KAIST MFE, 2024 Fall

Kim Hyeonghwan

2024-09-02

Table of contents

Welcome!

안녕하세요, KAIST MFE 24년 가을학기에 이수한 과목의 과제 등을 정리해두었습니다.

Part I

머신러닝('24 가을)

머신러닝 1주차

하이퍼파라미터

머신러닝에 이용할 모델에 대한 파리미터(α, β 등)가 아닌,

학습알고리즘의 파라미터(학습률 등)

$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$

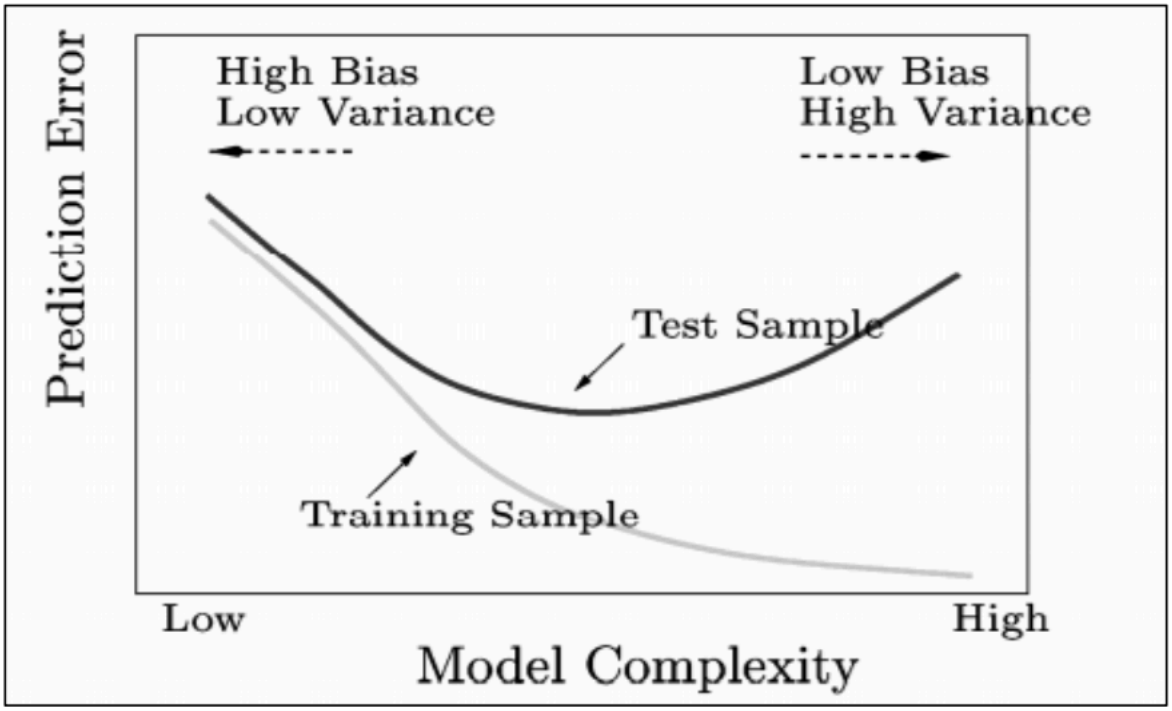
여기서, β_n은 파라미터이며 주어진 데이터를 학습하여 파라미터를 산출하는 것임.

근데 만약에 모델 성능 향상을 위해 각 β의 제약조건(constraint)를 정한다?

해당 제약조건은 하이퍼파라미터(hyper-parameter, h-para)가 되는 것임

이런 회귀분석을 릿지(Ridge regression)이라고 함.

모델의 평가와 검증



낮은 복잡도 = 선형회귀분석 or logistic 분류면 높은 복잡도 = 변수를 추가한 모델 (과대적합 케이스)

훈련데이터(training sample)은 복잡도가 높아질수록 예측오차가 줄어듬 (우하향)

평가데이터(text sample)은 복잡도가 높아지면 오차가 줄어들기는 하지만,

너무 복잡도가 높아지면 평가데이터에서는 오차가 오히려 발생함

즉, 일반화가 어렵고 과대적합(overfitting) 문제가 발생함

일반화 오차

평가데이터를 이용하였을 때 발생하는 오차를 **일반화 오차**라고 함

(Generalization error, test error)

$$= {}^{2} + \quad +$$

편향(Bias)

모집단에서 크기 m의 (x,y) 순서쌍을 샘플링할 때,

해당 샘플링을 n번 반복해서 모델링을 한다고 하면,

각각의 $f_1, ..., f_n$이 있을 것이고, $\bar{f} = mean(f_m)$이면,

실제 모집단을 나타내는 모델인 f_{true}와 $\bar{f}$의 차이를 편향(bias)이라고 합니다.

분산(Variance)

한편, $f_1, ..., f_n$의 추정모델간의 편차의 제곱합이 분산이 됩니다.

관계

즉, 모델이 단순할수록 실제로는 더 복잡한 모델을 잘 반영하기 어렵기때문에 편향이 큰 대신,

추정모델간의 오차는 작아지므로 분산이 작습니다.

하지만, 모델이 복잡할수록 추정모델을 평균하면 실제 모델과 유사해질 것 이므로 편향은 작고,

추정모델간의 오차는 클 것이므로 분산이 큽니다.

데이터의 분할 방법

Hold-out 방식

주어진 자료를 목적에 따라 훈련/검증/평가 데이터로 나누어서 활용.

(훈련, 검증이 8~90% / 평가가 1~20%)

검증데이터는 h-para tuning에 주로 사용함.

1. 각 h-para별로 훈련데이터를 통해 모델 도출
2. 각 모델에 대해 검증데이터를 이용해 평가 (MSE 산출)
3. 성능이 가장 좋은 h-para를 채택
4. 해당 h-para 및 훈련+검증데이터를 통해 최종모델 도출
5. 평가데이터를 이용해 최종모델을 평가하여 성능 확인

단점 : 전체 데이터에서 평가데이터는 따로 빼놔야해서 자료가 충분치 않으면 사용하기 애매함

K-fold 교차검증(Cross-validation) 방식을 이용한 검증

데이터가 그다지 많지 않을때 유용.

모든 데이터가 훈련, 검증, 평가에 활용될 수 있음.

