General Idea

This work focuses on **portfolio optimization** involving the allocation of investments across multiple financial instruments and techniques.

The objective is to **determine an optimal strategy that balances returns and risks** while accounting for uncertainty in market conditions.

The analysis considers various approaches to resource allocation, enabling the formulation of strategies that align with predefined performance and risk management criteria.

The problem is addressed in a general framework to accommodate diverse **instruments** and market **scenarios**.

Introduction

- We will propose an optimization model designed to **maximize the amount of money invested** that an individual could have at the start of a future period.
- The model considers the availability of financial instruments that can be leveraged to adjust the current budget, enabling the purchase of additional stocks in the future.

Context

- **Protagonist**: Thomas Smith has saved $1.000.000 from his hard work and wants to invest it.
- **Financial Advisor's Recommendation**: Invest in *Apple, Google and Microsoft stocks*, which is expected to show an upward trend starting from t_1 (next week).
- **Uncertainty**: The stock price this week can follow one of 5 possible scenarios, each with a price (p_{ji}) and probability (π_i).
- **Goal**: Maximize the amount of money invested by Thomas Smith at the start of next week.

Investment Options

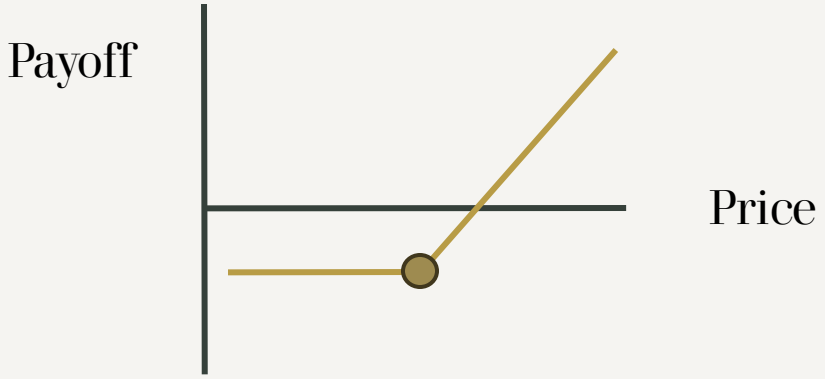
a. Buy a number (s_0) of shares today at the current price (p_0).
b. Purchase call options (c) at price (p_c), with a strike price (p_{sp}) exercisable next week.
c. Enter a futures contract (f_a) to sell shares at an agreed price (p_f), deciding whether to:
    - Buy shares today (f_0), or
    - Buy and sell them directly next week (f_1).
d. Save some money (w) to buy shares next week.

Key Decisions

1. How many shares to buy today?
2. How many call options to purchase?
3. How many shares to commit to selling in the futures contract?
4. How to acquire shares for the futures contract (buy today or next week)?
5. How much money to reserve for next week’s purchases?
6. How many shares to buy under each possible scenario?

The two used derivatives:

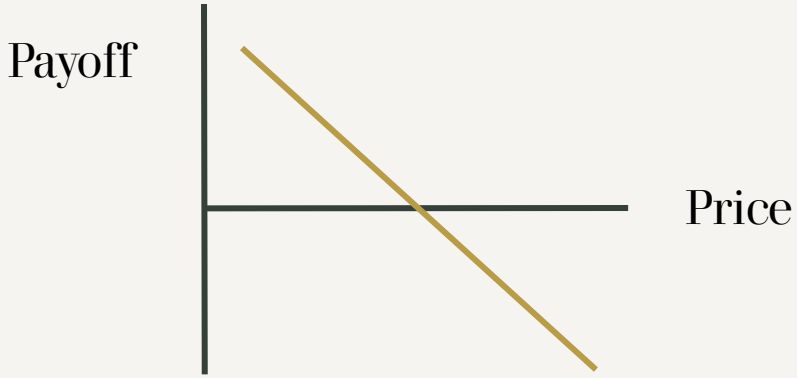
Long Call Option



Strike Price

A **long call option** gives Thomas Smith the right to buy an asset at a fixed price at the expiration date.

Short Futures



A **short futures** position obligates Thomas Smith to deliver an asset at a predetermined price on the expiration date.

