금융공학 프로그래밍3 기말고사 대체과제 (2024 Spring)

1. Markowitz의 평균 분산 모델에 따르면 $k$개의 자산으로 구성된 포트폴리오에서 주어진 기대수익률에 대해 분산이 가장 작아지게 하는 각 자산 별 투자비중은 아래와 같이 도출할 수 있다.

① $X_1, \dots, X_k$은 각 자산 별 수익률을 나타내는 유한한 분산 값을 가지는 확률변수로, $X_1, \dots, X_k$의 공분산행렬을 $Q_{k\times k}$, 기대값 벡터는 $r_{k\times 1} = (r_1, \dots, r_k) = (E[X_1], \dots, E[X_k])$라고 하자.

② 각 자산 별 투자비중을 $\mathrm{w}_{k\times 1} = (w_1, \dots, w_k)$ (단, $\vec{1}^T w = \sum w_i = 1$, $\vec{1}_{k\times 1} = (1,1,\dots,1)$)로 하는 포트폴리오 $P$를 가정할 때, 포트폴리오 $P$ 의 기대수익률은 $\mu_p = r^T w = \sum w_i r_i$, 분산은 $\sigma_p^2 = w^T Q w$가 된다.

③ 이 경우 다음을 만족하는 $w$를 찾는 최적화 문제를 고려하자.

$$\min\{w^T Q w \mid w \in \mathcal{R}^k, \quad \vec{1}^T w = 1, \quad r^T w = \mu_p\}$$

즉 포트폴리오의 기대수익률 $\mu_p$이 원하는 수준에서 주어진 상태에서 포트폴리오의 분산 $w^T Q w$를 최소로 하기 위해 필요한 각 자산 별 가중치 $w = (w_1, \dots, w_k)$를 찾는 문제이다.

④ 이는 선형제약식을 가진 2차 함수에 대한 최적화 문제로, 라그랑지 이론에 따르면 아래와 같은 선형방정식계를 이용하여 풀 수 있다.

$$\begin{bmatrix} Q & \vec{1} & r \\ \vec{1}^T & 0 & 0 \\ r^T & 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ \mu_p \end{bmatrix}$$

(1) 이상의 내용을 수행하기 위한 파이썬 함수 MVportfolio를 작성하여라. 함수의 인자는 k개의 자산 별 가격을 m일 동안 일별로 수집한 $\mathrm{m} \times \mathrm{k}$의 데이터프레임 asset과, 원하는 포트폴리오 기대수익률 ($\mu_p$)인 mu_p이고, 함수를 실행한 결과 k개의 자산별 가중치 $w = (w_1, \dots, w_k)$와 포트폴리오 분산 $\sigma_p^2$ 값이 반환되어야 한다. 단, 각 자산 별 일별 수익률은 log(해당일가격/전일가격)으로 구하여라.

(2) 5개의 자산 가격을 일별로 수집한 'it.csv'의 자료에 (1)에서 작성한 MVportfolio 함수를 적용하여, $\mu_p$가 -0.001에서 0.0010사이의 값을 가질 때 최적의 투자비중으로 계산된 분산 $\sigma_p^2$를 도출한 뒤, 이를 이용하여 효율적인 투자선 (가로축은 $\sigma_p^2$, 세로축은 $\mu_p$인 선그림)을 도출하여라.

2. 다음은 12개월 모멘텀을 이용하여 10개의 종목을 선택하는 문제이다.

    (1) 'price.csv' 파일을 불러온 뒤, 'date' 열이 DatetimeIndex 자료형이 되도록 변경한 뒤, 이를 Index로 하는 pd.DataFrame 객체 price를 생성하여라.

    (2) price에서 2019년도에 해당하는 자료만 선택하여 price_sub 로 저장하여라.

    (3) price_sub에서 각 종목 별(열 별)로 일별 수익률(=1+변화율)에 대한 누적곱으로 누적 수익률을 구한 뒤 cum_ret라는 pd.Series 객체로 저장하여라.

    (4) cum_ret에서 누적수익률이 높은 순서로 10개 종목을 출력한 뒤, 가장 누적수익률이 높은 종목에 대한 2019년 price의 시계열 그림을 출력하여라.

    (5) price_sub에서 각 종목 별(열 별)로 일별 변화율에 대한 표준편차에 252의 제곱근을 곱하여, 각 종목 별 연율화 변동성을 구한 뒤 이를 std라는 pd.Series 객체로 저장하여라.

    (6) std에서 std가 0인 경우와 NaN인 경우를 제외하여라.

    (7) 종목 별 cum_ret와 std의 비율로 정의되는 위험조정 수익률 노게 (= cum_ret / std)를 pd.Series 객체로 저장하여라.

    (8) 위험조정 수익률 shrp에 NaN이 포함된 경우는 이를 shrp의 최소값으로 대체하여라.

    (9) 위험조정 수익률 shrp의 값이 큰 순서대로 10개 종목을 선택하여라.

    (10) 선택된 10개 종목을 index로 하고, 10개 종목에 대한 누적수익률(cum_ret 값), 변동성(std 값), 위험조정수익률(shrp 값)을 column으로 하는 pd.DataFrame 객체 final_result를 생성한 뒤 출력하여라.

3. 유러피안 콜 옵션 가격 몬테카를로 시뮬레이션 다음 ECallSimul_1은 유러피안 콜 옵션 가격을 몬테카를로 시뮬레이션으로 도출하기 위한 함수를 작성한 것이다. (산출식, 변수명 등은 ch3 강의에서 다룬 내용과 동일함)

```
def ECallSimul_1(S0, K, T, r, sigma, M, I=250000):
    import math
    import random
    S = []
    dt = T/M
    for i in range(I):
        path = []
        for t in range (M+1):
            if t == 0 :
                path.append(S0)
            else:
                z=random.gauss(0., 1.)
                St = path[t-1] * math.exp( (r-0.5*sigma**2)*dt + sigma * math.sqrt(dt)*z )
                path.append(St)
        S.append(path)
    sum_val = 0.0
     for path in S:
        sum_val += max(path[-1]-K, 0)
    C0 = math.exp(-r*T)*sum_val/I
    return ( round(C0, 3) )
```

(1) ECallSimul_1 함수의 body에서 가능한 모든 부분을 Numpy의 기능을 활용하는 것으로 수 정한 뒤 ECallSimul_2라는 함수로 저장하여라.

(2) S0=100., K=105., T=1., r=0.05, sigma=0.2, M=50, I=250000인 경우에 대하여 두 함수 ECallSimaul_1과 ECallSimaul_2의 결과가 유사한 지 확인해 보고, 두 함수의 연산시간을 비교하여라. (※ Jupyter Notebook에서 %time이라는 매직 명령어 뒤에 실행하고자 하는 코 드를 작성하면 그 코드를 실행하는데 소요된 CPU time을 출력할 수 있음. )

<결과 예시>

```
In [3]: %time ECallSimul_1(S0=100., K=105., T=1., r=0.05, sigma=0.2, M=50, I=250000)
        Wall time: 12.9 s
Out[3]: 8.01

In [4]: %time ECallSimul_2(S0=100., K=105., T=1., r=0.05, sigma=0.2, M=50, I=250000)
        Wall time: 747 ms
Out[4]: 7.987
```

→ ECallSimal_1의 경우 12.9 seconds, ECallSimul_2의 경우 747 milliseconds로 Numpy를 활용한 경우 연산속도가 대략 17배 빠름을 확인할 수 있음.