주어진 자료를 K개로 분할하여 반복활용

(3-fold cv 예시)

1. 주어진 자료를 3개로 분할 (1,2 훈련 + 3 검증 / 1,3 훈련 + 2 검증 / 2,3 훈련 + 1 검증)
2. 각 분할데이터로 특정 h-para에 대해 훈련 + 검증데이터로 성능 평가(MSE)
3. 3개의 분할데이터의 성능의 평균이 해당 h-para의 검증결과임
4. 모든 h-para에 대해 1~3 반복
5. h-para의 검증결과 중 가장 성능이 좋은 h-para 채택
6. 다시 주어진 자료를 3개로 분할 (훈련+평가)
7. 각 분할데이터로 훈련 및 평가를 통해 성능 평가
8. 성능의 평균값이 우리의 모델의 성능임.

방법론에 따라 한꺼번에 훈련시켜서 성능을 평가하기도 하고,

이러한 분할(folding)을 수회~수백회 반복해서 모델의 성능을 추정하기도 함.

(folding별 성능의 평균/표준편차 고려)

::: {.callout, title="Imbalanced data"} 머신러닝으로 분류문제를 해결하는 경우,

실제 세상에서는 분류대상의 비율이 매우 적은 경우가 많음.

이러한 샘플을 imbalanced data라고 하며,

Hold-out, K-fold cv 등을 할 때,

원 자료의 분류대상의 비율을 유지한채로 주어진 자료를 분할해야 함. :::

Part II

딥러닝('24 가을)

딥러닝 1주차

머신러닝 vs. 딥러닝?

암튼 머신러닝이란건 기계가 스스로 학습해서 문제를 해결한다는 의미.

우리가 컴퓨터에게 적정한 모델을 입력하면, 해당 모델을 가지고 주어진 데이터를 반복계산하여 적절한 결과값을 도출.

딥러닝은 머신러닝의 한 분류로, 적정한 모델을 인공신경망 기반의 모델을 사용한 것을 의미한다고 보면 됨.

인공신경망이 뭔지는 아직 모름. 매우 복잡하고, 적절한 결과값 도출 과정을 해석하기 어렵고, 예측력은 매우 높음.

학습데이터가 매우 많이 필요하고, 많은 계산시간, 모델 탐색시간, 하이퍼파라미터 조율시간 등이 필요함.

오류에 대한 디버깅이나 해석도 어렵다고 함.

머신러닝 알고리즘의 구분

지도학습

입력값에 대한 결과값이 주어진 경우,

모델이 입력값과 결과값을 모두 알고 학습하여 새로운 입력값에 대한 적정 결과값 추정치를 제공하게 됨.

결과값(label)이 숫자형이면 회귀(regression) 알고리즘, 범주형이면 분류(classification) 알고리즘으로 구분.

비지도학습

결과값이 없고, 주어진 입력값만으로 학습함.

의미있는 패턴을 추출하는 것이 목적.

군집화(행을 묶음) 및 차원축소(열을 묶어 열의 갯수를 감소시킴)에 주로 활용

강화학습

모델 자체가 어떠한 변화를 주도하는데,

해당 변화에 따른 보상/패널티를 주는 환경을 구성함.

모델은 이러한 변화에 따른 누적보상이 최대가 되도록하는 변화패턴을 학습함

지도학습 알고리즘의 절차

1. 전처리 및 탐색
2. 적절한 모델 선택
3. 주어진 데이터로 모델 훈련
4. 새로운 데이터를 통해 결과값을 예측하여 모델의 성능을 평가

경사하강법 (Gradient Descent Method)

다차원 선형회귀 모형에서 모델결과값과 실제결과값의 차이(MSE; Mean Square Error)를 최소화하는 방식

MSE의 미분계수(Gradient)의 일정수준(학습률)만큼 선형회귀모형의 각 파라미터가 모두 감소하도록 지속 업데이트하면서,

결과적으로 MSE의 Gradient가 0이 되면 학습이 종료되는 방식.

MSE의 미분계수가 0이면 최솟값이고, 모델결과값의 오차가 최소가 되므로 가장 적정한 파라미터를 추론한 것으로 판단.

경사하강법 종류

한번의 회귀계수 업데이트에 모든 훈련데이터를 사용하면 배치 경사하강법

임의추출로 하나의 훈련데이터만 사용하면 확률 경사하강법(SGD)

일부만 사용하면 미니 경사하강법.

많이 사용할수록 규칙적으로 MSE가 감소하고 일관적으로 움직이나, 산출시간이 오래걸리고, 지역최소값에 갇힐 가능성이 높아짐.

모델 평가 및 검증

low bias - high vol

high bias - low vol

제대로 못들어서 정리 필요

자료의 구분

일반적으로 전체 데이터를 임의로 3개로 나눔.

훈련 데이터 / 검증 데이터 / 평가 데이터

e.g. 훈련데이터로 여러 h-para에 대해 모델 돌림

검증데이터를 이용해 각 h-para별 성능 평가

제일 좋은 h-para로 모델을 구성하여 훈련+검증데이터로 다시 훈련

최종 모델을 평가데이터를 이용하여 평가

Part III

시뮬레이션 방법론('24 가을)

시뮬레이션방법론 Ch1

몬테카를로 시뮬레이션 기초

블랙숄즈공식 예시

1. $f_t + \frac{1}{2}\sigma^2 S^2 f_{ss} + rSf_s - rf = 0$

-> 수치해석적인 방법으로 풀게 됨, FDM(Finite Difference Method)

2. $P(0) = e^{-rT}E^Q[P(T)]$

-> 마팅게일, 몬테카를로 시뮬레이션(Montecarlo simulation, MCS)을 주로 사용함

Volume과 적분

$x \sim uniform[0,1]$, $\alpha = E[f(x)] = \int_0^1 f(x)dx$

그러나, MCS를 이용하는 경우 임의변수 $x_1, x_2, ..., x_n$을 샘플링하여 $\hat{\alpha} = \frac{1}{N}\sum_i^N f(x_i)$로 산출함

두 값이 정확히 일치하지는 않지만, 표본이 커질수록 그 오차는 0으로 수렴함($\alpha \approx \hat{\alpha}$)

이는 **대수의 법칙과 중심극한정리**에 따라 수학적으로 정의할 수 있음

> 중심극한정리
>
> 표본평균($\hat{\alpha}$)은 정규분포를 따르므로, $\hat{\alpha} - \alpha \sim N(0, \frac{\sigma^2}{N})$
> 즉, 표본의 크기가 커질수록 두 차이는 0으로 수렴함(probibility convergence)
> 오차의 표준편차는 $\frac{\sigma}{\sqrt{N}}$이므로, 표본의 크기가 100배 증가하면 오차의 표준편차는 10배 감소함

이외에도 간단힌 사다리꼴(trapezoidal) 방식을 이용해볼 수 있음.

3. $\alpha \approx \frac{f(0)+f(1)}{2n} + \frac{1}{n}\sum_{i=1}^{n-1} f(\frac{i}{n})$ ( )

이는 매우 간단하고 효율적인 방법이지만, 변수가 늘어날 수록 효율이 급감함.

MCS 기초

몬테카를로 시뮬레이션을 개념적으로 설명함

예를 들어, 1X1 사각형에 내접한 원에 대하여, 사각형 안에 임의의 점을 찍을 때 원에 포함될 확률?

직관적으로 면적을 통해 $\Pi/4$임을 알 수 있음.

이를 다변수, 다차원, 복잡한 함수꼴로 확장한다면 면적을 구하는 적분을 통해 구할 수 있음을 의미함.

근데 그런 복잡한 계산 대신에 랜덤변수를 생성해서 시행횟수를 수없이 시행하고,

원(면적) 안에 속할 확률을 구한다면? 이게 몬테카를로 시뮬레이션의 기초임.

수없이 많은 (x, y)를 생성하고, 좌표평면의 1X1 사각형에 대해 원안에 속할 확률은 $x^2 + y^2 < 1/4$임.

이러한 확률을 구하는 것은 기대값으로 표현할 수 있게 되고, 결국 이 확률은 $\Pi/4$로 수렴

$$Pr(x \in B) = E(\int_A 1_B) = \Pi/4$$

확률기대값 및 원주율 계산 예시

```
import numpy as np
n = 10000
x = np.random.rand(n) # uniform random number in (0,1)
x -= 0.5
y = np.random.rand(n)
y -= 0.5

d = np.sqrt(x**2+y**2)
i = d<0.5
prob = i.sum() / n
pi = 4 * i.sum() / n

print(prob,pi,sep="\n")
```

```
0.7841
3.1364
```

표본표준편차 계산 : numpy는 n으로 나누고, pandas는 n-1로 나누는 것이 기본

```
import pandas as pd
np_s = i.std()
pd_s = pd.Series(i).std()
np_s_df1 = i.std(ddof=1)
print(np_s, pd_s, np_s_df1, sep = "\n")
```

```
0.41144524544585515
0.4114658192511757
0.4114658192511757
```

표준오차 계산 및 95% 신뢰구간 계산

```
se = pd_s / np.sqrt(n)
prob_95lb = prob - 2*se
prob_95ub = prob + 2*se
pi_95lb = prob_95lb*4
pi_95ub = prob_95ub*4
print(se, pi_95lb, pi_95ub, sep="\n")
```

```
0.004114658192511757
3.103482734459906
3.1693172655400943
```

경로의존성 (Path-dependent)

일반적인(Plain vanilla) 옵션은 pay-off가 기초자산의 만기시점의 가격 $S(T)$에 의해서만 결정되므로,

그 사이의 기초자산의 가격을 생성할 필요는 없음(0~T)

그러나, 아시안옵션 등은 $S(T)$ 뿐만 아니라 그 과정에 의해서 pay-off가 결정되므로 그 경로를 알아야 함.

또한, 블랙숄즈의 가정이 성립하지 않는 경우 모델링을 하기 위해서도 그 경로를 알아야 할 필요가 있음.

이를 **경로의존성**이라고 함.

시뮬레이션 예시

일반적인 주가에 대한 확률과정이 GBM을 따른다면,

$dS(t) = rS(t)dt + \sigma S(t)dW(t)$

그러나, 변동성이 주가에 따라 변하면 주가의 흐름에 따라 변동성이 바뀌므로 경로의존성이 발생

즉, $dS(t) = rS(t)dt + \sigma(S(t))S(t)dW(t)$를 따르게 되므로

우리가 앞서 사용한 $S(T) = S(0)e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z}$를 사용할 수 없음.

따라서, Analytic solution이 없으므로 근사치를 구할 수 밖에 없으며 그 예시로 **이산오일러근사**가 있음

(0~T) 구간을 m개로 나누고, 각 구간의 길이 $\frac{T}{m} = \Delta t$라고 하면 기초자산의 경로 $S(t)$는,

$$S(t + \Delta t) = S(t) + rS(t)\Delta t + \sigma(S(t))S(t)\sqrt{\Delta t}Z$$

다만, 이러한 경우에는 그 경로의 길이를 얼마나 짧게 구성하는지에 따라 시뮬레이션 정밀도에 영향을 미침.

즉, 시뮬레이션 횟수 n과 경로의 길이 m이 모두 정확도를 결정하는 파라미터가 됨.

MCS 추정치 개선 방향

MCS의 효율성은 아래 3개의 기준에 따라 평가할 수 있습니다.

1. 계산시간 (Computing time)
2. 편의 (Bias)
3. 분산 (Variance)

여기서, 시뮬레이션의 $Prediction\ error\ =\ Variance\ +\ Bias^2$

> $Var[\epsilon] = E[\epsilon^2] - (E[\epsilon])^2$
> $MSE = E[\epsilon^2) = Var[\epsilon] + (E[\epsilon])^2 = Variance + Bias^2$

분산감소와 계산시간

시행횟수가 증가하면 분산은 감소함. ($n \to \infty, Var[\epsilon] \to 0$)

한번의 시뮬레이션에 정확한방법을 사용할 수록 편의는 감소함($m \to \infty, Bias \to 0$)

(정확한방법을 사용할 수록 분산은 증가할 수 있음 (머신러닝 overfitting 같은 문제?))

(정확한방법을 쓸수록 계산비용이 증가하여 시뮬레이션 횟수가 감소함, 분산이 그래서 증가함)

시뮬레이션의 횟수

계산 예산에 s이고, 한번의 시뮬레이션의 계산량이 τ일 때, 가능한 시뮬레이션 횟수는 s/τ임

이 때, 추정치의 분포 $\sqrt{\frac{s}{\tau}}[\hat{C} - C] \to N(0, \sigma_c^2)$

$\Rightarrow [\hat{C} - C] \to N(0, \sigma_c^2(\frac{\tau}{c}))$ 이므로,

계산오차는 분산이 $\sigma_c^2(\frac{tau}{c})$인 정규분포에 수렴함을 의미

편의

경로의존성이 있는 시뮬레이션 중, 과거 연속적인 수치에 따라 pay-off가 정해진다면,

이산오일러근사를 사용할 때 편의가 발생함.

e.g. 룩백옵션의 경우 시뮬레이션이 항상 실제 pay-off를 과소평가 = (-) bias 존재

이 때, 이산구간의 간격 m을 작게할 수록 편의는 감소함.

또는, 기초자산이 비선형구조인 경우 등에도 편의가 발생할 수 있음.

e.g. Compound 옵션의 경우 기초자산인 옵션 가격이 비선형이므로,

Compound 옵션을 Analytic solution을 적용하여 푸는 경우 항상 실제 옵션보다 가격이 높음 = (+) bias 존재

이 때, $T_1 \sim T_2$의 n_2개의 경로를 추가로 생성하여 경로를 이중으로 구성한다면 bias 제거가 가능함.

Asian Option 평가 해볼 것

Part IV

이자율파생상품('24 가을)

이자율파생상품 1주차

Part V

수치해석학('24 가을)

수치해석학 Ch1

금융 수치해석의 소개

강의 개요 : 금융수치해석의 필요성

주로 파생상품 평가와 최적화 방법론에 대해서 다룰 예정

파생상품 평가

$ds = rSdt + \sigma SdW^Q$

기하학적 브라운운동을 따르는 기초자산에 대한 파생상품의 가격 $f(t, S)$는 아래의 PDE로 표현됨

$f_t + \frac{1}{2}\sigma^2 S^2 f_{ss} + rSf_s - rf = 0$

이 블랙숄즈 미분방정식을 컴퓨터로 풀어내는 것이 주요 내용임

여기에는 반드시 연속적인 수식을 이산화하는 과정이 필요하며, 다양한 수치해석적인 기법이 활용됨

대표적으로 유한차분법(Finite Difference Method, FDM)이 존재

최적화 방법론

이외의 다양한 최적화방법론은 시간이 여유롭다면 이것저것 다룰 예정

- Minimum Variance Portfolio : Single-period에 대해 Sharpe ratio 극대화 등
- Stochastic programming : Multi-period에 대해 Minimum var 문제 해결 등
- Non-convex optimization : 미분을 통해 극값을 산출할 수 없는 경우의 최적화
- Parameter estimation 또는 Model calibration : $min_{\theta,\sigma,k} \sum (model\ price - market\ price)^2$와 같은 문제 등

컴퓨터 연산에 대한 이해

수치해석기법을 사용할 때 필연적으로 오차(error) 발생

1. Truncation error : 연속적인 수학적인 모델을 이산화하면서 발생하는 오차(e.g. 미분계수)
2. Rounding error : 컴퓨터 시스템상 실수(real number)를 정확히 표현할 수 없는 데에서 기인(2진법 vs. 10진법)

```
import numpy as np

a = 0.1

print(a+a+a==0.3,a+a+a+a==0.4)
```

```
False True
```

Rounding error 관련

컴퓨터가 실수를 나타내는 방법은 일반적으로 $x = \pm n \times b^e$ 로 나타냄.

여기서 n은 가수, e는 지수이며, 일반적으로 밑인 b는 2를 사용함.

컴퓨터에서 많이 사용하는 float타입 실수는 32bit를 사용하여 실수를 표현하며,

이는 $2^3$2가지로 모든 실수를 표현하게됨을 의미함. (정수는 int타입으로 모두 표현가능)

따라서 소수점에 따라 정확한 값을 나타내지 못하는 문제는 항상 존재.

Precision of floating point arithmetic

실수표현의 정밀도는 $float(1 + \epsilon_{math}) > 1$이 되는 가장 작은 ϵ_{math}를 의미

```
e = 1
while 1 + e > 1:
    e = e/2
e_math = 2 * e
print(e_math)
```

```
2.220446049250313e-16
```

내장함수 활용 가능. 파이썬에서는 기본적으로 64bit double타입을 사용함

```
import numpy as np
print(np.finfo(np.double).eps,
      np.finfo(float).eps)
```

```
2.220446049250313e-16 2.220446049250313e-16
```

```
print(1+e, 1+e+e, 1+2*e, 1+1.0000001*e)
```

```
1.0 1.0 1.0000000000000002 1.0000000000000002
```

많이 쓰이는 double타입의 경우 64bit로 실수를 표현하는데,

$x = \pm n \times 2^e$ 에서 부호(±) 1자리, 가수(n) 52자리, 지수 11자리(e)를 의미

계산오차

절대오차 : $|\hat{x} - x|$

상대오차 : $\frac{|\hat{x}-x|}{|x|}$

결합오차 : $e_{comb} = \frac{|\hat{x}-x|}{|x|+1}$

유한차분을 이용한 도함수의 근사

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

컴퓨터로는 $h \to 0$을 정확히 표현할 수 없음.

따라서, 적당히 작은 값으로 이를 대체하여 $f'(x)$를 근사해야함.