Model Setup

Parameters

- p_{0j}: Price of Stock j today
- p_{ij}: The price of Stock j in scenario i.
- π_i: Probability of scenario i occurring.
- p_{cj}: Price of each call option of stock j.
- p_{fj}: Agreed price of futures contract for stock j.
- B: Total budget available.
- p_{spj}: Strike price of the call options of stock j.
- N: Number of different stocks.
- M: Number of scenarios.
- β: Risk aversion parameter

Variables

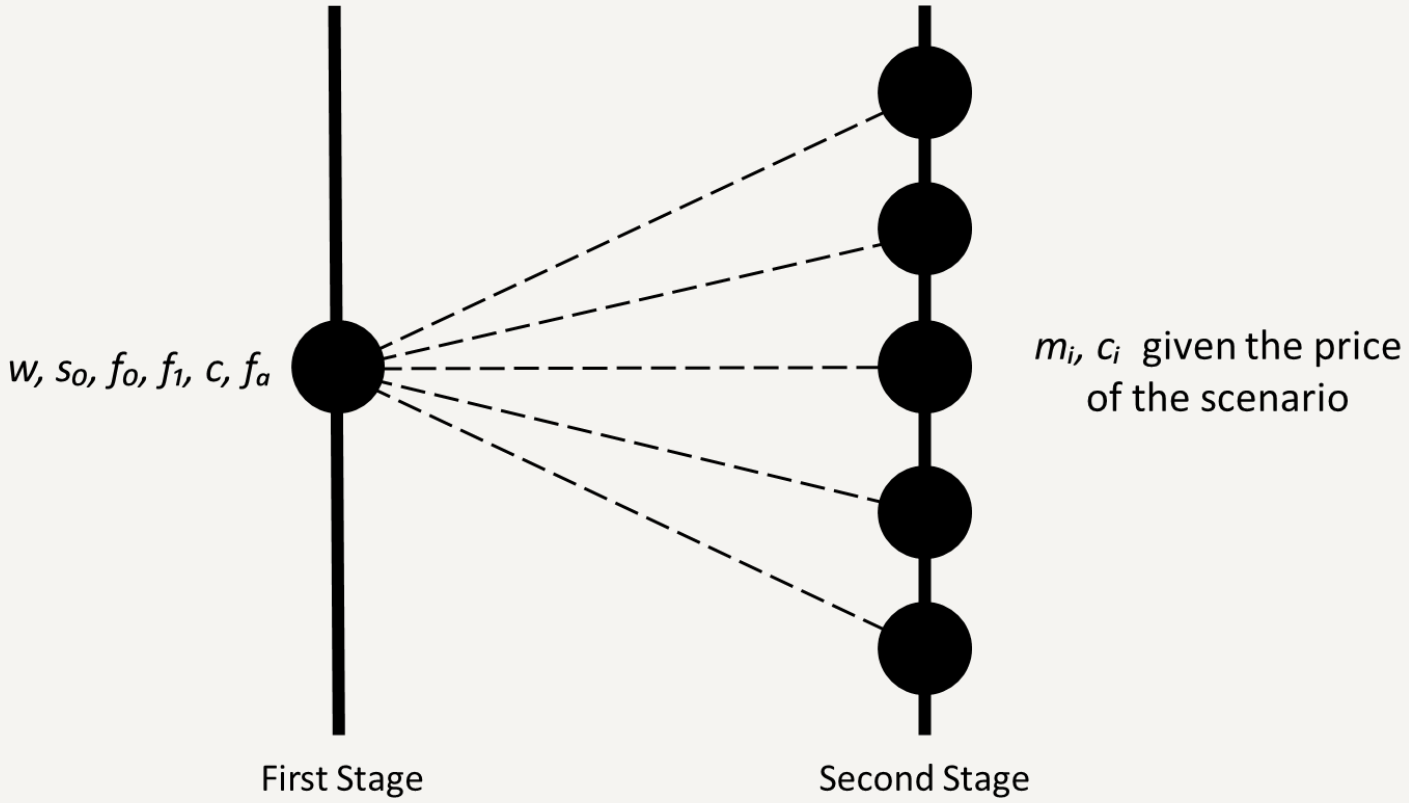
- s_{oj}: Number of stock j bought at time zero.
- c_j: The number of call options of stock j purchased at time zero
- c_{ji}: Amount of call options of stock j executed in scenario i.
- f_{aj}: amount of stock j agreed in the futures contract.
- f_{0j}: Number of stock j bought at $t = 0$ that will be used to comply with the amount agreed in the futures contract.
- f_{1j}: Number of stock j bought at $t = 1$ that will be used to comply with the amount agreed in the futures contract.
- w: Money from the budget that is not used today.
- $\boldsymbol{m_{ji}}$: amount of stock j that are bought at market price in scenario i.
- ξ: parameter for the Value at Risk threshold.
- η_i :excess over the threshold of scenario i.

Objective Function

- The objective function is still amount of money invested at the beginning of next week, but now we have to consider N different stocks to maximize. That value is given by:

$$\max\ \beta\left(\sum_{i=1}^{M}\pi_i\left(\sum_{j=1}^{N}p_{ji}(s_{j0}+m_{ji}+c_{ji})\right)\right)+\ (1\ -\beta)\left(\xi\ -\ \frac{1}{1\ -\ \alpha}\sum_{i=1}^{M}\pi_i\eta_i\right)$$

Scenario Tree



Constraints

- Budget constraint: ensures that the money we spend on $t = 0$ plus the money we do not spend are equal to the budget we have.

$$w + \sum_{j=1}^{N} c_j \cdot p_{cj} + f_{0j} \cdot p_{0j} + s_{0j} \cdot p_{0j} = B$$

- Second stage budget constraint: money we spend on the second stage (left side of the equation) has to be less or equal than the money we have gained or retained from the first stage.

$$\sum^{N} c_{ji} \cdot p_{spj} + m_{ji} \cdot p_{ij} \leq w + \sum_{j=1}^{N} f_{0j} \cdot p_{fj} + f_{1j}(p_{fj} - p_{ij}) \quad \forall i$$

Constraints

- Futures contract satisfaction: this ensures that the stocks bought to comply with the contract are sufficient.

$$f_{0j} + f_{1j} = f_{aj} \quad \forall j$$

- Maximum of stocks to buy in t = 1 for futures contract: we need to make sure that we have enough money to buy all the stocks we still need to satisfy the futures contract.

$$f_{1j} \leq \frac{w + \sum_{j=1}^{N} f_{0j} \cdot p_f}{p_{ij}} \quad \forall i, j$$

- Calculation of executed options: the amount of executed options will be equal to the amount of options bought if the price of the stock on the scenario is greater than the strike price and 0 if not.

$$cij = \begin{cases} 0 & if \quad p_{ij} < p_{spj} \\ c_j & if \quad p_{ij} > p_{spj} \end{cases} \quad \forall i, j$$

Constraints

- Penalization of CVaR lower bound when active:

$$\xi - \sum_{j=1}^{N} p_{ji}(s_{j0} + m_{ji} + c_{ji}) \leq \eta_i \quad \forall i$$

- Non negativity of variables. given that all the variables are amounts of stocks to be traded none of them can take a negative value.

$$s_{oj}, c_j, c_{ij}, f_{aj}, f_{0j}, f_{1j}, w, m_{ij}, \eta_i \geq 0$$

Code and Architecture

Main Functions

Get_current_prices: a function that using yfinance library retrieves real time prices for chosen stocks.

Monte_carlo_simulation: a function that simulates prices evolution using historical data.

Get_option_prices: using yfinance the function retrieves prices of call and put options and their relative strike prices. The indexes defines the time for expiration data and price collection.

Get_futures_prices: The function does the same of get_option_prices using different tickers due to the different code of futures.

Pretty_params: This function is used to define values useful for the simulated values approach.

```python
def get_option_prices(tickers):
    """
    Fetches the first available call and put option prices for each stock.

    Args:
        tickers (list): List of stock ticker symbols.

    Returns:
        tuple: Two lists, `oc` (call prices) and `op` (put prices), and `psp` (strike prices for call
options).
    """
    oc = []  # Option call prices
    op = []  # Option put prices
    psp = []  # Strike prices for call options

    for ticker in tickers:
        try:
            stock = yf.Ticker(ticker)

            # Get the first available expiration date
            expiration_dates = stock.options
            if not expiration_dates:
                raise ValueError(f"No options available for {ticker}.")
            first_expiration = expiration_dates[1]

            # Fetch the option chain for the first expiration date
            option_chain = stock.option_chain(first_expiration)

            # Get the current stock price
            current_price = stock.history(period="1d").iloc[-1]['Close']

            # Find the first call option with a strike price >= current price
            call_price = None
            strike_price = None
            for _, row in option_chain.calls.iterrows():
                if row['strike'] >= current_price:
                    call_price = row['lastPrice']
                    strike_price = row['strike']
                    break

            if call_price is None or strike_price is None:
                raise ValueError(f"No suitable call options found for {ticker}.")

            # Get the first put price (at the first strike available)
            put_price = option_chain.puts.iloc[0]['lastPrice']

            # Append to the lists
            oc.append(call_price)
            op.append(put_price)
            psp.append(strike_price)

        except Exception as e:
            print(f"Failed to fetch options for {ticker}: {e}")
            oc.append(None)  # Handle missing data gracefully
            op.append(None)
            psp.append(None)

    return oc, psp
```

[Explication of get_option_prices function]

Parameters and Variables

The parameters are simply defined calling the function explained earlier.
To **solve the LP** we used the library **cvxpy**, Python-embedded modeling language for convex optimization problems.
To make parameters readable by the cvxpy library we have to convert them in array using **NumPy**'s array function.
The decision variables contains the index as N or M where **N** is the **number of asset** and **M** is the **number of scenario**.
Due to the method used by cvxpy, the **integer constraint** must be put in this section.

```python
import matplotlib.pyplot as plt
import numpy as np
import cvxpy as cp

# Parametri
tickers = ["AAPL", "GOOGL", "MSFT"]  # Replace with your tickers
p0 = np.array(get_current_prices(tickers))

prob = np.array([0.2, 0.2, 0.2, 0.2, 0.2]) # Probabilità di ciascuno scenario

# Inizializza prezzi e scenari
simulated_prices = monte_carlo_simulation(tickers, days=5, scenarios=5)  # (N x M matrix)
p1 = np.array(simulated_prices)
print(p1.shape)
pc, psp = np.array(get_option_prices(tickers))  # Prezzi delle opzioni, ecc.
#pc = pc/100
B = 1000000
N = 3  # Numero di asset
M = 5  # Numero di scenari
alpha = 0.95  # Livello di confidenza

futures_tickers = ["APC.F", "ABEC.F", "MSF.F"]  # Esempi di futures tickers
futures_data = get_futures_prices(futures_tickers)

# Prezzi dei futures
pf = np.array([item['last_price'] for item in futures_data])


beta = 0.5

# Define the decision variables
s0 = cp.Variable(N, integer=True)

c = cp.Variable(N, integer=True)

ci  = cp.Variable((N,M), integer=True)

fa  = cp.Variable(N, integer=True)

f0 = cp.Variable(N, integer=True)

f1 = cp.Variable(N, integer=True)

mj = cp.Variable((N,M),  integer=True)

w = cp.Variable(nonneg=True)

ni = cp.Variable(M, nonneg=True)

xi = cp.Variable()
```

Objective Function and Constraints

The **objective function** is defined **section by section**, where each variable correspond to a part of the expression in the mathematical formulation.

The main methods are taken from the cvxpy library. Cp.sum() is used to express the summation, while cp.maximize() is used to express the maximization problem.

The constraints are defined using the same approach.

The **non-negativity constraint** is set as a unique list constraints in order to avoid the double parameter in the cp.Variable() method.

```
# Objective function

# First term: Weighted expected payoff
inner_sum = cp.sum(cp.multiply(p1, s0[:, None] + mj + ci), axis=0)  # Sum over j for each i
expected_payoff = cp.sum(cp.multiply(prob, inner_sum))  # Sum over i

# Second term: CVaR term
cvar_term = xi - (1 / (1 - alpha)) * cp.sum(cp.multiply(prob, ni))

# Full objective function
objective = cp.Maximize(beta * expected_payoff + (1 - beta) * cvar_term)

budget_constraint = (
    w + cp.sum(cp.multiply(pc, c)) + cp.sum(cp.multiply(f0, p0)) + cp.sum(cp.multiply(s0, p0)) == B
)

contract_constraints = [
    cp.sum(cp.multiply(psp, ci[:, i]) + cp.multiply(p1[:, i], mj[:, i])) <=
    w + cp.sum(cp.multiply(f0, pf)) + cp.sum(cp.multiply(f1, pf - p1[:, i]))
    for i in range(M)
]

stock_allocation_constraints = [f0[j] + f1[j] == fa[j] for j in range(N)]

futures_constraints = [
    f1[j] <= (w + cp.sum(cp.multiply(f0, pf))) / p1[j, i]
    for i in range(M) for j in range(N)
]

executed_options_constraints = [
    ci[j, i] <= cp.multiply((p1[j, i] >= psp[j]), c[j]) for i in range(M) for j in range(N)
]

cvar_constraints = [
      xi - cp.sum(cp.multiply(p1, s0[:, None] + mj + ci), axis=0) <=ni
    ]

nneg_constraints = [
    s0 >= 0,           # Non-negative constraint for s0
    c >= 0,            # Non-negative constraint for c
    ci >= 0,           # Non-negative constraint for ci
    fa >= 0,           # Non-negative constraint for fa
    f0 >= 0,           # Non-negative constraint for f0
    f1 >= 0,           # Non-negative constraint for f1
    mj >= 0            # Non-negative constraint for mj
]
```

