portfolio_construction_with_bond&stock

February 8, 2021

0.1 1. Import packages

[32]:
```
import numpy as np
from scipy.optimize import linprog
```

0.2 2. set parameters

[10]:
```
# 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]]))
```

0.3 3. calculation

[45]:
```
# Get expected profits
expected_profit_stock = np.mean(stock_prices_future-stock_price)
expected_profit_bond = 10

# get returns for stock and bond
stock_returns = (stock_prices_future - stock_price)/stock_price;
bond_returns = (bond_prices_future - bond_price)/bond_price;

# expected returns for stock
expected_stock_return = np.mean(stock_returns)
expected_bond_return = np.mean(bond_returns)
print('expected_stock_return: ',expected_stock_return)
print('expected_bond_return: ',expected_bond_return)
```

```
expected_stock_return:  0.19999999999999998
expected_bond_return:  0.1111111111111111
```

0.4 4.solve linear program maximizing expected profit

[46]:
```
# define the function to maximize
# why negative here? Because linprog is used to minimize
f = [-1*expected_profit_stock, -1*expected_profit_bond]
```

```
A = [[20, 90]]
b = [50000]
x0_bounds = (0, None)
x1_bounds = (0, None)
result = linprog(f, A_ub=A, b_ub=b, bounds=[x0_bounds, x1_bounds])
```

[47]:
```
print(result)
```

```
     con: array([], dtype=float64)
     fun: -9999.999985319324
 message: 'Optimization terminated successfully.'
     nit: 5
   slack: array([6.92499598e-05])
  status: 0
 success: True
       x: array([2.5000000e+03, 1.0383546e-07])
```

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

[50]:
```
# what is expected profit?
expected_profit =  -np.dot(np.array(f),np.array(result.x));
expected_profit
```

[50]:
```
9999.999985319324
```

[59]:
```
# compute cost of portfolio
cost = np.dot(np.array([stock_price,bond_price]), np.array(result.x))
print(cost)
```

```
49999.99993075004
```

[63]:
```
np.squeeze(bond_price_future)
```

[63]:
```
array([100, 100, 100])
```

[66]:
```
# computation of Risk

portfolio_prices_future = np.dot(np.transpose(np.array([stock_price_future,np.
 ↪squeeze(bond_price_future)])), 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:   [99999.99985319 49999.99993179 29999.99996323]
portfolio_returns:   [ 1.00000000e+00  2.07671838e-11 -4.00000000e-01]
```

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

[67]:
```
# 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 0.33968311027865006
```

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

[89]:
```
np.sqrt(np.sum((portfolio_returns-mu)**2/3))
```

[89]: 0.5887840576451439

[91]:
```
sigma
```

[91]: 0.5887840576451439

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