1. Truncation error 최소화를 위해서는 h는 작을 수록 좋음
2. 그러나, 너무 작은 값을 선택하면 rounding error가 발생하여 $x = x + h$ 될 가능성

Taylor expansion

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}$$

이를 도함수에 적용하면,

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots + \frac{h^n}{n!}f^{(n)}(x) + R_n(x+h)$$

$n = 1$을 적용하면,

$\Rightarrow\ f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\xi)\ for\ \xi \in [x, x+h]$

$\Rightarrow\ f'(x) = \frac{f(x+h)-f(x)}{h} - \frac{h}{2}f''(\xi)\ (Forward\ Approximation)$

$n = 2$를 적용하고 forward - backward를 정리하면,

$f'(x) = \frac{f(x+h)-f(x-h)}{h} - \frac{h^2}{3}f'''(\xi)\ (Central\ Difference\ Approximation)$

::: {.callout, title="Central Difference Approximation"} $for\ n = 2$,

$(Forward)\ f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3!}f'''(\xi_+),\ \xi \in [x, x+h]$

$(Backward)\ f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{3!}f'''(\xi_-),\ \xi \in [x-h, x]$

$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^2}{6}\{f'''(\xi_+) + f'''(\xi_-)\}$

$\Rightarrow\ f'(x) = \frac{f(x+h)-f(x-h)}{h} - \frac{h^2}{3}f'''(\xi),\ \xi \in [x-h, x+h]$ :::

위의 식에서 볼 수 있는 것처럼, Central 방식에서는 truncation error의 order가 h^2이므로,

다른 방식에 비해서 오차가 훨씬 줄어들게 됨

유사한 방식으로 이계도함수와 편도함수를 유도하면,

$f''(x) = \frac{f(x+h)+f(x-h)-2f(x)}{h^2} - \frac{h^2}{24}f^{(4)}(\xi)$

$f_x(x,y) = \frac{f(x+h_x,y)-f(x-h_x,y)}{2h_x} + trunc.\ error$

총오차 및 최적의 h 산출

Forward difference approximation을 사용하고, $|f''(x)| <= M$이라고 하면,

$|f'_h(x) - f'(x)| = \frac{h}{2}|f''(x)| <= \frac{h}{2}M\ (trunc.\ error)$

유인물 참조

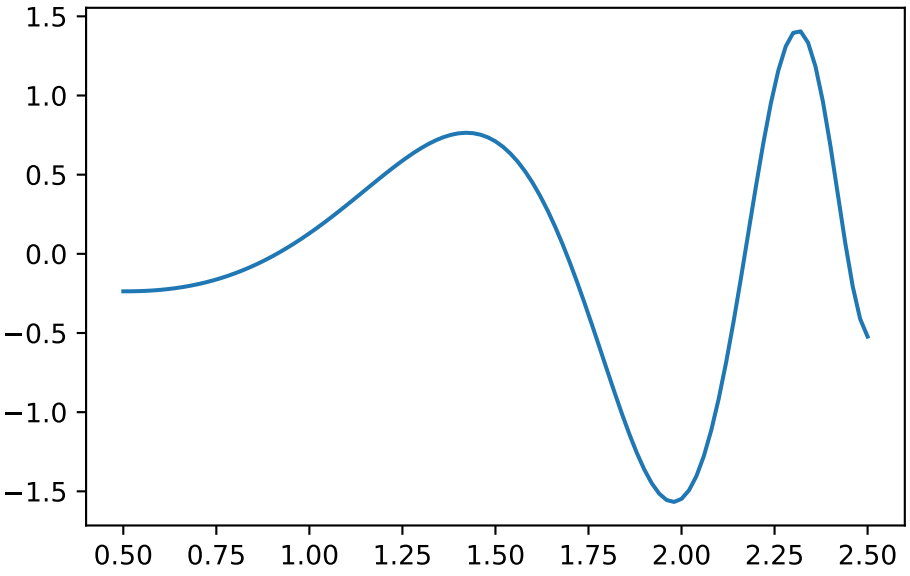
총오차 최소화를 위한 h^* 산출이 목표

유한차분을 이용한 도함수 근사 예시

$f(x) = cos(x^x) - sin(e^x)$

함수 및 도함수(***analytic form***) 정의 및 도식화

```
import numpy as np
import matplotlib.pyplot as plt
def fun(x):
    return np.cos(x**x) - np.sin(np.exp(x))

def fprime(x):
    return -np.sin(x**x)*(x**x)*(np.log(x)+1)  - np.cos(np.exp(x))*np.exp(x)

x = np.linspace(0.5,2.5,101)
y = fun(x)
plt.plot(x,y,'-')
```



미분계수 산출

```
x = 1.5
d = fprime(x)
print("derivative = ", d)
```

```
derivative =  -1.466199173237208
```

***forward* 및 *central difference approx.* 산출 및 비교, 총오차를 *log scale*로 표현**

trunc. error는 h가 작아질수록 감소하지만 특정구간 이후에는 rounding error가 발생하므로

총오차는 항상 감소하지 않게 됨.