Comparative Analysis

CASE 1 - Inputs

- **p0** (Current Prices):

  248.13,189.82,447.26
- **pc** (Historical Price Changes):

  2.34,3.75,4.64
- **psp** (Projected Scenario Prices):

  250.0,190.0,450.0
- **np.mean(p1, axis=1)** (Row-wise Mean of Simulated Scenario Prices):

  250.50,190.557,435.06
- **pf_** (Market Data for Futures Contracts):
  - Ticker: APC.F, Last Price: 236.19
  - Ticker: ABEC.F, Last Price: 184.17
  - Ticker: MSF.F, Last Price: 425.25

Additional parameters:

- **N** (Number of Assets): 3
- **M** (Number of Scenarios): 5
- **alpha** (Confidence Level): 0.95
- **Beta**: Range from 0.01 to 1.01

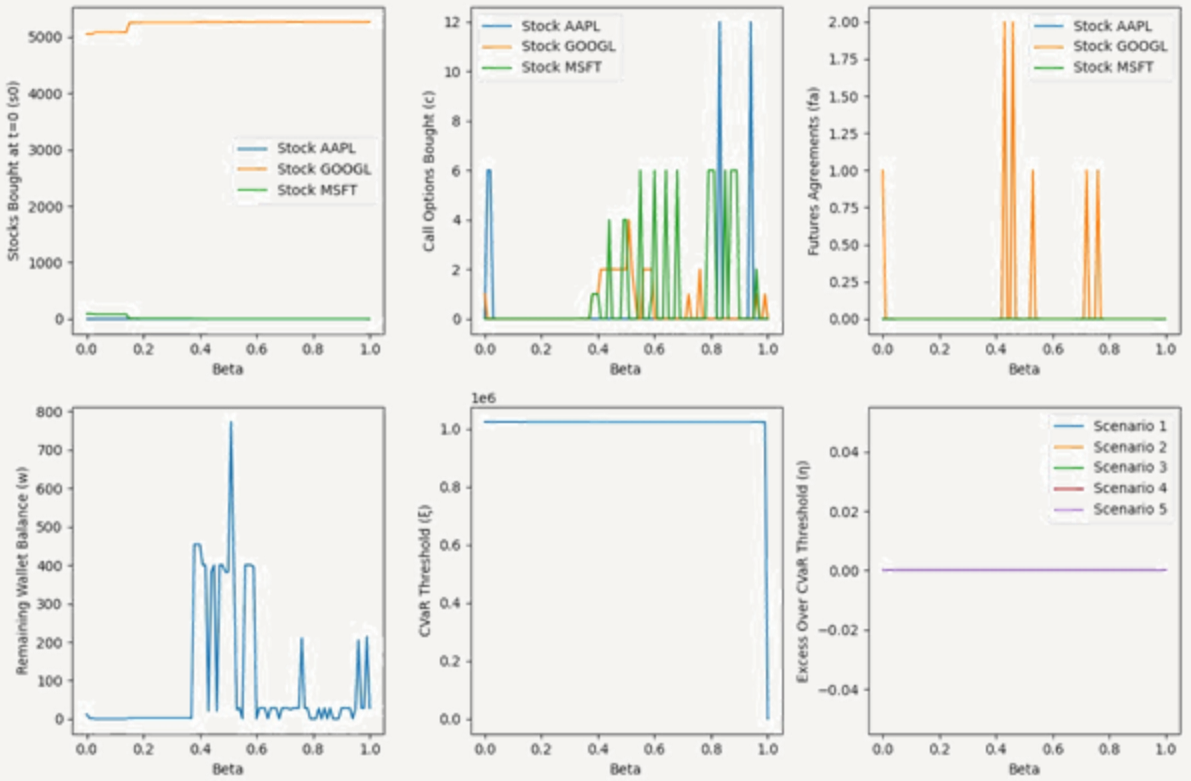
Probability:

- **0.2** for each scenario

CASE 1

We took into consideration the case with **equal probabilities**, **real data** and solved with **integer constraints**:

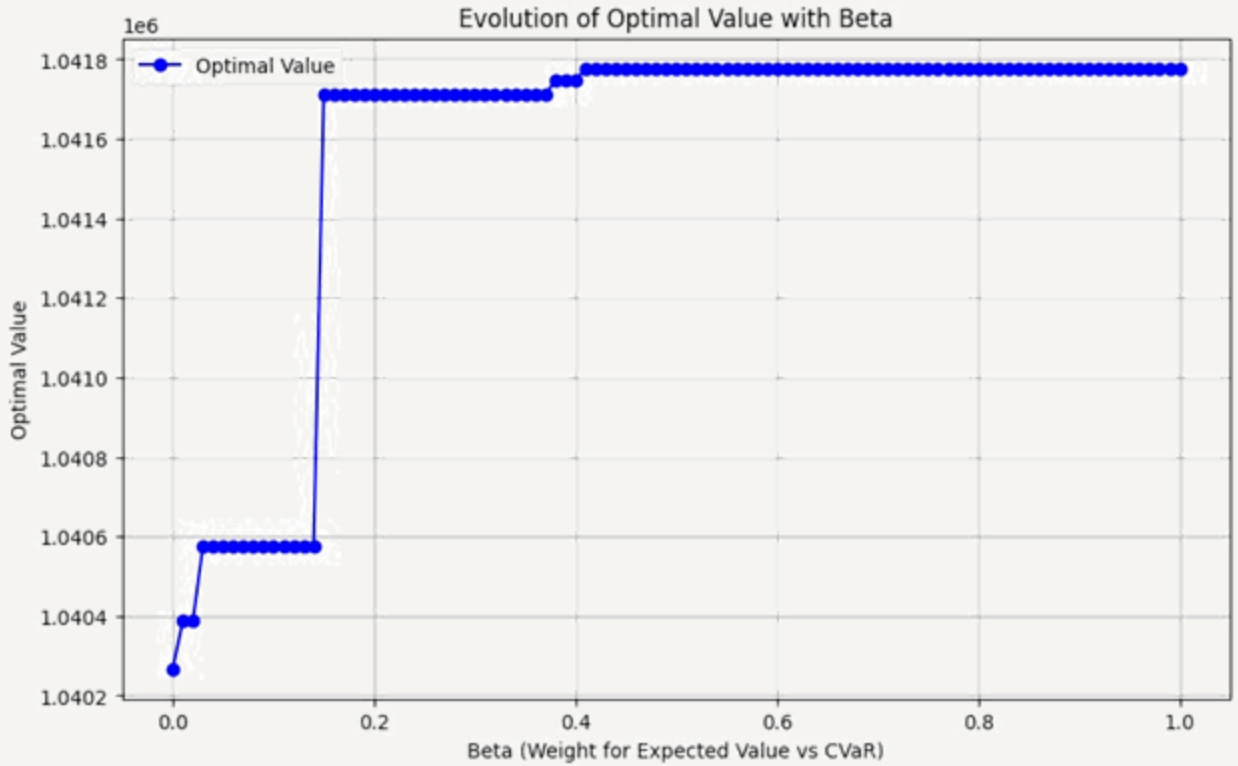
- For **very low Betas** all the purchasable items are considered, making the saved wallet empty.
- For **medium levels of Beta,** the main changes are in call options and futures bought. **MSFT** is now purchased but only as a **call option**.
- As the **beta increases,** *AAPL* becomes the most likeable **option** to buy, while **futures** are completely ignored.



CASE 1

The **optimal value** in terms of **payoff** is **never equal** to the **invested money for any value of β.** It also seems that the increase is not proportional as we can see in the spike at approximately 0.175β.

Although after the spike the optimal value increase other **two times**, after reaching a **β value of 0.4** it remains linear for all the remaining value.



