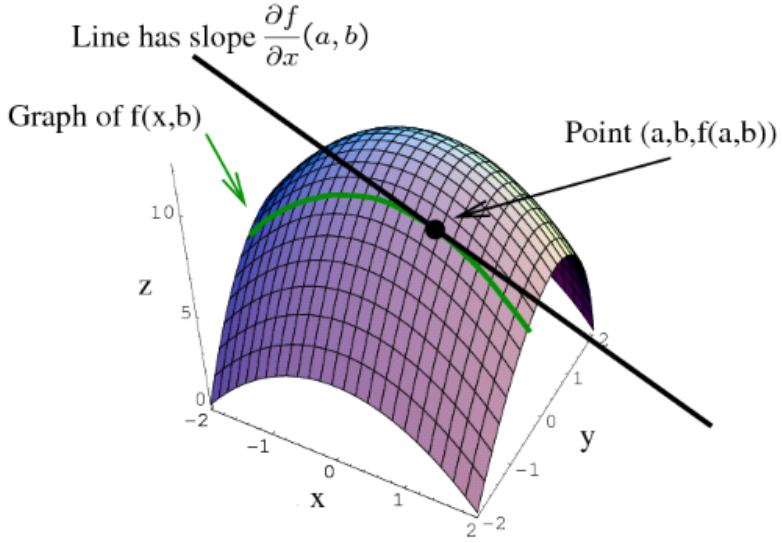
# Python으로 파생상품 가치평가




1. Financial Time Series

2. Stochastic Process

3. Real Data Statistics

4. Derivatives Valuation

Lecturer : Lee, Seunghee
storm@netsgo.com
github.com/seunghee-lee/python

# 1. Financial Time Series

Time is what keeps everything from happening at once.
—Ray Cummings

### 1-1 Kospi 자료 수집 및 분석

필요 패키지 설치

| | |
|---|---|
| 1 | !pip install -U finance-datareader<br>!pip install pandas-datareader<br>!pip install yfinance |

# 먼저 필요한 library를 불러옴

| | |
|---|---|
| 2 | import FinanceDataReader as fdr<br>import yfinance as yf<br>import seaborn as sns<br>import math<br>import pandas_datareader as pdr<br>import numpy as np<br>import pandas as pd<br>import matplotlib.pyplot as plt<br>plt.style.use('fivethirtyeight')<br>%matplotlib inline<br>from datetime import datetime |

| | |
|---|---|
| 3 | # Set the start and end date<br>start_date = '2000-01-01'<br>end_date = '2023-08-03' |

| | |
|---|---|
| 4 | # 주가 데이터를 불러옴<br># (yahoo finance open API, 공식적으로 없어졌지만, 비공식적으로 서비스는 제공되고 있음)<br><br># Get the data<br>data1 = yf.download('^ks11', start_date, end_date) |

| | |
|---|---|
| 5 | # Print 5 rows<br>data1.tail() |

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| 2023-07-27 | 2599.820068 | 2617.610107 | 2586.360107 | 2603.810059 | 2603.810059 | 591900 |
| 2023-07-28 | 2593.669922 | 2609.540039 | 2586.330078 | 2608.320068 | 2608.320068 | 497300 |
| 2023-07-31 | 2628.870117 | 2639.449951 | 2622.209961 | 2632.580078 | 2632.580078 | 454200 |
| 2023-08-01 | 2644.340088 | 2668.209961 | 2639.100098 | 2667.070068 | 2667.070068 | 493600 |

2023-08-02 2651.530029 2660.899902 2611.770020 2616.469971 2616.469971 694525

FinanceDataReader에서 자료 수집

| | |
|---|---|
| 6 | data2 = fdr.DataReader('ks11', '2000') #investing.com에서 코드를 확인하고 입력<br>data2.tail() |

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| 2023-07-27 | 2599.820068 | 2617.610107 | 2586.360107 | 2603.810059 | 2603.810059 | 591900.0 |
| 2023-07-28 | 2593.669922 | 2609.540039 | 2586.330078 | 2608.320068 | 2608.320068 | 497300.0 |
| 2023-07-31 | 2628.870117 | 2639.449951 | 2622.209961 | 2632.580078 | 2632.580078 | 454200.0 |
| 2023-08-01 | 2644.340088 | 2668.209961 | 2639.100098 | 2667.070068 | 2667.070068 | 493600.0 |
| 2023-08-02 | 2651.530029 | 2660.899902 | 2611.770020 | 2616.469971 | 2616.469971 | 694525.0 |

# 주가 차트 두 가지를 그려 봄

| | |
|---|---|
| 7 | datag1=data1[['Adj Close']]<br>plt.plot(datag1) |


| | |
|---|---|
| 8 | datag2=data2[['Adj Close']]<br>plt.plot(datag2) |

| | |
|---|---|
| 9 | Kospi=data1 |

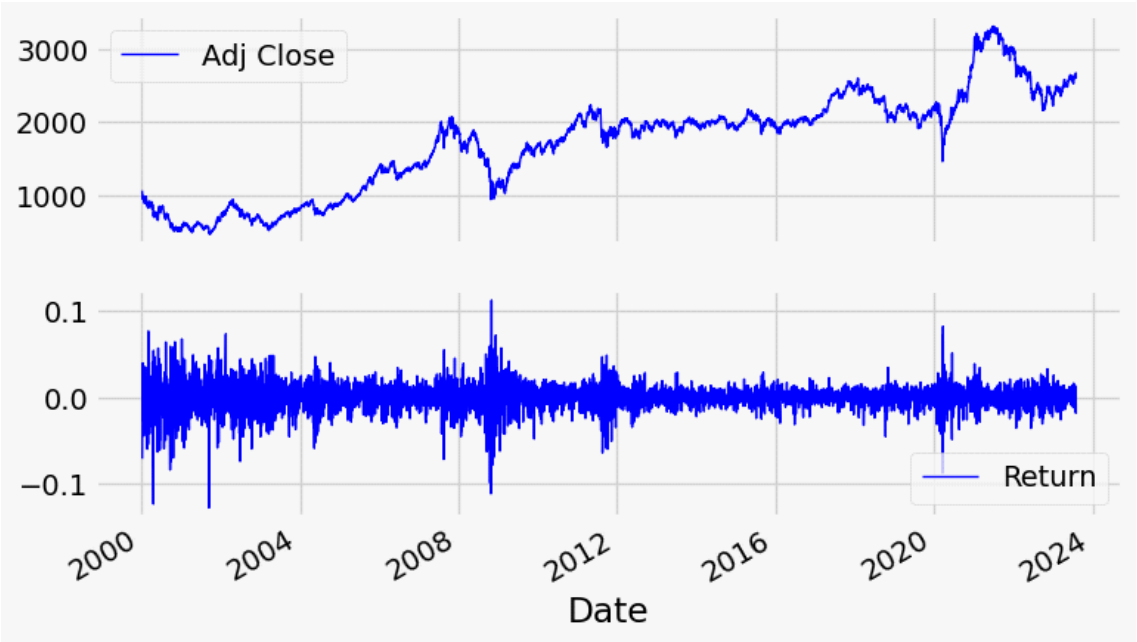
| | |
|---|---|
| 10 | Kospi['Return']=np.log(Kospi['Adj Close']/Kospi['Adj Close'].shift(1)) |

Kospi 일간 log 수익률 계산

| | |
|---|---|
| 11 | Kospi.tail() |

| Date | Open | High | Low | Close | Adj Close | Volume | Return |
|---|---|---|---|---|---|---|---|
| 2023-07-27 | 2599.820068 | 2617.610107 | 2586.360107 | 2603.810059 | 2603.810059 | 591900 | 0.004407 |
| 2023-07-28 | 2593.669922 | 2609.540039 | 2586.330078 | 2608.320068 | 2608.320068 | 497300 | 0.001731 |
| 2023-07-31 | 2628.870117 | 2639.449951 | 2622.209961 | 2632.580078 | 2632.580078 | 454200 | 0.009258 |
| 2023-08-01 | 2644.340088 | 2668.209961 | 2639.100098 | 2667.070068 | 2667.070068 | 493600 | 0.013016 |
| 2023-08-02 | 2651.530029 | 2660.899902 | 2611.770020 | 2616.469971 | 2616.469971 | 694525 | -0.019154 |

| | |
|---|---|
| 12 | Kospi[['Adj Close','Return']].plot(subplots=True, style='b', lw='1', figsize=(8,5)) |




| | |
|---|---|
| 13 | Kospi['40d']=Kospi['Adj Close'].rolling(window=40).mean()<br>Kospi['240d']=Kospi['Adj Close'].rolling(window=240).mean() |

Kospi 40일 이동평균과 240일 이동평균 계산

| | |
|---|---|
| 14 | Kospi[['Adj Close', '40d', '240d']].tail() |

| Date | Adj Close | 40d | 240d |
|---|---|---|---|
| 2023-07-27 | 2603.810059 | 2595.460254 | 2440.750458 |
| 2023-07-28 | 2608.320068 | 2596.439008 | 2441.187375 |
| 2023-07-31 | 2632.580078 | 2597.219507 | 2441.819459 |
| 2023-08-01 | 2667.070068 | 2598.511011 | 2442.416501 |
| 2023-08-02 | 2616.469971 | 2598.532758 | 2442.785376 |

| | |
|---|---|
| 15 | Kospi[['Adj Close', '40d', '240d']].plot(lw='1', figsize=(8,5)) |


| | |
|---|---|
| 16 | Kospi['Mov_vol']=Kospi['Return'].rolling(window=245).std()*math.sqrt(245) |

연간 역사적 변동성(historical volatility) 항목 추가

| | |
|---|---|
| 17 | Kospi[['Adj Close','Mov_vol', 'Return']].plot(subplots=True, style='b', lw='1', figsize=(8,7)) |


Kospi지수와 연간 변동성, 일간 수익률

## 1-2 주요 주가지수 분석

| | |
|---|---|
| 18 | kospi = yf.download('^KS11', start_date, end_date)<br>snp = yf.download('^GSPC',start_date, end_date) #S&P500<br>nikkei = yf.download('^N225',start_date, end_date) # Nikkei 225<br>euronext = yf.download('^N100',start_date, end_date) # EURONEXT 100 |

# 여러 지수 자료를 다운 받음

| | |
|---|---|
| 19 | kospi.head() |

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| 2000-01-04 | 1028.329956 | 1066.180054 | 1016.590027 | 1059.040039 | 1059.040039 | 195900 |
| 2000-01-05 | 1006.869995 | 1026.520020 | 984.049988 | 986.309998 | 986.309998 | 257700 |
| 2000-01-06 | 1013.950012 | 1014.900024 | 953.500000 | 960.789978 | 960.789978 | 203500 |
| 2000-01-07 | 949.169983 | 970.159973 | 930.840027 | 948.650024 | 948.650024 | 215700 |
| 2000-01-10 | 979.669983 | 994.940002 | 974.820007 | 987.239990 | 987.239990 | 240200 |

| | |
|---|---|
| 20 | # 지수 자료를 병합<br>eqt1 = pd.merge(snp['Adj Close'].to_frame(), kospi['Adj Close'].to_frame(),left_index=True, right_index=True, how='inner')<br>eqt1.columns=['SP500','KOSPI']<br><br>eqt2 = pd.merge(nikkei['Adj Close'].to_frame(), euronext['Adj Close'].to_frame(),left_index=True, right_index=True, how='inner')<br>eqt2.columns=['nikkei','euronext']<br><br>eqt3 = pd.merge(eqt1, eqt2,left_index=True, right_index=True, how='inner') |

| | |
|---|---|
| 21 | eqt3.head() |

| Date | SP500 | KOSPI | nikkei | euronext |
|---|---|---|---|---|
| 2000-01-04 | 1399.420044 | 1059.040039 | 19002.859375 | 955.969971 |
| 2000-01-05 | 1402.109985 | 986.309998 | 18542.550781 | 930.260010 |
| 2000-01-06 | 1403.449951 | 960.789978 | 18168.269531 | 922.460022 |
| 2000-01-07 | 1441.469971 | 948.650024 | 18193.410156 | 943.880005 |
| 2000-01-11 | 1438.560059 | 981.330017 | 18850.919922 | 954.059998 |

# 연속시간 수익률 (로그 차분)

| 22 | rtn= ( np.log(eqt3) - np.log(eqt3.shift(1)) )*100<br>rtn.columns=['r_sp','r_kp','r_nk','r_ux']<br>rtn.head() |
|---|---|

| Date | r_sp | r_kp | r_nk | r_ux |
|---|---|---|---|---|
| 2000-01-04 | NaN | NaN | NaN | NaN |
| 2000-01-05 | 0.192034 | -7.114745 | -2.452133 | -2.726237 |
| 2000-01-06 | 0.095522 | -2.621486 | -2.039149 | -0.842009 |
| 2000-01-07 | 2.672995 | -1.271589 | 0.138281 | 2.295501 |
| 2000-01-11 | -0.202075 | 3.386886 | 3.550227 | 1.072752 |

| 23 | eqt = pd.merge(eqt3,rtn,left_index=True, right_index=True, how='inner')<br>eqt.head() |
|---|---|

| Date | SP500 | KOSPI | nikkei | euronext | r_sp | r_kp | r_nk | r_ux |
|---|---|---|---|---|---|---|---|---|
| 2000-01-04 | 1399.420044 | 1059.040039 | 19002.859375 | 955.969971 | NaN | NaN | NaN | NaN |
| 2000-01-05 | 1402.109985 | 986.309998 | 18542.550781 | 930.260010 | 0.192034 | -7.114745 | -2.452133 | -2.726237 |
| 2000-01-06 | 1403.449951 | 960.789978 | 18168.269531 | 922.460022 | 0.095522 | -2.621486 | -2.039149 | -0.842009 |
| 2000-01-07 | 1441.469971 | 948.650024 | 18193.410156 | 943.880005 | 2.672995 | -1.271589 | 0.138281 | 2.295501 |
| 2000-01-11 | 1438.560059 | 981.330017 | 18850.919922 | 954.059998 | -0.202075 | 3.386886 | 3.550227 | 1.072752 |

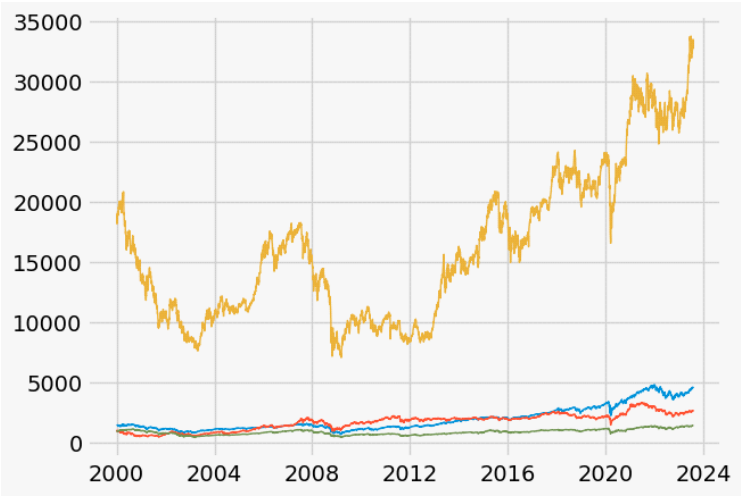
| 24 | pr = eqt[['SP500', 'KOSPI', 'nikkei','euronext']]<br>pr.head() |
|---|---|

| Date | SP500 | KOSPI | nikkei | euronext |
|---|---|---|---|---|
| 2000-01-04 | 1399.420044 | 1059.040039 | 19002.859375 | 955.969971 |
| 2000-01-05 | 1402.109985 | 986.309998 | 18542.550781 | 930.260010 |
| 2000-01-06 | 1403.449951 | 960.789978 | 18168.269531 | 922.460022 |
| 2000-01-07 | 1441.469971 | 948.650024 | 18193.410156 | 943.880005 |
| 2000-01-11 | 1438.560059 | 981.330017 | 18850.919922 | 954.059998 |

| 25 | rt = eqt[['r_sp','r_kp','r_nk','r_ux']]<br>rt.head() |
|---|---|

| Date | r_sp | r_kp | r_nk | r_ux |
|---|---|---|---|---|
| 2000-01-04 | NaN | NaN | NaN | NaN |
| 2000-01-05 | 0.192034 | -7.114745 | -2.452133 | -2.726237 |
| 2000-01-06 | 0.095522 | -2.621486 | -2.039149 | -0.842009 |
| 2000-01-07 | 2.672995 | -1.271589 | 0.138281 | 2.295501 |
| 2000-01-11 | -0.202075 | 3.386886 | 3.550227 | 1.072752 |

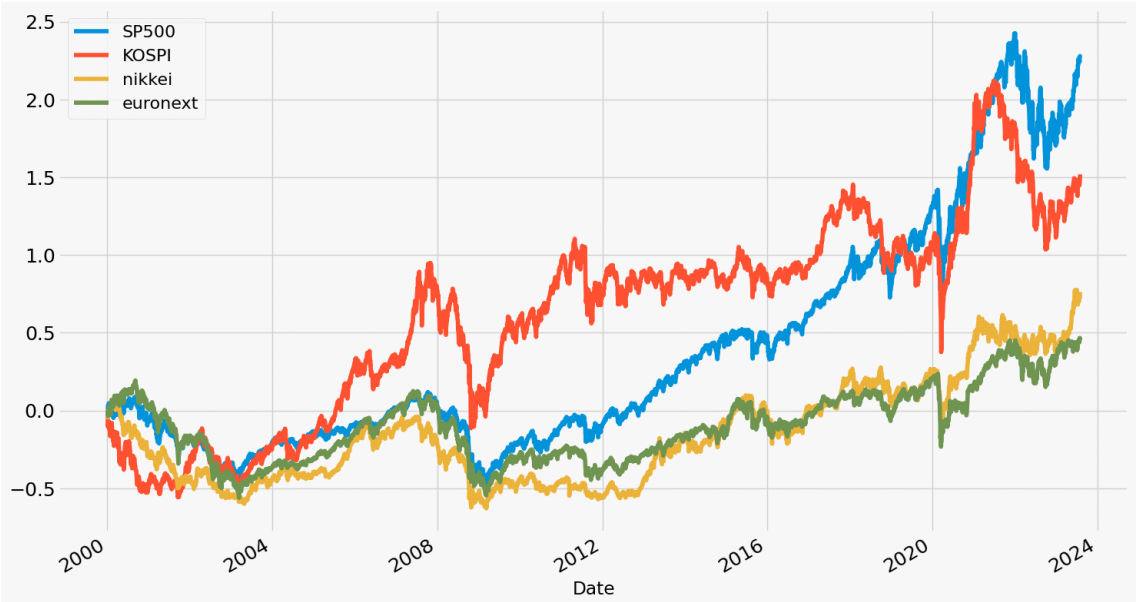
| | |
|---|---|
| 26 | # 원 자료로 그림을 그려 봄<br>plt.plot(pr, lw='1') |




지수 간 수치 차이 때문에 특징이 나타나지 않음

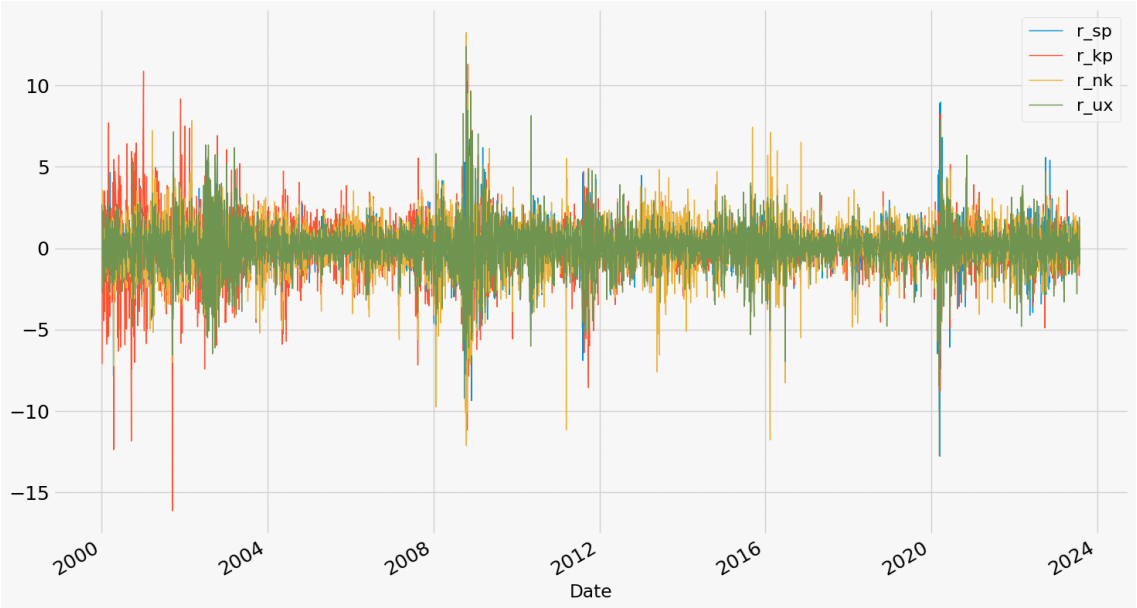
| | |
|---|---|
| 27 | # 초기시점을 0 으로 기준으로 잡고 상대 주가(relative stock price)를 표현<br>pr_0 = pr / pr.iloc[0] - 1.0 |

| | |
|---|---|
| 28 | # 상대가격을 그래프로 표현<br>plt.rcParams['legend.fontsize'] = 16<br>pr_0.plot( figsize=(16,10), fontsize=18, lw='1') |




| | |
|---|---|
| 29 | # 수익률 그래프도 표현<br>plt.rcParams['legend.fontsize'] = 16 |

| | rt.plot( figsize=(16,10), lw='1', fontsize=18) |
|---|---|




주가수익률의 전형적인 특성 (stylized facts of stock returns)

- volatility clustering
- leverage effect: negative skewness, asymmetric
- Fat tail: leptokurtic

| 30 | pr.describe() | | | |
|---|---|---|---|---|
| | SP500 | KOSPI | nikkei | euronext |
| count | 5299.000000 | 5299.000000 | 5299.000000 | 5299.000000 |
| mean | 1927.024021 | 1710.779401 | 16129.170073 | 863.766220 |
| std | 1018.532964 | 671.434492 | 6196.655771 | 219.691801 |
| min | 676.530029 | 468.760010 | 7054.979980 | 419.950012 |
| 25% | 1188.059998 | 1128.619995 | 10640.814941 | 677.750000 |
| 50% | 1444.260010 | 1907.130005 | 15314.570312 | 853.979980 |
| 75% | 2467.765015 | 2096.685059 | 20224.349609 | 1017.989990 |
| max | 4793.540039 | 3305.209961 | 33753.328125 | 1401.479980 |

| 31 | rt.describe() | | | |
|---|---|---|---|---|
| | r_sp | r_kp | r_nk | r_ux |
| count | 5298.000000 | 5298.000000 | 5298.000000 | 5298.000000 |
| mean | 0.022366 | 0.017433 | 0.010688 | 0.007034 |
| std | 1.302782 | 1.540356 | 1.531104 | 1.374493 |
| min | -12.765220 | -16.115370 | -12.111020 | -12.751740 |
| 25% | -0.508744 | -0.627022 | -0.737845 | -0.616184 |
| 50% | 0.067868 | 0.076017 | 0.051654 | 0.062080 |
| 75% | 0.614028 | 0.759577 | 0.835895 | 0.671615 |

max 10.423562 11.284352 13.234592 12.378520

| | |
|---|---|
| 32 | print(rt.describe())<br>print('skeness: ', rt.skew(axis=0))<br>print('kurtosis: ', rt.kurtosis(axis=0)) |

| | r_sp | r_kp | r_nk | r_ux |
|---|---|---|---|---|
| count | 5298.000000 | 5298.000000 | 5298.000000 | 5298.000000 |
| mean | 0.022366 | 0.017433 | 0.010688 | 0.007034 |
| std | 1.302782 | 1.540356 | 1.531104 | 1.374493 |
| min | -12.765220 | -16.115370 | -12.111020 | -12.751740 |
| 25% | -0.508744 | -0.627022 | -0.737845 | -0.616184 |
| 50% | 0.067868 | 0.076017 | 0.051654 | 0.062080 |
| 75% | 0.614028 | 0.759577 | 0.835895 | 0.671615 |
| max | 10.423562 | 11.284352 | 13.234592 | 12.378520 |

skeness: r_sp -0.436115
r_kp -0.601422
r_nk -0.429547
r_ux -0.132776
dtype: float64
kurtosis: r_sp 9.740583
r_kp 9.034665
r_nk 6.502399
r_ux 7.180151
dtype: float64

| | |
|---|---|
| 33 | # 우리가 원하는 통계량이 모두 포함된 기초통계표를 만들음<br>df=pr<br>stats = df.describe()<br>stats.loc['var'] = df.var().tolist()<br>stats.loc['skew'] = df.skew().tolist()<br>stats.loc['kurt'] = df.kurtosis().tolist()<br>print(stats) |

| | SP500 | KOSPI | nikkei | euronext |
|---|---|---|---|---|
| count | 5.299000e+03 | 5299.000000 | 5.299000e+03 | 5299.000000 |
| mean | 1.927024e+03 | 1710.779401 | 1.612917e+04 | 863.766220 |
| std | 1.018533e+03 | 671.434492 | 6.196656e+03 | 219.691801 |
| min | 6.765300e+02 | 468.760010 | 7.054980e+03 | 419.950012 |
| 25% | 1.188060e+03 | 1128.619995 | 1.064081e+04 | 677.750000 |
| 50% | 1.444260e+03 | 1907.130005 | 1.531457e+04 | 853.979980 |
| 75% | 2.467765e+03 | 2096.685059 | 2.022435e+04 | 1017.989990 |
| max | 4.793540e+03 | 3305.209961 | 3.375333e+04 | 1401.479980 |
| var | 1.037409e+06 | 450824.276460 | 3.839854e+07 | 48264.487482 |
| skew | 1.175674e+00 | -0.120515 | 6.622540e-01 | 0.339634 |

kurt 2.908057e-01 -0.681092 -5.021932e-01 -0.612041

| 34 | df=rt<br>stats = df.describe()<br>stats.loc['var'] = df.var().tolist()<br>stats.loc['skew'] = df.skew().tolist()<br>stats.loc['kurt'] = df.kurtosis().tolist()<br>print(stats) |
|---|---|

| | r_sp | r_kp | r_nk | r_ux |
|---|---|---|---|---|
| count | 5298.000000 | 5298.000000 | 5298.000000 | 5298.000000 |
| mean | 0.022366 | 0.017433 | 0.010688 | 0.007034 |
| std | 1.302782 | 1.540356 | 1.531104 | 1.374493 |
| min | -12.765220 | -16.115370 | -12.111020 | -12.751740 |
| 25% | -0.508744 | -0.627022 | -0.737845 | -0.616184 |
| 50% | 0.067868 | 0.076017 | 0.051654 | 0.062080 |
| 75% | 0.614028 | 0.759577 | 0.835895 | 0.671615 |
| max | 10.423562 | 11.284352 | 13.234592 | 12.378520 |
| var | 1.697241 | 2.372698 | 2.344281 | 1.889232 |
| skew | -0.436115 | -0.601422 | -0.429547 | -0.132776 |
| kurt | 9.740583 | 9.034665 | 6.502399 | 7.180151 |

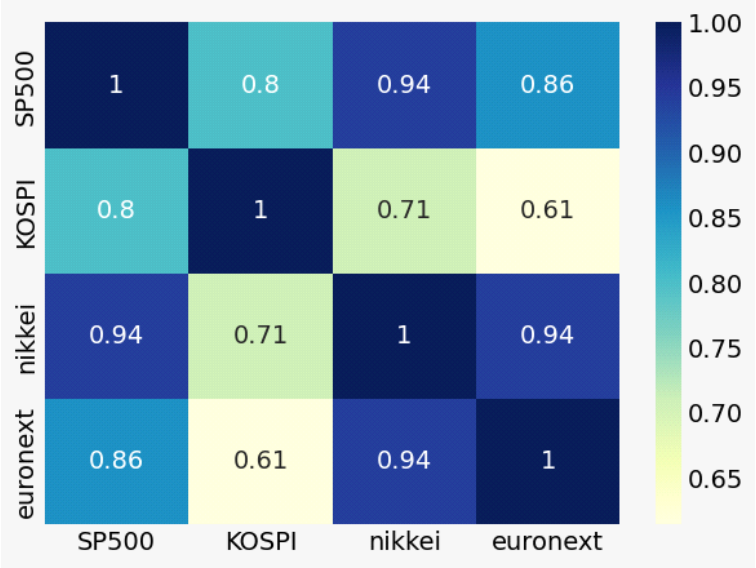
| 35 | pr.corr() |
|---|---|

| | SP500 | KOSPI | nikkei | euronext |
|---|---|---|---|---|
| SP500 | 1.000000 | 0.798445 | 0.940057 | 0.857766 |
| KOSPI | 0.798445 | 1.000000 | 0.707485 | 0.614202 |
| nikkei | 0.940057 | 0.707485 | 1.000000 | 0.939735 |
| euro~ | 0.857766 | 0.614202 | 0.939735 | 1.000000 |