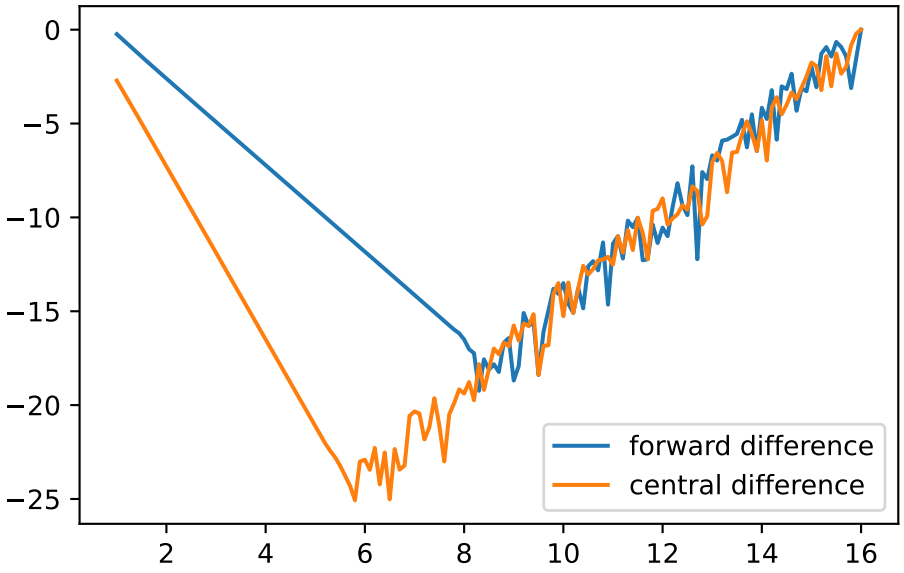
최적 h^*를 찾는 것이 매우 중요함

```
p = np.linspace(1,16,151)
h = 10**(-p)

def forward_difference(x,h):
    return (fun(x+h)-fun(x)) / h

def central_difference(x,h):
    return (fun(x+h)-fun(x-h)) / (2*h)

fd = forward_difference(x, h)
cd = central_difference(x, h)
print("forward = ", fd)
print("central = ", cd)

fd_error = np.log(np.abs(fd-d)/np.abs(d))
cd_error = np.log(np.abs(cd-d)/np.abs(d))
plt.plot(p,fd_error, p, cd_error)
plt.legend(['forward difference', 'central difference'])
```

```
forward =  [-2.62212289 -2.37366424 -2.17930621 -2.02733993 -1.90838758 -1.81511559
 -1.74184228 -1.68417626 -1.63872005 -1.60283855 -1.57448171 -1.55204964
 -1.53429022 -1.52022092 -1.50906909 -1.50022597 -1.49321118 -1.48764519
 -1.48322779 -1.47972134 -1.47693759 -1.47472735 -1.4729723  -1.4715786
 -1.47047179 -1.46959277 -1.46889464 -1.46834015 -1.46789975 -1.46754995
 -1.4672721  -1.46705142 -1.46687612 -1.46673689 -1.46662629 -1.46653844
 -1.46646866 -1.46641323 -1.46636921 -1.46633424 -1.46630646 -1.46628439
 -1.46626686 -1.46625294 -1.46624188 -1.4662331  -1.46622612 -1.46622058
 -1.46621618 -1.46621268 -1.4662099  -1.4662077  -1.46620594 -1.46620455
 -1.46620344 -1.46620257 -1.46620187 -1.46620131 -1.46620087 -1.46620053
 -1.46620025 -1.46620002 -1.46619985 -1.46619971 -1.46619959 -1.46619951
```

```
 -1.46619944 -1.46619939 -1.46619934 -1.46619931 -1.46619927 -1.46619923
 -1.46619922 -1.46619918 -1.46619921 -1.46619915 -1.46619915 -1.46619919
 -1.46619909 -1.46619907 -1.46619916 -1.46619915 -1.46619876 -1.46619938
 -1.46619893 -1.46619919 -1.46619932 -1.46619869 -1.46619769 -1.4662003
 -1.46619716 -1.46619985 -1.46619876 -1.46620071 -1.46619865 -1.46619427
 -1.46619269 -1.46620314 -1.46618158 -1.46619854 -1.46618273 -1.46617469
 -1.46619173 -1.46614311 -1.46615961 -1.46626449 -1.46620595 -1.46619201
 -1.46615356 -1.46621617 -1.46616053 -1.46617469 -1.46608615 -1.46578868
 -1.46632693 -1.46612406 -1.46518938 -1.46619201 -1.46545305 -1.46568705
 -1.46438417 -1.46757238 -1.46221506 -1.46202287 -1.46130718 -1.46050672
 -1.45413969 -1.46897416 -1.45004198 -1.46392328 -1.44328993 -1.4535955
 -1.40766793 -1.46202287 -1.39437708 -1.40433339 -1.32596324 -1.44671698
 -1.40100674 -1.41101039 -1.66533454 -1.39768798 -1.05575095 -0.88607446
 -1.11550166 -0.70216669 -0.88397549 -1.11285921 -1.40100674 -1.76376299
  0.        ]
central =  [-1.5635526  -1.52856423 -1.50592274 -1.49141188 -1.48216656 -1.4762975
 -1.47258018 -1.47022905 -1.46874334 -1.46780503 -1.46721263 -1.46683872
 -1.46660274 -1.46645382 -1.46635985 -1.46630056 -1.46626314 -1.46623954
 -1.46622464 -1.46621524 -1.46620931 -1.46620557 -1.46620321 -1.46620172
 -1.46620078 -1.46620019 -1.46619981 -1.46619958 -1.46619943 -1.46619933
 -1.46619927 -1.46619924 -1.46619921 -1.4661992  -1.46619919 -1.46619918
 -1.46619918 -1.46619918 -1.46619918 -1.46619917 -1.46619917 -1.46619917
 -1.46619917 -1.46619917 -1.46619917 -1.46619917 -1.46619917 -1.46619917
 -1.46619917 -1.46619917 -1.46619917 -1.46619917 -1.46619917 -1.46619917
 -1.46619917 -1.46619917 -1.46619917 -1.46619917 -1.46619917 -1.46619917
 -1.46619918 -1.46619917 -1.46619917 -1.46619917 -1.46619917 -1.46619917
 -1.46619917 -1.46619917 -1.46619918 -1.46619918 -1.46619917 -1.46619916
 -1.46619918 -1.46619915 -1.46619918 -1.46619915 -1.46619923 -1.46619922
 -1.46619909 -1.46619924 -1.46619938 -1.46619908 -1.46619894 -1.46619938
 -1.46619879 -1.46619919 -1.4661991  -1.46619925 -1.46619804 -1.46620118
 -1.46619883 -1.46620125 -1.46619876 -1.46620071 -1.46620423 -1.46619603
 -1.4661949  -1.46620592 -1.46619208 -1.46620736 -1.46619383 -1.46617469
 -1.46620932 -1.46616527 -1.4661875  -1.46626449 -1.46622805 -1.46619201
 -1.46629366 -1.46630436 -1.46638257 -1.46624457 -1.46626211 -1.46612096
 -1.46632693 -1.4662996  -1.46585236 -1.46647023 -1.46615356 -1.46612799
 -1.46493928 -1.46827122 -1.46485444 -1.46645324 -1.46409593 -1.46401756
```

```
-1.4607695  -1.47732061 -1.46054953 -1.46392328 -1.45439216 -1.46757238
-1.44285963 -1.50632659 -1.45015216 -1.43944172 -1.41436079 -1.50235994
-1.40100674 -1.58738669 -1.72084569 -1.67722557 -1.40766793 -1.10759308
-1.39437708 -1.05325004 -1.32596324 -1.66928882 -2.10151011 -2.64564449
 0.        ]
```



Part VI

금융시장 리스크관리('24 가을)

금융시장 리스크관리 1주차

Lecture2 : How Traders Manage Their Risks?

Greeks letters & Scenario analysis

Delta hedging

$\Delta = \frac{\partial P}{\partial S}$

가장 기본적인 헷징방법으로, 파생상품과 같은 금융상품으로 구성된 포트폴리오에 대해

기초자산의 가격변동에 대한 민감도인 델타를 계산하여 이를 0으로 만듦으로써

기초자산의 가격변화로 인한 포트폴리오의 가치변화를 0으로 만드는 방법.

