SFERS DATA Seminar with Python: 3

Yitae Kwon

SFERS of SNU

2024-1

Contents

Stochastic Simulation: Single Period

Monte Carlo Method

대수의 법칙에 의하여, n이 충분히 크다면,

$$\int_A g(x)f(x)dx = \mathbb{E}[g(X)] \approx \frac{1}{n}\sum_{i=1}^{n} g(X_i), \quad X_i \overset{\text{i.i.d.}}{\sim} F$$

입니다. 따라서 우리는 수리적인 계산을 통해 $g(X)$의 분포를 정확히 알지는 못하더라도, 그 기댓값은 근사해줄 수 있습니다.
이처럼 표본 추출을 통해 확률변수 X에 관련한 적분값들을 예측하는 방법을 **Monte Carlo method**라고 부릅니다.

한편 동일한 방식으로 신뢰구간 역시 구할 수 있습니다. $g(X)$의 표준편차 추정량은

$$\widehat{\mathrm{sd}}(g(X)) = \left(\frac{1}{n-1} \sum_{i=1}^{n} (g(X_i) - \mathbb{E}_n[g(X)])^2 \right)^{1/2}$$

으로 주어질 것이며, 중심극한정리를 이용한 근사적인 $100(1-\alpha)$퍼센트 신뢰구간은

$$\left(\mathbb{E}_n[g(X)] - z_{1-\alpha/2} \frac{\widehat{\mathrm{sd}}(g(X))}{\sqrt{n}}, \mathbb{E}_n[g(X)] + z_{1-\alpha/2} \frac{\widehat{\mathrm{sd}}(g(X))}{\sqrt{n}} \right)$$

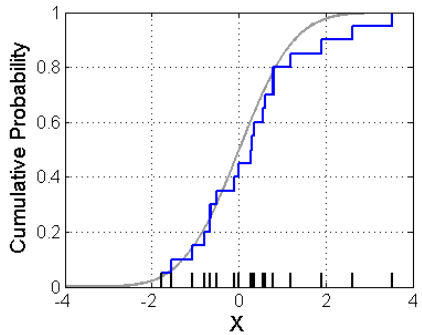
으로 얻을 수 있을 것입니다.

Monte Carlo Method

한편 분포 자체를 근사하려면 어떻게 해야 할까요? 일반적으로 우리는 해당 분포가 가지는 누적분포함수를 **경험적 누적분포함수**(empirical cumulative distribution function; eCDF)로써 얻습니다. 이는

$$\hat{F}(x) = \frac{1}{n} \sum_{i=1}^{n} I(X_i \leq x)$$

처럼 계산됩니다.



Monte Carlo Method

따라서 사실 우리가 $g(X)$의 평균과 분산을 구했던 것은, $F(x)$대신 $\hat{F}(x)$를 CDF로 하는 분포에서 그 평균과 분산을 구한 것이나 마찬가지입니다. 이를 이용하여 추후 세션에서 **Bootstrap**과 같은 다양한 비모수적 추정/검정 방법론들 역시 알아볼 예정입니다.

Note: eCDF가 CDF를 근사할 때, 그 error는 $n^{-1/2}$에 비례함이 알려져 있습니다. 따라서 Monte Carlo 방법의 오류 역시 대개 $n^{-1/2}$에 비례합니다. 다르게 말하면,

$$\mathbb{E}_n[g(X)] - \mathbb{E}[g(X)] = O(n^{-1/2})$$

Monte Carlo Method: Variance Reduction Methods

한편 $n^{-1/2}$는 그 수렴이 느린 편에 속합니다. 따라서 그 수렴을 빠르게 하기 위하여, 분산을 감소시킬 수 있습니다. 지난 시간에 간단히 언급하였듯, 추정량의 분산이 감소하는 것은 해당 추정량의 효율이 증가함을 의미합니다.

- Importance sampling: measure change를 통해 reduction
- Antithetic variable method: 각 X_i에 대하여 $Y_i = F(X_i)$로 정의하면 Y_i가 균등확률분포를 따르는 것을 이용하여 음의 상관관계를 가지고 있는 두 추정량의 합을 이용
- Control variable method: $g(X)$와 높은 상관관계를 갖는 적당한 control variable Y를 도입하여 최소 분산을 갖는 unbiased estimator 얻음

이외에도 여러 방법이 개발되어 있습니다. 하지만 Naïve MC가 아직도 사용되는 이유 중 하나는, 이들 방법에 비해 압도적으로 간단하기 때문입니다. 그냥 샘플링해서, 평균만 내면 되니까요.

Random Number Generation

결국 Monte Carlo simulation을 하려면 어떠한 분포를 따르는 확률변수를 랜덤하게(i.i.d.로) n개 만들 수 있어야 합니다. 그런데 과연 컴퓨터는 어떻게 이 랜덤한 확률변수들을 만들 수 있을까요? 정답은... **할 수 없다**입니다. 컴퓨터가 만드는 무작위의 수들은 정말 '무작위'라고 볼 수는 없습니다. 단지 무작위로 보일 뿐인 의사난수(pseudo-random number)들일 뿐입니다. 따라서 우리가 **시드(seed)**를 동일하게 하면 동일한 표본이 나올 수 있게 되는 것입니다.

Note: 파이썬의 `numpy` 모듈에서는 O'Neill의 **permuted congruential generator**(PCG)를 pseudo-random number generator로 사용합니다. 물론 이외에도 다양한 generator가 있으며, 이들을 옵션으로 사용할 수 있습니다.
Link: `https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf`

Random Number Generation

2014년 PCG가 사용되기 전에는 **linear congruential generator**(LCG)를 많이 사용했습니다. PCG 대신 비교적 간단한 LCG에 대하여 소개해 보도록 하겠습니다. congruence(합동)을 이용한다는 점에서 둘은 유사하며, LCG를 기반으로 PCG가 개발되었다고 보시면 되겠습니다.

- 미리 정해진 자연수
  - a: 승수(multiplier)
  - c: 시프트(shift)
  - M: 모듈로(modulo)
- 시드 U_0

Random Number Generation

1. 먼저, 아래 점화식과 시드 U_0에 의하여 생성되는 수열 $\{U_n\}_{n<M}$을 생성합니다. (**Note**: 이 수열은 $M-1$의 약수를 주기로 함)

$$U_{n+1} = aU_n + c \pmod{M}$$

2. 만약 M이 소수라면, 적절한 승수 a를 원시근이 되도록 결정할 경우 $U_1, \cdots, U_{M-1}$의 $M-1$개가 모두 다른 값이며 $\{1, 2, \cdots, M-1\}$의 뒤섞음(permutation)이고, 그 주기가 $M-1$입니다. 컴퓨터에서 사용하기 적절한 M은 2^n 형태입니다. $M-1 = 2^n - 1$ 형태 중 소수가 되는 수들을 메르센 소수라고 합니다. 대표적으로, $2^{31}-1$은 소수이며, 원시근 $a = 7^5$와 쌍을 이루어 $2^{31}-1$을 주기로 가지는 $1 \sim 2^{31}-1$ 사이의 수를 거의 무작위로 생성할 수 있습니다.
3. 이제 $U_1, \cdots, U_{M-1}$을 $M-1$으로 나누면, $(0, 1]$ 사이에 있는 난수를 무작위로 생성하는 난수생성기가 만들어집니다.

Random Number Generation

이제 균등분포를 따르는 난수를 생성하는 방법을 압니다. 어떠한 확률변수의 CDF가 F라고 한다면, $Y_i = F(X_i)$를 정의해 보겠습니다. 그렇다면, Y_i는 $[0,1]$의 확률변수고 I.I.D로 분포합니다. 또한 $x \in [0,1]$에 대해

$$P(Y_i \leq x) = P(F(X_i) \leq x) = P(X_i \leq F^{-1}(x)) = F(F^{-1}(x)) = x$$

이므로, Y_i는 $U(0,1)$에서 랜덤하게 표집된 확률변수열과 같습니다.

반대로 말하면, 우리가 균등분포를 따르는 의사난수 $Y_1, \cdots, Y_n$을 만들 경우

$$X_i = F^{-1}(Y_i)$$

도록 X_i를 다시 생성하면 이는 분포 F를 따르는 의사난수가 됩니다.

