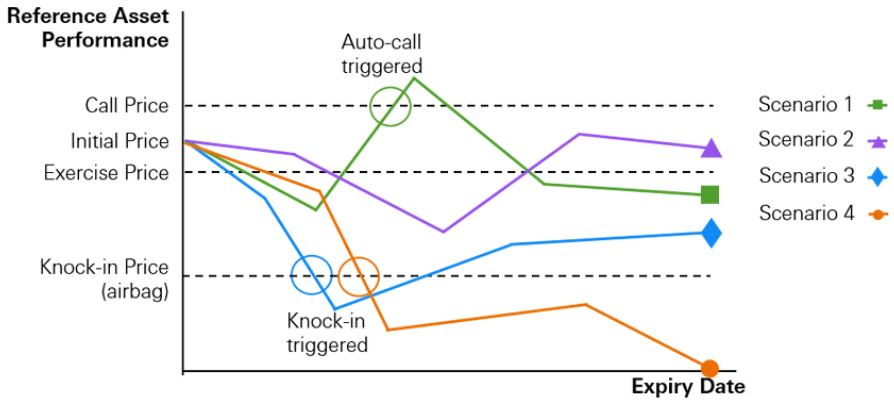
Python으로 ELS 가치평가 및 헤징



Ⅰ. Python for ELS

Ⅱ. 몬테카를로 시뮬레이션(Monte Carlo simulation)

Ⅲ. 유한차분법(Finite Difference Method)

Ⅳ. ELS 헤징 운용

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

Ⅰ. Python for ELS

1. 기본 명령어

자료 입력

01	a=1 print(a)

변수 a에 1을 대입

02	a=2 b=a+1 c=4/3 d=4//3 e=a*3 f=b**3 g=5%3 print('a=%f\n b=%f\n c=%f\n d=%d\n e=%f\n f=%f\n g=%d' %(a,b,c,d,e,f,g))

연산자 실습

//는 몫, **는 거듭제곱, %는 나눈 나머지를 의미

03	c=[1,2,3] length = len(c) print('(c[0], c[1]), c[2]=%d %d %d'%(c[0], c[1], c[2])) print('length=%d'%(length)) c=[[1,2],[3,4]] length = len(c) print('(c[0][0], c[0][1], c[1][0], c[1][1])=%d %d %d %d'%(c[0][0], c[0][1], c[1][0], c[1][1])) print('length=%d'%(length))

파이썬 자료구조 중 list는 대괄호 [ ]로 형성

파이썬에서 인덱스 시작은 0부터

내장함수 len으로 리스트 길이를 확인

04	a=range(10) print('a=%s'%(a))

a는 0부터 9까지 10개의 요소로 이루어진 리스트 자료형 변수

```
05
import numpy as np
a=np.arange(10)
b=np.arange(2,10)
c=np.arange(1,10,2)
print('a=%s'%(a))
print('b=%s'%(b))
print('c=%s'%(c))
```

numpy의 arange함수 이용
a는 0~9 숫자가 리스트 형식으로 저장
b는 2~9 숫자가 리스트 형식으로 저장
c는 1부터 2씩 증가하여 10 이전 숫자가 리스트 형식으로 저장

```
06
a=np.arange(10)
b=a[1:5]
c=a[-1]
d=a[-2]
e=a[3:-1]
f=a[range(3)]
print('a=%s'%(a))
print('b=%s'%(b))
print('c=%s'%(c))
print('d=%s'%(d))
print('e=%s'%(e))
print('f=%s'%(f))
```

인덱스 슬라이싱
b는 인덱스 1부터 4까지
c는 인덱스 맨 마지막, d는 인덱스 맨 마지막에서 두 번째

```
07
x1=np.array([0])
x2=np.arange(64,132,2)
x3=np.array([170,180,200,207,213,223,242,255,267,278,289,300])
x=np.r_[x1,x2,x3]
print('x=%s'%(x))
```

numpy의 r_함수를 이용하여 배열 x1, x2, x3를 하나의 배열로 생성
하나의 행으로 생성할 때 r_함수 이용, 하나의 열로 생성할 때 c_함수 이용

```
08
a=np.arange(10)
a=a.reshape([2,5])
print('a=%s'%(a))
```

reshape함수는 배열을 정해진 크기로 변형, 1행 10열의 배열을 2행 5열로 변형

```
09
a=np.array([[1,2],[3,4]])
print('a=%s'%(a))
```

array 함수를 이용하여 배열 형태를 사용

10	a=np.arange(10) b=a+2 length1=a.size length2=len(a) print('a=%s'%(a)) print('b=%s'%(b)) print('length1=%d'%(length1)) print('length2=%d'%(length2)) c=np.multiply(a,b) print('c=%s'%(c))

numpy 자료형은 b=a+2와 같은 산술연산이 가능
multiply 함수를 이용하여 각 원소끼리 곱셈 가능

11	a=np.array([[1,2],[3,4]]) b=a+3 print('a=%s'%(a)) print('b=%s'%(b)) c=np.dot(a,b) d=np.multiply(a,b) e=a*b print('c=%s'%(c)) print('d=%s'%(d)) print('e=%s'%(e))

dot 함수는 행렬의 곱셈
multiply 함수와 * 연산자는 각 원소끼리 곱셈

행렬 $A=\begin{bmatrix}a\\b\end{bmatrix}$와 $B=\begin{bmatrix}c & d\end{bmatrix}$가 있을 때, 이 둘의 곱셈은 다음과 같다.

$$A\times B=\begin{bmatrix}a\\b\end{bmatrix}\times\begin{bmatrix}c & d\end{bmatrix}=\begin{bmatrix}ac & ad\\bc & bd\end{bmatrix}$$

$$A^T\times B^T=\begin{bmatrix}a & b\end{bmatrix}\times\begin{bmatrix}c\\d\end{bmatrix}=ac+bd$$

$$B\times A=\begin{bmatrix}c & d\end{bmatrix}\times\begin{bmatrix}a\\b\end{bmatrix}=ca+db$$

$$B^T\times A^T=\begin{bmatrix}c\\d\end{bmatrix}\times\begin{bmatrix}a & b\end{bmatrix}=\begin{bmatrix}ca & cb\\da & db\end{bmatrix}$$

행렬 $A=\begin{bmatrix}a & b\\c & d\end{bmatrix}$와 $B=\begin{bmatrix}e & f\\g & h\end{bmatrix}$가 있을 때, 이 둘의 곱셈은 다음과 같다.

$$A\times B=\begin{bmatrix}a & b\\c & d\end{bmatrix}\times\begin{bmatrix}e & f\\g & h\end{bmatrix}=\begin{bmatrix}ae+bg & af+bh\\ce+dg & cf+dh\end{bmatrix}$$

12	a=np.array([[1,2],[3,4]]) b=np.transpose(a) print('a=%s'%(a)) print('b=%s'%(b))

transpose 함수를 이용하여 전치행렬 생성

13	a=np.zeros([3]) b=np.zeros([3,2]) print('a=%s'%(a)) print('b=%s'%(b))

14	a=np.ones([3]) b=np.ones([3,2]) print('a=%s'%(a)) print('b=%s'%(b))

15	a=np.array([[1,2],[3,4]]) b=a+3 b=a+b print('a=%s'%(a)) print('b=%s'%(b)) c=np.dot(a,b) print('c=%s'%(c)) c=np.sum(np.dot(a,b)) d=np.sum(np.dot(a,b), axis=1) e=np.sum(np.dot(a,b), axis=0) print('c=%s'%(c)) print('d=%s'%(d)) print('e=%s'%(e))

sum 함수에서 axis가 1이면 행끼리 더하고, axis가 0이면 열끼리 더한다.

16	a=np.array([[1,2],[3,4]]) b=np.identity(3) print('a=%s'%(a)) print('b=%s'%(b)) c=np.linalg.inv(a) d=np.linalg.cholesky(b) print('c=%s'%(c)) print('d=%s'%(d))

identity 함수는 단위행렬을 만들고, inv 함수는 역행렬, cholesky 함수는 촐레스키 분해를 제공한다.

for 문(반복문)

파이썬에서는 들여쓰기로 문장의 실행 범위를 결정

17	for i in range(5): print(i)

18	a=['H','e', 'l', 'l', 'o'] for ch in a: print(ch)

while 문(반복문)

참일 때 계속 반복, 무한 루프에 조심

19	i=0
	while i<5:
	    print(i)
	    i +=1

if 문(조건문)

if 문은 elif와 else 문과 연결

20	i=3
	if i>0:
	    print('Positive')
	elif i<5:
	    print('i<5')
	else:
	    print('i<=0')

21	i=3
	if i>0:
	    print('Positive')
	if i<5:
	    print('i<5')
	else:
	    print('i<=0')

비교 연산자와 논리 연산자

22	a=10 b=20 l1=(a==b) l2=(a!=b) l3=(a>b)

```
l4=(a<b)
l5=(True and False)
l6=(True or False)
l7=not(False)
l8=(a==b) and (a>b)
print(l1,l2,l3,l4,l5,l6,l7,l8)
```

break 문

break 명령은 조건을 중단할 때 사용

```
for i in range(1,8):
    if i<6:
        print(i)
    if i>3:
        break
```

중첩 for 문

```
for i in range(1,8):
    for j in range(11,13):
        if i<6:
            print(i,j)
        else:
            print('i >=6')
        if i>3:
            print('break!')
            break
        print('i<=3')
```

2. ELS 평가에 유용한 함수들

linspace 문

linspace는 a와 b 사이의 간격이 동일한 n개의 성분을 가진 벡터를 만들 때 사용

```
import numpy as np
a=np.linspace(0,50)  # 0부터 50을 50개의 요소로 나눈 결과
print(a)
```

```
import numpy as np
b=np.linspace(0,50,51) # 0부터 50을 51개의 요소로 나눈 결과
print(b)
```

27	import numpy as np
	c=np.linspace(0,50,endpoint=False) # 끝점 미포함
	print(c)

where 문
n개의 성분을 가진 벡터에서 특정한 값의 위치를 찾고 싶을 때 사용

28	import numpy as np a=np.linspace(1,51,51) find = np.where(a==25) print(find) print(a[find])

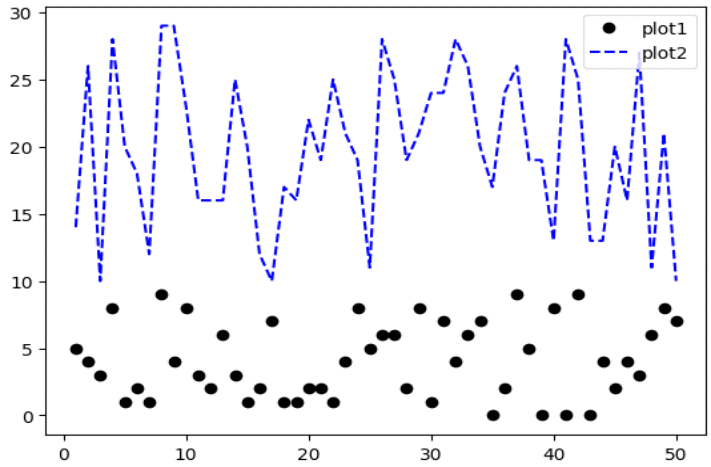
시각화
파이썬 실행결과를 그래프로 시각화하고자 할 때 matplotlib 라이브러리 많이 사용
2차원 평면에 데이터를 표현하는 plot

29	import numpy as np import matplotlib.pyplot as plt %matplotlib inline plt.close()

30	num = 50; low = 0; mid = 10; high = 10 x = np.linspace(1, 50, num) y1 = np.random.randint(low, high, num) y2 = np.random.randint(mid, 3*high, num) plt.plot(x, y1, 'ko', label = 'plot1') plt.plot(x, y2, 'b--', label = 'plot2') plt.legend(loc='upper right')

분포에 관련된 함수
파이썬으로 난수를 생성할 때 주로 numpy의 random 패키지 이용

31	import numpy as np uniform_rnd1 = np.random.rand(3) uniform_rnd2 = np.random.rand(1, 3) norm_rnd = np.random.randn(2, 3) print(uniform_rnd1) print(uniform_rnd2) print(norm_rnd)



uniform_rnd1과 uniform_rnd2는 0과 1 사이에서 균일분포(uniform distribution)로 무작위 수를 생성
norm_rnd는 평균이 0이고 표준편차가 1인 표준정규분포(standard normal distribution)를 따르는 난수를 생성

32	print(uniform_rnd1[0], uniform_rnd2[0]) print(uniform_rnd2[0][0]) print(norm_rnd[0][1])

행렬 곱셈

33	import numpy as np import time num = 1000 a = np.random.rand(num, num) b = np.random.rand(num, num) timer = time.time() c = np.dot(a, b) timer = time.time() - timer print("%.3f seconds" % timer)

역행렬
n × n 행렬의 역행렬을 구하는 작업은 꽤 많은 리소스가 필요
numpy의 linalg 패키지를 이용한 역행렬 계산 예제

34	from numpy.linalg import inv import numpy as np import time num = 1000 a = np.random.rand(num, num) timer = time.time()

```
inv_a = inv(a)
timer = time.time() - timer
print("%.3f seconds" % timer)
```

사용자 정의 함수 만들기

def 명령어를 이용하여 사용자가 임의의 함수를 생성

- 반복되는 구문을 한번만 기술하면 다음에 다시 사용 가능
- 정의되어 있지 않은 함수를 맞춤형으로 생성
- 코드의 가독성을 높임

```
def 함수이름(입력변수 A, 입력변수 B, …):
    코드
```

$ax^2 + bx + c = 0$ 방정식 해를 근의 공식으로 구하는 함수

35
```
from math import sqrt
def root_finding(a, b, c):
    x1=(-b+sqrt(b**2-4*a*c))/(2*a)
    x2=(-b-sqrt(b**2-4*a*c))/(2*a)
    ans = [x1, x2]
    return ans
```

36
```
# example 1 : x^2 -4x +3
a=1; b=-4; c=3
result = root_finding(a, b, c)
print("solution : ", result)
# example 2 : (2x+1)^2
a=4; b=4; c=1
result = root_finding(a, b, c)
print("solution : ", result)
```

특정 입력 변수를 선택 사항으로 작성한 사례

Ⅱ. 몬테카를로 시뮬레이션(Monte Carlo simulation)

1. Monte Carlo simulation 소개

- 복잡한 문제를 난수를 이용하여 얻어진 수차례 시뮬레이션 결과를 종합하여 근사적으로 값을 계산하는 기법
- 예제

  가로 세로가 2인 정사각형에 다트를 수 없이 던지는 게임

  원의 넓이 $= \pi r^2$

  가로 세로가 2인 정사각형에 내접하고 반지름이 1인 원 안에 들어온 다트의 개수 파악

  원의 반지름이 1이므로 다트의 좌표(x, y)가 $\sqrt{x^2+y^2} < 1$을 만족하면 다트가 원 안에 있다고 파악

  무작위로 수많은 다트를 던졌을 때 다트가 원 안에 들어올 확률 P

  $$P \approx \frac{\text{원 안에 들어온 다트의 개수}}{\text{던진 총 다트의 개수}} \approx \frac{\text{원의 넓이}}{\text{정사각형의 넓이}} = \frac{\pi}{4}$$

  $$\pi \approx 4 \times \frac{\text{원 안에 들어온 다트의 개수}}{\text{던진 총 다트의 개수}}$$

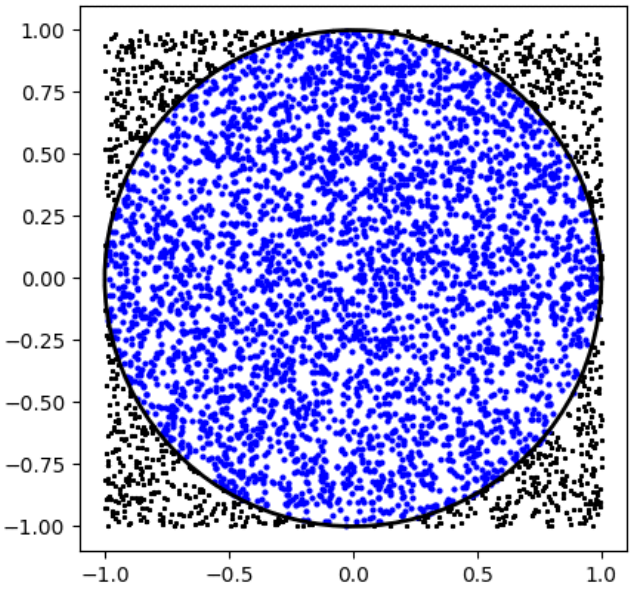
01	import numpy as np import matplotlib.pyplot as plt

02	N=3000; count=0; x=np.zeros([N, 1]); y=np.zeros([N, 1]) z=np.zeros([N, 1]); r=np.zeros([N, 1])

03	x=2*np.random.random((N,1))-1 y=2*np.random.random((N,1))-1 z=np.power(x,2)+np.power(y,2) r=np.sqrt(z)

04	for i in range(N):
	  if r[i]<1:
	  count= count+1
	  plt.plot(x[i], y[i], 'ro', markersize=2)
	  else:
	  plt.plot(x[i], y[i], 'k*', markersize=2)
	t=np.linspace(0, 2*np.pi, 100)
	plt.plot(np.cos(t), np.sin(t), 'k', linewidth=2)
	plt.axis('image'); plt.show()

5	Pi=4*count/N print('Pi=%f'%Pi)



Monte Carlo simulation ELS 가치평가 Flow

구분	단계별 task	세부내용
1단계	ELS 발행조건 확인	- 기초자산 종류 및 개수, 변동성, 상관계수 점검 - ELS 상환조건, 일자 확인 - 쿠폰 이자율, 만기 상환율 확인 - Knock-in 조건 확인
2단계	주가 경로(process)에 대한 모델 결정	모델의 종류 - Geometric Brownian Motion - Constant Elasticity of Variance Model - Jump Diffusion Model - Stochastic Volatility Model
3단계	주가 경로 생성	난수 생성법 - Inverse Transform Method - Box_Muller Approach - Quasi-Random Sequences 기초자산이 2개 이상인 경우 Cholesky 분해를 이용하여 상관관계 반영된 난수 생성
4단계	ELS Payoffs 계산	- 주가 경로에 따른 파생상품 Payoff 계산 - 2, 3단계를 시뮬레이션 횟수만큼 반복하여 Payoff 값들을 도출
5단계	Payoffs 기댓값 추정	- Payoffs 값들의 기댓값 추정(중도상환일, 만기일)
6단계	ELS가치 도출	- 기댓값을 무위험이자율로 할인하여 ELS가치 도출 - 분산감소기법을 적용하여 추정치의 표준오차 감소

2. 주가 경로(stock Process) 생성하기

- 확률과정이란 시간의 진행에 대해 확률적인 변화를 가지는 과정을 의미

- 미래의 주가나 주가지수는 확률과정에 따른다고 가정
- 주가의 확률 과정은 기하 브라운 모형(Geometric Brownian motion)을 따른다고 가정
- 주가 확률 과정

$\frac{dS}{S} = rdt + \sigma dW$, 여기서 σ는 변동성

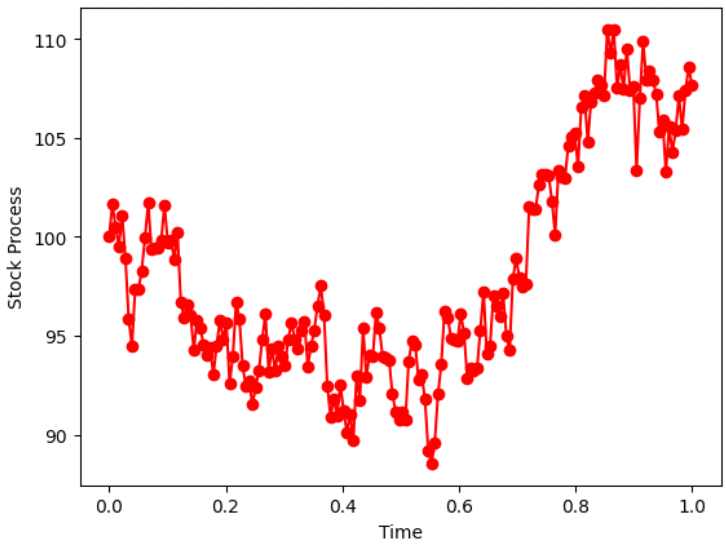
- 주가 경로식

$S(t+\Delta t) = S(t)\exp[(r-\frac{\sigma^2}{2})\Delta t + \sigma\sqrt{\Delta t}\,Z],\ Z \sim N(0,1)$

6	import numpy as np import matplotlib.pyplot as plt

주가 경로 1개(변동성 20%, 무위험이자율 2%, 180일)

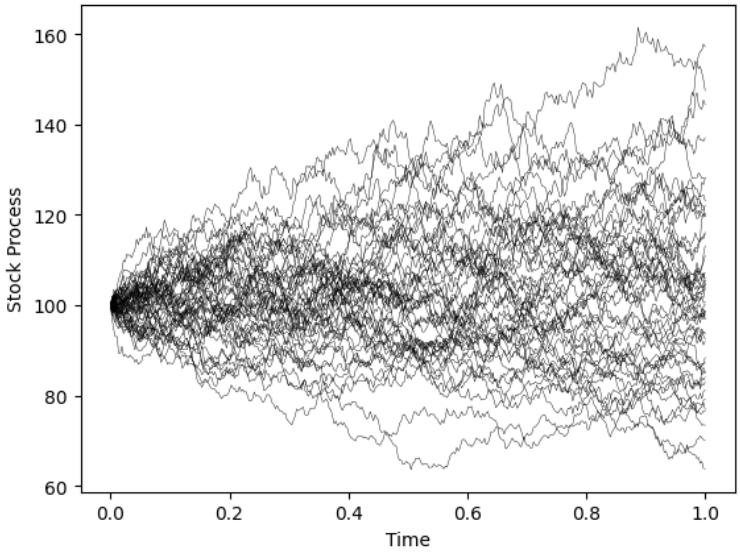
7	N=180; S=np.zeros([N,1]); S[0]=100 vol=0.2; r=0.02; T=1; dt=T/N t=np.linspace(0, T, N) z=np.random.normal(0, 1, N) for i in range(N-1): S[i+1,0]=S[i,0]*np.exp((r-0.5*vol**2)*dt+vol*z[i]*np.sqrt(dt)) plt.plot(t, S[:,0], 'bo-') plt.xlabel('Time') plt.ylabel('Stock Process') plt.show()



주가경로 50개(변동성 20%, 무위험이자율 2%, 365일)

8	N=365; S=np.zeros([N,1]); S[0]=100 vol=0.2; r=0.02; T=1; dt=T/N t=np.linspace(0, T, N)

```
plt.xlabel('Time')
plt.ylabel('Stock Process')
for k in range(0,50):
    z=np.random.normal(0, 1, N)
    for i in range(N-1):
        S[i+1,0]=S[i,0]*np.exp((r-0.5*vol**2)*dt+vol*z[i]*np.sqrt(dt))
    plt.plot(t[:], S[:], 'k-', linewidth=0.3)
plt.show()
```



3. MC를 이용한 ELS 가격결정

1) 기초자산이 1개인 ELS

① KOSPI200이 기초자산인 ELS 평가

투자설명서 요약

- 기초자산가격 변동성 : KOSPI200 지수 : 17.78%

  변동성 기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005)) 하여 Volatility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용

- 공정가격 : 2018년 03월 13일 기준 9,733.63원으로 추산
- 최초기준가격평가일 : 2018년 03월 23일
- 자동조기상환평가일 및 상환금액

차수	자동조기상환평가일	자동조기상환 조건	상환금액
1차	2018년 09월 19일	최초기준가격의 95%이상	액면금액 × 102.20%
2차	2019년 03월 20일	최초기준가격의 95%이상	액면금액 × 104.40%
3차	2019년 09월 19일	최초기준가격의 95%이상	액면금액 × 106.60%
4차	2020년 03월 19일	최초기준가격의 90%이상	액면금액 × 108.80%
5차	2020년 09월 21일	최초기준가격의 90%이상	액면금액 × 111.00%

- 만기평가일 : 2021년 03월 19일

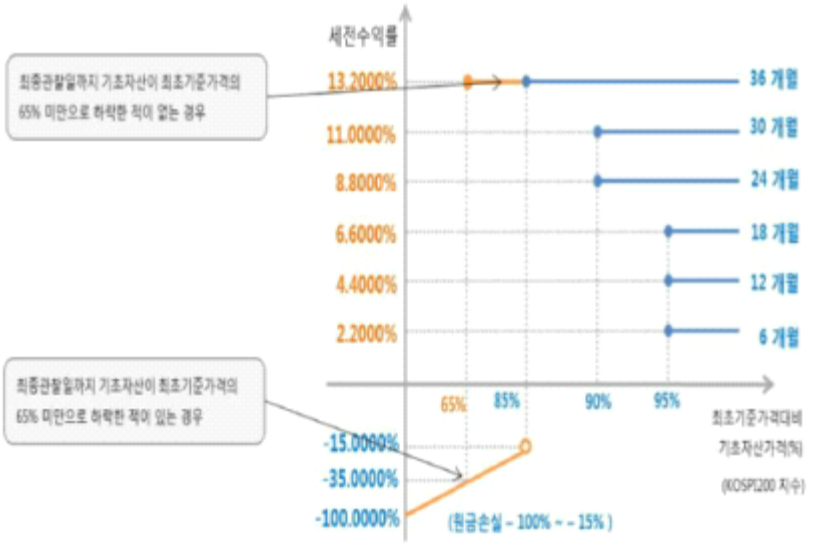
- 만기상환 조건

ⓐ 기초자산의 만기평가가격이 최초기준가격의 85% 이상인 경우: 액면금액 × 113.20%

ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격평가일 익일로부터 최종관찰일(포함)까지 기초자산의 평가가격이 최초기준가격의 65%미만으로 하락한 적이 없는 경우: 액면금액 × 113.20%

ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격평가일 익일로부터 최종관찰일(포함)까지 기초자산의 평가가격이 최초기준가격의 65%미만으로 하락한 적이 있는 경우: 액면금액 × (만기평가가격/최초기준가격)

- 예상 손익구조 그래프



- 파이썬 코드로 구현

9	import numpy as np from datetime import date

10	n=10000 # 시뮬레이션 횟수
	r=0.0210 # 이자율, 3년 국고채 금리
	vol=0.1778 # 변동성
	n0=date.toordinal(date(2018,3,23)) # 최초 기준가격 결정일
	n1=date.toordinal(date(2018,9,19)) # 1차 조기 상환일
	n2=date.toordinal(date(2019,3,20)) # 2차 조기 상환일
	n3=date.toordinal(date(2019,9,19)) # 3차 조기 상환일
	n4=date.toordinal(date(2020,3,19)) # 4차 조기 상환일
	n5=date.toordinal(date(2020,9,21)) # 5차 조기 상환일
	n6=date.toordinal(date(2021,3,19)) # 만기 상환일
	
	check_day=np.array([n1-n0, n2-n0, n3-n0, n4-n0, n5-n0, n6-n0])
	oneyear=365; tot_date=n6-n0
	dt=1/oneyear

```
S=np.zeros([tot_date+1, 1])
S[0]=100.0  # 기초자산 초기값
strike_price=np.array([0.95, 0.95, 0.95, 0.90, 0.90, 0.85])*S[0]  # 조기 행사가격
repay_n=len(strike_price) # 조기상환 횟수

early_count = np.zeros([repay_n, 1]) # 각 조기상환기간 충족횟수
maturity_count = 0 # 만기상환 발생횟수
lose_count = 0 # 만기손실 발생횟수

coupon_rate=np.array([0.022, 0.044, 0.066, 0.088, 0.11, 0.132])  # 조기 상환시 쿠폰 이자율
payment=np.zeros([repay_n, 1])
facevalue=10000  # 액면금액
tot_payoff=np.zeros([repay_n, 1])  # 전체 페이오프
payoff=np.zeros([repay_n, 1])  # 페이오프
discount_payoff=np.zeros([repay_n, 1])  # 페이오프의 현가
kib=0.65*S[0]; dummy=0.132  # 낙인 배리어, 더미 이자율
```

11

```
for j in range(repay_n):
    payment[j]=facevalue*(1+coupon_rate[j])
```

12

```
for i in range(n):
    z=np.random.normal(0, 1, size=[tot_date,1])  # 만기상환일 만큼의 난수 생성
    for j in range(tot_date):
        S[j+1]=S[j]*np.exp((r-0.5*vol**2)*dt+vol*np.sqrt(dt)*z[j])  # 임의의 주가 경로 생성
    S_checkday=S[check_day]
    payoff=np.zeros([repay_n, 1]); repay_event=0
    for j in range(repay_n):
        if S_checkday[j] >= strike_price[j]:  # 조기상환일에 주가를 체크, 조기상환여부 결정
            payoff[j]=payment[j]
            early_count[j] += 1
            repay_event=1
            break
    if repay_event == 0:  # 조기상환 되지 않고 만기까지 온 경우
       if min(S) > kib:  # 낙인 배리어 아래로 내려간 적이 없는 경우
           payoff[-1]=facevalue*(1+dummy)
```

```
                maturity_count += 1
            else:  # 낙인 배리어 아래로 내려간 적이 있는 경우
                payoff[-1]=facevalue*(S[-1]/S[0])
                lose_count += 1
        tot_payoff=tot_payoff+payoff
mean_payoff=tot_payoff/n
for j in range(repay_n):  # 페이오프를 무위험 이자율로 할인하여 현재 가격을 구함
        discount_payoff[j]=mean_payoff[j]*np.exp(-r*check_day[j]/oneyear)
price=np.sum(discount_payoff)
print(price)
```

9871.538134971748 (ELS 최종가격)

13

```
print('총 시뮬레이션 횟수 : %d' % (n))
print('조기상환 발생횟수')
for j in range(repay_n):
    print('%d차 : %d, 발생빈도 : %.2f' % (j+1, early_count[j], early_count[j]/n*100),"%")

print('만기상환 발생횟수 : %d, 발생빈도 : %.2f' % (maturity_count, (maturity_count)/n*100), "%")
print('만기손실 발생횟수 : %d, 발생빈도 : %.2f' % (lose_count, (lose_count/n)*100), "%")
```

총 시뮬레이션 횟수 : 10000
조기상환 발생횟수
1차 : 6556, 발생빈도 : 65.56 %
2차 : 1014, 발생빈도 : 10.14 %
3차 : 459, 발생빈도 : 4.59 %
4차 : 388, 발생빈도 : 3.88 %
5차 : 215, 발생빈도 : 2.15 %
6차 : 243, 발생빈도 : 2.43 %
만기상환 발생횟수 : 283, 발생빈도 : 2.83 %
만기손실 발생횟수 : 842, 발생빈도 : 8.42 %

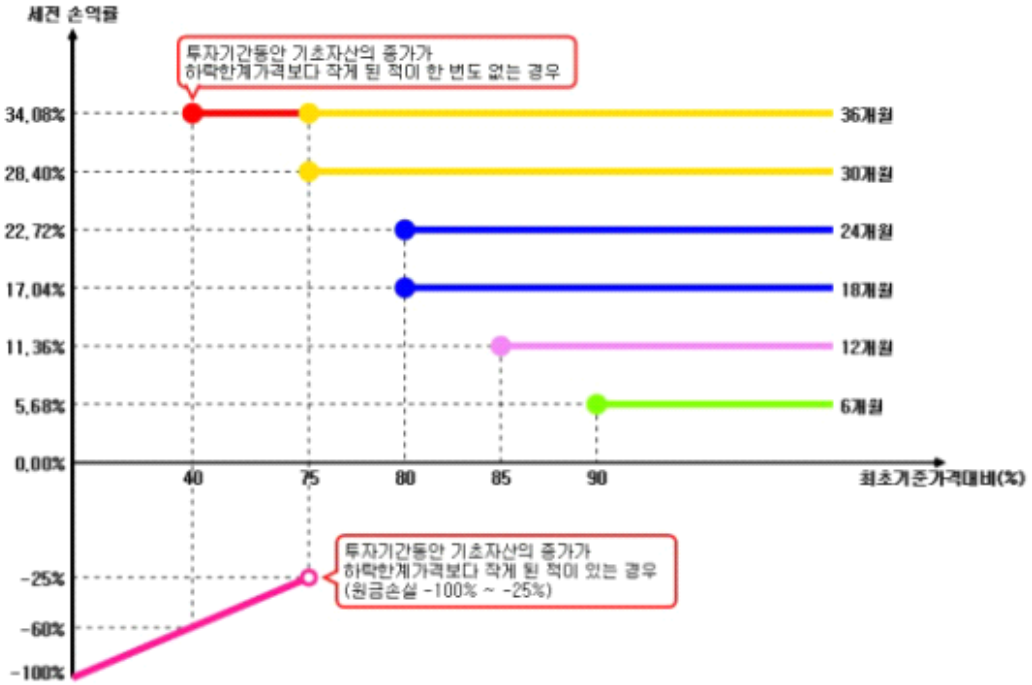
② 개별주식이 기초자산인 ELS 평가

투자설명서 요약
- 기초자산 : LG화학 보통주(051910)
- 기초자산가격 변동성 : LG화학 보통주(051910) : 42.17%
  기초자산가격 변동성 산출기준 LG화학 보통주(051910): 6개월 변동성과 3년 변동성의 평균
- 공정가격 : 일괄신고추가서류 제출일 전일 기준 9,562.87원으로 추산
- 최초기준가격결정일 : 2023년 05월 09일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2023년 11월 08일	최초기준가격의 90%이상	액면금액 × 105.68%
2차	2024년 05월 08일	최초기준가격의 85%이상	액면금액 × 111.36%
3차	2024년 11월 08일	최초기준가격의 80%이상	액면금액 × 117.04%
4차	2025년 05월 08일	최초기준가격의 80%이상	액면금액 × 122.72%
5차	2025년 11월 07일	최초기준가격의 75%이상	액면금액 × 128.40%

- 만기평가일 : 2026년 05월 06일, 2026년 05월 07일, 2026년 05월 08일
- 최종기준가격 : LG화학 보통주(051910) 최종기준가격 결정일 현재 기초자산의 거래소 종가 3개의 산술평균
- 만기행사가격 : LG화학 보통주(051910) 최초기준가격 × 75%
- 하락한계가격 : LG화학 보통주(051910) 최초기준가격 × 40%
- 만기상환 조건
  ⓐ 기초자산의 최종기준가격이 최초기준가격의 75% 이상인 경우: 액면금액 × 134.08%
  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 마지막 최종기준가격 결정일(포함) 이전까지 기초자산이 종가에 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 134.08%
  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 마지막 최종기준가격 결정일(포함) 이전까지 기초자산이 종가에 한 번이라도 하락한계가격보다 작게 된 적이 있는 경우: 액면금액 × (최종기준가격 / 최초기준가격)

- 예상 손익구조 그래프



- 파이썬 코드로 구현

14	import numpy as np from datetime import date

15	n=10000 # 시뮬레이션 횟수
	r=0.0325 # 이자율, 3년 국고채 금리
	vol=0.4217 # 변동성
	n0=date.toordinal(date(2023,5,8)) # 최초 기준가격 결정일
	n1=date.toordinal(date(2023,11,8)) # 1차 조기 상환일
	n2=date.toordinal(date(2024,5,8)) # 2차 조기 상환일
	n3=date.toordinal(date(2024,11,8)) # 3차 조기 상환일
	n4=date.toordinal(date(2025,5,8)) # 4차 조기 상환일
	n5=date.toordinal(date(2025,11,7)) # 5차 조기 상환일
	n6=date.toordinal(date(2026,5,8)) # 만기 상환일
	
	check_day=np.array([n1-n0, n2-n0, n3-n0, n4-n0, n5-n0, n6-n0])
	oneyear=365; tot_date=n6-n0
	dt=1/oneyear
	S=np.zeros([tot_date+1, 1])
	S[0]=100.0 # 기초자산 초기값
	strike_price=np.array([0.90, 0.85, 0.80, 0.80, 0.75, 0.75])*S[0] # 조기 행사가

```
격
repay_n=len(strike_price) # 조기상환 횟수

early_count = np.zeros([repay_n, 1]) # 각 조기상환기간 충족횟수
maturity_count = 0 # 만기상환 발생횟수
lose_count = 0 # 만기손실 발생횟수

coupon_rate=np.array([0.0568, 0.1136, 0.1704, 0.2272, 0.284, 0.3408])  # 조기 상환시 쿠폰 이자율
payment=np.zeros([repay_n, 1])
facevalue=10000  # 액면금액
tot_payoff=np.zeros([repay_n, 1])  # 전체 페이오프
payoff=np.zeros([repay_n, 1])  # 페이오프
discount_payoff=np.zeros([repay_n, 1])  # 페이오프의 현가
kib=0.4*S[0]; dummy=0.3408  # 낙인 배리어, 더미 이자율
```

16

```
for j in range(repay_n):
    payment[j]=facevalue*(1+coupon_rate[j])
```

17

```
for i in range(n):
    z=np.random.normal(0, 1, size=[tot_date,1])  # 만기상환일 만큼 난수 생성
    for j in range(tot_date):
        S[j+1]=S[j]*np.exp((r-0.5*vol**2)*dt+vol*np.sqrt(dt)*z[j])  # 임의의 주가 경로 생성
    S_checkday=S[check_day]
    payoff=np.zeros([repay_n, 1]); repay_event=0
    for j in range(repay_n):
        if S_checkday[j] >= strike_price[j]:  # 조기상환일에 주가를 체크, 조기상환여부 결정
            payoff[j]=payment[j]
            early_count[j] += 1
            repay_event=1
            break
    if repay_event == 0:  # 조기상환 되지 않고 만기까지 온 경우
        if min(S) > kib:  # 낙인 배리어 아래로 내려간 적이 없는 경우
            payoff[-1]=facevalue*(1+dummy)
            maturity_count += 1
        else:  # 낙인 배리어 아래로 내려간 적이 있는 경우
            St = (S[-3]+S[-2]+S[-1])/3
            payoff[-1]=facevalue*(St/S[0])
            lose_count += 1
```

```
        tot_payoff=tot_payoff+payoff
mean_payoff=tot_payoff/n
for j in range(repay_n):  # 페이오프를 무위험 이자율로 할인하여 현재 가격을 구함
        discount_payoff[j]=mean_payoff[j]*np.exp(-r*check_day[j]/oneyear)
price=np.sum(discount_payoff)
print(price)
```

9656.191074369899 (ELS 최종가격)

18

```
print('총 시뮬레이션 횟수 : %d' % (n))
print('조기상환 발생횟수')
for j in range(repay_n):
        print('%d차 : %d, 발생빈도 : %.2f' % (j+1, early_count[j], early_count[j]/n*100),"%")

print('만기상환 발생횟수 : %d, 발생빈도 : %.2f' % (maturity_count, (maturity_count)/n*100), "%")
print('만기손실 발생횟수 : %d, 발생빈도 : %.2f' % (lose_count, (lose_count/n)*100), "%")
```

총 시뮬레이션 횟수 : 10000
조기상환 발생횟수
1차 : 6076, 발생빈도 : 60.76 %
2차 : 1173, 발생빈도 : 11.73 %
3차 : 550, 발생빈도 : 5.50 %
4차 : 281, 발생빈도 : 2.81 %
5차 : 221, 발생빈도 : 2.21 %
6차 : 151, 발생빈도 : 1.51 %
만기상환 발생횟수 : 215, 발생빈도 : 2.15 %
만기손실 발생횟수 : 1333, 발생빈도 : 13.33 %

2) 기초자산이 2개인 ELS

① 2개의 기초자산이 상관계수를 갖는 주가 경로 생성
- 기초자산이 2개인 경우 2개의 주가 움직임이 독립적이지 않고 상관관계를 가짐
- 2개의 주가경로를 무작위로 생성하기 위해 촐레스키 분해(Cholesky decomposition)을 이용하여 상관관계를 갖는 난수 생성이 필요
- 상관관계를 갖는 난수 생성 과정
  ⓐ 표준정규분포를 따르는 난수 Φ_1과 Φ_2를 생성
  ⓑ 기초자산 1과 2의 상관계수(ρ) 행렬 A를 생성

$$A = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$$

ⓒ 행렬 A를 촐레스키 분해

$$A = LL^T = \begin{pmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{pmatrix} \begin{pmatrix} 1 & \rho \\ 0 & \sqrt{1-\rho^2} \end{pmatrix}$$

ⓓ 분해된 행렬 L을 이용하여 상관계수가 반영된 난수 Φ_1^*와 Φ_2^*로 전환

$$\begin{pmatrix} \Phi_1^* \\ \Phi_2^* \end{pmatrix} = L \begin{pmatrix} \Phi_1 \\ \Phi_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{pmatrix} \begin{pmatrix} \Phi_1 \\ \Phi_2 \end{pmatrix} = \begin{pmatrix} \Phi_1 \\ \Phi_1\rho + \Phi_2\sqrt{1-\rho^2} \end{pmatrix}$$

- 파이썬에서는 np.linalg.cholesky이라는 내장함수를 이용하여 상관관계를 갖는 난수 생성
- 주가 경로식

$$S_1(t+\Delta t) = S_1(t)\exp[(r - \frac{\sigma_1^2}{2})\Delta t + \sigma_1\sqrt{\Delta t}\,\Phi_1^*],\ \Phi_1^* \sim N(0,1)$$

$$S_2(t+\Delta t) = S_2(t)\exp[(r - \frac{\sigma_2^2}{2})\Delta t + \sigma_2\sqrt{\Delta t}\,\Phi_2^*],\ \Phi_2^* \sim N(0,1)$$

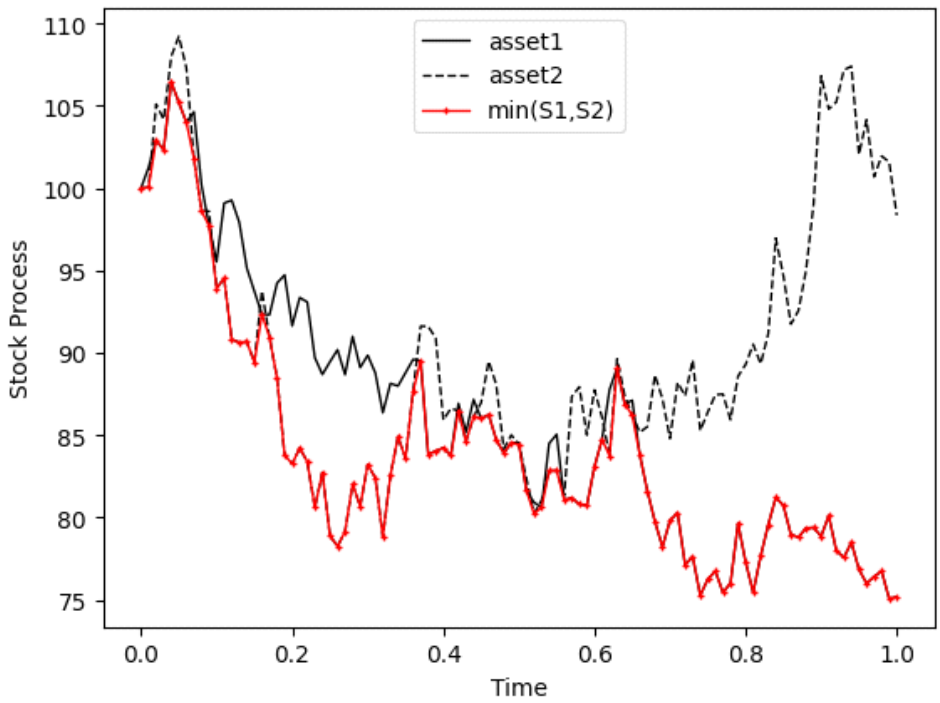
- 파이썬 코드로 구현

19	import numpy as np import matplotlib.pyplot as plt from numpy.ma.core import correlate plt.close()

20	x_vol=0.25; y_vol=0.30
	r=0.035
	rho=0.3
	N=100; T=1
	S1=np.zeros((N+1, 1))
	S2=np.zeros((N+1, 1))
	S1[0]=100; S2[0]=100
	dt=T/N; t=np.linspace(0, T, N+1)
	correlation=np.array([[1, rho], [rho, 1]])
	cholesky=np.linalg.cholesky(correlation)
	z0=np.random.normal(0, 1, size=[N, 2])
	np.random.seed(56)
	z0=np.transpose(z0) # 행벡터를 열벡터로 전치
	z=np.matmul(cholesky, z0) # 상관관계가 반영된 난수생성
	Worst_performer=np.zeros((N+1, 1))

21	for i in range(N): S1[i+1]=S1[i]*np.exp((r-0.5*x_vol**2)*dt+x_vol*z[0,i]*np.sqrt(dt)) S2[i+1]=S2[i]*np.exp((r-0.5*y_vol**2)*dt+y_vol*z[1,i]*np.sqrt(dt)) Worst_performer[i]=min(S1[i,0], S2[i,0]) Worst_performer[-1]=min(S1[-1,0], S2[-1,0])

```
plt.plot(t, S1[:], 'k-', label='asset1', linewidth=1, markersize=3.5)
plt.plot(t, S2[:], 'k--', label='asset2', linewidth=1, markersize=3.5)
plt.plot(t, Worst_performer[:], 'r+-', label='min(S1,S2)', linewidth=1, markersize=3.5)

# plt.xlim(0, 1.0)
# plt.ylim(70, 130)
plt.xlabel('Time')
plt.ylabel('Stock Process')
plt.legend(prop={'size':10})
plt.show()
```



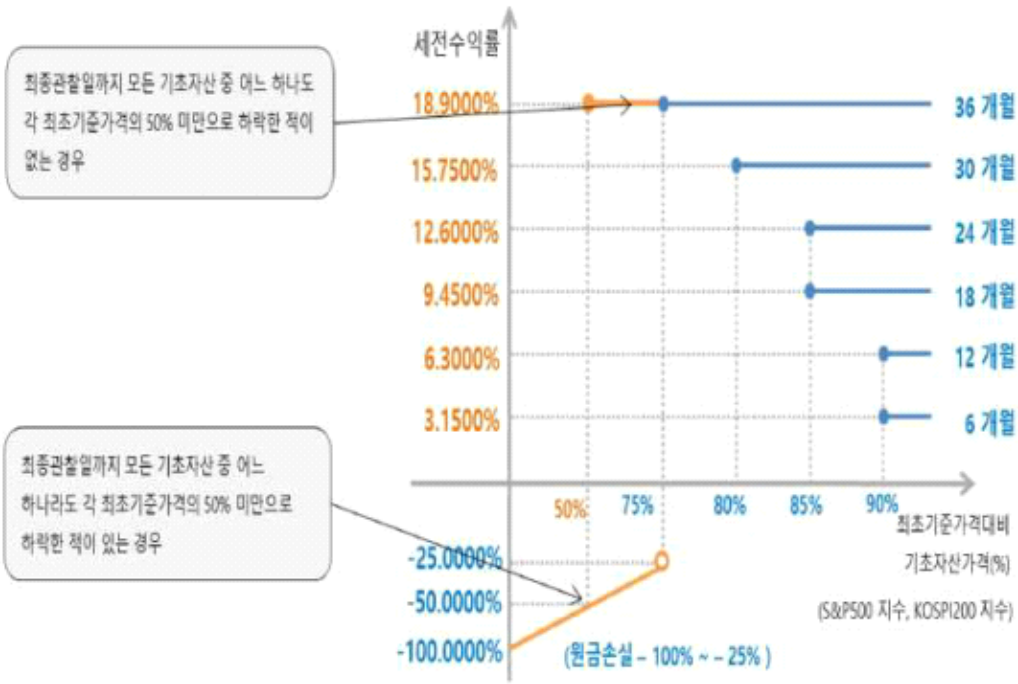
② 기초자산이 두 개(지수)인 ELS 평가

투자설명서 요약
- 기초자산 : KOSPI200, S&P500
- 기초자산가격 변동성 : KOSPI200 : 23.22%, S&P500 : 27.38%
  변동성 산출기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005))하여 Vola tility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용
- 기초자산간의 상관계수 KOSPI200, S&P500 : 0.0489
  상관계수 산출기준 : 180 영업일 역사적 상관계수

- 공정가격 : 본 증권의 공정가격은 2023년 04월 25일 기준 9,446.82원으로 추산됩니다.
- 최초기준가격결정일 : 2023년 05월 15일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2023년 11월 10일	두 지수 모두 최초기준가격의 90%이상	액면금액 × 103.15%
2차	2024년 05월 9일	두 지수 모두 최초기준가격의 90%이상	액면금액 × 106.30%
3차	2024년 11월 12일	두 지수 모두 최초기준가격의 85%이상	액면금액 × 109.45%
4차	2025년 05월 12일	두 지수 모두 최초기준가격의 85%이상	액면금액 × 112.60%
5차	2025년 11월 11일	두 지수 모두 최초기준가격의 80%이상	액면금액 × 115.75%

- 만기평가일 : 2026년 05월 12일
- 최종기준가격 : 최종기준가격 결정일 현재 기초자산의 거래소 종가 또는 산출된 종가
- 만기행사가격 : KOSPI200 최초기준가격 × 75%, S&P500 최초기준가격 × 75%
- 하락한계가격 : KOSPI200 최초기준가격 × 50%, S&P500 최초기준가격 × 50%
- 만기상환 조건
  ⓐ 기초자산의 최종기준가격이 모두 최초기준가격의 75% 이상인 경우: 액면금액 × 118.90%
  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 각각의 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 118.90%
  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 한 번이라도 각각의 하락한계 가격보다 작게 된 적이 있는 경우: 기준종목 기준으로 {(만기평가가격/최초기준가격)- 1} × 100%
  ※ 기준종목 : 모든 기초자산 중 [만기평가가격/최초기준가격]의 비율이 가장 낮은 기초자산



- 파이썬 코드로 구현

22	import numpy as np from numpy.ma.core import correlate from datetime import date

23	n=10000; r=0.0355 # 시뮬레이션 횟수, 무위험이자율
	x_vol=0.2322; y_vol=0.2738 # 두 지수의 연간 변동성
	n0=date.toordinal(date(2023, 5, 15)) # 최초 기준가격 결정일
	n1=date.toordinal(date(2023, 11, 10)) # 1차 조기 상환일
	n2=date.toordinal(date(2024, 5, 9)) # 2차 조기 상환일
	n3=date.toordinal(date(2024, 11, 12)) # 3차 조기 상환일
	n4=date.toordinal(date(2025, 5, 12)) # 4차 조기 상환일
	n5=date.toordinal(date(2025, 11, 11)) # 5차 조기 상환일
	n6=date.toordinal(date(2026, 5, 12)) # 만기 상환일
	check_day=np.array([n1-n0, n2-n0, n3-n0, n4-n0, n5-n0, n6-n0])
	rho=0.0489; corr=np.array([[1, rho], [rho, 1]]) # 상관계수
	coupon_rate=np.array([0.0315, 0.0630, 0.0945, 0.1260, 0.1575, 0.1890]) # 조기 상환시 쿠폰 이자율
	oneyear=365; tot_date=n6-n0; dt=1/oneyear
	k=np.linalg.cholesky(corr)
	S1=np.zeros([tot_date+1, 1])
	S2=np.zeros([tot_date+1, 1])
	S1[0]=100.0; S2[0]=100.0 # 기초자산 초기값
	ratio_S1=S1[0]; ratio_S2=S2[0]
	strike_price=np.array([0.90, 0.90, 0.85, 0.85, 0.80, 0.75]) # 조기 행사가격
	repay_n=len(strike_price) # 조기상환 횟수
	payment=np.zeros([repay_n, 1])
	payoff=np.zeros([repay_n, 1]) # 페이오프
	tot_payoff=np.zeros([repay_n, 1]) # 전체 페이오프
	discount_payoff=np.zeros([repay_n, 1]) # 페이오프의 현가
	early_count = np.zeros([repay_n, 1]) # 각 조기상환기간 충족횟수
	maturity_count = 0 # 만기상환 발생횟수
	lose_count = 0 # 만기손실 발생횟수
	facevalue=10**4 # 액면금액
	kib=0.5; dummy=0.1890 # 낙인 배리어, 더미 이자율
	for j in range(repay_n):
	  payment[j]=facevalue*(1+coupon_rate[j])

24

```
for i in range(n):
    w0=np.random.normal(0, 1, size=[tot_date,2])  # 만기상환일 만큼의 난수 생성
    w0=np.transpose(w0)
    w=np.matmul(k, w0)
    for j in range(tot_date):
        S1[j+1]=S1[j]*np.exp((r-0.5*x_vol**2)*dt+x_vol*np.sqrt(dt)*w[0, j])  #
        S2[j+1]=S2[j]*np.exp((r-0.5*y_vol**2)*dt+y_vol*np.sqrt(dt)*w[1, j])  #
    R1=S1/ratio_S1; R2=S2/ratio_S2
    WP=np.minimum(R1, R2)
    WP_checkday=WP[check_day]
    payoff=np.zeros([repay_n, 1]); repay_event=0
    for j in range(repay_n):
        if WP_checkday[j] >= strike_price[j]:  # 조기상환일에 주가를 체크, 조기 상환여부 결정
            payoff[j]=payment[j]
            early_count[j] += 1
            repay_event=1
            break
    if repay_event == 0:  # 조기상환 되지 않고 만기까지 온 경우
       if min(WP) > kib:  # 낙인 배리어 아래로 내려간 적이 없는 경우
           payoff[-1]=facevalue*(1+dummy)
           maturity_count += 1
       else:  # 낙인 배리어 아래로 내려간 적이 있는 경우
           payoff[-1]=facevalue*WP[-1]
           lose_count += 1
    tot_payoff=tot_payoff + payoff
mean_payoff=tot_payoff/n
for j in range(repay_n):  # 페이오프를 무위험 이자율로 할인하여 현재 가격을 구함
    discount_payoff[j]=mean_payoff[j]*np.exp(-r*check_day[j]/oneyear)
price=np.sum(discount_payoff)
print(price)
```

9542.36494500768 (ELS 최종가격)

25

```
print('총 시뮬레이션 횟수 : %d' % (n))
print('조기상환 발생횟수')
for j in range(repay_n):
    print('%d차 : %d, 발생빈도 : %.2f' % (j+1, early_count[j], early_count[j]/n*100),"%")
```

	print('만기상환 발생횟수 : %d, 발생빈도 : %.2f' % (maturity_count, (maturity_count)/n*100), "%") print('만기손실 발생횟수 : %d, 발생빈도 : %.2f' % (lose_count, (lose_count/n)*100), "%")

총 시뮬레이션 횟수 : 10000
조기상환 발생횟수
1차 : 5376, 발생빈도 : 53.76 %
2차 : 1000, 발생빈도 : 10.00 %
3차 : 788, 발생빈도 : 7.88 %
4차 : 334, 발생빈도 : 3.34 %
5차 : 379, 발생빈도 : 3.79 %
6차 : 285, 발생빈도 : 2.85 %
만기상환 발생횟수 : 622, 발생빈도 : 6.22 %
만기손실 발생횟수 : 1216, 발생빈도 : 12.16 %

③ 기초자산이 두 개(개별종목)인 ELS 평가

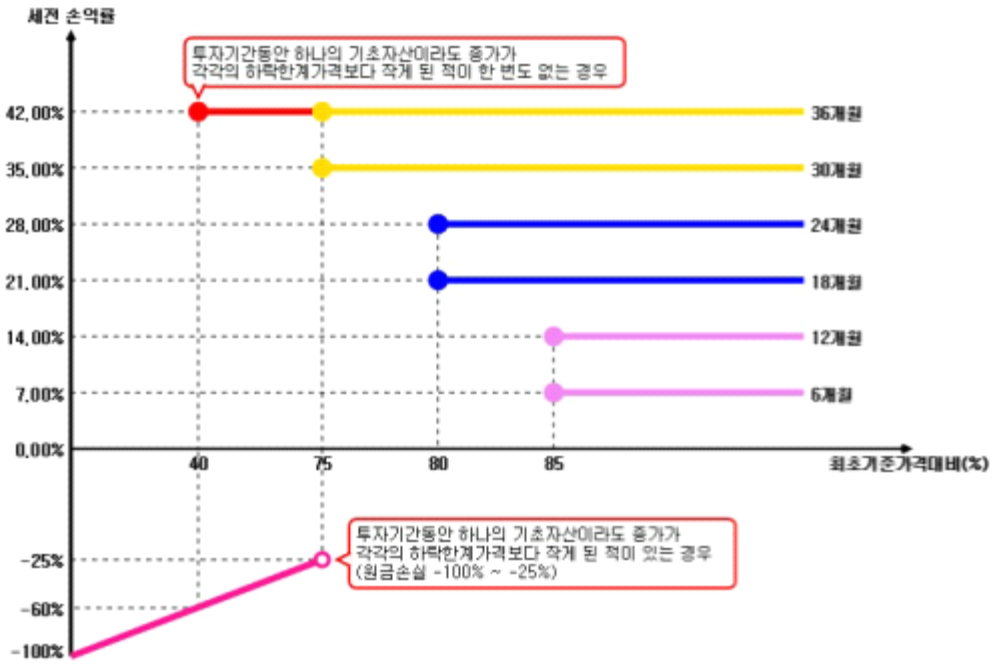
투자설명서 요약
- 기초자산 : LG화학 보통주(051910), LG전자 보통주(066570)
- 기초자산가격 변동성 : LG화학 보통주(051910) : 42.17%, LG전자 보통주(066570) : 35.09%
  변동성 산출기준: 6개월 변동성과 3년 변동성의 평균
- 기초자산간의 상관계수 LG화학 보통주(051910), LG전자 보통주(066570) : 0.29555
  상관계수 산출기준 : 6개월 상관계수와 3년 상관계수의 평균
- 공정가격 : 본 일괄신고추가서류 제출일 전일 기준 [9,361.62원]으로 추산됩니다.
- 최초기준가격결정일 : 2023년 05월 12일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2023년 11월 10일	두 종목 모두 최초기준가격의 85%이상	액면금액 × 107%
2차	2024년 05월 10일	두 종목 모두 최초기준가격의 85%이상	액면금액 × 114%
3차	2024년 11월 12일	두 종목 모두 최초기준가격의 80%이상	액면금액 × 121%
4차	2025년 05월 12일	두 종목 모두 최초기준가격의 80%이상	액면금액 × 128%
5차	2025년 11월 12일	두 종목 모두 최초기준가격의 75%이상	액면금액 × 135%

- 만기평가일 : 2026년 05월 12일
- 최종기준가격 : 최종기준가격 결정일 현재 기초자산의 거래소 종가 3개의 산술평균
- 만기행사가격 : LG화학 보통주 최초기준가격 × 75%, LG전자 보통주 최초기준가격 × 75%
- 하락한계가격 : LG화학 보통주 최초기준가격 × 40%, LG전자 보통주 최초기준가격 × 40%
- 만기상환 조건
  ⓐ 기초자산의 최종기준가격이 모두 최초기준가격의 75% 이상인 경우: 액면금액 × 142%

ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 각각의 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 142%

ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 한 번이라도 각각의 하락한계 가격보다 작게 된 적이 있는 경우: [액면가액 × (Worst 가격변동률 + 1)]을 만기상환금액으로 지급



- 파이썬 코드로 구현

22	import numpy as np from numpy.ma.core import correlate from datetime import date

23	n=10000; r=0.0355 # 시뮬레이션 횟수, 무위험이자율
	x_vol=0.4217; y_vol=0.3509 # 두 지수의 연간 변동성
	n0=date.toordinal(date(2023, 5, 12)) # 최초 기준가격 결정일
	n1=date.toordinal(date(2023, 11, 10)) # 1차 조기 상환일
	n2=date.toordinal(date(2024, 5, 10)) # 2차 조기 상환일
	n3=date.toordinal(date(2024, 11, 12)) # 3차 조기 상환일
	n4=date.toordinal(date(2025, 5, 12)) # 4차 조기 상환일
	n5=date.toordinal(date(2025, 11, 12)) # 5차 조기 상환일
	n6=date.toordinal(date(2026, 5, 12)) # 만기 상환일
	check_day=np.array([n1-n0, n2-n0, n3-n0, n4-n0, n5-n0, n6-n0])
	rho=0.29555; corr=np.array([[1, rho], [rho, 1]]) # 상관계수
	coupon_rate=np.array([0.07, 0.14, 0.21, 0.28, 0.35, 0.42]) # 조기 상환시 쿠폰 이자율
	oneyear=365; tot_date=n6-n0; dt=1/oneyear
	k=np.linalg.cholesky(corr)
	S1=np.zeros([tot_date+1, 1])
	S2=np.zeros([tot_date+1, 1])
	S1[0]=100.0; S2[0]=100.0 # 기초자산 초기값
	ratio_S1=S1[0]; ratio_S2=S2[0]
	strike_price=np.array([0.85, 0.85, 0.80, 0.80, 0.75, 0.75]) # 조기 행사가격
	repay_n=len(strike_price) # 조기상환 횟수
	payment=np.zeros([repay_n, 1])
	payoff=np.zeros([repay_n, 1]) # 페이오프
	tot_payoff=np.zeros([repay_n, 1]) # 전체 페이오프
	discount_payoff=np.zeros([repay_n, 1]) # 페이오프의 현가
	early_count = np.zeros([repay_n, 1]) # 각 조기상환기간 충족횟수
	maturity_count = 0 # 만기상환 발생횟수
	lose_count = 0 # 만기손실 발생횟수
	facevalue=10**4 # 액면금액
	kib=0.4; dummy=0.42 # 낙인 배리어, 더미 이자율
	for j in range(repay_n):
	  payment[j]=facevalue*(1+coupon_rate[j])

24	for i in range(n):
	    w0=np.random.normal(0, 1, size=[tot_date,2]) # 만기상환일 만큼의 난수 생성
	    w0=np.transpose(w0)
	    w=np.matmul(k, w0)
	    for j in range(tot_date):
	        S1[j+1]=S1[j]*np.exp((r-0.5*x_vol**2)*dt+x_vol*np.sqrt(dt)*w[0, j]) #
	        S2[j+1]=S2[j]*np.exp((r-0.5*y_vol**2)*dt+y_vol*np.sqrt(dt)*w[1, j]) #
	    R1=S1/ratio_S1; R2=S2/ratio_S2
	    WP=np.minimum(R1, R2)
	    WP_checkday=WP[check_day]
	    payoff=np.zeros([repay_n, 1]); repay_event=0
	    for j in range(repay_n):
	        if WP_checkday[j] >= strike_price[j]: # 조기상환일에 주가를 체크, 조기 상환여부 결정
	            payoff[j]=payment[j]
	            early_count[j] += 1
	            repay_event=1
	            break
	    if repay_event == 0: # 조기상환 되지 않고 만기까지 온 경우
	       if min(WP) > kib: # 낙인 배리어 아래로 내려간 적이 없는 경우
	            payoff[-1]=facevalue*(1+dummy)
	            maturity_count += 1
	       else: # 낙인 배리어 아래로 내려간 적이 있는 경우
	            WPL=(WP[-1]+WP[-2]+WP[-3])/3
	            payoff[-1]=facevalue*WPL
	            lose_count += 1
	    tot_payoff=tot_payoff + payoff
	mean_payoff=tot_payoff/n
	for j in range(repay_n): # 페이오프를 무위험 이자율로 할인하여 현재 가격을 구함
	    discount_payoff[j]=mean_payoff[j]*np.exp(-r*check_day[j]/oneyear)
	price=np.sum(discount_payoff)
	print(price)

9431.288051116415 (ELS 최종가격)

25	print('총 시뮬레이션 횟수 : %d' % (n)) print('조기상환 발생횟수') for j in range(repay_n): print('%d차 : %d, 발생빈도 : %.2f' % (j+1, early_count[j],

```
        early_count[j]/n*100),"%")

        print('만기상환 발생횟수 : %d, 발생빈도 : %.2f' % (maturity_count, (maturity_count)/n*100), "%")
        print('만기손실 발생횟수 : %d, 발생빈도 : %.2f' % (lose_count, (lose_count/n)*100), "%")
```

총 시뮬레이션 횟수 : 10000
조기상환 발생횟수
1차 : 5220, 발생빈도 : 52.20 %
2차 : 974, 발생빈도 : 9.74 %
3차 : 634, 발생빈도 : 6.34 %
4차 : 281, 발생빈도 : 2.81 %
5차 : 293, 발생빈도 : 2.93 %
6차 : 167, 발생빈도 : 1.67 %
만기상환 발생횟수 : 461, 발생빈도 : 4.61 %
만기손실 발생횟수 : 1970, 발생빈도 : 19.70 %

3) 기초자산이 3개인 ELS

① 3개의 기초자산이 상관계수를 갖는 주가 경로 생성

- 기초자산이 3개인 경우 3개의 주가 움직임이 독립적이지 않고 상관관계를 가짐
- 3개의 주가경로를 무작위로 생성하기 위해 촐레스키 분해(Cholesky decomposition)을 이용하여 상관관계를 갖는 난수 생성이 필요
- 상관관계를 갖는 난수 생성 과정
  ⓐ 표준정규분포를 따르는 난수 Φ_1과 Φ_2, Φ_3을 생성
  ⓑ 기초자산 1과 2, 3의 상관계수(ρ) 행렬 B를 생성

$$B=\begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{12} & 1 & \rho_{23} \\ \rho_{13} & \rho_{23} & 1 \end{pmatrix}$$

  ⓒ 행렬 B를 촐레스키 분해

$$B=LL^{T}=\begin{pmatrix} 1 & 0 & 0 \\ \rho_{12} & \sqrt{1-\rho_{12}^2} & 0 \\ \rho_{13} & \dfrac{\rho_{23}-\rho_{12}\rho_{13}}{\sqrt{1-\rho_{12}^2}} & \sqrt{1-\rho_{13}^2-\dfrac{(\rho_{23}-\rho_{12}\rho_{13})^2}{1-\rho_{12}^2}} \end{pmatrix}\times\begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ 0 & \sqrt{1-\rho_{12}^2} & \dfrac{\rho_{23}-\rho_{12}\rho_{13}}{\sqrt{1-\rho_{12}^2}} \\ 0 & 0 & \sqrt{1-\rho_{13}^2-\dfrac{(\rho_{23}-\rho_{12}\rho_{13})^2}{1-\rho_{12}^2}} \end{pmatrix}$$

  ⓓ 분해된 행렬 L을 이용하여 상관계수가 반영된 난수 Φ_1^*와 Φ_2^*, Φ_3^*으로 전환

$$\begin{pmatrix} \Phi_1^* \\ \Phi_2^* \\ \Phi_3^* \end{pmatrix}=\begin{pmatrix} 1 & 0 & 0 \\ \rho_{12} & \sqrt{1-\rho_{12}^2} & 0 \\ \rho_{13} & \dfrac{\rho_{23}-\rho_{12}\rho_{13}}{\sqrt{1-\rho_{12}^2}} & \sqrt{1-\rho_{13}^2-\dfrac{(\rho_{23}-\rho_{12}\rho_{13})^2}{1-\rho_{12}^2}} \end{pmatrix}\times\begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \end{pmatrix}$$

$$\Phi_1^*=\Phi_1$$
$$\Phi_2^*=\Phi_1\rho_{12}+\Phi_2\sqrt{1-\rho_{12}^2}$$
$$\Phi_3^*=\Phi_1\rho_{13}+\Phi_2\frac{\rho_{23}-\rho_{12}\rho_{13}}{\sqrt{1-\rho_{12}^2}}+\Phi_3\sqrt{1-\rho_{13}^2-\frac{(\rho_{23}-\rho_{12}\rho_{13})^2}{1-\rho_{12}^2}}$$

- 파이썬에서는 np.linalg.cholesky이라는 내장함수를 이용하여 상관관계를 갖는 난수 생성
- 주가 경로식

$$S_1(t+\Delta t)=S_1(t)\exp[(r-\frac{\sigma_1^2}{2})\Delta t+\sigma_1\sqrt{\Delta t}\,\Phi_1^*],\ \Phi_1^*\sim N(0,1)$$

$$S_2(t+\Delta t)=S_2(t)\exp[(r-\frac{\sigma_2^2}{2})\Delta t+\sigma_2\sqrt{\Delta t}\,\Phi_2^*],\ \Phi_2^*\sim N(0,1)$$

$$S_3(t+\Delta t)=S_3(t)\exp[(r-\frac{\sigma_3^2}{2})\Delta t+\sigma_3\sqrt{\Delta t}\,\Phi_3^*],\ \Phi_3^*\sim N(0,1)$$

- 파이썬 코드로 구현

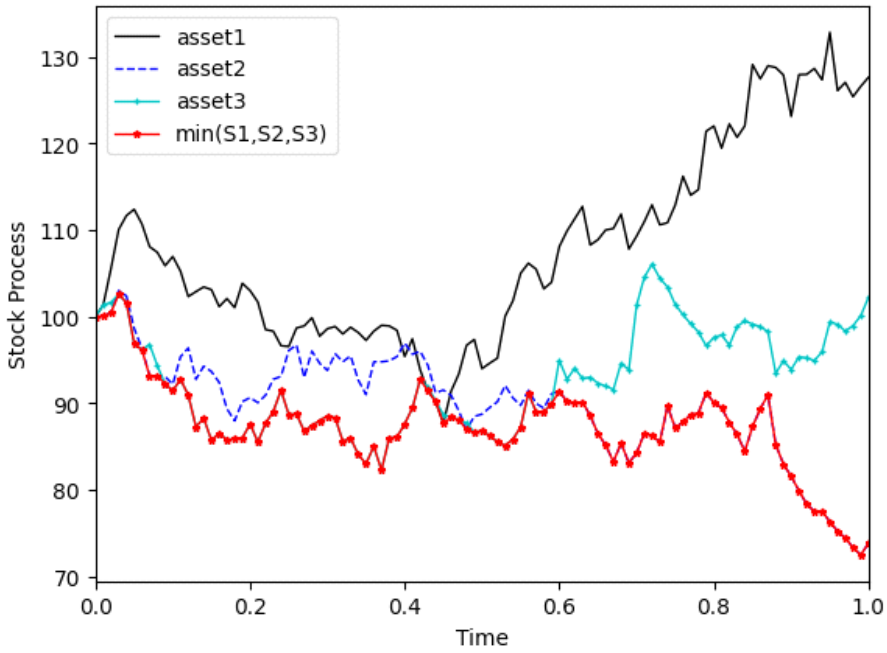
26	import numpy as np import matplotlib.pyplot as plt from numpy.ma.core import correlate plt.close()

27	import numpy as np import matplotlib.pyplot as plt from numpy.ma.core import correlate

28	x_vol=0.2662; y_vol=0.2105; z_vol=0.2111 r=0.035 rho_xy=0.279; rho_xz=0.2895; rho_yz=0.5256 N=100; T=1; dt=T/N S1=np.zeros((N+1, 1)) S2=np.zeros((N+1, 1)) S3=np.zeros((N+1, 1)) S1[0]=100; S2[0]=100; S3[0]=100 t=np.linspace(0, T, N+1) correlation=np.array([[1, rho_xy, rho_xz], [rho_xy, 1, rho_yz], [rho_xz, rho_yz, 1]]) cholesky=np.linalg.cholesky(correlation) z0=np.random.normal(0, 1, size=[N, 3]) np.random.seed(42) z0=np.transpose(z0) z=np.matmul(cholesky, z0) Worst_performer=np.zeros((N+1, 1))

29	plt.xlim(0, 1)
	plt.ylim(60, 170)
	
	for i in range(N):
	  S1[i+1]=S1[i]*np.exp((r-0.5*x_vol**2)*dt+x_vol*z[0,i]*np.sqrt(dt))
	  S2[i+1]=S2[i]*np.exp((r-0.5*y_vol**2)*dt+y_vol*z[1,i]*np.sqrt(dt))
	  S3[i+1]=S3[i]*np.exp((r-0.5*z_vol**2)*dt+z_vol*z[2,i]*np.sqrt(dt))
	  Worst_performer[i]=min(S1[i,0], S2[i,0], S3[i,0])
	  Worst_performer[-1]=min(S1[-1,0], S2[-1,0], S2[-1,0])
	
	plt.plot(t, S1[:], 'k-', label='asset1', linewidth=1, markersize=3.5)
	plt.plot(t, S2[:], 'b--', label='asset2', linewidth=1, markersize=3.5)
	plt.plot(t, S3[:], 'c+-', label='asset3', linewidth=1, markersize=3.5)

```
plt.plot(t, Worst_performer[:], 'r*-', label='min(S1,S2,S3)', linewidth=1, markersize=3.5)

plt.xlabel('Time')
plt.ylabel('Stock Process')
plt.legend(prop={'size':10})
plt.show()
```



② 3개의 기초자산 ELS 가격 계산

Case 1

투자설명서 요약

- 기초자산 : EUROSTOXX50 지수, S&P500 지수, KOSPI200 지수
- 기초자산가격 변동성 : EUROSTOXX50지수 : 24.97%, S&P500지수 : 27.38%, KOSPI200 지수 : 23.22%

  변동성 산출기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005))하여 Vola tility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용
- 기초자산간의 상관계수: EUROSTOXX50, S&P500 : 0.5955 EUROSTOXX50, KOSPI200 : 0.2311, S&P500, KOSPI200 : 0.0489

  상관계수 산출기준 : 180 영업일 역사적 상관계수
- 공정가격 : 본 증권의 공정가격은 2023년 04월 25일 기준 9,389.12원으로 추산됩니다.
- 최초기준가격결정일 : 2023년 05월 15일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2023년 11월 10일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 103.5%
2차	2024년 05월 9일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 107.0%
3차	2024년 11월 12일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 110.5%
4차	2025년 05월 12일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 114.0%
5차	2025년 11월 11일	세 지수 모두 최초기준가격의 80%이상	액면금액 × 117.5%

- 만기평가일 : 2026년 05월 12일
- 최종기준가격 : 만기평가일 각 기초자산 종가 (현지시간 기준)
- 만기행사가격 : 각 기초자산 최초기준가격 × 75%
- 하락한계가격 : 각 기초자산 최초기준가격 × 45%
- 만기상환 조건

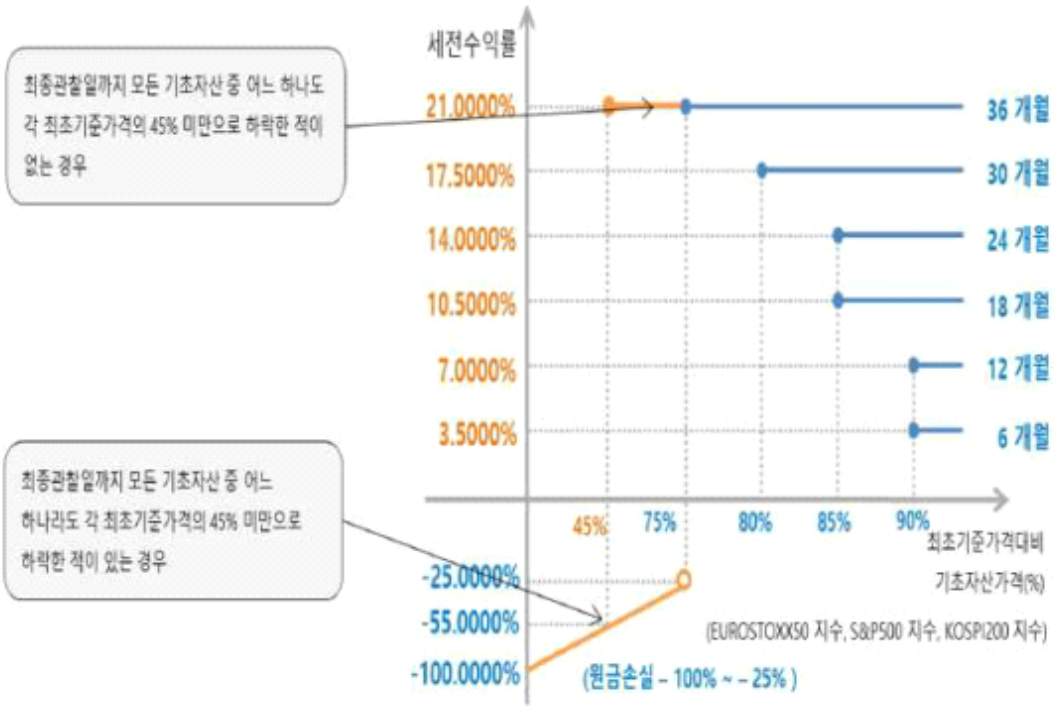
  ⓐ 모든 기초자산의 만기평가가격이 각 최초기준가격의 75% 이상인 경우: 액면금액 × 121.0%

  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 각각의 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 121.0%

  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 한 번이라도 각각의 하락한계 가격보다 작게 된 적이 있는 경우: 기준종목 기준으로 {(만기평가가격/최초기준가격)- 1} × 100%

  ※ 기준종목 : 모든 기초자산 중 [만기평가가격/최초기준가격]의 비율이 가장 낮은 기초자산

- 예상 손익구조 그래프


최종관찰일까지 모든 기초자산 중 어느 하나도 각 최초기준가격의 45% 미만으로 하락한 적이 없는 경우
최종관찰일까지 모든 기초자산 중 어느 하나라도 각 최초기준가격의 45% 미만으로 하락한 적이 있는 경우
세전수익률
21.0000%
17.5000%
14.0000%
10.5000%
7.0000%
3.5000%
-25.0000%
-55.0000%
-100.0000%
36 개월
30 개월
24 개월
18 개월
12 개월
6 개월
45%
75%
80%
85%
90%
최초기준가격대비
기초자산가격(%)
(EUROSTOXX50 지수, S&P500 지수, KOSPI200 지수)
(원금손실 – 100% ~ – 25%)

- 파이썬 코드로 구현

30	import numpy as np from numpy.ma.core import correlate from datetime import date

31	n=10000; r=0.0355
	x_vol=0.2497; y_vol=0.2738; z_vol=0.2322
	n0=date.toordinal(date(2023, 5, 15)) # 최초 기준가격 결정일
	n1=date.toordinal(date(2023, 11, 10)) # 1차 조기 상환일
	n2=date.toordinal(date(2024, 5, 9)) # 2차 조기 상환일
	n3=date.toordinal(date(2024 , 11, 12)) # 3차 조기 상환일
	n4=date.toordinal(date(2025, 5, 12)) # 4차 조기 상환일
	n5=date.toordinal(date(2025, 11, 11)) # 5차 조기 상환일
	n6=date.toordinal(date(2026, 5, 12)) # 만기 상환일
	check_day=np.array([n1-n0, n2-n0, n3-n0, n4-n0, n5-n0, n6-n0])
	rho_xy=0.5955; rho_xz=0.2311; rho_yz=0.0489 # 상관계수
	corr=np.array([[1, rho_xy, rho_xz], [rho_xy, 1, rho_yz], [rho_xz, rho_yz, 1]])
	k=np.linalg.cholesky(corr)
	oneyear=365; tot_date=n6-n0; dt=1/oneyear
	S1=np.zeros((tot_date+1, 1))
	S2=np.zeros((tot_date+1, 1))
	S3=np.zeros((tot_date+1, 1))
	S1[0]=100.0; S2[0]=100.0; S3[0]=100.0
	ratio_S1=S1[0]; ratio_S2=S2[0]; ratio_S3=S3[0]
	strike_price=np.array([0.90, 0.90, 0.85, 0.85, 0.80, 0.75]) # 조기 행사가격
	repay_n=len(strike_price) # 조기상환 횟수
	coupon_rate=np.array([0.035, 0.07, 0.105, 0.14, 0.175, 0.21]) # 조기 상환시 쿠폰 이자율
	tot_payoff=np.zeros([repay_n, 1]) # 전체 페이오프
	payoff=np.zeros([repay_n, 1]) # 페이오프
	discount_payoff=np.zeros([repay_n, 1]) # 페이오프의 현가
	payment=np.zeros([repay_n, 1])
	facevalue=10**4 # 액면금액
	kib=0.45; dummy=0.21 # 낙인 배리어, 더미 이자율
	early_count = np.zeros([repay_n, 1]) # 각 조기상환기간 충족횟수
	maturity_count = 0 # 만기상환 발생횟수
	lose_count = 0 # 만기손실 발생횟수
	for j in range(repay_n):
	  payment[j]=facevalue*(1+coupon_rate[j])

```
for i in range(n):
    w0=np.random.normal(0, 1, size=[tot_date, 3])  # 만기상환일 만큼의 난수 생성
    w0=np.transpose(w0)
    w=np.matmul(k, w0)
    payoff=np.zeros([repay_n, 1]); repay_event=0
    for j in range(tot_date):
        S1[j+1]=S1[j]*np.exp((r-0.5*x_vol**2)*dt+x_vol*np.sqrt(dt)*w[0, j])  #
        S2[j+1]=S2[j]*np.exp((r-0.5*y_vol**2)*dt+y_vol*np.sqrt(dt)*w[1, j])  #
        S3[j+1]=S3[j]*np.exp((r-0.5*z_vol**2)*dt+z_vol*np.sqrt(dt)*w[2, j])  #
    R1=S1/ratio_S1; R2=S2/ratio_S2; R3=S3/ratio_S3
    WP=np.minimum(R1, R2, R3)
    WP_checkday=WP[check_day]

    for j in range(repay_n):
        if WP_checkday[j] >= strike_price[j]:  # 조기상환일에 주가를 체크, 조기 상환여부 결정
32              payoff[j]=payment[j]
                early_count[j] += 1
                repay_event=1
                break
    if repay_event == 0:  # 조기상환 되지 않고 만기까지 온 경우
       if min(WP) > kib:  # 낙인 배리어 아래로 내려간 적이 없는 경우
            payoff[-1]=facevalue*(1+dummy)
            maturity_count += 1
       else:  # 낙인 배리어 아래로 내려간 적이 있는 경우
            payoff[-1]=facevalue*WP[-1]
            lose_count += 1
    tot_payoff=tot_payoff + payoff
mean_payoff=tot_payoff/n
for j in range(repay_n):  # 페이오프를 무위험 이자율로 할인하여 현재 가격을 구함
    discount_payoff[j]=mean_payoff[j]*np.exp(-r*check_day[j]/oneyear)
price=np.sum(discount_payoff)
print(price)
```

9854.861945002594 (ELS 최종가격)

33	print('총 시뮬레이션 횟수 : %d' % (n)) print('조기상환 발생횟수') for j in range(repay_n): print('%d차 : %d, 발생빈도 : %.2f' % (j+1, early_count[j], early_count[j]/n*100),"%") print('만기상환 발생횟수 : %d, 발생빈도 : %.2f' % (maturity_count, (maturity_count)/n*100), "%") print('만기손실 발생횟수 : %d, 발생빈도 : %.2f' % (lose_count, (lose_count/n)*100), "%")

총 시뮬레이션 횟수 : 10000
조기상환 발생횟수
1차 : 5956, 발생빈도 : 59.56 %
2차 : 939, 발생빈도 : 9.39 %
3차 : 685, 발생빈도 : 6.85 %
4차 : 319, 발생빈도 : 3.19 %
5차 : 331, 발생빈도 : 3.31 %
6차 : 259, 발생빈도 : 2.59 %
만기상환 발생횟수 : 766, 발생빈도 : 7.66 %
만기손실 발생횟수 : 745, 발생빈도 : 7.45 %

Case 2

투자설명서 요약

- 기초자산 : EUROSTOXX50 지수, S&P500 지수, HSCEI 지수
- 기초자산가격 변동성 : EUROSTOXX50지수 : 28.1%, S&P500지수 : 33.51%, HSCEI지수 : 27,63%

  변동성 산출기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005))하여 Vola tility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용
- 기초자산간의 상관계수: EUROSTOXX50-S&P500 : 0.5678 EUROSTOXX50-HSCEI지수 : 0.3786, S&P500-HSCEI지수 : 0.2219

  상관계수 산출기준 : 180 영업일 역사적 상관계수
- 공정가격 : 본 증권의 공정가격은 2021년 03월 02일 기준 8,425.94원으로 추산됩니다.
- 최초기준가격결정일 : 2021년 03월 15일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2021년 09월 10일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 102.65%
2차	2022년 03월 10일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 105.30%
3차	2022년 09월 08일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 107.95%
4차	2023년 03월 10일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 110.60%
5차	2023년 09월 12일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 113.25%

- 만기평가일 : 2024년 03월 12일
- 최종기준가격 : 만기평가일 각 기초자산 종가 (현지시간 기준)
- 만기행사가격 : 각 기초자산 최초기준가격 × 75%
- 하락한계가격 : 각 기초자산 최초기준가격 × 47%
- 만기상환 조건

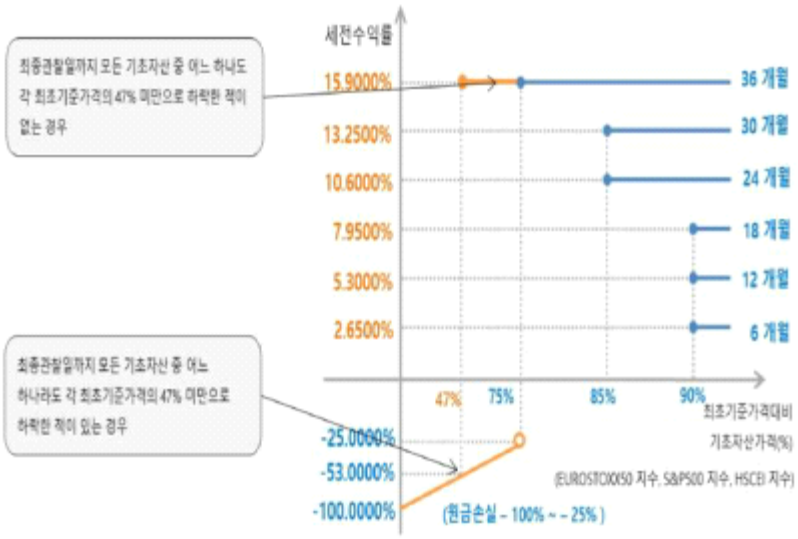
  ⓐ 모든 기초자산의 만기평가가격이 각 최초기준가격의 75% 이상인 경우: 액면금액 × 115.90%(연 5.30%)

  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 각각의 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 115.90%(연 5.30%)

  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 한 번이라도 각각의 하락한계 가격보다 작게 된 적이 있는 경우: 기준종목 기준으로 {(만기평가가격/최초기준가격)- 1} × 100%

  ※ 기준종목 : 모든 기초자산 중 [만기평가가격/최초기준가격]의 비율이 가장 낮은 기초자산

- 예상 손익구조 그래프



미래에셋대우 제29534회 (예상 손익구조 그래프)

- 파이썬 코드로 구현

34	import numpy as np from numpy.ma.core import correlate from datetime import date

35	n=10000; r=0.012
	x_vol=0.281; y_vol=0.3351; z_vol=0.2763
	n0=date.toordinal(date(2021, 3, 15)) # 최초 기준가격 결정일
	n1=date.toordinal(date(2021, 9, 10)) # 1차 조기 상환일
	n2=date.toordinal(date(2022, 3, 10)) # 2차 조기 상환일
	n3=date.toordinal(date(2022, 9, 8)) # 3차 조기 상환일
	n4=date.toordinal(date(2023 , 3, 10)) # 4차 조기 상환일
	n5=date.toordinal(date(2023, 9, 12)) # 5차 조기 상환일
	n6=date.toordinal(date(2024, 3, 12)) # 만기 상환일
	check_day=np.array([n1-n0, n2-n0, n3-n0, n4-n0, n5-n0, n6-n0])
	rho_xy=0.5678; rho_xz=0.3786; rho_yz=0.2219 # 상관계수
	corr=np.array([[1, rho_xy, rho_xz], [rho_xy, 1, rho_yz], [rho_xz, rho_yz, 1]])
	k=np.linalg.cholesky(corr)
	oneyear=365; tot_date=n6-n0; dt=1/oneyear
	S1=np.zeros((tot_date+1, 1))
	S2=np.zeros((tot_date+1, 1))
	S3=np.zeros((tot_date+1, 1))

```
S1[0]=100.0; S2[0]=100.0; S3[0]=100.0
ratio_S1=S1[0]; ratio_S2=S2[0]; ratio_S3=S3[0]
strike_price=np.array([0.90, 0.90, 0.90, 0.85, 0.85, 0.75])  # 조기 행사가격
repay_n=len(strike_price) # 조기상환 횟수
coupon_rate=np.array([0.0265, 0.053, 0.0795, 0.106, 0.1325, 0.159])  # 조기상환시 쿠폰 이자율
tot_payoff=np.zeros([repay_n, 1])  # 전체 페이오프
payoff=np.zeros([repay_n, 1])  # 페이오프
discount_payoff=np.zeros([repay_n, 1])  # 페이오프의 현가
payment=np.zeros([repay_n, 1])
facevalue=10**4  # 액면금액
kib=0.47; dummy=0.159  # 낙인 배리어, 더미 이자율
early_count = np.zeros([repay_n, 1]) # 각 조기상환기간 충족횟수
maturity_count = 0 # 만기상환 발생횟수
lose_count = 0 # 만기손실 발생횟수
for j in range(repay_n):
    payment[j]=facevalue*(1+coupon_rate[j])
```

36

```
for i in range(n):
    w0=np.random.normal(0, 1, size=[tot_date, 3])  # 만기상환일 만큼의 난수 생성
    w0=np.transpose(w0)
    w=np.matmul(k, w0)
    payoff=np.zeros([repay_n, 1]); repay_event=0
    for j in range(tot_date):
        S1[j+1]=S1[j]*np.exp((r-0.5*x_vol**2)*dt+x_vol*np.sqrt(dt)*w[0, j])  #
        S2[j+1]=S2[j]*np.exp((r-0.5*y_vol**2)*dt+y_vol*np.sqrt(dt)*w[1, j])  #
        S3[j+1]=S3[j]*np.exp((r-0.5*z_vol**2)*dt+z_vol*np.sqrt(dt)*w[2, j])  #
    R1=S1/ratio_S1; R2=S2/ratio_S2; R3=S3/ratio_S3
    WP=np.minimum(R1, R2, R3)
    WP_checkday=WP[check_day]

    for j in range(repay_n):
        if WP_checkday[j] >= strike_price[j]:  # 조기상환일에 주가를 체크, 조기 상환여부 결정
            payoff[j]=payment[j]
            early_count[j] += 1
            repay_event=1
            break
    if repay_event == 0:  # 조기상환 되지 않고 만기까지 온 경우
```

```
        if min(WP) > kib:  # 낙인 배리어 아래로 내려간 적이 없는 경우
            payoff[-1]=facevalue*(1+dummy)
            maturity_count += 1
        else:  # 낙인 배리어 아래로 내려간 적이 있는 경우
            payoff[-1]=facevalue*WP[-1]
            lose_count += 1
    tot_payoff=tot_payoff + payoff
mean_payoff=tot_payoff/n
for j in range(repay_n):  # 페이오프를 무위험 이자율로 할인하여 현재 가격을 구함
    discount_payoff[j]=mean_payoff[j]*np.exp(-r*check_day[j]/oneyear)
price=np.sum(discount_payoff)
print(price)
```

9471.05712719215 (ELS 최종가격)

37

```
print('총 시뮬레이션 횟수 : %d' % (n))
print('조기상환 발생횟수')
for j in range(repay_n):
    print('%d차 : %d, 발생빈도 : %.2f' % (j+1, early_count[j], early_count[j]/n*100),"%")

print('만기상환 발생횟수 : %d, 발생빈도 : %.2f' % (maturity_count, (maturity_count)/n*100), "%")
print('만기손실 발생횟수 : %d, 발생빈도 : %.2f' % (lose_count, (lose_count/n)*100), "%")
```

총 시뮬레이션 횟수 : 10000
조기상환 발생횟수
1차 : 5787, 발생빈도 : 57.87 %
2차 : 968, 발생빈도 : 9.68 %
3차 : 411, 발생빈도 : 4.11 %
4차 : 392, 발생빈도 : 3.92 %
5차 : 187, 발생빈도 : 1.87 %
6차 : 278, 발생빈도 : 2.78 %
만기상환 발생횟수 : 474, 발생빈도 : 4.74 %
만기손실 발생횟수 : 1503, 발생빈도 : 15.03 %

Case 3

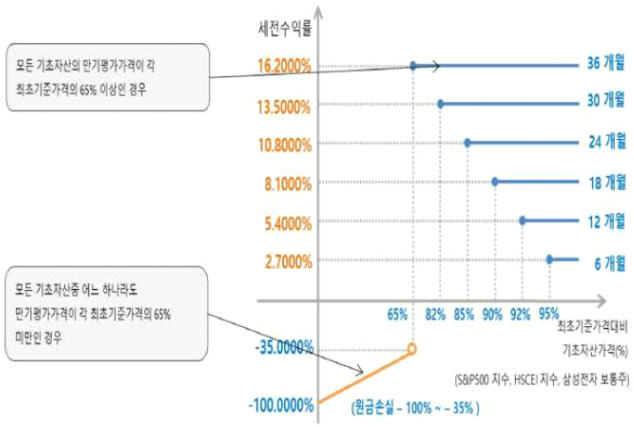
투자설명서 요약

- 기초자산 : S&P500 지수, HSCEI 지수, 삼성전자 보통주
- 기초자산가격 변동성 : S&P500지수 : 30,05%, HSCEI지수 : 27,33%, 삼성전자 : 31.54%
  변동성 산출기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005))하여 Vola tility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용
- 기초자산간의 상관계수: S&P500-HSCEI지수 : 0.1952, S&P500-삼성전자 : 0.101, HSCEI지수-삼성전자 : 0.4479
  상관계수 산출기준 : 180 영업일 역사적 상관계수
- 공정가격 : 본 증권의 공정가격은 2021년 04월 28일 기준 7,814.84원으로 추산됩니다.
- 최초기준가격결정일 : 2021년 05월 07일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2021년 11월 02일	세 지수 모두 최초기준가격의 95%이상	액면금액 × 102.70%
2차	2022년 04월 29일	세 지수 모두 최초기준가격의 92%이상	액면금액 × 105.40%
3차	2022년 11월 02일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 108.10%
4차	2023년 04월 28일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 110.80%
5차	2023년 11월 02일	세 지수 모두 최초기준가격의 82%이상	액면금액 × 113.50%

- 만기평가일 : 2024년 04월 30일
- 최종기준가격 : 만기평가일 각 기초자산 종가 (현지시간 기준)
- 만기행사가격 : 각 기초자산 최초기준가격 × 65%
- 만기상환 조건
  ⓐ 모든 기초자산의 만기평가가격이 각 최초기준가격의 65% 이상인 경우: 액면금액 × 116.20%(연 5.30%)
  ⓑ 모든 기초자산 중 어느 하나라도 만기평가가격이 각 최초 기준가격의 65% 미만인 경우: 기준종목 기준으로 {(만기평가가격/최초기준가격)- 1} × 100%
  ※ 기준종목 : 모든 기초자산 중 [만기평가가격/최초기준가격]의 비율이 가장 낮은 기초자산

- 예상 손익구조 그래프



미래에셋증권 제29629회 (예상 손익구조 그래프)

- 파이썬 코드로 구현

```
38
import numpy as np
from numpy.ma.core import correlate
from datetime import date
```

```
39
n=10000; r=0.012
x_vol=0.3005; y_vol=0.27331; z_vol=0.3154
n0=date.toordinal(date(2021, 5, 7))  # 최초 기준가격 결정일
n1=date.toordinal(date(2021, 11, 2))  # 1차 조기 상환일
n2=date.toordinal(date(2022, 4, 29))  # 2차 조기 상환일
n3=date.toordinal(date(2022, 11, 2))  # 3차 조기 상환일
n4=date.toordinal(date(2023 , 4, 28))  # 4차 조기 상환일
n5=date.toordinal(date(2023, 11, 2))  # 5차 조기 상환일
n6=date.toordinal(date(2024, 4, 30))  # 만기 상환일
check_day=np.array([n1-n0, n2-n0, n3-n0, n4-n0, n5-n0, n6-n0])
rho_xy=0.1952; rho_xz=0.101; rho_yz=0.4479  # 상관계수
corr=np.array([[1, rho_xy, rho_xz], [rho_xy, 1, rho_yz], [rho_xz, rho_yz, 1]])
k=np.linalg.cholesky(corr)
oneyear=365; tot_date=n6-n0; dt=1/oneyear
S1=np.zeros((tot_date+1, 1))
S2=np.zeros((tot_date+1, 1))
S3=np.zeros((tot_date+1, 1))
S1[0]=100.0; S2[0]=100.0; S3[0]=100.0
ratio_S1=S1[0]; ratio_S2=S2[0]; ratio_S3=S3[0]
strike_price=np.array([0.95, 0.92, 0.90, 0.85, 0.82, 0.65])  # 조기 행사가격
repay_n=len(strike_price) # 조기상환 횟수
coupon_rate=np.array([0.027, 0.054, 0.081, 0.108, 0.135, 0.162])  # 조기 상환시 쿠폰 이자율
tot_payoff=np.zeros([repay_n, 1])  # 전체 페이오프
payoff=np.zeros([repay_n, 1])  # 페이오프
discount_payoff=np.zeros([repay_n, 1])  # 페이오프의 현가
payment=np.zeros([repay_n, 1])
facevalue=10**4  # 액면금액
kib=0.0; dummy=0.162  # 낙인 배리어, 더미 이자율
early_count = np.zeros([repay_n, 1]) # 각 조기상환기간 충족횟수
maturity_count = 0 # 만기상환 발생횟수
lose_count = 0 # 만기손실 발생횟수
for j in range(repay_n):
    payment[j]=facevalue*(1+coupon_rate[j])
```

```
for i in range(n):
    w0=np.random.normal(0, 1, size=[tot_date, 3])  # 만기상환일 만큼의 난수 생성
    w0=np.transpose(w0)
    w=np.matmul(k, w0)
    payoff=np.zeros([repay_n, 1]); repay_event=0
    for j in range(tot_date):
        S1[j+1]=S1[j]*np.exp((r-0.5*x_vol**2)*dt+x_vol*np.sqrt(dt)*w[0, j])  #
        S2[j+1]=S2[j]*np.exp((r-0.5*y_vol**2)*dt+y_vol*np.sqrt(dt)*w[1, j])  #
        S3[j+1]=S3[j]*np.exp((r-0.5*z_vol**2)*dt+z_vol*np.sqrt(dt)*w[2, j])  #
    R1=S1/ratio_S1; R2=S2/ratio_S2; R3=S3/ratio_S3
    WP=np.minimum(R1, R2, R3)
    WP_checkday=WP[check_day]

    for j in range(repay_n):
        if WP_checkday[j] >= strike_price[j]:  # 조기상환일에 주가를 체크, 조기 상환여부 결정
40              payoff[j]=payment[j]
                early_count[j] += 1
                repay_event=1
                break
    if repay_event == 0:  # 조기상환 되지 않고 만기까지 온 경우
       if WP[-1] >= 0.65:  # 기준종목이 65% 이상 경우
            payoff[-1]=facevalue*(1+dummy)
            maturity_count += 1
       else:  # 기준종목이 65% 미만 경우
            payoff[-1]=facevalue*WP[-1]
            lose_count += 1
    tot_payoff=tot_payoff + payoff
mean_payoff=tot_payoff/n
for j in range(repay_n):  # 페이오프를 무위험 이자율로 할인하여 현재 가격을 구함
    discount_payoff[j]=mean_payoff[j]*np.exp(-r*check_day[j]/oneyear)
price=np.sum(discount_payoff)
print(price)
```

8893.111178336816 (ELS 최종가격)

41	print('총 시뮬레이션 횟수 : %d' % (n)) print('조기상환 발생횟수') for j in range(repay_n): print('%d차 : %d, 발생빈도 : %.2f' % (j+1, early_count[j], early_count[j]/n*100),"%") print('만기상환 발생횟수 : %d, 발생빈도 : %.2f' % (maturity_count, (maturity_count)/n*100), "%") print('만기손실 발생횟수 : %d, 발생빈도 : %.2f' % (lose_count, (lose_count/n)*100), "%")

총 시뮬레이션 횟수 : 10000
조기상환 발생횟수
1차 : 3559, 발생빈도 : 35.59 %
2차 : 1287, 발생빈도 : 12.87 %
3차 : 678, 발생빈도 : 6.78 %
4차 : 592, 발생빈도 : 5.92 %
5차 : 363, 발생빈도 : 3.63 %
6차 : 862, 발생빈도 : 8.62 %
만기상환 발생횟수 : 0, 발생빈도 : 0.00 %
만기손실 발생횟수 : 2659, 발생빈도 : 26.59 %

Ⅲ. 유한차분법(Finite Difference Method)

1. 유한차분법 소개

- 유한차분법(Finite Difference Method, FDM)은 공간을 작은 격자망으로 구성하여 미분방정식(differential equation)의 미분을 '차분방정식(difference equation)'으로 근사하여 해를 구하는 수치계산법으로 유한차분법에서는 Taylor 정리를 이용하여 미분을 근사한다.
- 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):

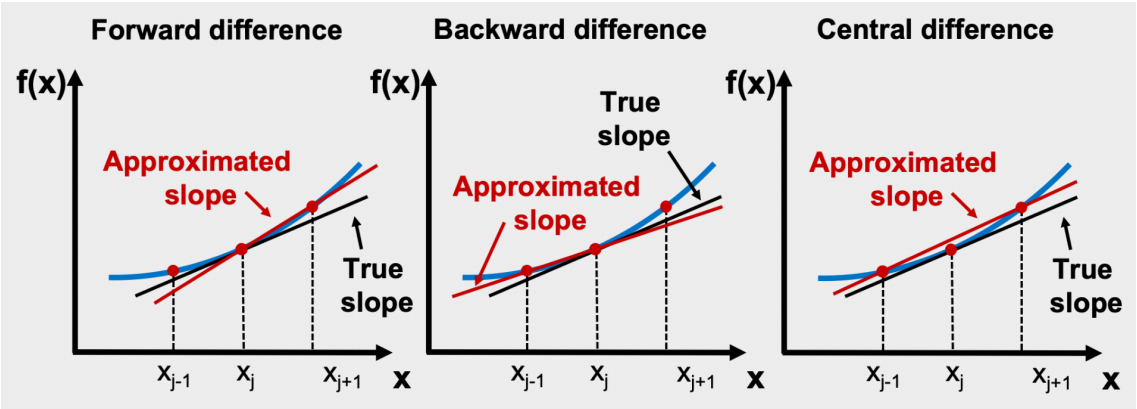
$f'(x)\approx\{f(x+\Delta x)-f(x)\}/\Delta x$

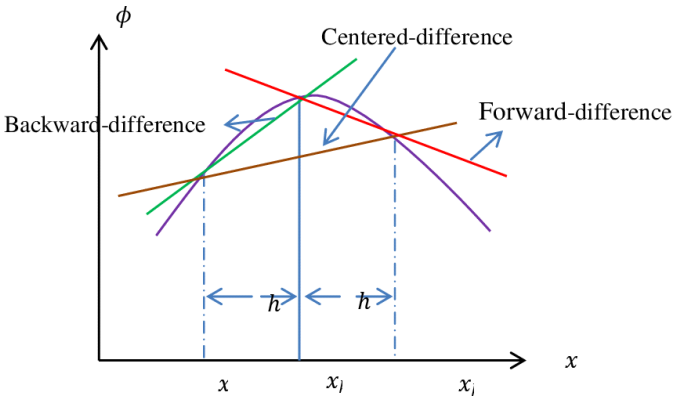
② 역방향 차분(Backward Difference):

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

③ 중앙 차분(Central Difference):

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





- 이러한 차분을 이용하여 주어진 미분방정식이나 경계값 문제를 유한차분법으로 근사화하고, 계산을 통해 수치적인 해를 구할 수 있다. 유한차분법은 특히 컴퓨터를 활용한 수치해석에서 많이 사용되며, 물리학, 공학, 금융 등 다양한 분야에서 응용된다.

