금공프3 중간대체과제

MFE 20249433 최재필

In [ ]:
```
import numpy as np

import math
```

1. 채권 가격과 듀레이션

(1)

** 문제에 제시된 공식이 잘못되어 아래와 같이 수정했습니다.

- 채권 가격:
  - 마지막 기에 $\frac{FV}{(1+\frac{(y/100)}{f})^t}$ 더해줘야 함. (만기 원금)
- 듀레이션:
  - t가 아닌 $\frac{t}{f}$ 를 곱해줘야 함.
  - 또한 채권 가격과 마찬가지로 만기 원금을 더해줘야 함.

즉,

$$D = \frac{1}{P}(\sum_{t=1}^{n} \frac{t}{f} \cdot \frac{C_t}{(1+\frac{(y/100)}{f})^t} + \frac{n}{f} \cdot \frac{FV}{(1+\frac{(y/100)}{f})^n})$$

로 고쳐져야 합니다.

위와 같이 고쳤을 때 테스트 케이스와 일치하는 값이 반환됩니다.

In [ ]:
```python
def bondftn(facevalue, couprate, y, maturity, frequency):
    """계산된 채권가격과 듀레이션을 튜플로 반환하는 함수

    Args:
        facevalue (float): 액면가격
        couprate (float): 쿠폰이자율
        y (float): 만기수익률
        maturity (float): 만기
        frequency (float): 연간쿠폰지급횟수

    Returns:
        tuple: (채권가격, 듀레이션)
    """
    frequencies = {
        'annual': 1,
        'semi-annual': 2,
        'quarterly': 4,
    }

    if frequency in frequencies:
        f = frequencies[frequency]
    else:
        print(f'Invalid frequency: {frequency}')
        return

    c = couprate / 100
    ytm = y / 100
    c_dollar = facevalue * c / f
    nper = maturity * f

    ## 채권 가격
    P = 0
    for t in range(1, nper+1):
        P += c_dollar / (1 + ytm/f)**t

    P += facevalue / (1 + ytm/f)**t

    ## 듀레이션
    D = 0
    for t in range(1, nper+1):
```

```
        D += t/f * ( c_dollar / (1 + ytm/f)**t )

    D += t/f * ( facevalue / (1 + ytm/f)**t )
    D = D/P

    return P, D
```

In [ ]:
```
test_case = {
    'facevalue': 100,
    'couprate': 5,
    'y': 4.5,
    'maturity': 2,
    'frequency': 'quarterly',
}
```

In [ ]:
```
bondftn(**test_case)
```

Out[ ]:
```
(100.95121625257656, 1.9161694881599696)
```

(2)

In [ ]:
```
def price_change(facevalue, couprate, y_old, y_new, maturity, frequency):
    """만기수익률 변화에 따른 가격변화율을 계산하는 함수

    Args:
        y_old (float): 변화 전 만기수익률
        y_new (float): 변화 후 만기수익률

    Returns:
        float: 가격변화율
    """
    old_price = bondftn(facevalue, couprate, y_old, maturity, frequency)[0]
    new_price = bondftn(facevalue, couprate, y_new, maturity, frequency)[0]

    return (new_price - old_price) / old_price
```

In [ ]:
```
y_old = 10
y_new = 11
frequency = 'annual'
facevalue = 100

result_dict = {}

test_maturities = [5, 4, 3, 2, 1]
test_couprates = [5, 4, 3, 2, 1]

for m in test_maturities:
    result_dict[f'M={m}'] = {}
    for c in test_couprates:
        result_dict[f'M={m}'][f'{c}%'] = price_change(facevalue=facevalue, couprate=c, y_old=y_old, y_new=y_new, maturity=m, f
```

In [ ]:
```
result_dict
```

Out[ ]:
```
{'M=5': {'5%': -0.03974836055305566,
  '4%': -0.04047048346784374,
  '3%': -0.041267129839987905,
  '2%': -0.04215046362141572,
  '1%': -0.04313544833326965},
 'M=4': {'5%': -0.03286185142470099,
  '4%': -0.03331563660900725,
  '3%': -0.033806394433436325,
  '2%': -0.034338835372434776,
  '1%': -0.0349185055695231 2},
 'M=3': {'5%': -0.025444064500651814,
  '4%': -0.0256807665472403,
  '3%': -0.0259317228242334,
  '2%': -0.026198260892201106,
  '1%': -0.026481878449181942},
 'M=2': {'5%': -0.01749248331124319,
  '4%': -0.017574470850625305,
  '3%': -0.01765969778478984,
  '2%': -0.01774835996965999,
  '1%': -0.017840669374671377},
 'M=1': {'5%': -0.009009009009008976,
  '4%': -0.009009009009009075,
  '3%': -0.009009009009009023,
  '2%': -0.009009009009008973,
  '1%': -0.009009009009008919}}
```

In [ ]:
```
result_dict['M=5']['5%']
```

Out[ ]:
```
-0.03974836055305566
```

(3)

In [ ]:
```
result_dict_dur = {}

for m in test_maturities:
    result_dict_dur[f'M={m}'] = {}
    for c in test_couprates:
        result_dict_dur[f'M={m}'][f'{c}%'] = bondftn(facevalue=facevalue, couprate=c, y=y_old, maturity=m, frequency=frequency
```

In [ ]:
```
result_dict_dur['M=5']['4%']
```

Out[ ]:
```
4.570186239555571
```

2. 자동차 보험회사에 관한 몬테카를로 시뮬레이션

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
# poisson (연간청구건수)
poi_mean = 100

