Case Assignment 3

adapted from *Practical Management Science*

Hedging Risk

``Kate Torelli, a security analyst for Lion-Fund, has identified a gold mining stock (ticker symbol GMS) as a particularly attractive investment. Torelli believes that the company has invested wisely in new mining equipment. Furthermore, the company has recently purchased mining rights on land that has high potential for successful gold extraction. Torelli notes that gold has underperformed the stock market in the last decade and believes that the time is ripe for a large increase in gold prices. In addition, she reasons that conditions in the global monetary system make it likely that investors may once again turn to gold as a safe haven in which to park assets. Finally, supply and demand conditions have improved to the point where there could be significant upward pressure on gold prices.

GMS is a highly leveraged company, so it is a risky investment by itself. Torelli is mindful of a passage from the annual report of a competitor, Baupost, which has an extraordinarily successful investment record: "Baupost has managed a decade of consistently profitable results despite, and perhaps in some respect due to, consistent emphasis on the avoidance of downside risk. We have frequently carried both high cash balances and costly market hedges. Our results are particularly satisfying when considered in the light of this sustained risk aversion." She would therefore like to hedge the stock purchase—that is, reduce the risk of an investment in GMS stock.

Currently GMS is trading at $100 per share. Torelli has constructed seven scenarios for the price of GMS stock one month from now. These scenarios and corresponding probabilities are shown below.

To hedge an investment in GMS stock, Torelli can invest in other securities whose prices tend to move in the direction opposite to that of GMS stock. In particular, she is considering over-the-counter put options on GMS stock as potential hedging instruments. The value of a put option increases as the price of the underlying stock decreases. For example, consider a put option with a strike price of $100 and a time to expiration of one month. This means that the owner of the put has the right t 100 per share one month in the future. Suppose that the price of GMS falls to $80 at that time. Then the holder of the put option can exercise the option and receive $(100 - 80$ 70, the option would be worth (100 - 70). However, if the price of GMS rises to $100 or more, the option expires worthless.

Torelli called an options trader at a large investment bank for quotes. The prices for three (European-style) put options are shown below. Torelli wants to invest $10 million in GMS stock and put options.

In [1]:
```
from itertools import product
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from copy import deepcopy
from scipy.optimize import fsolve
import scipy.stats as ss
from gurobipy import Model, GRB, QuadExpr
np.random.seed(42)
```

In [2]:
```
# Creating a new model
model = Model("portfolio_optimization")

# Adding variables (x1, x2, x3, and x4 for GMS stock and 3 put options)
x = model.addVars(4, lb=0, name="x")

# Objective: Minimize the portfolio variance (quadratic)
variance = QuadExpr()

# Inputs
total_investment = 10000000
scenarios = {
        0: {"prob": .05, "stock_price": 150},
        1: {"prob": .1, "stock_price": 130},
        2: {"prob": .2, "stock_price": 110},
        3: {"prob": .3, "stock_price": 100},
        4: {"prob": .2, "stock_price": 90},
        5: {"prob": .1, "stock_price": 80},
        6: {"prob": .05, "stock_price": 70}
}

options = {
    "A": {"strike_price": 90, "option_price": 2.20},
    "B": {"strike_price": 100, "option_price": 6.40},
    "C": {"strike_price": 110, "option_price": 12.50},
}
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2025-10-25
```

In [3]:
```
initial_stock_price = 100
investment_amount = 10_000_000
num_shares = investment_amount / initial_stock_price

returns = []
probabilities = []
for scenario in scenarios.values():
    final_stock_price = scenario["stock_price"]
    stock_return = (final_stock_price - initial_stock_price) / initial_stock_price
    returns.append(stock_return)
```

```
    probabilities.append(scenario["prob"])

mean_return_gms = np.dot(returns, probabilities)
std_return_gms = np.sqrt(np.dot(probabilities, (np.array(returns) - mean_return_gms

{"Mean Return on GMS": mean_return_gms,
"Std Dev Return on GMS": std_return_gms}
```

Out[3]:
```
{'Mean Return on GMS': 0.019999999999999997,
 'Std Dev Return on GMS': 0.18330302779823363}
```

