.NET for systematic strategies

A brief recap on systematic strategies

Mnacho Echenim

Grenoble INP-Ensimag

2024-2025

Outline

On systematic strategies

Definition

A systematic strategy can be executed without any human intervention

In other words, this is a portfolio management strategy in which an algorithm

- Computes the moments when the portfolio is updated
- Computes the composition of the portfolio

On systematic strategies

Definition

A systematic strategy can be executed without any human intervention

In other words, this is a portfolio management strategy in which an algorithm

- Computes the moments when the portfolio is updated
- Computes the composition of the portfolio

Input : Initial value V_{t_0} of the portfolio (how much to invest)
Input : Market data for the underlying assets
Output: Portfolio values

```
1  begin
2      UpdateCompo(t_0) // nb: there can be additional parameters
3      foreach date t > t_0 do
4          V_t := UpdatePortfolioValue(t)
5          if RebalancingTime(t) then
6              UpdateCompo(t)
7          end
8      end
9      return (V_{t_0}, ..., V_{t_N})
10 end
```

Algorithm 2: Overview

On systematic strategies (2)

Constraint to satisfy:

- The portfolio must be **self-financing**
  - The portfolio value immediately before and after an update must be the same
- It is forbidden to use data ahead of time
  - For example, `UpdateCompo`(t) cannot use any data from time $t + \delta$, where $\delta > 0$

On systematic strategies (2)

Constraint to satisfy:

- The portfolio must be **self-financing**
  - The portfolio value immediately before and after an update must be the same
- It is forbidden to use data ahead of time
  - For example, `UpdateCompo`(t) cannot use any data from time $t + \delta$, where $\delta > 0$

What distinguishes systematic strategies

- When does `RebalancingTime` return **true**?
- How does `UpdateCompo` compute a new composition?

Example 1: uniform strategy

Principle: Construct a portfolio with n assets and every Monday, invest the same amount of cash in each asset

Example 1: uniform strategy

Principle: Construct a portfolio with n assets and every Monday, invest the same amount of cash in each asset

```
RebalancingTime(t)
begin
    return t.Day == Monday
end
```

```
UpdateCompo(t, V_t)
begin
    for i = 1, ..., n do
        q_i ⟵ V_t / (n · S_i^t)
    end
    return (q_1, ..., q_n)
end
```

Example 2: min vol with target return

$$\texttt{OptWeights}(\Sigma, \rho, \tau) = \min_{\omega \in [0,1]^n} (\omega^{\mathrm{T}} \cdot \Sigma \cdot \omega) \quad \text{s.t.} \quad \omega^{\mathrm{T}} \rho \geq \tau$$

Example 2: min vol with target return

$$\texttt{OptWeights}(\Sigma, \rho, \tau) = \min_{\omega \in [0,1]^n} (\omega^{\mathrm{T}} \cdot \Sigma \cdot \omega) \quad \text{s.t.} \quad \omega^{\mathrm{T}} \rho \geq \tau$$

```
1 UpdateCompo(t, V_t, Σ_t, ρ_t, τ_t)
2 begin
3     ω ⟵ OptWeights(Σ_t, ρ_t, τ_t)
4     for i = 1, ..., n do
5         q_i ⟵ ω_i · V_t / S_i^t
6     end
7     return (q_1, ..., q_n)
8 end
```

Example 2: min vol with target return

$$\texttt{OptWeights}(\Sigma, \rho, \tau) = \min_{\omega \in [0,1]^n} (\omega^{\mathrm{T}} \cdot \Sigma \cdot \omega) \quad \text{s.t.} \quad \omega^{\mathrm{T}} \rho \geq \tau$$

```
1 UpdateCompo(t, V_t, Σ_t, ρ_t, τ_t)
2 begin
3     ω ⟵ OptWeights(Σ_t, ρ_t, τ_t)
4     for i = 1, ..., n do
5         q_i ⟵ ω_i · V_t / S_i^t
6     end
7     return (q_1, ..., q_n)
8 end
```

Question

- V_t is already computed in the main loop
- How are Σ_t, ρ_t and τ_t computed?

Outline

General organization

- Starts on Aug. 29th, ends on Sept. 5th (computer room E200)
  - No oral defense
  - **Warning:** file sharing does not work with Windows
  - Git repo: `gitlab.ensimag.fr` (if necessary, install and use the github client to access files)
- 1 or 2 students per group
  - The groups should be created on `teide`
  - Grades may be different for the members of a same group
- Source code should be uploaded to `teide` on Sept. 5th
- **Nb:** your code will be executed automatically for the evaluation phase, it is important to make sure the provided python script works as expected before posting your code

Goal of the project

Tool to develop

A library that permits to perform forward and backtests on a hedging portfolio

- Quick recap: given an option O with maturity T and initial price p, the hedging portfolio for O
  - Invests in the underlying assets of the option and in the risk-free rate
  - Has an initial value $V_0 = p$
  - At time t_i contains a quantity $\delta^j_{t_i}$ of the jth risky asset
  - Has a final value $V_T = \mathrm{payoff}(O, \mathtt{mkt})$
- Option to hedge:
  - Average basket: $\left(\sum_{i=1}^n \omega_i S_T^i - K\right)_+$, where $n \geq 1$ and $\omega_i > 0$ for $i = 1, \ldots, n$

Forwardtest or backtest?

In general:

- A forwardtest consists in running the algorithm on simulated data
- A backtest consists in running the algorithm on historical data (the data is simulated for this project)

Code organization

The code should not depend on the data source that is used

What you will learn to use from the .NET framework

- Programming in C# on Visual Studio 2022 Community
- Handling dependencies (*Nuget*)
- Working with persistent data (*LINQ*)
- Creating and running unit tests (*NUnit*, ...)
- Possibly some advanced features
  - gRPC

What is provided

Pricing library

- Test parameters
  - Basket, PricingParameters
  - RebalancingOracleDescription
- Utilities.MarketDataFeed
  - ShareValue, DataFeed, SimulatedMarketValueProvider
  - RiskFreeRateProvider
- Computations
  - Pricer, PricingResults

What is provided (cont'd)

- Test data
  - A folder containing some test parameters and market data on which preliminary tests can be carried out
- *result_analysis.ipynb*
  - Simple notebook to analyze the hedging results that are output by the application
  - Visualization of the difference between the theoretical price and the hedging portfolio's value
  - Computation of the tracking error
- *generate-backtest-results.py*
  - Python script that will be used at the end of the project on all groups for the evaluation

Final recommendations

- Always have a clear definitions of the functionalities you are about to code
- Use an incremental approach to your development
  - Each increment must have a clear outcome
- Make sure you spend enough time and effort on code refactoring
  - Your code must be readable and maintainable at all times