포트폴리오의 payoff가 선형이라면, 한번의 헷징만으로 완전헷지(perfect hedge)가 가능

이를 Hedge and forget이라고 함.

그러나 비선형이라면, 기초자산의 가격변동에 따라 델타도 변하게 됨.

> **델타헷지 예시**
>
> 은행이 특정 주식 10만주에 대한 콜옵션을 30만불에 매도할 수 있음.
> 블랙숄즈공식에 따른 이 옵션의 가치는 24만불 ($S_0 = 49, K = 50, r = 0.05, \sigma = 0.2, T = 20w$)
> 어떻게 6만불의 차익거래를 실현시킬지?
>
> 1. 풋콜페리티 또는 시장에서 동일한 옵션을 24만불에 매수하여 실현
> 2. 그러나, 옵션매수가 불가능한 경우 기초자산 주식을 이용한 델타헷징을 반복
>
> 즉, 옵션 매도포지션의 델타만큼 주식을 매수하고 매주 리밸런싱
> 20주 후 주식 매도수를 반복하여 구축한 델타헷징은 약 26만불의 비용이 발생하였음
> -> 약 4만불의 차익거래를 실현함
> 2만불은 어디로 증발함? : 델타헷징에 드는 비용 (거래비용 등)

> 헷지를 자주할수록, 거래비용이 적을수록, 기초자산의 가격변동이 작을수록 차익은 6만불로 수렴

기초자산을 이용한 델타헷징은 비용이 발생할수밖에 없음.

콜옵션을 기준으로 할 때, 기초자산의 가격이 상승하면 콜옵션의 머니니스가 증가하면서 델타가 증가함.

콜옵션 매도를 델타헷징하다보면, 주가 상승 -> 델타 상승 -> 주식 매수

반대로, 주가 하락 -> 델타 감소 -> 주식 매도

즉, 주식이 오르면 팔고 내리면 팔아야함 (Sell low, Buy high Strategy)

기타 그릭스

Gamma ($\Gamma = \frac{\partial \Delta}{\partial S} = \frac{\partial^2 P}{\partial S^2}$)

베가 로 그런거는 대충넘어갔음

Taylor Series Expansion

테일러 전개는 다항전개식의 일종으로, 복잡한 함수를 다항함수를 이용하여 간단히 전개할 수 있어 근사식에 많이 활용

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \cdots$$

금융시장에서 이를 적용한다면? $f(x)$는 포트폴리오의 가격함수이며, x는 기초자산가격으로 대입 가능

$\Rightarrow f(x) - f(x_0) = f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2$

$\Rightarrow \Delta f(x) = f'(x_0)\Delta x + \frac{1}{2}f''(x_0)\Delta x^2$

기초자산의 변화(Δx)에 따른 포트폴리오 가치변화(Δf)는 델타(듀레이션) 및 감마(컨벡시티)로 근사 가능

포트폴리오 P를 기초자산의 가격 및 시간에 따른 함수 $P(S, t)$라고 한다면, (변동성은 상수로 가정)

$$\Delta P = \frac{\partial P}{\partial S}\Delta S + \frac{\partial P}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 P}{\partial S^2}\Delta S^2 + \frac{1}{2}\frac{\partial^2 P}{\partial t^2}\Delta t^2 + \frac{\partial^2 P}{\partial S \partial t}\Delta S \Delta t + \cdots$$

일반적으로 $\Delta t^2 = 0, \Delta S \Delta t = 0$으로 가정하므로,

$$\Rightarrow \Delta P \approx \frac{\partial P}{\partial S}\Delta S + \frac{\partial P}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 P}{\partial S^2}\Delta S^2$$

즉, 포트폴리오의 가치변화는 델타, 세타, 감마로 표현되며 델타중립 포트폴리오를 구성했다면,

$$\Delta P = \Theta \Delta t + \frac{1}{2}\Gamma \Delta S^2$$

> **Note**
>
> 아래로 볼록한 형태인 **옵션 매수는 positive gamma**,
> 위로 볼록한 형태인 **옵션 매도는 negative gamma** (관리 어려움)

만약 변동성이 변수라면?

$\Delta P = \delta \Delta S + Vega \Delta \sigma + \Theta \Delta t + \frac{1}{2}\Gamma \Delta S^2$

Hedging in practice

델타헷징은 보통 매일하고, 감마나 베가는 영향이 매우 크지는 않아서 모니터링하다가,

일정 임계치를 넘어가면 헷지 시작(헷지도 어렵고 비용도 보다 많이 듬)

특히, 만기가 임박한 ATM옵션은 감마와 베가가 매우 크므로, 주로 관리하게됨

Part VII

미시경제학('24 가을)

미시경제학 Ch1

Demand, Supply, Elasticity

Law of demand

가격이 상승하면 수요는 하락하고, 가격이 하락하면 수요는 증가한다

when Other things equal(ceteris paribus)

따라서, x축이 수요량이고 y축이 가격일 때 수요곡선은 우하향함

Other things?

이는 수요곡선 자체를 변화시키는 모든 변수 일체를 의미함.

1. Income : normal goods vs. inferior goods
2. Number of buyers
3. Substitutes vs. Complements
4. Tastes

1. Income

일반적인 재화는 소득이 증가하면 수요가 증가함.

즉, 수요곡선을 오른쪽으로 평행이동시킴 ***Normal Goods***

그러나, 대중교통이나 감자같은 재화는 소득이 증가하면 오히려 수요가 감소할 수 있음.

이 경우는 수요곡선을 왼쪽으로 평행이동시킴 ***Inferior Goods***

이는 재화에 따라 고정되어있지 않으며,

때로는 소득수준이 증가함에 따라 수요가 증가하는 Normal goods였다가 소득수준이 더 크게 증가하면 수요가 감소하는 Inferior goods가 되기도 함. (e.g. Hamburger)

2. Substitutes vs. Complements

대체제(e.g. 맥도날드, 버거킹)는 대체관계에 있는 재화들로, 대체제의 수요가 감소하면 재화의 수요가 증가함.

즉, 대체제의 가격상승은 재화의 수요곡선을 우측 평행이동시킴

보완재(자동차, 기름)은 상호보완관계에 있는 재화로, 보완재의 수요가 증가하면 재화의 수요가 감소함. 보완재 가격상승은 수요곡선 좌측 평행이동

이외의 수요곡선을 이동시키는건 매우 많을 수 있음

중요한건, 특정재화의 수요에 영향을 미치는 방법은

1. 다른 요소를 건드려서 수요곡선 자체를 이동시키거나
2. 재화의 가격을 건드려서 수요곡선 내에서 이동시키는거임

Law of Supply

공급의 법칙

똑같음. 가격이 올라가면 공급이 늘어나고 감소하면 공급도 감소함

따라서 일반적으로 우상향하는 공급곡선이 나타남.

언제? 다른 모든 것들이 동일할 때

공급곡선의 이동

1. 생산가격 Input price
2. Technology
3. Number of sellers

1. Input price

생산단가가 감소하면 공급은 증가함. 공급곡선 우측 이동

생산단가 증가 - 공급량 감소 - 곡선 좌측 이동

2. Technology

기술수준이 상승하면 생산단가가 감소함. 공급곡선 우측이동.

Microanalysis of financial economics Assignment1

20249132 Kim Hyeonghwan (김형환)

Sample Question

sample question:

- Suppose that due to more stringent environmental regulation it becomes more expensive for steel production firms to operate. Also, recent technological advances in plastics has reduced the demand for steel products.

Q1) Use Supply and Demand analysis to predict how these shocks will affect equilibrium price and quantity of steel.

Q2) Can we say with certainty that the market price for steel will fall? Why?

Answer

Becomes more expensive for steel production

-> Increase input prices for steel production

-> Supply curve shifts to the left.

Reduced the demand for steel products

-> Decrease number of buyers for steel production

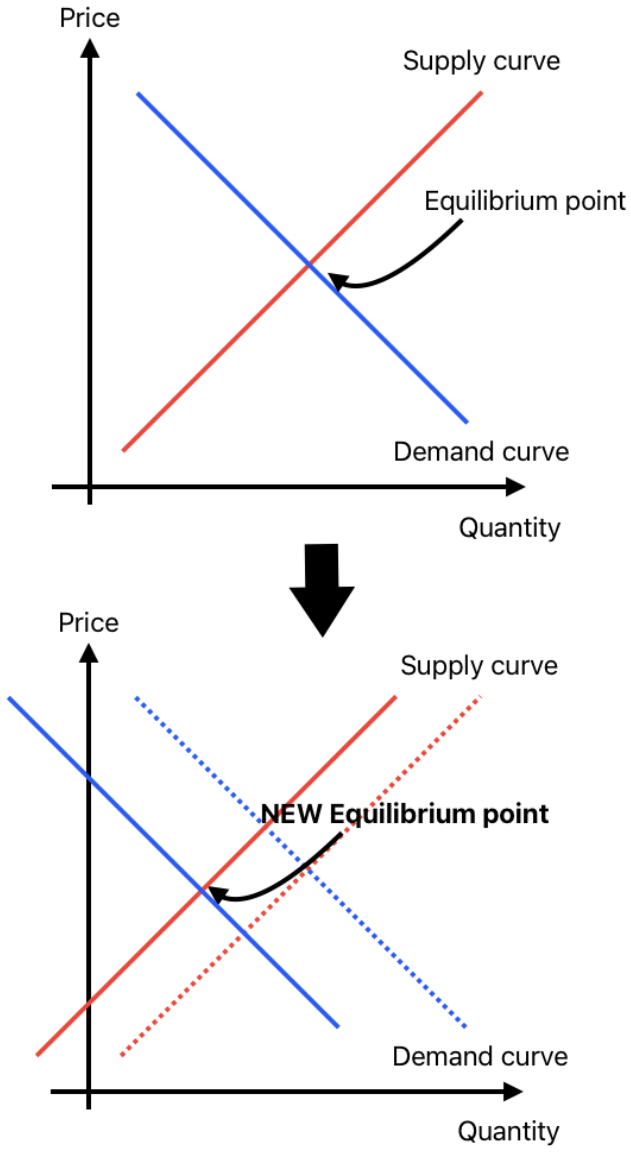
-> Demand curve shifts to the left.

Both curves will shift to the left side,

so NEW equilibrium point also shift to the left side.

It is clear that NEW equilibrium quantity will increase, but price is not.

Whether equilibrium price increase or not, it depends on how each curves shifts and their elasticity.



1. More expansive for steel production
-> Left shift of the supply curve

2. Reduced the demand for steel production
-> Left shift of the demand curve

Therefore, both curves shift to the left side,

Also equilibrium point shifts to the left.

It is obvious that **quantities of equilibrium is decr**

But **price of equilibrium can be decrease or incre**

It depends on how much each curve shifts.

Figure 1: Sample question

Assignments 1

Assignments: Due 9/10 (Tuesday)

- **Examine the behavior of supply and demand for Wheat**

Demand: $Q_D = 3550 - 266P$ *Supply*: $Q_S = 1800 + 240P$

the market-clearing price of wheat:

$Q_S = Q_D$

$1800 + 240P = 3550 - 66P$

$506P = 1750$

P = \$3.46 per bushel

Substituting into the supply curve equation, we get

$Q = 1800 + (240)(3.46) = 2630$ million bushels

==========

•What is the price E of Demand?

•What is the price E of Supply?

Answer

Let $P_0 = 3.46$ and $Q_0^D = Q_0^S = 2630$,

Set $\frac{\Delta P_{(=P_1 - P_0)}}{P_0} = 0.01$, so $P_1 = 3.46 + 0.0346 = 3.4946$

Then, $Q_1^D = 3550 - 266P_1 = 2620.4364$ and $Q_1^S = 1800 + 240P_1 = 2638.704$

So, $\frac{\Delta Q^D}{Q_0^D} = \frac{-9.5636}{2630} \approx -0.36\%$ and $\frac{\Delta Q^S}{Q_0^S} = \frac{8.704}{2630} \approx 0.33\%$

Finally, Price elasticity of Demand is $\frac{-0.36\%}{1\%} \approx -0.3636$.

Price elasticity of Supply is $\frac{0.33\%}{1\%} \approx 0.3310$

Part VIII

글로벌 지속가능회계('24 가을)

글로벌 지속가능회계 1주차