# gamma (청구건수 별 청구금액)
alpha = 2 # 모양
beta = 1/2 # 척도

# uniform (청구건수 별 청구발생시점)
start = 0
end = 1

# 보험료 수입
slope = 150
```

(1)

In [ ]:
```
# 연간 청구 건수를 포아송 분포에서 샘플링
poisson_samples = np.random.poisson(lam=poi_mean, size=10000)

case_count = np.random.choice(poisson_samples, 1)[0]
case_count
```

Out[ ]:
```
114
```

In [ ]:
```
# 청구 건수별로 청구금액을 감마 분포에서 샘플링
claims = np.random.gamma(alpha, scale=beta, size=case_count)
```

In [ ]:
```
# 청구 건수별 청구 발생시점을 균등 분포에서 샘플링
times = np.random.uniform(start, end, size=case_count)
```

In [ ]:
```
sort_idx = np.argsort(times) # 시간순으로 정렬하기 위한 인덱스

claims_timeseries = claims[sort_idx]
times_timeseries = times[sort_idx]
revenue_timeseries = slope * times_timeseries # 보험료 수입

cumulative_claims_timeseries = np.cumsum(claims_timeseries) # 누적 청구금액
balance_timeseries = revenue_timeseries - cumulative_claims_timeseries # 누적 수입 - 누적 청구금액
```

In [ ]:
```
balance = np.insert(balance_timeseries, 0, 0) # 첫 번째 값은 0으로 삽입
balance
```

Out[ ]:
```
array([ 0.        , -0.26227715, -1.6180239 , -1.82463633, -1.53089757,
       -2.46839791, -2.20997289, -1.2079736 , -1.47598418, -0.86802952,
        0.19901351, -0.71986213,  1.29431308,  0.71045542,  1.42182522,
        3.96899387,  3.22860918,  5.0048229 ,  9.23842309, 11.96273557,
       11.72405712, 12.87905416, 13.30443201, 13.43980583, 12.52498499,
       11.60429671, 14.53106592, 15.10440251, 17.28381553, 16.5711021 ,
       19.22145751, 19.24898417, 20.12286951, 19.16216323, 19.15369614,
       18.92150726, 18.45788688, 19.85570357, 21.03005458, 21.98517586,
       24.8563027 , 29.46789881, 29.41446588, 28.73813918, 27.05527385,
       26.31907777, 28.127566  , 27.25316095, 26.65294555, 27.72929123,
       26.05134375, 26.22319595, 26.25914594, 27.91199516, 27.14190793,
       27.8215918 , 28.00206147, 27.38688218, 26.75384067, 26.46275544,
       24.61527075, 25.25216238, 24.9845234 , 24.51949075, 26.03631959,
       26.64880291, 25.63082927, 24.0519857 , 25.91985378, 26.84760604,
       26.84612513, 26.44281845, 27.75413662, 28.74075064, 29.94714107,
       30.55295038, 29.34245207, 28.92148218, 30.47504904, 28.61465751,
       31.29143915, 30.75692533, 33.80380902, 32.56890702, 33.56630241,
       33.77149224, 33.89648082, 33.54392874, 32.54996614, 31.3620031 ,
       32.06302275, 36.68756394, 35.88979524, 36.20364485, 36.16121612,
       33.64857627, 33.23687058, 34.65241891, 34.45934454, 35.4917505 ,
       38.95738146, 38.24324673, 39.04172957, 39.15510184, 39.18276481,
       38.07176085, 37.8513551 , 36.45430046, 35.0233919 , 36.33094952,
       36.37338146, 38.95739864, 41.89933844, 41.62230624, 45.49691662])
```

(2)

In [ ]:
```python
def generate_balance_path(
        poisson_size=10000,
        poi_mean=100,
        alpha=2,
        beta=1/2,
        start=0,
        end=1,
        slope=150
        ):
    """Monte Carlo 실험을 위해 balance의 path를 generate하는 함수

    Returns:
        np.ndarray: 잔고의 path
    """

    # 연간 청구 건수를 포아송 분포에서 샘플링
    poisson_samples = np.random.poisson(lam=poi_mean, size=poisson_size)
    case_count = np.random.choice(poisson_samples, 1)[0]

    # 청구 건수별로 청구금액을 감마 분포에서 샘플링
    claims = np.random.gamma(alpha, scale=beta, size=case_count)

    # 청구 건수별 청구 발생시점을 균등 분포에서 샘플링
    times = np.random.uniform(start, end, size=case_count)

    sort_idx = np.argsort(times) # 시간순으로 정렬하기 위한 인덱스

    claims_timeseries = claims[sort_idx]
    times_timeseries = times[sort_idx]
    revenue_timeseries = slope * times_timeseries # 보험료 수입

    cumulative_claims_timeseries = np.cumsum(claims_timeseries) # 누적 청구금액
    balance_timeseries = revenue_timeseries - cumulative_claims_timeseries # 누적 수입 - 누적 청구금액

    balance = np.insert(balance_timeseries, 0, 0) # 첫 번째 값은 0으로 삽입

    return balance
```

(a)

In [ ]:
```
num_experiments = 10000

# 최종 balance만 generate
simulate_final_balance = [generate_balance_path()[-1] for _ in range(num_experiments)]
```

In [ ]:
```
# balance의 기대값
np.mean(simulate_final_balance)
```

Out[ ]:
```
48.34827097170984
```

(b)

In [ ]:
```
# balance path들을 generate
balance_paths = [generate_balance_path() for _ in range(num_experiments)]
```

In [ ]:
```
# 1년 중 한 번 이상 -5 이하로 떨어질 확률
p = np.mean([np.any(balance <= -5) for balance in balance_paths])
p
```

Out[ ]:
```
0.0633
```