CASE 2 - Inputs

- **p0_** (Current Prices):

  248.13, 189.82, 447.26

- **pc_** (Historical Price Changes):

  4.15, 12.12, 10.62

- **psp_** (Projected Scenario Prices):

  330, 240, 595

- **np.mean(p1_, axis=1)** (Row-wise Mean of Simulated Scenario Prices):

  288.43, 225.01, 466.26

- **pf_** (Market Data for Futures Contracts):
  - Ticker: APC.F, Last Price: 258.05
  - Ticker: ABEC.F, Last Price: 197.41
  - Ticker: MSF.F, Last Price: 465.16

Additional parameters:

- **N** (Number of Assets): 3
- **M** (Number of Scenarios): 5
- **alpha** (Confidence Level): 0.95

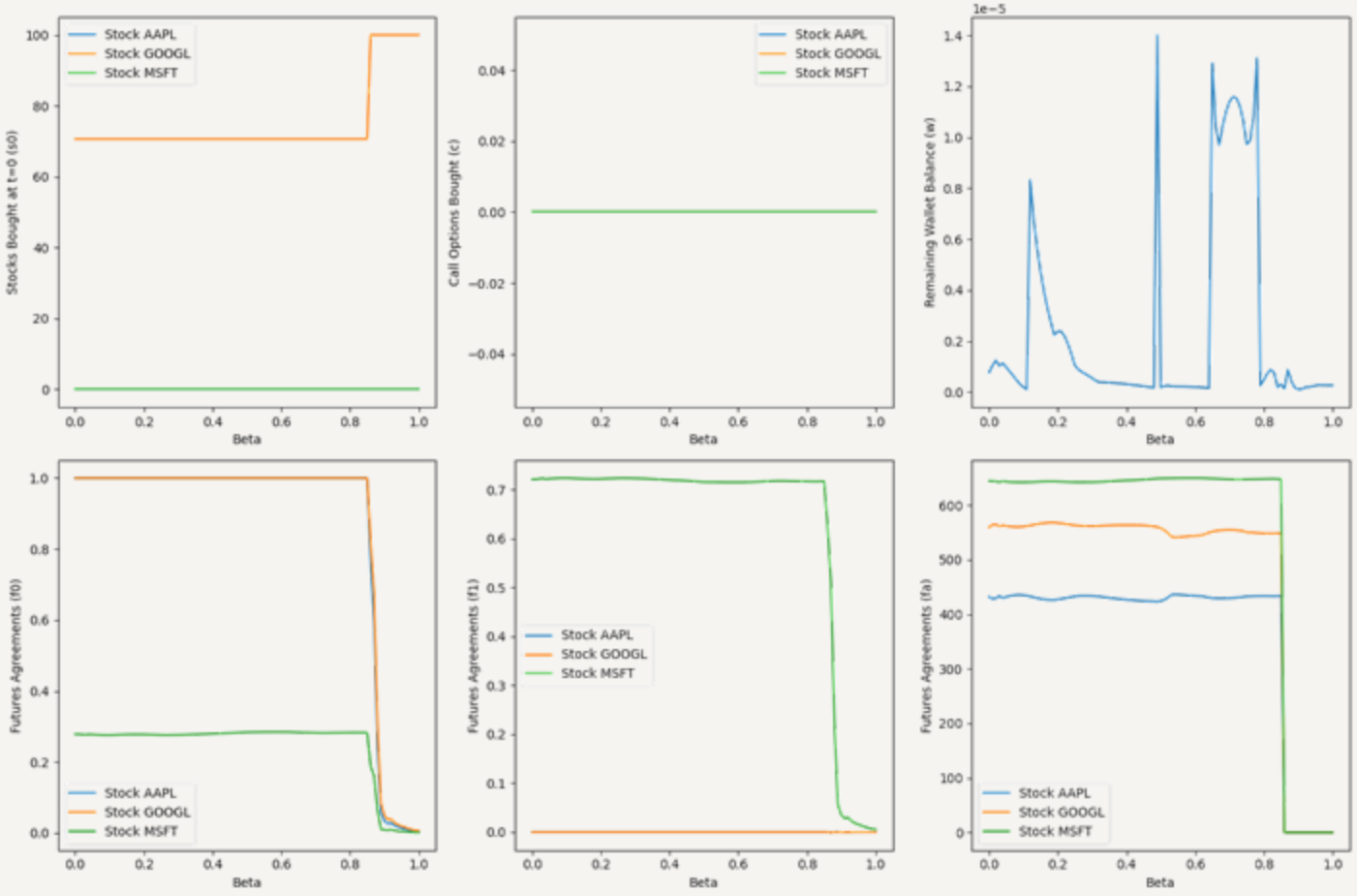
Probability:

- **prob_**= 0.05,0.25,0.4,0.25,0.05

CASE 2

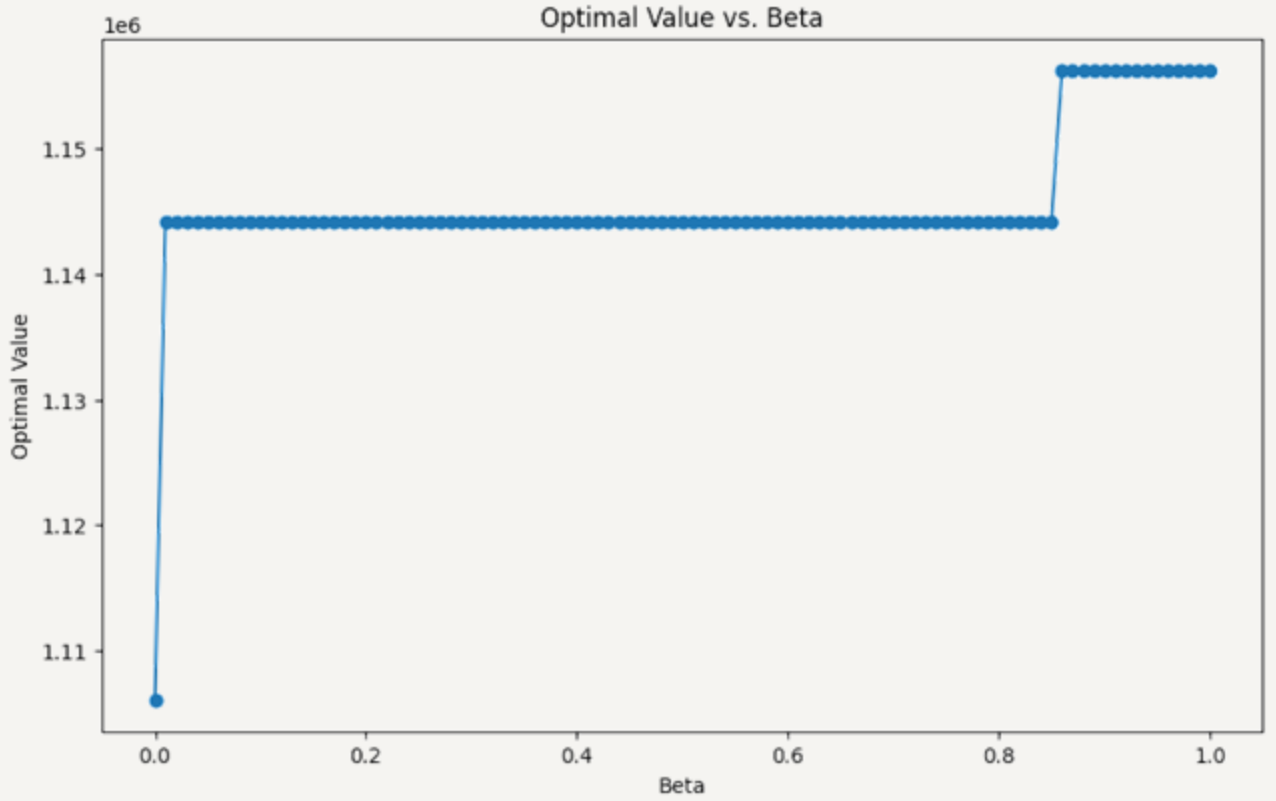
We took into consideration the case with **non-equal probabilities** and **artificial data:**

- The **wallet** is always used **completely**.
- For the **low and medium Beta** levels, the **GOOGL stock** is bought, together with **futures** in a **mixed strategy** for f0 and f1.
- **No call options** are used.
- As the **beta increases**, more **GOOGL** stocks are bought, as the **futures are abandoned** as a strategy.



CASE 2

- The **optimal value** in terms of **payoff** is **always higher** than the invested money, remaining **stable until Beta is 0.85**, then increasing more.



Conclusions

Conclusions

- The model incorporates **complex investment strategies**, allowing investors to choose from various instruments, including derivatives.
- **Realistic data** shows that **markets already price in future expectations**, aligning with the used **scenario-based approach**.
- Instruments like **call options and futures reflect this information in their pricing**, leading the model to view them as **risky or safe depending on the context**.
- In **real markets, stock and derivative performance tends to converge, preventing arbitrage opportunities**.
- When **simulated values** are used, the model shifts focus to futures, exploiting **potential arbitrage for higher payoffs**.
- The model is **theoretically correct** but demonstrates that **markets efficiently integrate most available information**, making the model primarily **sensitive to the input uncertainties**.