Final project type I: convex optimization practice

23-1 ESC 1조 김민주, 임승현, 윤이경

1. Implementation of function

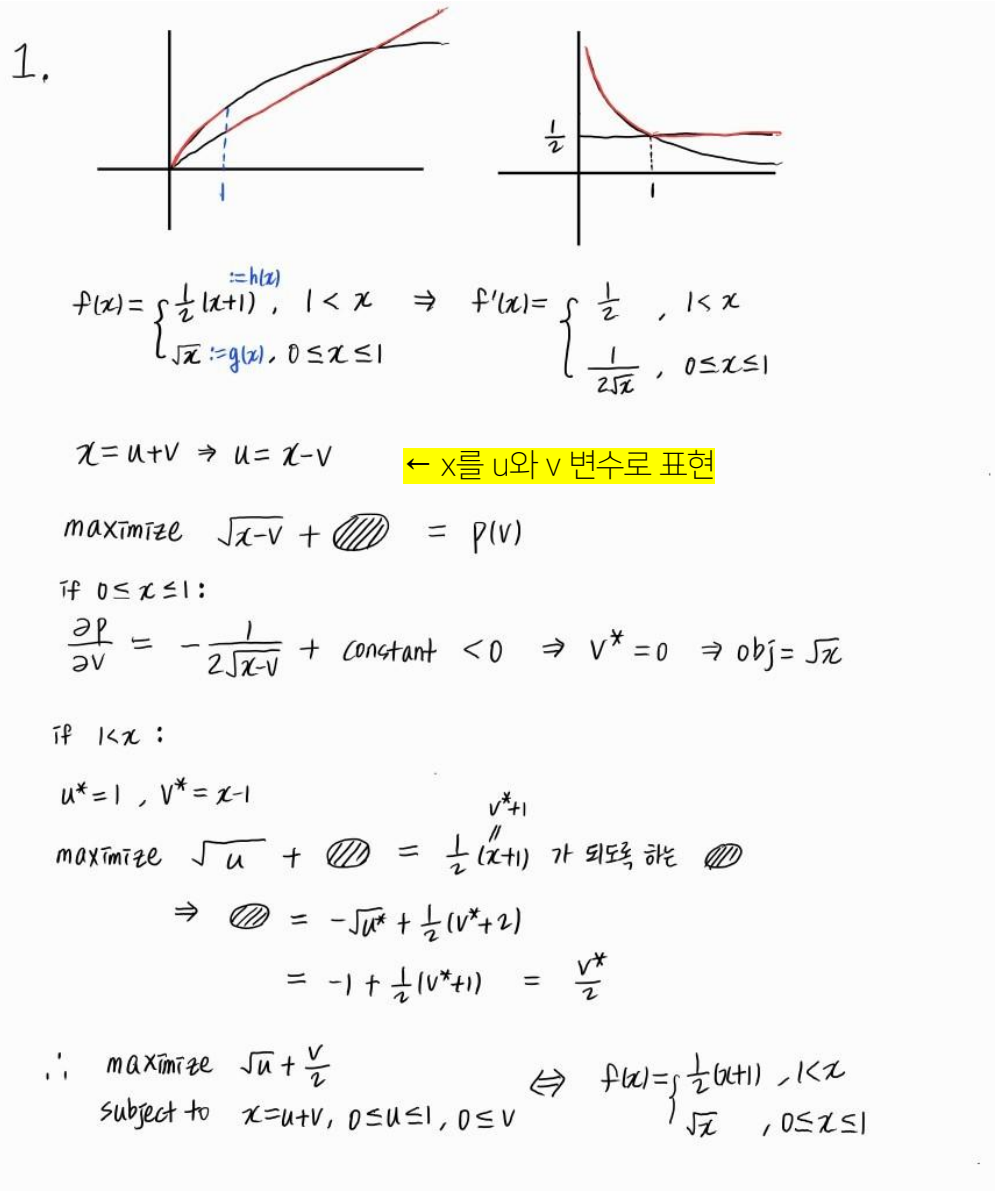
1. *Implementation of function*

$$f(x) = \begin{cases} \dfrac{(x+1)}{2}, & x > 1 \\ \sqrt{x}, & 0 \le x \le 1 \end{cases}$$

[주어진 조건]

(1) $\mathbf{f(x)}$: Concave Function
(2) $f: \mathbb{R} \to \mathbb{R},\ domf = \mathbb{R}+$

(3) CVX 패키지의 optimization problem 이용
(4) 힌트 _ 변수 추가



1. Implementation of function

$$f(x) = \begin{cases} \dfrac{(x+1)}{2}, & x > 1 \\ \sqrt{x}, & 0 \le x \le 1 \end{cases}$$

[주어진 조건]

(1) $\mathbf{f(x)}$: Concave Function
(2) $f: \mathbb{R} \to \mathbb{R},\ domf = \mathbb{R}+$

(3) CVX 패키지의 optimization problem 이용
(4) 힌트 _ 변수 추가

```python
import cvxpy as cp

# Variables
u = cp.Variable()
v = cp.Variable()
x = input()

# Constraints
constraints = [
    u + v == x,
    0 <= u,
    u <= 1,
    v >= 1,
]

# Objective functions
f_u = cp.sqrt(u)
f_v = v / 2

# Maximize both functions
objective = cp.Maximize(f_u + f_v)

# Define and solve problem
prob = cp.Problem(objective, constraints)
print(prob.solve())
```

← x를 u와 v 변수로 표현

```
10
5.499999999512643
```

2. Censored linear regression

2. Censored linear regression

K개의 일반적인 데이터 셋 (X, Y)

이 중에서 M개의 데이터 셋은 (X,Y) 모두 주어진 데이터 셋 → Uncensored Data!
그러나, 나머지 (K-M)개의 데이터에는 Y값이 없는 데이터 셋 → Censored Data!

EX) 환자 집단의 생존 분석에서 y를 사망 연령이라고 할 때, 아직 사망하지 않은 사람들의 데이터는 Censored Data가 됨.

→ 대신 우리가 가진 정보는 Y 데이터의 Lower Bound
→ 2번 문제는 결국 Censored Data에 대해서 Least Squares Problem을 풀어 보자!

$$J(\beta)=\sum_{i=1}^{K}(y_i-x_i^T\beta)^2\ ,\ \beta\in\mathbb{R}, y_i\in\mathbb{R}$$

2. Censored linear regression

```
import numpy as np
import pandas as pd
import cvxpy as cp
p = 20 # x feature 수
M = 25 # uncensored data의 개수
K = 100 # 총 데이터 개수
β_true = np.array([[-0.31232848],
                   [ 0.33928471],
                   [-0.15590853],
                   [-0.50178967],
                   [ 0.23556889],
                   [-1.76360526],
                   [-1.09586204],
                   [-1.08776574],
                   [-0.30517005],
                   [-0.47374837],
                   [-0.20059454],
                   [ 0.35519677],
                   [ 0.68951772],
                   [ 0.41058968],
                   [-0.56497844],
                   [ 0.59939069],
                   [-0.16293631],
                   [ 1.6002145 ],
                   [ 0.6816272 ],
                   [ 0.0148801 ]])
D = -1.85849413 #lower bound
```

	y	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
0	-11.201894	0.605992	-2.454759	0.045563	0.176614	-0.298426	1.804337	2.912316	1.060903	0.607922	...	-1.077106	0.632956	0.208577	-0.815701	0.400402	-0.488416	-1.410603	-1.033419	-0.918270	1.122871
1	-8.142421	1.978272	0.335854	-1.655557	0.825853	1.207102	0.800215	1.186441	0.857392	0.288278	...	1.007526	0.274830	-1.406585	-0.995646	1.394034	-0.074804	1.562823	-0.715595	0.283256	-2.310862
2	-7.403662	0.083425	-1.546519	-0.778424	-0.451652	0.046494	0.966535	-0.635014	1.597609	0.498389	...	-0.351244	0.076550	-0.628422	-0.475858	-1.470566	-0.709172	0.932744	-1.081364	-1.959578	0.153491
3	-6.413732	-0.250321	-0.052321	0.523791	0.727909	-0.096599	1.635037	0.282871	-0.937371	0.522907	...	-0.482903	0.498134	-0.847970	2.081939	1.227125	-1.407075	0.752502	-0.553323	-0.847878	2.355897
4	-6.186743	-0.178037	0.880932	-0.586190	-0.233675	-1.477150	1.237730	1.022461	1.607310	0.300730	...	0.782327	-0.667418	-1.702026	-1.427234	0.287688	-0.983605	-0.076180	0.365278	-1.042955	-1.758906
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	0.000000	-0.826695	-0.789142	0.080355	-1.425464	-0.860742	-1.558056	1.020550	-1.632681	0.605547	...	-0.132517	-0.436779	0.067746	-0.275510	-0.630586	-0.815057	-0.966263	1.572824	-1.331925	-1.195894
96	0.000000	1.669023	1.171124	-0.015557	-0.924814	-0.164354	-1.087456	-0.580078	-0.354280	-1.855985	...	1.320371	0.050361	1.505892	0.659734	0.553889	-0.004163	-0.862689	0.830780	1.414414	0.554244
97	0.000000	0.168937	0.761347	0.756033	0.172604	0.707614	-2.025401	-0.711304	-0.544211	-0.547473	...	-0.426870	-0.162099	-0.411064	-0.170058	-0.024208	0.942211	0.123983	0.507248	0.442128	-1.344965
98	0.000000	1.548128	-0.151305	0.803737	-1.098207	-0.360687	-1.308025	-1.378180	-0.666023	0.506453	...	0.048728	1.038008	0.120236	1.484881	0.609249	-0.917663	-0.331270	2.386918	0.149070	-1.218094
99	0.000000	-0.614257	-0.623052	0.759762	0.040780	0.522877	-1.672332	-0.478030	-0.814404	1.198434	...	0.933796	0.476497	1.726208	-1.056191	-0.933670	0.254366	-0.585158	1.133261	1.097397	0.533182

100 rows × 21 columns

(1) 제공된 데이터 셋 q2

→ 총 X 변수 개수 : 20개
→ 총 데이터 개수 : 100개

→ 앞에서 25개만 Y값 존재 (뒤의 75개는 없음)
→ lower bound와 B_true 값 주어짐

2. Censored linear regression

문제) True β에 대한 relative error와 Censored data를 무시하고 fitting 한 경우 비교하자!

```
x_uncensored = x[:M, :]

# Variable
β = cp.Variable(shape = p)

# Objecive and Constraints
objective = cp.Minimize(cp.sum_squares(x @ β - y))
constraints = [x[M:,:] @ β >= D]   ← censored data에 대해서 lower bound 적용

# Define and solve problem
prob = cp.Problem(objective, constraints)
result = prob.solve()

β_cvx = np.array(β.value).flatten()
β_cvx
```

```
array([-0.3407799 ,  0.34988757, -0.36899209, -0.13934053,  0.00663958,
       -0.72087572, -0.91928032, -0.8136302 , -0.11148206, -0.37230353,
       -0.04908198, -0.15773714,  0.45087434,  0.22994262, -0.33951828,
        0.43688188,  0.05152723,  0.56884203,  0.49509364,  0.13989614])
```

```
np.sum((x @ β_true - y)**2)
```

```
189995.5167338592
```

```
np.sum((x @ β_cvx - y)**2)
```

```
281.6618771875786
```

```
np.sum((x @ β_ols - y)**2)
```

```
845.5069622849991
```

3. Portfolio optimization

3. Portfolio optimization

(문제 상황) assert 1, 2, 3, 4가 있을 때

(1) 수익률(return의 기댓값)을 최대화 하면서,
(2) 동시에 risk(return의 분산)을 최소화 하는 assert의 최적 구매 비율 w를 찾는 것

→ Maximize $\bar{p}^{\top} w$

→ Minimize $w^{\top} \Sigma w$

(주어진 정보)
- $\bar{p}$: assert의 가격 변화율
- Σ: assert의 가격 변화율의 분산

$$\text{가격 변화율} = \frac{(\text{만기 가격} - \text{현재 가격})}{\text{현재 가격}}$$

$$\bar{p} = \begin{bmatrix} 0.12 \\ 0.10 \\ 0.07 \\ 0.03 \end{bmatrix}, \Sigma = \begin{bmatrix} 0.0064 & 0.0008 & -0.0011 & 0 \\ 0.0008 & 0.0025 & 0 & 0 \\ -0.0011 & 0 & 0.0004 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

가격 변화율이 가장 높은 assert1은 파생 상품이라 보면
가장 작은 assert은 이자가 3%로 고정된 은행 예금이라 볼 수 있다.

3. Portfolio optimization (Q. 3-1)

Optimization problem으로 표현하면 다음과 같다.

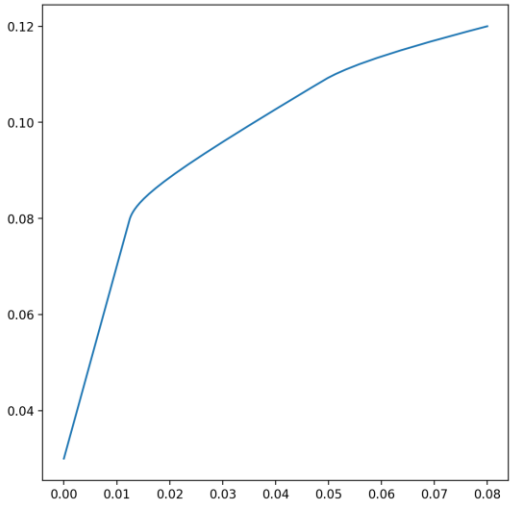
$$\text{minimize } -\bar{p}^{\top}w+\lambda w^{\top}\Sigma w$$
$$\text{subject to } 1^{\top}w=1, w\geq 0$$

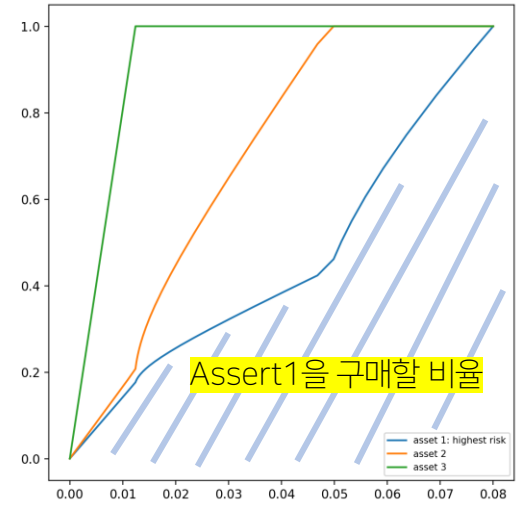
```
for i in range(100):

    objective = cp.Minimize(-p @ w + ₩λ[i] * cp.quad_form(w, Σ))
    constraints = [cp.sum(w) == 1,
                   w >= 0]

    prob = cp.Problem(objective, constraints)
    result = prob.solve()

    w_opt[i, :] = w.value ₩
    expect_return[i] = p @ w.value
    optimal_std[i] = np.sqrt(w.value @ Σ @ w.value)
```





Optimal standard deviation과,
Optimal expected return과 w의 area plot

→ (왼) 높은 리스크일 수록 기대 수익률도 높아짐
→ (오) 그리고 assert 1의 비중도 높아짐

리스크가 0인 경우 assert4의 구성이 100% 이고,
반대로 리스크가 0.08까지 커지면 assert1의 구성이 100%

3. Portfolio optimization (Q. 3-2)

(추가 조건)

(1) P : 평균 - $\bar{p}$ & 분산 $\boldsymbol{\Sigma}$ gaussian r.v.

(2) 표준 정규 분포의 CDF, inverse CDF 이용

$$\text{minimize} \; -\bar{p}^{\top} w$$
$$\text{subject to } \mathbf{prob}(p^{\top} w \leq 0) \leq \eta$$
$$1^{\top} w = 1,\; w \geq 0$$

→ 손해를 볼 확률을
일정 수치 이하로 조절하자!

```
from scipy.special import erfcinv

η = np.logspace(-4, -1, 100)
w = cp.Variable(shape = 4)

vals, vectors = np.linalg.eig(Σ)
sqrt_Σ = vectors.T @ np.sqrt(np.diag(vals)) @ vectors

w_opt = np.zeros((100, 4))
expect_return = np.zeros(100)
optimal_std = np.zeros(100)


for i in range(100):

    γ = - np.sqrt(2) * erfcinv(2 * η[i])
    objective = cp.Maximize(p @ w)
    constraints = [cp.sum(w) == 1,
                   w >= 0,
                   p @ w + γ * cp.norm(w @ sqrt_Σ) >= 0]
    prob = cp.Problem(objective, constraints)
    result = prob.solve()

    w_opt[i, :] = w.value
    expect_return[i] = p @ w.value
    optimal_std[i] = np.sqrt(w.value @ Σ @ w.value)
```

3. Portfolio optimization (Q. 3-2)

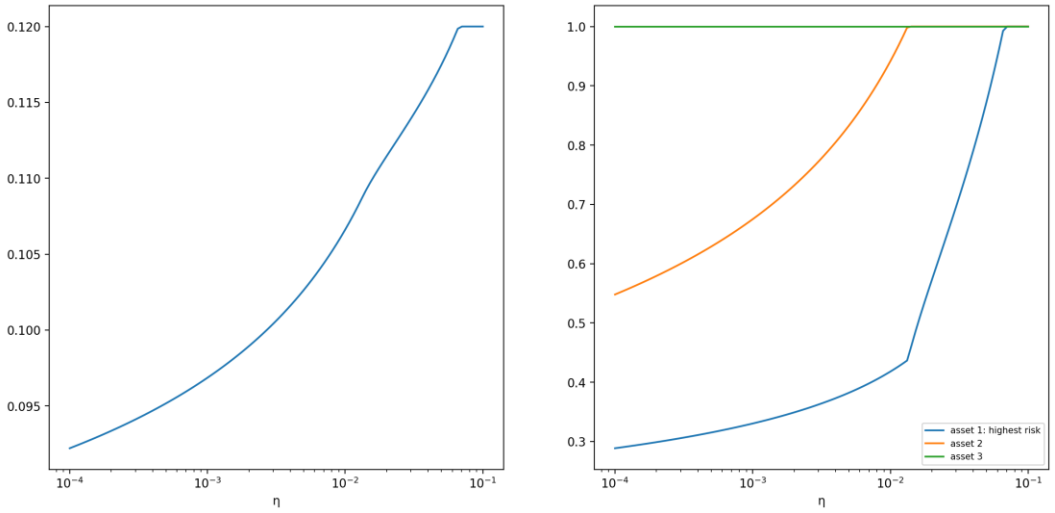
(추가 조건)

(1) P : 평균 - $\bar{p}$ & 분산 Σ gaussian r.v.

(2) 표준 정규 분포의 CDF, inverse CDF 이용

$$\text{minimize} \; -\bar{p}^{\mathsf{T}} w$$
$$\text{subject to } \mathbf{prob}(p^{\mathsf{T}} w \le 0) \le \eta$$
$$1^{\mathsf{T}} w = 1, \; w \ge 0$$

→ 손해를 볼 확률을
일정 수치 이하로 조절하자!



→ (왼) 허용 가능한 손해 확률을 늘릴 수록 수익률은 높아짐
→ (오) 허용 가능한 손해 확률을 높일 수록 assert1 비중 높아짐

손해 확률이 0이 아니므로 어느정도 리스크는 감수한다는 의미
때문에 손해 확률이 없는 assert4는 구매할 이유가 없다.

→ 낮은 손해 확률의 경우에는 assert3의 비중이 높다.

3. Portfolio optimization (Q. 3-3)

(문제) η를 0.05로 설정하고, 3-2의 문제를 10000번 시뮬레이션 해보세요.

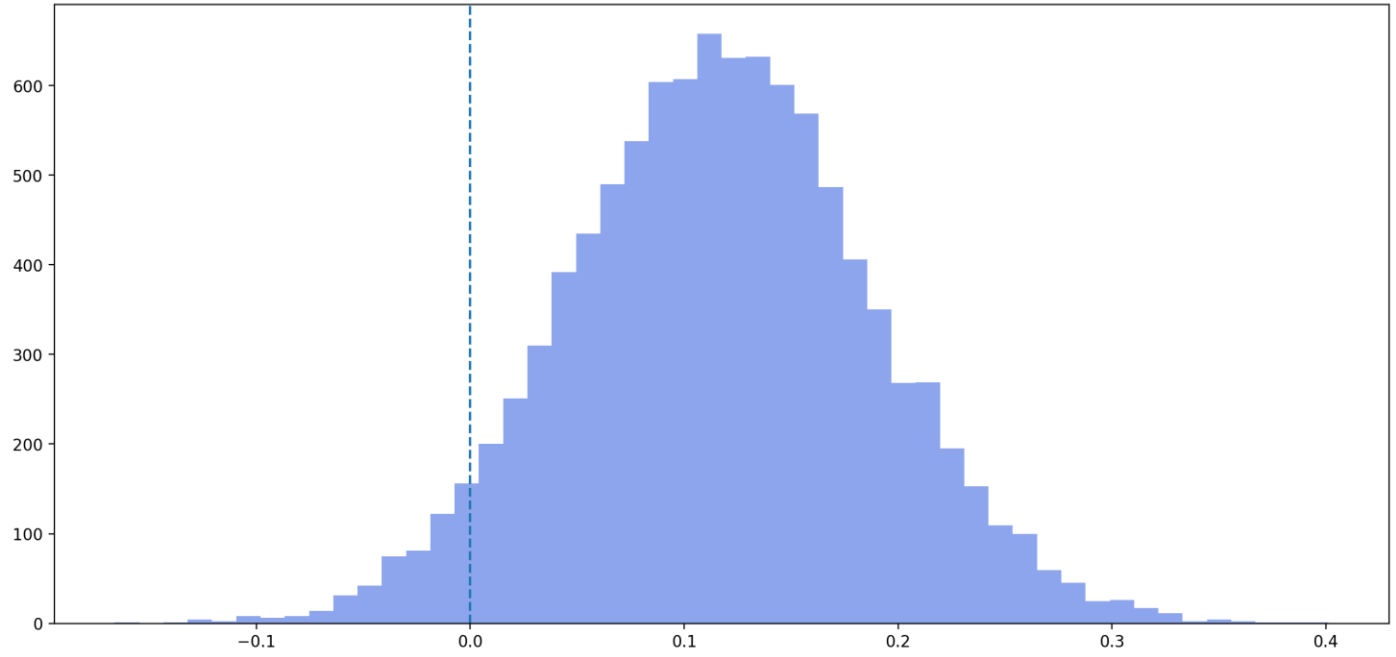
→ 평균적으로 12%의 수익을 얻고, 손실을 볼 경우는 10000번 중 498번이다.

```
returns.mean()
```

```
0.1170539859181814
```

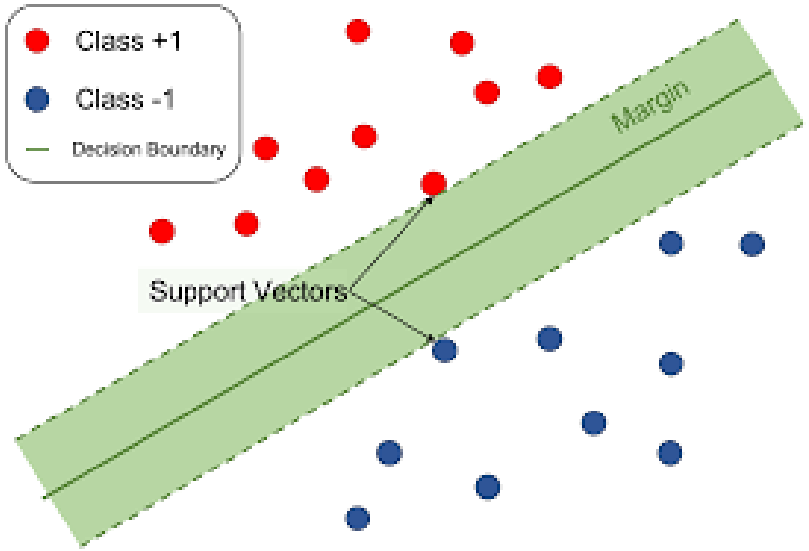
```
np.sum(returns < 0)
```

```
498
```



4. Multiclass SVM

4. Multiclass SVM (Q. 4-1)



(문제) 목적 함수가 다음과 같이 나타나는 이유는?

1) 주어진 데이터 셋 $(x_i, y_i) \in R_n$

2) Classifier affine function

$$f_k(x) = w_k^{\top} x + b_k,$$
$$= Wx + b, \quad \mathbb{R}^n \rightarrow \mathbb{R}^k$$

3) 추정 값 $\hat{y} = \arg\max_k \ f_k(x)$

Hinge loss function :

$$L(W, b) = \sum_{i=1}^{m} (1 + \max_{k \neq y_i} f_k(x_i) - f_{y_i}(x_i))_+$$

(u_+)는 $\max(0, u)$를 나타냅니다

Objective function :

$$L(W,b) + \lambda \|W\| F$$

4. Multiclass SVM (Q. 4-2)

```
from tqdm import tqdm

λ = np.logspace(-2, 2, 30)
errorTrain = []
errorTest = []


for i in tqdm(λ):
    W = cp.Variable((K, n))
    b = cp.Variable((K, 1))
    f = W @ x + b

    L = 0
    for k in range(K):
        ind = np.delete(np.arange(K), k)
        L += cp.sum(cp.pos(1 + cp.max(f[ind][:, y==k], axis=0) - f[k, y==k]))

    objective = cp.Minimize(L + i * cp.sum(cp.sum(W**2)))
    constraints = [cp.sum(b) == 0]
    problem = cp.Problem(objective, constraints)
    problem.solve(solver = cp.ECOS)

    indTrain = np.argmax(W.value @ x + b.value, axis=0)
    errorTrain.append(np.sum(indTrain != y) / mTrain)

    indTest = np.argmax(W.value @ xtest + b.value, axis=0)
    errorTest.append(np.sum(indTest != ytest) / mTest)
```

(문제) 데이터 q4 이용해서 SVM 구현하고,
parameter λ 값에 따른 test loss를 나타내 보세요.