components of the class include:

- **Assets**: A list of asset tickers included in the portfolio.
- **Asset Returns**: Historical returns of the assets.
- **Predicted Volatilities**: Predicted volatilities for each asset.
- **Optimizer**: An instance of the optimizer used for portfolio optimization.
- **Risk-Free Rate**: The risk-free rate used in calculations (e.g., for the Sharpe ratio).
- **Broker Fee**: Transaction cost rate applied to trades.

Optimal Weights Calculation

The method `get_optimal_weights` computes the optimal portfolio weights based on the selected strategy. It utilizes the optimizer and strategy classes to solve the optimization problem as formulated in previous sections. The optimal weights $\mathbf{w}$ are obtained by:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \left(\text{Objective Function}(\mathbf{w}; \boldsymbol{\mu}, \Sigma)\right)$$

subject to the relevant constraints for the chosen strategy.

Strategy Evaluation Method

The method `evaluate_portfolio` performs the following steps to calculate the expected performance of the portfolio based on the selected strategy:

1. **Initialization**: Set up initial variables, including lists to store optimal weights, predicted volatilities, covariance matrices, and daily returns.
2. **Loop Over Time Periods**:
   a) For each day t in the dataset:
      - **Data Preparation**: Collect the asset returns $\boldsymbol{\mu}_t$ and predicted volatilities σ_t up to day t.
      - **Set Predictions**: Update the optimizer with the current predictions using the methods:
        - `set_predictions` for single-day predictions.
        - `set_cml_log_return` or `set_predictions_cml` for cumulative predictions.
      - **Covariance Matrix Calculation**: Compute the covariance matrix Σ_t using the predicted volatilities and a given correlation matrix ρ:

$$\Sigma_t = \text{diag}(\sigma_t)\, \rho\, \text{diag}(\sigma_t)$$

      - **Optimal Weights Determination**: Compute the optimal weights $\mathbf{w}_t$ for day t using the selected strategy:

$$\mathbf{w}_t = \arg\min_{\mathbf{w}} \left(\text{Objective Function}(\mathbf{w}; \boldsymbol{\mu}_t, \Sigma_t)\right)$$

      - **Predicted Portfolio Performance**: Calculate the predicted portfolio return $R_{\text{portfolio},t}$ and volatility $\sigma_{\text{portfolio},t}$:

$$R_{\text{portfolio},t} = \mathbf{w}_t^\top \boldsymbol{\mu}_t$$

$$\sigma_{\text{portfolio},t} = \sqrt{\mathbf{w}_t^\top \Sigma_t \mathbf{w}_t}$$

b) **Store Results**: Save the optimal weights and predicted volatilities for later analysis.

3. **Performance Metrics Calculation**:

   - **Cumulative Predicted Return**: Compute the cumulative predicted return over the time horizon:

$$\text{Cumulative Predicted Return} = \prod_{t=1}^{T}(1 + R_{\text{portfolio},t}) - 1$$

   - **Average Predicted Volatility**: Calculate the average predicted portfolio volatility:

$$\overline{\sigma}_{\text{portfolio}} = \frac{1}{T}\sum_{t=1}^{T}\sigma_{\text{portfolio},t}$$

Comparison with Backtesting Results

The predicted portfolio performance metrics are compared with the actual performance obtained from backtesting to assess the accuracy and effectiveness of the optimization strategies. Key comparisons include:

- **Predicted vs. Actual Returns**: Comparing the cumulative predicted return with the cumulative actual return from backtesting.
- **Predicted vs. Actual Volatility**: Comparing the predicted portfolio volatility with the realized volatility during backtesting.
- **Sharpe Ratio Analysis**: Evaluating the predicted Sharpe ratio against the actual Sharpe ratio obtained from backtesting.

Implementation Notes

- **Covariance Matrix Estimation**: The covariance matrix Σ_t is estimated using the predicted volatilities and a given correlation matrix, capturing the relationships between assets.

- **Dynamic Updating**: The optimizer is updated at each time step with new predictions to reflect changes in market conditions.
- **Handling Log Returns**: If log returns are used, they are converted appropriately when calculating cumulative returns:

$$R_{\text{portfolio},t} = e^{R^{\log}_{\text{portfolio},t}} - 1$$

- **Multivariate Normal Distribution**: The joint distribution of asset returns is modeled using a multivariate normal distribution with mean $\boldsymbol{\mu}_t$ and covariance Σ_t, which is useful for probabilistic assessments in dynamic strategies.
- **Comparison of Strategies**: By evaluating the predicted performance of different strategies (e.g., maximum Sharpe ratio, maximum return, minimum volatility), we can identify which strategies are expected to perform better under the predicted market conditions.
- **Code Integration**: The `evaluate_portfolio` method integrates with the optimizer and strategies seamlessly, ensuring that the performance calculation is consistent with the optimization process.

Conclusion

The strategy performance calculation provides essential insights into the expected performance of the portfolio optimization strategies based on predicted returns and volatilities. By comparing these predictions with actual backtested results, we can evaluate the robustness and reliability of the strategies in different market conditions. This process aids in refining the strategies and improving their practical applicability in portfolio management.

4.3 Backtesting Results

Provide detailed results from the backtesting, including cumulative returns, volatility, Sharpe ratios, and transaction costs for each strategy.

4.3.1 Transaction Costs impact

4.3.2 Compared with Predicted Performance

4.3.3 Different market conditions

4.3.4 Out-of-sample performance

4.3.5 Statistical significance tests

5 Discussion

5.1 Interpretation of Results

5.1.1 Key findings and insights

Discuss what the results mean in the context of your research objectives.

5.1.2 Dynamic Strategy Insights

Delve into why the dynamic strategy outperformed others, considering market conditions and model performance.

5.1.3 Model robustness and generalization

5.2 Implications for Practitioners

Explain how your findings can be applied in real-world portfolio management.

5.2.1 Limitations of the approach

Acknowledge any limitations in your study, such as data constraints, model assumptions, or external factors.

5.2.2 Future research directions

Suggest potential areas for further research based on your findings.

5.3 Comparative Analysis

Comparison with existing methods

5.3.1 Advantages and disadvantages

Discuss the pros and cons of your approach compared to traditional portfolio optimization strategies.

5.3.2 Implementation challenges

Discuss any practical difficulties in applying your approach to real-world scenarios.

5.3.3 Market impact considerations

6 Conclusion

6.1 Summary of Findings

6.1.1 Summary of Findings

Recap the main results and how they address your research questions.

6.1.2 Key findings and insights

Discuss what the results mean in the context of your research objectives.

6.2 Recommendations

Offer suggestions for practitioners based on your findings.

6.3 Future Research Directions

6.3.1 Model improvements

Discuss potential enhancements to your model or methodology.

6.3.2 Additional strategy considerations

Suggest new strategies or modifications to existing ones.

6.3.3 Alternative applications

Propose other areas where your approach could be useful.

6.3.4 Scalability considerations

Abbreviations

List of Figures

List of Tables

Bibliography

[HPW17] J. B. Heaton, N. G. Polson, and J. H. Witte. “Deep Learning for Finance: Deep Portfolios”. In: *Applied Stochastic Models in Business and Industry* 33.1 (2017), pp. 3–12.

[Lin65] J. Lintner. “The Valuation of Risk Assets and the Selection of Risky Investments in Stock Portfolios and Capital Budgets”. In: *The Review of Economics and Statistics* 47.1 (1965), pp. 13–37.

[Mar52] H. Markowitz. “Portfolio Selection”. In: *The Journal of Finance* 7.1 (1952), pp. 77–91.

[Mos66] J. Mossin. “Equilibrium in a Capital Asset Market”. In: *Econometrica* 34.4 (1966), pp. 768–783.

[Qia05] E. Qian. *Risk Parity Portfolios: Efficient Portfolios Through True Diversification*. Panagora Asset Management, 2005.

[Sha64] W. F. Sharpe. “Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk”. In: *The Journal of Finance* 19.3 (1964), pp. 425–442.