In [4]:
```
option_a_strike = options["A"]["strike_price"]
option_a_cost = options["A"]["option_price"] * num_shares

returns_with_option_a = []
for scenario in scenarios.values():
    final_stock_price = scenario["stock_price"]
    option_payout = max(0, option_a_strike - final_stock_price) * num_shares
    total_return = ((final_stock_price * num_shares + option_payout - investment_am
                     / (investment_amount + option_a_cost))
    returns_with_option_a.append(total_return)

mean_return_option_a = np.dot(returns_with_option_a, probabilities)
std_return_option_a = np.sqrt(np.dot(probabilities, (np.array(returns_with_option_a

{
"Mean Return with Option A": mean_return_option_a,
    "Std Dev Return with Option A": std_return_option_a
}
```

Out[4]:
```
{'Mean Return with Option A': 0.01761252446183953,
 'Std Dev Return with Option A': 0.1559430278914797}
```

In [5]:
```
returns = []
for s in scenarios.values():
    stock_return = (s["stock_price"] / 100) - 1
    put_a_return = max(0, options["A"]["strike_price"] - s["stock_price"]) / option
    put_b_return = max(0, options["B"]["strike_price"] - s["stock_price"]) / option
    put_c_return = max(0, options["C"]["strike_price"] - s["stock_price"]) / option
    weighted_return = sum(p * r for p, r in zip([x[i] for i in range(4)], [stock_re
    returns.append(weighted_return)

mean_return = sum(s["prob"] * r for s, r in zip(scenarios.values(), returns))
for s, r in zip(scenarios.values(), returns):
    variance.add(s["prob"] * ((r - mean_return) ** 2))

model.setObjective(variance, GRB.MINIMIZE)

#constraints
model.addConstr(sum(x[i] for i in range(4)) == total_investment, "TotalInvestment")
model.addConstrs((x[i] >= 0 for i in range(4)), "NonNegative")

# Optimize the model
model.optimize()

if model.status == GRB.OPTIMAL:
```

```
    print(f"Optimal objective value: {model.objVal}")
    for v in model.getVars():
        print(f"{v.varName}: {v.x}")
else:
    print("Optimization was unsuccessful.")
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 11.0 (22631.2))

CPU model: 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz, instruction set [SSE2|AVX
|AVX2|AVX512]
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads

Optimize a model with 5 rows, 4 columns and 8 nonzeros
Model fingerprint: 0x2e96d301
Model has 10 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [7e-02, 1e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+07, 1e+07]
Presolve removed 4 rows and 0 columns
Presolve time: 0.03s
Presolved: 1 rows, 4 columns, 4 nonzeros
Presolved model has 10 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 3
 AA' NZ     : 6.000e+00
 Factor NZ  : 1.000e+01
 Factor Ops : 3.000e+01 (less than 1 second per iteration)
 Threads    : 1

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   6.98939120e+15  2.24432230e+13  1.11e+09 3.77e+09  1.98e+14     0s
   1   3.44967507e+12  4.76028473e+13  9.05e+06 3.07e+07  3.85e+12     0s
   2   1.09768306e+12 -8.21767266e+11  9.05e+00 3.07e+01  4.80e+11     0s
   3   7.48765259e+11  5.80518460e+11  3.24e-01 1.10e+00  4.21e+10     0s
   4   6.42174128e+11  6.29908638e+11  3.07e-07 1.10e-06  3.07e+09     0s
   5   6.32336781e+11  6.32307451e+11  1.86e-09 5.82e-11  7.33e+06     0s
   6   6.32309773e+11  6.32309744e+11  2.33e-10 9.27e-11  7.34e+03     0s
   7   6.32309746e+11  6.32309746e+11  1.86e-09 2.60e-12  7.34e+00     0s
   8   6.32309746e+11  6.32309746e+11  1.86e-09 2.91e-11  7.35e-03     0s
   9   6.32309746e+11  6.32309746e+11  1.86e-09 3.49e-10  7.36e-06     0s
  10   6.32309746e+11  6.32309746e+11  8.73e-11 6.90e-27  7.37e-09     0s

Barrier solved model in 10 iterations and 0.08 seconds (0.00 work units)
Optimal objective 6.32309746e+11

Optimal objective value: 632309746328.4358
x[0]: 8491321.7623498
x[1]: 7.913785017357825e-15
x[2]: 6.222519840645219e-14
x[3]: 1508678.2376502007
```

