portfolio_construction_with_bond&stock&options

February 8, 2021

0.1 1. Import packages

[92]:
```
import numpy as np
from scipy.optimize import linprog
```

0.2 2. set parameters

[93]:
```
# Get Prices and future prices, each has 1/3 probability
stock_price = 20
stock_prices_future = np.transpose(np.array([[40,20,12]]))
bond_price = 90
bond_prices_future = np.transpose(np.array([[100,100,100]]))

option1_price = 10
option2_price = 6
option1_prices_future = np.transpose(np.array([[25,5,0]])) #strike 15
option2_prices_future = np.transpose(np.array([[20,0,0]])) #strike 20
```

0.3 3. calculation

[95]:
```
# Get expected profits
expected_profit_stock = np.mean(stock_prices_future-stock_price)
expected_profit_bond = 10
expected_profit_option1 = np.mean(option1_prices_future - option1_price)
expected_profit_option2 = np.mean(option2_prices_future - option2_price)


# get returns for stock , bond and options
stock_returns = (stock_prices_future - stock_price)/stock_price;
bond_returns = (bond_prices_future - bond_price)/bond_price;
option1_returns =(option1_prices_future - option1_price)/option1_price;
option2_returns =(option2_prices_future - option2_price)/option2_price;

# expected returns for stock
expected_stock_return = np.mean(stock_returns)
expected_bond_return = np.mean(bond_returns)
expected_option1_return = np.mean(option1_returns)
```

```python
expected_option2_return = np.mean(option2_returns)
print('expected_stock_return: ',expected_stock_return)
print('expected_bond_return: ',expected_bond_return)
print('expected_option1_return: ',expected_option1_return)
print('expected_option2_return: ',expected_option2_return)
```

```
expected_stock_return:  0.19999999999999998
expected_bond_return:  0.1111111111111111
expected_option1_return:  0.0
expected_option2_return:  0.11111111111111116
```

0.4 4.solve linear program maximizing expected profit

[103]:
```python
# define the function to maximize
# why negative here? Because linprog is used to minimize
f = [-1*expected_profit_stock, -1*expected_profit_bond,␣
 ↪-expected_profit_option1 ,-expected_profit_option2]
A = [[20, 90, 10, 6]]
b = [50000]
x0_bounds = (0, None)
x1_bounds = (0, None)
x2_bounds = (-6000, None)
x3_bounds = (-6000, None)
result = linprog(f, A_ub=A, b_ub=b, bounds=[x0_bounds, x1_bounds, x2_bounds,␣
 ↪x3_bounds])
```

[104]:
```python
f
```

[104]:
```
[-4.0, -10, -0.0, -0.6666666666666666]
```

[105]:
```python
print(result)
```

```
     con: array([], dtype=float64)
     fun: -25199.984437743547
 message: 'Optimization terminated successfully.'
     nit: 5
   slack: array([0.07528294])
  status: 0
 success: True
       x: array([ 7.29999598e+03,  4.13270180e-05, -5.99999995e+03,
-5.99999984e+03])
```

0.5 5. More Computation for cost, profit, risk and sharpe

[106]:
```
# what is expected profit?
expected_profit =  -np.dot(np.array(f),np.array(result.x));
expected_profit
```

[106]:
```
25199.984437743547
```

[107]:
```
# compute cost of portfolio
cost = np.dot(np.array([stock_price,bond_price, option1_price, option2_price]),␣
↪np.array(result.x))
print(cost)
```

```
49999.924717059126
```

[111]:
```
# computation of Risk
future_prices = np.transpose(np.array([stock_price_future,np.
↪squeeze(bond_price_future),
                                        np.squeeze(option1_prices_future),  np.
↪squeeze(option2_prices_future)]))
future_prices # the four asset's future price
```

[111]:
```
array([[ 40, 100,  25,  20],
       [ 20, 100,   5,   0],
       [ 12, 100,   0,   0]])
```

[112]:
```
portfolio_prices_future = np.dot(future_prices, np.array(result.x))
print('portfolio_prices_future:  ',portfolio_prices_future)

portfolio_returns = (portfolio_prices_future- cost)/cost
print('portfolio_returns:  ',portfolio_returns)
```

```
portfolio_prices_future:   [ 21999.84760369 115999.92396603  87599.95589469]
portfolio_returns:   [-0.56000239  1.32000197  0.75200176]
```

0.5.1 In python np.std use n as denominator not n-1, matlab std use n-1 that's where the difference!

[113]:
```
# In python np.std use n as denominator not n-1, matlab std use n-1 that's␣
↪where the difference!
mu = np.mean(portfolio_returns)
sigma = np.std(portfolio_returns)

# calculate sharpe!
sharpe = mu/sigma
print('sharpe', sharpe)
```

```
sharpe 0.6401732544952227
```

0.6 Here we should use n as denominator because the variance we calculate here is not an estimation

[114]:
```
np.sqrt(np.sum((portfolio_returns-mu)**2/3))
```

[114]: 0.7872875726509282

[115]:
```
sigma
```

[115]: 0.7872875726509282

0.7 Why we should choose only stock? [2.5000000e+03, 1.0383546e-07], because stock has a higher return expected rate