금공프3 중간대체과제

MFE 20249433 최재필

In [ ]:
```
import numpy as np
```

1. 채권 가격과 듀레이션

(1)

** 문제에 제시된 공식을 아래와 같이 구현하였습니다.

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

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 수정.

In [ ]:
```
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 = {}
```

```python
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,
            frequency=frequency,
            )
```

In [ ]:

```python
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%': -0.0333156366090725,
  '3%': -0.033806394433436325,
  '2%': -0.034338835372434776,
  '1%': -0.03491850556952312},
 '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
                )[1]
```

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

Out[ ]: 4.570186239555571

2. 자동차 보험회사에 관한 몬테카를로 시뮬레이션

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[ ]: 110

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([ 0.00000000e+00, -1.07972074e+00,  1.50886162e+00,  4.28157738e-01,
       -9.42877741e-01, -1.44171883e+00, -9.32929195e-01, -1.35647123e+00,
       -3.29282363e+00, -4.37979469e+00, -3.62834379e+00, -2.84995774e+00,
       -2.90561369e+00, -2.42744123e+00, -3.45143331e+00, -3.37311821e+00,
       -1.61996981e+00, -1.07685558e+00, -1.21732346e+00, -2.22586904e+00,
       -2.72724252e+00, -1.71853579e+00, -2.19973083e+00, -2.25026123e+00,
        4.52829109e-01, -2.67348317e-02,  2.47443609e-02, -5.90084108e-02,
        4.61792787e-01,  4.34099905e+00,  4.14820372e+00,  3.39836424e+00,
        3.23019627e+00,  3.98504065e+00,  4.55653517e+00,  3.48609245e+00,
        5.26689317e+00,  9.84552973e+00,  9.54694617e+00,  1.05533305e+01,
        9.35558806e+00,  8.36645021e+00,  1.06454623e+01,  1.14455226e+01,
        1.27676591e+01,  1.39382022e+01,  1.08924490e+01,  9.41360963e+00,
        9.26728015e+00,  8.03465964e+00,  9.26701969e+00,  1.26158893e+01,
        1.37150396e+01,  1.39912695e+01,  1.36993478e+01,  1.37852480e+01,
        1.28738881e+01,  1.31313303e+01,  1.48969555e+01,  1.50030808e+01,
        1.49417742e+01,  1.47451857e+01,  1.47463252e+01,  1.43020762e+01,
        1.46992082e+01,  1.49197317e+01,  1.57139045e+01,  1.55582151e+01,
        1.56387909e+01,  1.40948365e+01,  1.72545298e+01,  1.85903851e+01,
        1.93982885e+01,  2.03317742e+01,  2.21420232e+01,  2.55934531e+01,
        2.67844068e+01,  2.97919817e+01,  3.10641374e+01,  3.03085791e+01,
        3.19292345e+01,  3.14420930e+01,  3.33117413e+01,  3.27176518e+01,
        3.75026947e+01,  3.72408553e+01,  3.76503579e+01,  3.76218412e+01,
        3.64273332e+01,  3.57591655e+01,  3.45083871e+01,  3.36211937e+01,
        3.29620050e+01,  3.21953560e+01,  3.18713019e+01,  3.15154514e+01,
        3.36727237e+01,  3.44396580e+01,  3.39766569e+01,  3.52317077e+01,
        3.49824029e+01,  3.41152045e+01,  3.44138322e+01,  3.43781050e+01,
        3.54046971e+01,  3.47350033e+01,  3.37686576e+01,  3.54831670e+01,
        3.60020244e+01,  3.59110720e+01,  3.53188497e+01])
```

(2)

In [ ]:
```python
def generate_balance_path(
        poisson_size=10000,
        poi_mean=100,
        alpha=2,
        beta=1/2,
        start=0,
        end=1,
        slope=150
```

```python
    ):
    """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 [ ]:
```python
num_experiments = 10000

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

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

Out[ ]: 48.46796707755993

(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[ ]: 0.0694