

RSM8301_Assignment_2

April 2, 2024

1 Question 1

1. An individual divides an investment between hedge funds that earn (before fees) -21%, -11%, +21%, +25%, +27%, and +31%. All hedge funds charge 2 plus 20%.

- What is the overall return on the investments?
- How is it divided between the hedge fund and the investor?
- How does your answer change if a fund of funds charging 1 plus 5% is used.

(Assume that a hedge fund's incentive fee of 20% is paid on profits net of the management fee. The fund of fund incentive fee of 5% applies to the total profits, net of management fees, from the hedge funds.)

Assuming that this individual is dividing his/her investment to the listed hedge funds in equal portions, the breakdown of the overall investment return is as followed:

- Overall return, before fee: **12%**, which can be divided into:
 - Fee payment of fund managers: **6.8%**
 - Profit retained by the investor: **5.2%**

If a fund of fund rate of 1 plus 5% is further charged, then the breakdown will be changed:

- Overall return before fee will remain unchanged: **12%**, including:
 - Fee payment of fund managers: **5.51%**
 - (New) fee payment to investment managers (fund of funds): **1.29%**
 - Profit retained by the investor: **5.2%**

Detailed calculation can be seen below:

1.1 Q1 Calculations

```
[ ]: import scipy.stats as sps
import numpy as np
import numpy_financial as npf
import pandas as pd

[ ]: def fee_calculation(pre_fee_profit: float, management_fee: float = 0.02,
    ↪incentive_rate: float = 0.2, verbose = 1):
    if pre_fee_profit <= 0:
        # For un-profitd funds, only management fee applies:
        man_fee = management_fee
        inc_fee = 0
    else:
        # For profit funds:
        man_fee = management_fee
        inc_fee = incentive_rate * (pre_fee_profit - management_fee)
    total_fee = man_fee + inc_fee
    total_profit = pre_fee_profit - total_fee
    if verbose ==1:
        print(f'Fund return (before fees): {100*pre_fee_profit}%, fee at
    ↪{100*management_fee} plus {100*incentive_rate}% | Total fee: {100*total_fee:.
    ↪1f}% | Post-fee return {100*total_profit:.1f}%')
    return (total_fee, total_profit)

[ ]: pre_fee_profits = [-0.21, -0.11, 0.21, 0.25, 0.27, 0.31]
fee = []
overall_return = []

for profit in pre_fee_profits:
    result = fee_calculation(profit)
    fee.append(result[0])
    overall_return.append(result[1])

fund_of_funds = fee_calculation(np.mean(overall_return), 0.01, 0.05, 0)

print('-----')
print(f'overall fund return, before fees: {np.mean(pre_fee_profits)*100:.2f}%')
print(f'overall fund return, after fees: {np.mean(overall_return)*100:.2f}%')
print(f'dividend paid to individual fund managers: {np.mean(fee)*100:.2f}%')
print('-----')
print(f'overall fund return, after hedge fund fees & fund of funds fee: {np.
    ↪mean(fund_of_funds[1])*100:.2f}%')
print(f'dividend paid to individual fund managers: {np.mean(fee)*100:.2f}%')
print(f'dividend paid to investment manager: {np.mean(fund_of_funds[0])*100:.
    ↪2f}%')
```

Fund return (before fees): -21.0%, fee at 2.0 plus 20.0% | Total fee: 2.0% |

Post-fee return -23.0%
Fund return (before fees): -11.0%, fee at 2.0 plus 20.0% | Total fee: 2.0% |
Post-fee return -13.0%
Fund return (before fees): 21.0%, fee at 2.0 plus 20.0% | Total fee: 5.8% |
Post-fee return 15.2%
Fund return (before fees): 25.0%, fee at 2.0 plus 20.0% | Total fee: 6.6% |
Post-fee return 18.4%
Fund return (before fees): 27.0%, fee at 2.0 plus 20.0% | Total fee: 7.0% |
Post-fee return 20.0%
Fund return (before fees): 31.0%, fee at 2.0 plus 20.0% | Total fee: 7.8% |
Post-fee return 23.2%

overall fund return, before fees: 12.00%
overall fund return, after fees: 6.80%
dividend paid to individual fund managers: 5.20%

overall fund return, after hedge fund fees & fund of funds fee: 5.51%
dividend paid to individual fund managers: 5.20%
dividend paid to investment manager: 1.29%

2 Question 2

2. How does Table 7.1 in text change if the principal assigned to the senior, mezzanine, and equity tranche in Figure 7.4 are 72%, 22%, and 6% for the ABS and 70%, 25% and 5% for the ABS CDO?

The updated table can be seen below. Calculations are attached at at the next page.

| | Losses to Subprime Portfolios | Losses to Mezzanine Tranche of ABS | Losses to Equity Tranche of ABS CDO | Losses to Mezzanine Tranche of ABS CDO | Losses to Senior Tranche of ABS CDO |
|---|-------------------------------------|--|---|---|--|
| 0 | 10% | 18% | 100% | 53% | 0% |
| 1 | 15% | 41% | 100% | 100% | 16% |
| 2 | 20% | 64% | 100% | 100% | 48% |
| 3 | 25% | 86% | 100% | 100% | 81% |

2.1 Q2 Calculations

```
[ ]: abs_combo = pd.Series({'senior': 0.72, 'mezzanine': 0.22, 'equity': 0.06})  
    ↪ #AAA, BBB, Equity  
cdo_combo = pd.Series({'senior': 0.70, 'mezzanine': 0.25, 'equity': 0.05})  
    ↪ #AAA, BBB, Equity  
  
losses = [0.1, 0.15, 0.2, 0.25]  
sub_losses = []  
losses_to_tranches = pd.DataFrame()  
  
for loss in losses:  
    # Calculate losses to subprime portfolio:  
    mez_trench_loss = (loss - abs_combo.equity)/abs_combo.mezzanine  
    sub_losses.append(mez_trench_loss)  
    # Calculate losses to ABS CDO:  
    cdo_loss = pd.Series(np.zeros(3), index=['equity', 'mezzanine', 'senior'])  
    loss = mez_trench_loss  
    for tranch in ['equity', 'mezzanine', 'senior']:  
        if loss >= cdo_combo[tranch]:  
            cdo_loss[tranch] = 1  
            loss -= cdo_combo[tranch]  
        else:  
            cdo_loss[tranch] = loss/cdo_combo[tranch]  
            loss = 0  
    losses_to_tranches[mez_trench_loss] = cdo_loss  
  
losses_to_tranches = losses_to_tranches.T.reset_index()  
losses_to_tranches['Losses to Subprime Portfolios'] = losses  
losses_to_tranches = losses_to_tranches.iloc[:, [4,0,1,2,3]]  
losses_to_tranches.rename(columns = {'index': 'Losses to Mezzanine Tranche of',  
    ↪ ABS',  
    'equity': 'Losses to Equity Tranche of ABS',  
    ↪ CDO',  
    'mezzanine': 'Losses to Mezzanine Tranche',  
    ↪ of ABS CDO',  
    'senior': 'Losses to Senior Tranche of ABS',  
    ↪ CDO'}, inplace=True)  
  
def change_data_type(cell):  
    return format(cell, "%.0%")  
losses_to_tranches = losses_to_tranches.map(change_data_type)  
print('Output table for question 2:')  
losses_to_tranches
```

3 Question 3

3. Variable x has a uniform distribution with values between 5 and 15 being equally likely. Variable y has a Pareto distribution. A Gaussian copula is used to define the correlation between the two distributions. The Pareto distribution for variable y has a probability density function:

$$\frac{ac^a}{y^{a+1}}$$

For value of y between c and *infinity* where $c = 4$ and $a = 0.5$. Produce a table similar to Table 9.5 in the text considering values of x equal to 7, 9, 11, and 13 and values of y equal to 5, 10, 30 and 60. Assume a copula correlation of 0.4. A spreadsheet for calculating the cumulative bivariate normal distribution is provided on the author's website www-2.rotman.utoronto.ca/~hull/riskman.

Mapping of given values to standard normal distribution is done below. Further calculation of cumulative joint probability distribution of both values under bivariate normal distribution with a copula correlation of 0.4 is done in the given spreadsheet, with the result attached below:

| | Y | | | |
|-----|-------|-------|-------|-------|
| X | 5 | 10 | 15 | 30 |
| 0.2 | 0.046 | 0.119 | 0.167 | 0.180 |
| 0.4 | 0.072 | 0.208 | 0.312 | 0.346 |
| 0.6 | 0.089 | 0.279 | 0.442 | 0.498 |
| 0.8 | 0.100 | 0.334 | 0.553 | 0.634 |

3.1 Q3 Calculations

```
[ ]: x = [7, 9, 11, 13]
     y = [5, 10, 30, 60]

     x_dist = sps.uniform(loc=5, scale=15-5)
     y_dist = sps.pareto(b=0.5, scale = 4) #scale = c
     std_norm = sps.norm

[ ]: x_percentile = [x_dist.cdf(x_val) for x_val in x]
     x_map = [std_norm.ppf(x_perc) for x_perc in x_percentile]

     y_percentile = [y_dist.cdf(y_val) for y_val in y]
     y_map = [std_norm.ppf(y_perc) for y_perc in y_percentile]

     print(f'Percentile of distribution of x values: {x_percentile}')
     print(f'Mapped x values to std. normal dist.: {x_map}')
     print(f'Percentile of distribution of y values: {y_percentile}')
     print(f'Mapped y values to std. normal dist.: {y_map}')
```

Percentile of distribution of x values: [0.2, 0.4, 0.6, 0.8]
Mapped x values to std. normal dist.: [-0.8416212335729142,
-0.2533471031357997, 0.2533471031357997, 0.8416212335729143]
Percentile of distribution of y values: [0.10557280900008414,
0.3675444679663241, 0.6348516283298893, 0.7418011102528388]
Mapped y values to std. normal dist.: [-1.2504214910006977,
-0.33836395189680224, 0.34473082330391736, 0.6489080990191797]

4 Question 4

4. Five years of history for the S&P 500 is attached. March 2020 was a volatile period for the index. Imagine that it is March 13, 2020. Use the previous 251 days (250 percentage changes) to calculate the one-day value at risk and expected shortfall for a portfolio with \$1000 invested in the index. Ignore dividends. Provide results for four different methods:
- a) The basic historical simulation approach
 - b) Exponential weighting with $\lambda = 0.995$
 - c) Volatility scaling with $\lambda = 0.94$ (assume an initial variance equal to the sample variance for the 250 changes)
 - d) Extreme value theory with $u = 25$

For all calculations below, we assume a **VaR Confidence level of 99%**. All calculations are done in python with `scipy` and `numpy` (as attached below) with the exception of the very last part, which is done using Excel Solver.

- Under the basic historical simulation approach, the **VaR value is \$48.90**, and the **expected shortfall is \$85.55**. VaR corresponds to the 3rd highest cost scenario, which is scenario 248 detailing the change from Mar 10 to Mar 11 2020 (4.9% decrease).
- Under the exponential weighting approach with $\lambda = 0.995$, the **VaR value is \$76.00**, and the **expected shortfall is \$89.30**. VaR corresponds to the 2nd highest cost scenario, which is scenario 246 detailing the change from Mar 8 to Mar 9 2020 (7.6% decrease).
- Under the volatility scaling approach with $\lambda = 0.94$, the **VaR value is \$137.70**, and the **expected shortfall is \$183.74**.
- Under the extreme value approach with $\mu = 25$, the **VaR value is 49.28** dollars, and the **expected shortfall is 79.98** dollars. There are 11 scenarios where the loss is greater than the given threshold of 25 dollars. Under such configuration, the solver finds the most optimal B to be 12.739, and the most optimal ξ to be 0.3268,

4.1 Q4 Calculations

```
[ ]: original = pd.read_excel('8301_S&P_250days.xlsx')
     scenarios = pd.DataFrame()

[ ]: scenarios['change'] = original['Change %'].iloc[1:]
     scenarios['loss'] = [1000 * (1*(-change)) for change in scenarios.change]
     scenarios_unsorted = scenarios.copy(deep=True)

     scenarios.sort_values(by='loss', ascending=True, inplace=True)
     basic_var = np.percentile(scenarios.loss, 99)
     basic_var_man = scenarios.loss.iloc[int(250*0.99)]
     basic_es_man = scenarios[scenarios.loss > basic_var_man].loss.mean()

     print(f'Value at risk - basic historical simulation approach with 95%_
           ↪confidence level: ${basic_var_man:.2f}')
     print(f'Expected shortfall - basic historical simulation approach with 95%_
           ↪confidence level: ${basic_es_man:.2f}')
```

Value at risk - basic historical simulation approach with 95% confidence level:
\$48.90

Expected shortfall - basic historical simulation approach with 95% confidence
level: \$85.55

```
[ ]: num = scenarios.shape[0]
     lbda = 0.995

     scenarios['weight'] = [(pow(lbda,num-index)*(1-lbda))/(1-pow(lbda,num)) for_
           ↪index in scenarios.index.to_list()]
     scenarios['weight_acc'] = [scenarios.weight.iloc[0:i+1].sum() for i in_
           ↪range(scenarios.shape[0])]
     ew_var = scenarios[scenarios.weight_acc >= 0.99].iloc[0].loss
     ew_es_df = scenarios[scenarios.loss >= ew_var]
     ew_es_man = 0

     for i, row in ew_es_df.reset_index(drop=True).iterrows():
         if i == 0:
             residual_weight = 0.01 - ew_es_df.iloc[i+1:].weight.sum()
             ew_es_man += residual_weight * row.loss
             #print(residual_weight, ew_es_man)
         else:
             ew_es_man += row.weight * row.loss
             #print(row.weight, ew_es_man)

     ew_es_man = ew_es_man/0.01

     print(f'Value at risk - exponential weighting: ${ew_var:.2f}')
     print(f'Expected shortfall - exponential weighting: ${ew_es_man:.2f}')
```

Value at risk - exponential weighting: \$76.00
Expected shortfall - exponential weighting: \$89.30

```
[ ]: lbda_vol = 0.94
scenarios_vol = scenarios_unsorted.copy(deep=True)
variance_daily = np.zeros_like(scenarios_vol.loss)
# EWMA volatility estimation:
for i in range(0, len(variance_daily)):
    if i == 0: # for the first row of scenarios_unsorted
        variance_daily[i] = np.var(scenarios_vol.change)
        #print(np.var(scenarios_vol.change))
    else: # variance is 0-indexed, scenarios_vol is 1-indexed # for the second
    ↪row upwards
        variance_daily[i] = lbda_vol * variance_daily[i-1] + (1-lbda_vol) *
    ↪(scenarios_vol.iloc[i-1].change**2)
        #print(scenarios_vol.iloc[i-1].change, variance_daily[i])

volatilities = np.sqrt(variance_daily)
scenarios_vol['variance'] = variance_daily
scenarios_vol['volatility'] = volatilities

# Apply volatility scaling to original loss value:
scenarios_vol['loss_vol'] = [(scenarios_vol.loc[i].loss * (scenarios_vol.
    ↪iloc[-1].volatility/scenarios_vol.loc[i].volatility)) for i in scenarios_vol.
    ↪index.to_list()]

svol2 = scenarios_vol.sort_values(by='loss_vol', ascending=True)
vol_var_man = svol2.iloc[int(250*0.99)].loss_vol
vol_es_man = scenarios_vol[scenarios_vol.loss_vol > vol_var_man].loss_vol.mean()

print(f'Value at risk - volatility scaling approach with 95% confidence level:
    ↪${vol_var_man:.2f}')
print(f'Expected shortfall - volatility scaling approach with 95% confidence
    ↪level: ${vol_es_man:.2f}')
```

Value at risk - volatility scaling approach with 95% confidence level: \$137.70
Expected shortfall - volatility scaling approach with 95% confidence level:
\$183.74

```
[ ]: scenarios_eva = scenarios_unsorted.copy(deep=True)
scenarios_eva.sort_values(by='loss', ascending=False, inplace=True)
scenarios_eva.to_excel('scenarios_eva.xlsx')
# Rest of the calculation done in provided excel solver
```