유한차분법으로 파생상품 가치평가 Flow Chart

구분	단계별 task	세부내용
1단계	ELS 발행조건 확인	- 기초자산 종류 및 개수, 변동성, 상관계수 점검 - ELS 상환조건, 일자 확인 - 쿠폰 이자율, 만기 상환율 확인 - Knock-in 조건 확인
2단계	Black-Sholes 편미분 방정식(PDE) 도출	- 기초자산이 1개인 경우 편미분 방정식 - 기초자산이 2개인 경우 편미분 방정식 - 기초자산이 3개인 경우 편미분 방정식
3단계	격자 구성	- 주가 노드, 시간 노드 정의 - $u_{i,j}$ 정의: (i,j)점에서의 옵션 가치 - 경계조건(boundary condition) 정의 및 설정 - 균일격자, 비균일격자 설정
4단계	PDE 편도함수들의 차분근사식 도출	- Explicit Finite Difference Method - Implicit Finite Difference Method - Crank-Nicolson Finite Difference Method : Explicit과 Implicit의 평균을 이용 - Operator Splitting method(OSM)
5단계	차분근사식을 원래 PDE에 대입하여 연립방정식 도출	- Explicit Finite Difference Method - Implicit Finite Difference Method - Crank-Nicolson Finite Difference Method
6단계	경계조건 이용 연립방정식의 해 도출	- Implicit/Crank-Nicolson Finite Difference Method의 경우 연립차분방정식 해를 도출하기 위해 LU Decomposition 필요
7단계	현재 시점까지 반복절차(역행) 통한 옵션가치 도출	- linear Interpolation을 통한 현재 가격에 대응하는 옵션가치 도출

- 기초자산이 1개인 경우 편미분 방정식

$$\frac{\partial u}{\partial t} = \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 u}{\partial x^2} + rx\frac{\partial u}{\partial x} - ru$$

- 기초자산이 2개인 경우 편미분 방정식

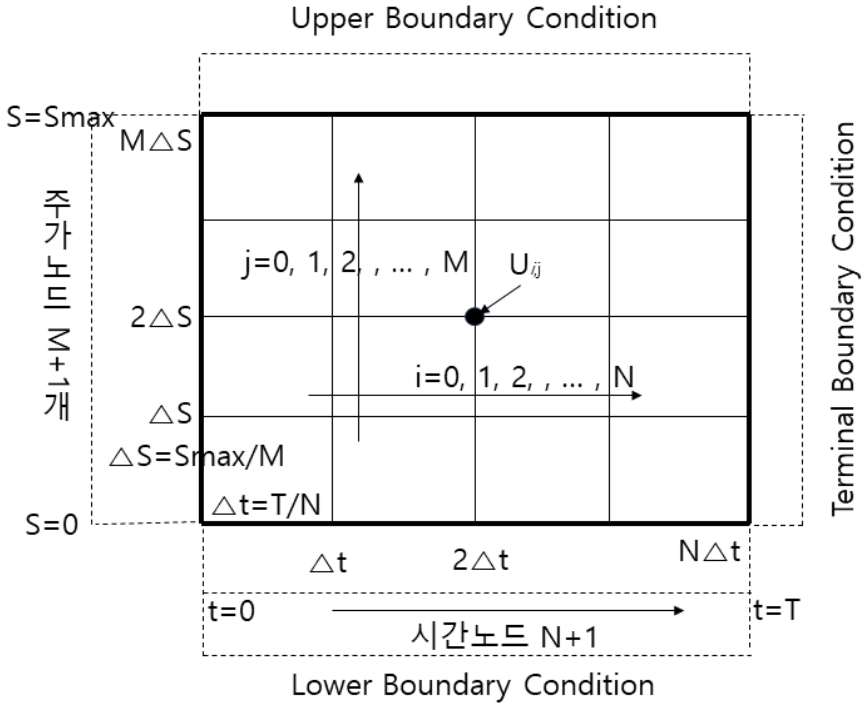
$$\frac{\partial u}{\partial t} = \frac{1}{2}\sigma_x^2 x^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\sigma_y^2 y^2 \frac{\partial^2 u}{\partial y^2} + rx\frac{\partial u}{\partial x} + ry\frac{\partial u}{\partial y} + \rho_{xy}\sigma_x\sigma_y xy\frac{\partial^2 u}{\partial x \partial y} - ru$$

- 기초자산이 3개인 경우 편미분 방정식

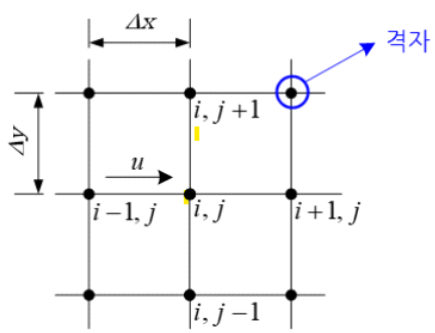
$$\frac{\partial u}{\partial t} = \frac{1}{2}\sigma_x^2 x^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\sigma_y^2 y^2 \frac{\partial^2 u}{\partial y^2} + \frac{1}{2}\sigma_z^2 z^2 \frac{\partial^2 u}{\partial z^2} + rx\frac{\partial u}{\partial x} + ry\frac{\partial u}{\partial y} + rz\frac{\partial u}{\partial z}$$
$$+ \rho_{xy}\sigma_x\sigma_y xy\frac{\partial^2 u}{\partial x \partial y} + \rho_{yz}\sigma_y\sigma_z yz\frac{\partial^2 u}{\partial y \partial z} + \rho_{xz}\sigma_x\sigma_z xz\frac{\partial^2 u}{\partial x \partial z} - ru$$

- 유한차분법은 기하학적 영역 내에 유한개의 점들을 생성하고 서로 이웃하는 점들 사이에서 자연현상의 위치에 따른 변화를 이용하여 미분방정식을 행렬방정식으로 전환시킨다. 기하학적 영역 안에 생성된 유한개의 점들을 격자(grid)라고 부르고 격자의 조밀도에 따라 근사해의 정확도는 증가한다.

(M+1)X(N+1) 격자 구성



이경수, 권영은, 신진호(2008). 파생상품 Modeling I: Matlab 활용. 서울: 도서출판 아진.

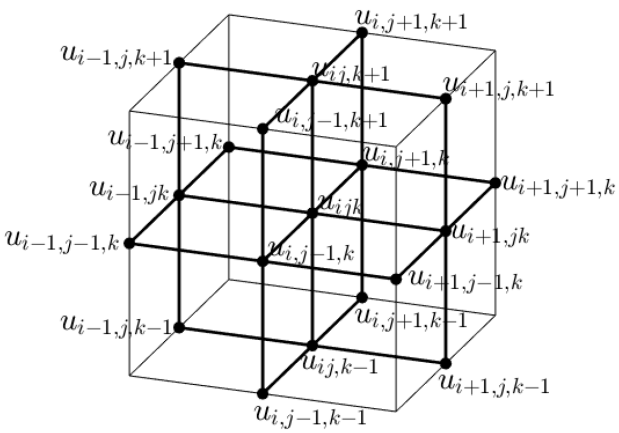


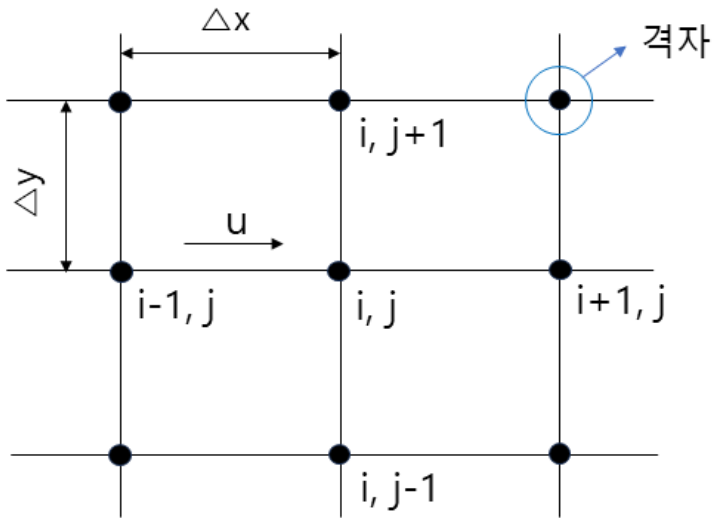
미분식 ⇒ 차분식

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x}$$

$$\frac{\partial u}{\partial y} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y}$$

- 3중 대각 행렬 알고리즘(tridiagonal matrix algorithm)은 토마스 알고리즘(Thomas algorithm)으로 알려져 있으며 가우스 소거법의 단순화된 형태다.





$$\begin{pmatrix} b_1 & c_1 & 0 & \cdot & 0 \\ a_2 & b_2 & c_2 & \cdot & 0 \\ 0 & a_3 & b_3 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & c_{n-1} \\ 0 & 0 & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \cdot \\ d_n \end{pmatrix}$$

- 다음은 토마스 알고리즘을 구현하는 파이썬 코드다.

01	import numpy as np
	
	def thomas(alpha, beta, gamma, f):
	  n=len(f)
	  v=np.zeros(n)
	  [aa, dd, cc, bb]=map(np.array,[alpha, beta, gamma, f])
	  for i in range(1, n):
	  mult=aa[i]/dd[i-1]

```
        dd[i]=dd[i]-mult*cc[i-1]
        bb[i]=bb[i]-mult*bb[i-1]
    v[n-1]=bb[n-1]/dd[n-1]
    for i in range(n-2, -1, -1):
        v[i]=(bb[i]-cc[i]*v[i+1])/dd[i]
    return v
```

- 다음은 토마스 알고리즘을 검증하는 파이썬 코드다.

```
# 토마스 알고리즘 검증
import numpy as np
import matplotlib.pyplot as plt

n=30; x=np.linspace(0, n-1, n)
a=np.random.rand(n)
d=100+np.random.rand(n)
c=np.random.rand(n)
A=np.zeros((n, n))

for i in range(1, n):
    A[i, i-1]=a[i]
for i in range(0, n):
    A[i, i]=d[i]
for i in range(0, n-1):
    A[i, i+1]=c[i]
x1=np.random.rand(n)
b=np.dot(A,x1)
x2=thomas(a,d,c,b)
plt.scatter(x, x1, s=65, facecolors='none', edgecolors='k', label='x1')
plt.scatter(x, x2, color='k', marker='x', label='x2')
plt.legend()
plt.show()
```

2. FDM을 이용한 옵션 가격 결정

1) 기초자산이 1개인 콜옵션 가격 결정

- 기초자산이 1개인 경우 편미분 방정식

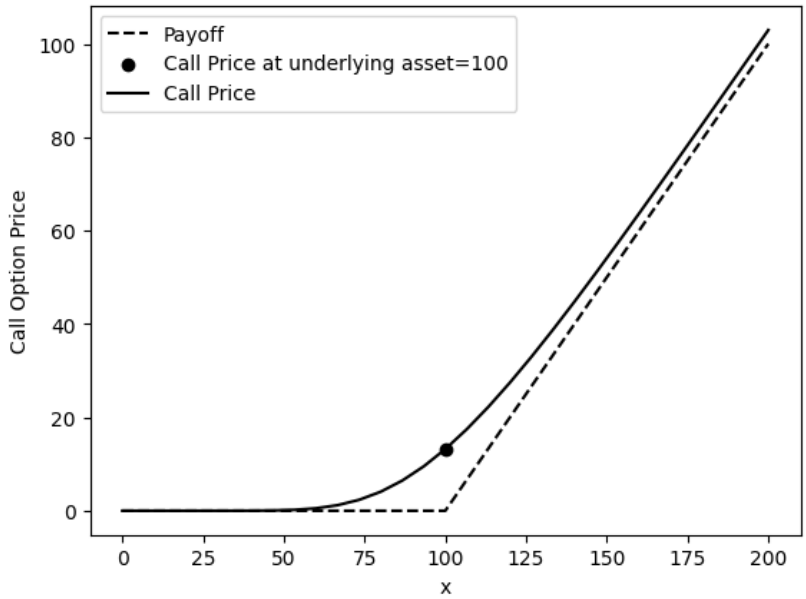
$$\frac{\partial u}{\partial t} = \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 u}{\partial x^2} + rx\frac{\partial u}{\partial x} - ru$$

FDM으로 블랙-숄즈 방정식의 해를 구하는 파이썬 코드

① 균일 격자

```
03
import numpy as np
import matplotlib.pyplot as plt
K=100; R=200; volatility=0.3
r=0.03; T=1
Nx=31; Nt=360*T; dt=T/Nt
x=np.linspace(0, R, Nx)
h=x[1]-x[0]
u=np.zeros((Nx, Nt+1))

for i in range (0,Nx):
    u[i,0]=np.maximum(x[i]-K,0)
plt.plot(x,u[:,0],'k--',label='Payoff')
[a,d,c,b]=map(np.zeros, [Nx, Nx, Nx, Nx])
for i in range (0,Nx):
    a[i]=r*x[i]/(2*h)-(volatility*x[i])**2/(2*h**2)
    d[i]=(1/dt)+(volatility*x[i])**2/(h**2)+r
    c[i]=-r*x[i]/(2*h)-(volatility*x[i])**2/(2*h**2)
a[Nx-1]=a[Nx-1]-c[Nx-1]
d[Nx-1]=d[Nx-1]+2*c[Nx-1]

for n in range (0,Nt):
    b=u[:,n]/dt
    u[:,n+1]=thomas(a,d,c,b)
ii=np.where(x==100)

plt.scatter(x[ii], u[ii,Nt], color='k', label='Call Price at underlying
asset=100')
plt.plot(x,u[:,Nt], 'k-', label='Call Price')
plt.xlabel('x', fontsize=10)
plt.ylabel('Call Option Price', fontsize=10)
plt.legend()
plt.show()
print('Price=%f'%(u[ii,Nt]))
```

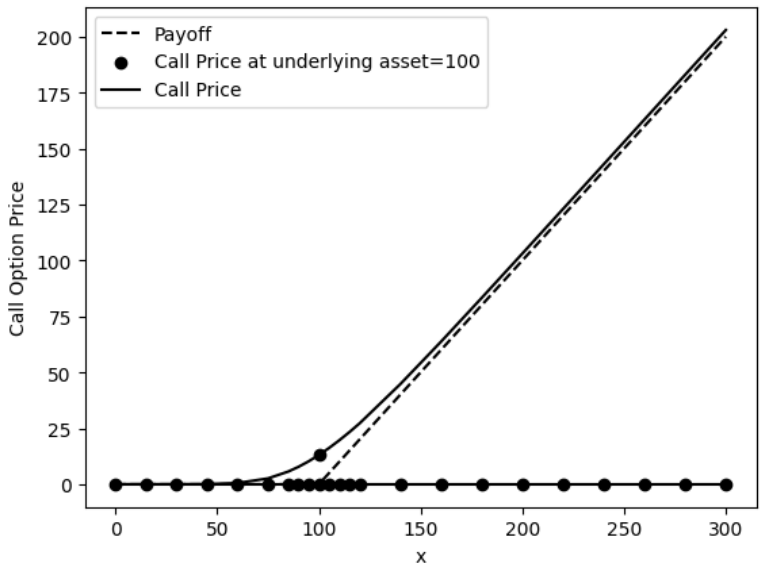
Price=13.207513



② 비균일 격자

04	import numpy as np import matplotlib.pyplot as plt K=100; volatility=0.3 r=0.03; T=1 Nt=360*T; dt=T/Nt A=np.arange(0,80,15) B=np.arange(85,125,5) C=np.arange(140,320,20) x=np.r_(A,B,C) h=np.diff(x); h=np.r_[h[0], h, h[-1]] Nx=len(x); u=np.zeros((Nx, Nt+1)) u[:,0]=np.maximum(x-K,0) plt.plot(x,u[:,0],'k--',label='Payoff') [a,d,c,b]=map(np.zeros, [Nx-1, Nx-1, Nx-1, Nx-1]) a[:]=(-volatility**2*x[1:Nx]**2+r*x[1:Nx]*h[2:Nx+1])/(h[1:Nx]*(h[1:Nx]+h[2:Nx+1])) d[:]=(volatility**2*x[1:Nx])**2-r*x[1:Nx]*(h[2:Nx+1]-h[1:Nx]))/(h[1:Nx]*h[2:Nx+1])+r+1/dt c[:]=(-volatility**2*x[1:Nx]**2-r*x[1:Nx]*h[1:Nx])/(h[2:Nx+1]*(h[1:Nx]+hx[2:Nx+1])) a[Nx-2]=a[Nx-2]-c[Nx-2]

```
d[Nx-2]=d[Nx-2]+2*c[Nx-2]

for n in range (0,Nt):
    b=u[1:Nx, n]/dt
    u[1:Nx,n+1]=thomas(a,d,c,b)
ii=np.where(x==100)

plt.scatter(x[ii], u[ii,Nt], color='k', label='Call Price at underlying asset=100')
plt.plot(x,u[:,Nt], 'k-', label='Call Price')
plt.plot(x,0*u[:,Nt], 'ko-')
plt.xlabel('x', fontsize=10)
plt.ylabel('Call Option Price', fontsize=10)
plt.legend()
plt.show()
print('Price=%f'%(u[ii,Nt]))
```



Price=13.069568

2) 기초자산이 2개인 콜옵션 가격 결정

- 기초자산이 2개인 경우 편미분 방정식

$$\frac{\partial u}{\partial t} = \frac{1}{2}\sigma_x^2 x^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\sigma_y^2 y^2 \frac{\partial^2 u}{\partial y^2} + rx\frac{\partial u}{\partial x} + ry\frac{\partial u}{\partial y} + \rho_{xy}\sigma_x\sigma_y xy\frac{\partial^2 u}{\partial x \partial y} - ru$$

Operator Splitting method(OSM), 연산자 분해 방법은 n차원의 문제를 n개의 1차원으로 분해하여 계산하는 방법

05

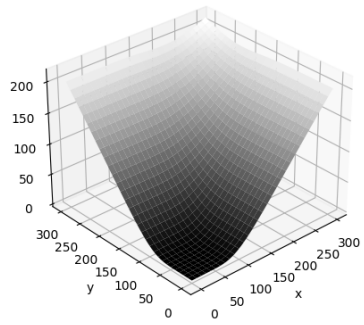
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

R=300; Kx=100; Ky=100; Nx=61; Ny=61;
x=np.linspace(0, R, Nx); y=np.linspace(0, R, Ny)
x_volatility=0.3; y_volatility=0.3; rho=0.3; r=0.03; h=x[1]-x[0];
dt=1/365; T=1; Nt=T/dt
u0=np.zeros((Nx, Ny))
for i in range(Nx):
    for j in range(Ny):
        u0[i,j]=np.maximum(np.maximum(x[i]-Kx,y[j]-Ky),0)
```

06

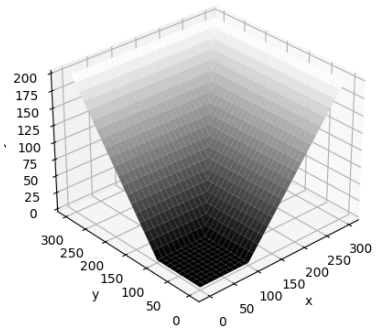
```
X, Y = np.meshgrid(x,y)
fig1=plt.figure()
ax=fig1.add_subplot(projection='3d')
ax.plot_surface(X,Y,u0[:,:],cmap=plt.cm.gray)
ax.view_init(elev=30., azim=-132)
ax.set_xlabel('x', fontsize=10)
ax.set_ylabel('y', fontsize=10)
ax.zaxis.set_rotate_label(False)
ax.set_zlabel('Payoff', rotation=90, fontsize=10)
plt.show()

[ax,dx,cx,ay,dy,cy]=map(np.zeros, [Nx-2, Nx-2, Nx-2, Nx-2, Nx-2, Nx-2])
ax[:]=-(x_volatility*x[1:Nx-1])**2/(2*h**2)+r*x[1:Nx-1]/(2*h)
dx[:]=1/dt+(x_volatility*x[1:Nx-1]/h)**2+r/2
cx[:]=-(x_volatility*x[1:Nx-1])**2/(2*h**2)-r*x[1:Nx-1]/(2*h)
ay[:]=-(y_volatility*y[1:Ny-1])**2/(2*h**2)+r*y[1:Ny-1]/(2*h)
dy[:]=1/dt+(y_volatility*y[1:Ny-1]/h)**2+r/2
cy[:]=-(y_volatility*y[1:Ny-1])**2/(2*h**2)-r*y[1:Ny-1]/(2*h)
u=u0; u2=u0;
[fx,fy]=map(np.zeros, [Nx-2, Ny-2])

for n in range (int(Nt)):
    u[0,0:Ny-1]=2*u[1,0:Ny-1]-u[2,0:Ny-1]
    u[0:Nx-1,0]=2*u[0:Nx-1,1]-u[0:Nx-1,2]
    u[Nx-1,2:Ny-2]=2*u[Nx-2,2:Ny-2]-u[Nx-3,2:Ny-2]
    u[2:Nx-2,Ny-1]=2*u[2:Nx-2,Ny-2]-u[2:Nx-2,Ny-3]
    u[1,Ny-1]=2*u[2,Ny-2]-u[3,Ny-3]
```

```
        u[0,Ny-2]=2*u[1,Ny-3]-u[2,Ny-4]
        u[0,Ny-1]=2*u[1,Ny-2]-u[2,Ny-3]
        u[Nx-1,1]=2*u[Nx-2,2]-u[Nx-3,3]
        u[Nx-2,0]=2*u[Nx-3,1]-u[Nx-4,2]
        u[Nx-1,0]=2*u[Nx-2,1]-u[Nx-3,2]
        u[Nx-1,Ny-1]=2*u[Nx-2,Ny-2]-u[Nx-3,Ny-3]
        u[Nx-2,Ny-1]=2*u[Nx-3,Ny-2]-u[Nx-4,Ny-3]
        u[Nx-1,Ny-2]=2*u[Nx-2,Ny-3]-u[Nx-3,Ny-4]
        # x축으로 풀기
        for j in range(1,Ny-1):
            fx[:]=u[1:Nx-1,j]/dt+0.5*rho*x_volatility*y_volatility\
              *x[1:Nx-1]*y[j]*(u[2:Nx,j+1]+u[0:Nx-2,j-1]-u[0:Nx-2,j+1]\
              -u[2:Nx,j-1])/(4*h**2)
            fx[0]=fx[0]-ax[0]*u[0,j]
            fx[Nx-3]=fx[Nx-3]-cx[Nx-3]*u[-1,j]
            u2[1:Nx-1, j]=thomas(ax,dx,cx,fx)
        # hybrid 경계조건
        u2[0,0:Ny-1]=2*u2[1,0:Ny-1]-u2[2,0:Ny-1]
        u2[0:Nx-1,0]=2*u2[0:Nx-1,1]-u2[0:Nx-1,2]
        u2[Nx-1,2:Ny-2]=2*u2[Nx-2,2:Ny-2]-u2[Nx-3,2:Ny-2]
        u2[2:Nx-2,Ny-1]=2*u2[2:Nx-2,Ny-2]-u2[2:Nx-2,Ny-3]
        u2[1,Ny-1]=2*u2[2,Ny-2]-u2[3,Ny-3]
        u2[0,Ny-2]=2*u2[1,Ny-3]-u2[2,Ny-4]
        u2[0,Ny-1]=2*u2[1,Ny-2]-u2[2,Ny-3]
        u2[Nx-1,1]=2*u2[Nx-2,2]-u2[Nx-3,3]
        u2[Nx-2,0]=2*u2[Nx-3,1]-u2[Nx-4,2]
        u2[Nx-1,0]=2*u2[Nx-2,1]-u2[Nx-3,2]
        u2[Nx-1,Ny-1]=2*u2[Nx-2,Ny-2]-u2[Nx-3,Ny-3]
        u2[Nx-2,Ny-1]=2*u2[Nx-3,Ny-2]-u2[Nx-4,Ny-3]
        u2[Nx-1,Ny-2]=2*u2[Nx-2,Ny-3]-u2[Nx-3,Ny-4]
        # y축으로 풀기
        for i in range(1,Ny-1):
            fy[:]=u2[i,1:Nx-1]/dt+0.5*rho*x_volatility*y_volatility\
              *x[i]*y[1:Ny-1]*(u2[i+1,2:Ny]+u2[i-1,0:Ny-2]-u[i-1, 2:Ny]\
              -u[i+1,0:Ny-2])/(4*h**2)
            fy[0]=fy[0]-ay[0]*u2[i,0]
            fy[Ny-3]=fy[Ny-3]-cy[Ny-3]*u2[i,-1]
            u[i,1:Ny-1]=thomas(ay,dy,cy,fy)

    fig2=plt.figure()
    bx=fig2.add_subplot(projection='3d')
```

```
bx.plot_surface(X,Y,u[:,:],cmap=plt.cm.gray)
bx.view_init(elev=30., azim=-132)
bx.set_xlabel('x', fontsize=10)
bx.set_ylabel('y', fontsize=10)
bx.zaxis.set_rotate_label(False)
bx.set_zlabel('Call option price', rotation=90, fontsize=10)
plt.show()
ii=np.argwhere(x==100)
jj=np.argwhere(y==100)
print('Price=%f'%(u[ii,jj]))
```



<초기조건>



<OSM으로 푼 방정식>

Price=21.532784

3) 비균일 격자로 기초자산이 3개인 콜옵션 가격 결정

- 기초자산이 3개인 경우 편미분 방정식

$$\frac{\partial u}{\partial t}=\frac{1}{2}\sigma_x^2x^2\frac{\partial^2u}{\partial x^2}+\frac{1}{2}\sigma_y^2y^2\frac{\partial^2u}{\partial y^2}+\frac{1}{2}\sigma_z^2z^2\frac{\partial^2u}{\partial z^2}+rx\frac{\partial u}{\partial x}+ry\frac{\partial u}{\partial y}+rz\frac{\partial u}{\partial z}$$
$$+\rho_{xy}\sigma_x\sigma_yxy\frac{\partial^2u}{\partial x\partial y}+\rho_{yz}\sigma_y\sigma_zyz\frac{\partial^2u}{\partial y\partial z}+\rho_{xz}\sigma_x\sigma_zxz\frac{\partial^2u}{\partial x\partial z}-ru$$

07

```
import numpy as np
x_volatility=0.3; y_volatility=0.3; z_volatility=0.3 # 각각의 변동성
rho_xy=0.4; rho_yz=0.4; rho_xz=0.4; # 상관계수
r=0.015;  # 무위험 이자율
T=1; # 만기
K1=100; K2=K1; K3=K1
Nt=360; dt=1/Nt  # 최초 기준가격

x1=np.array([0])
```

```
x2=np.arange(64,132,2)
x3=np.array([170,180,200,201,213,223,242,255,267,278,289,300])
x=np.r_[x1,x2,x3]
y1=np.array([0])
y2=np.arange(60,126,2)
y3=np.array([160,175,200,202,214,215,252,267,278,279,282,300])
y=np.r_[y1,y2,y3]
z1=np.array([0])
z2=np.arange(70,136,2)
z3=np.array([165,173,200,205,206,222,234,268,269,270,282,300])
z=np.r_[z1,z2,z3]
# x, y, z벡터의 크기
Nx=len(x); Ny=len(y); Nz=len(z);
hx=np.diff(x); hy=np.diff(y); hz=np.diff(z);
u0=np.zeros((Nx,Ny,Nz))
u=np.zeros((Nx,Ny,Nz))
u1=np.zeros((Nx,Ny,Nz))
u2=np.zeros((Nx,Ny,Nz))
# 유한차분법으로 콜옵션 가격을 구하기 위한 초기값
for i in range(Nx):
    for j in range(Ny):
        for k in range(Nz):
            u0[i,j,k]=np.maximum(np.maximum(np.maximum(x[i]-K1,\
              y[j]-K2),z[k]-K3),0)
# 유한차분법을 사용하기 위한 계수
[ax,dx,cx,ay,dy,cy,az,dz,cz]=map(np.zeros,[Nx-2,Nx-2,Nx-2,Ny-2,Ny-2,Ny-2,
Nz-2,Nz-2,Nz-2])
ax[:]=(-(x_volatility*x[1:Nx-1])**2+r*x[1:Nx-1]*hx[1:Nx-1])/(hx[0:Nx-2]*(hx[0:
Nx-2]+hx[1:Nx-1]))
dx[:]=1.0/dt+(x_volatility*x[1:Nx-1])**2/(hx[0:Nx-2]*hx[1:Nx-1])-r*x[1:Nx-1]*(
hx[1:Nx-1]-hx[0:Nx-2])/(hx[0:Nx-2]*hx[1:Nx-1])+r/3
cx[:]=-((x_volatility*x[1:Nx-1])**2+r*x[1:Nx-1]*hx[0:Nx-2])/(hx[1:Nx-1]*(hx[0:
Nx-2]+hx[1:Nx-1]))
# 선형 경계조건
ax[Nx-3]=ax[Nx-3]-cx[Nx-3]
dx[Nx-3]=dx[Nx-3]+2*cx[Nx-3]
ay[:]=(-(y_volatility*y[1:Ny-1])**2+r*y[1:Ny-1]*hy[1:Ny-1])/(hy[0:Ny-2]*(hy[0:
Ny-2]+hy[1:Ny-1]))
dy[:]=1.0/dt+(y_volatility*y[1:Ny-1])**2/(hy[0:Ny-2]*hy[1:Ny-1])-r*y[1:Ny-1]*(
hy[1:Ny-1]-hy[0:Ny-2])/(hy[0:Ny-2]*hy[1:Ny-1])+r/3
cy[:]=-((y_volatility*y[1:Ny-1])**2+r*y[1:Ny-1]*hy[0:Ny-2])/(hy[1:Ny-1]*(hy[0:
```

```
Ny-2]+hy[1:Ny-1]))
ay[Ny-3]=ay[Ny-3]-cy[Ny-3]
dy[Ny-3]=dy[Ny-3]+2*cy[Ny-3]
az[:]=(-(z_volatility*z[1:Nz-1])**2+r*z[1:Nz-1]*hz[1:Nz-1])/(hz[0:Nz-2]*(hz[0:N
z-2]+hz[1:Nz-1]))
dz[:]=1.0/dt+(z_volatility*z[1:Nz-1])**2/(hz[0:Nz-2]*hz[1:Nz-1])-r*z[1:Nz-1]*(h
z[1:Nz-1]-hz[0:Nz-2])/(hz[0:Nz-2]*hz[1:Nz-1])+r/3
cz[:]=-((z_volatility*z[1:Nz-1])**2+r*z[1:Nz-1]*hz[0:Nz-2])/(hz[1:Nz-1]*(hz[0:N
z-2]+hz[1:Nz-1]))
az[Nz-3]=az[Nz-3]-cz[Nz-3]
dz[Nz-3]=dz[Nz-3]+2*cz[Nz-3]
[fx,fy,fz]=map(np.zeros, [Nx-2,Ny-2,Nz-2])
u=u0

# OSM과 토마스 알고리즘을 이용하여 u값 계산
for n in range(Nt):
    # x축으로 풀기
    for j in range(1,Ny-1):
        for k in range(1,Nz-1):
            fx[0:Nx-1]=(1/3)*(rho_xy*x_volatility\
              *y_volatility*x[1:Nx-1]*y[j]\
              *(u[2:Nx,j+1,k]-u[2:Nx,j-1,k]\
              -u[0:Nx-2,j+1,k]+u[0:Nx-2,j-1,k])\
              /(hx[0:Nx-2]*hy[j]+hx[1:Nx-1]\
              *hy[j]+hx[1:Nx-1]*hy[j-1]\
              +hx[0:Nx-2]*hy[j-1])+rho_xz\
              *x_volatility*z_volatility\
              *x[1:Nx-1]*z[k]*(u[2:Nx,j,k+1]\
              -u[2:Nx,j,k-1]-u[0:Nx-2,j,k+1]\
              +u[0:Nx-2,j,k-1])/(hx[0:Nx-2]\
              *hz[k]+hx[1:Nx-1]*hz[k]+hx[1:Nx-1]\
              *hz[k-1]+hx[0:Nx-2]*hz[k-1])+rho_yz
              *y_volatility*z_volatility*y[j]*z[k]*\
              (u[1:Nx-1,j+1,k+1]-u[1:Nx-1,j+1,k-1]\
              -u[1:Nx-1,j-1,k+1]+u[1:Nx-1,j-1,k-1])\
              /(hy[j-1]*hz[k]+hy[j]*hz[k]+hy[j]\
              *hz[k-1]+hy[j-1]*hz[k-1]))+u[1:Nx-1,j,k]/dt
            u1[1:Nx-1,j,k]=thomas(ax,dx,cx,fx)
    # y축으로 풀기
    for k in range(1,Nz-1):
        for i in range(1,Nx-1):
```

```
            fy[0:Ny-1]=(1/3)*(rho_xy*x_volatility\
              *y_volatility*x[i]*y[1:Ny-1]\
              *(u1[i+1,2:Ny,k]-u1[i+1,0:Ny-2,k]\
              -u1[i-1,2:Ny,k]+u1[i-1,0:Ny-2,k])\
              /(hx[i-1]*hy[1:Ny-1]+hx[i]\
              *hy[1:Ny-1]+hx[i]*hy[0:Ny-2]\
              +hx[i-1]*hy[0:Ny-2])+rho_xz\
              *x_volatility*z_volatility\
              *x[i]*z[k]*(u1[i+1,1:Ny-1,k+1]\
              -u1[i+1,1:Ny-1,k-1]-u1[i-1,1:Ny-1,k+1]\
              +u1[i-1,1:Ny-1,k-1])/(hx[i-1]*hz[k]\
              +hx[i]*hz[k]+hx[i]*hz[k-1]\
              +hx[i-1]*hz[k-1])+rho_yz*y_volatility\
              *z_volatility*y[1:Ny-1]*z[k]\
              *(u1[i,2:Ny,k+1]-u1[i,2:Ny,k-1]\
              -u1[i,0:Ny-2,k+1]+u1[i,0:Ny-2,k-1])\
              /(hy[0:Ny-2]*hz[k]+hy[1:Ny-1]*hz[k]\
              +hy[1:Ny-1]*hz[k-1]+hy[0:Ny-2]*hz[k-1]))\
              +u1[i,1:Ny-1,k]/dt
            u2[i,1:Ny-1,k]=thomas(ay,dy,cy,fy)
    # z 축으로 풀기
    for i in range(1,Nx-1):
        for j in range(1,Ny-1):
            fz[0:Nz-1]=(1/3)*(rho_xy*x_volatility\
              *y_volatility*x[i]*y[j]\
              *(u2[i+1,j+1,1:Nz-1]\
              -u2[i+1,j-1,1:Nz-1]\
              -u2[i-1,j+1,1:Nz-1]\
              +u[i-1,j-1,1:Nz-1])/(hx[i-1]*hy[j]\
              +hx[i]*hy[j]+hx[i]*hy[j-1]+hx[i-1]\
              *hy[j-1])+rho_xz*x_volatility\
              *z_volatility*x[i]*z[1:Nz-1]\
              *(u2[i+1,j,2:Nz]-u2[i+1,j,0:Nz-2]\
              -u2[i-1,j,2:Nz]+u2[i-1,j,0:Nz-2])\
              /(hx[i-1]*hz[1:Nz-1]\
              +hx[i]*hz[1:Nz-1]+hx[i]*hz[0:Nz-2]\
              +hx[i-1]*hz[0:Nz-2])+rho_yz*y_volatility\
              *z_volatility*y[j]*z[1:Nz-1]\
              *(u2[i,j+1,2:Nz]-u2[i,j+1,0:Nz-2]\
              -u2[i,j-1,2:Nz]+u2[i,j-1,0:Nz-2])\
              /(hy[j-1]*hz[1:Nz-1]+hy[j]*hz[1:Nz-1]\
```

```
                    +hy[j]*hz[0:Nz-2]+hy[j-1]*hz[0:Nz-2]))\
                    +u2[i,j,1:Nz-1]/dt
                u[i,j,1:Nz-1]=thomas(az,dz,cz,fz)


ii=np.argwhere(x==100)
jj=np.argwhere(y==100)
kk=np.argwhere(z==100)
print('Price=%f'%(u[ii,jj,kk]))
```

Price=25.159229

3. FDM을 이용한 ELS 가격결정

1) 기초자산이 1개인 ELS

① KOSPI200이 기초자산인 ELS 평가

투자설명서 요약

- 기초자산가격 변동성 : KOSPI200 지수 : 17.78%

  변동성 기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005)) 하여 Volatility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용

- 공정가격 : 2018년 03월 13일 기준 9,733.63원으로 추산
- 최초기준가격평가일 : 2018년 03월 23일
- 자동조기상환평가일 및 상환금액

차수	자동조기상환평가일	자동조기상환 조건	상환금액
1차	2018년 09월 19일	최초기준가격의 95%이상	액면금액 × 102.20%
2차	2019년 03월 20일	최초기준가격의 95%이상	액면금액 × 104.40%
3차	2019년 09월 19일	최초기준가격의 95%이상	액면금액 × 106.60%
4차	2020년 03월 19일	최초기준가격의 90%이상	액면금액 × 108.80%
5차	2020년 09월 21일	최초기준가격의 90%이상	액면금액 × 111.00%

- 만기평가일 : 2021년 03월 19일
- 만기상환 조건

  ⓐ 기초자산의 만기평가가격이 최초기준가격의 85% 이상인 경우: 액면금액 × 113.20%

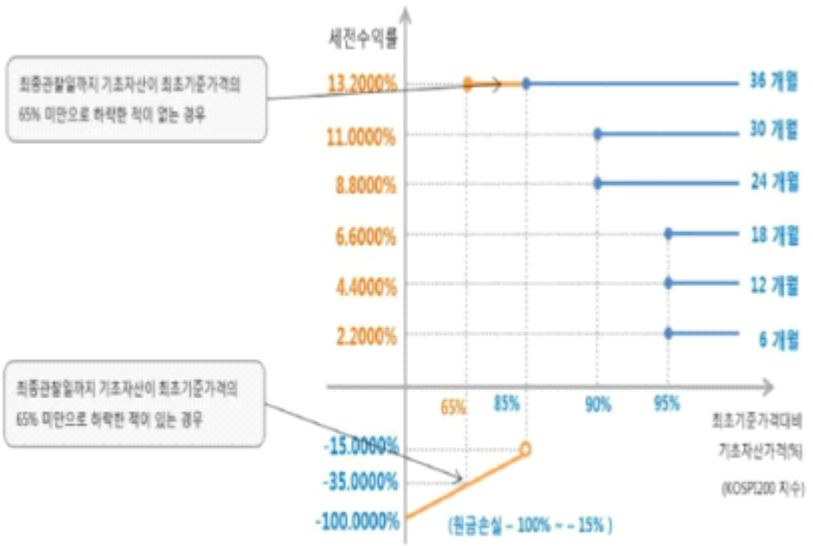
  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격평가일 익일로부터 최종관찰일(포함)까지 기초자산의 평가가격이 최초기준가격의 65%미만으로 하락한 적이 없는 경우: 액면금액 × 113.20%

  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격평가일 익일로부터 최종관찰일(포함)까지 기초자산의 평가가격이 최초기준가격의 65%미만으로 하락한 적이 있는 경우: 액면금액 × (만기평가가격/최초기준가격)

- 예상 손익구조 그래프

- 파이썬 코드로 구현

```
3  import numpy as np
```



0.022])

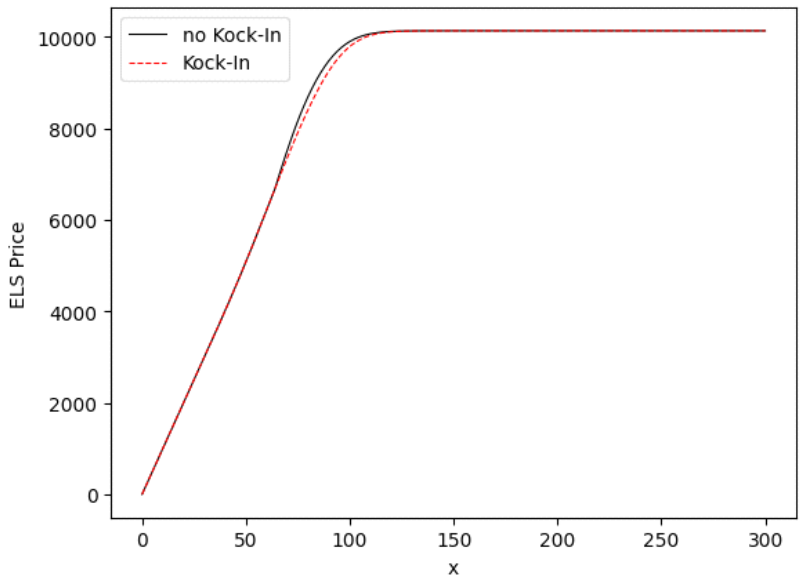
np.rint(3*Nt/6),

4	for i in range (0,Nx):
	  if (x[i]<kib*x0):
	  u[i,0]=x[i]/x0*facevalue
	  ku[i,0]=x[i]/x0*facevalue
	  elif (x[i]<strike_price[0]*x0):
	  u[i,0]=facevalue*(1+dummy)
	  ku[i,0]=x[i]/x0*facevalue
	  else:
	  u[i,0]=facevalue*(1+coupon_rate[0])
	  ku[i,0]=facevalue*(1+coupon_rate[0])
	[a,d,c,b]=map(np.zeros, [Nx, Nx, Nx, Nx])
	a[:]=r*x/(2*h)-(volatility*x)**2/(2*h**2)
	d[:]=(volatility*x)**2/(h**2)+r+(1/dt)
	c[:]=-r*x/(2*h)-(volatility*x)**2/(2*h**2)
	a[Nx-1]=a[Nx-1]-c[Nx-1]; d[Nx-1]=d[Nx-1]+2*c[Nx-1]
	tag=0
	
	for n in range (0,Nt):
	  if (n==step[tag]):
	  s=np.min(np.where(x>=x0*strike_price[tag+1]))
	  u[s:Nx+1,n]=facevalue*(1+coupon_rate[tag+1])
	  ku[s:Nx+1,n]=facevalue*(1+coupon_rate[tag+1])
	  tag=tag+1
	  s=np.min(np.where(x>=x0*kib))
	  u[0:s,n]=ku[0:s, n]
	  b=u[:,n]/dt

```
        u[:,n+1]=thomas(a,d,c,b)
        b=ku[:,n]/dt
        ku[:,n+1]=thomas(a,d,c,b)

ii=np.where(x==100)
print('Price=%f'%(u[ii,Nt]))
```

Price=9899.781594 (ELS 최종가격)

```
plt.figure(1)
plt.plot(u[:,Nt-1], 'k', linewidth=0.8, label='no Kock-In')
plt.plot(ku[:,Nt-1], 'r--', linewidth=0.8, label='Kock-In')
plt.xlabel("x", fontsize=10)
plt.ylabel("ELS Price", fontsize=10)
plt.legend()
plt.show()
```



② 개별주식이 기초자산인 ELS 평가

투자설명서 요약
- 기초자산 : LG화학 보통주(051910)
- 기초자산가격 변동성 : LG화학 보통주(051910) : 42.17%
  기초자산가격 변동성 산출기준 LG화학 보통주(051910): 6개월 변동성과 3년 변동성의 평균
- 공정가격 : 일괄신고추가서류 제출일 전일 기준 9,562.87원으로 추산
- 최초기준가격결정일 : 2023년 05월 09일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2023년 11월 08일	최초기준가격의 90%이상	액면금액 × 105.68%
2차	2024년 05월 08일	최초기준가격의 85%이상	액면금액 × 111.36%
3차	2024년 11월 08일	최초기준가격의 80%이상	액면금액 × 117.04%
4차	2025년 05월 08일	최초기준가격의 80%이상	액면금액 × 122.72%
5차	2025년 11월 07일	최초기준가격의 75%이상	액면금액 × 128.40%

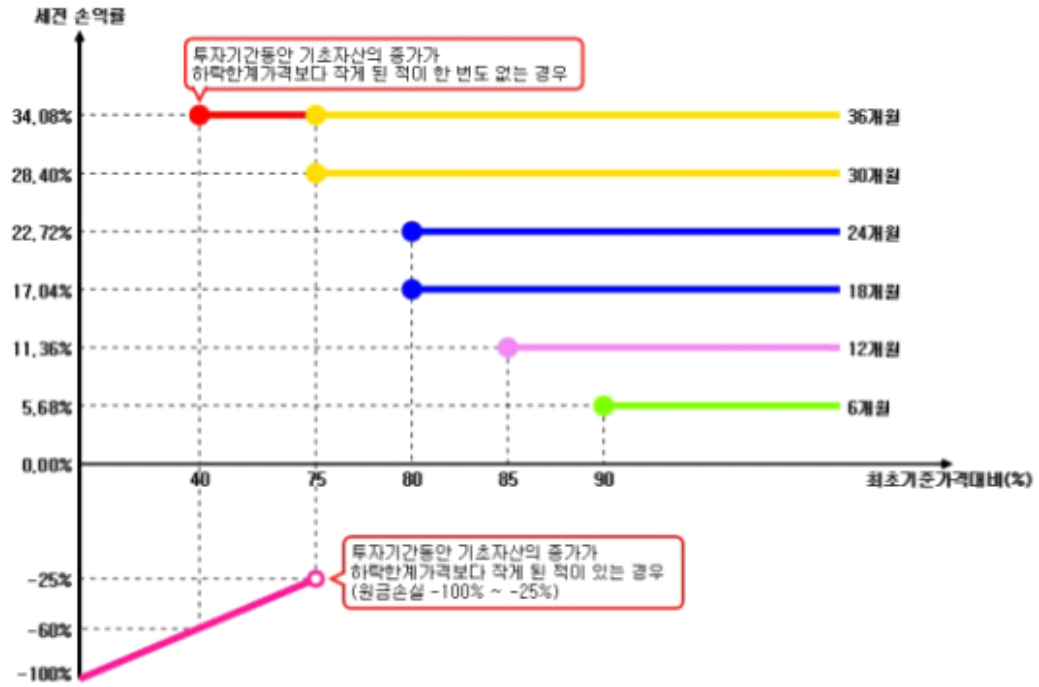
- 만기평가일 : 2026년 05월 06일, 2026년 05월 07일, 2026년 05월 08일
- 최종기준가격 : LG화학 보통주(051910) 최종기준가격 결정일 현재 기초자산의 거래소 종가 3개의 산술평균
- 만기행사가격 : LG화학 보통주(051910) 최초기준가격 × 75%
- 하락한계가격 : LG화학 보통주(051910) 최초기준가격 × 40%
- 만기상환 조건

  ⓐ 기초자산의 최종기준가격이 최초기준가격의 75% 이상인 경우: 액면금액 × 134.08%

  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 마지막 최종기준가격 결정일(포함) 이전까지 기초자산이 종가에 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 134.08%

  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 마지막 최종기준가격 결정일(포함) 이전까지 기초자산이 종가에 한 번이라도 하락한계가격보다 작게 된 적이 있는 경우: 액면금액 × (최종기준가격 / 최초기준가격)

- 예상 손익구조 그래프



- 파이썬 코드로 구현

6

```
import numpy as np
import matplotlib.pyplot as plt

facevalue=10000; R=300; volatility=0.4217
r=0.0355; Nx=301; h=R/Nx; x0=100
x=np.linspace(0, R, Nx); T=3
Nt=360*T; dt=T/Nt
u=np.zeros((Nx, Nt+1)); ku=np.zeros((Nx, Nt+1))
coupon_rate=np.array([0.3408, 0.2840, 0.2272, 0.1704, 0.1136, 0.0568])
strike_price=np.array([0.75, 0.75, 0.80, 0.80, 0.85, 0.90])
step=np.array([np.rint(Nt/6), np.rint(2*Nt/6), np.rint(3*Nt/6),
np.rint(4*Nt/6), np.rint(5*Nt/6), np.rint(6*Nt/6),Nt+1])
dummy=0.3408; kib=0.40
```

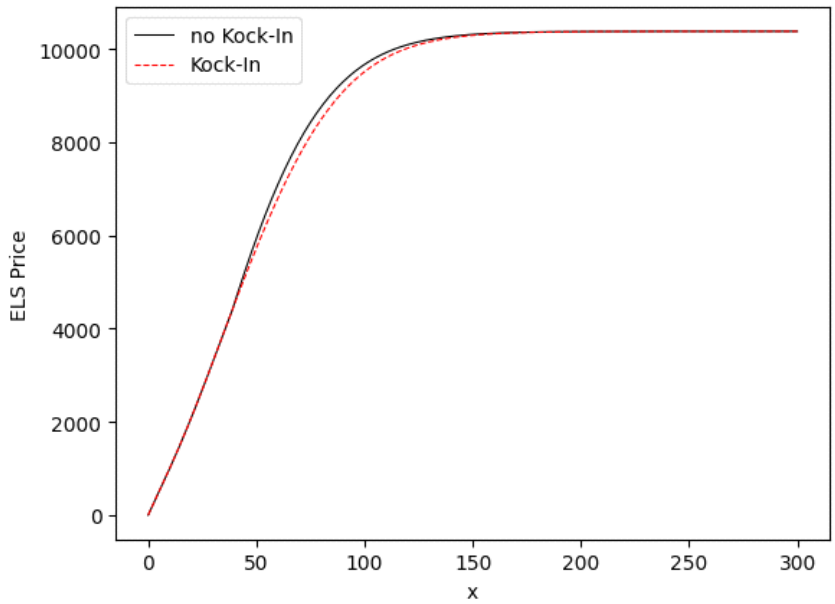
7

```
for i in range (0,Nx):
    if (x[i]<kib*x0):
        u[i,0]=x[i]/x0*facevalue
        ku[i,0]=x[i]/x0*facevalue
    elif (x[i]<strike_price[0]*x0):
        u[i,0]=facevalue*(1+dummy)
        ku[i,0]=x[i]/x0*facevalue
    else:
        u[i,0]=facevalue*(1+coupon_rate[0])
        ku[i,0]=facevalue*(1+coupon_rate[0])
[a,d,c,b]=map(np.zeros, [Nx, Nx, Nx, Nx])
a[:]=r*x/(2*h)-(volatility*x)**2/(2*h**2)
d[:]=(volatility*x)**2/(h**2)+r+(1/dt)
c[:]=-r*x/(2*h)-(volatility*x)**2/(2*h**2)
a[Nx-1]=a[Nx-1]-c[Nx-1]; d[Nx-1]=d[Nx-1]+2*c[Nx-1]
tag=0

for n in range (0,Nt):
    if (n==step[tag]):
        s=np.min(np.where(x>=x0*strike_price[tag+1]))
        u[s:Nx+1,n]=facevalue*(1+coupon_rate[tag+1])
        ku[s:Nx+1,n]=facevalue*(1+coupon_rate[tag+1])
        tag=tag+1
    s=np.min(np.where(x>=x0*kib))
    u[0:s,n]=ku[0:s, n]
    b=u[:,n]/dt
```

```
        u[:,n+1]=thomas(a,d,c,b)
        b=ku[:,n]/dt
        ku[:,n+1]=thomas(a,d,c,b)

ii=np.where(x==100)
print('Price=%f'%(u[ii,Nt]))
```

Price=9669.423079 (ELS 최종가격)

```
8
plt.figure(1)
plt.plot(u[:,Nt-1], 'k', linewidth=0.8, label='no Kock-In')
plt.plot(ku[:,Nt-1], 'r--', linewidth=0.8, label='Kock-In')
plt.xlabel("x", fontsize=10)
plt.ylabel("ELS Price", fontsize=10)
plt.legend()
plt.show()
```



2) 기초자산이 2개인 ELS

① 기초자산이 두 개(지수)인 ELS 평가

투자설명서 요약

- 기초자산 : KOSPI200, S&P500
- 기초자산가격 변동성 : KOSPI200 : 23.22%, S&P500 : 27.38%
  변동성 산출기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005))하여 Vola tility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용
- 기초자산간의 상관계수 KOSPI200, S&P500 : 0.0489

상관계수 산출기준 : 180 영업일 역사적 상관계수

- 공정가격 : 본 증권의 공정가격은 2023년 04월 25일 기준 9,446.82원으로 추산됩니다.
- 최초기준가격결정일 : 2023년 05월 15일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2023년 11월 10일	두 지수 모두 최초기준가격의 90%이상	액면금액 × 103.15%
2차	2024년 05월 9일	두 지수 모두 최초기준가격의 90%이상	액면금액 × 106.30%
3차	2024년 11월 12일	두 지수 모두 최초기준가격의 85%이상	액면금액 × 109.45%
4차	2025년 05월 12일	두 지수 모두 최초기준가격의 85%이상	액면금액 × 112.60%
5차	2025년 11월 11일	두 지수 모두 최초기준가격의 80%이상	액면금액 × 115.75%

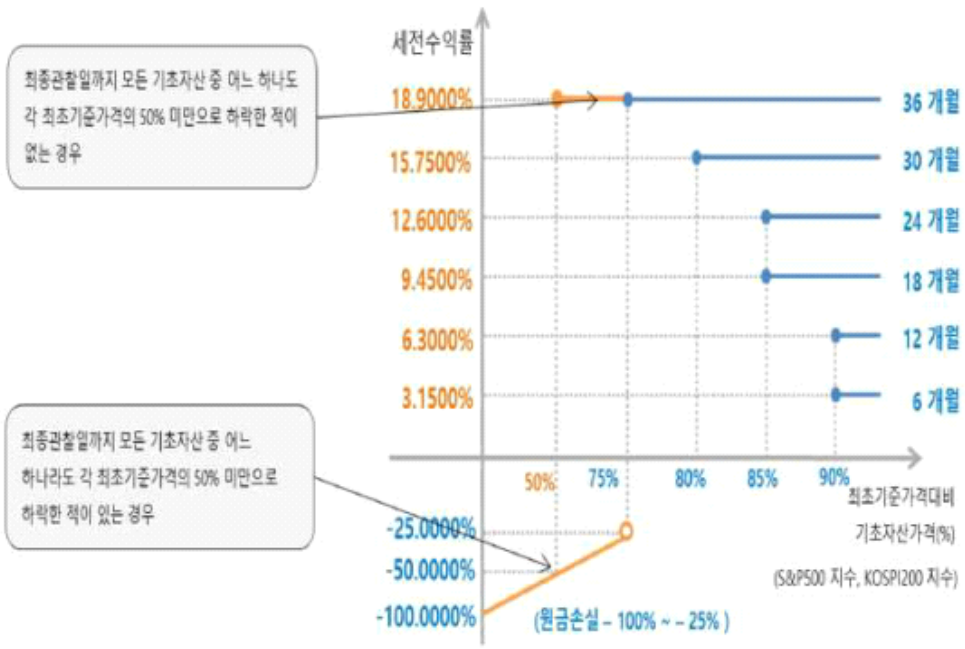
- 만기평가일 : 2026년 05월 12일
- 최종기준가격 : 최종기준가격 결정일 현재 기초자산의 거래소 종가 또는 산출된 종가
- 만기행사가격 : KOSPI200 최초기준가격 × 75%, S&P500 최초기준가격 × 75%
- 하락한계가격 : KOSPI200 최초기준가격 × 50%, S&P500 최초기준가격 × 50%
- 만기상환 조건

ⓐ 기초자산의 최종기준가격이 모두 최초기준가격의 75% 이상인 경우: 액면금액 × 118.90%

ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 각각의 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 118.90%

ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 한 번이라도 각각의 하락한계 가격보다 작게 된 적이 있는 경우: 기준종목 기준으로 {(만기평가가격/최초기준가격)- 1} × 100%

※ 기준종목 : 모든 기초자산 중 [만기평가가격/최초기준가격]의 비율이 가장 낮은 기초자산



- 파이썬 코드로 구현

9

```
def thomas(alpha, beta, gamma, f):
    n=len(f)
    v=np.zeros(n)
    [aa, dd, cc, bb]=map(np.array,[alpha, beta, gamma, f])
    for i in range(1, n):
        mult=aa[i]/dd[i-1]
        dd[i]=dd[i]-mult*cc[i-1]
        bb[i]=bb[i]-mult*bb[i-1]
    v[n-1]=bb[n-1]/dd[n-1]
    for i in range(n-2, -1, -1):
        v[i]=(bb[i]-cc[i]*v[i+1])/dd[i]
    return v
```

10

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

facevalue=10000; R=300;
x_volatility=0.249; y_volatility=0.2182;
rho=0.0489; r=0.0355; Nx=61; Ny=Nx; h=R/Nx;
x0=100; y0=100; x=np.linspace(0, R, Nx); y=x;
T=3; Nt=360*T; dt=T/Nt; lst=[Nx, Ny]
[u, ku, old_u, old_ku]=map(np.zeros, [lst, lst, lst, lst])
coupon_rate=np.array([0.189, 0.1575, 0.126, 0.0945, 0.063, 0.0315])
strike_price=np.array([0.75, 0.80, 0.85, 0.85, 0.90, 0.90])
step=np.array([np.rint(Nt/6), np.rint(2*Nt/6), np.rint(3*Nt/6),
np.rint(4*Nt/6), np.rint(5*Nt/6), np.rint(6*Nt/6),Nt+1])
dummy=0.189; kib=0.50
```

11

```
for i in range (0,Nx):
    for j in range(0, Ny):
        if (x[i]<kib*x0 or y[j]<kib*y0):
            u[i,j]=np.minimum(x[i], y[j])/x0*facevalue
            ku[i,j]=np.minimum(x[i], y[j])/x0*facevalue
        elif (x[i]<=strike_price[0]*x0 or y[j]<=strike_price[0]*x0):
            u[i,j]=facevalue*(1+dummy)
            ku[i,j]=np.minimum(x[i], y[j])/x0*facevalue
        else:
            u[i,j]=facevalue*(1+coupon_rate[0])
```

```
                ku[i,j]=facevalue*(1+coupon_rate[0])
[ax,dx,cx,ay,dy,cy]=map(np.zeros, [Nx-2, Nx-2, Nx-2, Ny-2, Ny-2, Ny-2])
ax[:]=-0.5*(x_volatility*x[1:Nx-1]/h)**2+0.5*r*x[1:Nx-1]/h
dx[:]=1/dt+(x_volatility*x[1:Nx-1]/h)**2+r*0.5
cx[:]=-0.5*(x_volatility*x[1:Nx-1]/h)**2-0.5*r*x[1:Nx-1]/h
ax[Nx-3]=ax[Nx-3]-cx[Nx-3]
dx[Nx-3]=dx[Nx-3]+2*cx[Nx-3]

ay[:]=-0.5*(y_volatility*y[1:Ny-1]/h)**2+0.5*r*y[1:Ny-1]/h
dy[:]=1/dt+(y_volatility*y[1:Ny-1]/h)**2+r*0.5
cy[:]=-0.5*(y_volatility*y[1:Ny-1]/h)**2-0.5*r*y[1:Ny-1]/h
ay[Ny-3]=ay[Ny-3]-cy[Ny-3]
dy[Ny-3]=dy[Ny-3]+2*cy[Ny-3];
tag=0
bx=np.zeros(Nx-2); by=np.zeros(Nx-2);

for n in range (0,Nt):
    if (n==step[tag]):
        gx=np.min(np.where(x>=x0*strike_price[tag+1]))
        gy=np.min(np.where(y>=y0*strike_price[tag+1]))
        u[gx:Nx-1,gy:Ny-1]=facevalue*(1+coupon_rate[tag+1])
        ku[gx:Nx-1,gy:Ny-1]=facevalue*(1+coupon_rate[tag+1])
        tag+=1
    gx=np.min(np.where(x>=x0*kib))
    gy=np.min(np.where(y>=y0*kib))
    u[:,0:gy+1]=ku[:,0:gy+1]
    u[0:gx+1,:]=ku[0:gx+1,:]
    old_u=u; old_ku=ku;
    for j in range(1,Ny-1):

bx[0:Nx-1]=old_u[1:Nx-1,j]/dt+0.5*rho*x_volatility*y_volatility*x[1:Nx-1]*y[j]*
(old_u[2:Nx,j+1]-old_u[2:Nx,j-1]-old_u[0:Nx-2,j+1]+old_u[0:Nx-2,j-1])/(4*h**2
)
        u[1:Nx-1, j]=thomas(ax,dx,cx,bx)
    u[Nx-1,1:Ny-1]=2*u[Nx-2,1:Ny-1]-u[Nx-3,1:Ny-1]
    u[:,Ny-1]=2*u[:,Ny-2]-u[:,Ny-3]
    old_u=u;
    for i in range(1,Nx-1):

by[0:Ny-1]=old_u[i,1:Ny-1]/dt+0.5*rho*x_volatility*y_volatility*x[i]*y[1:Ny-1]*
(old_u[i+1,2:Ny]-old_u[i+1,0:Ny-2]-old_u[i-1,2:Ny]+old_u[i-1,0:Ny-2])/(4*h**2
```

```
)
        u[i,1:Ny-1]=thomas(ay,dy,cy,by)
    u[1:Nx-1,Ny-1]=2*u[1:Nx-1,Ny-2]-u[1:Nx-1,Ny-3]
    u[Nx-1,:]=2*u[Nx-2,:]-u[Nx-3,:]
    for j in range(1,Ny-1):

bx[0:Nx-1]=old_ku[1:Nx-1,j]/dt+0.5*rho*x_volatility*y_volatility*x[1:Nx-1]*y[j]
*(old_ku[2:Nx,j+1]-old_ku[2:Nx,j-1]-old_ku[0:Nx-2,j+1]+old_ku[0:Nx-2,j-1])/(4
*h**2)
        ku[1:Nx-1, j]=thomas(ax,dx,cx,bx)
    ku[Nx-1,1:Ny-1]=2*ku[Nx-2,1:Ny-1]-ku[Nx-3,1:Ny-1]
    ku[:,Ny-1]=2*ku[:,Ny-2]-ku[:,Ny-3]
    old_ku=ku;
    for i in range(1,Nx-1):

by[0:Ny-1]=old_ku[i,1:Ny-1]/dt+0.5*rho*x_volatility*y_volatility*x[i]*y[1:Ny-1]
*(old_ku[i+1,2:Ny]-old_ku[i+1,0:Ny-2]-old_ku[i-1,2:Ny]+old_ku[i-1,0:Ny-2])/(4
*h**2)
        ku[i,1:Ny-1]=thomas(ay,dy,cy,by)
    ku[1:Nx-1,Ny-1]=2*ku[1:Nx-1,Ny-2]-ku[1:Nx-1,Ny-3]
    ku[Nx-1,:]=2*ku[Nx-2,:]-ku[Nx-3,:]

ii=np.where(x==100)
jj=np.where(y==100)
print('Price=%f'%(u[ii,jj]))
```