Random Number Generation

다만 F^{-1}를 구하기 어려운 경우, 이는 좋은 방법이 아닙니다. 우리는 정규분포를 많이 만들게 될텐데, 정규분포 CDF의 역함수는 구하기 어렵습니다. 이를 위해 대용적으로 많이 사용되는 방법 중 하나가 **Box-Müller method** 입니다. 그 원리는 생략하고 방법만 설명하면, 아래와 같습니다.

1. 서로 독립이고 $U(0,1)$을 따르는 의사난수 u_1, u_2를 생성한다.
2. 아래처럼 서로 독립인 표준정규분포를 따르는 z_1, z_2를 생성한다.

$$z_1 := \sqrt{-2 \ln u_2} \cos(2\pi u_1), \quad z_2 := \sqrt{-2 \ln u_2} \sin(2\pi u_1)$$

3. 원하는 기댓값 μ와 공분산 행렬 Σ에 대하여,

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \mu + \Sigma^{1/2} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

는 $\mathcal{N}(\mu, \Sigma)$를 따르는 정규난수이다.

Quasi-Monte Carlo Method

Monte Carlo method는 pseudo-random number를 이용하여 적분의 값을 예측하는 방법이었습니다. 그러나 그 오차가 $O(n^{-1/2})$ 수준이고, 여러 variance reduction method도 점근적으로 그 효율을 크게 개선시키지는 못합니다. 이 때문에 많은 계산이 필요한 금융 분야 시뮬레이션에서는 더 빠른 속도와 높은 효율로 수렴시키기 위한 **quasi-Monte Carlo Method** 등이 개발되었습니다.

quasi-random number

준난수(quasi-random number), 혹은 **저불일치점열**(low-discrepancy sequence)는 집합에 있는 기존 숫자로부터 최대한 멀리 떨어져 있는 연속된 숫자가 생성되도록 만들어진 난수 혹은 점열을 의미합니다.

Low Discrepancy Sequence

$[0,1]$의 범위를 가지는 길이 n인 점열 P의 **스타 불일치도**(star discrepancy)는 아래와 같이 정의됩니다.

$$D_n^*(P) := \sup_{y \in [0,1]} \left| \hat{F}(y) - y \right|$$

즉 이는 점열에서 얻은 eCDF $\hat{F}$와 균등분포의 CDF 사이 차이를 의미합니다. 만약 어떤 점열 P에 대해 앞의 n개를 P_n이라 쓸 경우,

$$D_n^*(P_n) \leq c \frac{\log n}{n}$$

이 성립한다면 이를 **저불일치점열**(low discrepancy sequence)라고 부릅니다. 더불어, 이 P_n은 n개의 준난수로 취급할 수 있습니다.

Quasi-Monte Carlo Method

준몬테카를로법은 몬테카를로법과 동일하나, 의사난수를 준난수로만 대체합니다. 만약 Y_i가 균등확률변수를 따르는 준난수라면, 아래의 정리가 만족함이 알려져 있습니다.

Koksma-Hlawka Inequality

$Y_1, \cdots, Y_n$이 균등확률변수를 따르는 준난수이고 g가 C^1 함수라면,

$$\left| \frac{1}{n} \sum_{i=1}^{n} g(Y_i) - \mathbb{E}[g(Y_i)] \right| \leq D_n^*(Y_1, \cdots, Y_n) \times \int_0^1 |g'(u)| dy$$

가 성립한다. D_n^*가 $O(\log n/n)$을 따르므로, 준몬테카를로법의 오차는 $O(n^{-1} \log n)$ 수준으로 $O(n^{-1/2})$인 몬테카를로법보다 우월하다. 더불어 몬테카를로법과 달리 오차의 결정된 상한을 제공한다는 점에서 의미가 있다.

Quasi-Monte Carlo Method

대표적인 저불일치점열의 생성 과정은 아래와 같습니다.

- Van der Corput sequence
- Halton sequence
- Faure sequence
- Sobol sequence
- Niederreiter Sequence

한편 준난수열은 완벽히 난수의 성질을 띠지는 않지만, 오히려 이러한 점 때문에 난수들을 균등하게 분포시킬 수 있어 몬테카를로법에서 특정 부분에 수열이 많이 쏠리면서 나타나는 효율의 저하를 방지할 수 있는 것으로 알려져 있습니다. 다만 준몬테카를로 방법은 예측력을 높인 대신 통계적인 성질을 일부 잃었기에, 그 결과를 통계적으로 해석하기에는 어렵습니다.

Quasi-Monte Carlo Method in Python

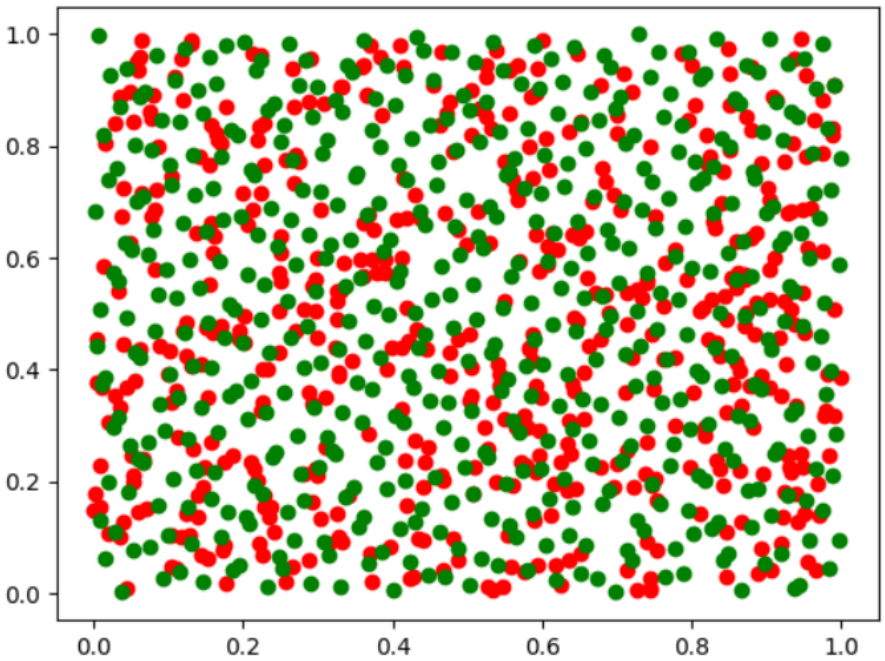
파이썬에서 난수를 생성하기 위해 자주 사용되는 라이브러리들은 아래와 같습니다.

- `numpy.random`: PCG에 기반하여 pseudo-random number를 만듭니다. 대부분의 분포 형태에 대하여 이미 함수가 구현되어 있습니다. 예를 들어, `.rand()`를 사용하면 균등분포를, `.multivariate_normal()`을 사용하면 다변량 정규분포를 만들어 줍니다.
- `scipy.stats.qmc`: 다양한 quasi-random number generater를 제공합니다. 디폴트로는 `.Sobol()` 을 이용하며, 오직 균등분포와 다변량 정규분포만 함수로 제공하고 이외 분포가 필요할 경우 직접 구현하셔야 합니다.

Note: 시뮬레이션의 경우 재현성을 위해 시드를 설정해 주세요.

Quasi-Monte Carlo Method in Python

```
# difference of two RNGs
sample_pseudo_1 = np.random.rand(512, 2)
sample_pseudo_2 = qmc.Sobol(d = 2, seed = 1234).random_base2(m = 9)
plt.scatter(sample_pseudo_1[:,0], sample_pseudo_1[:,1], color = 'red')
plt.scatter(sample_pseudo_2[:,0], sample_pseudo_2[:,1], color = 'green')
plt.show()
```



Quasi-Monte Carlo Method in Python

Aim: 몬테카를로법과 준몬테카를로법을 통하여 유럽형콜옵션의 가격을 추정하여 보고, 블랙-숄즈 공식과 비교해 보자.

Data:

- $T = 1$: 만기
- $r = 0.2$: 무위험자산인 예금의 이자율
- $S_0 = 1$: 주식의 초기 가격
- $\sigma = 0.5$: 주가의 변동성
- $K = 1.2$: 행사가격

BS Model:

$$S_T = S_0 \exp\left((r - \frac{1}{2}\sigma^2)T + \sigma Z_T\right), \quad \text{payoff} : (S_T - K)_+$$

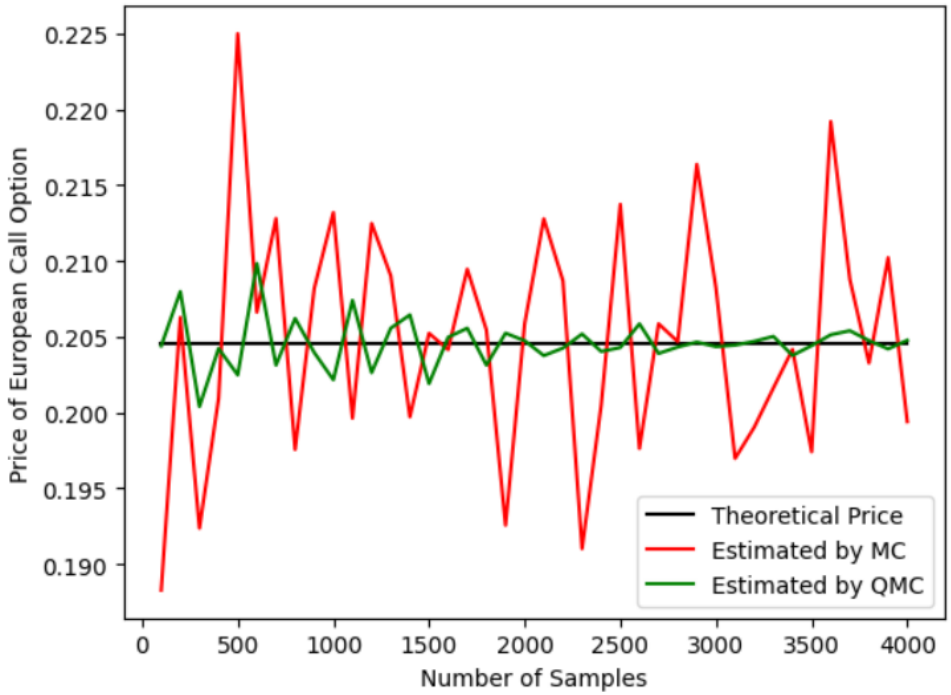
$$V_0 = S_0 N(d_1) - K e^{-rT} N(d_2)$$

Quasi-Monte Carlo Method in Python

무차익거래 가설에 의하여, 위험중립측도 하에서

$$V_0 = \mathbb{E}[e^{-rT}(S_T - K)_+]$$

이므로, S_T를 (준)몬테카를로로 날린 다음 저 함수의 표본평균을 취하여 V_0을 추정할 수 있습니다. n이 증가할수록 QMC의 효율이 압도적인 걸(수렴을 빨리하는 걸) 확인해줄 수 있습니다.



Stochastic Simulation: Multi-Period

Stochastic Processes

확률과정(stochastic process)는 각 시간 t에 대하여 확률변수가 존재하는 것으로 볼 수 있습니다. 한편 확률변수는 어떠한 사건 ω에 상응하는 값을 내놓는 함수로 볼 수 있으므로, 확률과정은

$$\{X_t\},\ \{X_t(\omega)\},\ \{X(\omega, t)\}$$

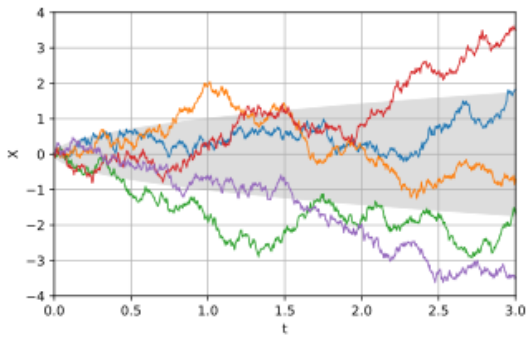
와 같은 다양한 형태로 쓰며, 이들은 모두 사건 ω가 발생하는 경우, 시간 t에서 관찰한 반응변수 X의 값을 의미하게 됩니다. 우리는 어떠한 한 사건 $\bar{\omega} \in \Omega$에서의 X_t를 관찰할 수밖에 없는데, 이때의 $X_t(\bar{\omega})$를 **표본경로**(sample path)라고 부릅니다.

한편 통계학과 경제학에서는 $t \in \{0, 1, \cdots, T\}$처럼 이산 시간의 값들만 관찰할 수 있으므로, 이들을 **시계열**(time series)로써 분석합니다. 반면 이론적 상황, 혹은 시간 간격이 매우 좁은 경우 $t \in [0, T]$처럼 연속된 시간 스케일에서 분석합니다.

Brownian Motion

대표적인 확률과정 중 하나는 **브라운 운동**(Brownian motion)입니다. 브라운 온동은 물 위에 띄운 꽃가루처럼 무작위적인 운동을 설명하는 데 적합한 모형입니다. 한편 경제학에서 자주 접하셨을 **random walk**는 브라운 운동을 이산시간에서 근사하는 방법 중 하나입니다. 브라운 운동 B_t는 아래 조건을 만족하는 확률과정을 말합니다.

1. $B_0 = 0$
2. For $s > t$ and $h > 0$, $B_{s+h} - B_s \perp\!\!\!\perp B_t$
3. $B_{s+h} - B_s \sim \mathcal{N}(0, h)$
4. 모든 sample path $B_t(\omega)$는 연속이다.



Brownian Motion

브라운 운동은 확률과정들이 가지는 좋은 성질들을 가지고 있습니다.

- **Lévy Process**: 브라운 운동은 Lévy 확률과정입니다. 이는 앞선 브라운 운동의 정의에서 3번 가정을 $B_{s+h} - B_s \overset{d}{=} B_h$로 바꾼 것입니다. 즉 특정 시점의 브라운 운동은 이전 시점의 브라운 운동과 닮아 있습니다.
- **Martingale**: 어떠한 확률과정이 마팅게일이라는 것은 특정 시점 이후의 기댓값이 직전 시점의 값으로써 주어짐을 의미합니다. 다르게 말하면, 특정 시점 이후의 값을 가장 잘 예측하는 값이 바로 직전 시점의 값입니다. 즉 $h > 0$에 대해

$$\mathbb{E}[X_{t+h}|\mathcal{F}_t] = X_t$$

입니다. 이때 $\mathcal{F}_t = \mathcal{I}_t$는 t 시점 이전까지의 X_t에 대한 정보를 의미합니다.

Brownian Motion

- **Markov Property**: 어떠한 확률과정이 마코프 성질을 가진다는 것은

$$P(X_{t+h} \in A|\mathcal{F}_t) = P(X_{t+h} \in A|X_t)$$

  임을 의미합니다. 즉 t시점 이전의 정보를 모두 요구하지 않아도, 직전 시점의 정보만 알면 이후의 값이 어떻게 움직일지에 대한 확률을 알 수 있습니다.
- **quadratic variation**: t가 $t \to t + dt$로 변할 때, $B_t \to B_{t+dt}$로 변할 것입니다. 이때 $dB_t := B_{t+dt} - B_t$를 정의하여 봅시다. 그렇다면,

$$(dB_t)^2 \to dt$$

  임이 알려져 있습니다. 즉 $dt \to 0$이면, $(dB_t)^2 = dt$입니다.

만약 어떠한 확률과정 X_t가 시간 t와 해당 사건 ω가 발생했을 때의 브라운 운동 B_t의 합으로 결정된다고 합시다. 즉

$$X_t = t + B_t$$

입니다. 양변을 미분형식(differential form)으로 쓰게 되면,

$$dX_t = dt + dB_t$$

처럼 쓸 수 있습니다. 즉 X_t의 변동은 t에 의한 변동과 B_t에 의한 변동으로 분해하여 쓸 수 있습니다.

Itô Process

만약 확률과정의 동태가 위에서처럼 시간 t와 B_t에 의해 결정된다고 할 때, 즉,

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dB_t$$

처럼 쓸 수 있다면, X_t를 **Itô process**라고 부릅니다. 이때 μ를 **drift term**, σ를 **diffusion term**이라고 부릅니다.

만약 동태가 상수 μ, σ에 대하여

$$dX_t = \mu X_t dt + \sigma X_t dB_t$$

로 주어진다면(혹은, $X_t = X_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma B_t}$이라면), 이 X_t를 **geometric Brownian motion starting at** X_0(기하브라운운동; GBM)이라 부릅니다.

Black-Scholes Model

블랙-숄즈 모형에서는 두 개의 asset이 시장에 존재한다고 가정합니다.

1. riskless asset(bank account) G_t

$$G_t = e^{rt}, \quad dG_t = rG_t dt$$

2. risky asset(stock) S_t

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma B_t}, \quad dS_t = \mu S_t dt + \sigma S_t dB_t$$

이로부터 다양한 방식을 통해 효율적 시장 하에서 다양한 에셋들의 가격들을 구해줄 수 있습니다. 다만 여기에서는 모형이 이렇다는 것만 소개하고 넘어갑니다.

Generation of Stochastic Processes

연속시간을 가정했던 앞선 모형과는 다르게, 우리의 실제 세계와 컴퓨터 상에서는 discrete time을 가정해야 합니다. 여기에서는 적절한 시간 단위에서 이산 시간으로 기하브라운운동을 근사하는 방법에 대해 알아 보겠습니다.

1. 표준정규분포를 따르는 난수열 $z_1, z_2, \cdots, z_n$을 발생시킨 뒤, $0 = t_0 < t_1 < \cdots < t_n$에 대하여 아래를 계산합니다.

$$W_{t_j} = \sum_{k=1}^{j} \sqrt{t_k - t_{k-1}} z_k$$

   이 W_{t_j}가 브라운운동의 한 표본경로를 이산화한 것이 됩니다.

2. 생성한 W_{t_j}에 대하여,

$$X_{t_j} = \exp\left(\left(\mu - \frac{1}{2}\sigma^2\right) t_j + \sigma W_{t_j}\right)$$

   를 계산합니다. 이 X_{t_j}는 drift가 μ고 volatility가 σ인 기하브라운운동의 표본경로를 이산화합니다.

Generation of Stochastic Processes

Aim: 대표적인 이색옵션(exotic option)인 Down-and-Out 배리어 옵션(barrier option)의 가격결정을 해보자.

Data:

- 상수들은 앞과 동일
- knock-out barrier $L = 0.7 < K = 1.2$
- 지급구조: $I(m_T^S > L)(S_T - K)_+$ where $m_T^S = \min\{S_t : 0 \le t \le T\}$

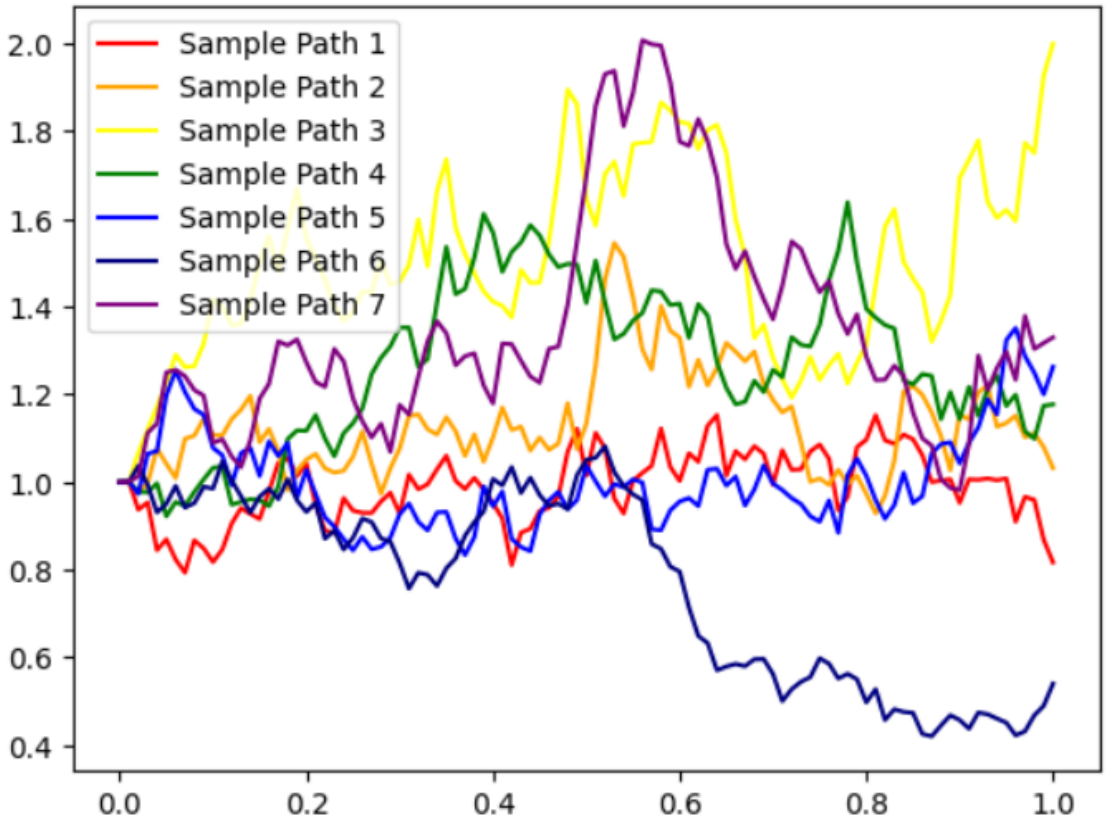
Theory: 이론적인 논의를 통해

$$V_0 = sN(d_1) - Ke^{-rT}N(d_2) - L\left(\frac{L}{s}\right)^{\frac{2r}{\sigma^2}} N(d_3) + Ke^{-rT}\left(\frac{L}{s}\right)^{\frac{2}{\sigma^2}(r-\frac{1}{2}\sigma^2)} N(d_4)$$

라는 공식이 밝혀져 있습니다.

Generation of Stochastic Processes

시간 간격을 충분히 줄인 다음 이를 수행해 보겠습니다. 저는 만기를 1, 시간 간격을 1/100으로 하여 먼저 기하브라운운동의 여러 표본경로를 만들어 제가 잘 짰는지 확인해 보았습니다.



Generation of Stochastic Processes

```
# numerical soln
iteration = 10000
sum = 0
for i in range(iteration):
    # i: iteration number
    x = GBM(dt, T, r, sigma)
    m_T = min(x)
    S_T = x[int(T/dt)]
    payoff = barrierpayoff(S_T, m_T, L, K)
    sum = sum + payoff
print("Estimated Price of Barrier Option:", sum/iteration)
```

✓ 53.7s

```
Estimated Price of Barrier Option: 0.23576890788980867
```

```
# theoretical soln
d1 = (np.log(S_0/K) + (r + np.power(sigma, 2)/2)*T)/(sigma * np.sqrt(T))
d2 = d1 - sigma * np.sqrt(T)
d3 = (np.log(np.power(L, 2)/(S_0 * K)) + (r + np.power(sigma, 2)/2)*T)/(sigma * np.sqrt(T))
d4 = d3 - sigma * np.sqrt(T)

V_0_barrier = S_0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)\
    - L * np.power(L/S_0, 2*r/np.power(sigma, 2)) * norm.cdf(d3)\
        + K * np.exp(-r * T) * np.power(L/S_0, 2*r/np.power(sigma, 2) - 1) * norm.cdf(d4)
V_0_barrier
```

✓ 0.0s

```
0.1943251283791747
```

도움되는 링크들

- 최병선 교수님 금융공학 노트들: https://sites.google.com/view/cbsdataly/research
- 원중호 교수님 통계계산 노트: https://won-j.github.io/M1399_000100-2022spring/
- 박형빈 교수님 금융수학2 노트: https://m.blog.naver.com/dhkdwnddml/222654118609
- scipy.stats.amc 모듈 소개: https://docs.scipy.org/doc/scipy/reference/stats.qmc.html
- 조금 더 고급의 모형이 궁금하시면: https://www.quantstart.com/articles/geometric-brownian-motion-simulation-with-python/

Optimization

Mathematical Programming

수학적 **최적화**(optimization), 혹은 **수학적 계획법**(mathematical programming)은 아래의 형태를 가지고 있습니다.

$$\begin{cases} \text{minimize} & f(x) \\ \text{subject to} & x \in C \end{cases}$$

이때 $x \in \mathbb{R}^d$를 **optimization variable**, f를 **목적함수**(objective function), C를 **제약집합**(constraint set)이라고 부릅니다. 그 해 $x^* \in C$는

$$f(x^*) \leq f(x),\ \forall x \in C$$

을 만족할 것입니다.

Convexity

어떠한 함수 f가 convex라는 것은, $x, y \in \mathbb{R}^d$와 $\forall \alpha \in [0,1]$에 대하여

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(x)$$

임과 동치입니다.

한편 어떠한 집합 C가 convex라는 것은, $x, y \in C$와 $\forall \alpha \in [0,1]$에 대하여

$$x, y \in C \;\Rightarrow\; \alpha x + (1-\alpha)y \in C$$

임을 의미합니다.

만약 목적함수 f와 제약집합 C가 convex라면, 여러 최적화 문제의 해를 구하기 간편함이 알려져 있습니다.

Convexity

우리가 일반적으로 경제학에서 접하는 최적화 문제는

$$\begin{cases} \text{maximize} & u(x) \\ \text{subject to} & p_1 x_1 + \cdots + p_d x_d = W \end{cases}$$

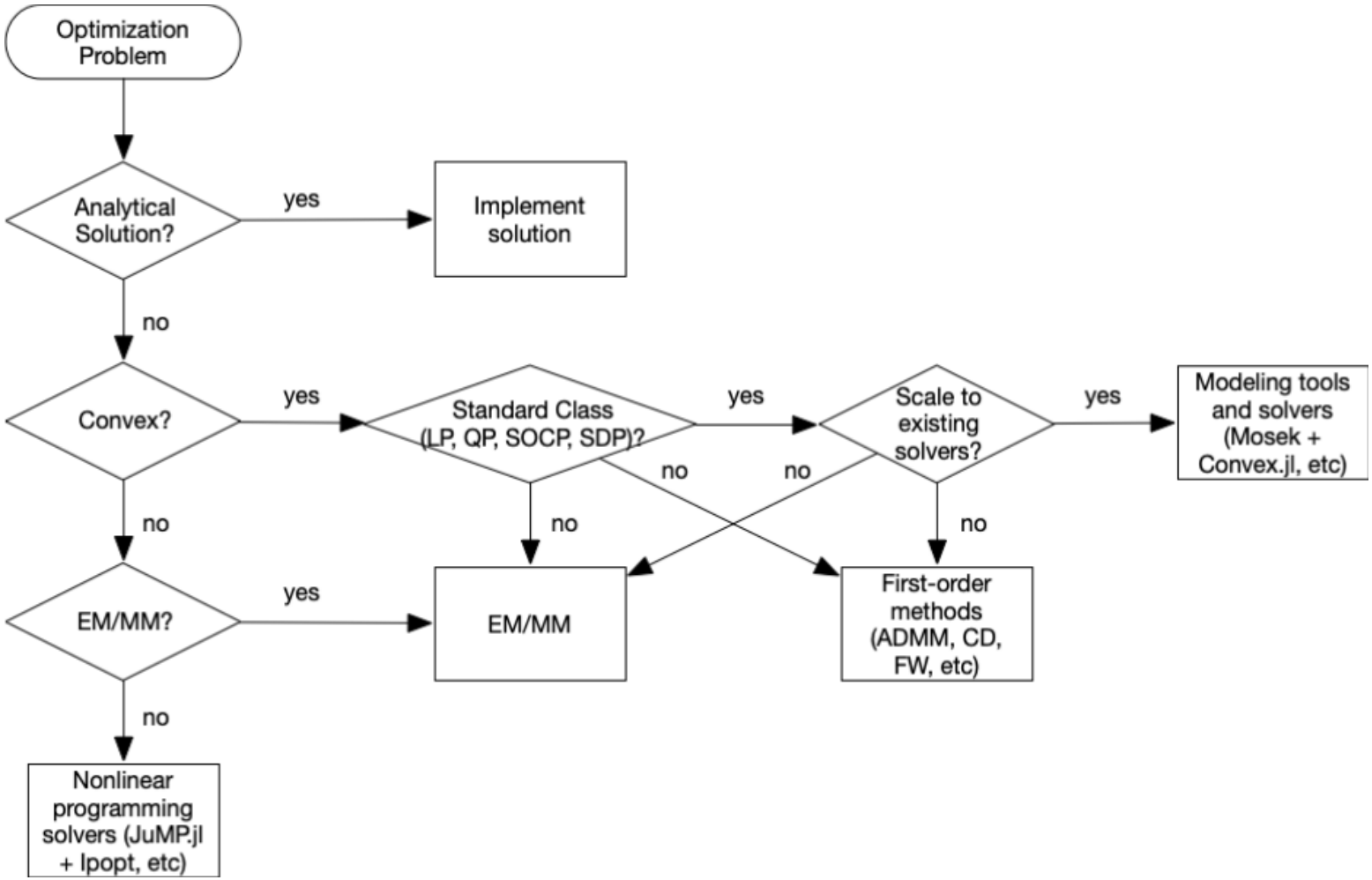
처럼 어떠한 예산제약 하에서 증가하는 concave 효용함수 u를 최대화시키는 형태입니다. 그리고 이 때의 $x_1, \cdots, x_d$은 i기의 소비가 될 수도 있고, 투자가 될 수도 있으며, 혹은 i번째 자산의 양이 될 수도 있습니다. 이를 수학적 최적화의 형태로 쓰면,

$$\begin{cases} \text{minimize} & -u(x) \\ \text{subject to} & p \cdot x = W \end{cases}$$

입니다. $-u$는 convex 함수이며, $p \cdot x = W$ 형태의 초평면(hyperplane) 역시 convex 집합이므로 이러한 문제는 **컨벡스 최적화**(convex optimization)에서 사용하는 테크닉들로 풀 수 있습니다.

Numerical Solver

다만 컨벡스 최적화라도 이를 손으로 풀기에는 그 닫힌 형태가 없을 수도 있습니다. 이에 따라서 아래 차트를 따라 컴퓨터로 최적화를 수행해야 합니다.



Numerical Solver

우리는 미시에서 많이 등장하는 형태의 최적화에만 관심이 있기 때문에,

- analytical solution이 없고
- convex 문제이며
- standard calss에 속하는

형태의 문제들을 풀어보려 합니다. 특히 우리는 다양한 문제를 풀 수 있다고 알려진 학술용 솔버인 `Mosek`을 이용하고자 합니다.

Note: cmd에서 `pip install Mosek -user`

Note: `https://docs.mosek.com/latest/pythonapi/license-agreement.html`에서 라이센스 관련을 꼭 읽고 사용하시길 추천드립니다.

Numerical Solver

$$\begin{cases} \text{maximize} & x \\ \text{subject to} & 2.0 \leq x \leq 3.0 \end{cases}$$

```
with Env() as env:                                  # Create Environment
  with env.Task(0, 1) as task:                      # Create Task
    task.appendvars(1)                                # 1 variable x
    task.putcj(0, 1.0)                                # c_0 = 1.0
    task.putvarbound(0, boundkey.ra, 2.0, 3.0)        # 2.0 <= x <= 3.0
    task.putobjsense(objsense.minimize)               # minimize

    task.optimize()                                 # Optimize

    x = task.getxx(soltype.itr)                     # Get solution
    print("Solution x = {}".format(x[0]))           # Print solution
```

✓ 0.1s

```
Solution x = 2.0
```

Linear Programming

선형계획법(linear programming; LP)은 아래 형태를 가지는 최적화 문제를 말합니다. 추후 알아볼 분위수 회귀, SVM 등이 이 문제로 풀립니다.

$$\begin{cases} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & Gx \leq h \end{cases}$$

한편 굳이 경제학적 상황에 맞춰 고려해보자면, 최소 생산량이 있는 경우의 비용최적화 문제가 대표적입니다.

- x: 상품들의 생산량
- c: 각 상품의 생산가격
- $A, b = 0, 0$
- $G = -1^T$, $h \in \mathbb{R}$으로 설정하면, $x_1 + \cdots + x_n \geq -h$로 최소한의 생산량 제한이 걸린 경우로 고려할 수 있음

Linear Programming

$$\begin{cases} \text{minimize} & 5x_1 + 6x_2 + 11x_3 \\ \text{subject to} & x_1 + x_2 + 2x_3 \geq 15 \\ & 10 \geq x_1 \geq 0 \\ & 7 \geq x_2 \geq 0 \\ & 2 \geq x_3 \geq 0 \end{cases}$$

를 한 번 풀어 보겠습니다. 이는 대표적인 LP 문제입니다. 그 해는

```
...
x[0]=10.0
x[1]=1.0
x[2]=2.0
Minimal cost: 78.0
```

으로 계산됩니다. 자세한 방법은 여기서 소개하기에는 너무 길어서, 아래 링크를 참조하면서 직접 구현해 보세요.

Link: `https://docs.mosek.com/latest/pythonapi/index.html`

Quadratic Programming

(선형 제약을 가진) **이차계획법**(quadratic programming; QP)는 아래의 형태를 가집니다. 최소제곱법, LASSO 등이 이를 통해 풀립니다. 경제학적 상황으로는 mean-variance analysis, 선형수요함수가 주어질 때의 monopolist's profit maximization 등에서 사용 가능합니다.

$$\begin{cases} \text{minimize} & \frac{1}{2}x^T Px + q^T x + r \\ \text{subject to} & Ax = b \\ & Gx \leq h \end{cases}$$

단, 이때 P는 positive semidefinite가 보장되어야 합니다.

Quadratic Programming

x를 포트폴리오의 각 asset에 대한 투자 비율, r을 각 에셋의 기대수익률, Σ를 에셋들 사이의 공분산 행렬이라고 두겠습니다. 그러면 최적 포트폴리오는

$$\begin{cases} \text{maximize} & r^T x - \lambda x^T \Sigma x \\ \text{subject to} & 1^T x = 1 \\ & x \geq 0 \end{cases}$$

으로 쓸 수 있고, 적절히 변형하면 이는 QP formulation을 가집니다.
Note: $r^x = \mathbb{E}[X]$, $x^T \Sigma x = \text{Var}(X)$에 상응합니다.

```
...
  Solution status : OPTIMAL
  Primal.  obj: 1.0945572705e-01    nrm: 1e+00    Viol.  con: 2e-12    var: 0e+00
  Dual.    obj: 1.0945572706e-01    nrm: 4e-02    Viol.  con: 0e+00    var: 4e-12
Optimal solution: array('d', [0.26125101548267643, 0.35385865147176454, 0.38489033304707126])
```

다른 형태의 formulation 역시 풀 수 있습니다.

Example 1. SOCP(Second-Order Cone Programming)

$$\begin{cases} \text{minimize} & f^T x \\ \text{subject to} & \|A_i x + b_i\|_2 \leq c_i^T x + d_i, \quad i = 1, 2, \cdots, m \\ & Fx = g \end{cases}$$

Example 2. QCQP(Quadratically Constrained Quadratic Programming)

$$\begin{cases} \text{minimize} & \frac{1}{2} x^T P_0 x + q_0^T x \\ \text{subject to} & \frac{1}{2} x^T P_i x + q_i^T x + r_i \leq 0, \quad i = 1, 2, \cdots, m \\ & Fx = g \end{cases}$$

where P_i is positive semidefinite.

다른 형태의 formulation 역시 풀 수 있습니다.

Example 3. SDP(SemiDefinite Programming)

$$\begin{cases} \text{minimize} & f^T x \\ \text{subject to} & x_1 F_1 + \cdots + x_m F_m + G \preceq 0 \\ & Ax = b \end{cases}$$

Example 4. GP(Geometric Programming)

$$\begin{cases} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 1, \quad i = 1, 2, \cdots, m \\ & h_i(x) = 1, \quad i = 1, 2, \cdots, p \end{cases}$$

where f_i is **posynomial** and h_i is **monomial**.

GP

GP에 대해 한 번 자세하게 알아보겠습니다.

monomial

$$f(x) = cx_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$$

where $c > 0$ and $a_i \in \mathbb{R}$

posynomial

$$f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}}$$

where $c_k > 0$ and $a_{ik} \in \mathbb{R}$

Quiz: $n = 1$이라면, 어디서 많이 보셨나요? (Hint: 정보경제학)
Quiz: $n = 2$이라면, 어디서 많이 보셨나요? (Hint: 모든 전필에 등장)

도움이 되는 링크들

- Mosek Tutorials: `https://github.com/MOSEK/Tutorials`
- Mosek 예시들: `https://docs.mosek.com/latest/pythonapi/examples-list.html`
- Portfolio Optimization with Mosek: `https://github.com/MOSEK/PortfolioOptimization/tree/main`
- Mosek Fusion API를 이용한 최적화: `https://docs.mosek.com/latest/pythonfusion/index.html`
- 금융 분야의 다양한 문제들, 특히 유한한 개수의 자산이 있을 때 특정 효용함수 하에서 포트폴리오 선택 문제를 풀 때 이들 테크닉이 많이 사용됩니다. 대표적인 이유 중 하나는 transaction cost의 존재, 복잡한 utility function과 같은 상황에서 최적화는 손으로 하기 어렵기 때문입니다. 앞의 링크들 중 첫번째, 세번째 링크에 많은 다양한 예시가 존재하니 살펴보시길 바랍니다.

KKT Condition

이제 조금 익숙하실 이야기로 돌아와 보겠습니다. 여기에서는 convexity를 굳이 가정하지는 않습니다. 먼저 아래의 비제약 문제

$$\min_{x \in \mathbb{R}^d} f(x)$$

를 고려해 보겠습니다. 그렇다면 그 근 x^*는 아래를 만족합니다. 다만, 이를 직접 확인하기는 어렵습니다.

Zeroth-order optimality condition

$$f(x^*) \leq f(x), \quad \forall x \in \mathbb{R}^d$$

KKT Condition

그 다음으로 가장 많이 접하실 조건은 f가 C^1함수일 때
First-order optimality condition

$$\nabla f(x^*) = 0$$

입니다. x^*가 f를 최소화하는 점이라면 이는 항상 만족해야 하는 반면, 이것이 만족한다고 해서 최소화시키는 점은 아닙니다. 다만 이들 점을 후보로 하여 계산함으로써 x^*을 찾을 수 있습니다. 만약 f가 convex라면, 이는 필요충분조건이 됩니다.

KKT Condition

만약 f가 C^2 함수라면,
Second-order optimality condition

$$\nabla^2 f(x^*) \succeq 0$$

입니다. 이는 해당 점에서 local minimum이 형성되어야 함을 의미합니다. 따라서 우리는 보통

1. FOC, SOC를 만족하는 점을 모두 찾고
2. 그 점들에 대해 $f(x)$를 계산하여 최소로 만드는 점을 x^*로 결정

하는 방식으로써 최솟값을 찾습니다.

KKT Condition

이제 다시 제약이 있는 문제

$$\begin{cases} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, 2, \cdots, m \\ & h_i(x) = 0, \quad i = 1, 2, \cdots, p \end{cases}$$

를 고려해 봅시다. 이때의 FOC는 어떻게 변해야 할까요? 그 답이 바로 **Karush-Kuhn-Tucker(KKT) condition**입니다.

KKT Condition

이 문제의 **Lagrangian**은 아래와 같이 정의됩니다.

$$\mathcal{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$$

이때 λ와 ν를 **Lagrange multiplier vector** 혹은 **dual variables**라고 부릅니다. 그렇다면, local minimum x^*에서

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = \nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{p} \nu_i \nabla h_i(x^*) = 0$$

이어야만 합니다.

KKT Condition

이를 정리하면, $i = 1, 2, \cdots, m$ 혹은 $i = 1, 2, \cdots, p$에 대하여,

$$
\begin{aligned}
f_i(x^*) &\leq 0 \\
h_i(x^*) &= 0 \\
\lambda_i^* &\geq 0 \\
\lambda_i^* f_i(x^*) &= 0 \\
\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x^*) &= 0
\end{aligned}
$$

가 바로 그 KKT condition이 됩니다. 이때 λ_i^*와 ν_i^*는 적절히 이 조건을 만족시키도록 결정해 주어야 합니다. 참고로 이 condition은 $m + p + n$개의 제약을 걸고 있고, 추정해야 하는 모수도 λ에서 m개, ν에서 p개, x^*에서 n개이므로 잘 찾아줄 수 있습니다. 만약 optimization problem이 convex하다면, KKT condition을 만족하는 x^*에 대해 $f(x^*)$는 전역에서 최소가 됩니다.

Lagrange dual function은 아래와 같이 정의됩니다.

$$g(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu)$$

이때 이 dual function g는 원래 문제의 convexity에 상관없이, $\lambda > 0$에서 항상 concave합니다. 만약 $p^* = \inf_{x \in C} f_0(x)$로 원래 문제의 최솟값을 얻는다면, 우리는 항상

$$g(\lambda, \nu) \leq p^*$$

임을 알고 있습니다. 따라서 우리는 p^*을 찾는 대신 $g(\lambda, \nu)$의 최댓값을 찾아 $f_0(x)$의 하한선을 얻어줄 수 있습니다. 또한 g는 concave이므로, g를 maximize하는 문제는 convex function $-g$를 minimize하는 convex optimization 이기에 풀기 쉬운 경우가 많습니다.

Duality

따라서 **Lagrange dual problem**은 아래와 같이 정의됩니다.

$$\begin{cases} \text{maximize} & g(\lambda, \nu) \\ \text{subject to} & \lambda \geq 0 \end{cases}$$

또한 $d^* = \sup_{\lambda \geq 0} g(\lambda, \nu)$로 정의할 때, **weak duality**

$$d^* \leq p^*$$

가 성립하고 **duality gap** $p^* - d^*$은 항상 음이 아닌 실수입니다. 이제 원래 문제는 **primal problem**이라고 부르게 됩니다.

Duality

만약 $d^* = p^*$라면, 이때는 **strong duality**가 성립한다고 말합니다. 그러한 상황은 여러 가지가 있을 수 있으나, primal problem이 LP이거나, primal problem이 convex이면서 여러 좋은 조건들을 만족할 때(e.g. Slater's condition, Fenchel-Moreau theorem, Mangasarian-Fromovitz constraint qualification) 성립한다고 알려져 있습니다. nonconvex 문제들도 특정한 경우 이를 성립시킬 수 있습니다.

도움되는 링크들:

- 류경석 교수님 최적화 노트:
  `https://ernestryu.com/courses/optimization.html`
- 원중호 교수님 고급통계계산 노트:
  `https://won-j.github.io/M1399_000200-2023fall/`
- Boyd & Vandenbergue:
  `https://web.stanford.edu/~boyd/cvxbook/`

Dyanmic Programming

동적계획법(dynamic programming)은 여러 기에 걸쳐 이후 상황까지 고려해 최적화를 해야 할 때 사용되는 최적화 방법입니다. 대표적인 동적계획법 문제 중 하나는 **확률적제어이론**(stochastic optimal control theory)입니다.

Optimal control theory is a branch of control theory that deals with finding a control for a dynamical system over a period of time such that an objective function is optimized.

우리는 이 절에서 Merton의 ICAPM(Intertemporal Capital Asset Pricing Model, `https://www.jstor.org/stable/1913811`)을 유도해보고자 합니다.

Merton's Problem

먼저 우리는 두 개의 asset만 존재하는 상황을 고려해 보겠습니다. 이는 BS model에서의 상황과 동일합니다. 이때 **포트폴리오**(portfolio)는

$$h_t = (\pi_t, \phi_t)$$

으로 정의되며, 각각은 riskless asset과 risky asset의 구매량을 의미합니다. 즉 포트폴리오의 가격 V_t는

$$V_t = \pi_t G_t + \phi_t S_t$$

으로 정해집니다. 정해진 시간 $t \in [0, T]$에서, 우리는 적절히 에셋들을 사고 팖으로써 동적으로 π_t, ϕ_t를 조절할 수 있습니다.

Merton's Problem

어떠한 포트폴리오가 **자기금융**(self-financing)이라는 것은 자금의 흐름이 두 asset 사이에서만 이루어짐을 의미합니다. 혹은 포트폴리오의 가치 변화가 에셋의 가격 변화에 의해서만 이루어짐을 의미합니다. 즉

$$dV_t = \pi_t dG_t + \phi_t dS_t$$

입니다. 만약 $V_t = f(t, S_t)$라면, 이는

$$\pi_t = \frac{f(t, S_t) - f_s(t, S_t)S_t}{G_t}, \phi_t = f_s(t, S_t)$$

가 성립함이 알려져 있습니다. 이는 여러분이 알고 계신 델타헷징의 방법과 동일합니다.(greek $\Delta = \frac{\partial V_t}{\partial S_t} = f_s(t, S_t)$)

Merton's Problem

우리의 목표는 초기자본 x가 주어져 있을 때, 효용함수 u를 가지는 투자자가 가지는 기대효용을 최대화할 수 있도록 포트폴리오 h_t를 조절하는 방법을 공부하는 것입니다. 가장 기본적인 형태는

$$\begin{cases} \text{maximize} & \mathbb{E}[u(V_T)] \\ \text{subject to} & h: \text{ self-financing}, V_0 = x \end{cases}$$

으로 주어질 것입니다. 이러한 형태의 문제를 **머튼 문제**(Merton's problem)이라고 부릅니다. 따라서 여기에서는 기존과 달리 무한 차원의 h를 적절히 조정하여야 하기에, 풀기 매우 어렵습니다.

Merton's Problem

하나의 방법은 자기금융 포트폴리오의 동태

$$\begin{aligned} dV_t &= \pi_t dG_t + \phi_t dS_t \\ &= r\pi_t G_t dt + \mu\phi_t S_t dt + \sigma\phi_t S_t dB_t \\ &= (rV_t + (\mu - r)\phi_t S_t)dt + \sigma\phi_t S_t dB_t \end{aligned}$$

를 간단화하기 위하여 매 시점 t의 주식보유 비율을 나타내는 **제어 과정**(control process) $\alpha_t = \frac{\phi_t S_t}{V_t}$를 정의하고

$$dV_t = rV_t dt + (\mu - r)\alpha_t V_t dt + \sigma\alpha_t V_t dB_t$$

처럼 정리하는 것입니다. 이때 V_t는 초기자본 x, 제어과정 α의 포트폴리오가 갖는 시점 t에서의 가치라는 점에서 $V_t^{x,\alpha}$처럼 쓰기도 합니다.

Merton's Problem

이제 이를 더욱 일반화하여

$$\sup_{\alpha} \mathbb{E}\left[\int_0^T f(s, V_s^{x,\alpha}, \alpha_s)ds + g(V_T^{x,\alpha})\right]$$

를 푸는 방법을 확인해 보겠습니다. 이는 우리가 적절히 동태적으로 α를 제어함으로써 얻는 최대 기대효용 크기를 의미합니다. 다르게 말하면, 이 문제를 푸는 것은 아래의 의미가 있습니다.

- 최적 기대효용의 비교를 통하여 asset들의 선호구조를 확인할 수 있습니다.
- 이 값을 최대화하는 α를 찾아 실제 포트폴리오 운용 과정에서 사용할 수 있습니다.

Merton's Problem

이제 **이득함수**(gain function)

$$J(t,x,\alpha) = \mathbb{E}\left[\int_t^T f(s, V_s^{x,\alpha}, \alpha_s)ds + g(V_T^{x,\alpha})\right]$$

와 **가치함수**(value function)

$$v(t,x) = \sup_{\alpha} J(t,x,\alpha)$$

를 정의한다면, 가치함수는 **해밀턴-야코비-벨만 방정식**(Hamilton-Jacobi-Bellman equation; HJB)

$$\begin{cases} v_t + \sup_\alpha (f(t,x,\alpha) + \frac{1}{2}\sigma^2(t,x,\alpha)v_{xx} + b(t,x,\alpha)v_x) = 0 \\ v(T,x) = g(x) \end{cases}$$

의 근임이 알려져 있습니다.

Merton's Problem

해당 사실의 증명은 **동적계획법**(Dynamic Programming Principle; DPP)를 통합니다. 다만 저희가 보이기에는 너무 어려워서, 생략하도록 하겠습니다. 증명이 궁금하신 분은 앞서 소개한 박형빈 교수님 금융수학2 노트를 참고해 주세요.
우리의 일반적인 모형에서는,

$$b(t,x,\alpha) = rx + (\mu - r)\alpha x, \quad \sigma(t,x,\alpha) = \sigma\alpha x$$

으로 주어집니다. 그리고 이제 HJB equation을 다양한 PDE 해법, 혹은 수치적 해법으로 풀게 되면 $v(t,x)$를 얻게 되는 것입니다.

Intertemporal CAPM

ICAPM에서는 K명의 소비자이자 투자자가 있다고 생각합니다. 각각은 자신의 사망 시기 T_k와 wealth W_k에 대해

$$\mathbb{E}\left[\int_0^{T_k} u_k(c_k(s), s)ds + B_k(W_k(T_k), T_k)\right]$$

를 최대화시키는 방향으로 자신의 소비와 투자를 제어하려 합니다. 이때 u_k는 k번째 투자자의 생애 중 효용함수, B_k는 유산의 효용함수입니다. $c_k(s)$는 s 시점의 consumption입니다. 이제 $w_{k,i}$를 전체 부 중 i번째 asset에 투자한 비율이라고 보겠습니다. 0번째 asset은 riskless asset이고, $1 \sim n$번째 asset은 risky asset으로 총 $n+1$개의 asset이 존재합니다.

Intertemporal CAPM

그렇다면 우리는 자기금융포트폴리오의 정의에서

$$dW_k = (y_k - c_k)dt + \sum_{i=0}^{n} \left(\frac{w_{i,k} W_k}{S_{i,t}} \right) dS_{i,t}$$

를 얻습니다. y_k는 wage, c_k는 consumption을 의미합니다.

$$dS_{i,t} = \mu_i S_{i,t} dt + \sigma_i S_{i,t} dB_{i,t}$$

임을 고려하면, 아래처럼 W의 동태를 쓸 수 있습니다.

$$dW = \left(\sum_{i=1}^{n} w_{i,k}(\mu_i - r) + r \right) W_k dt + \sum_{i=1}^{n} w_{i,k} W_k \sigma_i dB_{i,t} + (y_k - c_k)dt$$

Intertemporal CAPM

$y = 0$으로 고정하고 Merton's problem을 적용하면,

$$w_{i,k} W_k = A_k \sum_{i=1}^{n} v_{ij}(\mu_i - r) + \sum_{j=1}^{m} \sum_{l=1}^{n} H_{l,k} \sigma_j g_l \eta_{jl} v_{ij}$$

이며, 이때 σ_{ij}는 i번째 asset과 j번째 asset의 covariance이고, v_{ij}는 $\Sigma = [\sigma_{ij}]$의 역행렬의 i행 j열 원소이며, 나머지는 적절한 상수라고 생각해주시면 되겠습니다. 더불어, 이득함수 J에 대하여

$$A = -\frac{J_W}{J_{WW}} = -\frac{u_c}{u_{cc}\frac{\partial c}{\partial W}} > 0$$

임을 보일 수 있습니다. 즉 이는 absolute risk aversion에 반비례하는 상수입니다. 따라서 위험회피성향이 커질수록 risk preminum은 작아지게 될 것입니다.

Intertemporal CAPM

즉 i번째 asset에 투자하는 양은

$$w_{i,k} W_k = A_k \sum_{i=1}^{n} v_{ij}(\mu_j - r) + \sum_{j=1}^{m} \sigma_j \times C_{j,k}$$

처럼 쓸 수 있는데, 첫째 항은 일반적인 CAPM에서 등장하는 수요함수입니다. 한편 둘째 항은 investment opportunity set에서 선호하지 않는 방향으로 움직이는 것을 헷지하기 위한 수요라고 취급할 수 있습니다. $C_{j,k}$의 부호는 미정이나, 그 변화량은 결국 특정 asset의 volatility에 의존합니다.

Intertemporal CAPM

만약 둘째 항들을 무시하고 다시 k를 불러오면, k번째 투자자는

$$w_{i,k} W_k = A_k \sum_{j=1}^{n} v_{ij}(\mu_j - r)$$

라는 수요를 가지게 됩니다. 한편 여기에서 two-fund를 가정하고 market portfolio가 균형에서 효율적임을 이용하면, $i = 1, 2, \cdots, n$에 대하여

$$\mu_i - r = \beta_i(\mu_M - r)$$

임을 보일 수 있고, $\beta_i = \sigma_{iM}/\sigma_M^2$으로 우리가 일반적으로 보아 온 market portfolio에 대한 CAPM 식과 동일함을 알 수 있습니다.

Intertemporal CAPM

둘째 항을 살린 채 적절히 정리하면,

$$w_{i,k} W_k = A_k \sum_{i=1}^{n} v_{ij}(\mu_j - r) + H_k \sum_{j=1}^{n} v_{ij}\sigma_{jr}$$

이 됩니다. 이때 $H_k = -(\partial c_k/\partial r)/(\partial c_k/\partial W_k)$이며, σ_{jr}은 j번째 asset과 interest rate 변화 사이의 correlation을 의미합니다. 만약 n번째 asset이 장기국채처럼 r에 완전히 음의 상관관계를 가지고 있다고 하면, r의 volatility g에 대하여

$$\sigma_{jr} = -g\sigma_{jn}/\sigma_n$$

이고 아래를 얻습니다.

$$w_{i,k} W_k = A_k \sum_{i=1}^{n} v_{ij}(\mu_i - r) \quad (i = 1, 2, \cdots, n-1)$$

$$w_{n,k} W_n = A_k \sum_{i=1}^{n} v_{nj}(\alpha_j - r) - gH_k/\sigma_n$$

Intertemporal CAPM

그 다음 $D_i = \sum_{k=1}^{K} w_{i,k} W_k$를 총수요로 보면,

$$D_i = A \sum_{i=1}^{n} v_{ij}(\mu_i - r) \quad (i = 1, 2, \cdots, n-1)$$

$$D_n = A \sum_{i=1}^{n} v_{nj}(\alpha_j - r) - Hg/\sigma_n$$

을 얻고, 적절한 논의를 통해

$$(\mu_i - r) = \frac{\sigma_i[\rho_{iM} - \rho_{in}\rho_{nM}]}{\sigma_M(1-\rho_{nM}^2)}(\mu_M - r) + \frac{\sigma_i[\rho_{in} - \rho_{iM}\rho_{nM}]}{\sigma_n(1-\rho_{Mn}^2)}(\mu_n - r)$$

이 $i = 1, 2, \cdots, n-1$에 대해 성립함을 알 수 있습니다. 예상할 수 있듯, ρ_{ij}는 두 asset 사이의 correlation입니다.

Intertemporal CAPM

이제 우리에게 익숙한 형태로 쓰면,

$$\alpha_i = r + \beta_{im}(\alpha_m - r) + \beta_{in}(\alpha_n - r)$$

입니다. 따라서 이는 특정 asset의 적정 수익률을 파악하는 데 이용될 수 있습니다. 또한 β_{im}과 β_{in}의 결정 과정과 추가항에서 볼 수 있듯이, 그 계산 과정 역시 CAPM에서의 β에, r의 변화를 헷징하는 n번째 bond 역시 중요한 기능을 하게 됩니다. 이는 위험회피적인 성향을 갖춘 개인들이 위험을 헷징하고 소비를 평탄화하려는 움직임을 묘사합니다.

Take-Home Messages

- ICAPM 부분은 솔직히 전혀 전달이 안 되었을 것으로 생각됩니다. 너무 많은 선행지식을 요구해서... 넣지 않으려 했으나,
  - 동적계획법/최적소비이론은 미시/거시하시는 분들께 도움이 되고
  - utility/multi-period를 고려한다는 점에서 금융팀에 아이디어도 줄 수 있을 것 같고
  - 금융수학쪽 내용을 살짝 전달할 수 있어서

  넣어봤습니다. 관심있으신 분들은 원 논문을 읽으시는 편이...
- 다른 부분도 엄청 간단히만 훑고 넘어갔는데, 저도 잘 몰라서가 가장 큰 이유입니다. 더 깊게 알고 싶으시면 책 하나 골라서 공부해 보심을 추천드립니다.
- 다음 세션은 중간고사 기간이 끝나고 한참 뒤인 4월 23일입니다.
- 오늘은 HW3이 없습니다.

Next Session

주제: 실험과 인과추론

- Statistical Inference in Experiment
  - ANOVA and Nonparametric Tests
- Introduction to Causal Inference
  - Confounding Effect and Endogeneity
  - Potential Outcome Framework
  - Natural Experiments
- Instrumental Variables
  - Endogeneity and Confounding Effects
  - IV, 2SLS and IV-GMM
- Matching, Weighting, and DR Estimator
  - Propensity Score
  - Matching, Weighting, and DR Estimator
  - Sensitivity Analysis
- Other Designs: DiD, Synthetic Control, and RDD