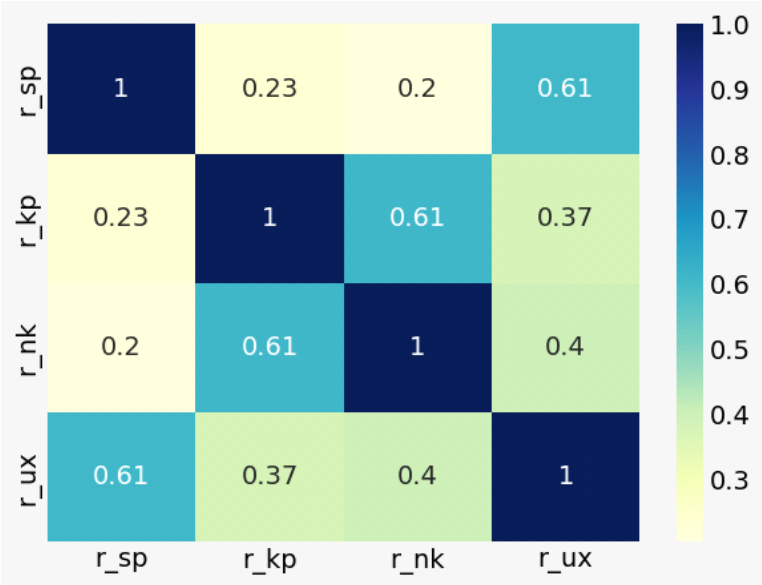
| 36 | rt.corr() |
|---|---|

| | r_sp | r_kp | r_nk | r_ux |
|---|---|---|---|---|
| r_sp | 1.000000 | 0.230965 | 0.203526 | 0.606108 |
| r_kp | 0.230965 | 1.000000 | 0.609524 | 0.374941 |
| r_nk | 0.203526 | 0.609524 | 1.000000 | 0.400374 |
| r_ux | 0.606108 | 0.374941 | 0.400374 | 1.000000 |

| | |
|---|---|
| 37 | sns.heatmap(pr.corr(), annot=True, cmap="YlGnBu")<br>plt.show() |




| | |
|---|---|
| 38 | sns.heatmap(rt.corr(), annot=True, cmap="YlGnBu")<br>plt.show() |



## 1-3 회귀 분석

### 1-3-1 data 수집

| | |
|---|---|
| 39 | snp = yf.download('^GSPC',start_date, end_date) #S&P500 지수<br>vix = yf.download('^vix',start_date, end_date) #VIX 지수 |

| | |
|---|---|
| 40 | data = pd.merge(snp['Adj Close'].to_frame(), vix['Adj Close'].to_frame(),left_index=True, right_index=True, how='inner')<br>data.columns=['SP500','VIX'] |

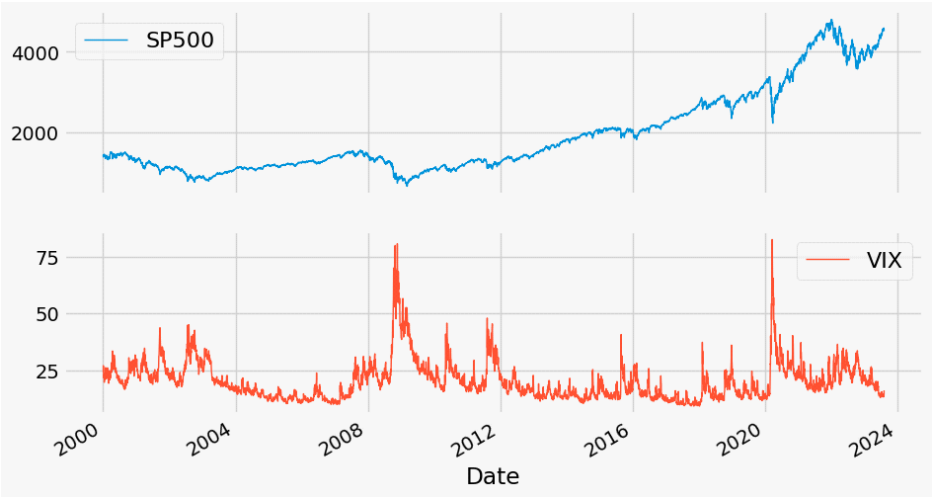
CBOE Volatility Index (INDEXCBOE:VIX)

- 변동성 지수. 주식시장이 빠르게 하락할 때 상승하므로 공포지수라고도 불린다.
- 인버스 종목과 차이점은 인버스는 주가가 떨어져야 오르는 종목이지만 변동성 지수는 주가가 떨어져도 지수가 올라가지 않을 수 있다. 주가가 급격히 떨어져야 지수가 오른다.
- 시카고옵션거래소에 상장된 S&P 500 지수옵션의 향후 30일간의 변동성에 대한 시장의 기대를 나타내는 수치(내재변동성, implied volatility)다.
- 인덱스가 있긴 하지만 다른 인덱스 펀드와는 다르게 VIX는 직접적으로 매수가 가능한 실물이 있지 않다. 또한 인덱스의 변동폭이 어마어마하게 높기 때문에 VIX의 선물을 기반으로 한 ETF나 ETN을 만들어 거래한다.

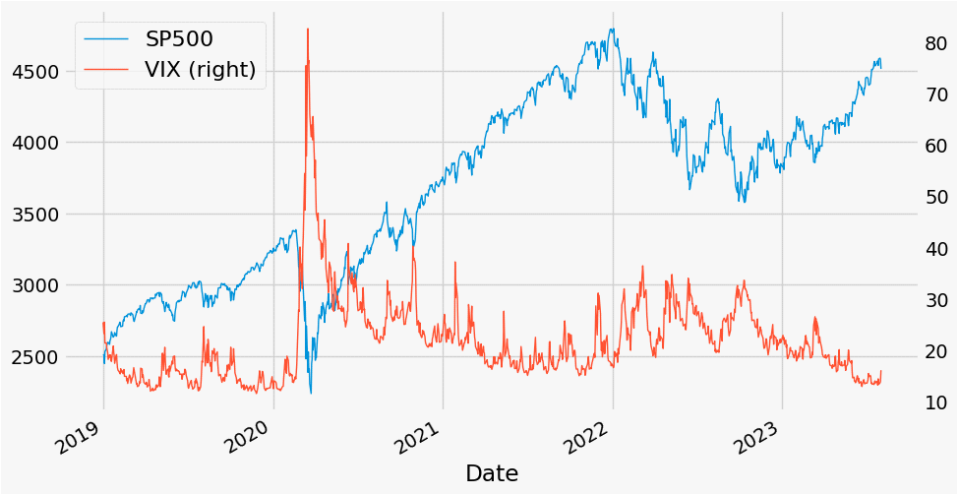
| | |
|---|---|
| 41 | data.tail() |

| Date | SP500 | VIX |
|---|---|---|
| 2023-07-27 | 4537.410156 | 14.41 |
| 2023-07-28 | 4582.229980 | 13.33 |
| 2023-07-31 | 4588.959961 | 13.63 |
| 2023-08-01 | 4576.729980 | 13.93 |
| 2023-08-02 | 4513.390137 | 16.09 |

| | |
|---|---|
| 42 | data.plot(subplots=True, figsize=(10, 6), lw='1') |



| | |
|---|---|
| 43 | data.loc['2018-12-31':].plot(secondary_y='VIX', figsize=(10, 6), lw='1') |




### 1-3-2 log 수익률

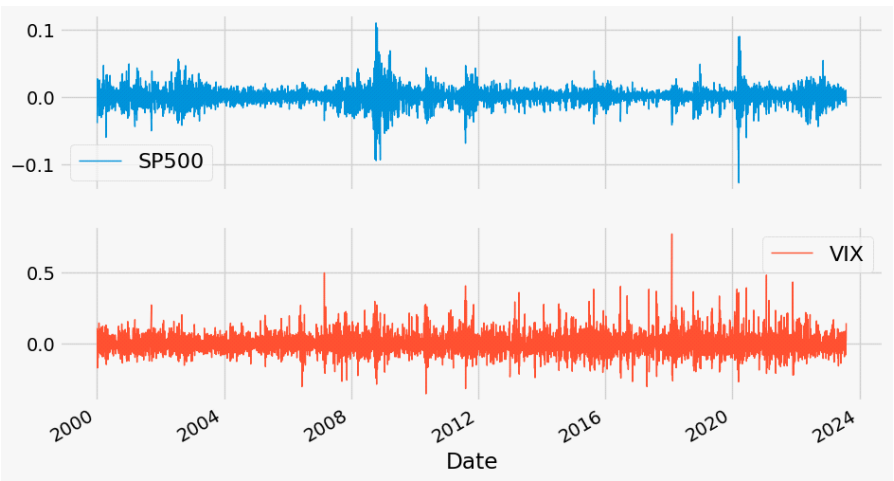
| | |
|---|---|
| 44 | rets = np.log(data / data.shift(1)) |

| | |
|---|---|
| 45 | rets.head() |

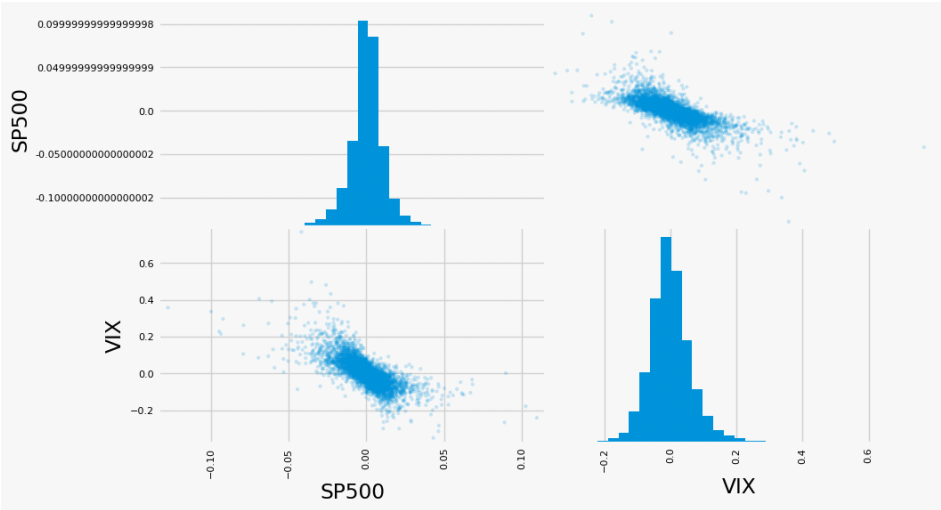
| Date | SP500 | VIX |
|---|---|---|
| 2000-01-03 | NaN | NaN |
| 2000-01-04 | -0.039099 | 0.109441 |
| 2000-01-05 | 0.001920 | -0.022464 |
| 2000-01-06 | 0.000955 | -0.026085 |
| 2000-01-07 | 0.026730 | -0.169424 |

| | |
|---|---|
| 46 | rets.dropna(inplace=True) |

| | |
|---|---|
| 47 | rets.plot(subplots=True, figsize=(10, 6), lw='1') |



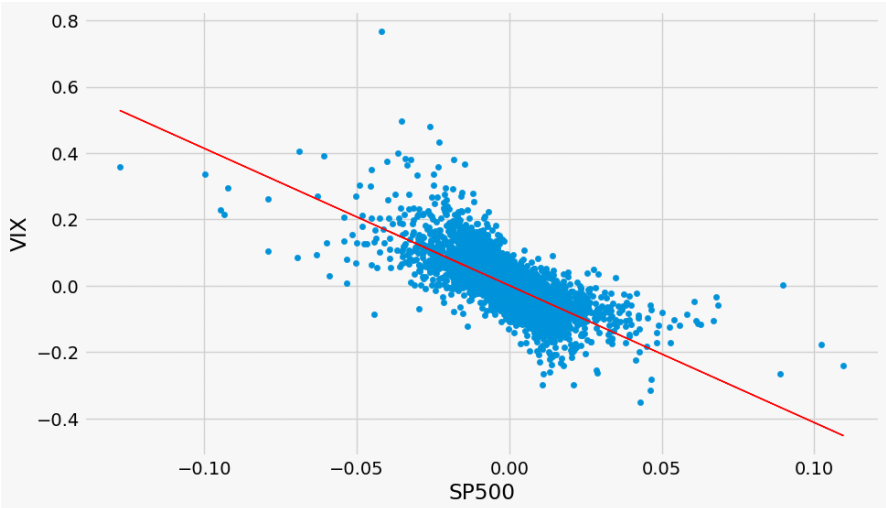
```
48  pd.plotting.scatter_matrix(rets,
                               alpha=0.2,
                               diagonal='hist',
                               hist_kwds={'bins': 35},
                               figsize=(10, 6))
```




### 1-3-3 OLS 회귀분석

```
49  reg = np.polyfit(rets['SP500'], rets['VIX'], deg=1)
```

```
50  ax = rets.plot(kind='scatter', x='SP500', y='VIX', figsize=(10, 6))
    ax.plot(rets['SP500'], np.polyval(reg, rets['SP500']), 'r', lw='1')
```

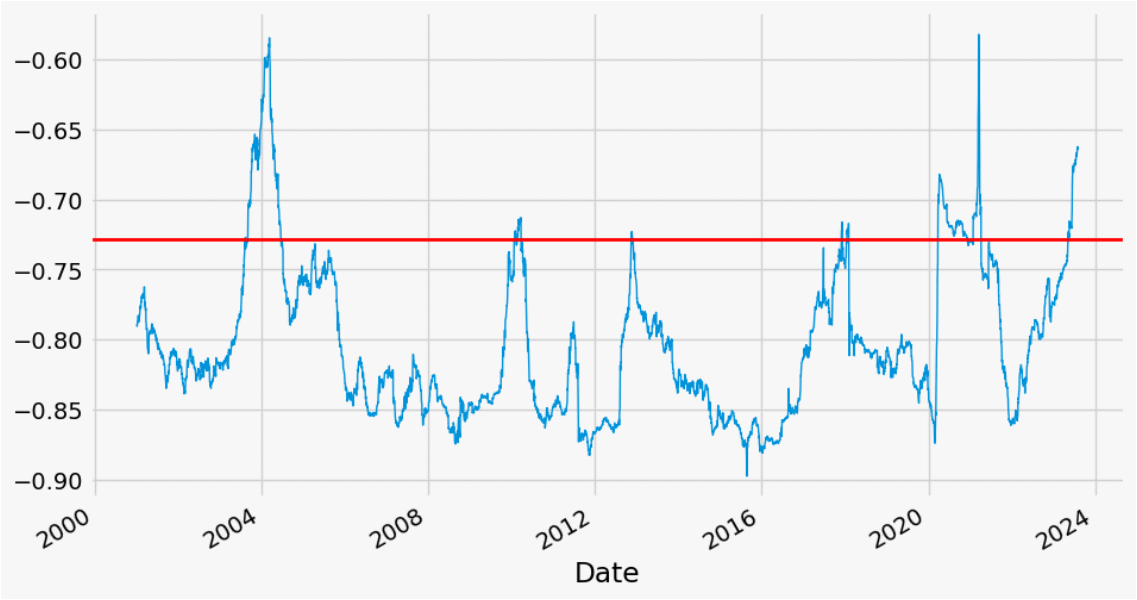



### 1-3-4 상관관계

```
51  rets.corr()
```

| | SP500 | VIX |
|---|---|---|
| SP500 | 1.00000 | -0.72838 |
| VIX | -0.72838 | 1.00000 |

| | |
|---|---|
| 52 | ax = rets['SP500'].rolling(window=252).corr(<br>rets['VIX']).plot(figsize=(10, 6), lw='1')<br>ax.axhline(rets.corr().iloc[0, 1], c='r', lw='2') |




- 이 장에서 금융 분야에서 가장 중요한 데이터 유형인 금융 시계열을 다뤘다.
- pandas는 시계열 데이터를 처리하는 강력한 패키지로, 효율적인 데이터 분석뿐만 아니라 손쉬운 시각화도 가능하다.
- 또한 pandas는 다양한 소스에서 시계열 데이터를 읽고 데이터를 다른 파일 형식으로 내보내는 데 유용하다.

# 2. Stochastic Process

Predictability is not how things will go, but how they can go.
—Raheel Farooq

추계학(stochastics, 推計學)

- 수리통계학(數理統計學)의 주류를 이루는 수학 분야로서 행동적인 추측과 계획의 학문을 뜻하며, 추측통계학이라고도 한다. 종래의 통계학 및 기술통계학(記述統計學)과 구별하기 위해서 명명된 것이다.
- 수리통계학은 기술통계학이 완성된 19세기 말경부터 점차 확률론적인 파악방법이 도입되었으며, 20세기 초가 되어 영국의 R.A.피셔에 의해 소견본(小見本)의 이론이 전개되어 추계학으로 발전하였다.
- 추계학 내용은 ① 비결정론적일 것 ② 집단적 규칙성을 보일 것의 두 성질을 구비하는 것이며, 그 발생분야인 농사시험법뿐만 아니라 그 밖의 자연과학의 각 부문에 큰 영향을 미치고 있다. 여론조사 ·제품검사, 의약의 효용, 교육면 등을 비롯하여 사회과학의 방법론에까지 이용된다.

출처: [네이버 지식백과] 추계학 [stochastics, 推計學] (두산백과 두피디아, 두산백과)

### 2-1 stochastics

- 추계학은 금융에서 가장 중요한 수학적, 수치 해석 분야
- 금융공학 초기인 1970~1980년대는 옵션 가격을 계산하는 닫힌 형태의 공식(closed-form solutions)을 구하는 것이 목표
- 이후에는 개별 상품의 가치평가도 중요하지만 전체 파생상품 포지션(whole derivatives books)을 일관되게 평가하는 것이 중요시 됨
- 또한 value-at-risk(VaR)과 신용평가 조정(credit valuation adjustments) 등을 평가하기 위해 효율적인 수치 해석 방법(numerical methods)이 필요
- 따라서 일반적인 확률과정(stochastics process)과 몬테카를로 시뮬레이션(Monte Carlo simulation)이 중요

| | |
|---|---|
| 1 | import math<br>import numpy as np<br>import numpy.random as npr<br>from pylab import plt, mpl |

| | |
|---|---|
| 2 | plt.style.use('seaborn-v0_8')<br>mpl.rcParams['font.family'] = 'serif'<br>%matplotlib inline |

### 2-2 난수 생성

numpy.random에서 제공하는 함수를 이용하여 난수(random numbers) 생성

| | |
|---|---|
| 3 | npr.seed(100)<br>np.set_printoptions(precision=4) |

| | |
|---|---|
| 4 | npr.rand(10) |

```
array([0.5434, 0.2784, 0.4245, 0.8448, 0.0047, 0.1216, 0.6707, 0.8259,
       0.1367, 0.5751])
```

rand 함수는 구간 (0,1) 사이의 난수를 함수 인수에서 정한 형태로 반환

유사 난수(pseudo random)

- 컴퓨터는 이론적으로 완벽한 난수를 생성할 수 없음
- 난수표, 난수 알고리즘, 알고리즘 초기화에 사용할 시드(seed) 값으로 난수 생성
- 시드가 필요한 난수를 유사 난수(pseudo random)라고 함
- 최근엔 알고리즘 대신 열 잡음, 광전자 등 신호의 노이즈를 이용해 시드가 필요없는 하드웨어 랜덤 번호 생성기(hardware random number generator)를 사용하기도 함
- 유사 난수의 가장 큰 특징은 예측 할 수 없는 난수를 생성한다는 점, 예를 들어 1부터 10까지 범위를 가지는 난수를 무작위로 열 번 생성 할 때, 같은 숫자가 두 번 이상 나오거나 한 번도 나오지 않는 숫자가 있는 경우가 있음

| | |
|---|---|
| 5 | npr.rand(5, 5) |

```
array([[0.8913, 0.2092, 0.1853, 0.1084, 0.2197],
       [0.9786, 0.8117, 0.1719, 0.8162, 0.2741],
       [0.4317, 0.94  , 0.8176, 0.3361, 0.1754],
       [0.3728, 0.0057, 0.2524, 0.7957, 0.0153],
       [0.5988, 0.6038, 0.1051, 0.3819, 0.0365]])
```

5~10 사이의 난수를 생성

| | |
|---|---|
| 6 | a = 5.<br>b = 10.<br>npr.rand(10) * (b - a) + a |

```
array([9.4521, 9.9046, 5.2997, 9.4527, 7.8845, 8.7124, 8.1509, 7.9092,
       5.1022, 6.0501])
```

NumPy broadcasting으로 다차원 난수 생성

| | |
|---|---|
| 7 | npr.rand(5, 5) * (b - a) + a |

```
array([[7.7234, 8.8456, 6.2535, 6.4295, 9.262 ],
       [9.875 , 9.4243, 6.7975, 7.9943, 6.774 ],
       [6.701 , 5.8904, 6.1885, 5.2243, 7.5272],
       [6.8813, 7.964 , 8.1497, 5.713 , 9.6692],
       [9.7319, 8.0115, 6.9388, 6.8159, 6.0217]])
```

**<간단한 난수를 생성하는 함수>**

**rand(d0, d1, ..., dn)** : Random values in a given shape.

**randn(d0, d1, ..., dn)** : Return a sample (or samples) from the "standard normal" distribution.

**randint(low[, high, size, dtype])** : Return random integers from low (inclusive) to high (exclusive).

**random_integers(low[, high, size])** : Random integers of type np.int_ between low and high, inclusive.

**random_sample([size])** : Return random floats in the half-open interval [0.0, 1.0).

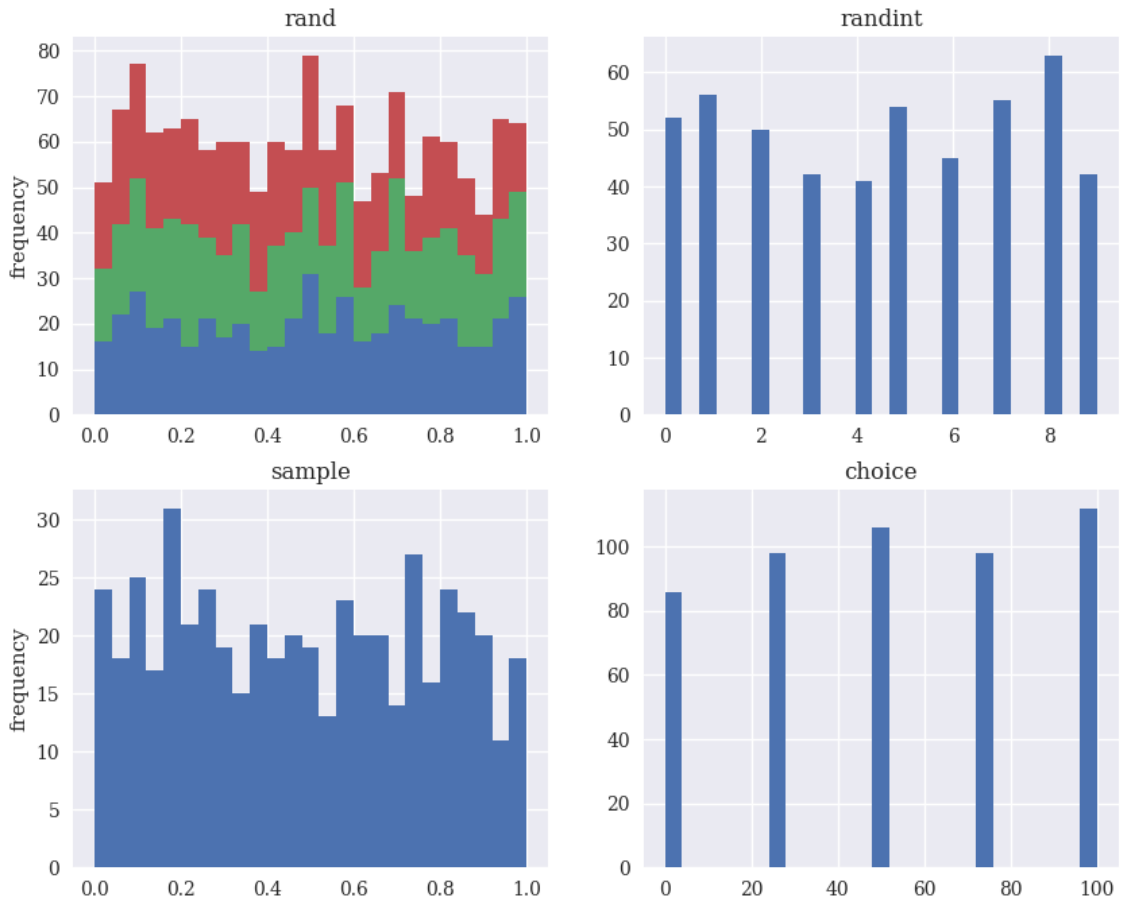
**choice(a[, size, replace, p])** : Generates a random sample from a given 1-D array

**bytes(length)** : Return random bytes.

출처: https://numpy.org/doc/stable/reference/random/legacy.html

| | |
|---|---|
| 8 | sample_size = 500<br>rn1 = npr.rand(sample_size, 3)<br>rn2 = npr.randint(0, 10, sample_size)<br>rn3 = npr.sample(size=sample_size)<br>a = [0, 25, 50, 75, 100]<br>rn4 = npr.choice(a, size=sample_size) |

| | |
|---|---|
| 9 | fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2,<br>figsize=(10, 8))<br>ax1.hist(rn1, bins=25, stacked=True)<br>ax1.set_title('rand')<br>ax1.set_ylabel('frequency')<br>ax2.hist(rn2, bins=25)<br>ax2.set_title('randint')<br>ax3.hist(rn3, bins=25)<br>ax3.set_title('sample')<br>ax3.set_ylabel('frequency')<br>ax4.hist(rn4, bins=25)<br>ax4.set_title('choice'); |




**<분포(Distributions) 함수>**

beta(a, b[, size]): Draw samples from a Beta distribution.
binomial(n, p[, size]): Draw samples from a binomial distribution.
chisquare(df[, size]): Draw samples from a chi-square distribution.
dirichlet(alpha[, size]): Draw samples from the Dirichlet distribution.
exponential([scale, size]): Draw samples from an exponential distribution.
f(dfnum, dfden[, size]): Draw samples from an F distribution.
gamma(shape[, scale, size]): Draw samples from a Gamma distribution.
geometric(p[, size]): Draw samples from the geometric distribution.
gumbel([loc, scale, size]): Draw samples from a Gumbel distribution.
hypergeometric(ngood, nbad, nsample[, size]): Draw samples from a Hypergeometric distribution.
laplace([loc, scale, size]): Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).
logistic([loc, scale, size]): Draw samples from a logistic distribution.
lognormal([mean, sigma, size]): Draw samples from a log-normal distribution.
logseries(p[, size]): Draw samples from a logarithmic series distribution.
multinomial(n, pvals[, size]): Draw samples from a multinomial distribution.

multivariate_normal(mean, cov[, size, ...]): Draw random samples from a multivariate normal distribution.
negative_binomial(n, p[, size]): Draw samples from a negative binomial distribution.
noncentral_chisquare(df, nonc[, size]): Draw samples from a noncentral chi-square distribution.
noncentral_f(dfnum, dfden, nonc[, size]): Draw samples from the noncentral F distribution.
normal([loc, scale, size]): Draw random samples from a normal (Gaussian) distribution.
pareto(a[, size]): Draw samples from a Pareto II or Lomax distribution with specified shape.
poisson([lam, size]): Draw samples from a Poisson distribution.
power(a[, size]): Draws samples in [0, 1] from a power distribution with positive exponent a - 1.
rayleigh([scale, size]): Draw samples from a Rayleigh distribution.
standard_cauchy([size]): Draw samples from a standard Cauchy distribution with mode = 0.
standard_exponential([size]): Draw samples from the standard exponential distribution.
standard_gamma(shape[, size]): Draw samples from a standard Gamma distribution.
standard_normal([size]): Draw samples from a standard Normal distribution (mean=0, stdev=1).
standard_t(df[, size]): Draw samples from a standard Student's t distribution with df degrees of freedom.
triangular(left, mode, right[, size]): Draw samples from the triangular distribution over the interval [left, right].
uniform([low, high, size]): Draw samples from a uniform distribution.
vonmises(mu, kappa[, size]): Draw samples from a von Mises distribution.
wald(mean, scale[, size]): Draw samples from a Wald, or inverse Gaussian, distribution.
weibull(a[, size]): Draw samples from a Weibull distribution.
zipf(a[, size]): Draw samples from a Zipf distribution.
출처: https://numpy.org/doc/stable/reference/random/legacy.html

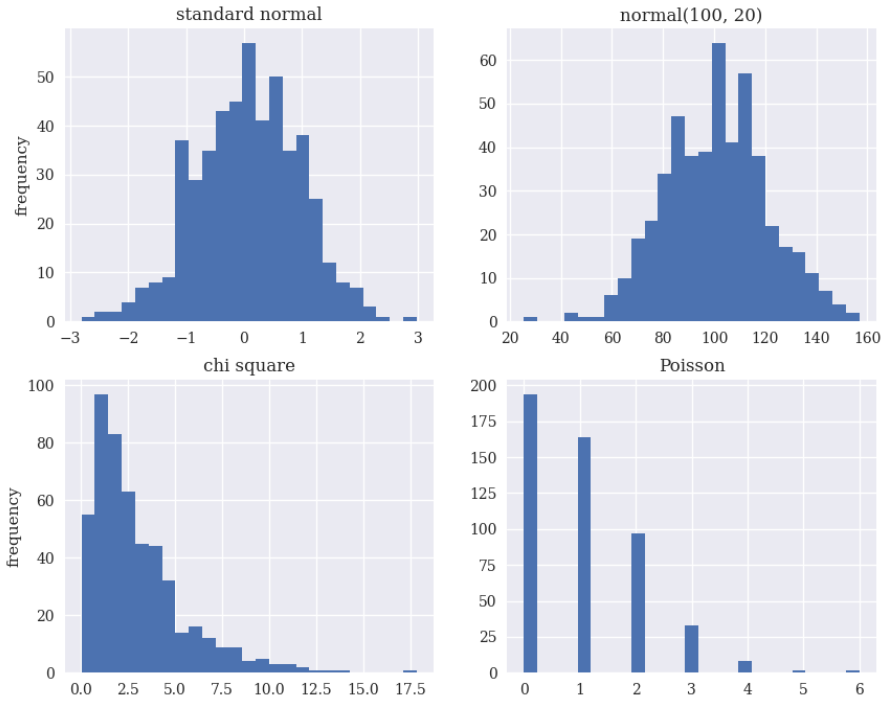
| | |
|---|---|
| 10 | sample_size = 500<br>rn1 = npr.standard_normal(sample_size)<br>rn2 = npr.normal(100, 20, sample_size)<br>rn3 = npr.chisquare(df=3, size=sample_size)<br>rn4 = npr.poisson(lam=1.0, size=sample_size) |

- 평균이 0이고 표준편차가 1인 표준정규분포

- 평균이 100이고 표준편차가 20인 정규분포
- 자유도가 3인 카이제곱분포
- 람다 계수가 1인 푸아송 분포

※ 카이제곱분포(chi-squared distribution): k개의 서로 독립적인 표준정규 확률변수를 각각 제곱한 다음 합해서 얻어지는 분포. k를 자유도라고 하며, 카이제곱 분포의 매개변수. 카이제곱 분포는 신뢰구간이나 가설검정 등의 모델에서 자주 등장
※ 푸아송분포(Poisson distribution): 발생확률이 작은 사건을 대략적으로 관찰할 때, 그 발생횟수가 만드는 분포, 가끔씩 발생하는 외부 쇼크나 금융상품가격의 갑작스런 변화를 반영하는 분포

| | |
|---|---|
| 11 | fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2,<br>figsize=(10, 8))<br>ax1.hist(rn1, bins=25)<br>ax1.set_title('standard normal')<br>ax1.set_ylabel('frequency')<br>ax2.hist(rn2, bins=25)<br>ax2.set_title('normal(100, 20)')<br>ax3.hist(rn3, bins=25)<br>ax3.set_title('chi square')<br>ax3.set_ylabel('frequency')<br>ax4.hist(rn4, bins=25)<br>ax4.set_title('Poisson'); |



## 2-3 시뮬레이션(Simulation)

※ 몬테카를로 시뮬레이션: 무작위 추출된 난수를 이용하여 원하는 함수의 값을 계산하기 위한 시뮬레이션 방법. 자유도가 높거나 닫힌 형태(closed form)의 해가 없는 문제들에 널리 쓰이는 방법이지만, 어느 정도의 오차를 감안해야만 하는 특징을 보유

### 2-3-1 확률 변수(Random Variables)

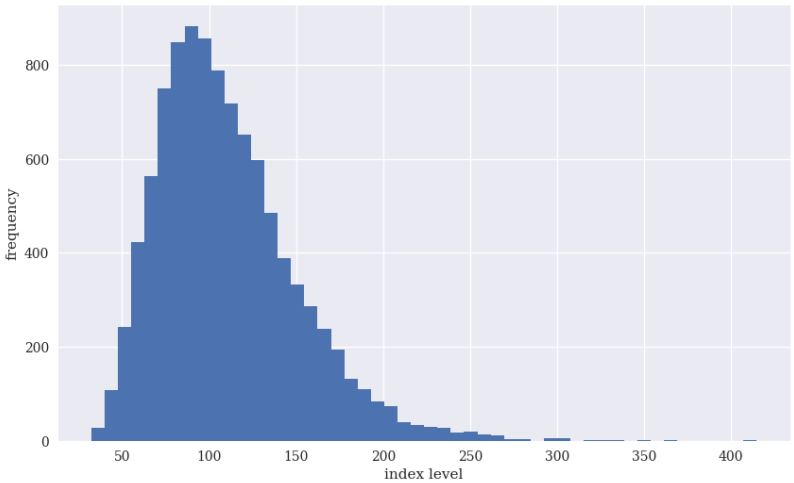
Black-Scholes-Merton 옵션 평가 모형에서 미래의 주가 시뮬레이션

$$S_T = S_0 \exp\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}z$$

여기서, $S_T$ : 날짜 $T$에서의 주가
$r$ : 무위험 단기 이자율
$\sigma$ : 주가 $S$에 대한 고정 변동성(수익률 표준편차)
$z$ : 표준정규분포를 따르는 확률 변수

| | |
|---|---|
| 12 | S0 = 100<br>r = 0.05<br>sigma = 0.25 # 고정된 변동성<br>T = 2.0 # 시간(연)<br>I = 10000 # 시뮬레이션 횟수<br>ST1 = S0 * np.exp((r - 0.5 * sigma ** 2) * T +<br>sigma * math.sqrt(T) * npr.standard_normal(I)) |

| | |
|---|---|
| 13 | plt.figure(figsize=(10, 6))<br>plt.hist(ST1, bins=50)<br>plt.xlabel('index level')<br>plt.ylabel('frequency'); |

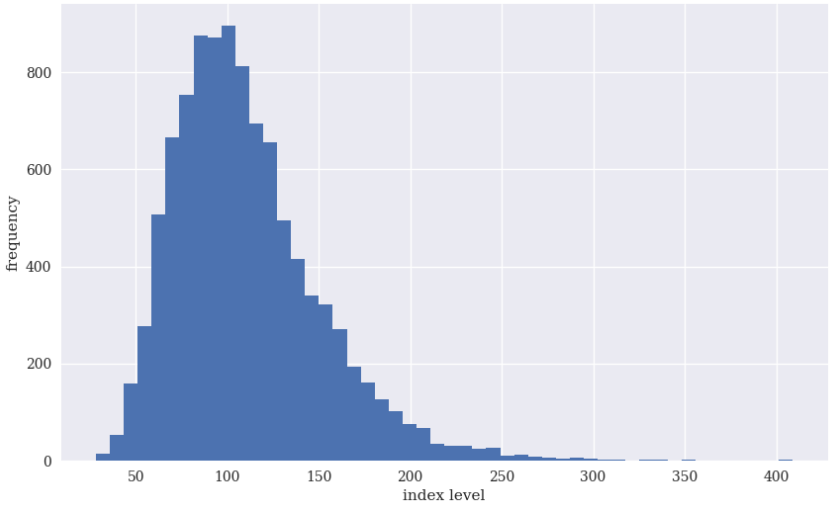



위의 시뮬레이션은 표준정규분포를 가정
Black-Scholes-Merton 옵션 평가 모형은 로그 정규분포를 가정함

lognormal 함수를 사용하여 평균과 표준편차를 인수로 로그정규분포를 생성,

| | |
|---|---|
| 14 | ST2 = S0 * npr.lognormal((r - 0.5 * sigma ** 2) * T,<br>sigma * math.sqrt(T), size=I) |

| | |
|---|---|
| 15 | plt.figure(figsize=(10, 6))<br>plt.hist(ST2, bins=50)<br>plt.xlabel('index level')<br>plt.ylabel('frequency'); |




표준정규분포와 로그 정규분포 결과는 유사함
시뮬레이션 결과의 통계 특성 비교

| | |
|---|---|
| 16 | import scipy.stats as scs |

| | |
|---|---|
| 17 | def print_statistics(a1, a2):<br>''' Prints selected statistics.<br><br>Parameters<br>==========<br>a1, a2: ndarray objects<br>results objects from simulation<br>'''<br>sta1 = scs.describe(a1)<br>sta2 = scs.describe(a2)<br>print('%14s %14s %14s' %<br>('statistic', 'data set 1', 'data set 2'))<br>print(45 * "-")<br>print('%14s %14.3f %14.3f' % ('size', sta1[0], sta2[0]))<br>print('%14s %14.3f %14.3f' % ('min', sta1[1][0], sta2[1][0]))<br>print('%14s %14.3f %14.3f' % ('max', sta1[1][1], sta2[1][1])) |

```
    print('%14s %14.3f %14.3f' % ('mean', sta1[2], sta2[2]))
    print('%14s %14.3f %14.3f' % ('std', np.sqrt(sta1[3]), np.sqrt(sta2[3])))
    print('%14s %14.3f %14.3f' % ('skew', sta1[4], sta2[4]))
    print('%14s %14.3f %14.3f' % ('kurtosis', sta1[5], sta2[5]))
```

```
18  print_statistics(ST1, ST2)
```

```
      statistic       data set 1       data set 2
--------------------------------------------------
           size         10000.000        10000.000
            min            32.327           28.230
            max           414.825          409.110
           mean           110.661          110.452
            std            40.192           39.838
           skew             1.096            1.113
       kurtosis             2.233            2.218
```

### 2-3-2 확률 과정(Stochastic Processes)

- 확률과정은 확률 변수들의 수열이라 할 수 있음
- 확률 과정을 시뮬레이션 한다는 것은 확률 변수를 반복해서 시뮬레이션하는 것과 유사
- 단 각 단계의 샘플이 완전 독립적이지 않고 이전의 결과에 의존한다는 차이점
- 금융에 사용되는 확률 과정은 Markov 속성을 나타낸다. 내일 값은 오늘 값에만 의존하며 다른 "역사적" 값이나 전체 경로에 의존하지 않는다는 것. 이 프로세스를 무기억성(memoryless)이라고도 한다.

#### 1) 기하 브라운 운동(geometric brownian motion; GBM) 모형

블랙-숄즈-머튼모형에서 자산 가격의 변화를 나타내는데 사용
확률 미분 방정식(stochastic differential equation; SDE)

$$dS_t = rS_t dt + \sigma S_t dZ_t$$

위의 식은 $S_t$의 변화량은 r과 $S_t$의 곱으로 결정되는데 $\sigma S_t$만큼의 변동성이 있다는 것이다.
$Z_t$는 표준 브라운 운동 과정 값을 나타낸다. $S_t$의 값은 로그 정규분포를 따르고 수익률$\frac{dS_t}{S_t}$은 정규분포를 나타낸다.

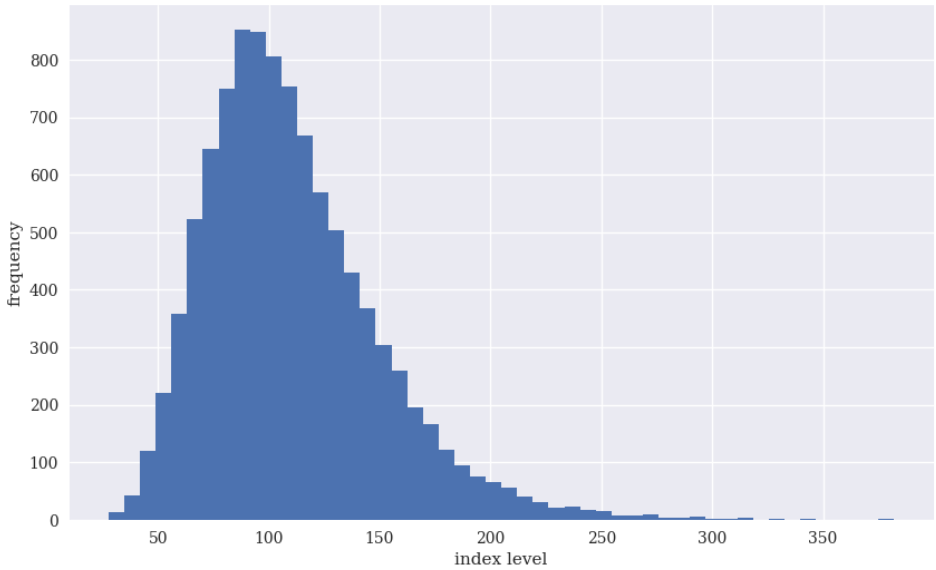
위의 확률 미분 방정식은 오일러 방법(Euler scheme)으로 정확히 이산화(discretization)할 수 있다. 즉, closed form solution으로 표현할 수 있다.

$$S_t = S_{t-\Delta t} \exp\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}\, z_t$$

위의 식에서 Δt는 고정된 시간 이산화 간격, $z_t$는 정규분포를 따르는 확률 변수

| | |
|---|---|
| 19 | I = 10000 # 시뮬레이션 횟수<br>M = 50 # The number of time intervals<br>dt = T / M<br>S = np.zeros((M + 1, I))<br>S[0] = S0<br>for t in range(1, M + 1):<br>S[t] = S[t - 1] * np.exp((r - 0.5 * sigma ** 2) * dt +<br>sigma * math.sqrt(dt) * npr.standard_normal(I)) |

| | |
|---|---|
| 20 | plt.figure(figsize=(10, 6))<br>plt.hist(S[-1], bins=50)<br>plt.xlabel('index level')<br>plt.ylabel('frequency'); |



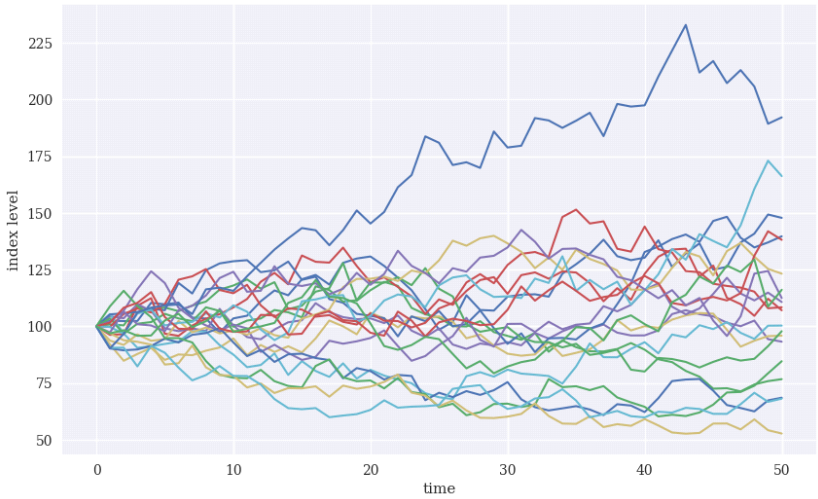

| | |
|---|---|
| 21 | print_statistics(S[-1], ST2) |

| statistic | data set 1 | data set 2 |
|---|---|---|
| size | 10000.000 | 10000.000 |
| min | 27.746 | 28.230 |
| max | 382.096 | 409.110 |
| mean | 110.416 | 110.452 |
| std | 39.182 | 39.838 |
| skew | 1.070 | 1.113 |
| kurtosis | 2.026 | 2.218 |

정적 시뮬레이션 방식(data set 2)과 유사한 통계치

20개의 시뮬레이션 경로 표현

| | |
|---|---|
| 22 | plt.figure(figsize=(10, 6))<br>plt.plot(S[:, :20], lw=1.5)<br>plt.xlabel('time')<br>plt.ylabel('index level'); |




2) **블랙-숄즈-머튼 모형**

1973년 피셔 블랙(Fischer Black)과 마이론 숄즈(Myron Scholes)는 주식옵션의 가격을 결정하는 문제에 돌파구를 여는 논문을 발표했다. 그들의 논문은 옵션의 가격결정 방법과 옵션의 헤지 방법을 제시함으로써 옵션연구에 큰 영향을 미쳤다. 블랙은 먼저 사망하였지만 숄즈는 이 모형의 공로를 인정받아 1997년 머튼(Merton)과 공동으로 노벨경제학상을 수상하였다.

**① 블랙-숄즈모형의 기본 가정**

블랙-숄즈는 옵션가격결정모형을 도출하기 위해 다음과 같은 가정을 하였다.

- 주식가격의 수익률은 로그정규분포(log-normal distribution)를 따른다.
- 거래비용과 세금이 존재하지 않는 완전자본시장이다.
- 옵션의 만기일까지 배당금이 없다.
- 차익거래 기회가 존재하지 않는다.
- 증권의 거래는 연속적으로 이루어지며 모든 증권은 완전히 분할가능하다.
- 무위험이자율은 일정하다.

현실적으로는 주식에 배당이 지급되며 완전시장은 존재하기 힘들지만, 이후 많은 연구결과에 따르면 이를 모형에 추가로 반영하여도 가격에 큰 영향을 미치지 않는 것으로 알려졌다.

**② 블랙-숄즈모형**

먼저 기초자산인 주식의 가격변화는 시간의 경과에 따라 일정하게 변하는 부분과 시간의 경과에 관계없이 불규칙하게 변하는 부분으로 구성된다고 보았다(이러한 변화방법은 Geometric Brownian Motion에 따른다고 한다).

$dS = \mu S dt + \sigma d W$

여기서, μ는 주식의 평균수익률을 나타내며, σ는 주식수익률의 표준편차를 의미한다.

여러 가정 하에 배당이 없는 유럽형 콜옵션의 가치를 블랙-숄즈 옵션가격결정모형으로 아래와 같이 표현할 수 있다.

$C = S \times N(d_1) - Xe^{-rT} \times N(d_2)$

여기서, $d_1 = \dfrac{\ln(\frac{S}{X}) + (r + \frac{1}{2}\sigma^2) \times T}{\sigma\sqrt{T}}$ 이고, $d_2 = d_1 - \sigma\sqrt{T}$이다.

S = 기초자산의 현재가, X = 행사가격, r= 무위험 이자율

T = 옵션의 잔존기간, σ= 옵션기초자산 수익률의 표준편차

N(d)= 누적 정규분포 확률함수

아주 복잡해 보이는 위의 계산식의 의미를 간단히 설명하면, 미래에 발생할 현금흐름( $S_T$-X, 또는 0)을 확률에 따라 기대값을 구한 후 이를 현재가치로 환산한 것이다.

즉, 첫 번째 항 S × N(d1)은 주가가 행사가격을 초과하는 경우 $S_T$, 아니면 0값을 갖는 기대값의 현재가치이며, 두 번째 항 $Xe^{-rT}$× N(d2)은 만기일에 지급할 행사가격(X) 기대값의 현재가치이다. N(d2)은 콜 옵션이 행사될 가능성을 나타낸다.

결국 블랙-숄즈모형에 의한 콜옵션의 가치는 만기일의 주가 기대값의 현재가치로부터 권리행사시 지급할 금액의 기대값의 현재가치를 차감한 금액이다.

풋옵션의 가격은 위 식과 풋-콜 패리티를 이용하여 구할 수 있는데 그 값은 다음과 같다.

$P = Xe^{-rT} \times N(-d_2) - S \times N(-d_1)$

여기서, $N(-d_1) = 1 - N(d_1)$이고, $N(-d_2) = 1 - N(d_2)$이다.

**③ 블랙-숄즈모형 파이썬 코드**

```
23
import numpy as np
import scipy.stats as stat

def BSM(S, X, T, r, vol, option):
  d1 = (np.log(S/X) + (r+(1/2*vol)**2)*T)/vol*np.sqrt(T)
  d2 = d1 - vol*np.sqrt(T)

  if option == 'call':
    BSM_price = S*stat.norm.cdf(d1) - X*np.exp(-r*T)*stat.norm.cdf(d2)
  else :
    BSM_price = -S*stat.norm.cdf(-d1) + X*np.exp(-r*T)*stat.norm.cdf(-d2)
  return(BSM_price)
```

S = 기초자산 현재가격
X = 행사가격
T = 잔존만기(residual values)
r = risk free rate
vol = volatility
option = call or put option

관찰된 변수들과 주어진 vol, 블랙숄즈 모형을 통해 option price를 계산한다.

| 24 | BSM(100, 100, 0.5, 0.035, 0.2, 'call') |
|---|---|

6.461573471608148

| 25 | BSM(100, 100, 0.5, 0.035, 0.2, 'put') |
|---|---|

4.726797038115457

**④ 옵션의 내재변동성 계산**

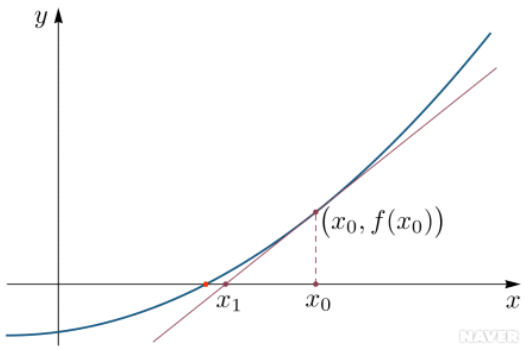
옵션의 민감도 지표(greeks) 중 하나인 베가는 기초자산의 변동성 변화에 따른 옵션가치의 민감도로 정의된다. 기초자산의 변동성 변화에 따라 옵션가치가 어떻게 변화하는지를 나타내는 값이라고 할 수 있다. 베가는 일반적으로 변동성의 1% 변화에 따른 옵션가격의 변화를 나타낸다.

$$\Lambda = \frac{\text{옵션의 가격변화}}{\text{기초자산의 가격변동성 변화}} = \frac{\partial c}{\partial \sigma} = S \times N(d_1) \times \sqrt{T}$$

만약 베가가 0.15인 경우 이는 변동성이 1% 변화하면 옵션가격이 0.15 증가함을 의미한다. 콜옵션이나 풋옵션 모두 베가의 값은 양수가 된다.

| 26 | # vega를 계산하는 함수<br>def vega(S, X, r, vol, T):<br>  d1 = (np.log(S/X) + (r+(1/2*vol)**2)*(T))/vol*np.sqrt(T)<br>  vega = S*np.sqrt(T)*stat.norm.cdf(d1)<br>  return(vega) |
|---|---|

내재변동성(implied volatility)이란 옵션시장에서 거래되고 있는 옵션가격을 가지고 역으로 계산한 변동성을 말한다. 즉, 시장의 옵션가격들이 반영하고 있는 변동성을 계산하는 것이다.



내재변동성 계산은 시행착오법으로 계산하는데, 시행착오를 축소하기 위해 뉴턴-랩슨법(Newton-Raphson Method)을 이용한다.

27
```
def implied_vol(S, X, r, T, market_price, option):
  max_iter = 300
  vol_old = 0.2    #initial value

  for i in range(max_iter):
    BSM_price = BSM(S=S, X=X, T=T, r=r, vol=vol_old, option=option)
    vega_value = vega(S=S, X=X, r=r, vol=vol_old, T=T)
    vol_new = vol_old - (BSM_price- market_price)/vega_value
    epsilon = abs(vol_new-vol_old)

    if epsilon < 0.00000001:
      break
    vol_old = vol_new

  implied_vol = vol_old
  return(implied_vol)
```

28
```
S, X, T, r = 100, 100, 0.5, 0.035
market_price = 7.5
implied_vol_est = implied_vol(S, X, r, T, market_price, 'call')
print("Implied Volatility is : ", implied_vol_est)
```

Implied Volatility is :  0.23774222208475462

**3) 제곱근 확산(Square-root diffusion) 모형**

- 단기 이자율이나 변동성 모형에 적용되는 평균 회귀 과정(mean-reverting processes)
- 평균 회귀 과정에 Cox, Ingersoll, and Ross가 제안한 제곱근 확산 모형이 많이 사용

$dx_t = k(\theta - x_t)dt + \sigma\sqrt{x_t}dZ_t)$

여기서, $x_t$ : 시간 $t$에서의 확률 과정 값
$k$ : 평균회귀계수
$\theta$ : 확률과정의 장기 평균
$\sigma$ : 고정변동성
$Z$ : 표준 브라운 운동

Euler discretization for square-root diffusion

$$\tilde{x}_t = \tilde{x}_s + \kappa(\theta - \tilde{x}_s^+)\Delta t + \sigma\sqrt{\tilde{x}_s^+}\sqrt{\Delta t}z_t$$

$$x_t = \tilde{x}_t^+$$

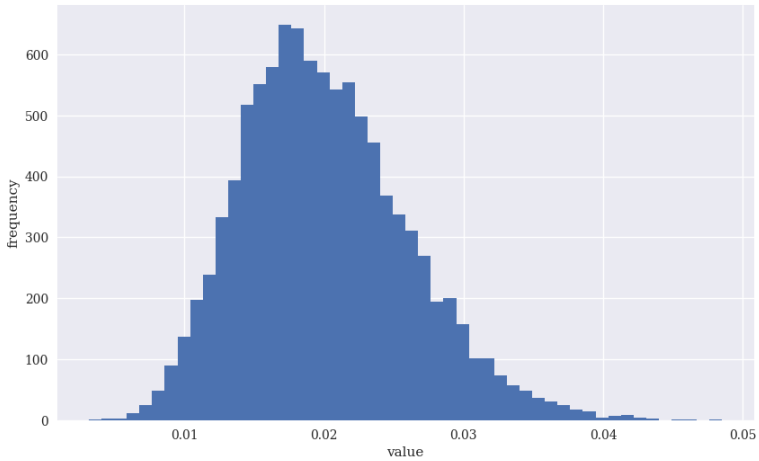
제곱근 확산 과정은 $x_t$값이 항상 양수로 유지된다는 특성
단순 오일러 방식으로 이산화하면 음수가 될 수 있으므로 원래의 시뮬레이션에서 양수 값만

취하는 방식을 사용

| | |
|---|---|
| 29 | x0 = 0.05 # The initial value (e.g., for a short rate)<br>kappa = 3.0 # The mean reversion factor<br>theta = 0.02 # The long-term mean value<br>sigma = 0.1 # The volatility factor<br>I = 10000<br>M = 50<br>dt = T / M |

| | |
|---|---|
| 30 | def srd_euler():<br>  xh = np.zeros((M + 1, I))<br>  x = np.zeros_like(xh)<br>  xh[0] = x0<br>  x[0] = x0<br>  for t in range(1, M + 1):<br>  xh[t] = (xh[t - 1] +<br>  kappa * (theta - np.maximum(xh[t - 1], 0)) * dt +<br>  sigma * np.sqrt(np.maximum(xh[t - 1], 0)) *<br>  math.sqrt(dt) * npr.standard_normal(I))<br>  x = np.maximum(xh, 0)<br>  return x<br>x1 = srd_euler() |

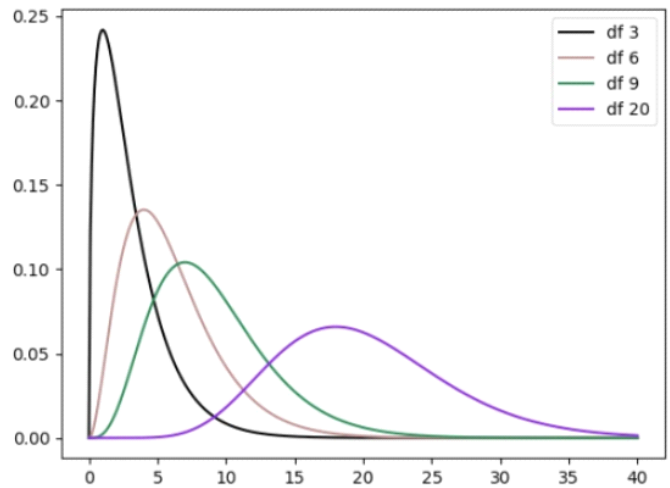
| | |
|---|---|
| 31 | plt.figure(figsize=(10, 6))<br>plt.hist(x1[-1], bins=50)<br>plt.xlabel('value')<br>plt.ylabel('frequency'); |



| | |
|---|---|
| 32 | plt.figure(figsize=(10, 6))<br>plt.plot(x1[:, :20], lw=1.5)<br>plt.xlabel('time')<br>plt.ylabel('index level'); |


현재 값(0.05)이 장기평균(θ=0.02) 보다 높기 때문에 표류 경향(drift)이 음수가 되고 장기적으로는 평균으로 수렴하는 모습을 나타낸다.

그런데 이자율이나 변동성은 정규분포보다는 카이제곱 분포를 따른다고 알려진다. 카이제곱 분포는 자유도에 따라 다음과 같은 분포 모양을 가진다.




카이제곱 분포는 자유도가 작을수록 평균이 0에 근접하고 자유도가 커질수록 정규분포와 차이가 적어진다.

다음 수식은 제곱근 확산 과정을 자유도(df)와 비중심 인수(nc)를 가진 비중심 카이제곱 분포로 표현한 것이다.

$$df = \frac{4\theta\kappa}{\sigma^2}, \quad nc = \frac{4\kappa e^{-\kappa\Delta t}}{\sigma^2(1 - e^{-\kappa\Delta t})} x_s$$

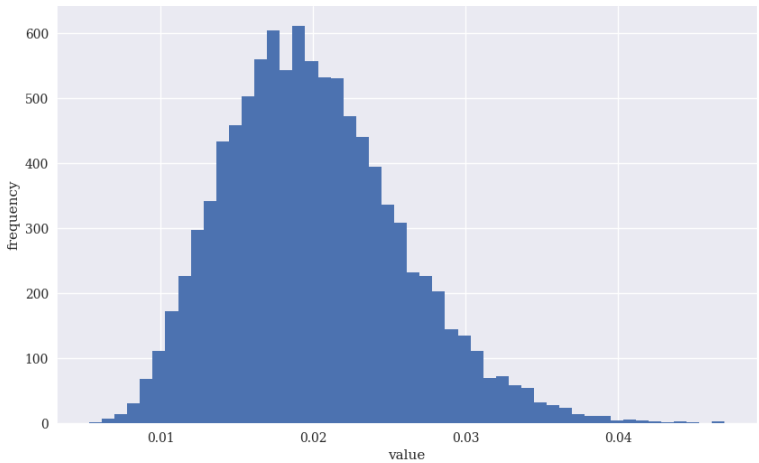
제곱근 확산 과정의 정확한 이산화

$$x_t = \frac{\sigma^2(1 - e^{-\kappa\Delta t})}{4\kappa}\chi'^2_d\left(\frac{4\kappa e^{-\kappa\Delta t}}{\sigma^2(1 - e^{-\kappa\Delta t})}x_s\right)$$

위의 이산화 과정을 파이썬으로 구현하면 다음과 같다.

33
```
def srd_exact():
    x = np.zeros((M + 1, I))
    x[0] = x0
    for t in range(1, M + 1):
        df = 4 * theta * kappa / sigma ** 2
        c = (sigma ** 2 * (1 - np.exp(-kappa * dt))) / (4 * kappa)
        nc = np.exp(-kappa * dt) / c * x[t - 1]
        x[t] = c * npr.noncentral_chisquare(df, nc, size=I)
    return x
x2 = srd_exact()
```

34
```
plt.figure(figsize=(10, 6))
plt.hist(x2[-1], bins=50)
plt.xlabel('value')
plt.ylabel('frequency');
```




35
```
plt.figure(figsize=(10, 6))
plt.plot(x2[:, :10], lw=1.5)
plt.xlabel('time')
plt.ylabel('index level');
```


이산화 방식의 차이를 통계치로 비교해보면 큰 차이는 나타나지 않는다.

| 36 | print_statistics(x1[-1], x2[-1]) |
|---|---|

```
      statistic       data set 1       data set 2
--------------------------------------------------
           size      10000.000        10000.000
            min          0.003            0.005
            max          0.049            0.047
           mean          0.020            0.020
            std          0.006            0.006
           skew          0.531            0.532
       kurtosis          0.294            0.274
```

실행 속도 면에서는 표준정규분포 방식보다 비중심 카이제곱 분포 방식이 계산의 부담이 크다. 다음은 더 많은 경로를 시뮬레이션하였을 때 속도차이다.

| 37 | I = 250000<br>%time x1 = srd_euler() |
|---|---|

```
CPU times: user 506 ms, sys: 125 ms, total: 631 ms
Wall time: 649 ms
```

| 38 | %time x2 = srd_exact() |
|---|---|

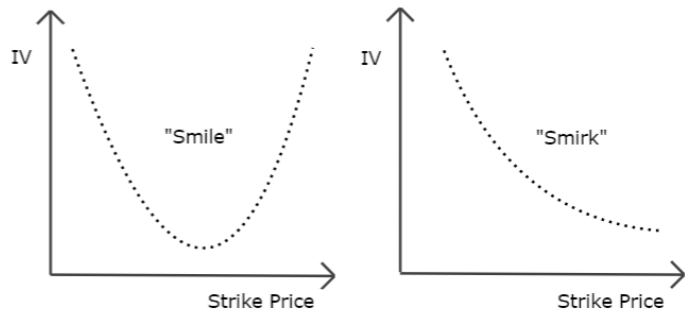
```
CPU times: user 1.96 s, sys: 56.6 ms, total: 2.02 s
Wall time: 2.06 s
```

| 39 | print_statistics(x1[-1], x2[-1])<br>x1 = 0.0; x2 = 0.0 |
|---|---|

| statistic | data set 1 | data set 2 |
|---|---|---|
| size | 250000.000 | 250000.000 |
| min | 0.002 | 0.003 |
| max | 0.071 | 0.055 |
| mean | 0.020 | 0.020 |
| std | 0.006 | 0.006 |
| skew | 0.563 | 0.579 |
| kurtosis | 0.491 | 0.520 |

**4) 확률적 변동성(Stochastic Volatility) 모형**

- Black-Scholes 모형은 옵션의 연구 및 거래에 있어서 가장 중추적인 역할을 담당
- 실증적 연구 결과 Black-Scholes 모형으로 추출한 내재변동성은 대체로 행사가격에 따라 체계적인 형태를 띠는 경향 - 변동성 스마일(volatility smile)
- 변동성 스마일(volatility smile)은 행사가격별로 내재변동성이 다르게 나타나는 모양을 일컫는 말. 등가격(ATM) 옵션의 내재변동성보다 내가격(ITM) 옵션과 외가격(OTM) 옵션의 내재변동성이 높게 형성되어 그림으로 나타내면 웃는 모양인 변동성 스마일(volatility smile) 형태를 나타내기도 하고, OTM옵션으로 갈수록 내재변동성이 커져서 기울어진 변동성 스머크(volatility smirk)의 형태가 나타나는 현상




- 변동성 스마일 현상에 대한 원인으로 옵션 연구자들은 “기초자산의 변동성이 옵션의 잔여만기 동안에 고정되어 있다”는 Black-Scholes 모형의 가정이 실제 금융시장의 자산가격의 운동과 부합하지 않는다는 사실에 주목
- 변동성 스마일 현상이 나타나는 이유를 설명하는 이론 중 기초자산의 수익률은 자산가격의 변동에 따라 임의로 변한다는 확률적 변동성(stochastic volatility) 모형이 있다. 확률적 변동성 모형 중 가장 인기 있는 모형 중 하나는 Heston(1993) 모형이다.
- Heston 모형의 확률 미분 방정식

$$
\begin{aligned}
dS_t &= rS_t dt + \sqrt{v_t} S_t dZ_t^1 \\
dv_t &= \kappa_v(\theta_v - v_t)dt + \sigma_v \sqrt{v_t} dZ_t^2 \\
dZ_t^1 dZ_t^2 &= \rho
\end{aligned}
$$

위의 식에서 인수 ρ는 두 표준 브라운 운동($Z_t^1, Z_t^2$)의 순간 상관계수

상관계수로 인해 시장이 하락할 때 변동성이 증가하고 시장이 상승할 때 변동성이 감소하는

레버리지 효과(leverage effect) 설명이 가능

Heston 모형에 대해 다음과 같은 인수 값 부여

| | |
|---|---|
| 40 | S0 = 100.<br>r = 0.05<br>v0 = 0.1 # Initial (instantaneous) volatility value<br>kappa = 3.0<br>theta = 0.25<br>sigma = 0.1<br>rho = 0.6 # Fixed correlation between the two Brownian motions<br>T = 1.0 |

두 확률 과정의 상관관계를 구하려면 상관행렬에 대한 Cholesky 분해가 필요

| | |
|---|---|
| 41 | corr_mat = np.zeros((2, 2))<br>corr_mat[0, :] = [1.0, rho]<br>corr_mat[1, :] = [rho, 1.0]<br>cho_mat = np.linalg.cholesky(corr_mat) |

| | |
|---|---|
| 42 | cho_mat # Cholesky decomposition and resulting matrix |

```
array([[1. , 0. ],
       [0.6, 0.8]])
```

시뮬레이션에 사용될 난수를 미리 생성

| | |
|---|---|
| 43 | M = 50<br>I = 10000<br>ran_num = npr.standard_normal((2, M + 1, I)) |

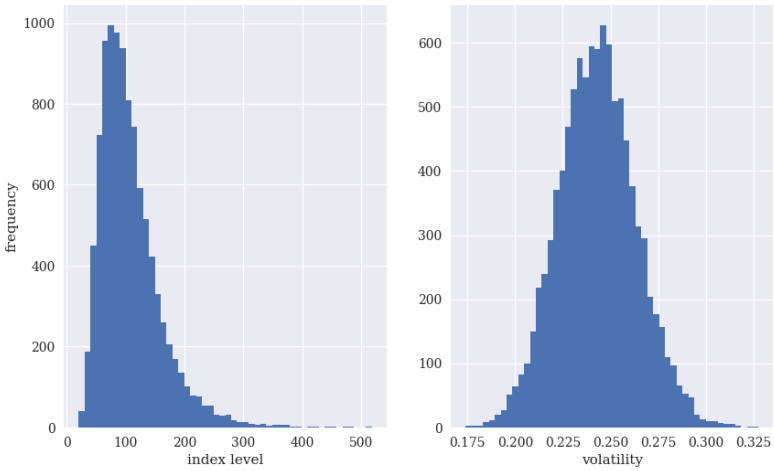
제곱근 확산 과정 모형에 기반한 변동성 과정에 대해서는 상관관계를 고려한 오일러 이산화 방법을 사용

| | |
|---|---|
| 44 | dt = T / M<br>v = np.zeros_like(ran_num[0])<br>vh = np.zeros_like(v)<br>v[0] = v0<br>vh[0] = v0<br>for t in range(1, M + 1):<br>ran = np.dot(cho_mat, ran_num[:, t, :])<br>vh[t] = (vh[t - 1] +<br>kappa * (theta - np.maximum(vh[t - 1], 0)) * dt +<br>sigma * np.sqrt(np.maximum(vh[t - 1], 0)) *<br>math.sqrt(dt) * ran[1])<br>v = np.maximum(vh, 0) |

주가 과정도 상관관계를 고려하고 기하 브라운 운동 오일러 방식으로 이산화

| | |
|---|---|
| 45 | S = np.zeros_like(ran_num[0])<br>S[0] = S0<br>for t in range(1, M + 1):<br>ran = np.dot(cho_mat, ran_num[:, t, :])<br>S[t] = S[t - 1] * np.exp((r - 0.5 * v[t]) * dt +<br>np.sqrt(v[t]) * ran[0] * np.sqrt(dt)) |

| | |
|---|---|
| 46 | fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))<br>ax1.hist(S[-1], bins=50)<br>ax1.set_xlabel('index level')<br>ax1.set_ylabel('frequency')<br>ax2.hist(v[-1], bins=50)<br>ax2.set_xlabel('volatility'); |




| | |
|---|---|
| 47 | print_statistics(S[-1], v[-1]) |

| statistic | data set 1 | data set 2 |
|---|---|---|
| size | 10000.000 | 10000.000 |
| min | 20.556 | 0.174 |
| max | 517.798 | 0.328 |
| mean | 107.820 | 0.243 |
| std | 51.316 | 0.020 |
| skew | 1.578 | 0.124 |
| kurtosis | 4.316 | 0.048 |

시뮬레이션 결과의 통계치를 보면 최종 주가의 최댓값이 기하 브라운 운동 모형보다 크게 나타남. 이는 변동성이 고정 값이 아니라 지속적으로 증가하였기 때문임

각 확률 과정의 시뮬레이션 경로를 10개씩 그래프로 표시하면 다음과 같다. 변동성 과정은 양의 표류 경향을 보이며 0.25로 수렴하는 모습을 나타낸다.

| | |
|---|---|
| 48 | fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True,<br>figsize=(10, 6))<br>ax1.plot(S[:, :10], lw=1.5)<br>ax1.set_ylabel('index level')<br>ax2.plot(v[:, :10], lw=1.5)<br>ax2.set_xlabel('time')<br>ax2.set_ylabel('volatility'); |


**5) 점프 확산(Jump-Diffusion) 모형**

- 시장에서 관측되는 특징 중 하나는 자산 가격이나 변동성이 점프하는 현상
- 1976년 머튼은 블랙-숄즈-머튼 모형을 개선한 점프 확산 모형을 발표하였고, 이후에도 다양한 형태의 점프 확산 모형이 제시됨
- 머튼 점프 확산 모형의 확률 미분 방정식

$dS_t = (r - r_j)S_t dt + \sigma S_t dZ_t + J_t S_t dN_t$

여기서, $S_t$ : 시간 $t$에서의 주가
$r$ : 고정 무위험 단기 이자율
$r_j$ : 점프의 위험 중립성을 보존하기 위한 표류계수 수정
$\sigma$ : $S$의 고정변동성
$Z_t$ : 표준 브라운 운동
$J_t$ : 시간 $t$에서의 점프
$N_t$ : 강도 $\lambda$를 가지는 포아송 과정

- 다음은 점프 확산 과정의 오일러 이산화 결과다.

$S_t = S_{t-\Delta t}(e^{(r-r_j-\sigma^2/2)\Delta t+\sigma\sqrt{\Delta z_t^1}} + (e^{\mu_j+\delta z_t^2}-1)y_t)$

여기서, $z_t^n$ : 표준정규분포

$y_t$ : 인수 $\lambda$를 가지는 포아송 분포

다음과 같은 인수 값을 적용

| | |
|---|---|
| 49 | S0 = 100.<br>r = 0.05<br>sigma = 0.2<br>lamb = 0.75 # The jump intensity<br>mu = -0.6 # The mean jump size<br>delta = 0.25 # The jump volatility<br>T = 1.0 |

| | |
|---|---|
| 50 | M = 50<br>I = 10000<br>dt = T / M<br>rj = lamb * (math.exp(mu + 0.5 * delta ** 2) - 1) # The drift correction |

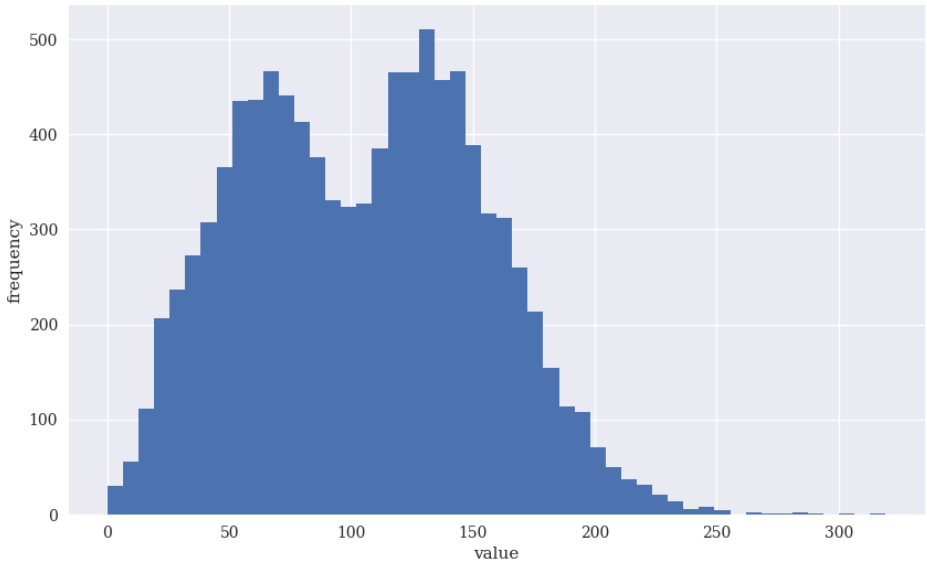
점프 확산 모형을 시뮬레이션하려면 3개의 독립적인 난수 집합이 필요

| | |
|---|---|
| 51 | S = np.zeros((M + 1, I))<br>S[0] = S0<br>sn1 = npr.standard_normal((M + 1, I)) # 표준정규분포 난수<br>sn2 = npr.standard_normal((M + 1, I)) # 표준정규분포 난수<br>poi = npr.poisson(lamb * dt, (M + 1, I)) # 포아송분포 난수<br>for t in range(1, M + 1, 1):<br>S[t] = S[t - 1] * (np.exp((r - rj - 0.5 * sigma ** 2) * dt +<br>sigma * math.sqrt(dt) * sn1[t]) +<br>(np.exp(mu + delta * sn2[t]) - 1) *<br>poi[t])<br>S[t] = np.maximum(S[t], 0) |

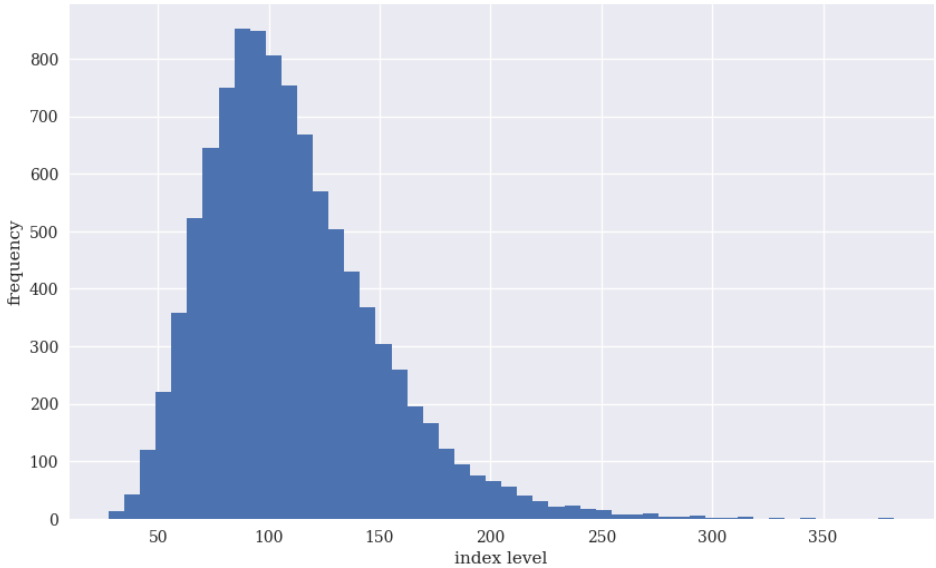
점프의 평균값에 대해 큰 음수를 가정하였기 때문에 주가를 시뮬레이션 하면 전형적인 로그 정규분포 보다 음으로 봉우리가 하나 있고 양으로 봉우리가 하나 있는 분포 모양이 나타남

| | |
|---|---|
| 52 | plt.figure(figsize=(10, 6))<br>plt.hist(S[-1], bins=50)<br>plt.xlabel('value')<br>plt.ylabel('frequency'); |

<점프 확산 모형 시뮬레이션 만기 결과 값>




<로그 정규분포 모형 시뮬레이션 만기 결과 값>




주가가 아래로 점프하는 현상은 다음과 같이 10개의 시뮬레이션 경로에서도 확인이 가능하다.

| | |
|---|---|
| 53 | plt.figure(figsize=(10, 6))<br>plt.plot(S[:, :10], lw=1.5)<br>plt.xlabel('time')<br>plt.ylabel('index level'); |


<옵션 거래자들의 현실>

- Heston식의 확률적 변동성 모형 등이 Black-Scholes 모형의 가격괴리를 상당히 개선
- 그러나 여러 옵션모형들 간의 비교분석에서 밝혀진 실증적 결과에도 불구하고 여태껏 어떠한 옵션모형도 전통적인 Black-Scholes 모형을 신뢰성 있게 대체하고 있다고 판정하기에는 섣부른 실정
- Black-Scholes 모형은 시장참여자들뿐만 아니라 옵션연구자들 사이에서도 가장 빈번히 사용되고 있는 옵션모형
- Black-Scholes 모형이 선호되는 이유
  - 첫째, Black-Scholes 모형은 확률적 변동성 모형에 비해 매우 단순, 모형의 적합성과 편의성이라는 상충 속에서 시장참여자는 확률적 변동성 모형 보다는 Black-Scholes 모형을 선호
  - 둘째, Black-Scholes 모형은 확률적 변동성 모형에 비해 훨씬 인색(parsimonious)하다. Black-Scholes 모형에서 관측되지 않는 모수의 수는 기초자산의 변동성 하나이나 Heston의 확률적 변동성 모형의 경우 관측되지 않은 모수의 수는 다섯 개로 확률적 변동성 모형을 옵션거래에 이용하고자 하는 자는 많은 모수를 추정해야 하는 어려움과 아울러 모수추정의 오류로부터 발생하는 문제에도 직면, 또한 확률적 변동성 모형의 추정 모수는 매우 비현실적이고 불안정한 값을 가지는 경향이 있는데, 이는 적시에 가격을 고시하고 시장을 조성해야 하는 옵션 딜러의 활동에 매우 심각한 제약으로 작용
- 옵션 딜러들은 Black-Scholes 모형의 가격괴리에 대응하여 Black-Scholes 모형을 토대로 한 '변동성 스마일 기법'(volatility smile technique)이란 임시변통 적이지만 아주 효과적인 방안을 개발하여 사용. 이 기법은 Black-Scholes 모형으로 각 옵션의 내재변동성을 행사가격별로 구하고 옵션의 가격도(moneyness) 정도에 따라 추정한 변동성을 Black-Scholes 모형에 투여하여 옵션의 가격을 평가하는 기법. 변동성 스마일 기법은 이론적 근거가 희박하지만, Black-Scholes 모형의 가격오차를 효과적으로 보완하는 방법으로 시장참여자들에 의해 폭넓게 활용되어 왔음.

# 3. Real Data Statistics

I can prove anything by statistics except the truth. – George Canning

## 3-1. 정규성 검증

평균-분산 포트폴리오 이론, 자본자산 가격결정 모형, 효율적 시장가설, 옵션 가격결정 모형 등 대부분의 금융이론은 주식시장 수익률이 정규분포라는 가정에 기반함.

### 1) 벤치마크 자료 분석

```
1  import math
   import numpy as np
   import scipy.stats as scs
   import statsmodels.api as sm
   from pylab import mpl, plt
```

```
2  plt.style.use('seaborn-v0_8')
   mpl.rcParams['font.family'] = 'serif'
   %matplotlib inline
```

```
3  def gen_paths(S0, r, sigma, T, M, I):
       dt = float(T) / M
       paths = np.zeros((M + 1, I), np.float64)
       paths[0] = S0
       for t in range(1, M + 1):
           rand = np.random.standard_normal(I)
           rand = (rand - rand.mean()) / rand.std()
           paths[t] = paths[t -1] * np.exp((r - 0.5 * sigma ** 2) * dt +
   sigma * math.sqrt(dt) * rand)
       return paths
```

기하 브라운 운동 모형에 대한 몬테카를로 경로를 생성

인수

S0 : float, 초기주가/지수 값　　r : float, 고정 단기 이자율

sigma : float, 고정 변동성　　T : float, 만기까지 시간

M : int, 시간 구간의 수　　I : int, 생성한 경로의 수

반환값

path : ndarray, shape (M + 1, I) # 주어진 인수

```
S0 = 100
r = 0.05
sigma = 0.2
T = 1.0
M = 250
I = 250000
np.random.seed(1000)
```

```
paths = gen_paths(S0, r, sigma, T, M, I)
```

```
S0 * math.exp(r * T)
```

105.12710963760242
시뮬레이션 기댓값

```
paths[-1].mean()
```

105.13190981518643
마지막 주가의 평균수치

```
plt.figure(figsize=(10, 6))
plt.plot(paths[:, :20])
plt.grid(True)
plt.xlabel('time steps')
plt.ylabel('index level')
```

| 9 | paths[:, 0].round(4) |
|---|---|

| 10 | log_returns = np.log(paths[1:] / paths[0:-1]) |
|---|---|

log 수익률을 계산하여 ndarray 객체를 생성

| 11 | log_returns[:, 0].round(4) |
|---|---|

| 12 | def print_statistics(array):<br>    ''' 통계치 출력<br>    인수<br>    ====<br>    array : ndarray 통계치를 계산할 자료<br>    '''<br>    sta = scs.describe(array)<br>    print("%14s %15s" % ('statistic', 'value'))<br>    print(30 * "-")<br>    print("%14s %15.5f" % ('size', sta[0]))<br>    print("%14s %15.5f" % ('min', sta[1][0]))<br>    print("%14s %15.5f" % ('max', sta[1][1]))<br>    print("%14s %15.5f" % ('mean', sta[2]))<br>    print("%14s %15.5f" % ('std', np.sqrt(sta[3])))<br>    print("%14s %15.5f" % ('skew', sta[4]))<br>    print("%14s %15.5f" % ('kurtosis', sta[5])) |
|---|---|

| 13 | print_statistics(log_returns.flatten()) |
|---|---|

```
      statistic            value
------------------------------
           size   62500000.00000
            min         -0.07365
            max          0.07190
           mean          0.00012
            std          0.01265
           skew          0.00042
       kurtosis          0.00050
```

62,500,000개의 자료가 -0.074~+0.072 사이에 있음
일간 평균 수익률과 표준편차를 연율화하면 0.05와 0.20이 되리라 예상
왜도(skew)는 0에 가깝고, 첨도(kurtosis)도 0에 가까움

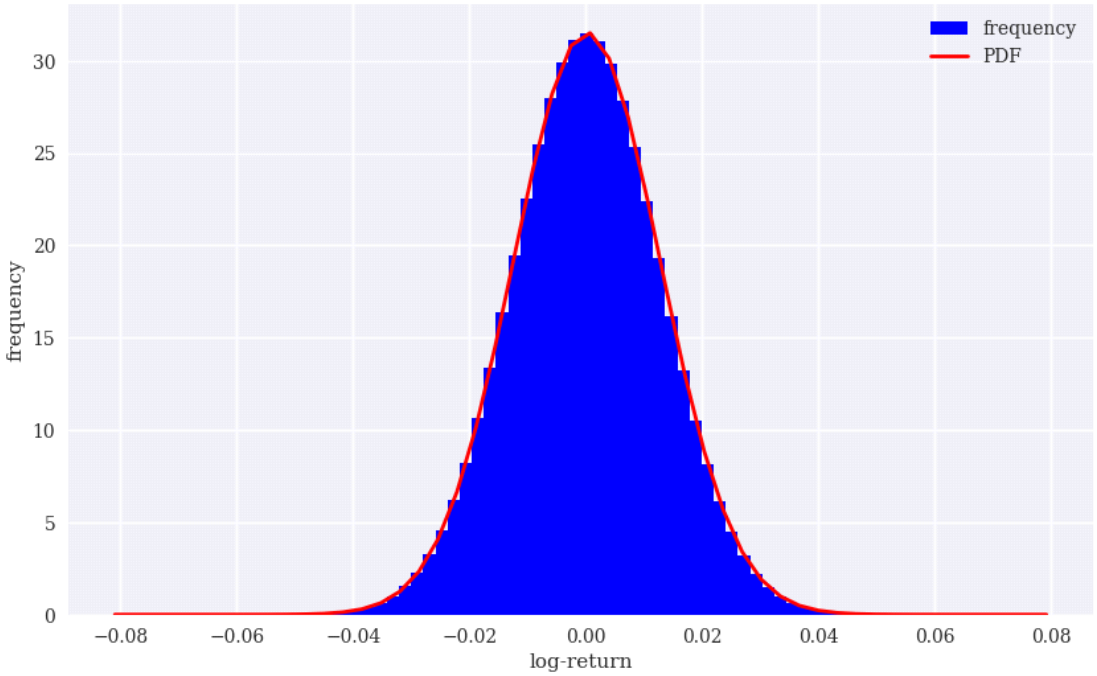
| 14 | log_returns.mean() * M + 0.5 * sigma ** 2 |
|---|---|

0.05000000000000017
연간 평균 수익률(Itô term 수정 후)

| 15 | log_returns.std() * math.sqrt(M) |
|---|---|

0.19999999999999987
연간 변동성

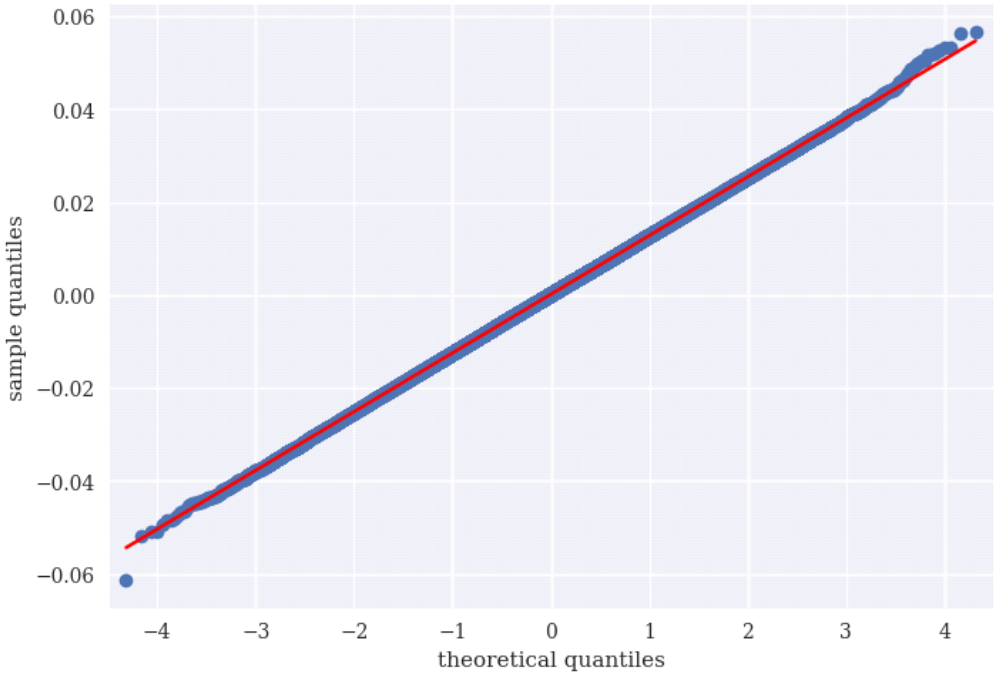
| | |
|---|---|
| 16 | plt.figure(figsize=(10, 6))<br>plt.hist(log_returns.flatten(), bins=70, density=True, label='frequency', color='b')<br>plt.grid(True)<br>plt.xlabel('log-return')<br>plt.ylabel('frequency')<br>x = np.linspace(plt.axis()[0], plt.axis()[1])<br>plt.plot(x, scs.norm.pdf(x, loc=r / M, scale=sigma / np.sqrt(M)), 'r', lw=2.0, label='pdf')<br>plt.legend() |




로그 수익률의 히스토그램과 정규분포의 확률 밀도함수(probability density function; PDF) 비교 – 시뮬레이션된 로그수익률 수치와 정규분포가 일치하는 모습을 나타냄

정규성을 시각적으로 검증하는 다른 방법은 분위수 대조도(quantile-quantile plots), Q-Q 플롯을 사용하는 것.
Q-Q 플롯은 샘플값의 샘플 분위수(quantile)와 정규분포 상의 이론적 분위수를 비교한 것으로 샘플 자료가 정규분포를 나타낸다면 대부분의 분위수 값이 일직선을 나타내게 됨

| | |
|---|---|
| 17 | sm.qqplot(log_returns.flatten()[::500], line='s')<br>plt.grid(True)<br>plt.xlabel('theoretical quantiles')<br>plt.ylabel('sample quantiles') |



<로그 수익률의 분위수 대조도(Q-Q 플롯)>

주어진 데이타가 정규분포인지 수치로 검정할 수 있다.

- 왜도 검정(Skewness test (skewtest()): 실제 분포가 정규분포와 근접하면 왜도가 0에 가까운 수치가 나타남
- 첨도 검정(Kurtosis test (kurtosistest()): 실제 분포가 정규분포와 근접하면 첨도가 0에 가까운 수치가 나타남
- 정규성 검정(Normality test (normaltest()): Student's t-test, 일원분산분석 및 이원분산분석(one-way and two-way ANOVA) 조합

| | |
|---|---|
| 18 | def normality_tests(arr):<br>''' 주어진 데이타가 정규분포인지 검정<br>인수<br>====<br>array: ndarray 통계치를 생성할 대상<br><br>'''<br>print("Skew of data set %14.3f" %scs.skew(arr))<br>print("Skew test p-value %14.3f" % scs.skewtest(arr)[1])<br>print("Kurt of data set %14.3f" % scs.kurtosis(arr))<br>print("Kurt test p-value %14.3f" % scs.kurtosistest(arr)[1])<br>print("Norm test p-value %14.3f" % scs.normaltest(arr)[1]) |

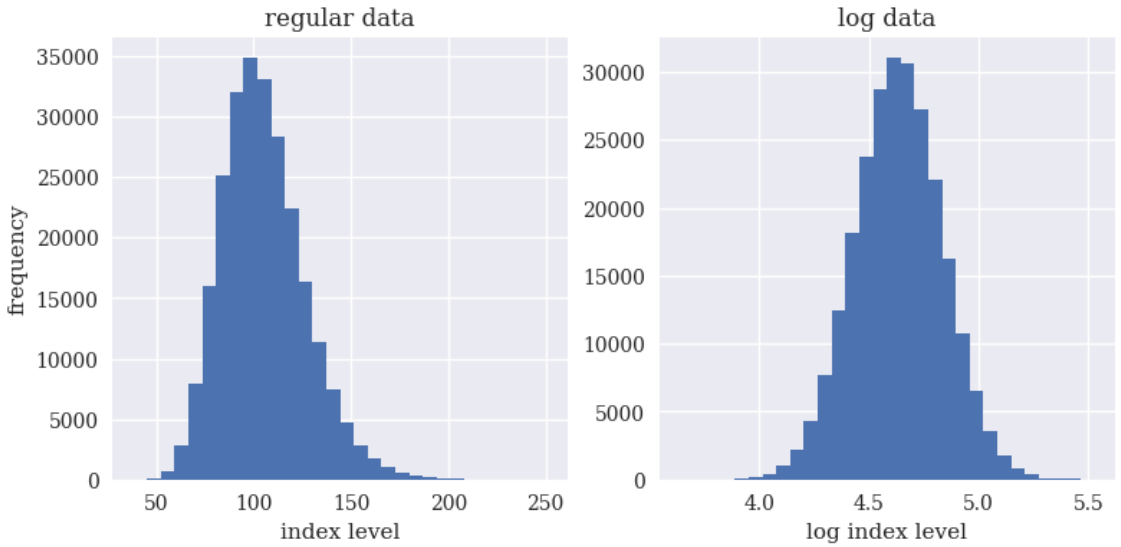
| | |
|---|---|
| 19 | normality_tests(log_returns.flatten()) |

| | |
|---|---|
| Skew of data set | 0.000 |
| Skew test p-value | 0.179 |
| Kurt of data set | 0.001 |
| Kurt test p-value | 0.415 |
| Norm test p-value | 0.291 |

정규성 검정결과 p-값이 0.05 이상이므로 로그 수익률이 정규분포임을 나타냄

몬테카를로 시뮬레이션한 주가 움직임 및 수익률 분석

| | |
|---|---|
| 20 | f, (ax1, ax2) = plt.subplots(1, 2, figsize=(9, 4))<br>ax1.hist(paths[-1], bins=30)<br>ax1.grid(True)<br>ax1.set_xlabel('index level')<br>ax1.set_ylabel('frequency')<br>ax1.set_title('regular data')<br>ax2.hist(np.log(paths[-1]), bins=30)<br>ax2.grid(True)<br>ax2.set_xlabel('log index level')<br>ax2.set_title('log data') |




| | |
|---|---|
| 21 | print_statistics(paths[-1]) |

| statistic | value |
|---|---|
| size | 250000.00000 |
| min | 38.07797 |
| max | 250.81963 |
| mean | 105.13191 |
| std | 21.26424 |
| skew | 0.61688 |
| kurtosis | 0.69162 |

주가의 최종 값 평균은 105에 가깝고 표준편차(변동성)는 20%에 가깝다.

| 22 | print_statistics(np.log(paths[-1])) |
|---|---|

```
      statistic                value
------------------------------
           size       250000.00000
            min            3.63964
            max            5.52473
           mean            4.63517
            std            0.20023
           skew           -0.00031
       kurtosis            0.00790
```

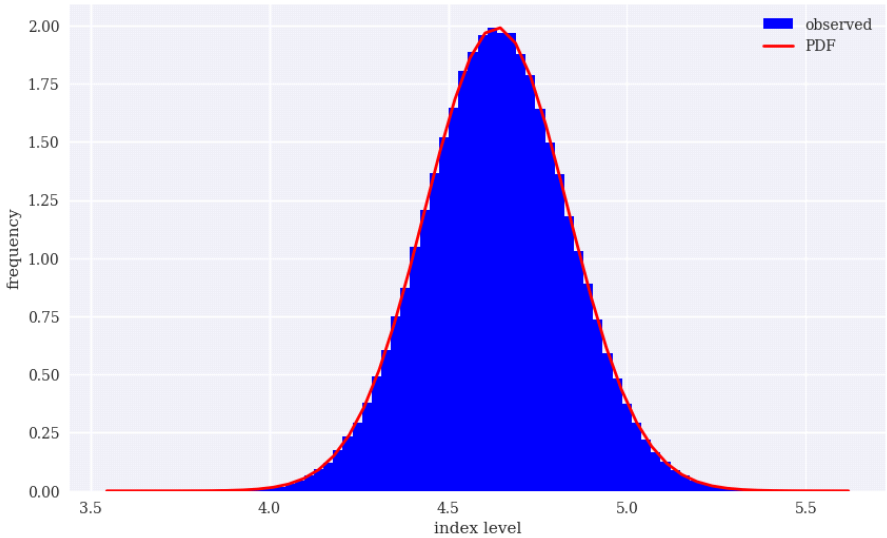
로그 변환한 값은 왜도와 첨도가 0에 가까워졌음

| 23 | normality_tests(np.log(paths[-1])) |
|---|---|

```
Skew of data set              -0.000
Skew test p-value              0.949
Kurt of data set               0.008
Kurt test p-value              0.418
Norm test p-value              0.719
```

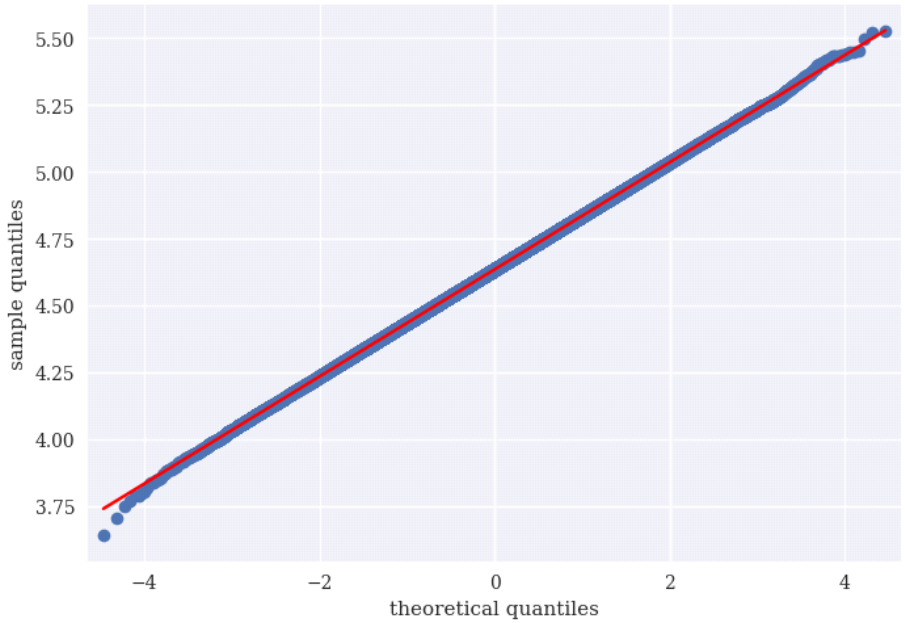
정규성 검정결과 p-값이 0.05 이상이므로 로그 수익률이 정규분포임을 나타냄

| 24 | plt.figure(figsize=(10, 6))<br>log_data = np.log(paths[-1])<br>plt.hist(log_data, bins=70, density=True, label='observed', color='b')<br>plt.grid(True)<br>plt.xlabel('index level')<br>plt.ylabel('frequency')<br>x = np.linspace(plt.axis()[0], plt.axis()[1])<br>plt.plot(x, scs.norm.pdf(x, log_data.mean(), log_data.std()), 'r', lw=2.0, label='PDF')<br>plt.legend() |
|---|---|




<로그 변환한 값의 히스토그램과 정규분포 확률 밀도함수>

| | |
|---|---|
| 25 | sm.qqplot(log_data, line='s')<br>plt.grid(True)<br>plt.xlabel('theoretical quantiles')<br>plt.ylabel('sample quantiles') |




<로그 변환한 값의 분위수 대조도(Q-Q 플롯)>

## 2) 현실 자료 분석

필요한 모듈 설치

```
26  !pip install -U finance-datareader
    !pip install pandas-datareader
    !pip install yfinance
```

필요한 모듈 불러오기

```
27  import pandas as pd
    import numpy as np
    import pandas_datareader as pdr
    import FinanceDataReader as fdr
    import yfinance as yf
```

날짜 셋팅

```
28  from datetime import datetime
    start = datetime(2000,1,1)
    end = datetime(2023,8,29)
```

Data 수집

```
29  from os import close
    data1 = pd.DataFrame()
    data2 = pd.DataFrame()
    data3 = pd.DataFrame()
    data4 = pd.DataFrame()
    data5 = pd.DataFrame()
    data6 = pd.DataFrame()

    data1 = yf.download('^GSPC', start, end) # S&P500
    data2 = yf.download('^GDAXI',start, end) # DAX
    data3 = yf.download('^KS11', start, end) # KOSPI
    data4 = yf.download('^N225', start, end) # NIKKEI
    data5 = yf.download('^HSI', start, end) # HANGSENG
    data6 = yf.download('^IXIC', start, end) # NASDAQ
```

```
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```

Data 관리(병합)

| | |
|---|---|
| 30 | data7 = pd.merge(data1['Adj Close'].to_frame(), data2['Adj Close'].to_frame(),left_index=True, right_index=True, how='inner')<br>data7.columns=['SP500','DAX']<br><br>data8 = pd.merge(data3['Adj Close'].to_frame(), data4['Adj Close'].to_frame(),left_index=True, right_index=True, how='inner')<br>data8.columns=['KOSPI','NIKKEI']<br><br>data9 = pd.merge(data5['Adj Close'].to_frame(), data6['Adj Close'].to_frame(),left_index=True, right_index=True, how='inner')<br>data9.columns=['HANGSENG','NASDAQ']<br><br>data10 = pd.merge(data7, data8,left_index=True, right_index=True, how='inner')<br>data = pd.merge(data10, data9,left_index=True, right_index=True, how='inner') |

Data 출력

| | |
|---|---|
| 31 | data.head() |

| Date | SP500 | DAX | KOSPI | NIKKEI | HANGSENG | NASDAQ |
|---|---|---|---|---|---|---|
| **2000-01-04** | 1399.420044 | 6586.950195 | 1059.040039 | 19002.859375 | 17072.820312 | 3901.689941 |
| **2000-01-05** | 1402.109985 | 6502.069824 | 986.309998 | 18542.550781 | 15846.719727 | 3877.540039 |
| **2000-01-06** | 1403.449951 | 6474.919922 | 960.789978 | 18168.269531 | 15153.230469 | 3727.129883 |
| **2000-01-07** | 1441.469971 | 6780.959961 | 948.650024 | 18193.410156 | 15405.629883 | 3882.620117 |
| **2000-01-11** | 1438.560059 | 6891.250000 | 981.330017 | 18850.919922 | 15862.099609 | 3921.189941 |

| | |
|---|---|
| 32 | data.tail() |

| Date | SP500 | DAX | KOSPI | NIKKEI | HANGSENG | NASDAQ |
|---|---|---|---|---|---|---|
| **2023-08-25** | 4405.709961 | 15631.820312 | 2519.139893 | 31624.279297 | 18119.390625 | 13590.650391 |
| **2023-08-28** | 4433.310059 | 15792.610352 | 2543.409912 | 32169.990234 | 18130.740234 | 13705.129883 |
| **2023-08-29** | 4497.629883 | 15930.879883 | 2552.159912 | 32226.970703 | 18484.029297 | 13943.759766 |
| **2023-08-30** | 4514.870117 | 15891.929688 | 2561.219971 | 32333.460938 | 18482.859375 | 14019.309570 |
| **2023-08-31** | 4507.660156 | 15947.080078 | 2556.270020 | 32619.339844 | 18382.060547 | 14034.969727 |

수집한 데이터 CSV파일로 저장

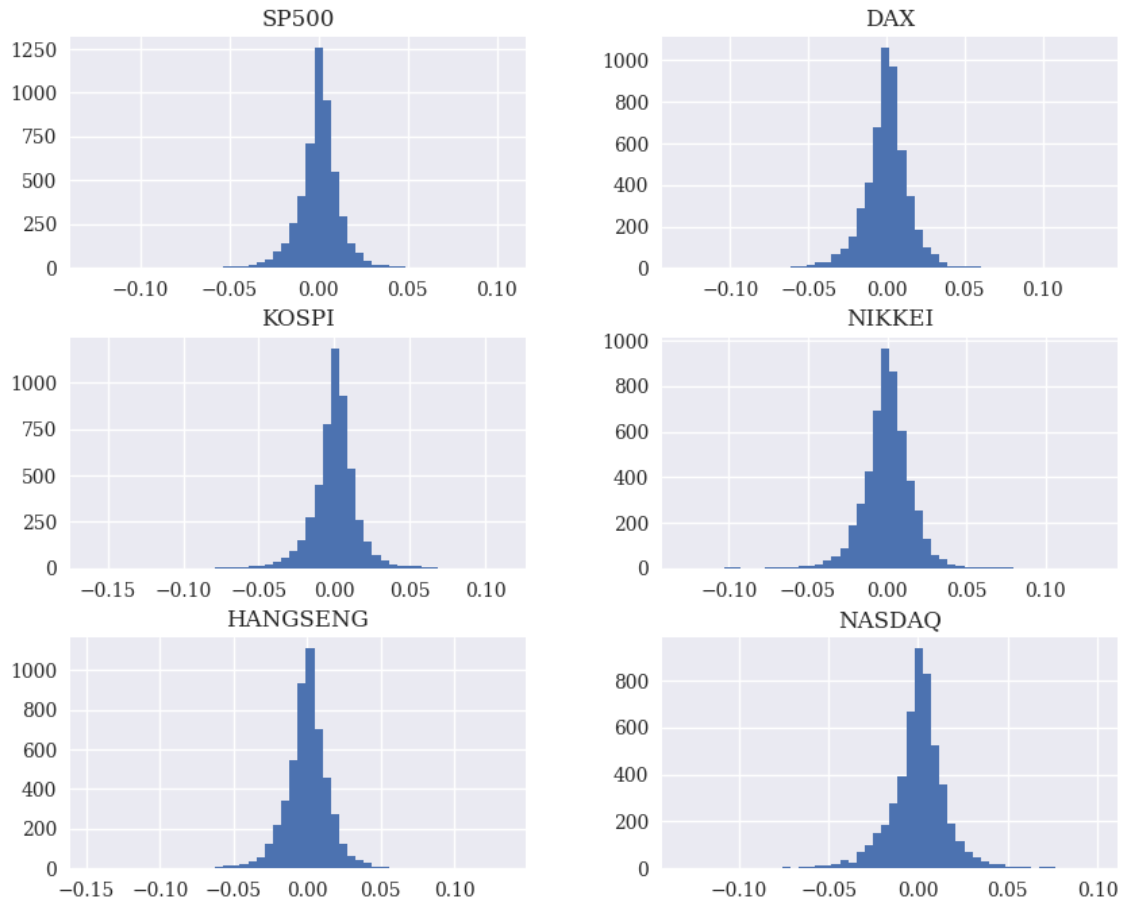
| | |
|---|---|
| 33 | data.to_csv('stocks.csv') |

| | |
|---|---|
| 34 | (data / data.iloc[0] * 100).plot(figsize=(12, 8)) |


| | |
|---|---|
| 35 | log_returns = np.log(data / data.shift(1))<br>log_returns.head() |

| Date | SP500 | DAX | KOSPI | NIKKEI | HANGSENG | NASDAQ |
|---|---|---|---|---|---|---|
| **2000-01-04** | NaN | NaN | NaN | NaN | NaN | NaN |
| **2000-01-05** | 0.001920 | -0.012970 | -0.071147 | -0.024521 | -0.074525 | -0.006209 |
| **2000-01-06** | 0.000955 | -0.004184 | -0.026215 | -0.020391 | -0.044749 | -0.039562 |
| **2000-01-07** | 0.026730 | 0.046182 | -0.012716 | 0.001383 | 0.016519 | 0.040872 |
| **2000-01-11** | -0.002021 | 0.016134 | 0.033869 | 0.035502 | 0.029200 | 0.009885 |

| | |
|---|---|
| 36 | log_returns.hist(bins=50, figsize=(10, 8)) |




| | |
|---|---|
| 37 | symbols = ['SP500', 'DAX', 'KOSPI', 'NIKKEI', 'HANGSENG', 'NASDAQ'] |

| | |
|---|---|
| 38 | for sym in symbols:<br>  print("\nResults for symbol %s" % sym)<br>  print(30 * "-")<br>  log_data = np.array(log_returns[sym].dropna())<br>  print_statistics(log_data) |

Results for symbol SP500

| statistic | value |
|---|---|
| size | 5168.00000 |
| min | -0.12765 |
| max | 0.10424 |
| mean | 0.00022 |
| std | 0.01317 |
| skew | -0.45398 |
| kurtosis | 9.54542 |

Results for symbol DAX

| statistic | value |
|---|---|
| size | 5168.00000 |
| min | -0.13055 |
| max | 0.13463 |
| mean | 0.00017 |
| std | 0.01545 |
| skew | -0.11604 |
| kurtosis | 6.26050 |

Results for symbol KOSPI

| statistic | value |
|---|---|
| size | 5168.00000 |
| min | -0.16115 |
| max | 0.11284 |
| mean | 0.00017 |
| std | 0.01564 |
| skew | -0.61368 |
| kurtosis | 8.73850 |

Results for symbol NIKKEI

| statistic | value |
|---|---|
| size | 5168.00000 |
| min | -0.12924 |
| max | 0.13235 |
| mean | 0.00010 |
| std | 0.01561 |
| skew | -0.49846 |
| kurtosis | 6.98470 |

Results for symbol HANGSENG

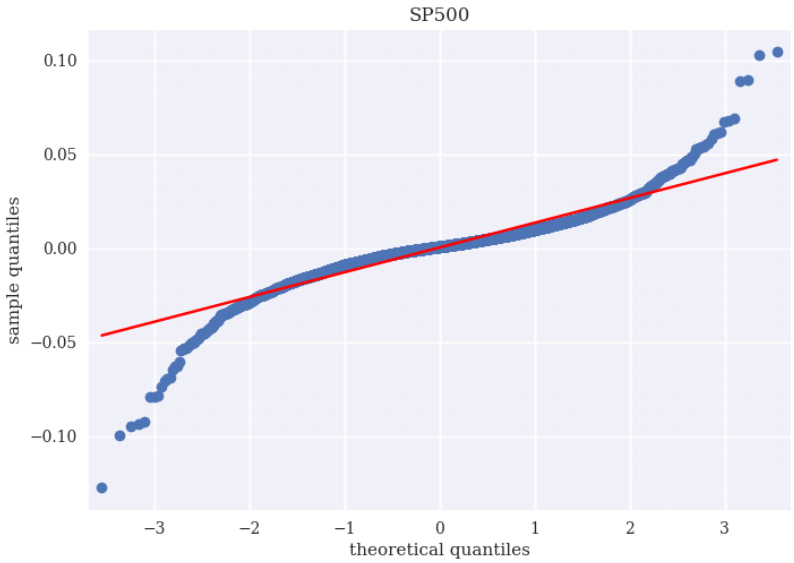
| statistic | value |
|---|---|
| size | 5168.00000 |
| min | -0.14695 |
| max | 0.13407 |
| mean | 0.00001 |
| std | 0.01582 |
| skew | -0.12110 |
| kurtosis | 8.56444 |

Results for symbol NASDAQ

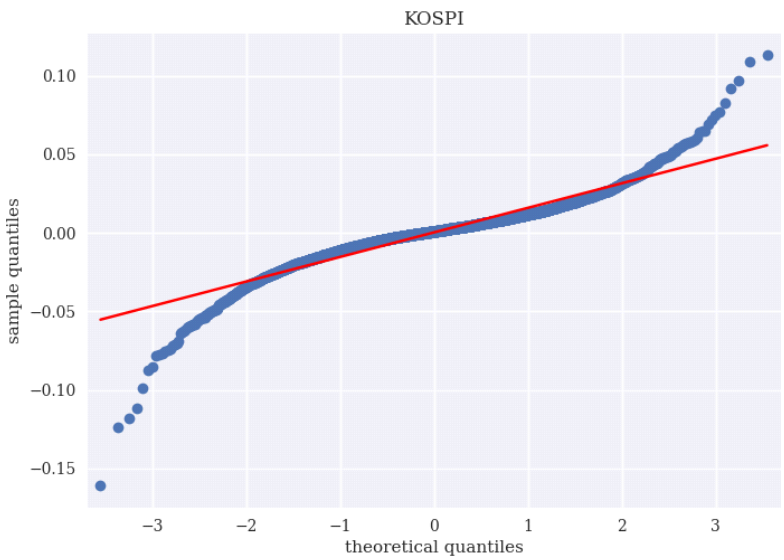
| statistic | value |
|---|---|
| size | 5168.00000 |
| min | -0.13149 |

| | |
|---|---|
| max | 0.09964 |
| mean | 0.00024 |
| std | 0.01669 |
| skew | -0.31114 |
| kurtosis | 5.21577 |

| | |
|---|---|
| 39 | sm.qqplot(log_returns['SP500'].dropna(), line='s')<br>plt.title('SP500')<br>plt.xlabel('theoretical quantiles')<br>plt.ylabel('sample quantiles') |




| | |
|---|---|
| 40 | sm.qqplot(log_returns['KOSPI'].dropna(), line='s')<br>plt.title('KOSPI')<br>plt.xlabel('theoretical quantiles')<br>plt.ylabel('sample quantiles') |



```
for sym in symbols:
    print("\nResults for symbol %s" % sym)
    print(32 * "-")
    log_data = np.array(log_returns[sym].dropna())
    normality_tests(log_data)
```

```
Results for symbol SP500
--------------------------------
Skew of data set          -0.454
Skew test p-value          0.000
Kurt of data set           9.547
Kurt test p-value          0.000
Norm test p-value          0.000

Results for symbol DAX
--------------------------------
Skew of data set          -0.116
Skew test p-value          0.001
Kurt of data set           6.261
Kurt test p-value          0.000
Norm test p-value          0.000

Results for symbol KOSPI
--------------------------------
Skew of data set          -0.614
Skew test p-value          0.000
Kurt of data set           8.739
Kurt test p-value          0.000
Norm test p-value          0.000

Results for symbol NIKKEI
--------------------------------
Skew of data set          -0.499
Skew test p-value          0.000
Kurt of data set           6.983
Kurt test p-value          0.000
Norm test p-value          0.000

Results for symbol HANGSENG
--------------------------------
Skew of data set          -0.121
Skew test p-value          0.000
Kurt of data set           8.567
Kurt test p-value          0.000
Norm test p-value          0.000

Results for symbol NASDAQ
--------------------------------
Skew of data set          -0.311
Skew test p-value          0.000
Kurt of data set           5.217
Kurt test p-value          0.000
Norm test p-value          0.000
```

## 3-2. 포트폴리오 최적화

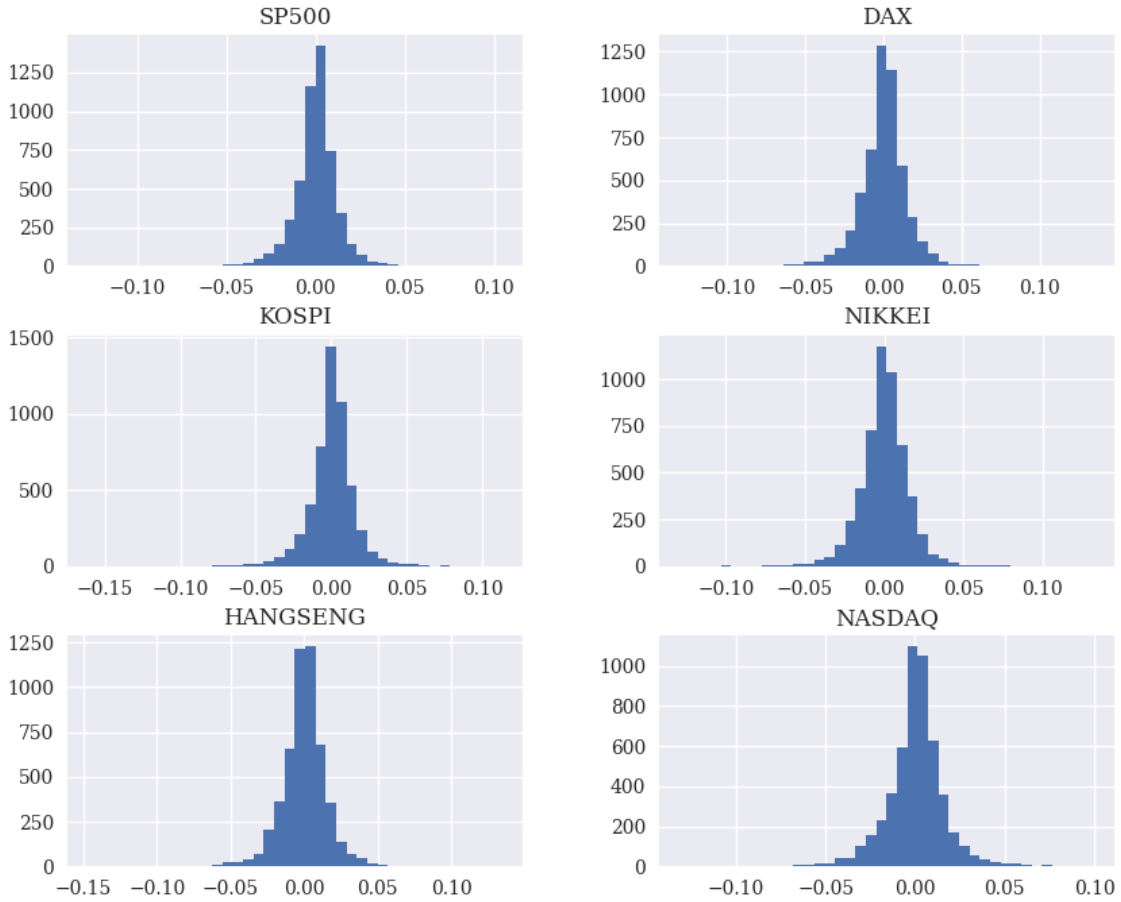
### 1) 자료 분석

| | |
|---|---|
| 42 | import pandas as pd<br>import numpy as np<br>pd.core.common.is_list_like = pd.api.types.is_list_like<br>import pandas_datareader.data as web<br>import matplotlib.pyplot as plt<br>%matplotlib inline |

| | |
|---|---|
| 43 | symbols = ['SP500', 'DAX', 'KOSPI', 'NIKKEI', 'HANGSENG', 'NASDAQ']<br>noa = len(symbols) |

| | |
|---|---|
| 44 | rets = np.log(data / data.shift(1)) |

| | |
|---|---|
| 45 | rets.hist(bins=40, figsize=(10, 8)); |




| | |
|---|---|
| 46 | rets.mean() * 252 |

SP500 0.056216
DAX 0.042631
KOSPI 0.042714
NIKKEI 0.025665
HANGSENG 0.002931
NASDAQ 0.061250
dtype: float64

| 47 | rets.cov() * 252 |
|---|---|

| | SP500 | DAX | KOSPI | NIKKEI | HANGSENG | NASDAQ |
|---|---|---|---|---|---|---|
| **SP500** | 0.043706 | 0.032185 | 0.012491 | 0.011053 | 0.014041 | 0.049918 |
| **DAX** | 0.032185 | 0.060162 | 0.022079 | 0.021903 | 0.024301 | 0.037417 |
| **KOSPI** | 0.012491 | 0.022079 | 0.061610 | 0.037411 | 0.038605 | 0.015856 |
| **NIKKEI** | 0.011053 | 0.021903 | 0.037411 | 0.061404 | 0.036957 | 0.012767 |
| **HANGSENG** | 0.014041 | 0.024301 | 0.038605 | 0.036957 | 0.063026 | 0.017458 |
| **NASDAQ** | 0.049918 | 0.037417 | 0.015856 | 0.012767 | 0.017458 | 0.070166 |

| 48 | rets.corr() |
|---|---|

| | SP500 | DAX | KOSPI | NIKKEI | HANGSENG | NASDAQ |
|---|---|---|---|---|---|---|
| **SP500** | 1.000000 | 0.627669 | 0.240742 | 0.213336 | 0.267715 | 0.901436 |
| **DAX** | 0.627669 | 1.000000 | 0.362645 | 0.360369 | 0.394689 | 0.575917 |
| **KOSPI** | 0.240742 | 0.362645 | 1.000000 | 0.608216 | 0.619474 | 0.241180 |
| **NIKKEI** | 0.213336 | 0.360369 | 0.608216 | 1.000000 | 0.593955 | 0.194512 |
| **HANGSENG** | 0.267715 | 0.394689 | 0.619474 | 0.593955 | 1.000000 | 0.262697 |
| **NASDAQ** | 0.901436 | 0.575917 | 0.241180 | 0.194512 | 0.262697 | 1.000000 |

| 49 | (data / data.iloc[0] * 100).plot(figsize=(15, 10)) |
|---|---|


## 2) 기초 이론

| 50 | weights = np.random.random(noa)<br>weights /= np.sum(weights) |
|---|---|

0~1 사이의 random 숫자를 noa 개수만큼 생성

random 숫자를 random 숫자 합계로 나누어 투자 비중을 결정

| 51 | weights |
|---|---|