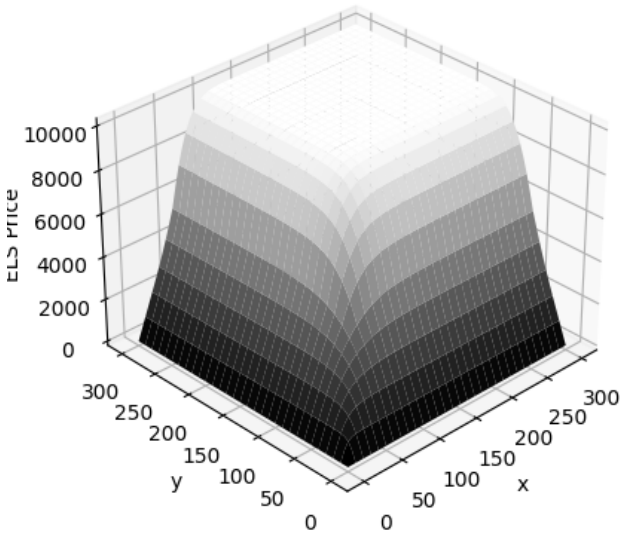
Price=9740.194849 (ELS 최종가격)

```
from matplotlib.figure import projections
X, Y = np.meshgrid(x,y)
fig1=plt.figure()
ax=fig1.add_subplot(projection='3d')
ax.plot_surface(X,Y,u,cmap=plt.cm.gray)
ax.view_init(elev=31,azim=-134)
ax.set_xlabel('x', fontsize=10)
ax.set_ylabel('y', fontsize=10)
ax.zaxis.set_rotate_label(False)
ax.set_zlabel('ELS Price',rotation=90,fontsize=10)

fig2=plt.figure()
bx=fig2.add_subplot(projection='3d')
```

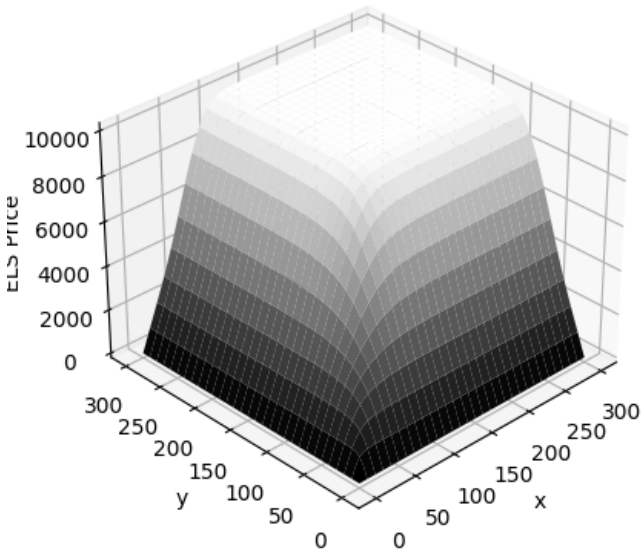
12

	bx.plot_surface(X,Y,ku,cmap=plt.cm.gray) bx.view_init(elev=31,azim=-134) bx.set_xlabel('x', fontsize=10) bx.set_ylabel('y', fontsize=10) bx.zaxis.set_rotate_label(False) bx.set_zlabel('ELS Price',rotation=90,fontsize=10)

<낙인 배리어 아래로 기초자산이 떨어지지 않았을 때의 ELS가격)



<낙인 배리어 아래로 기초자산이 떨어졌을 때의 ELS가격)



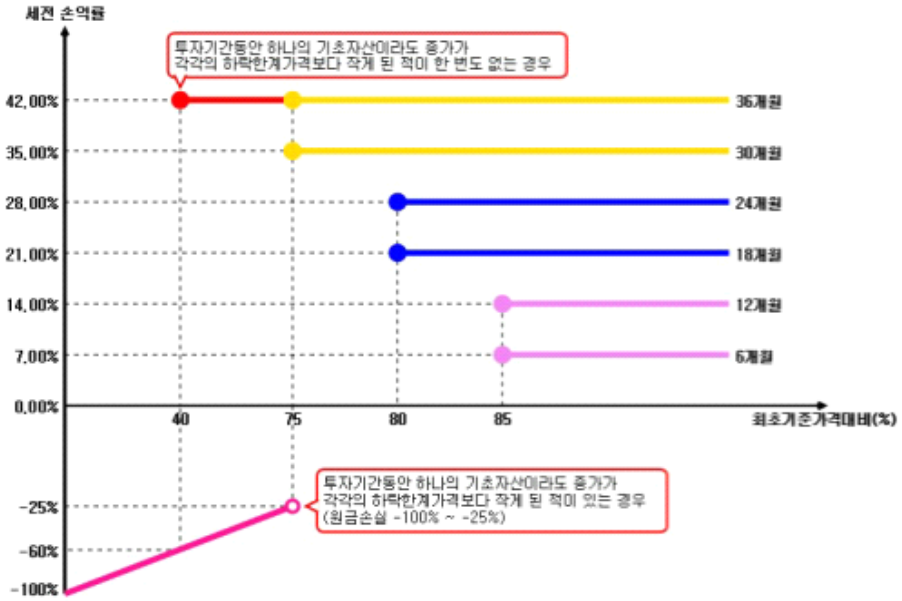
② 기초자산이 두 개(개별종목)인 ELS 평가

투자설명서 요약

- 기초자산 : LG화학 보통주(051910), LG전자 보통주(066570)
- 기초자산가격 변동성 : LG화학 보통주(051910) : 42.17%, LG전자 보통주(066570) : 35.09%
  변동성 산출기준: 6개월 변동성과 3년 변동성의 평균
- 기초자산간의 상관계수 LG화학 보통주(051910), LG전자 보통주(066570) : 0.29555
  상관계수 산출기준 : 6개월 상관계수와 3년 상관계수의 평균
- 공정가격 : 본 일괄신고추가서류 제출일 전일 기준 [9,361.62원]으로 추산됩니다.
- 최초기준가격결정일 : 2023년 05월 12일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2023년 11월 10일	두 종목 모두 최초기준가격의 85%이상	액면금액 × 107%
2차	2024년 05월 10일	두 종목 모두 최초기준가격의 85%이상	액면금액 × 114%
3차	2024년 11월 12일	두 종목 모두 최초기준가격의 80%이상	액면금액 × 121%
4차	2025년 05월 12일	두 종목 모두 최초기준가격의 80%이상	액면금액 × 128%
5차	2025년 11월 12일	두 종목 모두 최초기준가격의 75%이상	액면금액 × 135%

- 만기평가일 : 2026년 05월 12일
- 최종기준가격 : 최종기준가격 결정일 현재 기초자산의 거래소 종가 3개의 산술평균
- 만기행사가격 : LG화학 보통주 최초기준가격 × 75%, LG전자 보통주 최초기준가격 × 75%
- 하락한계가격 : LG화학 보통주 최초기준가격 × 40%, LG전자 보통주 최초기준가격 × 40%
- 만기상환 조건
  ⓐ 기초자산의 최종기준가격이 모두 최초기준가격의 75% 이상인 경우: 액면금액 × 142%
  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 각각의 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 142%
  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 한 번이라도 각각의 하락한계 가격보다 작게 된 적이 있는 경우: [액면가액 × (Worst 가격변동률 + 1)]을 만기상환금액으로 지급



- 파이썬 코드로 구현

10	import numpy as np
	import matplotlib.pyplot as plt
	from mpl_toolkits.mplot3d import Axes3D
	
	facevalue=10000; R=300;
	x_volatility=0.4217; y_volatility=0.3509;
	rho=0.29555; r=0.0355; Nx=61; Ny=Nx; h=R/Nx;
	x0=100; y0=100; x=np.linspace(0, R, Nx); y=x;
	T=3; Nt=360*T; dt=T/Nt; lst=[Nx, Ny]
	[u, ku, old_u, old_ku]=map(np.zeros, [lst, lst, lst, lst])
	coupon_rate=np.array([0.42, 0.35, 0.28, 0.21, 0.14, 0.07])
	strike_price=np.array([0.75, 0.75, 0.80, 0.80, 0.85, 0.85])
	step=np.array([np.rint(Nt/6), np.rint(2*Nt/6), np.rint(3*Nt/6),
	np.rint(4*Nt/6), np.rint(5*Nt/6), np.rint(6*Nt/6),Nt+1])
	dummy=0.42; kib=0.40

11	for i in range (0,Nx):
	  for j in range(0, Ny):
	  if (x[i]<kib*x0 or y[j]<kib*y0):
	  u[i,j]=np.minimum(x[i], y[j])/x0*facevalue
	  ku[i,j]=np.minimum(x[i], y[j])/x0*facevalue
	  elif (x[i]<=strike_price[0]*x0 or y[j]<=strike_price[0]*x0):

```
                u[i,j]=facevalue*(1+dummy)
                ku[i,j]=np.minimum(x[i], y[j])/x0*facevalue
            else:
                u[i,j]=facevalue*(1+coupon_rate[0])
                ku[i,j]=facevalue*(1+coupon_rate[0])
[ax,dx,cx,ay,dy,cy]=map(np.zeros, [Nx-2, Nx-2, Nx-2, Ny-2, Ny-2, Ny-2])
ax[:]=-0.5*(x_volatility*x[1:Nx-1]/h)**2+0.5*r*x[1:Nx-1]/h
dx[:]=1/dt+(x_volatility*x[1:Nx-1]/h)**2+r*0.5
cx[:]=-0.5*(x_volatility*x[1:Nx-1]/h)**2-0.5*r*x[1:Nx-1]/h
ax[Nx-3]=ax[Nx-3]-cx[Nx-3]
dx[Nx-3]=dx[Nx-3]+2*cx[Nx-3]

ay[:]=-0.5*(y_volatility*y[1:Ny-1]/h)**2+0.5*r*y[1:Ny-1]/h
dy[:]=1/dt+(y_volatility*y[1:Ny-1]/h)**2+r*0.5
cy[:]=-0.5*(y_volatility*y[1:Ny-1]/h)**2-0.5*r*y[1:Ny-1]/h
ay[Ny-3]=ay[Ny-3]-cy[Ny-3]
dy[Ny-3]=dy[Ny-3]+2*cy[Ny-3];
tag=0
bx=np.zeros(Nx-2); by=np.zeros(Nx-2);

for n in range (0,Nt):
    if (n==step[tag]):
        gx=np.min(np.where(x>=x0*strike_price[tag+1]))
        gy=np.min(np.where(y>=y0*strike_price[tag+1]))
        u[gx:Nx-1,gy:Ny-1]=facevalue*(1+coupon_rate[tag+1])
        ku[gx:Nx-1,gy:Ny-1]=facevalue*(1+coupon_rate[tag+1])
        tag+=1
    gx=np.min(np.where(x>=x0*kib))
    gy=np.min(np.where(y>=y0*kib))
    u[:,0:gy+1]=ku[:,0:gy+1]
    u[0:gx+1,:]=ku[0:gx+1,:]
    old_u=u; old_ku=ku;
    for j in range(1,Ny-1):

bx[0:Nx-1]=old_u[1:Nx-1,j]/dt+0.5*rho*x_volatility*y_volatility*x[1:Nx-1]*y[j]*
(old_u[2:Nx,j+1]-old_u[2:Nx,j-1]-old_u[0:Nx-2,j+1]+old_u[0:Nx-2,j-1])/(4*h**2
)
        u[1:Nx-1, j]=thomas(ax,dx,cx,bx)
    u[Nx-1,1:Ny-1]=2*u[Nx-2,1:Ny-1]-u[Nx-3,1:Ny-1]
    u[:,Ny-1]=2*u[:,Ny-2]-u[:,Ny-3]
    old_u=u;
```

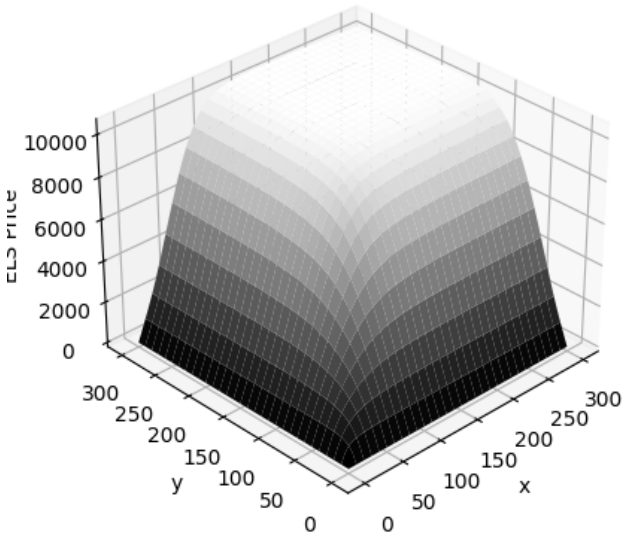
```
    for i in range(1,Nx-1):

by[0:Ny-1]=old_u[i,1:Ny-1]/dt+0.5*rho*x_volatility*y_volatility*x[i]*y[1:Ny-1]*(old_u[i+1,2:Ny]-old_u[i+1,0:Ny-2]-old_u[i-1,2:Ny]+old_u[i-1,0:Ny-2])/(4*h**2)
        u[i,1:Ny-1]=thomas(ay,dy,cy,by)
    u[1:Nx-1,Ny-1]=2*u[1:Nx-1,Ny-2]-u[1:Nx-1,Ny-3]
    u[Nx-1,:]=2*u[Nx-2,:]-u[Nx-3,:]
    for j in range(1,Ny-1):

bx[0:Nx-1]=old_ku[1:Nx-1,j]/dt+0.5*rho*x_volatility*y_volatility*x[1:Nx-1]*y[j]*(old_ku[2:Nx,j+1]-old_ku[2:Nx,j-1]-old_ku[0:Nx-2,j+1]+old_ku[0:Nx-2,j-1])/(4*h**2)
        ku[1:Nx-1, j]=thomas(ax,dx,cx,bx)
    ku[Nx-1,1:Ny-1]=2*ku[Nx-2,1:Ny-1]-ku[Nx-3,1:Ny-1]
    ku[:,Ny-1]=2*ku[:,Ny-2]-ku[:,Ny-3]
    old_ku=ku;
    for i in range(1,Nx-1):

by[0:Ny-1]=old_ku[i,1:Ny-1]/dt+0.5*rho*x_volatility*y_volatility*x[i]*y[1:Ny-1]*(old_ku[i+1,2:Ny]-old_ku[i+1,0:Ny-2]-old_ku[i-1,2:Ny]+old_ku[i-1,0:Ny-2])/(4*h**2)
        ku[i,1:Ny-1]=thomas(ay,dy,cy,by)
    ku[1:Nx-1,Ny-1]=2*ku[1:Nx-1,Ny-2]-ku[1:Nx-1,Ny-3]
    ku[Nx-1,:]=2*ku[Nx-2,:]-ku[Nx-3,:]

ii=np.where(x==100)
jj=np.where(y==100)
print('Price=%f'%(u[ii,jj]))
```

Price=9408.819595 (ELS 최종가격)

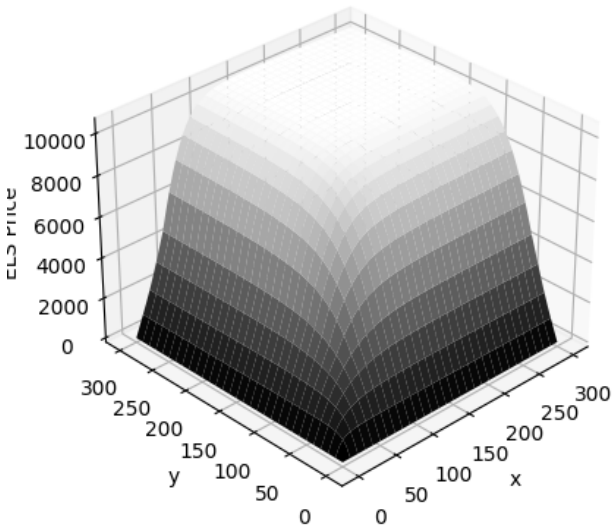
12

```
from matplotlib.figure import projections
X, Y = np.meshgrid(x,y)
fig1=plt.figure()
ax=fig1.add_subplot(projection='3d')
ax.plot_surface(X,Y,u,cmap=plt.cm.gray)
ax.view_init(elev=31,azim=-134)
ax.set_xlabel('x', fontsize=10)
ax.set_ylabel('y', fontsize=10)
ax.zaxis.set_rotate_label(False)
```

```
ax.set_zlabel('ELS Price',rotation=90,fontsize=10)


fig2=plt.figure()
bx=fig2.add_subplot(projection='3d')
bx.plot_surface(X,Y,ku,cmap=plt.cm.gray)
bx.view_init(elev=31,azim=-134)
bx.set_xlabel('x', fontsize=10)
bx.set_ylabel('y', fontsize=10)
bx.zaxis.set_rotate_label(False)
bx.set_zlabel('ELS Price',rotation=90,fontsize=10)
```



<낙인 배리어 아래로 기초자산이 떨어지지 않았을 때의 ELS가격)



<낙인 배리어 아래로 기초자산이 떨어졌을 때의 ELS가격)

3) 기초자산이 3개인 ELS 가격 계산

Case 1

투자설명서 요약

- 기초자산 : EUROSTOXX50 지수, S&P500 지수, KOSPI200 지수
- 기초자산가격 변동성 : EUROSTOXX50지수 : 24.97%, S&P500지수 : 27.38%, KOSPI200 지수 : 23.22%

  변동성 산출기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005))하여 Vola tility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용
- 기초자산간의 상관계수: EUROSTOXX50, S&P500 : 0.5955 EUROSTOXX50, KOSPI200 : 0.2311, S&P500, KOSPI200 : 0.0489

  상관계수 산출기준 : 180 영업일 역사적 상관계수
- 공정가격 : 본 증권의 공정가격은 2023년 04월 25일 기준 9,389.12원으로 추산됩니다.
- 최초기준가격결정일 : 2023년 05월 15일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2023년 11월 10일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 103.5%
2차	2024년 05월 9일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 107.0%
3차	2024년 11월 12일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 110.5%
4차	2025년 05월 12일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 114.0%
5차	2025년 11월 11일	세 지수 모두 최초기준가격의 80%이상	액면금액 × 117.5%

- 만기평가일 : 2026년 05월 12일
- 최종기준가격 : 만기평가일 각 기초자산 종가 (현지시간 기준)
- 만기행사가격 : 각 기초자산 최초기준가격 × 75%
- 하락한계가격 : 각 기초자산 최초기준가격 × 45%
- 만기상환 조건

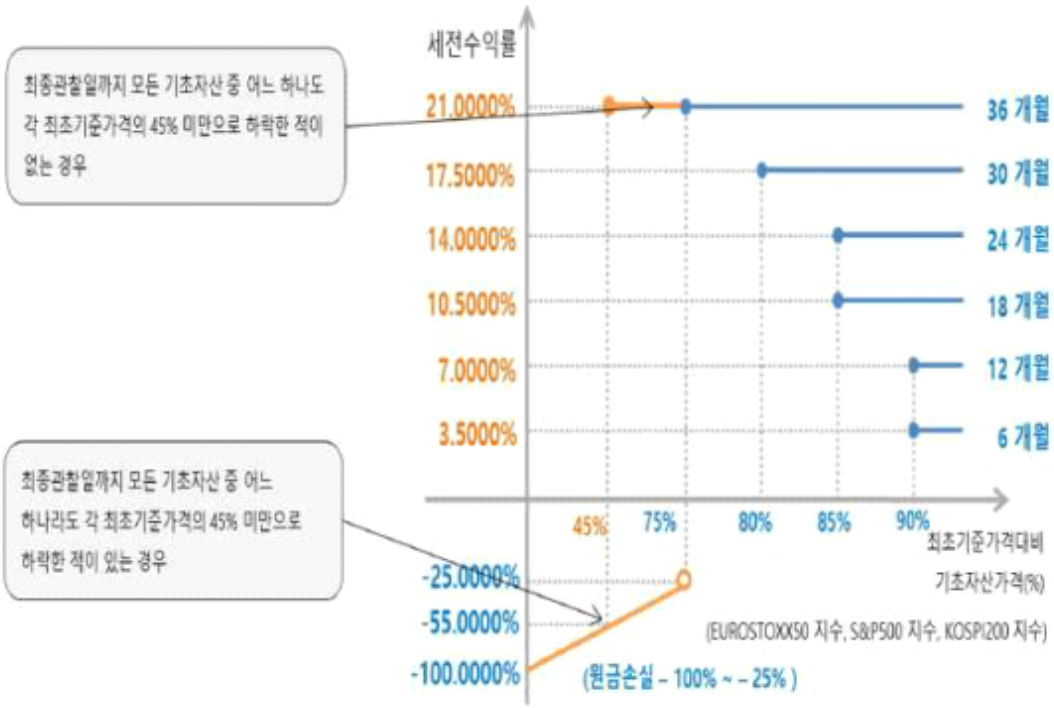
  ⓐ 모든 기초자산의 만기평가가격이 각 최초기준가격의 75% 이상인 경우: 액면금액 × 121.0%

  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 각각의 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 121.0%

  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 한 번이라도 각각의 하락한계 가격보다 작게 된 적이 있는 경우: 기준종목 기준으로 {(만기평가가격/최초기준가격)- 1} × 100%

  ※ 기준종목 : 모든 기초자산 중 [만기평가가격/최초기준가격]의 비율이 가장 낮은 기초자산

- 예상 손익구조 그래프



- 파이썬 코드로 구현

13	def thomas(alpha, beta, gamma, f):
	  n=len(f)
	  v=np.zeros(n)
	  [aa, dd, cc, bb]=map(np.array,[alpha, beta, gamma, f])
	  for i in range(1, n):
	  mult=aa[i]/dd[i-1]
	  dd[i]=dd[i]-mult*cc[i-1]
	  bb[i]=bb[i]-mult*bb[i-1]
	  v[n-1]=bb[n-1]/dd[n-1]
	  for i in range(n-2, -1, -1):
	  v[i]=(bb[i]-cc[i]*v[i+1])/dd[i]
	  return v

14	import numpy as np import matplotlib.pyplot as plt from mpl_toolkits.mplot3d import Axes3D facevalue=10000; x_volatility=0.2497; y_volatility=0.2738; z_volatility=0.2322 # 각각의 변동성

```
rho_xy=0.5955; rho_yz=0.0489; rho_zx=0.2311; # 상관계수
r=0.0355;  # 무위험 이자율
T=3; # 만기
x0=100; y0=100; z0=100;  # 최초 기준가격
# 조기상환시 쿠폰 이자율
coupon_rate=np.array([0.21, 0.175, 0.14, 0.105, 0.07, 0.035])
# 조기행사가
strike_price=np.array([0.75, 0.80, 0.85, 0.85, 0.90, 0.90])
dummy=0.21; kib=0.45; # 더미 이자율, 낙인 배리어
```

```
Nt=360*T; dt=T/Nt; # 시간격자 갯수, 시간격자 간격
A=np.array([0]); B=np.arange(65,132.5,2.5); C=np.array([160,180,200,220]);
x=np.r_[A,B,C]; y=x; z=x;
# x, y, z벡터의 크기
Nx=len(x); Ny=Nx; Nz=Nx;
hx=np.diff(x); hy=hx; hz=hx;
step=np.arange(1,8,1)*Nt/6
lst=[Nx,Ny,Nz]
[u,ku]=map(np.zeros,[lst,lst])
# 유한차분법으로 ELS 가격을 구하기 위한 초기값
for i in range(0,Nx):
    for j in range(0,Ny):
        for k in range(0,Nz):
            if (x[i]<=kib*x0 or y[j]<=kib*y0 or z[k]<=kib*z0):
                u[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
                ku[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
            elif (x[i]<strike_price[0]*x0 or y[j]<strike_price[0]*y0 or
z[k]<strike_price[0]*z0):
                u[i,j,k]=facevalue*(1.0+dummy)
                ku[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
            else:
                u[i,j,k]=facevalue*(1+coupon_rate[0])
                ku[i,j,k]=facevalue*(1+coupon_rate[0])
# 유한차분법을 사용하기 위한 계수
[ax,dx,cx,ay,dy,cy,az,dz,cz]=map(np.zeros,[Nx-2,Nx-2,Nx-2,Ny-2,Ny-2,Ny-2,
Nz-2,Nz-2,Nz-2])
ax[:]=(-(x_volatility*x[1:Nx-1])**2+r*x[1:Nx-1]*hx[1:Nx-1])/(hx[0:Nx-2]*(hx[0:
Nx-2]+hx[1:Nx-1]))
dx[:]=1/dt+(x_volatility*x[1:Nx-1])**2/(hx[0:Nx-2]*hx[1:Nx-1])-r*x[1:Nx-1]*(h
```

```
x[1:Nx-1]-hx[0:Nx-2])/(hx[0:Nx-2]*hx[1:Nx-1])+r/3
cx[:]=-((x_volatility*x[1:Nx-1])**2+r*x[1:Nx-1]*hx[0:Nx-2])/(hx[1:Nx-1]*(hx[0:
Nx-2]+hx[1:Nx-1]))
# 선형 경계조건
ax[Nx-3]=ax[Nx-3]-cx[Nx-3]
dx[Nx-3]=dx[Nx-3]+2*cx[Nx-3]
ay[:]=(-(y_volatility*y[1:Ny-1])**2+r*y[1:Ny-1]*hy[1:Ny-1])/(hy[0:Ny-2]*(hy[0:
Ny-2]+hy[1:Ny-1]))
dy[:]=1/dt+(y_volatility*y[1:Ny-1])**2/(hy[0:Ny-2]*hy[1:Ny-1])-r*y[1:Ny-1]*(h
y[1:Ny-1]-hy[0:Ny-2])/(hy[0:Ny-2]*hy[1:Ny-1])+r/3
cy[:]=-((y_volatility*y[1:Ny-1])**2+r*y[1:Ny-1]*hy[0:Ny-2])/(hy[1:Ny-1]*(hy[0:
Ny-2]+hy[1:Ny-1]))
ay[Ny-3]=ay[Ny-3]-cy[Ny-3]
dy[Ny-3]=dy[Ny-3]+2*cy[Ny-3]
az[:]=(-(z_volatility*z[1:Nz-1])**2+r*z[1:Nz-1]*hz[1:Nz-1])/(hz[0:Nz-2]*(hz[0:N
z-2]+hz[1:Nz-1]))
dz[:]=1/dt+(z_volatility*z[1:Nz-1])**2/(hz[0:Nz-2]*hz[1:Nz-1])-r*z[1:Nz-1]*(hz[
1:Nz-1]-hz[0:Nz-2])/(hz[0:Nz-2]*hz[1:Nz-1])+r/3
cz[:]=-((z_volatility*z[1:Nz-1])**2+r*z[1:Nz-1]*hz[0:Nz-2])/(hz[1:Nz-1]*(hz[0:N
z-2]+hz[1:Nz-1]))
az[Nz-3]=az[Nz-3]-cz[Nz-3]
dz[Nz-3]=dz[Nz-3]+2*cz[Nz-3]
# OS방법을 사용하기 위해 u, ku와 같은 초기의 행렬 생성
[old_u,old_ku]=map(np.zeros,[lst,lst])
[fx,fy,fz]=map(np.zeros,[Nx-2,Nx-2,Nx-2])
tag=0

for iter in range(0,Nt):
    # 조기상환일의 페이오프
    if iter==step[tag]:
        gx=np.min(np.where(x>=x0*strike_price[tag+1]))
        gy=np.min(np.where(y>=y0*strike_price[tag+1]))
        gz=np.min(np.where(z>=z0*strike_price[tag+1]))
        u[gx:Nx,gy:Ny,gz:Nz]=facevalue*(1+coupon_rate[tag+1])
        ku[gx:Nx,gy:Ny,gz:Nz]=facevalue*(1+coupon_rate[tag+1])
        tag += 1
    gx=np.min(np.where(x>=x0*kib))
    gy=np.min(np.where(y>=y0*kib))
    gz=np.min(np.where(z>=z0*kib))
    u[0:gx+1,:,:]=ku[0:gx+1,:,:];
    u[:,0:gy+1,:]=ku[:,0:gy+1,:];
```

```
    u[:,:,0:gz+1]=ku[:,:,0:gz+1]
    # OSM과 토마스 알고리즘을 이용하여 u값 계산
    # x축으로 풀기
    for j in range(1,Ny-1):
        for k in range(1,Nz-1):
            fx[0:Nx-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[1:Nx-1]*y[j]\
              *(u[2:Nx,j+1,k]-u[2:Nx,j-1,k]\
              -u[0:Nx-2,j+1,k]+u[0:Nx-2,j-1,k])\
              /(hx[0:Nx-2]*hy[j]+hx[1:Nx-1]\
              *hy[j]+hx[1:Nx-1]*hy[j-1]+hx[0:Nx-2]\
              *hy[j-1])+1/3*rho_zx*x_volatility\
              *z_volatility*x[1:Nx-1]*z[k]\
              *(u[2:Nx,j,k+1]-u[2:Nx,j,k-1]\
              -u[0:Nx-2,j,k+1]+u[0:Nx-2,j,k-1])\
              /(hx[0:Nx-2]*hz[k]+hx[1:Nx-1]*hz[k]\
              +hx[1:Nx-1]*hz[k-1]+hx[0:Nx-2]*hz[k-1])\
              +1/3*rho_yz*y_volatility*z_volatility\
              *y[j]*z[k]*(u[1:Nx-1,j+1,k+1]\
              -u[1:Nx-1,j+1,k-1]-u[1:Nx-1,j-1,k+1]\
              +u[1:Nx-1,j-1,k-1])/(hy[j-1]*hz[k]\
              +hy[j]*hz[k]+hy[j]*hz[k-1])+hy[j-1]\
              *hz[k-1]+u[1:Nx-1,j,k]/dt
            old_u[1:Nx-1,j,k]=thomas(ax,dx,cx,fx)

old_u[1:Nx-1,1:Ny-1,Nz-1]=2*old_u[1:Nx-1,1:Ny-1,Nz-2]-old_u[1:Nx-1,1:Ny-1
,Nz-3]

old_u[Nx-1,1:Ny-1,1:Nz]=2*old_u[Nx-2,1:Ny-1,1:Nz]-old_u[Nx-3,1:Ny-1,1:Nz]
    old_u[1:Nx,Ny-1,1:Nz]=2*old_u[1:Nx,Ny-2,1:Nz]-old_u[1:Nx,Ny-3,1:Nz]
    # y축으로 풀기
    for k in range(1,Nz-1):
        for i in range(1,Nx-1):
            fy[0:Ny-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[i]*y[1:Ny-1]\
              *(old_u[i+1,2:Ny,k]-old_u[i+1\
              ,0:Ny-2,k]-old_u[i-1,2:Ny,k]\
              +old_u[i-1,0:Ny-2,k])/(hx[i-1]\
              *hy[1:Ny-1]+hx[i]*hy[1:Ny-1]\
              +hx[i]*hy[0:Ny-2]+hx[i-1]\
              *hy[0:Ny-2])+1/3*rho_zx\
```

```
                *x_volatility*z_volatility\
                *x[i]*z[k]*(old_u[i+1,1:Ny-1,k+1]\
                -old_u[i+1,1:Ny-1,k-1]-old_u[i-1\
                ,1:Ny-1,k+1]+old_u[i-1,1:Ny-1,k-1])\
                /(hx[i-1]*hz[k]+hx[i]*hz[k]\
                +hx[i]*hz[k-1]+hx[i-1]*hz[k-1])\
                +1/3*rho_yz*y_volatility\
                *z_volatility*y[1:Ny-1]*z[k]\
                *(old_u[i,2:Ny,k+1]-old_u[i,2:Ny,k-1]\
                -old_u[i,0:Ny-2,k+1]+old_u[i,0:Ny-2\
                ,k-1])/(hy[0:Ny-2]*hz[k]+hy[1:Ny-1]\
                *hz[k]+hy[1:Ny-1]*hz[k-1]+hy[0:Ny-2]\
                *hz[k-1])+old_u[i,1:Ny-1,k]/dt
            u[i,1:Ny-1,k]=thomas(ay,dy,cy,fy)
    u[1:Nx-1,1:Ny-1,Nz-1]=2*u[1:Nx-1,1:Ny-1,Nz-2]-u[1:Nx-1,1:Ny-1,Nz-3]
    u[Nx-1,1:Ny-1,1:Nz]=2*u[Nx-2,1:Ny-1,1:Nz]-u[Nx-3,1:Ny-1,1:Nz]
    u[1:Nx,Ny-1,1:Nz]=2*u[1:Nx,Ny-2,1:Nz]-u[1:Nx,Ny-3,1:Nz]
    # z 축으로 풀기
    for j in range(1,Ny-1):
        for i in range(1,Nx-1):
            fz[0:Nz-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[i]*y[j]\
              *(u[i+1,j+1,1:Nz-1]-u[i+1,j-1,1:Nz-1]\
              -u[i-1,j+1,1:Nz-1]+u[i-1,j-1,1:Nz-1])\
              /(hx[i-1]*hy[j]+hx[i]*hy[j]+hx[i]\
              *hy[j-1]+hx[i-1]*hy[j-1])+1/3*rho_zx\
              *x_volatility*z_volatility*x[i]\
              *z[1:Nz-1]*(u[i+1,j,2:Nz]-u[i+1,j,0:Nz-2]-u[i-1,j,2:Nz]\
              +u[i-1,j,0:Nz-2])/(hx[i-1]*hz[1:Nz-1]\
              +hx[i]*hz[1:Nz-1]+hx[i]*hz[0:Nz-2]\
              +hx[i-1]*hz[0:Nz-2])+1/3*rho_yz\
              *y_volatility*z_volatility*y[j]*z[1:Nz-1]*(u[i,j+1,2:Nz]\
              -u[i,j+1,0:Nz-2]-u[i,j-1,2:Nz]+u[i,j-1,0:Nz-2])/(hy[j-1]\
              *hz[1:Nz-1]+hy[j]*hz[1:Nz-1]\
              +hy[j]*hz[0:Nz-2]+hy[j-1]*hz[0:Nz-2])\
              +u[i,j,1:Nz-1]/dt
            old_u[i,j,1:Nz-1]=thomas(az,dz,cz,fz)

old_u[1:Nx-1,1:Ny-1,Nz-1]=2*old_u[1:Nx-1,1:Ny-1,Nz-2]-old_u[1:Nx-1,1:Ny-1
,Nz-3]
```

```
old_u[Nx-1,1:Ny-1,1:Nz]=2*old_u[Nx-2,1:Ny-1,1:Nz]-old_u[Nx-3,1:Ny-1,1:Nz]
    old_u[1:Nx,Ny-1,1:Nz]=2*old_u[1:Nx,Ny-2,1:Nz]-old_u[1:Nx,Ny-3,1:Nz]
    u=old_u

    # OSM과 토마스 알고리즘을 이용하여 ku값 계산
    # x 축으로 풀기
    for j in range(1,Ny-1):
        for k in range(1,Nz-1):
            fx[0:Nx-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[1:Nx-1]*y[j]\
              *(ku[2:Nx,j+1,k]-ku[2:Nx,j-1,k]\
              -ku[0:Nx-2,j+1,k]+ku[0:Nx-2,j-1,k])\
              /(hx[0:Nx-2]*hy[j]+hx[1:Nx-1]*hy[j]\
              +hx[1:Nx-1]*hy[j-1]+hx[0:Nx-2]*hy[j-1])\
              +1/3*rho_zx*x_volatility*z_volatility\
              *x[1:Nx-1]*z[k]*(ku[2:Nx,j,k+1]\
              -ku[2:Nx,j,k-1]-ku[0:Nx-2,j,k+1]\
              +ku[0:Nx-2,j,k-1])/(hx[0:Nx-2]*hz[k]\
              +hx[1:Nx-1]*hz[k]+hx[1:Nx-1]*hz[k-1]\
              +hx[0:Nx-2]*hz[k-1])+1/3*rho_yz\
              *y_volatility*z_volatility*y[j]\
              *z[k]*(ku[1:Nx-1,j+1,k+1]\
              -ku[1:Nx-1,j+1,k-1]-ku[1:Nx-1,j-1,k+1]\
              +ku[1:Nx-1,j-1,k-1])/(hy[j-1]*hz[k]\
              +hy[j]*hz[k]+hy[j]*hz[k-1]+hy[j-1]\
              *hz[k-1])+ku[1:Nx-1,j,k]/dt
            old_ku[1:Nx-1,j,k]=thomas(ax,dx,cx,fx)

old_ku[1:Nx-1,1:Ny-1,Nz-1]=2*old_ku[1:Nx-1,1:Ny-1,Nz-2]-old_ku[1:Nx-1,1:
Ny-1,Nz-3]

old_ku[Nx-1,1:Ny-1,1:Nz]=2*old_ku[Nx-2,1:Ny-1,1:Nz]-old_ku[Nx-3,1:Ny-1,1:
Nz]

old_ku[1:Nx,Ny-1,1:Nz]=2*old_ku[1:Nx,Ny-2,1:Nz]-old_ku[1:Nx,Ny-3,1:Nz]
    # y 축으로 풀기
    for k in range(1,Nz-1):
        for i in range(1,Nx-1):
             fy[0:Ny-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[i]*y[1:Ny-1]\
              *(old_ku[i+1,2:Ny,k]-old_ku[i+1,0:Ny-2,\
```

```
                    k]-old_ku[i-1,2:Ny,k]+old_ku[i-1,0:Ny-2,k])/(hx[i-1]\
                    *hy[1:Ny-1]+hx[i]*hy[1:Ny-1]+hx[i]\
                    *hy[0:Ny-2]+hx[i-1]*hy[0:Ny-2])+1/3\
                    *rho_zx*x_volatility*z_volatility\
                    *x[i]*z[k]*(old_ku[i+1,1:Ny-1,k+1]\
                    -old_ku[i+1,1:Ny-1,k-1]-old_ku[i-1,\
                    1:Ny-1,k+1]+old_ku[i-1,1:Ny-1,k-1])/\
                    (hx[i-1]*hz[k]+hx[i]*hz[k]+hx[i]*hz[k-1]\
                    +hx[i-1]*hz[k-1])+1/3*rho_yz*\
                    y_volatility*z_volatility*y[1:Ny-1]*z[k]\
                    *(old_ku[i,2:Ny,k+1]-old_ku[i,2:Ny,k-1]\
                    -old_ku[i,0:Ny-2,k+1]+old_ku[i,0:Ny-2\
                    ,k-1])/(hy[0:Ny-2]*hz[k]+hy[1:Ny-1]\
                    *hz[k]+hy[1:Ny-1]*hz[k-1]+hy[0:Ny-2]\
                    *hz[k-1])+old_ku[i,1:Ny-1,k]/dt
                ku[i,1:Ny-1,k]=thomas(ay,dy,cy,fy)

ku[1:Nx-1,1:Ny-1,Nz-1]=2*ku[1:Nx-1,1:Ny-1,Nz-2]-ku[1:Nx-1,1:Ny-1,Nz-3]
    ku[Nx-1,1:Ny-1,1:Nz]=2*ku[Nx-2,1:Ny-1,1:Nz]-ku[Nx-3,1:Ny-1,1:Nz]
    ku[1:Nx,Ny-1,1:Nz]=2*ku[1:Nx,Ny-2,1:Nz]-ku[1:Nx,Ny-3,1:Nz]
    # z 축으로 풀기
    for j in range(1,Ny-1):
        for i in range(1,Nx-1):
            fz[0:Nz-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[i]*y[j]*(ku[i+1,j+1,1:Nz-1]\
              -ku[i+1,j-1,1:Nz-1]-ku[i-1,j+1,1:Nz-1]\
              +ku[i-1,j-1,1:Nz-1])/(hx[i-1]\
              *hy[j]+hx[i]*hy[j]+hx[i]*hy[j-1]\
              +hx[i-1]*hy[j-1])+1/3*rho_zx\
              *x_volatility*z_volatility*x[i]\
              *z[1:Nz-1]*(ku[i+1,j,2:Nz]\
              -ku[i+1,j,0:Nz-2]-ku[i-1,j,2:Nz]\
              +ku[i-1,j,0:Nz-2])/(hx[i-1]\
              *hz[1:Nz-1]+hx[i]*hz[1:Nz-1]\
              +hx[i]*hz[0:Nz-2]+hx[i-1]\
              *hz[0:Nz-2])+1/3*rho_yz\
              *y_volatility*z_volatility*y[j]\
              *z[1:Nz-1]*(ku[i,j+1,2:Nz]\
              -ku[i,j+1,0:Nz-2]-ku[i,j-1,2:Nz]\
              +ku[i,j-1,0:Nz-2])/(hy[j-1]\
              *hz[1:Nz-1]+hy[j]*hz[1:Nz-1]\
```

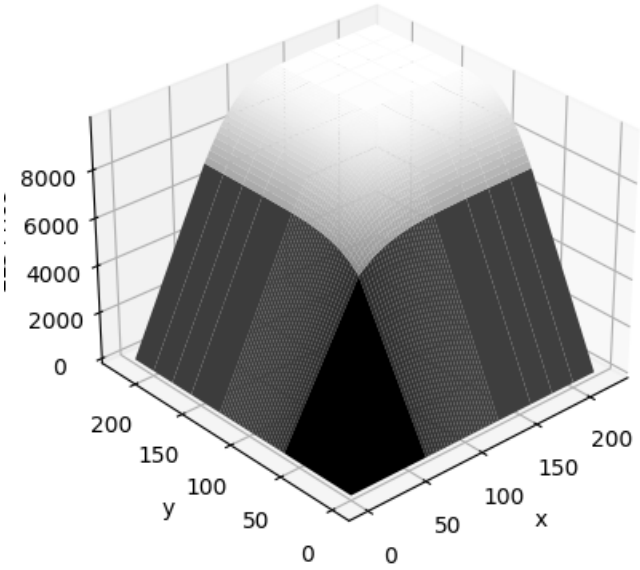
```
                    +hy[j]*hz[0:Nz-2]+hy[j-1]\
                    *hz[0:Nz-2])+ku[i,j,1:Nz-1]/dt
                old_ku[i,j,1:Nz-1]=thomas(az,dz,cz,fz)

old_ku[1:Nx-1,1:Ny-1,Nz-1]=2*old_ku[1:Nx-1,1:Ny-1,Nz-2]-old_ku[1:Nx-1,1:Ny-1,Nz-3]

old_ku[Nx-1,1:Ny-1,1:Nz]=2*old_ku[Nx-2,1:Ny-1,1:Nz]-old_ku[Nx-3,1:Ny-1,1:Nz]

old_ku[1:Nx,Ny-1,1:Nz]=2*old_ku[1:Nx,Ny-2,1:Nz]-old_ku[1:Nx,Ny-3,1:Nz]
    ku=old_ku

ii=np.where(x==100)
jj=np.where(y==100)
kk=np.where(z==100)
print('Price=%f'%(u[ii,jj,kk]))
```

Price=9037.842621 (ELS 최종가격)

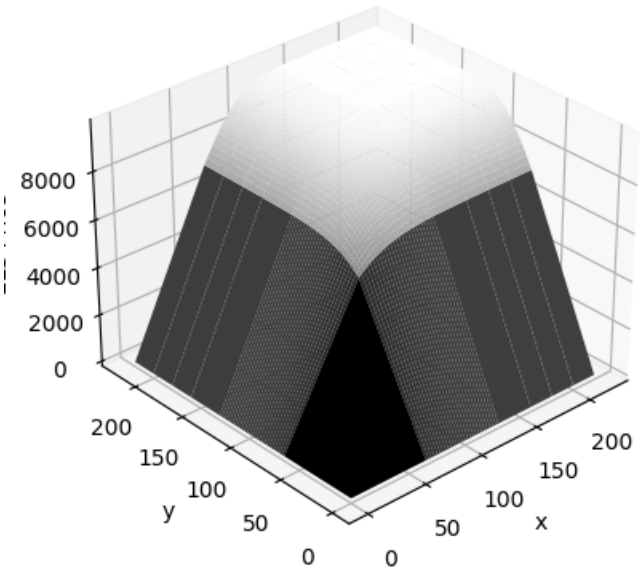
16

```
from matplotlib.figure import projections
X, Y = np.meshgrid(x,y)
kk=np.argwhere(z==100)

fig1=plt.figure()
ax=fig1.add_subplot(projection='3d')
ax.plot_surface(X,Y,u[:,:,int(kk)],cmap=plt.cm.gray)
ax.view_init(elev=30,azim=-132)
ax.set_xlabel('x', fontsize=10)
ax.set_ylabel('y', fontsize=10)
ax.zaxis.set_rotate_label(False)
ax.set_zlabel('ELS Price',rotation=90,fontsize=10)

fig2=plt.figure()
bx=fig2.add_subplot(projection='3d')
bx.plot_surface(X,Y,ku[:,:,int(kk)],cmap=plt.cm.gray)
bx.view_init(elev=30,azim=-132)
bx.set_xlabel('x', fontsize=10)
bx.set_ylabel('y', fontsize=10)
bx.zaxis.set_rotate_label(False)
bx.set_zlabel('ELS Price',rotation=90,fontsize=10)
```

	plt.show()



<낙인 배리어 아래로 기초자산이 떨어지지 않았을 때의 ELS가격)



<낙인 배리어 아래로 기초자산이 떨어졌을 때의 ELS가격)

Case 2

투자설명서 요약

- 기초자산 : EUROSTOXX50 지수, S&P500 지수, HSCEI 지수
- 기초자산가격 변동성 : EUROSTOXX50지수 : 28.1%, S&P500지수 : 33.51%, HSCEI지수 : 27,63%

  변동성 산출기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005))하여 Vola tility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용
- 기초자산간의 상관계수: EUROSTOXX50-S&P500 : 0.5678 EUROSTOXX50-HSCEI지수 : 0.3786, S&P500-HSCEI지수 : 0.2219

  상관계수 산출기준 : 180 영업일 역사적 상관계수
- 공정가격 : 본 증권의 공정가격은 2021년 03월 02일 기준 8,425.94원으로 추산됩니다.
- 최초기준가격결정일 : 2021년 03월 15일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2021년 09월 10일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 102.65%
2차	2022년 03월 10일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 105.30%
3차	2022년 09월 08일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 107.95%
4차	2023년 03월 10일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 110.60%
5차	2023년 09월 12일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 113.25%

- 만기평가일 : 2024년 03월 12일
- 최종기준가격 : 만기평가일 각 기초자산 종가 (현지시간 기준)
- 만기행사가격 : 각 기초자산 최초기준가격 × 75%
- 하락한계가격 : 각 기초자산 최초기준가격 × 47%
- 만기상환 조건

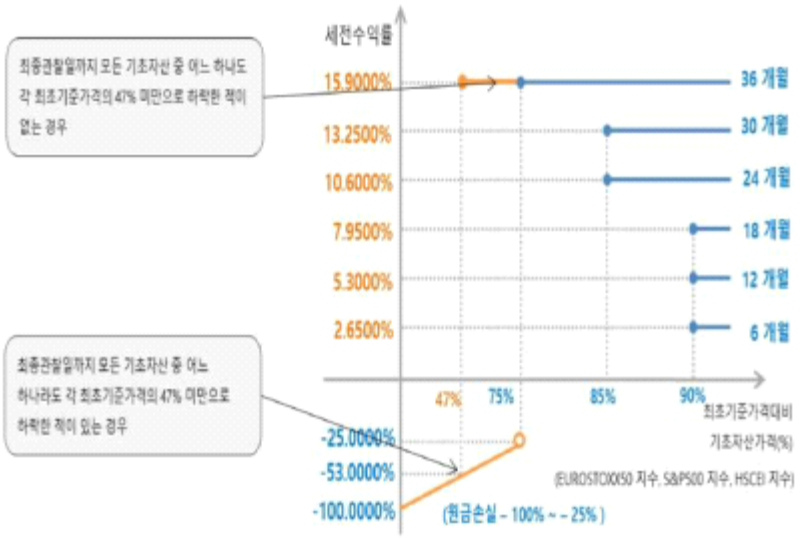
  ⓐ 모든 기초자산의 만기평가가격이 각 최초기준가격의 75% 이상인 경우: 액면금액 × 115.90%(연 5.30%)

  ⓑ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 각각의 하락한계가격보다 작게 된 적이 한 번도 없는 경우: 액면금액 × 115.90%(연 5.30%)

  ⓒ 위 ⓐ에 해당하지 않고, 최초기준가격 결정일(불포함) 이후 최종기준가격 결정일(포함) 이전까지 하나의 기초자산이라도 종가에 한 번이라도 각각의 하락한계 가격보다 작게 된 적이 있는 경우: 기준종목 기준으로 {(만기평가가격/최초기준가격)- 1} × 100%

  ※ 기준종목 : 모든 기초자산 중 [만기평가가격/최초기준가격]의 비율이 가장 낮은 기초자산

- 예상 손익구조 그래프



미래에셋대우 제29534회 (예상 손익구조 그래프)

- 파이썬 코드로 구현

18	import numpy as np
	import matplotlib.pyplot as plt
	from mpl_toolkits.mplot3d import Axes3D
	
	facevalue=10000;
	x_volatility=0.281; y_volatility=0.3351; z_volatility=0.2763 # 각각의 변동성
	rho_xy=0.5678; rho_yz=0.2219; rho_zx=0.3768; # 상관계수
	r=0.012; # 무위험 이자율
	T=3; # 만기
	x0=100; y0=100; z0=100; # 최초 기준가격
	# 조기상환시 쿠폰 이자율
	coupon_rate=np.array([0.159, 0.1325, 0.106, 0.0795, 0.053, 0.0265])
	# 조기행사가
	strike_price=np.array([0.75, 0.85, 0.85, 0.90, 0.90, 0.90])
	dummy=0.159; kib=0.47; # 더미 이자율, 낙인 배리어

19	Nt=360*T; dt=T/Nt; # 시간격자 갯수, 시간격자 간격
	A=np.array([0]); B=np.arange(65,132.5,2.5); C=np.array([160,180,200,220]);
	x=np.r_[A,B,C]; y=x; z=x;
	# x, y, z벡터의 크기
	Nx=len(x); Ny=Nx; Nz=Nx;

```
hx=np.diff(x); hy=hx; hz=hx;
step=np.arange(1,8,1)*Nt/6
lst=[Nx,Ny,Nz]
[u,ku]=map(np.zeros,[lst,lst])
# 유한차분법으로 ELS 가격을 구하기 위한 초기값
for i in range(0,Nx):
    for j in range(0,Ny):
        for k in range(0,Nz):
            if (x[i]<=kib*x0 or y[j]<=kib*y0 or z[k]<=kib*z0):
                u[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
                ku[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
            elif (x[i]<strike_price[0]*x0 or y[j]<strike_price[0]*y0 or
z[k]<strike_price[0]*z0):
                u[i,j,k]=facevalue*(1.0+dummy)
                ku[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
            else:
                u[i,j,k]=facevalue*(1+coupon_rate[0])
                ku[i,j,k]=facevalue*(1+coupon_rate[0])
# 유한차분법을 사용하기 위한 계수
[ax,dx,cx,ay,dy,cy,az,dz,cz]=map(np.zeros,[Nx-2,Nx-2,Nx-2,Ny-2,Ny-2,Ny-2,
Nz-2,Nz-2,Nz-2])
ax[:]=(-(x_volatility*x[1:Nx-1])**2+r*x[1:Nx-1]*hx[1:Nx-1])/(hx[0:Nx-2]*(hx[0:
Nx-2]+hx[1:Nx-1]))
dx[:]=1/dt+(x_volatility*x[1:Nx-1])**2/(hx[0:Nx-2]*hx[1:Nx-1])-r*x[1:Nx-1]*(h
x[1:Nx-1]-hx[0:Nx-2])/(hx[0:Nx-2]*hx[1:Nx-1])+r/3
cx[:]=-((x_volatility*x[1:Nx-1])**2+r*x[1:Nx-1]*hx[0:Nx-2])/(hx[1:Nx-1]*(hx[0:
Nx-2]+hx[1:Nx-1]))
# 선형 경계조건
ax[Nx-3]=ax[Nx-3]-cx[Nx-3]
dx[Nx-3]=dx[Nx-3]+2*cx[Nx-3]
ay[:]=(-(y_volatility*y[1:Ny-1])**2+r*y[1:Ny-1]*hy[1:Ny-1])/(hy[0:Ny-2]*(hy[0:
Ny-2]+hy[1:Ny-1]))
dy[:]=1/dt+(y_volatility*y[1:Ny-1])**2/(hy[0:Ny-2]*hy[1:Ny-1])-r*y[1:Ny-1]*(h
y[1:Ny-1]-hy[0:Ny-2])/(hy[0:Ny-2]*hy[1:Ny-1])+r/3
cy[:]=-((y_volatility*y[1:Ny-1])**2+r*y[1:Ny-1]*hy[0:Ny-2])/(hy[1:Ny-1]*(hy[0:
Ny-2]+hy[1:Ny-1]))
ay[Ny-3]=ay[Ny-3]-cy[Ny-3]
dy[Ny-3]=dy[Ny-3]+2*cy[Ny-3]
az[:]=(-(z_volatility*z[1:Nz-1])**2+r*z[1:Nz-1]*hz[1:Nz-1])/(hz[0:Nz-2]*(hz[0:N
z-2]+hz[1:Nz-1]))
dz[:]=1/dt+(z_volatility*z[1:Nz-1])**2/(hz[0:Nz-2]*hz[1:Nz-1])-r*z[1:Nz-1]*(hz[
```

```
1:Nz-1]-hz[0:Nz-2])/(hz[0:Nz-2]*hz[1:Nz-1])+r/3
cz[:]=-((z_volatility*z[1:Nz-1])**2+r*z[1:Nz-1]*hz[0:Nz-2])/(hz[1:Nz-1]*(hz[0:N
z-2]+hz[1:Nz-1]))
az[Nz-3]=az[Nz-3]-cz[Nz-3]
dz[Nz-3]=dz[Nz-3]+2*cz[Nz-3]
# OS방법을 사용하기 위해 u, ku와 같은 초기의 행렬 생성
[old_u,old_ku]=map(np.zeros,[lst,lst])
[fx,fy,fz]=map(np.zeros,[Nx-2,Nx-2,Nx-2])
tag=0

for iter in range(0,Nt):
    # 조기상환일의 페이오프
    if iter==step[tag]:
        gx=np.min(np.where(x>=x0*strike_price[tag+1]))
        gy=np.min(np.where(y>=y0*strike_price[tag+1]))
        gz=np.min(np.where(z>=z0*strike_price[tag+1]))
        u[gx:Nx,gy:Ny,gz:Nz]=facevalue*(1+coupon_rate[tag+1])
        ku[gx:Nx,gy:Ny,gz:Nz]=facevalue*(1+coupon_rate[tag+1])
        tag += 1
    gx=np.min(np.where(x>=x0*kib))
    gy=np.min(np.where(y>=y0*kib))
    gz=np.min(np.where(z>=z0*kib))
    u[0:gx+1,:,:]=ku[0:gx+1,:,:];
    u[:,0:gy+1,:]=ku[:,0:gy+1,:];
    u[:,:,0:gz+1]=ku[:,:,0:gz+1]
    # OSM과 토마스 알고리즘을 이용하여 u값 계산
    # x축으로 풀기
    for j in range(1,Ny-1):
        for k in range(1,Nz-1):
            fx[0:Nx-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[1:Nx-1]*y[j]\
              *(u[2:Nx,j+1,k]-u[2:Nx,j-1,k]\
              -u[0:Nx-2,j+1,k]+u[0:Nx-2,j-1,k])\
              /(hx[0:Nx-2]*hy[j]+hx[1:Nx-1]\
              *hy[j]+hx[1:Nx-1]*hy[j-1]+hx[0:Nx-2]\
              *hy[j-1])+1/3*rho_zx*x_volatility\
              *z_volatility*x[1:Nx-1]*z[k]\
              *(u[2:Nx,j,k+1]-u[2:Nx,j,k-1]\
              -u[0:Nx-2,j,k+1]+u[0:Nx-2,j,k-1])\
              /(hx[0:Nx-2]*hz[k]+hx[1:Nx-1]*hz[k]\
              +hx[1:Nx-1]*hz[k-1]+hx[0:Nx-2]*hz[k-1])\
```

```
                    +1/3*rho_yz*y_volatility*z_volatility\
                    *y[j]*z[k]*(u[1:Nx-1,j+1,k+1]\
                    -u[1:Nx-1,j+1,k-1]-u[1:Nx-1,j-1,k+1]\
                    +u[1:Nx-1,j-1,k-1])/(hy[j-1]*hz[k]\
                    +hy[j]*hz[k]+hy[j]*hz[k-1])+hy[j-1]\
                    *hz[k-1]+u[1:Nx-1,j,k]/dt
                old_u[1:Nx-1,j,k]=thomas(ax,dx,cx,fx)

old_u[1:Nx-1,1:Ny-1,Nz-1]=2*old_u[1:Nx-1,1:Ny-1,Nz-2]-old_u[1:Nx-1,1:Ny-1
,Nz-3]

old_u[Nx-1,1:Ny-1,1:Nz]=2*old_u[Nx-2,1:Ny-1,1:Nz]-old_u[Nx-3,1:Ny-1,1:Nz]
    old_u[1:Nx,Ny-1,1:Nz]=2*old_u[1:Nx,Ny-2,1:Nz]-old_u[1:Nx,Ny-3,1:Nz]
    # y축으로 풀기
    for k in range(1,Nz-1):
        for i in range(1,Nx-1):
            fy[0:Ny-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[i]*y[1:Ny-1]\
              *(old_u[i+1,2:Ny,k]-old_u[i+1\
              ,0:Ny-2,k]-old_u[i-1,2:Ny,k]\
              +old_u[i-1,0:Ny-2,k])/(hx[i-1]\
              *hy[1:Ny-1]+hx[i]*hy[1:Ny-1]\
              +hx[i]*hy[0:Ny-2]+hx[i-1]\
              *hy[0:Ny-2])+1/3*rho_zx\
              *x_volatility*z_volatility\
              *x[i]*z[k]*(old_u[i+1,1:Ny-1,k+1]\
              -old_u[i+1,1:Ny-1,k-1]-old_u[i-1\
              ,1:Ny-1,k+1]+old_u[i-1,1:Ny-1,k-1])\
              /(hx[i-1]*hz[k]+hx[i]*hz[k]\
              +hx[i]*hz[k-1]+hx[i-1]*hz[k-1])\
              +1/3*rho_yz*y_volatility\
              *z_volatility*y[1:Ny-1]*z[k]\
              *(old_u[i,2:Ny,k+1]-old_u[i,2:Ny,k-1]\
              -old_u[i,0:Ny-2,k+1]+old_u[i,0:Ny-2\
              ,k-1])/(hy[0:Ny-2]*hz[k]+hy[1:Ny-1]\
              *hz[k]+hy[1:Ny-1]*hz[k-1]+hy[0:Ny-2]\
              *hz[k-1])+old_u[i,1:Ny-1,k]/dt
            u[i,1:Ny-1,k]=thomas(ay,dy,cy,fy)
    u[1:Nx-1,1:Ny-1,Nz-1]=2*u[1:Nx-1,1:Ny-1,Nz-2]-u[1:Nx-1,1:Ny-1,Nz-3]
    u[Nx-1,1:Ny-1,1:Nz]=2*u[Nx-2,1:Ny-1,1:Nz]-u[Nx-3,1:Ny-1,1:Nz]
    u[1:Nx,Ny-1,1:Nz]=2*u[1:Nx,Ny-2,1:Nz]-u[1:Nx,Ny-3,1:Nz]
```

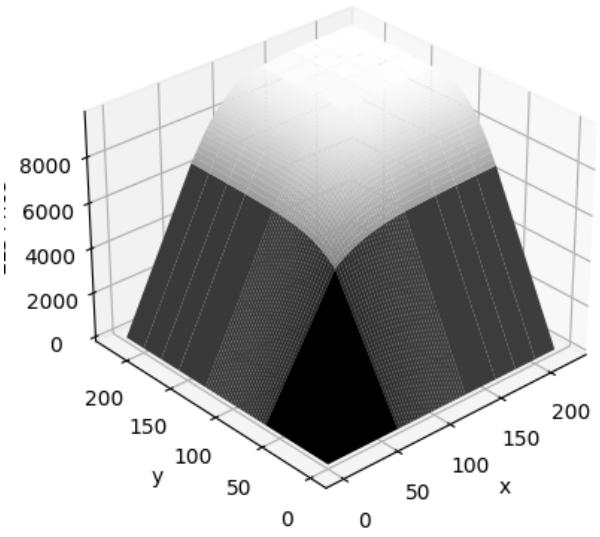
```
    # z 축으로 풀기
    for j in range(1,Ny-1):
        for i in range(1,Nx-1):
            fz[0:Nz-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[i]*y[j]\
              *(u[i+1,j+1,1:Nz-1]-u[i+1,j-1,1:Nz-1]\
              -u[i-1,j+1,1:Nz-1]+u[i-1,j-1,1:Nz-1])\
              /(hx[i-1]*hy[j]+hx[i]*hy[j]+hx[i]\
              *hy[j-1]+hx[i-1]*hy[j-1])+1/3*rho_zx\
              *x_volatility*z_volatility*x[i]\
              *z[1:Nz-1]*(u[i+1,j,2:Nz]-u[i+1,j,0:Nz-2]-u[i-1,j,2:Nz]\
              +u[i-1,j,0:Nz-2])/(hx[i-1]*hz[1:Nz-1]\
              +hx[i]*hz[1:Nz-1]+hx[i]*hz[0:Nz-2]\
              +hx[i-1]*hz[0:Nz-2])+1/3*rho_yz\
              *y_volatility*z_volatility*y[j]*z[1:Nz-1]*(u[i,j+1,2:Nz]\
              -u[i,j+1,0:Nz-2]-u[i,j-1,2:Nz]+u[i,j-1,0:Nz-2])/(hy[j-1]\
              *hz[1:Nz-1]+hy[j]*hz[1:Nz-1]\
              +hy[j]*hz[0:Nz-2]+hy[j-1]*hz[0:Nz-2])\
              +u[i,j,1:Nz-1]/dt
            old_u[i,j,1:Nz-1]=thomas(az,dz,cz,fz)

old_u[1:Nx-1,1:Ny-1,Nz-1]=2*old_u[1:Nx-1,1:Ny-1,Nz-2]-old_u[1:Nx-1,1:Ny-1
,Nz-3]

old_u[Nx-1,1:Ny-1,1:Nz]=2*old_u[Nx-2,1:Ny-1,1:Nz]-old_u[Nx-3,1:Ny-1,1:Nz]
    old_u[1:Nx,Ny-1,1:Nz]=2*old_u[1:Nx,Ny-2,1:Nz]-old_u[1:Nx,Ny-3,1:Nz]
    u=old_u

    # OSM과 토마스 알고리즘을 이용하여 ku값 계산
    # x 축으로 풀기
    for j in range(1,Ny-1):
        for k in range(1,Nz-1):
            fx[0:Nx-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[1:Nx-1]*y[j]\
              *(ku[2:Nx,j+1,k]-ku[2:Nx,j-1,k]\
              -ku[0:Nx-2,j+1,k]+ku[0:Nx-2,j-1,k])\
              /(hx[0:Nx-2]*hy[j]+hx[1:Nx-1]*hy[j]\
              +hx[1:Nx-1]*hy[j-1]+hx[0:Nx-2]*hy[j-1])\
              +1/3*rho_zx*x_volatility*z_volatility\
              *x[1:Nx-1]*z[k]*(ku[2:Nx,j,k+1]\
              -ku[2:Nx,j,k-1]-ku[0:Nx-2,j,k+1]\
```

```
                    +ku[0:Nx-2,j,k-1])/(hx[0:Nx-2]*hz[k]\
                    +hx[1:Nx-1]*hz[k]+hx[1:Nx-1]*hz[k-1]\
                    +hx[0:Nx-2]*hz[k-1])+1/3*rho_yz\
                    *y_volatility*z_volatility*y[j]\
                    *z[k]*(ku[1:Nx-1,j+1,k+1]\
                    -ku[1:Nx-1,j+1,k-1]-ku[1:Nx-1,j-1,k+1]\
                    +ku[1:Nx-1,j-1,k-1])/(hy[j-1]*hz[k]\
                    +hy[j]*hz[k]+hy[j]*hz[k-1]+hy[j-1]\
                    *hz[k-1])+ku[1:Nx-1,j,k]/dt
                old_ku[1:Nx-1,j,k]=thomas(ax,dx,cx,fx)

old_ku[1:Nx-1,1:Ny-1,Nz-1]=2*old_ku[1:Nx-1,1:Ny-1,Nz-2]-old_ku[1:Nx-1,1:
Ny-1,Nz-3]

old_ku[Nx-1,1:Ny-1,1:Nz]=2*old_ku[Nx-2,1:Ny-1,1:Nz]-old_ku[Nx-3,1:Ny-1,1:
Nz]

old_ku[1:Nx,Ny-1,1:Nz]=2*old_ku[1:Nx,Ny-2,1:Nz]-old_ku[1:Nx,Ny-3,1:Nz]
    # y 축으로 풀기
    for k in range(1,Nz-1):
        for i in range(1,Nx-1):
            fy[0:Ny-1]=1/3*rho_xy*x_volatility\
             *y_volatility*x[i]*y[1:Ny-1]\
             *(old_ku[i+1,2:Ny,k]-old_ku[i+1,0:Ny-2,\
             k]-old_ku[i-1,2:Ny,k]+old_ku[i-1,0:Ny-2,k])/(hx[i-1]\
             *hy[1:Ny-1]+hx[i]*hy[1:Ny-1]+hx[i]\
             *hy[0:Ny-2]+hx[i-1]*hy[0:Ny-2])+1/3\
             *rho_zx*x_volatility*z_volatility\
             *x[i]*z[k]*(old_ku[i+1,1:Ny-1,k+1]\
             -old_ku[i+1,1:Ny-1,k-1]-old_ku[i-1,\
             1:Ny-1,k+1]+old_ku[i-1,1:Ny-1,k-1])/\
             (hx[i-1]*hz[k]+hx[i]*hz[k]+hx[i]*hz[k-1]\
             +hx[i-1]*hz[k-1])+1/3*rho_yz*\
             y_volatility*z_volatility*y[1:Ny-1]*z[k]\
             *(old_ku[i,2:Ny,k+1]-old_ku[i,2:Ny,k-1]\
             -old_ku[i,0:Ny-2,k+1]+old_ku[i,0:Ny-2\
             ,k-1])/(hy[0:Ny-2]*hz[k]+hy[1:Ny-1]\
             *hz[k]+hy[1:Ny-1]*hz[k-1]+hy[0:Ny-2]\
             *hz[k-1])+old_ku[i,1:Ny-1,k]/dt
            ku[i,1:Ny-1,k]=thomas(ay,dy,cy,fy)
```

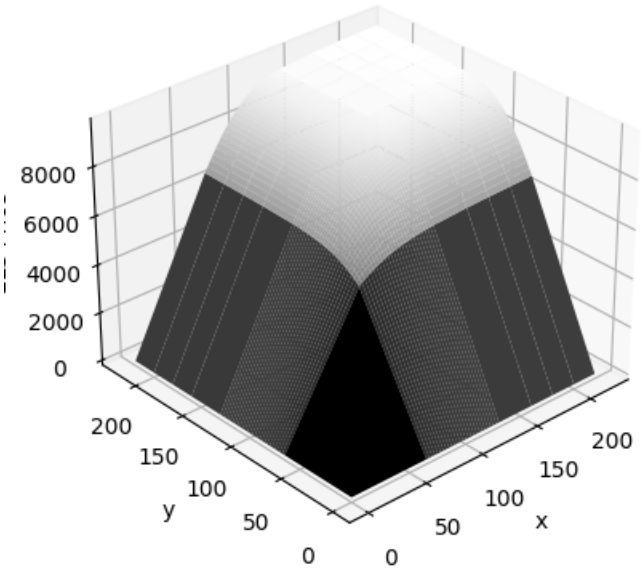
```
ku[1:Nx-1,1:Ny-1,Nz-1]=2*ku[1:Nx-1,1:Ny-1,Nz-2]-ku[1:Nx-1,1:Ny-1,Nz-3]
    ku[Nx-1,1:Ny-1,1:Nz]=2*ku[Nx-2,1:Ny-1,1:Nz]-ku[Nx-3,1:Ny-1,1:Nz]
    ku[1:Nx,Ny-1,1:Nz]=2*ku[1:Nx,Ny-2,1:Nz]-ku[1:Nx,Ny-3,1:Nz]
    # z 축으로 풀기
    for j in range(1,Ny-1):
        for i in range(1,Nx-1):
            fz[0:Nz-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[i]*y[j]*(ku[i+1,j+1,1:Nz-1]\
              -ku[i+1,j-1,1:Nz-1]-ku[i-1,j+1,1:Nz-1]\
              +ku[i-1,j-1,1:Nz-1])/(hx[i-1]\
              *hy[j]+hx[i]*hy[j]+hx[i]*hy[j-1]\
              +hx[i-1]*hy[j-1])+1/3*rho_zx\
              *x_volatility*z_volatility*x[i]\
              *z[1:Nz-1]*(ku[i+1,j,2:Nz]\
              -ku[i+1,j,0:Nz-2]-ku[i-1,j,2:Nz]\
              +ku[i-1,j,0:Nz-2])/(hx[i-1]\
              *hz[1:Nz-1]+hx[i]*hz[1:Nz-1]\
              +hx[i]*hz[0:Nz-2]+hx[i-1]\
              *hz[0:Nz-2])+1/3*rho_yz\
              *y_volatility*z_volatility*y[j]\
              *z[1:Nz-1]*(ku[i,j+1,2:Nz]\
              -ku[i,j+1,0:Nz-2]-ku[i,j-1,2:Nz]\
              +ku[i,j-1,0:Nz-2])/(hy[j-1]\
              *hz[1:Nz-1]+hy[j]*hz[1:Nz-1]\
              +hy[j]*hz[0:Nz-2]+hy[j-1]\
              *hz[0:Nz-2])+ku[i,j,1:Nz-1]/dt
            old_ku[i,j,1:Nz-1]=thomas(az,dz,cz,fz)

old_ku[1:Nx-1,1:Ny-1,Nz-1]=2*old_ku[1:Nx-1,1:Ny-1,Nz-2]-old_ku[1:Nx-1,1:
Ny-1,Nz-3]

old_ku[Nx-1,1:Ny-1,1:Nz]=2*old_ku[Nx-2,1:Ny-1,1:Nz]-old_ku[Nx-3,1:Ny-1,1:
Nz]

old_ku[1:Nx,Ny-1,1:Nz]=2*old_ku[1:Nx,Ny-2,1:Nz]-old_ku[1:Nx,Ny-3,1:Nz]
    ku=old_ku

ii=np.where(x==100)
jj=np.where(y==100)
kk=np.where(z==100)
print('Price=%f'%(u[ii,jj,kk]))
```

Price=8421.508776 (ELS 최종가격)

20	from matplotlib.figure import projections X, Y = np.meshgrid(x,y) kk=np.argwhere(z==100) fig1=plt.figure() ax=fig1.add_subplot(projection='3d') ax.plot_surface(X,Y,u[:,:,int(kk)],cmap=plt.cm.gray) ax.view_init(elev=30,azim=-132) ax.set_xlabel('x', fontsize=10) ax.set_ylabel('y', fontsize=10) ax.zaxis.set_rotate_label(False) ax.set_zlabel('ELS Price',rotation=90,fontsize=10) fig2=plt.figure() bx=fig2.add_subplot(projection='3d') bx.plot_surface(X,Y,ku[:,:,int(kk)],cmap=plt.cm.gray) bx.view_init(elev=30,azim=-132) bx.set_xlabel('x', fontsize=10) bx.set_ylabel('y', fontsize=10) bx.zaxis.set_rotate_label(False) bx.set_zlabel('ELS Price',rotation=90,fontsize=10) plt.show()



<낙인 배리어 아래로 기초자산이 떨어지지 않았을 때의 ELS가격)



<낙인 배리어 아래로 기초자산이 떨어졌을 때의 ELS가격)

Case 3

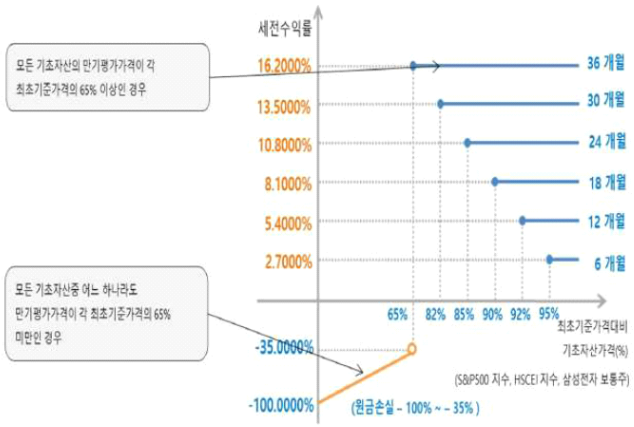
투자설명서 요약

- 기초자산 : S&P500 지수, HSCEI 지수, 삼성전자 보통주
- 기초자산가격 변동성 : S&P500지수 : 30,05%, HSCEI지수 : 27,33%, 삼성전자 : 31.54%
  변동성 산출기준: Volatility Surface에 대하여 VIX방법론을 이용(Jiang and Tian(2005))하여 Vola tility Term Structure를 산출한 뒤 해당만기에 상응하는 변동성을 적용
- 기초자산간의 상관계수: S&P500-HSCEI지수 : 0.1952, S&P500-삼성전자 : 0.101, HSCEI지수-삼성전자 : 0.4479
  상관계수 산출기준 : 180 영업일 역사적 상관계수
- 공정가격 : 본 증권의 공정가격은 2021년 04월 28일 기준 7,814.84원으로 추산됩니다.
- 최초기준가격결정일 : 2021년 05월 07일
- 자동조기상환평가일 및 상환금액

차수	중간기준가격 결정일	자동조기상환 조건	상환금액
1차	2021년 11월 02일	세 지수 모두 최초기준가격의 95%이상	액면금액 × 102.70%
2차	2022년 04월 29일	세 지수 모두 최초기준가격의 92%이상	액면금액 × 105.40%
3차	2022년 11월 02일	세 지수 모두 최초기준가격의 90%이상	액면금액 × 108.10%
4차	2023년 04월 28일	세 지수 모두 최초기준가격의 85%이상	액면금액 × 110.80%
5차	2023년 11월 02일	세 지수 모두 최초기준가격의 82%이상	액면금액 × 113.50%

- 만기평가일 : 2024년 04월 30일
- 최종기준가격 : 만기평가일 각 기초자산 종가 (현지시간 기준)
- 만기행사가격 : 각 기초자산 최초기준가격 × 65%
- 만기상환 조건
  ⓐ 모든 기초자산의 만기평가가격이 각 최초기준가격의 65% 이상인 경우: 액면금액 × 116.20%(연 5.30%)
  ⓑ 모든 기초자산 중 어느 하나라도 만기평가가격이 각 최초 기준가격의 65% 미만인 경우: 기준종목 기준으로 {(만기평가가격/최초기준가격)- 1} × 100%
  ※ 기준종목 : 모든 기초자산 중 [만기평가가격/최초기준가격]의 비율이 가장 낮은 기초자산

- 예상 손익구조 그래프



미래에셋증권 제29629회 (예상 손익구조 그래프)

- 파이썬 코드로 구현

21

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

facevalue=10000;
x_volatility=0.3005; y_volatility=0.2733; z_volatility=0.3154 # 각각의 변동성
rho_xy=0.1952; rho_yz=0.4479; rho_zx=0.101; # 상관계수
r=0.012;  # 무위험 이자율
T=3; # 만기
x0=100; y0=100; z0=100;  # 최초 기준가격
# 조기상환시 쿠폰 이자율
coupon_rate=np.array([0.162, 0.1135, 0.108, 0.081, 0.054, 0.027])
# 조기행사가
strike_price=np.array([0.65, 0.82, 0.85, 0.90, 0.92, 0.95])
dummy=0.162; kib=0.65; # 더미 이자율, 낙인 배리어
```

22

```
Nt=360*T; dt=T/Nt; # 시간격자 갯수, 시간격자 간격
A=np.array([0]); B=np.arange(65,132.5,2.5); C=np.array([160,180,200,220]);
x=np.r_[A,B,C]; y=x; z=x;
# x, y, z벡터의 크기
Nx=len(x); Ny=Nx; Nz=Nx;
hx=np.diff(x); hy=hx; hz=hx;
step=np.arange(1,8,1)*Nt/6
lst=[Nx,Ny,Nz]
[u,ku]=map(np.zeros,[lst,lst])
# 유한차분법으로 ELS 가격을 구하기 위한 초기값
for i in range(0,Nx):
    for j in range(0,Ny):
        for k in range(0,Nz):
            if (x[i]<=kib*x0 or y[j]<=kib*y0 or z[k]<=kib*z0):
                u[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
                ku[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
            elif (x[i]<strike_price[0]*x0 or y[j]<strike_price[0]*y0 or
z[k]<strike_price[0]*z0):
                u[i,j,k]=facevalue*(1.0+dummy)
                ku[i,j,k]=np.min([x[i],y[j],z[k]])/x0*facevalue
            else:
                u[i,j,k]=facevalue*(1+coupon_rate[0])
```

```
                    ku[i,j,k]=facevalue*(1+coupon_rate[0])
# 유한차분법을 사용하기 위한 계수
[ax,dx,cx,ay,dy,cy,az,dz,cz]=map(np.zeros,[Nx-2,Nx-2,Nx-2,Ny-2,Ny-2,Ny-2,
Nz-2,Nz-2,Nz-2])
ax[:]=(-(x_volatility*x[1:Nx-1])**2+r*x[1:Nx-1]*hx[1:Nx-1])/(hx[0:Nx-2]*(hx[0:
Nx-2]+hx[1:Nx-1]))
dx[:]=1/dt+(x_volatility*x[1:Nx-1])**2/(hx[0:Nx-2]*hx[1:Nx-1])-r*x[1:Nx-1]*(h
x[1:Nx-1]-hx[0:Nx-2])/(hx[0:Nx-2]*hx[1:Nx-1])+r/3
cx[:]=-((x_volatility*x[1:Nx-1])**2+r*x[1:Nx-1]*hx[0:Nx-2])/(hx[1:Nx-1]*(hx[0:
Nx-2]+hx[1:Nx-1]))
# 선형 경계조건
ax[Nx-3]=ax[Nx-3]-cx[Nx-3]
dx[Nx-3]=dx[Nx-3]+2*cx[Nx-3]
ay[:]=(-(y_volatility*y[1:Ny-1])**2+r*y[1:Ny-1]*hy[1:Ny-1])/(hy[0:Ny-2]*(hy[0:
Ny-2]+hy[1:Ny-1]))
dy[:]=1/dt+(y_volatility*y[1:Ny-1])**2/(hy[0:Ny-2]*hy[1:Ny-1])-r*y[1:Ny-1]*(h
y[1:Ny-1]-hy[0:Ny-2])/(hy[0:Ny-2]*hy[1:Ny-1])+r/3
cy[:]=-((y_volatility*y[1:Ny-1])**2+r*y[1:Ny-1]*hy[0:Ny-2])/(hy[1:Ny-1]*(hy[0:
Ny-2]+hy[1:Ny-1]))
ay[Ny-3]=ay[Ny-3]-cy[Ny-3]
dy[Ny-3]=dy[Ny-3]+2*cy[Ny-3]
az[:]=(-(z_volatility*z[1:Nz-1])**2+r*z[1:Nz-1]*hz[1:Nz-1])/(hz[0:Nz-2]*(hz[0:N
z-2]+hz[1:Nz-1]))
dz[:]=1/dt+(z_volatility*z[1:Nz-1])**2/(hz[0:Nz-2]*hz[1:Nz-1])-r*z[1:Nz-1]*(hz[
1:Nz-1]-hz[0:Nz-2])/(hz[0:Nz-2]*hz[1:Nz-1])+r/3
cz[:]=-((z_volatility*z[1:Nz-1])**2+r*z[1:Nz-1]*hz[0:Nz-2])/(hz[1:Nz-1]*(hz[0:N
z-2]+hz[1:Nz-1]))
az[Nz-3]=az[Nz-3]-cz[Nz-3]
dz[Nz-3]=dz[Nz-3]+2*cz[Nz-3]
# OS방법을 사용하기 위해 u, ku와 같은 초기의 행렬 생성
[old_u,old_ku]=map(np.zeros,[lst,lst])
[fx,fy,fz]=map(np.zeros,[Nx-2,Nx-2,Nx-2])
tag=0

for iter in range(0,Nt):
    # 조기상환일의 페이오프
    if iter==step[tag]:
        gx=np.min(np.where(x>=x0*strike_price[tag+1]))
        gy=np.min(np.where(y>=y0*strike_price[tag+1]))
        gz=np.min(np.where(z>=z0*strike_price[tag+1]))
        u[gx:Nx,gy:Ny,gz:Nz]=facevalue*(1+coupon_rate[tag+1])
```

```
            ku[gx:Nx,gy:Ny,gz:Nz]=facevalue*(1+coupon_rate[tag+1])
            tag += 1
        gx=np.min(np.where(x>=x0*kib))
        gy=np.min(np.where(y>=y0*kib))
        gz=np.min(np.where(z>=z0*kib))
        u[0:gx+1,:,:]=ku[0:gx+1,:,:];
        u[:,0:gy+1,:]=ku[:,0:gy+1,:];
        u[:,:,0:gz+1]=ku[:,:,0:gz+1]
        # OSM과 토마스 알고리즘을 이용하여 u값 계산
        # x축으로 풀기
        for j in range(1,Ny-1):
            for k in range(1,Nz-1):
                fx[0:Nx-1]=1/3*rho_xy*x_volatility\
                  *y_volatility*x[1:Nx-1]*y[j]\
                  *(u[2:Nx,j+1,k]-u[2:Nx,j-1,k]\
                  -u[0:Nx-2,j+1,k]+u[0:Nx-2,j-1,k])\
                  /(hx[0:Nx-2]*hy[j]+hx[1:Nx-1]\
                  *hy[j]+hx[1:Nx-1]*hy[j-1]+hx[0:Nx-2]\
                  *hy[j-1])+1/3*rho_zx*x_volatility\
                  *z_volatility*x[1:Nx-1]*z[k]\
                  *(u[2:Nx,j,k+1]-u[2:Nx,j,k-1]\
                  -u[0:Nx-2,j,k+1]+u[0:Nx-2,j,k-1])\
                  /(hx[0:Nx-2]*hz[k]+hx[1:Nx-1]*hz[k]\
                  +hx[1:Nx-1]*hz[k-1]+hx[0:Nx-2]*hz[k-1])\
                  +1/3*rho_yz*y_volatility*z_volatility\
                  *y[j]*z[k]*(u[1:Nx-1,j+1,k+1]\
                  -u[1:Nx-1,j+1,k-1]-u[1:Nx-1,j-1,k+1]\
                  +u[1:Nx-1,j-1,k-1])/(hy[j-1]*hz[k]\
                  +hy[j]*hz[k]+hy[j]*hz[k-1])+hy[j-1]\
                  *hz[k-1]+u[1:Nx-1,j,k]/dt
                old_u[1:Nx-1,j,k]=thomas(ax,dx,cx,fx)

old_u[1:Nx-1,1:Ny-1,Nz-1]=2*old_u[1:Nx-1,1:Ny-1,Nz-2]-old_u[1:Nx-1,1:Ny-1
,Nz-3]

old_u[Nx-1,1:Ny-1,1:Nz]=2*old_u[Nx-2,1:Ny-1,1:Nz]-old_u[Nx-3,1:Ny-1,1:Nz]
        old_u[1:Nx,Ny-1,1:Nz]=2*old_u[1:Nx,Ny-2,1:Nz]-old_u[1:Nx,Ny-3,1:Nz]
        # y축으로 풀기
        for k in range(1,Nz-1):
            for i in range(1,Nx-1):
                fy[0:Ny-1]=1/3*rho_xy*x_volatility\
```

```
                *y_volatility*x[i]*y[1:Ny-1]\
                *(old_u[i+1,2:Ny,k]-old_u[i+1\
                ,0:Ny-2,k]-old_u[i-1,2:Ny,k]\
                +old_u[i-1,0:Ny-2,k])/(hx[i-1]\
                *hy[1:Ny-1]+hx[i]*hy[1:Ny-1]\
                +hx[i]*hy[0:Ny-2]+hx[i-1]\
                *hy[0:Ny-2])+1/3*rho_zx\
                *x_volatility*z_volatility\
                *x[i]*z[k]*(old_u[i+1,1:Ny-1,k+1]\
                -old_u[i+1,1:Ny-1,k-1]-old_u[i-1\
                ,1:Ny-1,k+1]+old_u[i-1,1:Ny-1,k-1])\
                /(hx[i-1]*hz[k]+hx[i]*hz[k]\
                +hx[i]*hz[k-1]+hx[i-1]*hz[k-1])\
                +1/3*rho_yz*y_volatility\
                *z_volatility*y[1:Ny-1]*z[k]\
                *(old_u[i,2:Ny,k+1]-old_u[i,2:Ny,k-1]\
                -old_u[i,0:Ny-2,k+1]+old_u[i,0:Ny-2\
                ,k-1])/(hy[0:Ny-2]*hz[k]+hy[1:Ny-1]\
                *hz[k]+hy[1:Ny-1]*hz[k-1]+hy[0:Ny-2]\
                *hz[k-1])+old_u[i,1:Ny-1,k]/dt
            u[i,1:Ny-1,k]=thomas(ay,dy,cy,fy)
u[1:Nx-1,1:Ny-1,Nz-1]=2*u[1:Nx-1,1:Ny-1,Nz-2]-u[1:Nx-1,1:Ny-1,Nz-3]
u[Nx-1,1:Ny-1,1:Nz]=2*u[Nx-2,1:Ny-1,1:Nz]-u[Nx-3,1:Ny-1,1:Nz]
u[1:Nx,Ny-1,1:Nz]=2*u[1:Nx,Ny-2,1:Nz]-u[1:Nx,Ny-3,1:Nz]
# z 축으로 풀기
for j in range(1,Ny-1):
    for i in range(1,Nx-1):
        fz[0:Nz-1]=1/3*rho_xy*x_volatility\
          *y_volatility*x[i]*y[j]\
          *(u[i+1,j+1,1:Nz-1]-u[i+1,j-1,1:Nz-1]\
          -u[i-1,j+1,1:Nz-1]+u[i-1,j-1,1:Nz-1])\
          /(hx[i-1]*hy[j]+hx[i]*hy[j]+hx[i]\
          *hy[j-1]+hx[i-1]*hy[j-1])+1/3*rho_zx\
          *x_volatility*z_volatility*x[i]\
          *z[1:Nz-1]*(u[i+1,j,2:Nz]-u[i+1,j,0:Nz-2]-u[i-1,j,2:Nz]\
          +u[i-1,j,0:Nz-2])/(hx[i-1]*hz[1:Nz-1]\
          +hx[i]*hz[1:Nz-1]+hx[i]*hz[0:Nz-2]\
          +hx[i-1]*hz[0:Nz-2])+1/3*rho_yz\
          *y_volatility*z_volatility*y[j]*z[1:Nz-1]*(u[i,j+1,2:Nz]\
          -u[i,j+1,0:Nz-2]-u[i,j-1,2:Nz]+u[i,j-1,0:Nz-2])/(hy[j-1]\
          *hz[1:Nz-1]+hy[j]*hz[1:Nz-1]\
```

```
                +hy[j]*hz[0:Nz-2]+hy[j-1]*hz[0:Nz-2])\
                +u[i,j,1:Nz-1]/dt
            old_u[i,j,1:Nz-1]=thomas(az,dz,cz,fz)

old_u[1:Nx-1,1:Ny-1,Nz-1]=2*old_u[1:Nx-1,1:Ny-1,Nz-2]-old_u[1:Nx-1,1:Ny-1
,Nz-3]

old_u[Nx-1,1:Ny-1,1:Nz]=2*old_u[Nx-2,1:Ny-1,1:Nz]-old_u[Nx-3,1:Ny-1,1:Nz]
    old_u[1:Nx,Ny-1,1:Nz]=2*old_u[1:Nx,Ny-2,1:Nz]-old_u[1:Nx,Ny-3,1:Nz]
    u=old_u

    # OSM과 토마스 알고리즘을 이용하여 ku값 계산
    # x 축으로 풀기
    for j in range(1,Ny-1):
        for k in range(1,Nz-1):
            fx[0:Nx-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[1:Nx-1]*y[j]\
              *(ku[2:Nx,j+1,k]-ku[2:Nx,j-1,k]\
              -ku[0:Nx-2,j+1,k]+ku[0:Nx-2,j-1,k])\
              /(hx[0:Nx-2]*hy[j]+hx[1:Nx-1]*hy[j]\
              +hx[1:Nx-1]*hy[j-1]+hx[0:Nx-2]*hy[j-1])\
              +1/3*rho_zx*x_volatility*z_volatility\
              *x[1:Nx-1]*z[k]*(ku[2:Nx,j,k+1]\
              -ku[2:Nx,j,k-1]-ku[0:Nx-2,j,k+1]\
              +ku[0:Nx-2,j,k-1])/(hx[0:Nx-2]*hz[k]\
              +hx[1:Nx-1]*hz[k]+hx[1:Nx-1]*hz[k-1]\
              +hx[0:Nx-2]*hz[k-1])+1/3*rho_yz\
              *y_volatility*z_volatility*y[j]\
              *z[k]*(ku[1:Nx-1,j+1,k+1]\
              -ku[1:Nx-1,j+1,k-1]-ku[1:Nx-1,j-1,k+1]\
              +ku[1:Nx-1,j-1,k-1])/(hy[j-1]*hz[k]\
              +hy[j]*hz[k]+hy[j]*hz[k-1]+hy[j-1]\
              *hz[k-1])+ku[1:Nx-1,j,k]/dt
            old_ku[1:Nx-1,j,k]=thomas(ax,dx,cx,fx)

old_ku[1:Nx-1,1:Ny-1,Nz-1]=2*old_ku[1:Nx-1,1:Ny-1,Nz-2]-old_ku[1:Nx-1,1:
Ny-1,Nz-3]

old_ku[Nx-1,1:Ny-1,1:Nz]=2*old_ku[Nx-2,1:Ny-1,1:Nz]-old_ku[Nx-3,1:Ny-1,1:
Nz]
```

```
old_ku[1:Nx,Ny-1,1:Nz]=2*old_ku[1:Nx,Ny-2,1:Nz]-old_ku[1:Nx,Ny-3,1:Nz]
    # y 축으로 풀기
    for k in range(1,Nz-1):
        for i in range(1,Nx-1):
            fy[0:Ny-1]=1/3*rho_xy*x_volatility\
             *y_volatility*x[i]*y[1:Ny-1]\
             *(old_ku[i+1,2:Ny,k]-old_ku[i+1,0:Ny-2,\
             k]-old_ku[i-1,2:Ny,k]+old_ku[i-1,0:Ny-2,k])/(hx[i-1]\
             *hy[1:Ny-1]+hx[i]*hy[1:Ny-1]+hx[i]\
             *hy[0:Ny-2]+hx[i-1]*hy[0:Ny-2])+1/3\
             *rho_zx*x_volatility*z_volatility\
             *x[i]*z[k]*(old_ku[i+1,1:Ny-1,k+1]\
             -old_ku[i+1,1:Ny-1,k-1]-old_ku[i-1,\
             1:Ny-1,k+1]+old_ku[i-1,1:Ny-1,k-1])/\
             (hx[i-1]*hz[k]+hx[i]*hz[k]+hx[i]*hz[k-1]\
             +hx[i-1]*hz[k-1])+1/3*rho_yz*\
             y_volatility*z_volatility*y[1:Ny-1]*z[k]\
             *(old_ku[i,2:Ny,k+1]-old_ku[i,2:Ny,k-1]\
             -old_ku[i,0:Ny-2,k+1]+old_ku[i,0:Ny-2\
             ,k-1])/(hy[0:Ny-2]*hz[k]+hy[1:Ny-1]\
             *hz[k]+hy[1:Ny-1]*hz[k-1]+hy[0:Ny-2]\
             *hz[k-1])+old_ku[i,1:Ny-1,k]/dt
            ku[i,1:Ny-1,k]=thomas(ay,dy,cy,fy)

ku[1:Nx-1,1:Ny-1,Nz-1]=2*ku[1:Nx-1,1:Ny-1,Nz-2]-ku[1:Nx-1,1:Ny-1,Nz-3]
    ku[Nx-1,1:Ny-1,1:Nz]=2*ku[Nx-2,1:Ny-1,1:Nz]-ku[Nx-3,1:Ny-1,1:Nz]
    ku[1:Nx,Ny-1,1:Nz]=2*ku[1:Nx,Ny-2,1:Nz]-ku[1:Nx,Ny-3,1:Nz]
    # z 축으로 풀기
    for j in range(1,Ny-1):
        for i in range(1,Nx-1):
            fz[0:Nz-1]=1/3*rho_xy*x_volatility\
              *y_volatility*x[i]*y[j]*(ku[i+1,j+1,1:Nz-1]\
              -ku[i+1,j-1,1:Nz-1]-ku[i-1,j+1,1:Nz-1]\
              +ku[i-1,j-1,1:Nz-1])/(hx[i-1]\
              *hy[j]+hx[i]*hy[j]+hx[i]*hy[j-1]\
              +hx[i-1]*hy[j-1])+1/3*rho_zx\
              *x_volatility*z_volatility*x[i]\
              *z[1:Nz-1]*(ku[i+1,j,2:Nz]\
              -ku[i+1,j,0:Nz-2]-ku[i-1,j,2:Nz]\
              +ku[i-1,j,0:Nz-2])/(hx[i-1]\
              *hz[1:Nz-1]+hx[i]*hz[1:Nz-1]\
```

```
                    +hx[i]*hz[0:Nz-2]+hx[i-1]\
                    *hz[0:Nz-2])+1/3*rho_yz\
                    *y_volatility*z_volatility*y[j]\
                    *z[1:Nz-1]*(ku[i,j+1,2:Nz]\
                    -ku[i,j+1,0:Nz-2]-ku[i,j-1,2:Nz]\
                    +ku[i,j-1,0:Nz-2])/(hy[j-1]\
                    *hz[1:Nz-1]+hy[j]*hz[1:Nz-1]\
                    +hy[j]*hz[0:Nz-2]+hy[j-1]\
                    *hz[0:Nz-2])+ku[i,j,1:Nz-1]/dt
                old_ku[i,j,1:Nz-1]=thomas(az,dz,cz,fz)

old_ku[1:Nx-1,1:Ny-1,Nz-1]=2*old_ku[1:Nx-1,1:Ny-1,Nz-2]-old_ku[1:Nx-1,1:Ny-1,Nz-3]

old_ku[Nx-1,1:Ny-1,1:Nz]=2*old_ku[Nx-2,1:Ny-1,1:Nz]-old_ku[Nx-3,1:Ny-1,1:Nz]

old_ku[1:Nx,Ny-1,1:Nz]=2*old_ku[1:Nx,Ny-2,1:Nz]-old_ku[1:Nx,Ny-3,1:Nz]
    ku=old_ku

ii=np.where(x==100)
jj=np.where(y==100)
kk=np.where(z==100)
print('Price=%f'%(u[ii,jj,kk]))
```

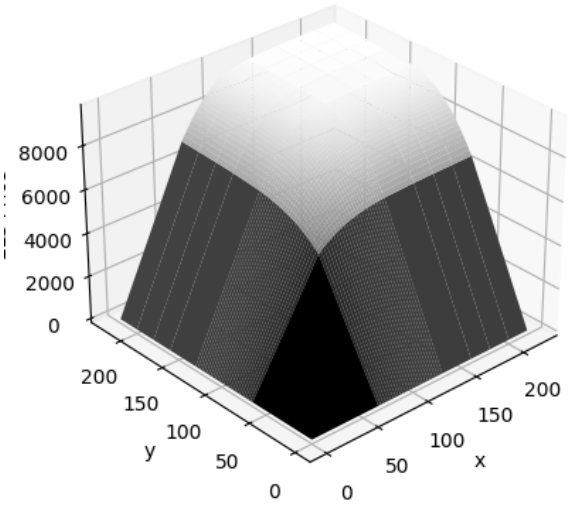
Price=8200.282098 (ELS 최종가격)

23

```
from matplotlib.figure import projections
X, Y = np.meshgrid(x,y)
kk=np.argwhere(z==100)

fig1=plt.figure()
ax=fig1.add_subplot(projection='3d')
ax.plot_surface(X,Y,u[:,:,int(kk)],cmap=plt.cm.gray)
ax.view_init(elev=30,azim=-132)
ax.set_xlabel('x', fontsize=10)
ax.set_ylabel('y', fontsize=10)
ax.zaxis.set_rotate_label(False)
ax.set_zlabel('ELS Price',rotation=90,fontsize=10)

fig2=plt.figure()
```

```
bx=fig2.add_subplot(projection='3d')
bx.plot_surface(X,Y,ku[:,:,int(kk)],cmap=plt.cm.gray)
bx.view_init(elev=30,azim=-132)
bx.set_xlabel('x', fontsize=10)
bx.set_ylabel('y', fontsize=10)
bx.zaxis.set_rotate_label(False)
bx.set_zlabel('ELS Price',rotation=90,fontsize=10)
plt.show()
```

<낙인 배리어 아래로 기초자산이 떨어지지 않았을 때의 ELS가격)



<낙인 배리어 아래로 기초자산이 떨어졌을 때의 ELS가격)