In [12]:
```
import math
x = [84913.2, 0, 0, 15086.8] # xi to be the amount allocated to investment i in hun

#returns
returns = []
for s in scenarios.values():
    stock_return = (s["stock_price"] / 100) - 1
    put_a_return = max(0, options["A"]["strike_price"] - s["stock_price"]) / option
    put_b_return = max(0, options["B"]["strike_price"] - s["stock_price"]) / option
    put_c_return = max(0, options["C"]["strike_price"] - s["stock_price"]) / option
    weighted_return = sum(p * r for p, r in zip(x, [stock_return, put_a_return, put
    returns.append(weighted_return)

# mean return
mean_return = sum(s["prob"] * r for s, r in zip(scenarios.values(), returns))
print(f"Mean Return: {mean_return:.4f}")

# variance
variance = sum(s["prob"] * ((r - mean_return) ** 2) for s, r in zip(scenarios.value

# standard deviation
standard_deviation = math.sqrt(variance)
print(f"Standard Deviation: {standard_deviation:.4f}")
```

```
Mean Return: 1094.7920
Standard Deviation: 7951.7907
```

In [7]:
```
strike_price_new_option = 120

#fair price for the $120 option
fair_price_new_option = sum(
    scenario["prob"] * max(strike_price_new_option - scenario["stock_price"], 0)
    for scenario in scenarios.values()
)

print(f"Fair price of the $120 strike price option: {fair_price_new_option}")
```

```
Fair price of the $120 strike price option: 20.5
```

In [8]:
```
model = Model("portfolio_optimization2")
x = model.addVars(5, lb=0, name="x")
variance = QuadExpr()

# inputs
total_investment = 10000000
scenarios = {
        0: {"prob": .05, "stock_price": 150},
        1: {"prob": .1, "stock_price": 130},
        2: {"prob": .2, "stock_price": 110},
        3: {"prob": .3, "stock_price": 100},
        4: {"prob": .2, "stock_price": 90},
        5: {"prob": .1, "stock_price": 80},
        6: {"prob": .05, "stock_price": 70}
}

options = {
```

```
    "A": {"strike_price": 90, "option_price": 2.20},
    "B": {"strike_price": 100, "option_price": 6.40},
    "C": {"strike_price": 110, "option_price": 12.50},
    "D": {"strike_price": 120, "option_price": 20.5},
}
```

In [9]:

```
returns = []
for s in scenarios.values():
    stock_return = (s["stock_price"] / 100) - 1
    put_a_return = max(0, options["A"]["strike_price"] - s["stock_price"]) / option
    put_b_return = max(0, options["B"]["strike_price"] - s["stock_price"]) / option
    put_c_return = max(0, options["C"]["strike_price"] - s["stock_price"]) / option
    put_d_return = max(0, options["D"]["strike_price"] - s["stock_price"]) / option

    weighted_return = sum(p * r for p, r in zip([x[i] for i in range(5)], [stock_re
    returns.append(weighted_return)

# variance
mean_return = sum(s["prob"] * r for s, r in zip(scenarios.values(), returns))
for s, r in zip(scenarios.values(), returns):
    variance.add(s["prob"] * ((r - mean_return) ** 2))

#the objective to minimize variance
model.setObjective(variance, GRB.MINIMIZE)

# constraints
model.addConstr(sum(x[i] for i in range(5)) == total_investment, "TotalInvestment")
model.addConstrs((x[i] >= 0 for i in range(5)), "NonNegative")

# Optimize the model
model.optimize()

# optimal solution
if model.status == GRB.OPTIMAL:
    print(f"Optimal objective value: {model.objVal}")
    for v in model.getVars():
        print(f"{v.varName}: {v.x}")
else:
    print("Optimization was unsuccessful.")
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 11.0 (22631.2))

CPU model: 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz, instruction set [SSE2|AVX
|AVX2|AVX512]
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads

Optimize a model with 6 rows, 5 columns and 10 nonzeros
Model fingerprint: 0xa98331cb
Model has 15 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [7e-02, 1e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+07, 1e+07]
Presolve removed 5 rows and 0 columns
Presolve time: 0.01s
Presolved: 1 rows, 5 columns, 5 nonzeros
Presolved model has 15 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 4
 AA' NZ     : 1.000e+01
 Factor NZ  : 1.500e+01
 Factor Ops : 5.500e+01 (less than 1 second per iteration)
 Threads    : 1

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   5.50598287e+15  3.06152462e+14  1.12e+09 9.50e+09  1.10e+15     0s
   1   6.92766406e+12  1.81680584e+13  1.47e+07 1.25e+08  2.56e+13     0s
   2   1.92465485e+12 -1.20930103e+13  1.47e+01 1.25e+02  2.80e+12     0s
   3   3.53203834e+11 -5.52709003e+11  1.85e-02 1.57e-01  1.81e+11     0s
   4   2.46043323e+11  2.12893415e+11  7.65e-05 6.49e-04  6.63e+09     0s
   5   2.16501459e+11  2.15771194e+11  1.86e-08 6.47e-10  1.46e+08     0s
   6   2.15810703e+11  2.15809973e+11  5.82e-11 5.82e-11  1.46e+05     0s
   7   2.15810012e+11  2.15810011e+11  1.46e-11 5.82e-11  1.46e+02     0s
   8   2.15810011e+11  2.15810011e+11  1.86e-09 1.25e-10  1.46e-01     0s
   9   2.15810011e+11  2.15810011e+11  2.91e-11 2.01e-11  1.46e-04     0s
  10   2.15810011e+11  2.15810011e+11  1.86e-09 5.82e-11  1.46e-07     0s
  11   2.15810011e+11  2.15810011e+11  1.46e-11 5.82e-11  1.46e-10     0s

Barrier solved model in 11 iterations and 0.04 seconds (0.00 work units)
Optimal objective 2.15810011e+11

Optimal objective value: 215810011419.66348
x[0]: 7885860.182173415
x[1]: 1.1190036512590855e-16
x[2]: 1.2266620517437152e-15
x[3]: 7.636359972646247e-16
x[4]: 2114139.817826584
```