```
array([0.06230658, 0.15912162, 0.17992472, 0.07177926, 0.06973564,
       0.45713218])
```

| 52 | weights.sum() |
|---|---|

1.0

| 53 | np.sum(rets.mean() * weights) * 252 |
|---|---|

0.048017583256158285

포트폴리오 수익률의 기댓값

| 54 | np.dot(weights.T, np.dot(rets.cov() * 252, weights)) # 포트폴리오 분산의 기댓값 |
|---|---|

0.03756710051306895

| 55 | math.sqrt(np.dot(weights.T, np.dot(rets.cov() * 252, weights))) # 포트폴리오 표준편차(변동성)의 기댓값 |
|---|---|

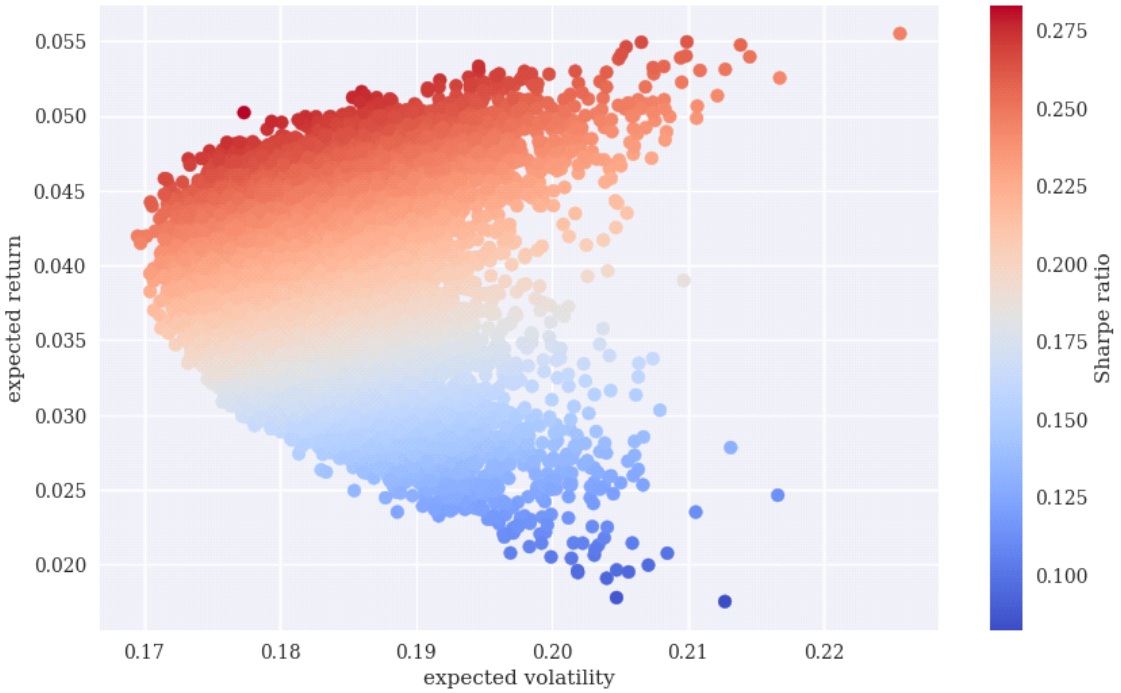
0.19382234265705528

| | |
|---|---|
| 56 | def port_ret(weights):<br>    return np.sum(rets.mean() * weights) * 252 |

| | |
|---|---|
| 57 | def port_vol(weights):<br>    return np.sqrt(np.dot(weights.T, np.dot(rets.cov() * 252, weights))) |

| | |
|---|---|
| 58 | prets = []<br>pvols = []<br>for p in range (25000):<br>    weights = np.random.random(noa)<br>    weights /= np.sum(weights)<br>    prets.append(port_ret(weights))<br>    pvols.append(port_vol(weights))<br>prets = np.array(prets)<br>pvols = np.array(pvols) |

| | |
|---|---|
| 59 | plt.figure(figsize=(10, 6))<br>plt.scatter(pvols, prets, c=prets / pvols, marker='o', cmap='coolwarm')<br>plt.grid(True)<br>plt.xlabel('expected volatility')<br>plt.ylabel('expected return')<br>plt.colorbar(label='Sharpe ratio')<br>pvols = np.array(pvols) |



### 3) 포트폴리오 최적화

| 60 | import scipy.optimize as sco |
|---|---|

| 61 | def min_func_sharpe(weights):<br>    return -port_ret(weights) / port_vol(weights) |
|---|---|

| 62 | cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1}) |
|---|---|

| 63 | bnds = tuple((0, 1) for x in range(noa)) |
|---|---|

| 64 | eweights = np.array(noa * [1. / noa,])<br>eweights |
|---|---|

```
array([0.16666667, 0.16666667, 0.16666667, 0.16666667, 0.16666667,
       0.16666667])
```

| 65 | min_func_sharpe(eweights) |
|---|---|

```
-0.2177903074706918
```

| 66 | %%time<br>opts = sco.minimize(min_func_sharpe, eweights, method='SLSQP', bounds=bnds, constraints=cons) |
|---|---|

```
CPU times: user 203 ms, sys: 0 ns, total: 203 ms
Wall time: 213 ms
```

| 67 | opts |
|---|---|

```
message: Optimization terminated successfully
 success: True
  status: 0
     fun: -0.29075899160421753
       x: [ 7.154e-01  0.000e+00  2.846e-01  1.605e-17  3.426e-17  1.865e-17]
     nit: 9
     jac: [-4.058e-05  2.598e-02  1.020e-04  2.380e-02  1.722e-01  2.042e-02]
    nfev: 63
    njev: 9
```

| 68 | opts['x'].round(3) |
|---|---|

```
array([0.715, 0.   , 0.285, 0.   , 0.   , 0.   ])
```

| 69 | port_ret(opts['x']).round(3) |
|---|---|

```
0.052
```

| 70 | port_vol(opts['x']).round(3) |
|---|---|

```
0.18
```

| 71 | port_ret(opts['x']) / port_vol(opts['x']) |
|---|---|

```
0.29075899160421753
```

| 72 | optv = sco.minimize(port_vol, eweights, method='SLSQP', bounds=bnds, constraints=cons) |
|---|---|

| 73 | optv |
|---|---|

```
message: Optimization terminated successfully
 success: True
  status: 0
     fun: 0.168843759733998
       x: [ 4.986e-01  2.861e-02  1.581e-01  1.980e-01  1.167e-01
            0.000e+00]
     nit: 9
     jac: [ 1.689e-01  1.684e-01  1.689e-01  1.689e-01  1.686e-01
            1.956e-01]
    nfev: 63
    njev: 9
```

| 74 | optv['x'].round(3) |
|---|---|

```
array([0.499, 0.029, 0.158, 0.198, 0.117, 0.   ])
```

| 75 | port_ret(optv['x']).round(3) |
|---|---|

```
0.041
```

| 76 | port_vol(optv['x']).round(3) |
|---|---|

```
0.169
```

| 77 | port_ret(optv['x']) / port_vol(optv['x']) |
|---|---|

```
0.2453563651418332
```

| | S&P500 | DAX | KOSPI | NIKKEI225 | HANSENG | NASDAQ |
|---|---|---|---|---|---|---|
| 최대샤프비율 | 71.8% | 0.0% | 28.2% | 0.0% | 0.0% | 0.0% |
| 최소분산 | 49.9% | 2.9% | 15.8% | 19.8% | 11.6% | 0.0% |

#### 4) 효율적 투자선

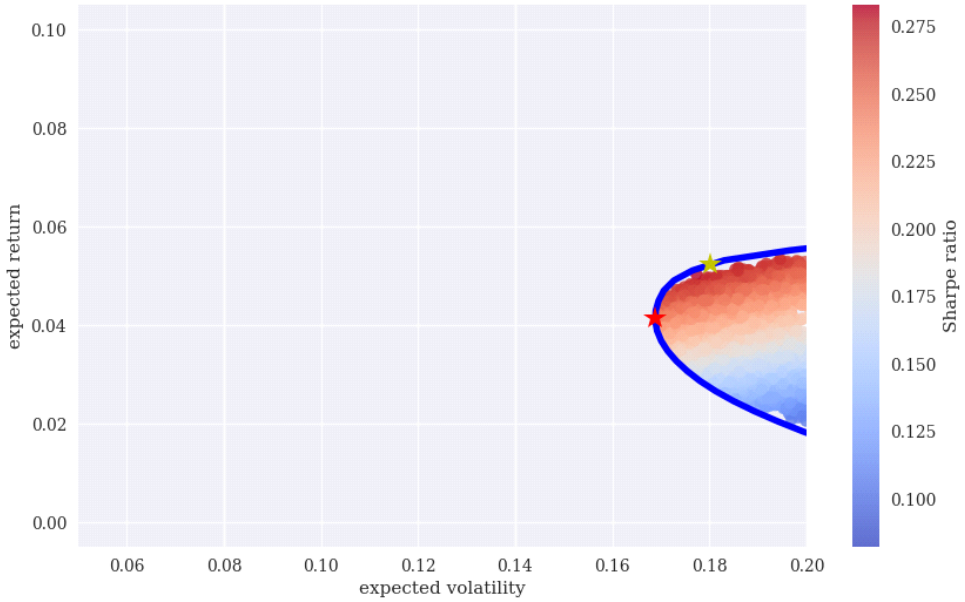
| 78 | cons = ({'type': 'eq', 'fun': lambda x: port_ret(x) - tret},<br>{'type': 'eq', 'fun': lambda x: np.sum(x) - 1}) |
|---|---|

| 79 | bnds = tuple((0, 1) for x in weights) |
|---|---|

| 80 | %%time<br>trets = np.linspace(0.02, 0.1, 50)<br>tvols = []<br>for tret in trets:<br>res = sco.minimize(port_vol, eweights, method='SLSQP', bounds=bnds, constraints=cons)<br>tvols.append(res['fun'])<br>tvols = np.array(tvols) |
|---|---|

```
CPU times: user 20.9 s, sys: 13.5 ms, total: 20.9 s
Wall time: 21.4 s
```

| | |
|---|---|
| 81 | plt.figure(figsize=(10, 6))<br>plt.scatter(pvols, prets, c=prets / pvols, marker='o', alpha=0.8, cmap='coolwarm') # 무작위 포트폴리오<br>plt.plot(tvols, trets, 'b', lw=4.0) # 효율적 투자선<br>plt.plot(port_vol(opts['x']), port_ret(opts['x']), 'y*', markersize=15.0) # 최대 샤프지수를 가진 포트폴리오<br>plt.plot(port_vol(optv['x']), port_ret(optv['x']), 'r*', markersize=15.0) # 최소 분산 포트폴리오<br>plt.grid(True)<br>plt.xlim([0.05, 0.2])<br>plt.xlabel('expected volatility')<br>plt.ylabel('expected return')<br>plt.colorbar(label='Sharpe ratio') |




| | |
|---|---|
| 82 | bnds = len(symbols) * [(0, 1),]<br>bnds |

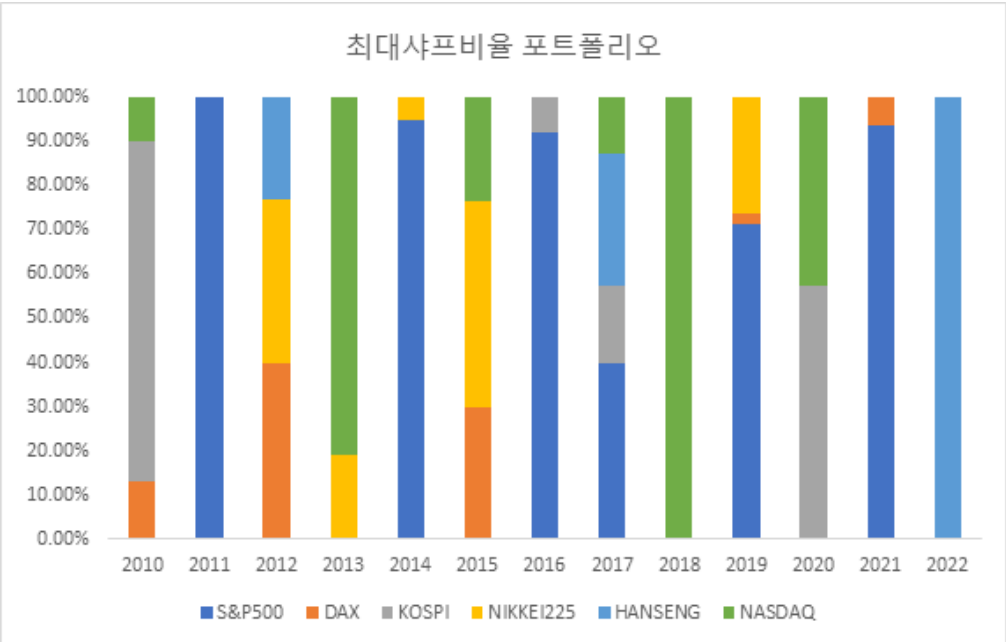
| | |
|---|---|
| 83 | cons = {'type': 'eq', 'fun': lambda weights: weights.sum() - 1} |

| | |
|---|---|
| 84 | def port_return(rets, weights):<br>  return np.dot(rets.mean(), weights) * 252 # annualized<br>def port_volatility(rets, weights):<br>  return np.dot(weights, np.dot(rets.cov() * 252 , weights)) ** 0.5 # annualized<br>def port_sharpe(rets, weights):<br>  return port_return(rets, weights) / port_volatility(rets, weights) |

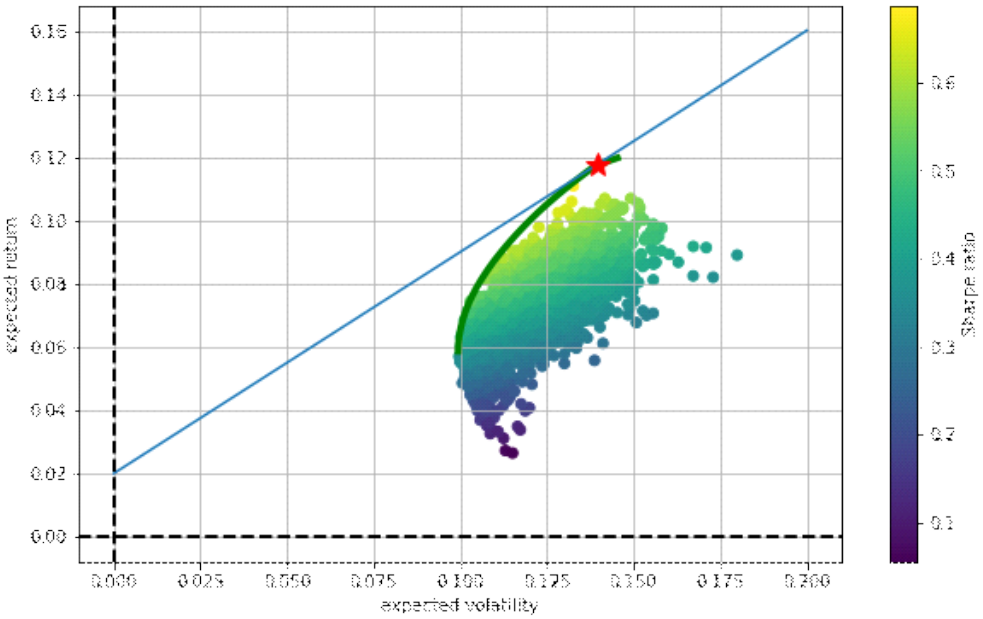
| | |
|---|---|
| 85 | opt_weights ={}<br>for year in range(2010, 2023):<br>rets_ = rets[symbols].loc[f'{year}-01-01':f'{year}-12-31']<br>ow = sco.minimize(lambda weights: -port_sharpe(rets_, weights),<br>len(symbols) * [1 / len(symbols)],<br>bounds=bnds,<br>constraints=cons)['x']<br>opt_weights[year] = ow.round(4) |

| | |
|---|---|
| 86 | opt_weights |

```
{2010: array([0.    , 0.1301, 0.7715, 0.    , 0.    , 0.0984]),
 2011: array([1., 0., 0., 0., 0., 0.]),
 2012: array([0.    , 0.3996, 0.    , 0.3706, 0.2298, 0.    ]),
 2013: array([0.    , 0.    , 0.    , 0.1912, 0.    , 0.8088]),
 2014: array([0.9481, 0.    , 0.    , 0.0519, 0.    , 0.    ]),
 2015: array([0.    , 0.299 , 0.    , 0.4653, 0.    , 0.2357]),
 2016: array([0.9207, 0.    , 0.0793, 0.    , 0.    , 0.    ]),
 2017: array([0.3991, 0.    , 0.1739, 0.    , 0.3008, 0.1263]),
 2018: array([0., 0., 0., 0., 0., 1.]),
 2019: array([0.7118, 0.0254, 0.    , 0.2628, 0.    , 0.    ]),
 2020: array([0.    , 0.    , 0.5746, 0.    , 0.    , 0.4254]),
 2021: array([0.9365, 0.0635, 0.    , 0.    , 0.    , 0.    ]),
 2022: array([0., 0., 0., 0., 1., 0.])}
```



5) 자본시장선




6) Fama-French 5 Factor

| | |
|---|---|
| 87 | import pandas as pd<br>import numpy as np<br><br>!pip install linearmodels<br>!pip install yfinance |

| | |
|---|---|
| 88 | from statsmodels.api import OLS, add_constant<br>import pandas_datareader.data as web<br>from linearmodels.asset_pricing import LinearFactorModel<br><br>import yfinance as yf<br>import plotly.graph_objects as go |

#팩터 불러오기(월별)

| | |
|---|---|
| 89 | ff_factor_data = web.DataReader('F-F_Research_Data_5_Factors_2x3', 'famafrench', start='2012-05', end='2022-07')[0]<br>ff_factor_data.info() |

<class 'pandas.core.frame.DataFrame'>
PeriodIndex: 123 entries, 2012-05 to 2022-07
Freq: M
Data columns (total 6 columns):

```
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   Mkt-RF   123 non-null     float64
 1   SMB      123 non-null     float64
 2   HML      123 non-null     float64
 3   RMW      123 non-null     float64
 4   CMA      123 non-null     float64
 5   RF       123 non-null     float64
dtypes: float64(6)
memory usage: 6.7 KB
```

#섹터별 월별 수익률

| | |
|---|---|
| 90 | ff_portfolio_data = web.DataReader('17_Industry_Portfolios', 'famafrench', start='2012-05', end='2022-07')[0]<br>ff_portfolio_data = ff_portfolio_data.sub(ff_factor_data.RF, axis=0) #리스크 프리미엄 계산: 섹터별 수익률 - 무위험 수익률<br>ff_portfolio_data |

| | |
|---|---|
| 91 | ff_factor_data = ff_factor_data.drop('RF',axis=1)<br>ff_factor_data.info() |

```
<class 'pandas.core.frame.DataFrame'>
PeriodIndex: 123 entries, 2012-05 to 2022-07
Freq: M
Data columns (total 5 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   Mkt-RF   123 non-null     float64
 1   SMB      123 non-null     float64
 2   HML      123 non-null     float64
 3   RMW      123 non-null     float64
 4   CMA      123 non-null     float64
dtypes: float64(5)
memory usage: 5.8 KB
```

| | |
|---|---|
| 92 | dt=ff_factor_data<br>dt.to_csv('ffdata.csv') |

| | |
|---|---|
| 93 | betas = []<br>for industry in ff_portfolio_data:<br>  step1 = OLS(endog=ff_portfolio_data.loc[ff_factor_data.index,industry], #종속변수 |

```
                exog=add_constant(ff_factor_data)).fit() #독립변수
    betas.append(step1.params.drop('const'))
```

```
94  betas = pd.DataFrame(betas, columns=ff_factor_data.columns,
    index=ff_portfolio_data.columns)
    betas
```

```
95  lambdas = []
    for period in ff_portfolio_data.index:
      step2 = OLS(endog=ff_portfolio_data.loc[period, betas.index],
                  exog=betas).fit()
      lambdas.append(step2.params)
```

```
96  lambdas = pd.DataFrame(lambdas, index=ff_portfolio_data.index, columns
    = betas.columns.tolist())
    lambdas
```

| Date | Mkt-RF | SMB | HML | RMW | CMA |
|---|---|---|---|---|---|
| 2012-05 | -5.326169 | -7.377076 | -2.926888 | 1.312326 | -2.098919 |
| 2012-06 | 2.114849 | -9.714655 | 7.281902 | 3.073534 | 3.912548 |
| 2012-07 | 0.839018 | -0.523027 | -1.638412 | -2.046938 | 0.242160 |
| 2012-08 | 2.809701 | 3.365547 | -2.098663 | -0.034542 | -3.804581 |
| 2012-09 | 2.257228 | 0.134870 | 0.554531 | -0.282333 | 2.351835 |
| ... | ... | ... | ... | ... | ... |
| 2022-03 | 3.370023 | -1.636803 | 2.151340 | -2.210892 | 9.875063 |
| 2022-04 | -9.693794 | -2.900080 | 9.742678 | 9.956858 | 5.408164 |
| 2022-05 | -0.139182 | -5.495673 | 9.143430 | -4.506473 | 2.909086 |
| 2022-06 | -7.753723 | -5.789114 | -5.887813 | -2.812574 | -2.143156 |
| 2022-07 | 10.330580 | 7.070721 | -5.827050 | -0.864880 | -4.801239 |

123 rows × 5 columns

```
97  model = LinearFactorModel(portfolios=ff_portfolio_data,
                              factors=ff_factor_data)
    res = model.fit()
    print(res)
```

LinearFactorModel Estimation Summary

| | | | |
|---|---|---|---|
| No. Test Portfolios: | 17 | R-squared: | 0.6890 |
| No. Factors: | 5 | J-statistic: | 12.007 |
| No. Observations: | 123 | P-value | 0.4451 |
| Date: | Thu, Sep 07 2023 | Distribution: | chi2(12) |

Time: 01:06:20
Cov. Estimator: robust

Risk Premia Estimates

| | Parameter | Std. Err. | T-stat | P-value | Lower CI | Upper CI |
|---|---|---|---|---|---|---|
| Mkt-RF | 1.1672 | 0.3982 | 2.9312 | 0.0034 | 0.3867 | 1.9476 |
| SMB | -0.7341 | 0.5138 | -1.4287 | 0.1531 | -1.7412 | 0.2730 |
| HML | -0.3165 | 0.5060 | -0.6255 | 0.5317 | -1.3082 | 0.6752 |
| RMW | 0.2267 | 0.5670 | 0.3999 | 0.6892 | -0.8846 | 1.3381 |
| CMA | -0.4249 | 0.3545 | -1.1988 | 0.2306 | -1.1197 | 0.2698 |

Covariance estimator:
HeteroskedasticCovariance
See full_summary for complete results

98
```
window = 24
lambdas_rolling = lambdas.rolling(window).mean().dropna()
lambdas_rolling
```

| Date | Mkt-RF | SMB | HML | RMW | CMA |
|---|---|---|---|---|---|
| 2014-04 | 1.565523 | -1.044468 | -0.076614 | 0.389193 | -0.662212 |
| 2014-05 | 1.868964 | -0.960029 | 0.089485 | 0.399823 | -0.683188 |
| 2014-06 | 1.878047 | -0.594882 | -0.161414 | 0.281844 | -0.774388 |
| 2014-07 | 1.717610 | -0.474730 | -0.067450 | 0.318841 | -0.762446 |
| 2014-08 | 1.762173 | -0.785513 | 0.078822 | 0.613365 | -0.638731 |
| … | … | … | … | … | … |
| 2022-03 | 2.867210 | 1.897996 | 0.263003 | -0.647973 | 1.339470 |
| 2022-04 | 1.835844 | 1.656625 | 0.737616 | 0.158299 | 1.346114 |
| 2022-05 | 1.612230 | 1.405611 | 1.159556 | -0.349594 | 1.696912 |
| 2022-06 | 1.175199 | 0.742647 | 1.235249 | -0.238004 | 1.617751 |
| 2022-07 | 1.322357 | 1.069125 | 1.457937 | -0.212396 | 1.385636 |

100 rows × 5 columns

99
```
colors = ['#ce4912','#fb8500','#042c71','#0b6ab0','#ffb703']
fig = go.Figure()

for factor,i in zip(['Mkt-RF','SMB','HML','RMW','CMA'],range(0,5)):
  fig.add_trace(go.Scatter(x=lambdas_rolling.index.astype('str'),
```

```
y=lambdas_rolling[factor],mode='lines',line=dict(width=1,color=colors[i]),name=factor))

fig.update_layout(
        title = dict(text = 'Fama-Macbeth Linear Regression',font = dict(size=20,color='black')),
        legend=dict(orientation="h", yanchor="bottom", y=0.93, xanchor="right", x=0.4),
        autosize = True,
        showlegend = True,
        font = dict(size = 8, color = 'black'),
        plot_bgcolor = 'white',paper_bgcolor='white',
        width=1000,height=600,
        xaxis = dict(title = 'Date', showline = True, showgrid = False, showticklabels = True, zeroline=False,mirror = True,
                     linecolor = 'black', linewidth = 0.8,ticks = 'outside'),
        yaxis = dict(title = 'Factor Risk Premium', showline = True, showgrid = False, showticklabels = True,zeroline=False, mirror = True,
                     linecolor = 'black', linewidth = 0.8,ticks = 'outside'),
        annotations=[
        dict(xref='paper',yref='paper',x=1.08,y=1.20,align='right',

text='Information',showarrow=False,font=dict(size=8,color='red')),
        dict(xref='paper',yref='paper',x=1.08,y=1.18,align='right',
            text='Linkedin:~/chacehkk',showarrow=False,font=dict(size=8,color='grey')),
        dict(xref='paper',yref='paper',x=1.08,y=1.16,align='right',
            text='Github:~/hyksun2015',showarrow=False,font=dict(size=8,color='grey')),
        dict(xref='paper',yref='paper',x=1.08,y=-0.08,align='right',
            text='Source:famafrench',showarrow=False,font=dict(size=8,color='black')),
        dict(xref='paper',yref='paper',x=1.08,y=-0.1,align='right',
            text='Data period: 2012.05.01 - 2022.07.31',showarrow=False,font=dict(size=8,color='black'))])
```

# Fama-Macbeth Linear Regression

Linkedin: ~/chaehkk
Github: ~/hyksun2015

# 4. Derivatives Valuation

Derivatives are a huge, complex issue. – Judd Gregg

## 4-1. Black Scholes valuation methods

### 1) 블랙-숄즈 모형

배당이 없는 유럽형 콜옵션의 가치를 블랙-숄즈 옵션가격결정모형은 closed formula로 다음과 같이 표현함

$C = S \times N(d_1) - Xe^{-rT} \times N(d_2)$

여기서, $d_1 = \frac{\ln(\frac{S}{X}) + (r + \frac{1}{2}\sigma^2) \times T}{\sigma\sqrt{T}}$ 이고, $d_2 = d_1 - \sigma\sqrt{T}$이다.

S = 기초자산의 현재가, X = 행사가격, r= 무위험 이자율
T = 옵션의 잔존기간, σ= 옵션기초자산 수익률의 표준편차
N(d)= 누적 정규분포 확률함수

풋옵션의 가격

$P = Xe^{-rT} \times N(-d_2) - S \times N(-d_1)$

여기서, $N(-d_1) = 1 - N(d_1)$이고, $N(-d_2) = 1 - N(d_2)$이다.

| | |
|---|---|
| 1 | import numpy as np<br>import scipy.stats as ss<br>from scipy.integrate import quad<br>from functools import partial<br>import matplotlib.pyplot as plt<br><br>%matplotlib inline |

| | |
|---|---|
| 2 | def BlackScholes(payoff="call", S0=100.0, K=100.0, T=1.0, r=0.1, sigma=0.2):<br>  d1 = (np.log(S0 / K) + (r + sigma**2 / 2) * T) / (sigma * np.sqrt(T))<br>  d2 = (np.log(S0 / K) + (r - sigma**2 / 2) * T) / (sigma * np.sqrt(T))<br>  if payoff == "call":<br>  return S0*ss.norm.cdf(d1) - K*np.exp(-r * T)*ss.norm.cdf(d2)<br>  elif payoff == "put":<br>  return K*np.exp(-r*T)*ss.norm.cdf(-d2) - S0*ss.norm.cdf(-d1)<br>  else:<br>  raise ValueError("invalid type. Set 'call' or 'put'") |

| | |
|---|---|
| 3 | S0 = 100.0  # spot stock price<br>K = 100.0  # strike<br>T = 1.0  # maturity<br>r = 0.035  # risk free rate<br>sig = 0.2  # volatility |

S0 : 기초자산 현재 가격
K : 행사가격
T : 연 단위 만기, 1은 1년
r : 무위험이자율
sig : 변동성

| | |
|---|---|
| 4 | call = BlackScholes("call", S0, K, T, r, sig)<br>put = BlackScholes("put", S0, K, T, r, sig)<br>print("Call price: ", call)<br>print("Put price: ", put) |

Call price:  9.667467371387701
Put price:  6.228008997144336

**2) Put-Call parity**

동일한 기초증권과 만기 및 행사가격을 가지고 있는 콜옵션과 풋옵션의 가격은 균형하에서 일정한 관계를 갖는데 이를 풋-콜 패리티라고 한다.
예를 들어 하나의 주식을 매입하고(매입가격 S) 이 주식에 대한 풋옵션 하나를 매입하고(매입가격 P), 동시에 동일한 행사가격을 갖는 콜옵션 하나를 발행한 경우(발행가격 C), 만기일에서의 포트폴리오를 구성할 수 있으며 이러한 포트폴리오의 수익률은 시장이 균형상태에 있을 때 무위험 수익률과 같게 된다.
따라서 이러한 무위험 포트폴리오의 원래 투자액(S+P-C)은 매매가치인 X를 무위험수익률로 할인한 X의 현재가치와 같아야 하므로, 다음과 같은 관계식이 성립되는데 이를 풋-콜 패리티라고 한다.

$S_0 + Put - Call = Ke^{-rt}$
$S_0 + Put = Call + Ke^{-rt}$
$Call - Put = S_0 - Ke^{-rt}$

| | |
|---|---|
| 5 | print(call)<br>print(put + S0 - K * np.exp(-r * T)) |

9.667467371387701
9.667467371387701

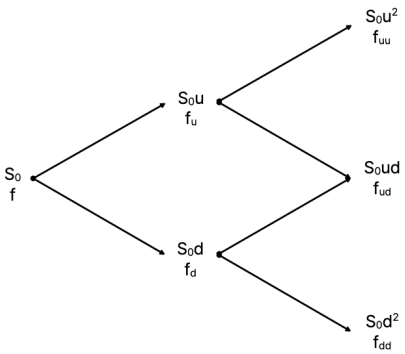
### 3) Monte Carlo method

| | |
|---|---|
| 6 | np.random.seed(seed=44)  # seed for random number generation<br>N = 10000000  # Number of random variables<br><br>W = ss.norm.rvs((r - 0.5 * sig**2) * T, np.sqrt(T) * sig, N)<br>S_T = S0 * np.exp(W)<br><br>call = np.mean(np.exp(-r * T) * np.maximum(S_T - K, 0))<br>put = np.mean(np.exp(-r * T) * np.maximum(K - S_T, 0))<br>call_err = ss.sem(np.exp(-r * T) * np.maximum(S_T - K, 0))  # standard error<br>put_err = ss.sem(np.exp(-r * T) * np.maximum(K - S_T, 0))  # standard error |

| | |
|---|---|
| 7 | print("Call price: {}, with error: {}".format(call, call_err))<br>print("Put price: {}, with error: {}".format(put, put_err)) |

Call price: 9.662697937910483, with error: 0.004510627984933323
Put price: 6.231549154069496, with error: 0.002901821083005663

### 4) 이항분포 모형



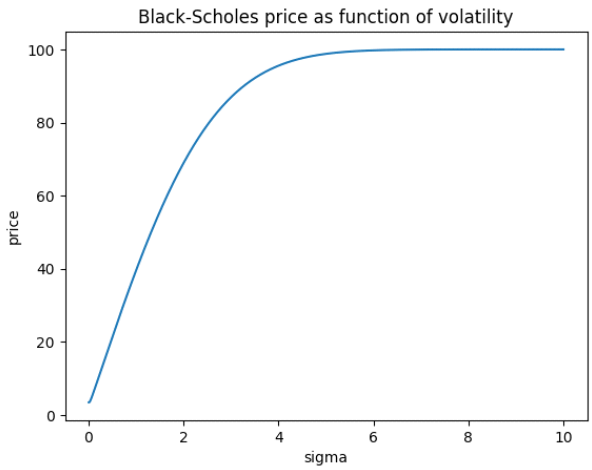

| | |
|---|---|
| 8 | N = 15000  # number of periods or number of time steps<br>payoff = "call"  # payoff<br><br>dT = float(T) / N  # Delta t<br>u = np.exp(sig * np.sqrt(dT))  # up factor<br>d = 1.0 / u  # down factor<br>V = np.zeros(N + 1)  # initialize the price vector<br><br># price S_T at time T<br>S_T = np.array([(S0 * u**j * d ** (N - j)) for j in range(N + 1)]) |

```
a = np.exp(r * dT)  # risk free compounded return
p = (a - d) / (u - d)  # risk neutral up probability
q = 1.0 - p  # risk neutral down probability

if payoff == "call":
    V[:] = np.maximum(S_T - K, 0.0)
else:
    V[:] = np.maximum(K - S_T, 0.0)

for i in range(N - 1, -1, -1):
    # the price vector is overwritten at each step
    V[:-1] = np.exp(-r * dT) * (p * V[1:] + q * V[:-1])

print("BS Tree Price: ", V[0])
```

BS Tree Price:  9.667334977885675

**5) 모형의 한계**

```
BS_sigma = partial(BlackScholes, "call", S0, K, T, r)  # binding the function
sigmas = np.linspace(0.01, 10, 1000)

plt.plot(sigmas, BS_sigma(sigmas))
plt.xlabel("sigma")
plt.ylabel("price")
plt.title("Black-Scholes price as function of volatility")
plt.show()
```




BS 공식은 변동성이 증가하면 가치가 증가하는 모델
그러나 변동성이 높을수록 그래프는 거의 평평해짐
0~400% 범위의 변동성에 대해 신뢰할 수 있는 모델임

## 4-2. American Option

### 1) 블랙-숄즈 편미분 방정식(The Black-Scholes PDE)

블랙-숄즈 편미분 방정식(partial differential equation, PDE)

$$\frac{\partial V(t,s)}{\partial t} + r\,s\frac{\partial V(t,s)}{\partial s} + \frac{1}{2}\sigma^2 s^2 \frac{\partial^2 V(t,s)}{\partial s^2} - rV(t,s) = 0.$$

```
10
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
%matplotlib inline

from scipy import sparse
from scipy.sparse.linalg import splu
from scipy.sparse.linalg import spsolve

from IPython.display import display
import sympy
sympy.init_printing()

def display_matrix(m):
    display(sympy.Matrix(m))
```

- Taylor 정리: 테일러 정리란 어떤 함수 $f(x)$의 모든 도함수가 $x=a$ 부근에서 연속적인 경우 이 함수가 다음과 같이 무한급수로 표현될 수 있다는 정리다.

  $f(x) = f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2 + \cdots + \frac{1}{n!}f^{(n)}(a)(x-a)^n + \cdots$

  이 식을 $x=a$에서 함수 $f(x)$의 테일러 급수라고 한다.

- 가장 간단한 형태의 유한차분법은 순방향 차분과 역방향 차분으로, 각각 함수의 전방과 후방의 차이를 이용하여 미분을 근사화한다. 이 방법은 다음과 같이 나타낼 수 있다.

  ① 순방향 차분(Forward Difference): explicit finite difference schemes

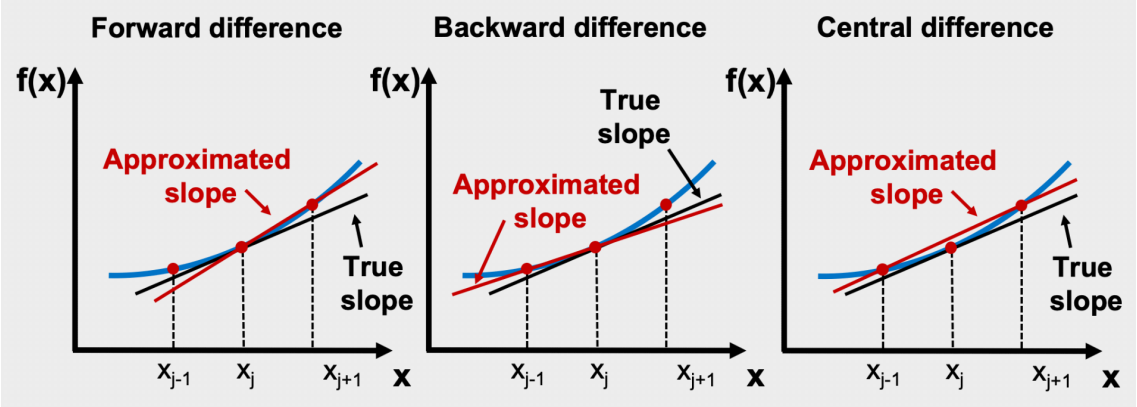
  $f'(x) \approx \{f(x+\Delta x) - f(x)\}/\Delta x$

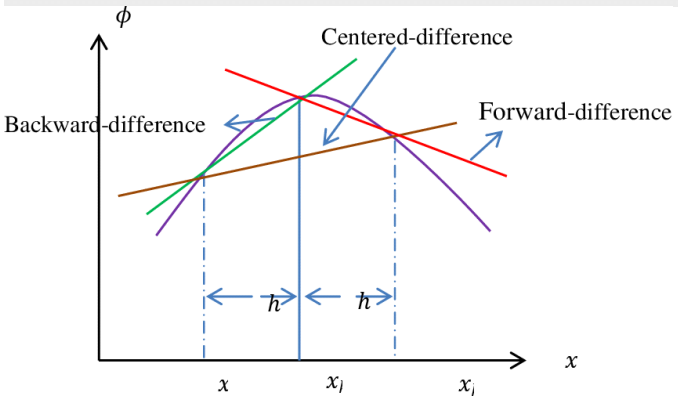
  ② 역방향 차분(Backward Difference): implicit finite difference schemes

  $f'(x) \approx \{f(x) - f(x-\Delta x)\}/\Delta x$

  ③ 중앙 차분(Central Difference):

  $f'(x) \approx \{f(x+\Delta x) - f(x-\Delta x)\}/2\Delta x$




| | |
|---|---|
| 11 | r = 0.035<br>sig = 0.2<br>S0 = 100<br>X0 = np.log(S0)<br>K = 100<br>Texpir = 1 |

| | |
|---|---|
| 12 | Nspace = 3000 # M space steps<br>Ntime = 2000 # N time steps<br><br>S_max = 3 * float(K)<br>S_min = float(K) / 3<br><br>x_max = np.log(S_max) # A2<br>x_min = np.log(S_min) # A1 |

| | |
|---|---|
| 13 | x, dx = np.linspace(x_min, x_max, Nspace, retstep=True) # space discretization<br>T, dt = np.linspace(0, Texpir, Ntime, retstep=True) # time discretization<br>Payoff = np.maximum(np.exp(x) - K, 0) # Call payoff |

| | |
|---|---|
| 14 | V = np.zeros((Nspace, Ntime))  # grid initialization<br>offset = np.zeros(Nspace - 2)  # vector to be used for the boundary terms<br><br>V[:, -1] = Payoff  # terminal conditions<br>V[-1, :] = np.exp(x_max) - K * np.exp(-r * T[::-1])  # boundary condition<br>V[0, :] = 0  # boundary condition |

| | |
|---|---|
| 15 | # construction of the tri-diagonal matrix D<br>sig2 = sig * sig<br>dxx = dx * dx<br><br>a = (dt / 2) * ((r - 0.5 * sig2) / dx - sig2 / dxx)<br>b = 1 + dt * (sig2 / dxx + r)<br>c = -(dt / 2) * ((r - 0.5 * sig2) / dx + sig2 / dxx)<br><br>D = sparse.diags([a, b, c], [-1, 0, 1], shape=(Nspace - 2, Nspace - 2)).tocsc() |

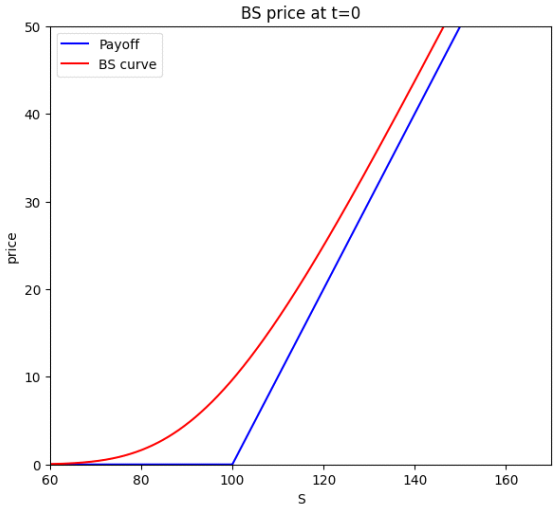
| | |
|---|---|
| 16 | # Backward iteration<br>for i in range(Ntime - 2, -1, -1):<br>    offset[0] = a * V[0, i]<br>    offset[-1] = c * V[-1, i]<br>    V[1:-1, i] = spsolve(D, (V[1:-1, i + 1] - offset)) |

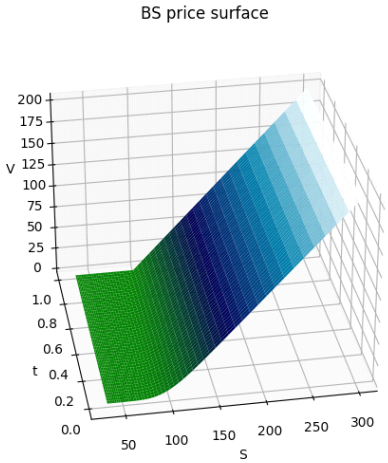
| | |
|---|---|
| 17 | # finds the option at S0<br>oPrice = np.interp(X0, x, V[:, 0])<br>print(oPrice) |

9.666964975862907

| | |
|---|---|
| 18 | S = np.exp(x)<br>fig = plt.figure(figsize=(15, 6))<br>ax1 = fig.add_subplot(121)<br>ax2 = fig.add_subplot(122, projection="3d")<br><br>ax1.plot(S, Payoff, color="blue", label="Payoff")<br>ax1.plot(S, V[:, 0], color="red", label="BS curve")<br>ax1.set_xlim(60, 170)<br>ax1.set_ylim(0, 50)<br>ax1.set_xlabel("S")<br>ax1.set_ylabel("price") |

| | |
|---|---|
| | ax1.legend(loc="upper left")<br>ax1.set_title("BS price at t=0")<br><br>X, Y = np.meshgrid(T, S)<br>ax2.plot_surface(Y, X, V, cmap=cm.ocean)<br>ax2.set_title("BS price surface")<br>ax2.set_xlabel("S")<br>ax2.set_ylabel("t")<br>ax2.set_zlabel("V")<br>ax2.view_init(30, -100) # this function rotates the 3d plot<br>plt.show() |




| | |
|---|---|
| 19 | # BS_pricer.py file upload<br>from google.colab import files<br>src = list(files.upload().values())[0] |

| | |
|---|---|
| 20 | open('BS_pricer.py','wb').write(src)<br>import BS_pricer |

| | |
|---|---|
| 21 | class Option_param:<br><br>def __init__(self, S0=15, K=15, T=1, v0=0.04, payoff="call", exercise="European"):<br>self.S0 = S0<br>self.v0 = v0<br>self.K = K<br>self.T = T |

```
        if exercise == "European" or exercise == "American":
            self.exercise = exercise
        else:
            raise ValueError("invalid type. Set 'European' or 'American'")

        if payoff == "call" or payoff == "put":
            self.payoff = payoff
        else:
            raise ValueError("invalid type. Set 'call' or 'put'")
```

22

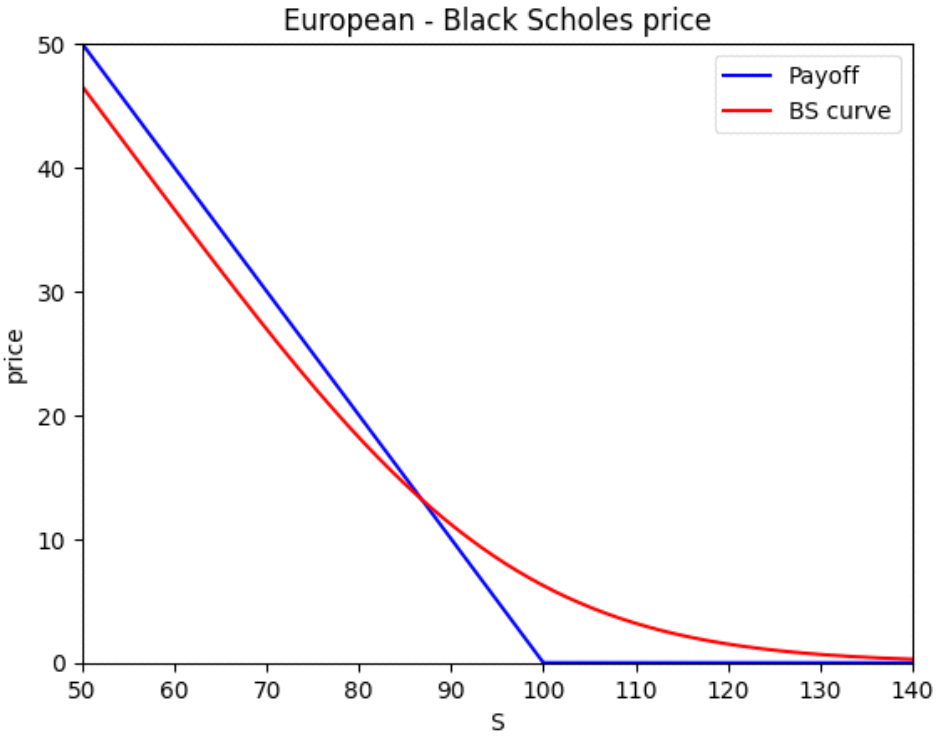
```
class Diffusion_process:
    """
    Class for the diffusion process:
    r = risk free constant rate
    sig = constant diffusion coefficient
    mu = constant drift
    """

    def __init__(self, r=0.035, sig=0.2, mu=0.1):
        self.r = r
        self.mu = mu
        if sig <= 0:
            raise ValueError("sig must be positive")
        else:
            self.sig = sig

    def exp_RV(self, S0, T, N):
        W = ss.norm.rvs((self.r - 0.5 * self.sig**2) * T, np.sqrt(T) *
self.sig, N)
        S_T = S0 * np.exp(W)
        return S_T.reshape((N, 1))
```

23

```
from BS_pricer import BS_pricer
# Creates the object with the parameters of the option
opt_param = Option_param(S0=100, K=100, T=1, exercise="European",
payoff="call")
# Creates the object with the parameters of the process
diff_param = Diffusion_process(r=0.035, sig=0.2)
# Creates the object of the pricer
BS = BS_pricer(opt_param, diff_param)
```

| | |
|---|---|
| 24 | put_param = Option_param(S0=100, K=100, T=1, exercise="European", payoff="put")<br># Creates the object of the put pricer<br>put = BS_pricer(put_param, diff_param)<br>print("price: ", put.PDE_price((5000, 4000)))<br>put.plot(axis=[50, 140, 0, 50]) |

price: 6.227773968725298




**2) 미국형 옵션(American options) pricing**

유럽식 옵션 : 만기 시에만 행사 가능.
미국식 옵션 : 만기 전 아무 때나 행사 가능.

유럽식 옵션 가격 ≤ 미국식 옵션 가격
유럽식 call 옵션 가격 = 미국식 call 옵션 가격
유럽식 put 옵션 가격 ≤ 미국식 Put 옵션 가격

| | |
|---|---|
| 25 | # Creates the object with the parameters of the option<br>opt_param = Option_param(S0=100, K=100, T=1, exercise="American", payoff="put")<br># Creates the object with the parameters of the process<br>diff_param = Diffusion_process(r=0.035, sig=0.2) |

| | |
|---|---|
| | # Creates the pricer object<br>BS = BS_pricer(opt_param, diff_param) |

**Longstaff - Schwartz Method**(LS Method)

This is a Monte Carlo algorithm proposed by Longstaff and Schwartz in the paper

| | |
|---|---|
| 26 | BS.LSM(N=10000, paths=10000, order=2) # Longstaff Schwartz Method |

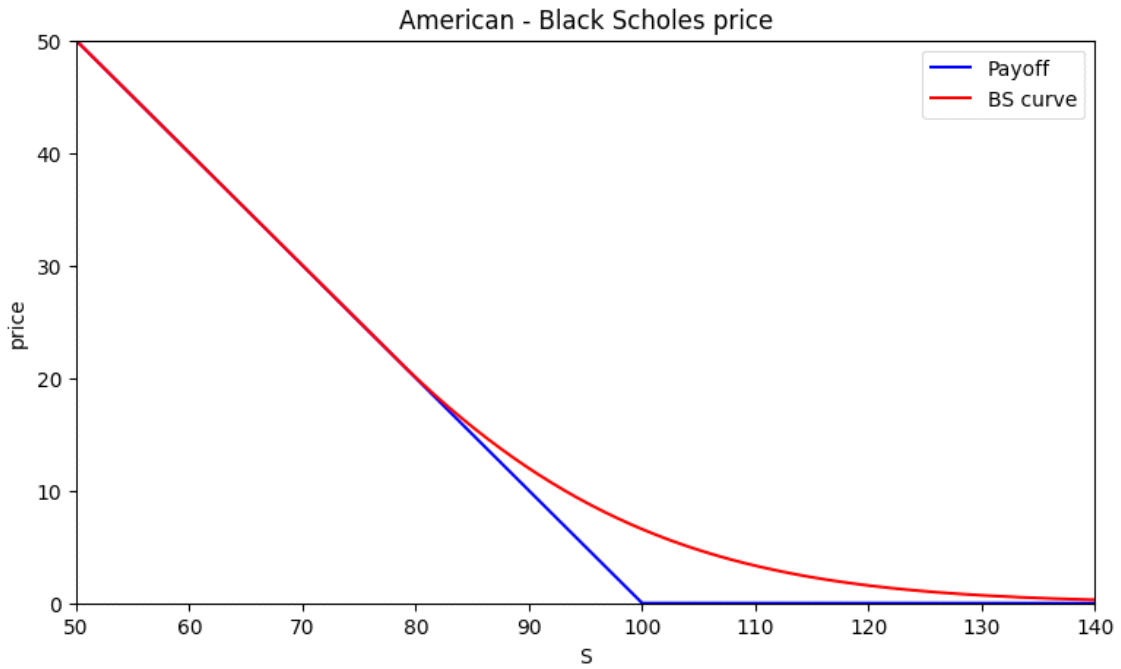
6.524544830058

**편미분 방정식**으로 American Put옵션 가격 계산

| | |
|---|---|
| 27 | N_space = 8000<br>N_time = 5000<br>BS.PDE_price((N_space, N_time)) |

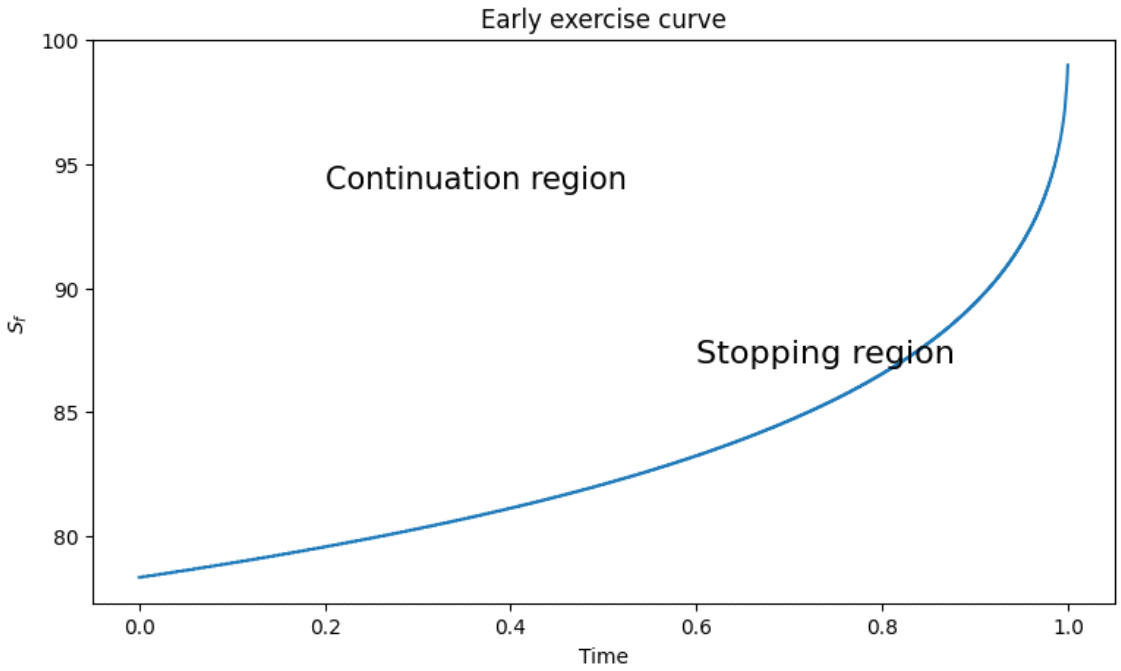
6.5697224344193

| | |
|---|---|
| 28 | fig = plt.figure(figsize=(9, 5))<br>BS.plot([50, 140, 0, 50]) |




| | |
|---|---|
| 29 | payoff = BS.payoff_f(BS.S_vec).reshape(len(BS.S_vec), 1) # Transform the payoff in a column vector<br>mesh = (BS.mesh - payoff)[1:-1, :-1] # I remove the boundary terms<br>optimal_indeces = np.argmax(np.abs(mesh) > 1e-10, axis=0) # I introduce the error 1e-10<br>T_vec = np.linspace(0, BS.T, N_time) # Time vector |

| | |
|---|---|
| 30 | fig = plt.figure(figsize=(9, 5))<br>plt.plot(T_vec[:-1], BS.S_vec[optimal_indeces])<br>plt.text(0.2, 94, "Continuation region", fontsize=15)<br>plt.text(0.6, 87, "Stopping region", fontsize=16)<br>plt.xlabel("Time")<br>plt.ylabel("$S_f$")<br>plt.title("Early exercise curve")<br>plt.show() |




**이항모형**

| | |
|---|---|
| 31 | S0 = 100.0 # spot stock price<br>K = 100.0 # strike<br>T = 1.0 # maturity<br>r = 0.035 # risk free rate<br>sig = 0.2 # diffusion coefficient or volatility |

| | |
|---|---|
| 32 | N = 25000 # number of periods or number of time steps<br>payoff = "put" # payoff<br><br>dT = float(T) / N # Delta t<br>u = np.exp(sig * np.sqrt(dT)) # up factor<br>d = 1.0 / u # down factor |

```
V = np.zeros(N + 1)  # initialize the price vector
S_T = np.array([(S0 * u**j * d ** (N - j)) for j in range(N + 1)])  # price S_T at time T

a = np.exp(r * dT)  # risk free compound return
p = (a - d) / (u - d)  # risk neutral up probability
q = 1.0 - p  # risk neutral down probability

if payoff == "call":
    V[:] = np.maximum(S_T - K, 0.0)
elif payoff == "put":
    V[:] = np.maximum(K - S_T, 0.0)

for i in range(N - 1, -1, -1):
    V[:-1] = np.exp(-r * dT) * (p * V[1:] + q * V[:-1])  # the price vector is overwritten at each step
    S_T = S_T * u  # it is a tricky way to obtain the price at the previous time step
    if payoff == "call":
        V = np.maximum(V, S_T - K)
    elif payoff == "put":
        V = np.maximum(V, K - S_T)

print("American BS Tree Price: ", V[0])
```

American BS Tree Price:  6.570166616871246

## 4-3. Exotic Option

| | |
|---|---|
| 33 | import numpy as np<br>import scipy.stats as ss<br>import matplotlib.pyplot as plt<br>import matplotlib.gridspec as gridspec<br><br>%matplotlib inline<br><br>from scipy import sparse<br>from scipy.sparse.linalg import splu<br>from scipy.sparse.linalg import spsolve |

| | |
|---|---|
| 34 | # Option variables<br>S0 = 100.0  # spot stock price<br>K = 100.0  # strike<br>T = 1.0  # maturity<br>r = 0.035  # risk free rate<br>sig = 0.2  # diffusion coefficient or volatility<br>X0 = np.log(S0)  # logprice<br>B = 120  # Barrier 1<br>BB = 90  # Barrier 2 |

### 1) Binary/digital options

cash or nothing CALL binary option

$$V(t,s) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[1_{\{S_T>K\}} \,\middle|\, S_t = s\right]$$

Q is the risk neutral measure

### Closed formula

| | |
|---|---|
| 35 | d2 = (np.log(S0 / K) + (r - sig**2 / 2) * T) / (sig * np.sqrt(T))<br>N2 = ss.norm.cdf(d2)<br>closed_digital = np.exp(-r * T) * N2<br>print("The price of the ATM digital call option by closed formula is: ", closed_digital) |

The price of the ATM digital call option by closed formula is:  0.5116672071325179

### Monte Carlo

| | |
|---|---|
| 36 | d2 = (np.log(S0 / K) + (r - sig**2 / 2) * T) / (sig * np.sqrt(T))<br>N2 = ss.norm.cdf(d2) |

```
closed_digital = np.exp(-r * T) * N2
print("The price of the ATM digital call option by closed formula is: ", closed_digital)
```

The price of the ATM digital call option by Monte Carlo is:  0.51183025534961
with standard error:  0.00010776267166593986

**PDE method**

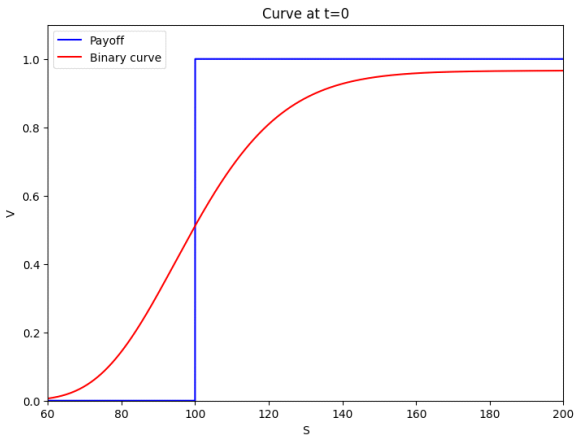
```
Nspace = 6000  # M space steps
Ntime = 6000  # N time steps
S_max = 3 * float(K)
S_min = float(K) / 3
x_max = np.log(S_max)  # A2
x_min = np.log(S_min)  # A1

x, dx = np.linspace(x_min, x_max, Nspace, retstep=True)  # space discretization
T_array, dt = np.linspace(0, T, Ntime, retstep=True)  # time discretization
Payoff = np.where(np.exp(x) > K, 1, 0)  # Binary payoff

V = np.zeros((Nspace, Ntime))  # grid initialization
offset = np.zeros(Nspace - 2)  # vector to be used for the boundary terms

V[:, -1] = Payoff  # terminal conditions
V[-1, :] = 1  # boundary condition
V[0, :] = 0  # boundary condition

# construction of the tri-diagonal matrix D
sig2 = sig * sig
dxx = dx * dx
a = (dt / 2) * ((r - 0.5 * sig2) / dx - sig2 / dxx)
b = 1 + dt * (sig2 / dxx + r)
c = -(dt / 2) * ((r - 0.5 * sig2) / dx + sig2 / dxx)
D = sparse.diags([a, b, c], [-1, 0, 1], shape=(Nspace - 2, Nspace - 2)).tocsc()
DD = splu(D)

# Backward iteration
for i in range(Ntime - 2, -1, -1):
    offset[0] = a * V[0, i]
```

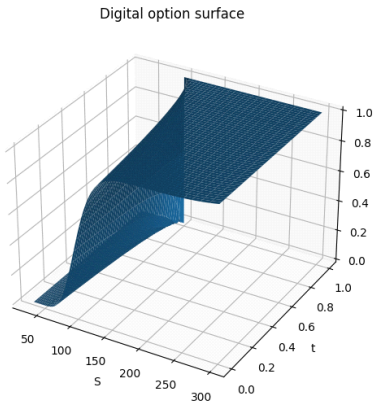
```
        offset[-1] = c * V[-1, i]
        V[1:-1, i] = DD.solve(V[1:-1, i + 1] - offset)

    # finds the option at S0
    oPrice = np.interp(X0, x, V[:, 0])
    print("The price of the ATM digital call option by PDE is: ", oPrice)
```

he price of the ATM digital call option by PDE is:  0.5116665597521938

```
S = np.exp(x)
fig = plt.figure(figsize=(18, 6))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122, projection="3d")
ax1.plot(S, Payoff, color="blue", label="Payoff")
ax1.plot(S, V[:, 0], color="red", label="Binary curve")
ax1.set_xlim(60, 200)
ax1.set_ylim(0, 1.1)
ax1.set_xlabel("S")
ax1.set_ylabel("V")
ax1.legend(loc="upper left")
ax1.set_title("Curve at t=0")

X, Y = np.meshgrid(T_array, S)
ax2.plot_surface(Y, X, V)
ax2.set_title("Digital option surface")
ax2.set_xlabel("S")
ax2.set_ylabel("t")
ax2.set_zlabel("V")
plt.show()
```

38



2) Barrier options

Up and Out CALL European option

$$V(t,s) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[(S_T - K)^+ 1_{\{M_T<B\}} \,\middle|\, S_t = s\right]$$

$$M_T = \max_{0\le t\le T} S_t$$

Down and In CALL European option

$$V(t,s) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[(S_T - K)^+ 1_{\{\tilde{M}_T\le B\}} \,\middle|\, S_t = s\right]$$

$$\tilde{M}_T = \min_{0\le t\le T} S_t$$

Closed formula

| | |
|---|---|
| 39 | d1 = lambda t, s: 1 / (sig * np.sqrt(t)) * (np.log(s) + (r + sig**2 / 2) * t)<br>d2 = lambda t, s: 1 / (sig * np.sqrt(t)) * (np.log(s) + (r - sig**2 / 2) * t) |

| | |
|---|---|
| 40 | closed_barrier_u = (<br>S0 * (ss.norm.cdf(d1(T, S0 / K)) - ss.norm.cdf(d1(T, S0 / B)))<br>- np.exp(-r * T) * K * (ss.norm.cdf(d2(T, S0 / K)) - ss.norm.cdf(d2(T, S0 / B)))<br>- B * (S0 / B) ** (-2 * r / sig**2) * (ss.norm.cdf(d1(T, B**2 / (S0 * K))) - ss.norm.cdf(d1(T, B / S0)))<br>+ np.exp(-r * T)<br>* K<br>* (S0 / B) ** (-2 * r / sig**2 + 1)<br>* (ss.norm.cdf(d2(T, B**2 / (S0 * K))) - ss.norm.cdf(d2(T, B / S0)))<br>) |

| | |
|---|---|
| 41 | print("The price of the Up and Out call option by closed formula is: ", closed_barrier_u) |

The price of the Up and Out call option by closed formula is:  1.1615536798122559

| | |
|---|---|
| 42 | closed_barrier_d = S0 * (BB / S0) ** (1 + 2 * r / sig**2) * (ss.norm.cdf(d1(T, BB**2 / (S0 * K)))) - np.exp(<br>-r * T<br>) * K * (BB / S0) ** (-1 + 2 * r / sig**2) * (ss.norm.cdf(d2(T, BB**2 / (S0 * K)))) |

| | |
|---|---|
| 43 | print("The price of the Down and In call option by closed formula is: ", closed_barrier_d) |

The price of the Down and In call option by closed formula is:  1.701611346364901

**Monte Carlo**

$$V_d(B) = V(Be^{\pm\beta_1\sigma\sqrt{\Delta t}})$$

$$\beta_1 = -\frac{\zeta(1/2)}{\sqrt{2\pi}} \approx 0.5826$$

| | |
|---|---|
| 44 | # correction<br>beta1 = 0.5826<br>B = B * np.exp( - beta1 * np.sqrt(dt) * sig)<br>BB = BB * np.exp(beta1 * np.sqrt(dt) * sig) |

| | |
|---|---|
| 45 | %%time<br>np.random.seed(seed=42)<br>N = 10000<br>paths = 20000<br>dt = T / (N - 1)  # time interval<br><br># path generation<br>X_0 = np.zeros((paths, 1))<br>increments = ss.norm.rvs(loc=(r - sig**2 / 2) * dt, scale=np.sqrt(dt) * sig, size=(paths, N - 1))<br>X = np.concatenate((X_0, increments), axis=1).cumsum(1)<br>S = S0 * np.exp(X)<br><br>M = np.amax(S, axis=1)  # maximum of each path<br>MM = np.amin(S, axis=1)  # minimum of each path<br><br># Up and Out  --  Discounted expected payoff<br>MC_barrier_u = np.exp(-r * T) * (np.maximum(S[:, -1] - K, 0) @ (M < B)) / paths<br>barrier_std_err_u = np.exp(-r * T) * ss.sem((np.maximum(S[:, -1] - K, 0) * (M < B)))<br><br>print("The price of the Up and Out call option by Monte Carlo is: ", MC_barrier_u)<br>print("with standard error: ", barrier_std_err_u)<br>print() |

```
# Down and In
MC_barrier_d = np.exp(-r * T) * (np.maximum(S[:, -1] - K, 0) @ (MM <= BB)) / paths
barrier_std_err_d = np.exp(-r * T) * ss.sem((np.maximum(S[:, -1] - K, 0) * (MM <= BB)))

print("The price of the Down and In call option by Monte Carlo is: ", MC_barrier_d)
print("with standard error: ", barrier_std_err_d)
```

The price of the Up and Out call option by Monte Carlo is:  1.1354341254328006
with standard error:  0.022079647034462926

The price of the Down and In call option by Monte Carlo is:  1.7007445281013338
with standard error:  0.0399796378332013
CPU times: user 10.3 s, sys: 1.57 s, total: 11.9 s
Wall time: 11.9 s

**PDE method - Up and Out**

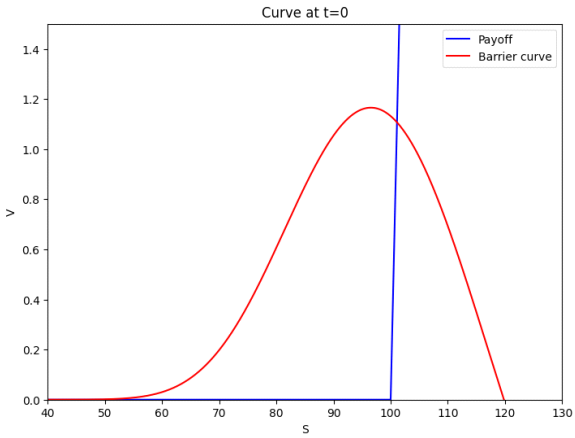
```
# PDE
Nspace = 14000  # M space steps
Ntime = 10000  # N time steps
S_max = B  # The max of S corresponds to the Barrier
S_min = float(K) / 3
x_max = np.log(S_max)  # A2
x_min = np.log(S_min)  # A1

x, dx = np.linspace(x_min, x_max, Nspace, retstep=True)  # space discretization
T_array, dt = np.linspace(0, T, Ntime, retstep=True)  # time discretization
Payoff = np.maximum(np.exp(x) - K, 0)  # Call payoff

V = np.zeros((Nspace, Ntime))  # grid initialization
offset = np.zeros(Nspace - 2)  # vector to be used for the boundary terms

V[:, -1] = Payoff  # terminal conditions
V[-1, :] = 0  # boundary condition
V[0, :] = 0  # boundary condition
```

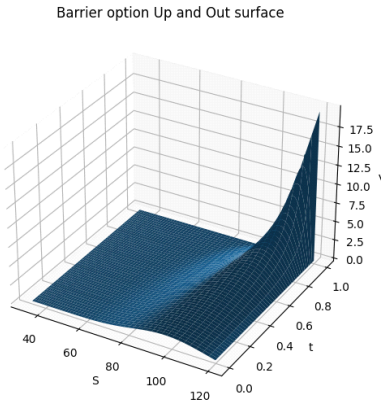
```
# construction of the tri-diagonal matrix D
sig2 = sig * sig
dxx = dx * dx
a = (dt / 2) * ((r - 0.5 * sig2) / dx - sig2 / dxx)
b = 1 + dt * (sig2 / dxx + r)
c = -(dt / 2) * ((r - 0.5 * sig2) / dx + sig2 / dxx)
D = sparse.diags([a, b, c], [-1, 0, 1], shape=(Nspace - 2, Nspace - 2)).tocsc()
DD = splu(D)

# Backward iteration
for i in range(Ntime - 2, -1, -1):
    offset[0] = a * V[0, i]
    offset[-1] = c * V[-1, i]
    V[1:-1, i] = DD.solve(V[1:-1, i + 1] - offset)

# finds the option at S0
oPrice = np.interp(X0, x, V[:, 0])
print("The price of the Up and Out option by PDE is: ", oPrice)
```

The price of the Up and Out option by PDE is:  1.1316897150355167

```
S = np.exp(x)
fig = plt.figure(figsize=(18, 6))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122, projection="3d")

ax1.plot(S, Payoff, color="blue", label="Payoff")
ax1.plot(S, V[:, 0], color="red", label="Barrier curve")
ax1.set_xlim(40, 130)
ax1.set_ylim(0, 1.5)
ax1.set_xlabel("S")
ax1.set_ylabel("V")
ax1.legend(loc="upper right")
ax1.set_title("Curve at t=0")

X, Y = np.meshgrid(T_array, S)
ax2.plot_surface(Y, X, V)
ax2.set_title("Barrier option Up and Out surface")
ax2.set_xlabel("S")
ax2.set_ylabel("t")
```

| | |
|---|---|
| | ax2.set_zlabel("V")<br>plt.show() |




**PDE method - Down and In**

In + Out = Vanilla

| | |
|---|---|
| 48 | # PDE for vanilla<br>Nspace = 5000 # M space steps<br>Ntime = 5000 # N time steps<br>S_max = K * 3 # The max of S corresponds to the Barrier<br>S_min = K / 3<br>x_max = np.log(S_max) # A2<br>x_min = np.log(S_min) # A1<br>B_log = np.log(BB)<br><br>x_b, dx = np.linspace(B_log, x_max, Nspace, retstep=True) # space discretization starting from the barrier<br>x = np.concatenate((np.arange(B_log, x_min, -dx)[::-1][:-1], x_b)) # total x vector<br>N_tot = len(x)<br><br>T_array, dt = np.linspace(0, T, Ntime, retstep=True) # time discretization<br>Payoff = np.maximum(np.exp(x) - K, 0) # Call payoff<br><br>V = np.zeros((N_tot, Ntime)) # grid initialization<br>offset = np.zeros(N_tot - 2) # vector to be used for the boundary terms<br><br>V[:, -1] = Payoff # terminal conditions<br>V[-1, :] = np.exp(x_max) - K * np.exp(-r * T_array[::-1]) # boundary |

```
condition
V[0, :] = 0  # boundary condition

# construction of the tri-diagonal matrix D
sig2 = sig * sig
dxx = dx * dx
a = (dt / 2) * ((r - 0.5 * sig2) / dx - sig2 / dxx)
b = 1 + dt * (sig2 / dxx + r)
c = -(dt / 2) * ((r - 0.5 * sig2) / dx + sig2 / dxx)
D = sparse.diags([a, b, c], [-1, 0, 1], shape=(N_tot - 2, N_tot - 2)).tocsc()
DD = splu(D)

# Backward iteration
for i in range(Ntime - 2, -1, -1):
    offset[0] = a * V[0, i]
    offset[-1] = c * V[-1, i]
    V[1:-1, i] = DD.solve(V[1:-1, i + 1] - offset)

# finds the option at S0
oPrice = np.interp(X0, x, V[:, 0])
print("The price of the vanilla option by PDE is: ", oPrice)
```

The price of the vanilla option by PDE is:  9.667260521704131

```
# PDE for barrier
VB = V[-Nspace:, :]
offset = np.zeros(Nspace - 2)  # vector to be used for the boundary terms

VB[:, -1] = 0  # terminal condition
V[-1, :] = 0  # lateral condition

# construction of the tri-diagonal matrix D
sig2 = sig * sig
dxx = dx * dx
a = (dt / 2) * ((r - 0.5 * sig2) / dx - sig2 / dxx)
b = 1 + dt * (sig2 / dxx + r)
c = -(dt / 2) * ((r - 0.5 * sig2) / dx + sig2 / dxx)
D = sparse.diags([a, b, c], [-1, 0, 1], shape=(Nspace - 2, Nspace - 2)).tocsc()
DD = splu(D)
```

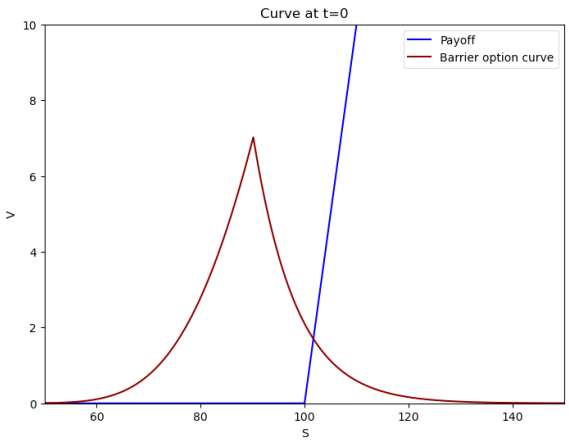
```
# Backward iteration
for i in range(Ntime - 2, -1, -1):
    offset[0] = a * VB[0, i]
    offset[-1] = c * VB[-1, i]
    VB[1:-1, i] = DD.solve(VB[1:-1, i + 1] - offset)

# finds the option at S0
oPrice = np.interp(X0, x_b, VB[:, 0])
print("The price of the Down and In option by PDE is: ", oPrice)
```

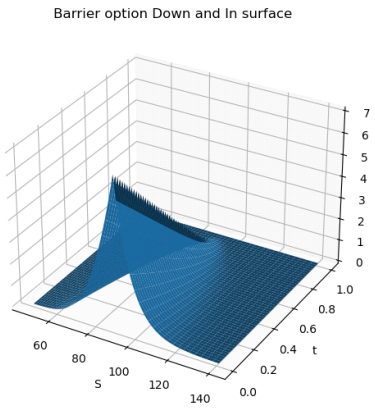
The price of the Down and In option by PDE is:  1.753323938280021

Plots

50

```
S = np.exp(x)
S_b = np.exp(x_b)
fig = plt.figure(figsize=(18, 6))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122, projection="3d")

ax1.plot(S, Payoff, color="blue", label="Payoff")
ax1.plot(S, V[:, 0], color="darkred", label="Barrier option curve")
ax1.set_xlim(50, 150)
ax1.set_ylim(0, 10)
ax1.set_xlabel("S")
ax1.set_ylabel("V")
ax1.legend(loc="upper right")
ax1.set_title("Curve at t=0")

X, Y = np.meshgrid(T_array, S[1500:6000])
ax2.plot_surface(Y, X, V[1500:6000, :])
ax2.set_title("Barrier option Down and In surface")
ax2.set_xlabel("S")
ax2.set_ylabel("t")
ax2.set_zlabel("V")
plt.show()
```




### 3) Asian options

Fixed Strike Arithmetic Average(평균가격) CALL European option

$$V(t,s) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\left(A_T - K\right)^+ \middle| S_t = s\right]$$

Floating Strike Arithmetic Average(평균행사가격) CALL European option

$$V(t,s) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\left(S_T - A_T\right)^+ \middle| S_t = s\right]$$

$$A_t = \frac{1}{t}\int_0^t S_u du$$

### Monte Carlo

51

```
%%time
np.random.seed(seed=41)
N = 10000
paths = 20000
dt = T / (N - 1)  # time interval

# path generation
X_0 = np.zeros((paths, 1))
increments = ss.norm.rvs(loc=(r - sig**2 / 2) * dt, scale=np.sqrt(dt) * sig, size=(paths, N - 1))
X = np.concatenate((X_0, increments), axis=1).cumsum(1)
S = S0 * np.exp(X)

A = np.mean(S, axis=1)  # Average of each path
```

```
# A = S[:,-1]                    # Uncomment to retrieve the Black-Scholes price

# FIXED STRIKE
MC_asian_fixed = np.exp(-r * T) * np.mean(np.maximum(A - K, 0))
asian_fixed_std_err = np.exp(-r * T) * ss.sem(np.maximum(A - K, 0))
print("The price of the fixed strike average call option by Monte Carlo is: ", MC_asian_fixed)
print("with standard error: ", asian_fixed_std_err)
print()

# FLOAT STRIKE
MC_asian_float = np.exp(-r * T) * np.mean(np.maximum(S[:, -1] - A, 0))
asian_float_std_err = np.exp(-r * T) * ss.sem(np.maximum(S[:, -1] - A, 0))
print("The price of the float strike average call option by Monte Carlo is: ", MC_asian_float)
print("with standard error: ", asian_float_std_err)
print()
```

```
The price of the fixed strike average call option by Monte Carlo is: 5.446554927613741
with standard error:  0.055322058241839625

The price of the float strike average call option by Monte Carlo is: 5.51972780624247
with standard error:  0.05877964853611856

CPU times: user 10 s, sys: 2.1 s, total: 12.1 s
Wall time: 12.2 s
```

**Fixed strike. PDE method**

52

```
def gamma(t):
    return 1 / (r * T) * (1 - np.exp(-r * (T - t)))

def get_X0(S0):
    """function that computes the variable x defined above"""
    return gamma(0) * S0 - np.exp(-r * T) * K
```

53

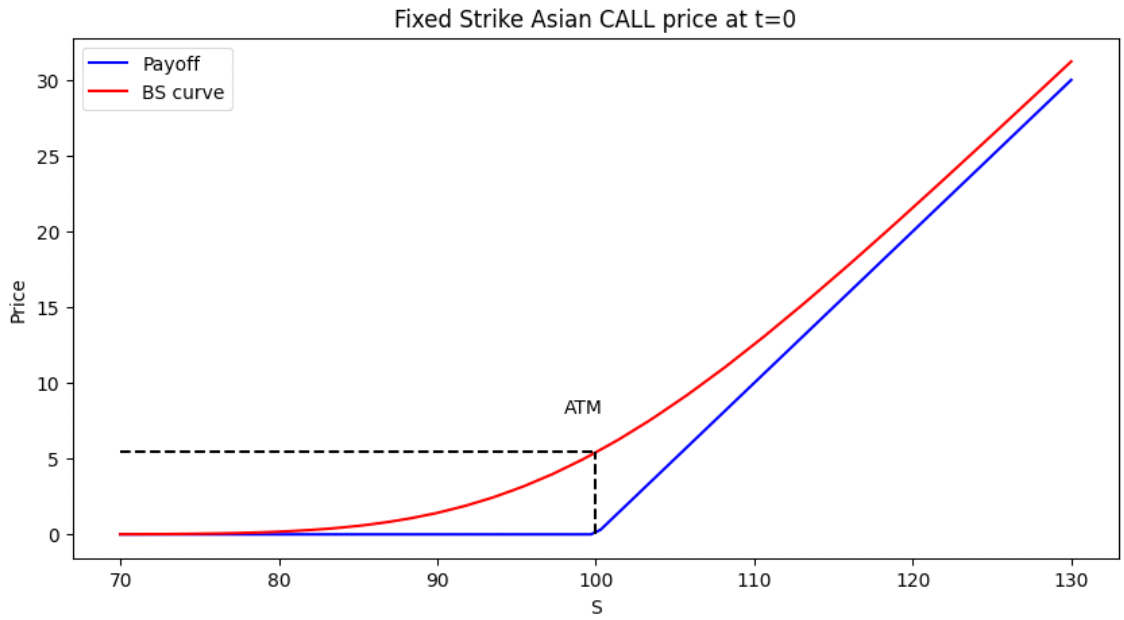
```
# PDE algorithm
Nspace = 6000  # M space steps
Ntime = 6000  # N time steps
```

```
y_max = 60  # A2
y_min = -60  # A1

y, dy = np.linspace(y_min, y_max, Nspace, retstep=True)  # space discretization
T_array, dt = np.linspace(0, T, Ntime, retstep=True)  # time discretization
Payoff = np.maximum(y, 0)  # payoff

G = np.zeros((Nspace, Ntime))  # grid initialization
offset = np.zeros(Nspace - 2)  # vector to be used for the boundary terms

G[:, -1] = Payoff  # terminal conditions
G[-1, :] = y_max  # boundary condition
G[0, :] = 0  # boundary condition

for n in range(Ntime - 2, -1, -1):
    # construction of the tri-diagonal matrix D
    sig2 = sig * sig
    dyy = dy * dy
    a = -0.5 * (dt / dyy) * sig2 * (gamma(T_array[n]) - y[1:-1]) ** 2
    b = 1 + (dt / dyy) * sig2 * (gamma(T_array[n]) - y[1:-1]) ** 2  # main diagonal
    a0 = a[0]
    cM = a[-1]  # boundary terms
    aa = a[1:]
    cc = a[:-1]  # upper and lower diagonals
    D = sparse.diags([aa, b, cc], [-1, 0, 1], shape=(Nspace - 2, Nspace - 2)).tocsc()

    # backward computation
    offset[0] = a0 * G[0, n]
    offset[-1] = cM * G[-1, n]
    G[1:-1, n] = spsolve(D, (G[1:-1, n + 1] - offset))

X0 = get_X0(S0)
oPrice = S0 * np.interp(X0 / S0, y, G[:, 0])
print("The price of the Fixed Strike Asian option by PDE is: ", oPrice)
```

The price of the Fixed Strike Asian option by PDE is:  5.417947407005549

```
S = np.linspace(70, 130, 100)
asian_PDE = S * np.interp(get_X0(S) / S, y, G[:, 0])

fig = plt.figure(figsize=(10, 5))
plt.plot(S, np.maximum(S - K, 0), color="blue", label="Payoff")
plt.plot(S, asian_PDE, color="red", label="BS curve")
plt.xlabel("S")
plt.ylabel("Price")
plt.text(98, 8, "ATM")
plt.plot([S0, S0], [0, oPrice], "k--")
plt.plot([70, S0], [oPrice, oPrice], "k--")
plt.legend(loc="upper left")
plt.title("Fixed Strike Asian CALL price at t=0")
plt.show()
```

54




**Floating strike. PDE method**

```
Nspace = 4000  # M space steps
Ntime = 7000  # N time steps
x_max = 10  # A2
x_min = 0  # A1
x, dx = np.linspace(x_min, x_max, Nspace, retstep=True)  # space discretization
T_array, dt = np.linspace(0.0001, T, Ntime, retstep=True)  # time discretization
Payoff = np.maximum(x - 1, 0)  # Call payoff
```

55

```
V = np.zeros((Nspace, Ntime))  # grid initialization
offset = np.zeros(Nspace - 2)  # vector to be used for the boundary terms
V[:, -1] = Payoff  # terminal conditions
V[-1, :] = x_max - 1  # boundary condition
V[0, :] = 0  # boundary condition

for n in range(Ntime - 2, -1, -1):
    # construction of the tri-diagonal matrix D
    sig2 = sig * sig
    dxx = dx * dx
    max_part = np.maximum(x[1:-1] * (r - (x[1:-1] - 1) / T_array[n]), 0) # upwind positive part
    min_part = np.minimum(x[1:-1] * (r - (x[1:-1] - 1) / T_array[n]), 0) # upwind negative part

    a = min_part * (dt / dx) - 0.5 * (dt / dxx) * sig2 * (x[1:-1]) ** 2
    b = (
        1 + dt * (r - (x[1:-1] - 1) / T_array[n]) + (dt / dxx) * sig2 * (x[1:-1]) ** 2 + dt / dx * (max_part - min_part)
    )  # main diagonal
    c = -max_part * (dt / dx) - 0.5 * (dt / dxx) * sig2 * (x[1:-1]) ** 2

    a0 = a[0]
    cM = c[-1]  # boundary terms
    aa = a[1:]
    cc = c[:-1]  # upper and lower diagonals
    D = sparse.diags([aa, b, cc], [-1, 0, 1], shape=(Nspace - 2, Nspace - 2)).tocsc()  # matrix D

    # backward computation
    offset[0] = a0 * V[0, n]
    offset[-1] = cM * V[-1, n]
    V[1:-1, n] = spsolve(D, (V[1:-1, n + 1] - offset))

# finds the option at S0, assuming it is ATM i.e. x=s/a=1
oPrice = S0 * np.interp(1, x, V[:, 0])
print("The price of the Floating Strike Asian option by PDE is: ", oPrice)
```

```
The price of the Floating Strike Asian option by PDE is:  5.478818157598196
```

| | |
|---|---|
| 56 | S = np.linspace(10, 200, 100)<br>A = np.linspace(80, 120, 100)<br>VV = np.zeros((100, 100))<br>for i in range(len(S)):<br>  for j in range(len(A)):<br>  VV[i, j] = A[j] * np.interp(S[i] / A[j], x, V[:, 0]) |

| | |
|---|---|
| 57 | fig = plt.figure(figsize=(18, 5))<br>gs = gridspec.GridSpec(1, 2, width_ratios=[3, 2])<br>ax1 = fig.add_subplot(gs[0], projection="3d")<br>ax2 = fig.add_subplot(gs[1])<br><br>X, Y = np.meshgrid(A, S)<br>ax1.plot_surface(Y, X, VV)<br>ax1.set_title("Floating strike Asian option surface at time zero")<br>ax1.set_xlabel("Spot price")<br>ax1.set_ylabel("Spot average")<br>ax1.set_zlabel("Price")<br>ax1.view_init(30, -100)<br><br>ax2.plot(x, Payoff, color="blue", label="Payoff")<br>ax2.plot(x, V[:, 0], color="red", label="W curve")<br>ax2.set_xlim(0, 2)<br>ax2.set_ylim(0, 1)<br>ax2.set_title("Auxiliary function W at time zero")<br>ax2.legend(loc="upper left")<br>plt.show() |