In [13]:
```
import math
x = [78858.6, 0, 0, 0, 21141.38] # xi to be the amount allocated to investment i in
```

```
#returns
returns = []
for s in scenarios.values():
    stock_return = (s["stock_price"] / 100) - 1
    put_a_return = max(0, options["A"]["strike_price"] - s["stock_price"]) / option
    put_b_return = max(0, options["B"]["strike_price"] - s["stock_price"]) / option
    put_c_return = max(0, options["C"]["strike_price"] - s["stock_price"]) / option
    put_d_return = max(0, options["D"]["strike_price"] - s["stock_price"]) / option
    weighted_return = sum(p * r for p, r in zip(x, [stock_return, put_a_return, put
    returns.append(weighted_return)

# mean return
mean_return = sum(s["prob"] * r for s, r in zip(scenarios.values(), returns))
print(f"Mean Return: {mean_return:.4f}")

# variance
variance = sum(s["prob"] * ((r - mean_return) ** 2) for s, r in zip(scenarios.value

# standard deviation
standard_deviation = math.sqrt(variance)
print(f"Standard Deviation: {standard_deviation:.4f}")
```

```
Mean Return: 1577.1720
Standard Deviation: 4645.5347
```

Questions

1. What is the mean and standard deviation of a return on GMS stock?
   - Answer: Mean Return: 0.0199, Std Dev Return: 0.1833

2. If put option A is purchased, what is the mean and standard deviation of the return on GMS stock with the option?
   - Answer: Mean Return with Option A: 0.0176, Std Dev Return with Option A: 0.1559

3. Assuming that Torelli's goal is to minimize the standard deviation of the portfolio return, what is the optimal portfolio that invests all $10 million? (For simplicity, assume that fractional numbers of stock shares and put options can be purchased. Assume that the amounts invested in each security must be nonnegative. However, the number of options purchased need not equal the number of shares of stock purchased.) What are the mean return and standard deviation of return of this portfolio? How many shares of GMS stock and how many of each put option does this portfolio correspond to?

   - Answer: Optimal objective value: 632309746328.4358 x[0]: 8491321.7623498 x[1]: 7.913785017357825e-15 x[2]: 6.222519840645219e-14 x[3]: 1508678.2376502007

Based on the calculated values, the optimal portfolio allocation involves investing the entire $10 million in GMS stock and put option C, ensuring the most effective use of the investment. Mean Return: 1.09% and Standard Deviation of Return: 7.95% Shares of GMS

stock: 8491321.7623498 / 100 = 84913 shares Put Options A: 0 B: 0 c: 1508678.23 / 100 = 15087 shares

In [ ]:
```
4. Suppose that an option with strike price $120 were available. What would you pay
    * Answer:
      Fair price of the $120 strike price option: 20.5
             Optimal objective value: 215810011419.66348
             x[0]: 7885860.182173416
             x[1]: 1.1190036546856483e-16
             x[2]: 1.2266620525002597e-15
             x[3]: 7.636359979368507e-16
             x[4]: 2114139.8178265844

      As we can see from the above two set of optimal values, there is a change if
      Mean Return: 1577.1720 (1.5%)
      Standard Deviation: 4645.5347 (4.6%)
      The returns increase with lesser risk
```

In [ ]: