

# Introduction

---

2023.01

Engineering Development Research Center (EDRC)

정 동 휘

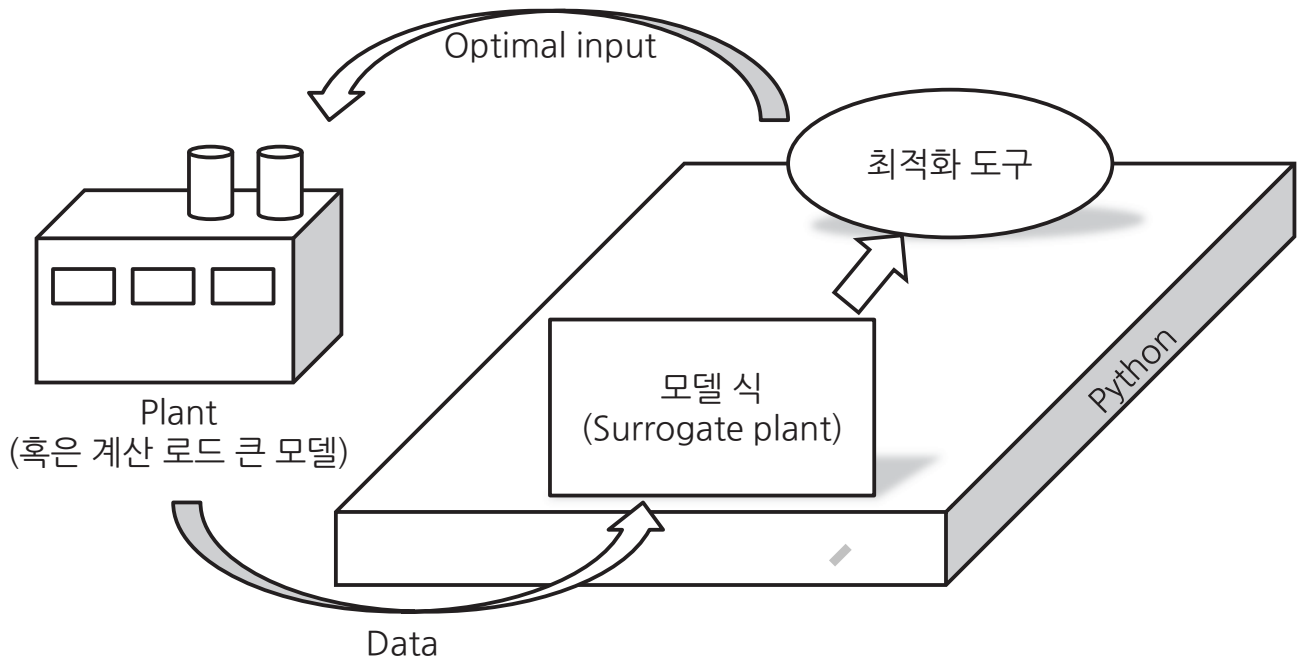
---

Introduction

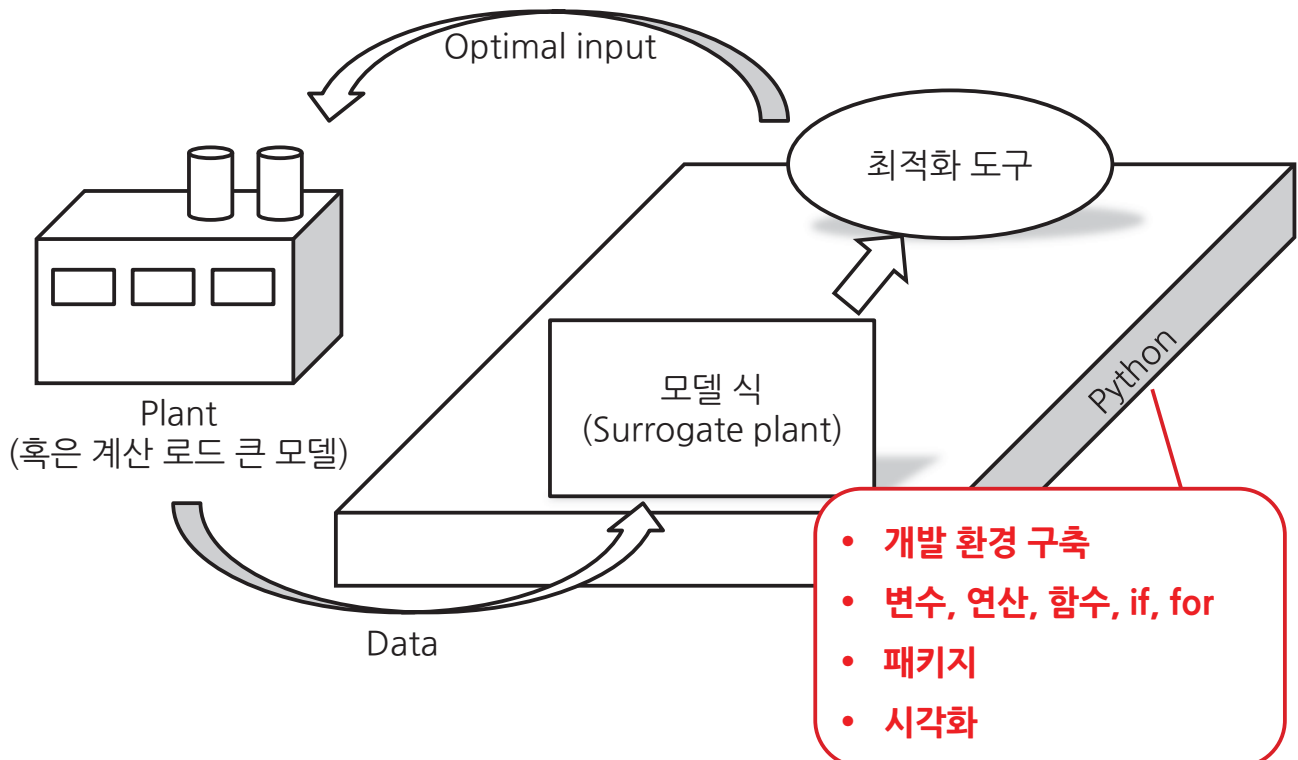
## 수업 개요 및 방식

1. 매 수업 시간을 통해 무엇을 얻어야 하는가?
2. 수업 방식 (숲 ↔ 나무) (쉬운 → 어려운) (반복)
3. 개념 학습
4. 예시 실습
5. 정리

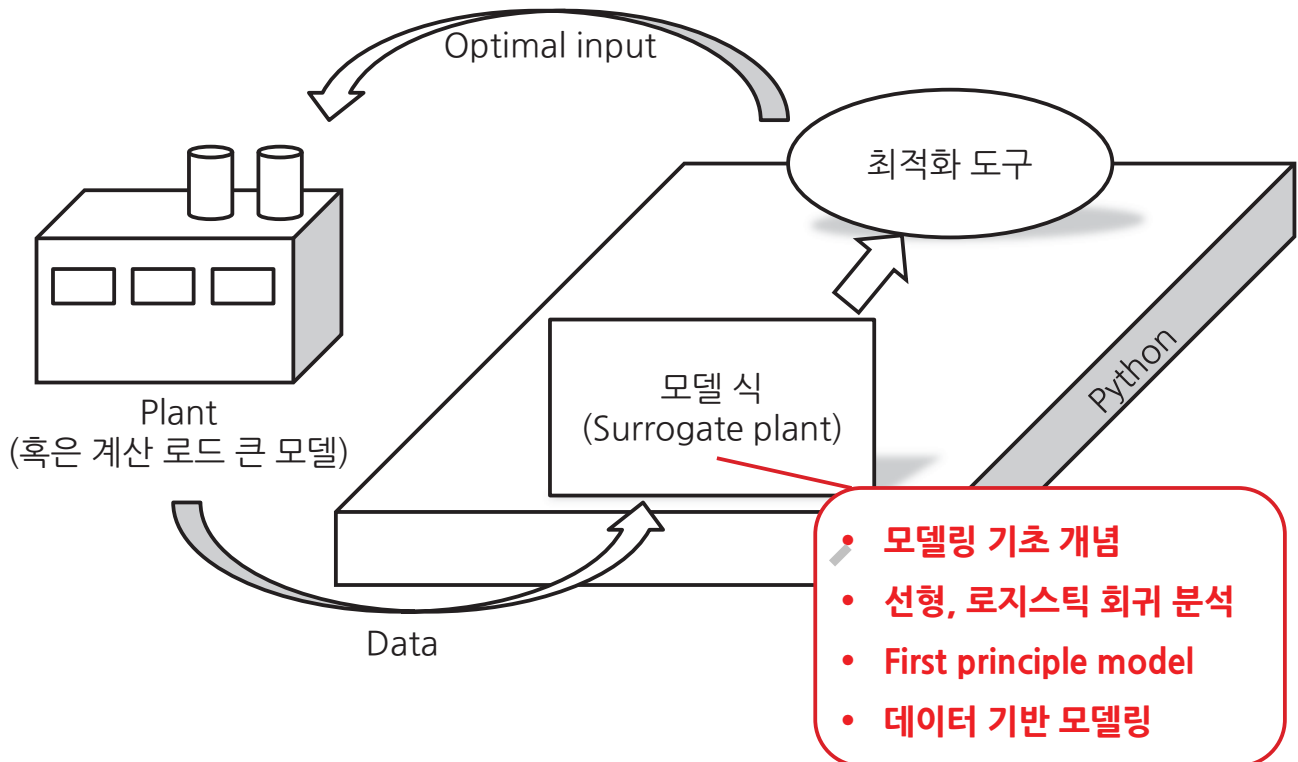
# 데이터, 모델링, 최적화, 파이썬



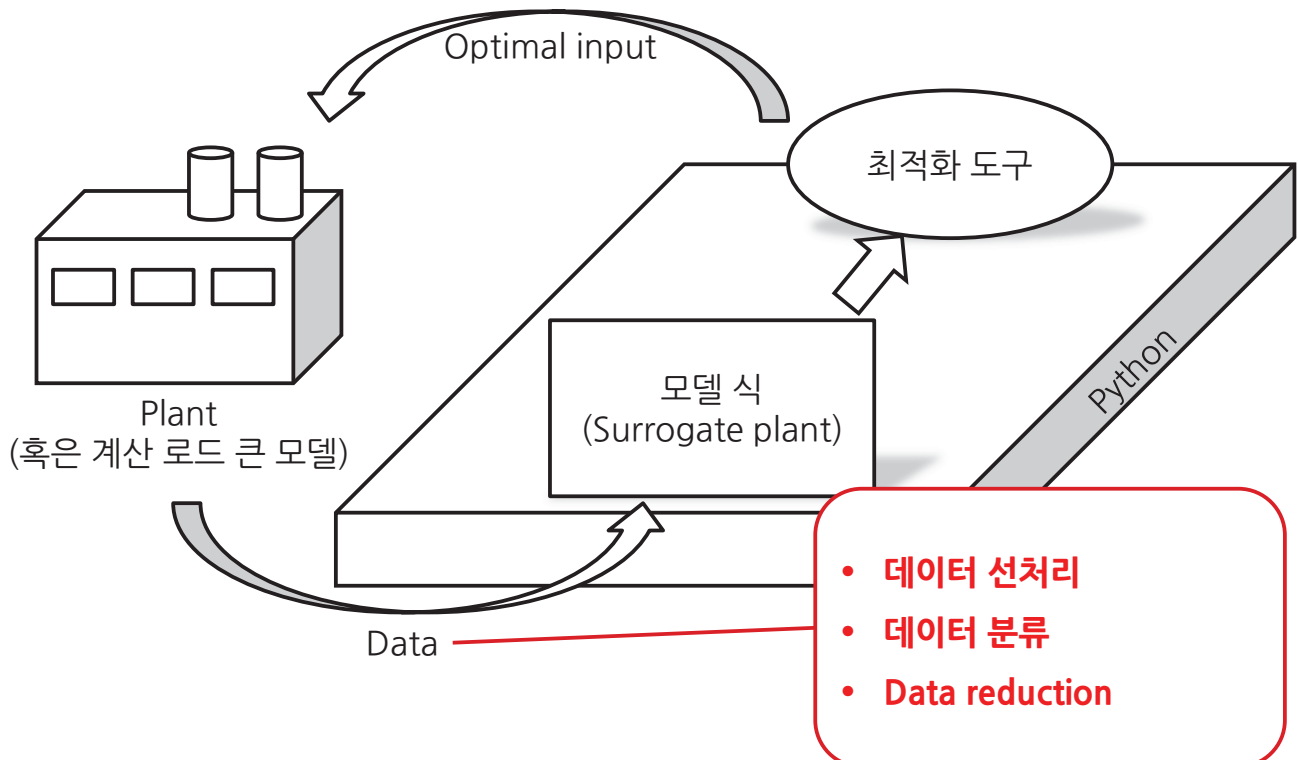
# 데이터, 모델링, 최적화, 파이썬



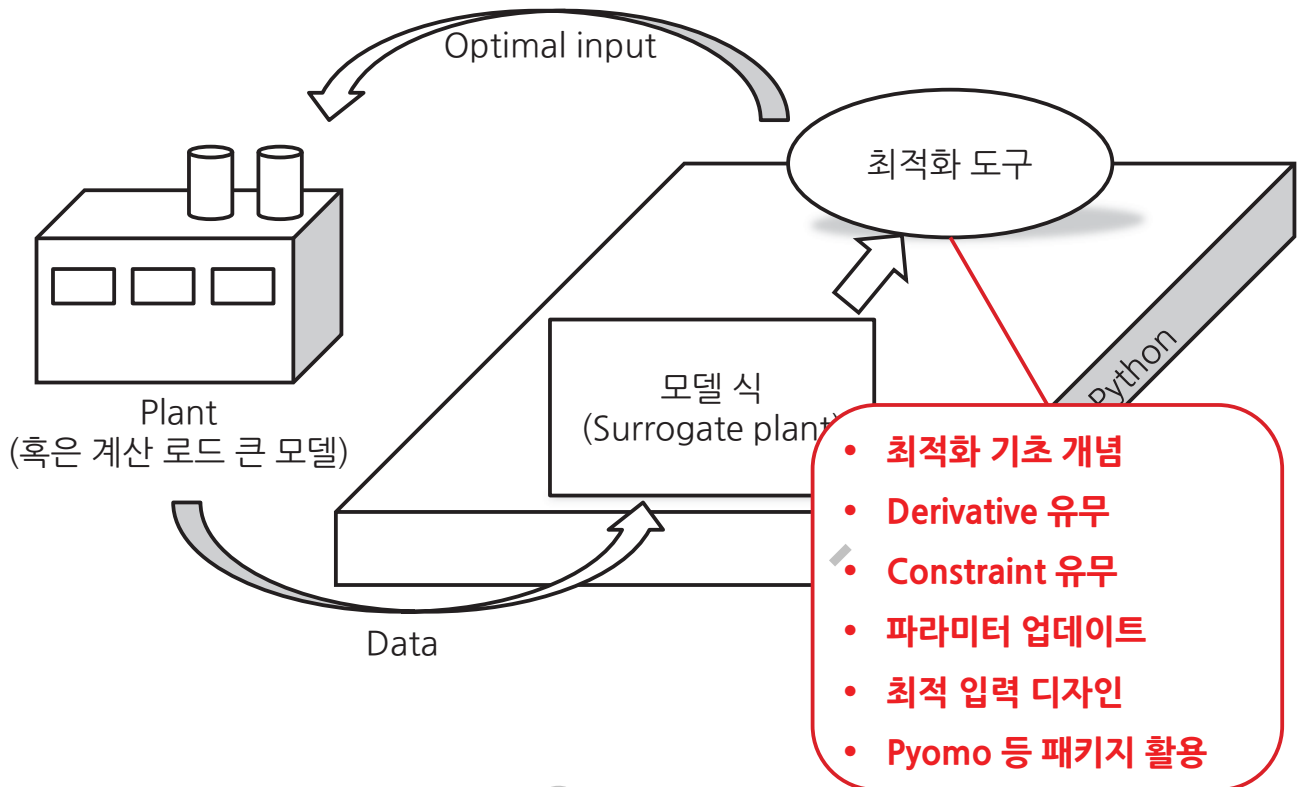
# 데이터, 모델링, 최적화, 파이썬



# 데이터, 모델링, 최적화, 파이썬



# 데이터, 모델링, 최적화, 파이썬



## 파이썬 기초: 개발 환경 구축

2023.01

Engineering Development Research Center (EDRC)

정동휘

## 개요

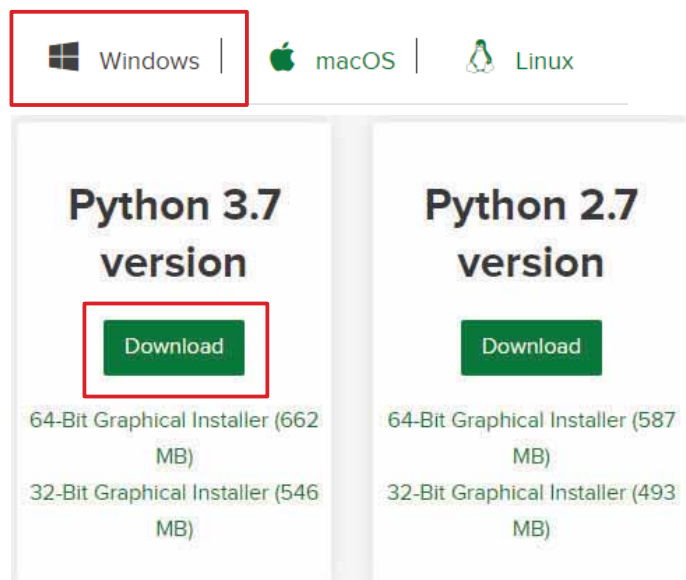
1. 아나콘다 설치
2. 명령 프롬프트 활용
3. 주피터 노트북
4. 파이참 설치
5. 가상 환경 구축
6. 완료 확인

COSMETIC

## 아나콘다 설치

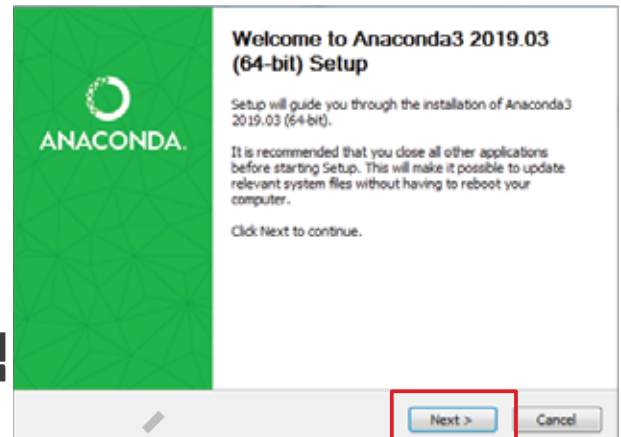
- <https://www.anaconda.com/distribution/#download-section>

- Windows 클릭
- 다운로드 클릭



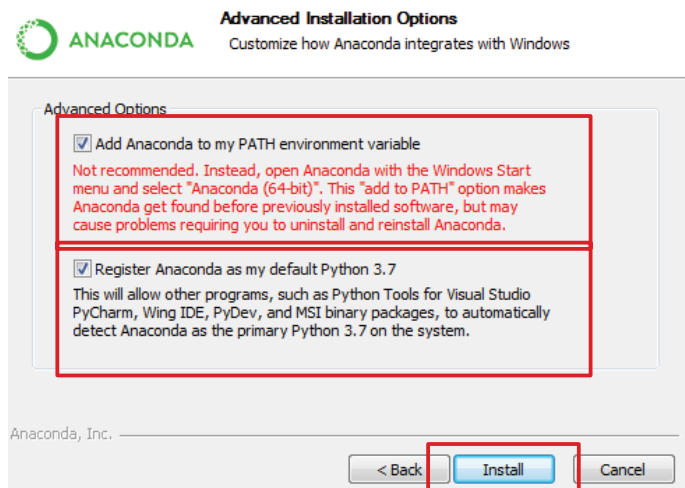
## 아나콘다 설치

- 다운로드 된 아이콘 클릭  Anaconda3-2019.03-Windows-x86\_64
- Next 클릭
- I Agree 클릭
- 'Just Me' 선택 후 Next 클릭
- 설치 위치 설정 후 Next 클릭



## 아나콘다 설치

- 체크 박스 클릭 후 Install 클릭
- Next-Next-Finish 클릭
- Done!



## 명령 프롬프트 활용

- 시작 버튼 (혹은 윈도우 키)
- 모든 프로그램
- Anaconda3 (64-bit)
- Anaconda Prompt 클릭
- 검은 화면에 python 입력

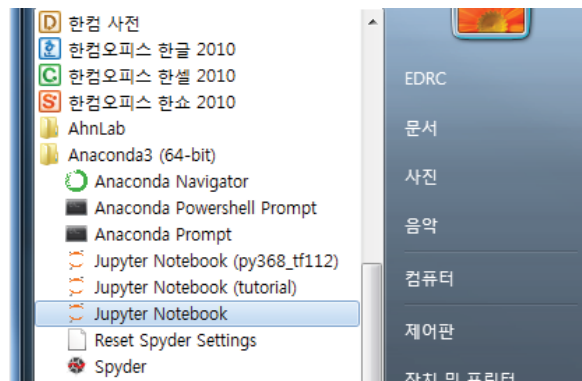
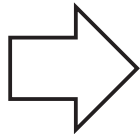
```
(base) C:\Users\probook4530s>python
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

## 명령 프롬프트 활용

- 검은 화면에 python 입력 후
- 1+1 입력
- a=1 입력
- b=2 입력
- a+b 입력
- Print("Hello, world!") 입력

## 주피터 노트북

- 시작 버튼
- 모든 프로그램
- 아나콘다3 파일



## 가상 환경 구축

- 아나콘다 명령 프롬프트
- 입력: activate 가상환경이름
- Y 입력

```
<base> C:\Users\probook4530s>python -m pip install --upgrade pip
```

```
<base> C:\Users\probook4530s>conda create -n EDRC python=3.6
```

```
Proceed <[y]/n>? y
```

- 설치 완료 확인



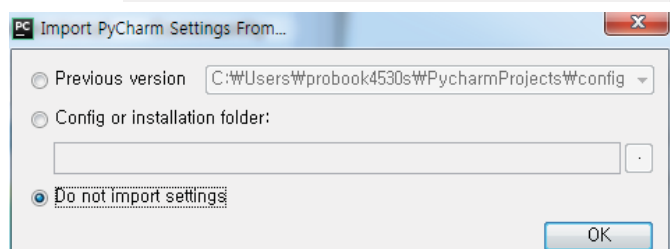
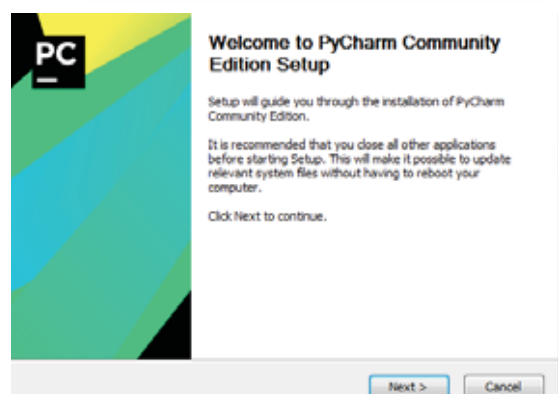
## 파이참 (Pycharm) 설치

- Integrated Development Environment (IDE)
- 좀 더 효율적인 개발을 위하여
- <https://www.jetbrains.com/pycharm/>
- 다운로드 클릭



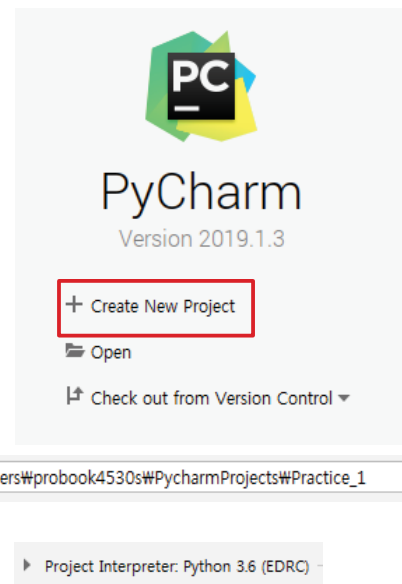
## 파이참 (Pycharm) 설치

- Community 버전 다운로드 클릭
- 다운로드된 아이콘 클릭설치
- Next 클릭
- Shortcut 체크 후 Next 클릭
- Install 클릭
- Do not import settings



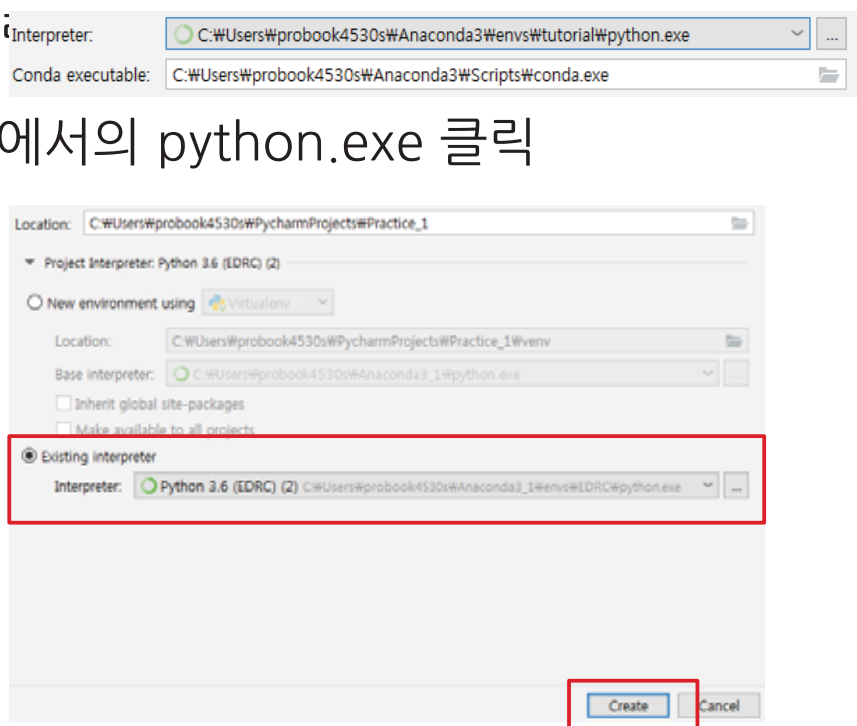
## 파이참 (Pycharm) 설치

- 세팅 후 'Create New Project' 클릭
- Project 이름 설정
- Project Interpreter 클릭
- Existing interpreter 클릭
- (...) 클릭
- 왼쪽에 Conda Environment 클릭



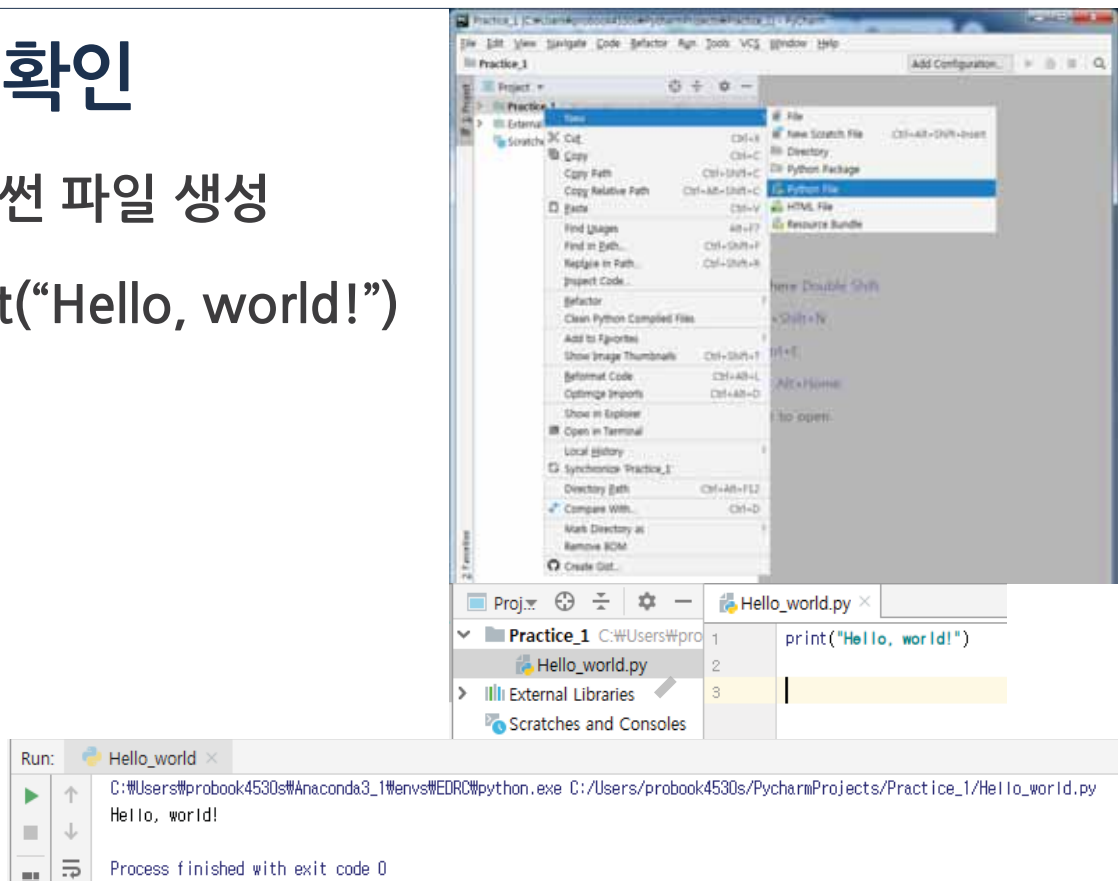
## 파이참 (Pycharm) 설치

- 오른쪽 위 (...) 클릭
- 생성한 가상 환경에서의 python.exe 클릭
- Create 클릭



## 완료 확인

- 파이썬 파일 생성
- `print("Hello, world!")`



## 파이썬 기초: 연산, 변수, 함수, if, for

2023.01

Engineering Development Research Center (EDRC)

정동휘

# 개요

1. 기본
2. 연산
3. 변수
4. 함수
5. If 구문
6. For 구문

COSMETIC

# 기본

#기본 입력과 출력

- Print()로 출력하기

```
print("Hello, Python!!!!")
print(2)
print('2')
print('a')
print('a', 2)
```

- 실행 단축키: ctrl+shift+f10

- # (혹은 ctrl+/)으로 activate/deactivate 시키기

## 연산

### • 산술

```
#산술
print("1+1의 결과는: %n", 1+1) #더하기
print("1+1의 결과는: %n", 3*2) #곱하기
print("1+1의 결과는: %n", 5/2) #나누기
print("1+1의 결과는: %n", 10%3) #나누기의 나머지
print("1+1의 결과는: %n", 10//3) #나누기의 몫
print("1+1의 결과는: %n", 3**4) #제곱
```



1+1의 결과는:  
6  
1+1의 결과는:  
2.5  
1+1의 결과는:  
1  
1+1의 결과는:  
3  
1+1의 결과는:  
81

### • 비교

```
#비교
print("1==1의 결과는: %n", 1==1)
print("1==1의 결과는: %n", 1!=1)
print("1==1의 결과는: %n", 1==2)
print("1==1의 결과는: %n", 1!=2)
```



1==1의 결과는:  
True  
1==1의 결과는:  
False  
1==1의 결과는:  
False  
1==1의 결과는:  
True

## 변수

### • 변수 선언

```
#변수
```

```
#변수 선언
```

```
a = 10
```

```
b = 20
```

```
#화면에 결과 출력하기
```

```
print(a)
```

```
print("a")
```

```
print('a')
```

```
print(b)
```

```
print(a+b) #변수 연산
```



10

a

a

20

30

### • 변수 연산

# 함수

## • 내장 함수

#내장 함수

```
x=sum([3,8])
print("내장 함수 sum의 결과는: ", x)

y=round(7.2)
print("내장 함수 round의 결과는: ", y)

z=range(5)
print("내장 함수 range의 결과는: ", z)
print("range 결과 안에 들어있는 것은: ", list(z))

l=list([2,3])
print("내장 함수 list의 결과는: ", l)
```



내장 함수 sum의 결과는:  
11  
내장 함수 round의 결과는:  
7  
내장 함수 range의 결과는:  
range(0, 5)  
range 결과 안에 들어있는 것은:  
[0, 1, 2, 3, 4]  
내장 함수 list의 결과는:  
[2, 3]

# 함수

## • 사용자 지정 함수

#user-define 함수 (Module)

```
def add(a,b):
    result = a+b
    return result

print("User-define 함수의 결과는: \n", add(1,2))
```



User-define 함수의 결과는:  
3

## If 구문

- 조건문이 True/False 일 때 다르게 결과 생성

#if 조건문

```
a=True
if a:
    print("조건문이 True이다.")
else:
    print("조건문이 False이다.")

b=10
if b>5:
    print("조건문이 True이다.")
else:
    print("조건문이 False이다.")
```



조건문이 True이다.  
조건문이 True이다.

## If 구문

- If 구문 + User-defined 함수

#if 조건문 + 함수 (Module)

```
def example(x):
    if x>10:
        print("조건문이 True이다.")
    else:
        print("조건문이 False이다.")

example(5)
```



조건문이 False이다.

## For 구문

- For 구문 안 쓸 때 vs 쓸 때

#for 구문 안 쓸 때

```
print(1)
print(2)
print(3)
print(4)
print(5)
```



1  
2  
3  
4  
5

#for 구문 쓸 때

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



## For 구문

- 예제

```
a = [1, 3, 5, 7, 9]
for i in a:
    print(i)
```



1  
3  
5  
7  
9

```
b = ['Apple', 'Orange', 'Strawberry']
for i in b:
    print(i)
```




Apple  
Orange  
Strawberry



## For 구문

- For 구문 + User-defined 함수

```
#for 구문 + 함수  
  
def example2(input):  
    for i in range(input):  
        print(i)  
  
example2(4)
```



0  
1  
2  
3

## 파이썬 기초: 패키지, 라이브러리 활용

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

## 개요

1. 패키지, 라이브러리란?
2. 사용 이유
3. 패키지 설치 방법
4. Numpy
5. Scipy
6. Pandas

## 패키지, 라이브러리란?

- 앞서 배웠던 함수
- 유용한 함수 모아놓은 파일
- 오픈 소스이므로 쉽게 import 해서 사용할 수 있다.
- Matrix, Plotting, Data analysis, Optimization, Numerical calculations, ...

## 사용 이유

- 땅파기 난이도 비교: 맨손 vs 삽 vs 포크레인
- 목적에 따른 다양한 User-defined 함수를 제공
- 서로 연결되어 있는 경우도 많음

## 패키지 설치 방법

- 설치 하기 전

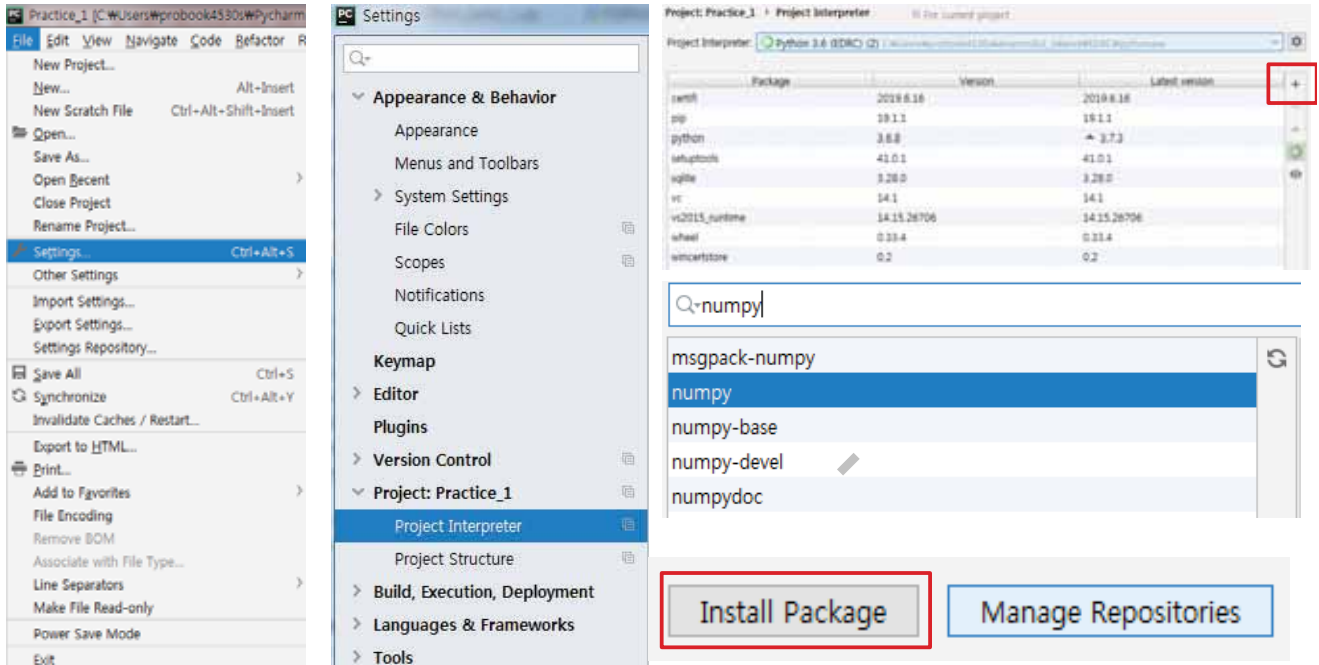
```
import numpy as np
```

```
C:\Users\probook4530s\Anaconda3_1\envs\EDRC\python.exe C:/Users/probook4530s/PycharmProjects/Practice_1/Python_packages_1.py
Traceback (most recent call last):
  File "C:/Users/probook4530s/PycharmProjects/Practice_1/Python_packages_1.py", line 3, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'

Process finished with exit code 1
```

# 패키지 설치 방법

## • 설치 방법 1



# 패키지 설치 방법

## • 설치 방법 2

```
<base> C:\#Users#probook4530s>activate EDRC
```

```
<EDRC> C:\#Users#probook4530s>pip install numpy
```

```
<EDRC> C:\#Users#probook4530s>conda install numpy
```

# Numpy

- List

```
import numpy as np

a = [2,4,6,7,10] #list
b = np.array(a) #numpy 배열 생성
print("list a는: \n", a)
print("numpy 배열 b는: \n", b)
print("numpy 배열 b의 차원은: \n", b.shape)
print("numpy 배열 b의 자료형은: \n", b.dtype)
```



list a는:  
[2, 4, 6, 7, 10]  
numpy 배열 b는:  
[ 2 4 6 7 10]  
numpy 배열 b의 차원은:  
(5,)  
numpy 배열 b의 자료형은:  
int32

# Numpy

- Row/Column matrix

```
a_ = [[2,4,6,7,10]] #row matrix
b_ = np.array(a_)
print("row matrix a_는: \n", a_)
print("numpy matrix b_는: \n", b_)
print("numpy 배열 b_의 차원은: \n", b_.shape)
print("numpy 배열 b_의 자료형은: \n", b_.dtype)
```



row matrix a\_는:  
[[2, 4, 6, 7, 10]]  
numpy matrix b\_는:  
[[ 2 4 6 7 10]]  
numpy 배열 b\_의 차원은:  
(1, 5)  
numpy 배열 b\_의 자료형은:  
int32

```
c=np.array([[1],[3],[5],[7],[9]]) #column matrix
print("numpy 배열 c는: \n", c)
print("numpy 배열 c의 차원은: \n", c.shape)
```



numpy 배열 c는:  
[[1]  
[3]  
[5]  
[7]  
[9]]  
numpy 배열 c의 차원은:  
(5, 1)

# Numpy

## • General matrix

```
d=np.array([[1,2,3],[4,5,6]]) #general matrix
print("numpy 배열 d는: %n", d)
print("numpy 배열 d의 차원은: %n", d.shape)
print("numpy 배열 d의 1행 2열 값은: %n", d[0,1])
print("numpy 배열 d의 2행 3열 값은: %n", d[1,2])
print("numpy 배열 d의 1열 모든 행의 값은: %n", d[:,0])
print("numpy 배열 d의 2행 모든 열의 값은: %n", d[1,:])
print("numpy 배열 d의 행끼리의 합은: %n", np.sum(d, axis=0))
print("numpy 배열 d의 열끼리의 합은: %n", np.sum(d, axis=1))
print("numpy 배열 d의 모든 요소의 합은: %n", np.sum(d))
```



```
numpy 배열 d는:
[[1 2 3]
 [4 5 6]]
numpy 배열 d의 차원은:
(2, 3)
numpy 배열 d의 1행 2열 값은:
2
numpy 배열 d의 2행 3열 값은:
6
numpy 배열 d의 1열 모든 행의 값은:
[1 4]
numpy 배열 d의 2행 모든 열의 값은:
[4 5 6]
numpy 배열 d의 행끼리의 합은:
[5 7 9]
numpy 배열 d의 열끼리의 합은:
[ 6 15]
numpy 배열 d의 모든 요소의 합은:
21
```

# Numpy

## • Matrix 연산 및 기타 함수

```
e = np.array([[1,3],[5,7]])
f = np.array([[2,4],[6,8]])
print("행렬 e와 f를 element끼리 곱하면: %n", e*f)
print("numpy 배열 e와 f의 행렬 곱 (product)는: %n", np.dot(e,f))
print("e와 f를 np.matmul을 사용하여 곱하면: %n", np.matmul(e,f))
```



```
행렬 e와 f를 element끼리 곱하면:
[[ 2 12]
 [30 56]]
numpy 배열 e와 f의 행렬 곱 (product)는:
[[20 28]
 [52 76]]
e와 f를 np.matmul을 사용하여 곱하면:
[[20 28]
 [52 76]]
```

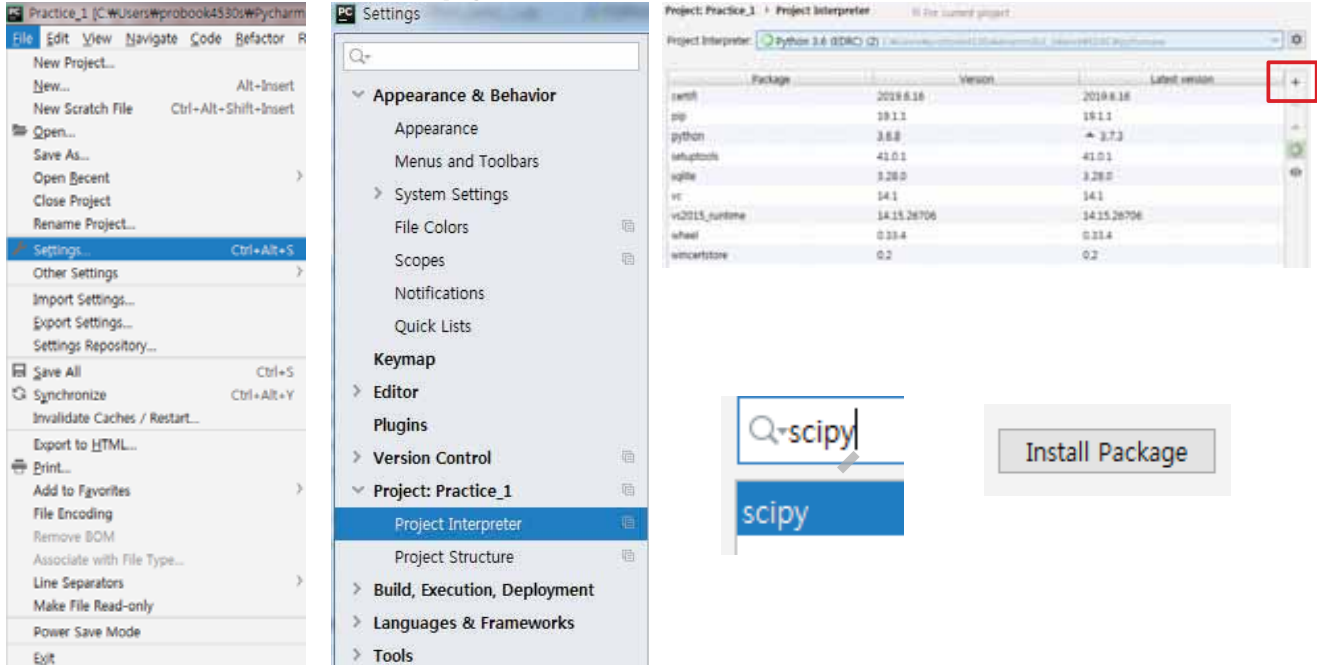
```
print(np.zeros(5))
print(np.ones(5))
print(np.linspace(1,3,num=5))
print(np.random.randn(2,3))
print(np.abs(-2))
print(np.sqrt(64))
print(np.square(3))
print(np.exp(2))
print(np.log(2))
print(np.log10(1000))
```



```
[0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1.]
[1. 1.5 2. 2.5 3. ]
[[0.48075322 1.99714059 0.25601408]
 [1.9716975 0.28215105 1.09652116]]
2
8.0
9
7.38905609893065
0.6931471805599453
3.0
```

# Scipy

## • 설치 방법: Numpy 설치와 동일



# Scipy

## • 적분

```
import scipy as sp
import scipy.integrate as integrate

#적분 할 함수
def fun(x):
    return x**2

result1 = integrate.quad(fun,0,1)
print("적분한 값은: %n", result1)
print("적분한 값 중 진짜는: %n", result1[0])
```



적분한 값은:  
(0.3333333333333337, 3.700743415417189e-15)  
적분한 값 중 진짜는:  
0.3333333333333337  
Scipy로 구한 솔루션은: [-0.45652174 -1.7826087 4.56521739]

# Scipy

## • 역행렬 구하기

#Ax=b의 솔루션

```
A=sp.array([[10,5,3.5],[5,0,0.5],[2,-1,2]])
b=sp.array([2.5,0,10])
x_sp = sp.linalg.solve(A,b)
A_eig_val, A_eig_vec = sp.linalg.eig(A)
print("Scipy로 구한 솔루션은: ", x_sp)
print("A의 역행렬은: \n", sp.linalg.inv(A))
print("A의 eigen value는: \n", A_eig_val)
print("A의 eigen vector는: \n", A_eig_vec)
```



Scipy로 구한 솔루션은: [-0.45652174 -1.7826087 4.56521739]  
A의 역행렬은:  
[[-0.00869565 0.23478261 -0.04347826]  
[ 0.15652174 -0.22608696 -0.2173913 ]  
[ 0.08695652 -0.34782609 0.43478261]]  
A의 eigen value는:  
[12.55124265+0.j -2.43367295+0.j 1.8824303 +0.j]  
A의 eigen vector는:  
[[-0.9180878 -0.43522414 -0.2092878 ]  
[-0.37126665 0.81602247 -0.30952087]  
[-0.13883758 0.3803778 0.92757504]]

# Pandas

## • 설치 방법: 앞선 과정과 동일

The image illustrates the process of installing the pandas package in PyCharm. It shows the 'File' menu, the 'Settings' dialog box, the 'Project Interpreter' window, and the search results for 'pandas'.



# Pandas

## • 데이터 입력

```
import pandas as pd
import numpy as np

#Data 1: column matrix
a = pd.DataFrame([[1, 3, 5, np.nan, 9]])

#Print data 1
print("matrix a는:\n", a)
print("matrix a의 사이즈는:\n", a.shape)
print("a에서 첫번째 ([0]) 행(row)의 값은:\n", a.iloc[0,:])
print("a에서 두번째 ([1]) 행(row)의 값은:\n", a.iloc[0,0])
```



```
matrix a는:
   0  1  2  3  4
0  1  3  5 NaN  9
matrix a의 사이즈는:
(1, 5)
a에서 첫번째 ([0]) 행(row)의 값은:
0    1.0
1    3.0
2    5.0
3    NaN
4    9.0
Name: 0, dtype: float64
a에서 두번째 ([1]) 행(row)의 값은:
1
```

# Pandas

## • 데이터 입력

```
#Data 2: row matrix
b = pd.DataFrame([[1],[3],[5],[7],[9]])

#Print data 2
print("matrix b는: ", b)
print("b에서 세번째 ([2]) 행(row)의 값은:\n", b.iloc[2])
```



```
matrix b는:      0
0  1
1  3
2  5
3  7
4  9
b에서 세번째 ([2]) 행(row)의 값은:
0    5
Name: 2, dtype: int64
```

# 파이썬 기초: 시각화 패키지 활용

---

2023.01

Engineering Development Research Center (EDRC)

정 동 휘

---

파이썬 기초: 시각화 패키지 활용

## 개요

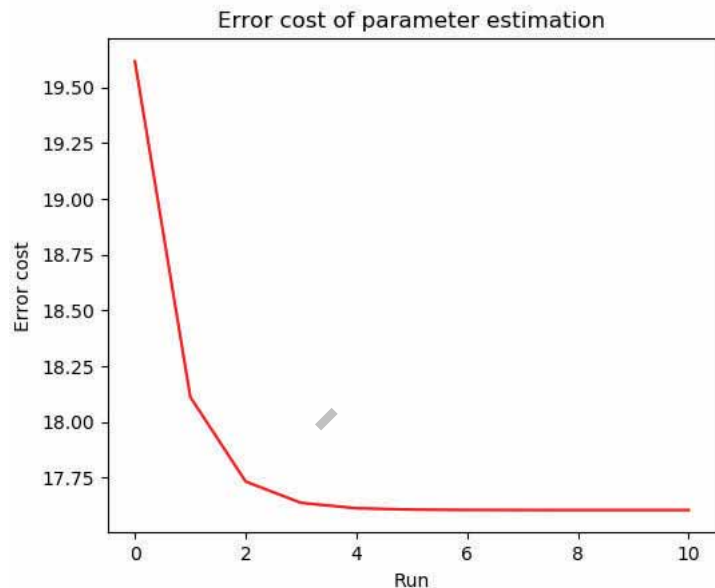
1. 시각화를 배워야 하는 이유
2. 대화형
3. 시각화 1: Matplotlib 패키지 활용
4. 시각화 2: 3D 그래프 그리기

COSMETIC

## 시각화를 배워야 하는 이유

- (방대한) 데이터 vs 그래프

```
Error cost 값은: [18.11201481]
Error cost 값은: [17.73277344]
Error cost 값은: [17.63718716]
Error cost 값은: [17.61309499]
Error cost 값은: [17.6070226]
Error cost 값은: [17.60549204]
Error cost 값은: [17.60510622]
Error cost 값은: [17.60500892]
Error cost 값은: [17.60498435]
Error cost 값은: [17.60497811]
```

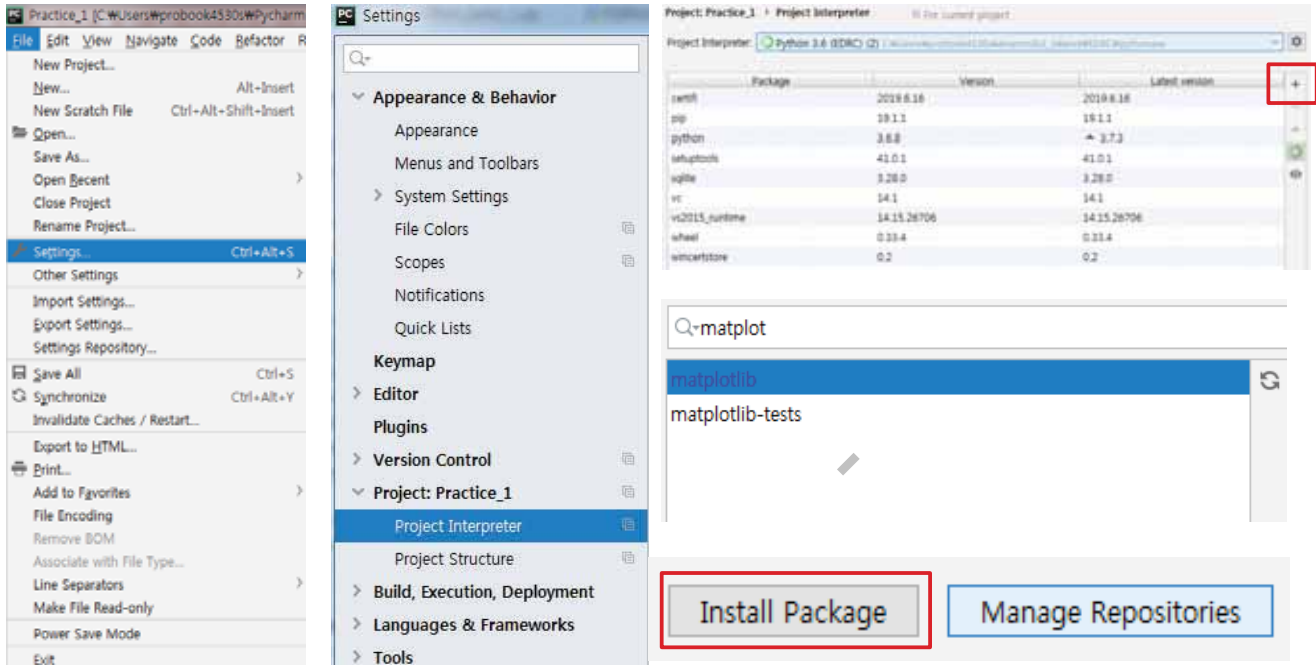


## 대화형

- Print()
- 입/출력

# 시각화 1: Matplotlib 패키지 활용

## • 설치 방법 1



# 시각화 1: Matplotlib 패키지 활용

```
#데이터 시각화 기초
import matplotlib.pyplot as plt
import numpy as np

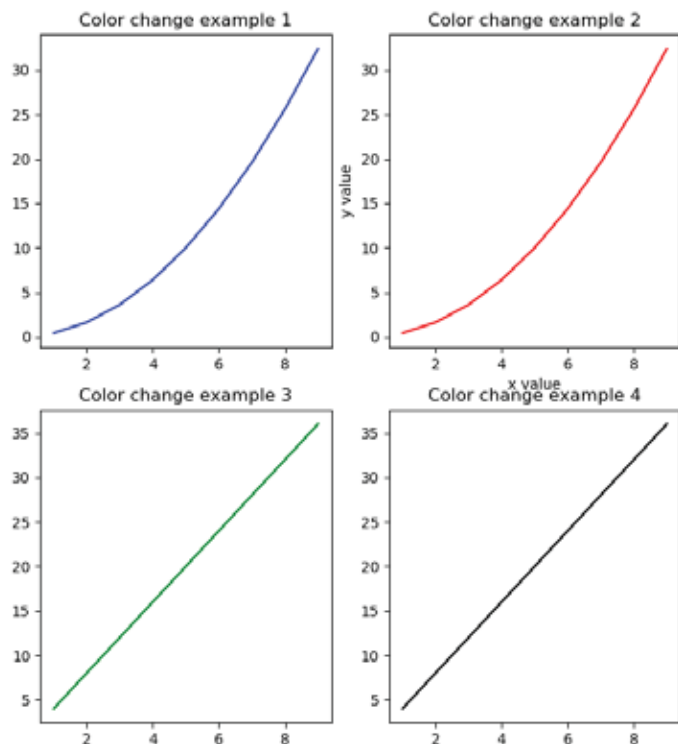
x = np.arange(1,10)
y1 = 0.4*x**2
y2 = 4*x

#색 변경
plt.figure(3)
plt.subplot(2,2,1)
plt.plot(x,y1,'b')
plt.title('Color change example 1')

plt.subplot(2,2,2)
plt.plot(x,y1,'r')
plt.title('Color change example 2')
plt.xlabel('x value')
plt.ylabel('y value')

plt.subplot(2,2,3)
plt.plot(x,y2,'g')
plt.title('Color change example 3')

plt.subplot(2,2,4)
plt.plot(x,y2,'k')
plt.title('Color change example 4')
plt.show()
```



## 시각화 2: 3D 그래프 그리기

#데이터 시각화 실화

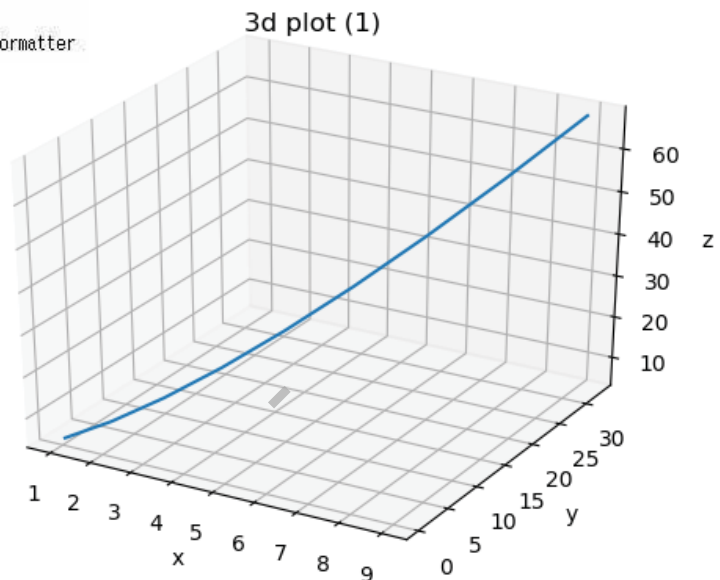
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

#3D print 기법

```
fig1 = plt.figure(1)
ax = fig1.add_subplot(111,projection='3d')
ax.plot(x,y,z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.title('3d plot (1)')
plt.show()

fig2 = plt.figure(2)
ax = fig2.add_subplot(111,projection='3d')
ax.scatter(x,y,z)
plt.show()

plt.figure(3).add_subplot(111,projection='3d')
plt.scatter(x,y,z)
plt.show()
```



## 시각화 2: 3D 그래프 그리기

### • Meshgrid 활용

#데이터 시각화 실화

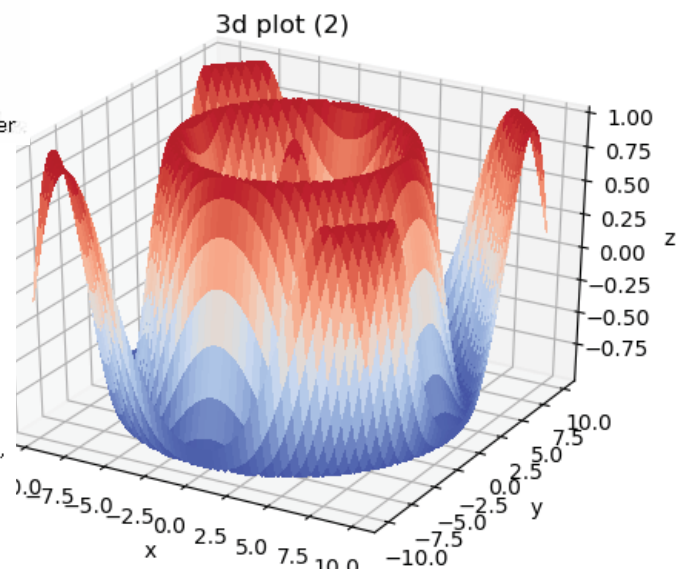
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

fig = plt.figure()
ax = fig.gca(projection='3d')

# Make data.
x = np.arange(-10, 10, 0.1)
y = np.arange(-10, 10, 0.1)
x_mesh, y_mesh = np.meshgrid(x, y) #meshgrid?
z_mesh = np.cos(np.sqrt(x_mesh**2 + y_mesh**2))

# Plot the surface.
surf = ax.plot_surface(x_mesh, y_mesh, z_mesh, cmap=cm.coolwarm,
                      linewidth=0, antialiased=False)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```



# 모델링 기초

---

2023.01

Engineering Development Research Center (EDRC)

정 동 휘

---

모델링 기초

## 개요

1. 형태 선택과 파라미터 추정
2. White/Black/Gray box
3. Classification/Regression
4. 비시계열/시계열
5. Supervised/Unsupervised
6. Linear/Nonlinear

## 형태 선택과 파라미터 추정

- 형태 선택 from (사전 공정 지식 + 데이터 + ...)
- 파라미터 추정 (= 가장 잘 맞는 파라미터 찾기)
- Optimization problem

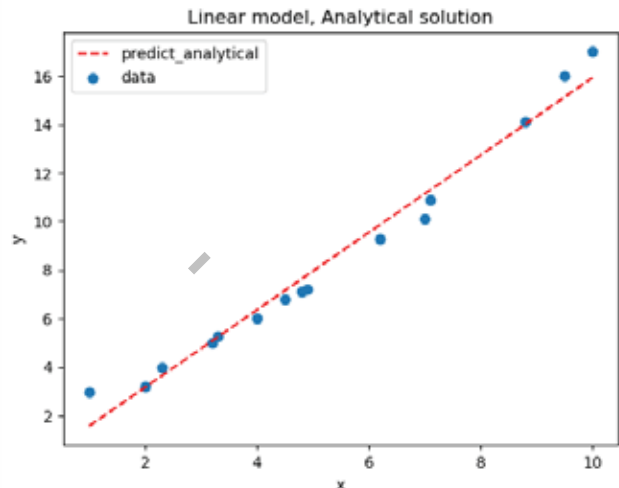
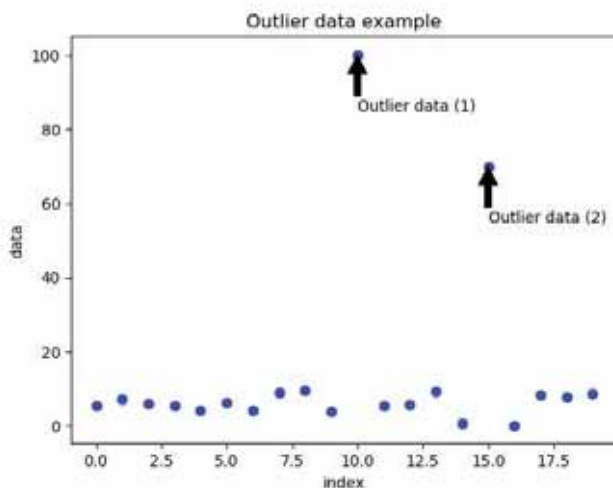
$$\begin{aligned} & \min_p \text{Cost}_{Error}(p) \\ & \text{s. t. } \text{Cost}_{Error}(p) = f(\text{데이터와 모델 차이}) \end{aligned}$$

## White/Black/Gray box model

- White-box model: 사전 공정 지식 이용  
예) 이상 기체 방정식, balance
- Black-box model: 데이터에만 의존  
예) (Deep) Neural network
- Gray-box model = (White + Black)-box model

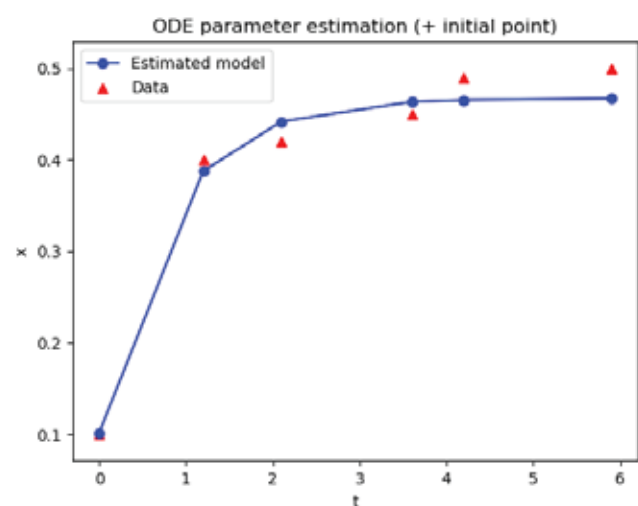
# Classification/Regression

- 분류: A냐 B냐
- 회귀:  $x=1$ 일 때  $y$ 의 값은 무엇이나



# 시계열/비시계열

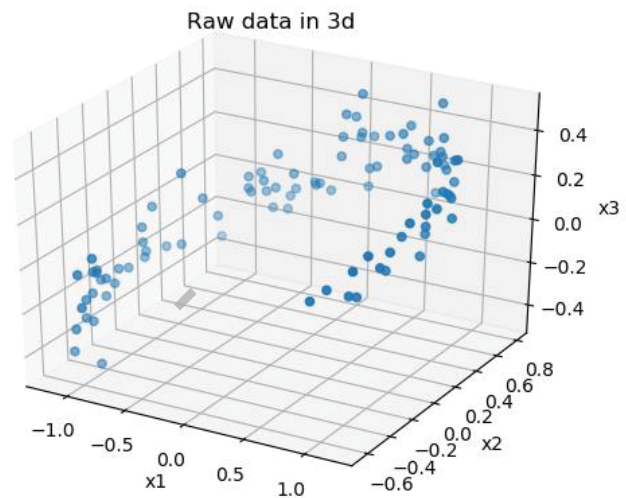
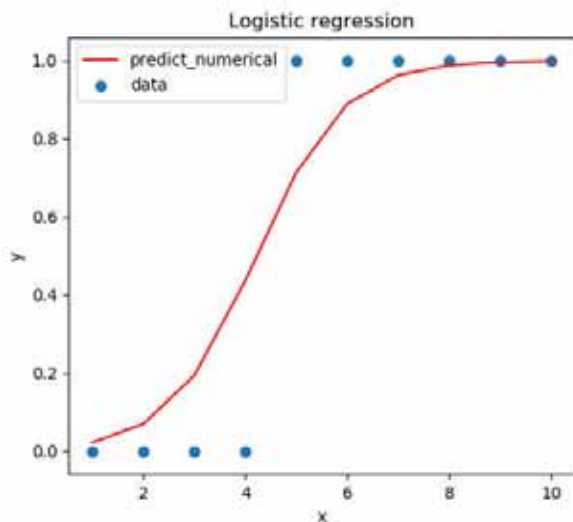
- 시계열: 앞의 state가 뒤의 state에 영향을 주는가
- 주의: 시간 데이터를 사용하면 시계열 분석 아님





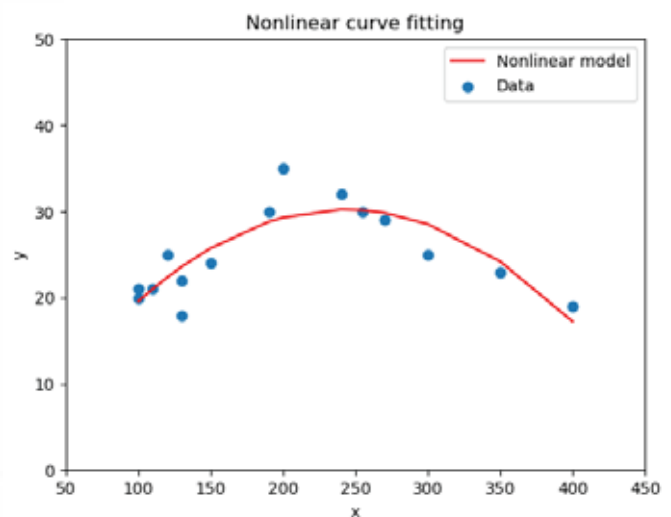
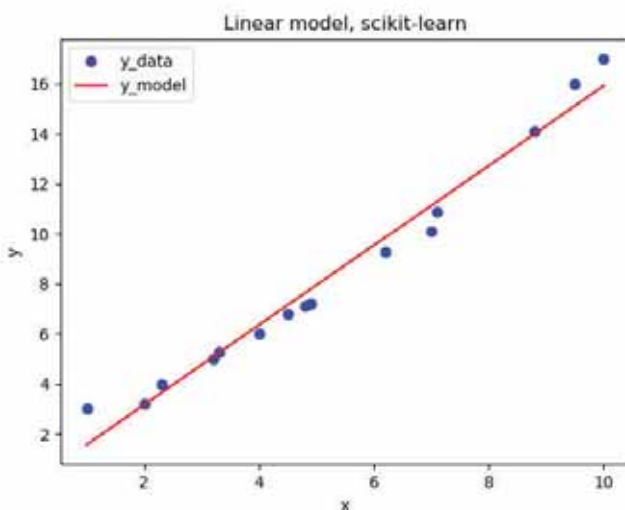
# Supervised/Unsupervised

- 지도:  $y$ 에 대한 설명이 붙어 있음
- 비지도:  $y$ 에 대한 설명이 안 붙어 있음



# Linear/Nonlinear

- 선형:  $x$ 가 2배 되면  $y$ 도 2배가 됨
- 비선형: 선형이 아닌 경우



# 모델링 심화: 선형 회귀 분석

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

---

모델링 심화: 선형 회귀 분석

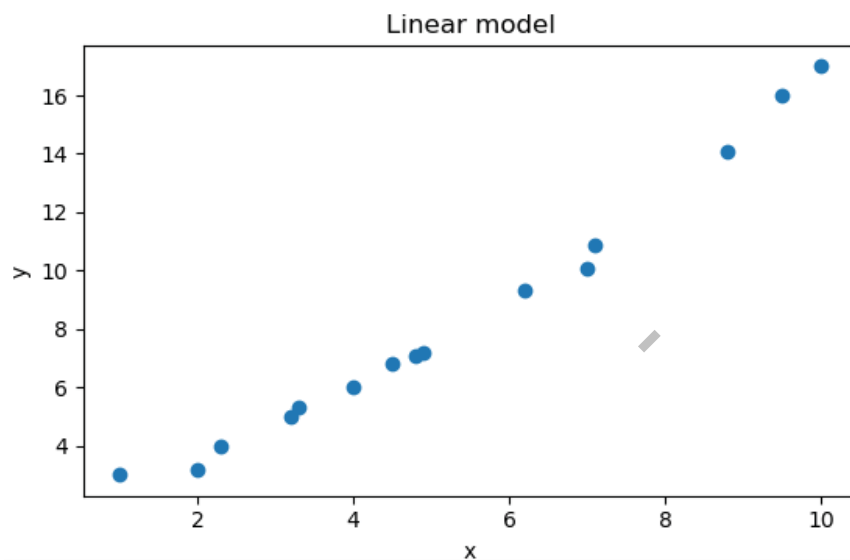
## 개요

1. 선형 모델
2. 회귀 분석
3. 파라미터 추정
4. Scikit-learn 패키지 활용
5. Tensorflow 패키지 활용

## 선형 모델

- 형태:  $y = ax$  (혹은  $y = ax + b$ )

$y_{\text{predict\_analytical}} = \text{coeff\_a} * x_{\text{data}} + \text{coeff\_b}$



## 회귀 분석 (Regression)

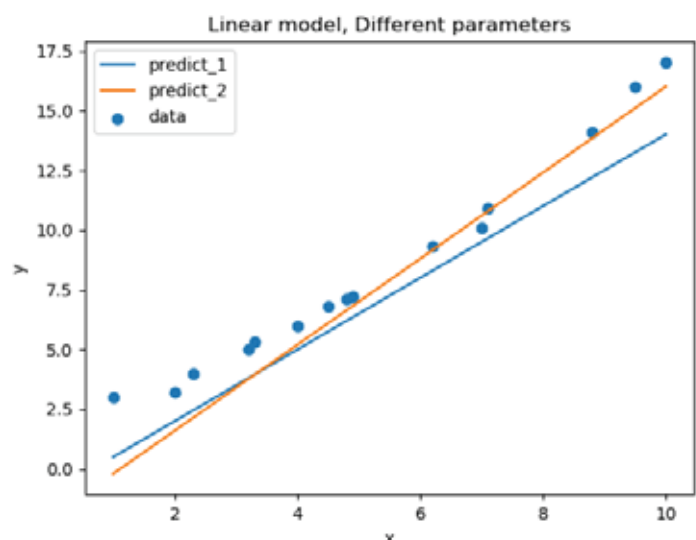
- 다른 파라미터일 때 회귀 분석 결과

*#일의의 파라미터*

$y_{\text{predict1}} = 1.5 * x_{\text{data}} - 1$

$y_{\text{predict2}} = 1.8 * x_{\text{data}} - 2$

```
plt.figure(2)
plt.scatter(x_data, y_data, label='data')
plt.plot(x_data, y_predict1, label='predict_1')
plt.plot(x_data, y_predict2, label='predict_2')
plt.legend(loc='upper left')
plt.title("Linear model, Different parameters")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



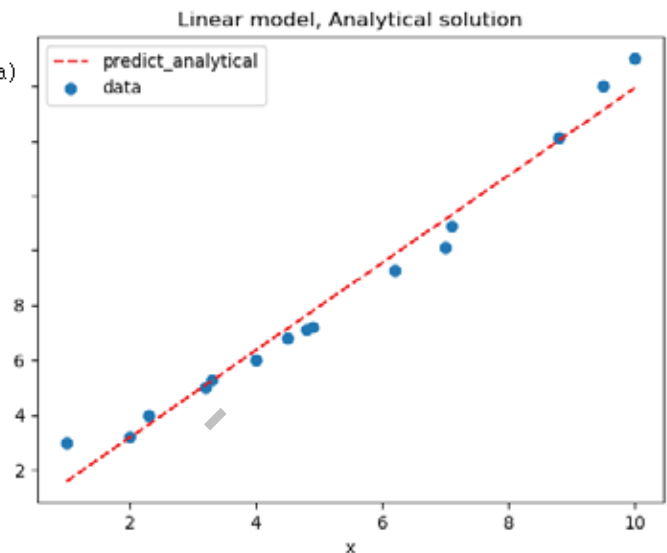
# 파라미터 추정

- Analytical solution

```
#Analytical solution
coeff = np.linalg.inv(
    x_data_mod.T.dot(x_data_mod)).dot(x_data_mod.T).dot(y_data)
print(coeff)
coeff_a = coeff[1,0]
coeff_b = coeff[0,0]
print("Analytical solution (slope): %n", coeff_a)
print("Analytical solution (bias): %n", coeff_b)
y_predict_analytical = coeff_a*x_data + coeff_b
```



Analytical solution (slope):  
1.5948928004723428  
Analytical solution (bias):  
-0.023904941141739133



# 파라미터 추정

- Numerical solution
- Gradient descent algorithm

```
#Error cost
def Linear_regression_cost(x_data_mod, y_data, p):
    y_predict = np.matmul(x_data_mod, p)
    N = np.size(x_data_mod, 0)
    cost = sum((y_data - y_predict)**2) / N
    return cost

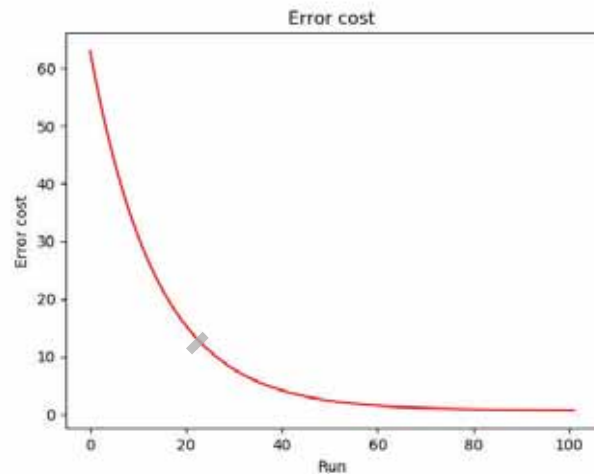
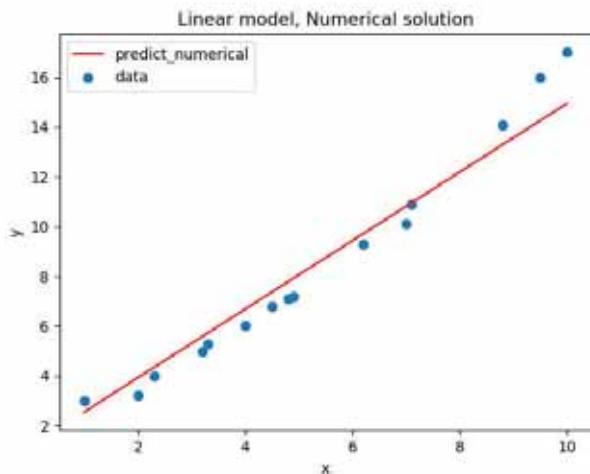
def Calculate_new_parameter(x_data_mod, y_data, p):
    p = np.array(p)
    step_size = 0.001
    N = np.size(x_data_mod, 0)
    Gradient = np.matmul((np.matmul(x_data_mod, p) - y_data).T, x_data_mod) / N
    p_new = p - step_size * Gradient
    p_new = p_new.T
    return p_new

for i in range(101):
    p_new_new = Calculate_new_parameter(x_data_mod, y_data, p_new[:, i].reshape(-1, 1))
    p_new = np.hstack((p_new, p_new_new))
    # print(p_new_new)
    cost_new = Linear_regression_cost(x_data_mod, y_data, p_new_new)
    cost = np.vstack((cost, cost_new))
    print(cost_new)
```

Numerical solution (slope):  
1.377479728512181  
Numerical solution (bias):  
1.1689134339039828

## 파라미터 추정

- Numerical solution



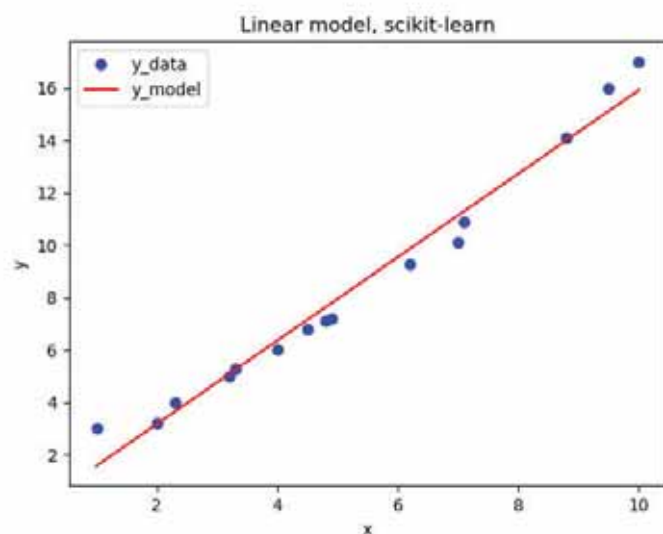
## Scikit-learn 패키지 활용

- 패키지 설치: scikit-learn 검색 후 install

Q-scikit-learn

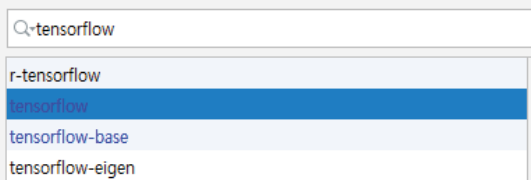
scikit-learn

```
#선형 회귀 분석 ( $y_{model} = b_0 + b_1 * x$ )
model = LinearRegression()
model.fit(x_data, y_data)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
y_model = model.intercept_ + model.coef_*x_data
```



# tensorflow 패키지 활용

- 패키지 설치: tensorflow 검색 후 install



```
W = tf.Variable([1.0], dtype=tf.float32) # 처음 보위로 줘도 상관없음
b = tf.Variable([0.0], dtype=tf.float32)

model = W*x_data + b

cost = tf.reduce_mean(tf.square(model - y_data))

rate = tf.Variable(0.002)
optimizer = tf.train.GradientDescentOptimizer(rate)
train = optimizer.minimize(cost)

#session 생성
sess = tf.Session()

#session 초기화 (*) 필수
init = tf.global_variables_initializer()
sess.run(init)
```

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

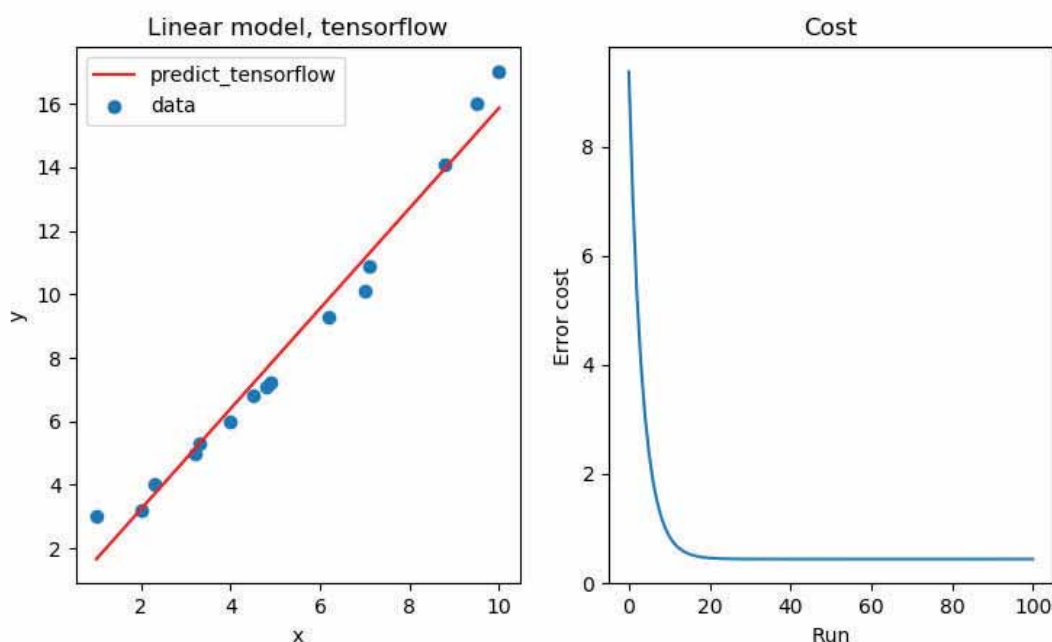
for i in range(101):
    sess.run(train)
    Error_cost=np.hstack((Error_cost, sess.run(cost)))

    if i%10 == 0:
        print(i, sess.run(cost), sess.run(W), sess.run(b))

#최종 결과를
coeff = sess.run(W)
bias = sess.run(b)
y_predict = coeff*x_data + bias
```

# tensorflow 패키지 활용

- 패키지 설치: tensorflow 검색 후 install



# 모델링 심화: 로지스틱 회귀 분석

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

---

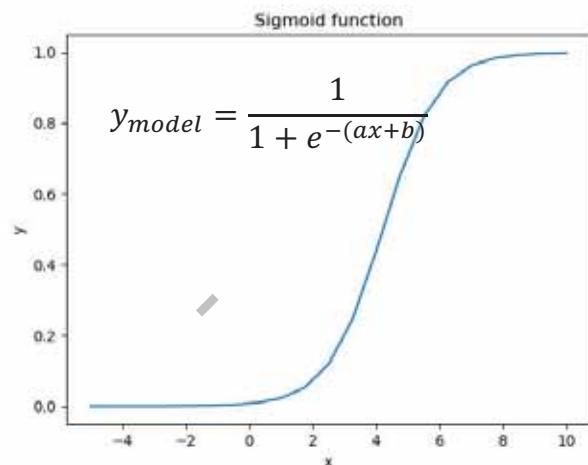
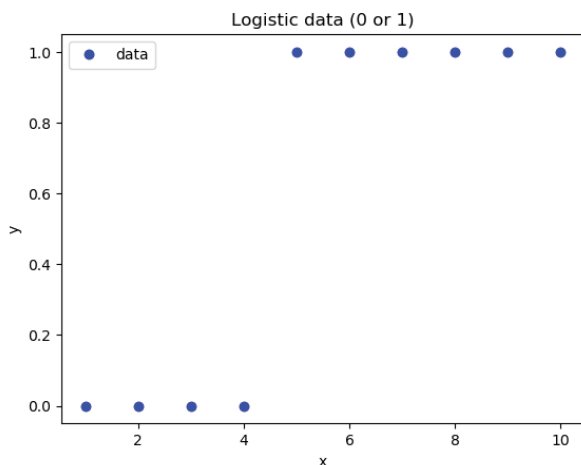
모델링 심화: 로지스틱 회귀 분석

## 개요

1. 로지스틱 회귀 분석 모델
2. Error cost 함수
3. 파라미터 추정
4. Scikit-learn 패키지 활용
5. 1-D 문제
6. 다차원 문제

## 로지스틱 회귀 분석 모델

- 일반적인 선형 회귀 모델과의 차이점: y 값의 특성
- 모델에 비선형 특성이 포함되어야 함



## Error cost 함수

- 자연 로그 (ln) 씌워서 비선형성을 펴줌.

$$y_{model} = \frac{1}{1 + e^{-(ax+b)}}$$

$$Cost_{Error} = -y_{data} * \ln y_{model} - (1 - y_{data}) * \ln(1 - y_{model})$$

```
#Logistic regression cost
def Logistic_regression_cost(x_data_mod,y_data,p):
    return -np.matmul(y_data.T, np.log(Sigmoid_fun(x_data_mod,p)))- np.matmul((1-y_data).T, np.log(1-Sigmoid_fun(x_data_mod,p)))
```



# 파라미터 추정

```
def Calculate_new_parameter(x_data_mod, y_data, p):
    p=np.array(p)
    step_size = 0.1
    N = np.size(x_data_mod, 0)
    Gradient = np.matmul(x_data_mod[:,1].T, Sigmoid_fun(x_data_mod, p) - y_data)/N
    p_new = p.T - step_size*Gradient
    p_new = p_new.T
    return p_new

#initial guess
p_new = np.array([[ -1.1],[5]])
cost = Logistic_regression_cost(x_data_mod, y_data, p_new)

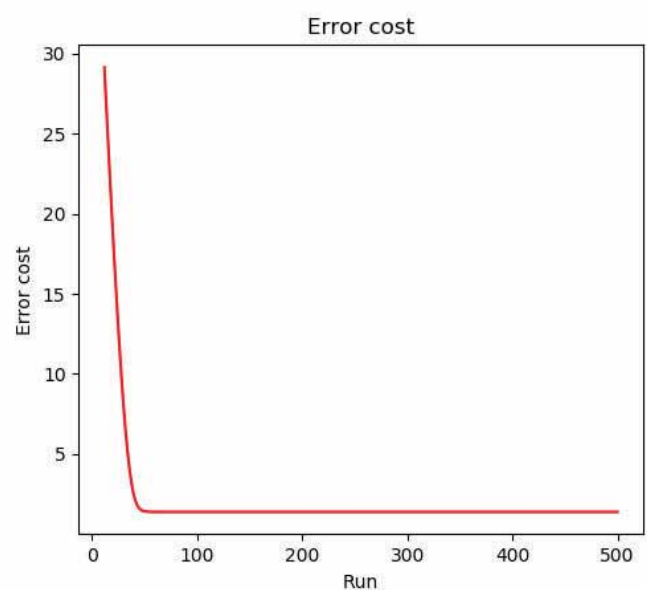
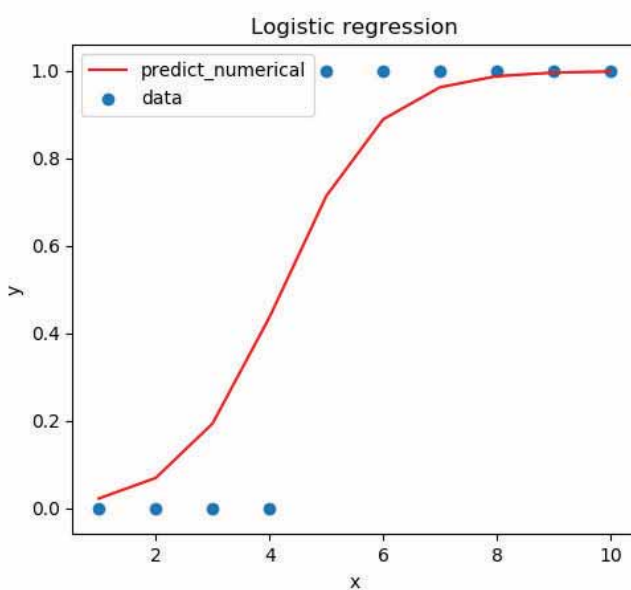
#iteration
for i in range(500):
    p_new_new = Calculate_new_parameter(x_data_mod, y_data, p_new[:, i].reshape(-1,1))
    p_new = np.hstack((p_new, p_new_new))
    cost_new = Logistic_regression_cost(x_data_mod, y_data, p_new_new)
    cost = np.vstack((cost, cost_new))
    print(cost_new)
```



Sigmoid cost value by p\_example

```
[[25.15828901]]
[[23.86409806]]
[[22.58598956]]
[[21.32546031]]
[[20.08399388]]
[[18.86307961]]
[[17.66425603]]
[[16.48917675]]
[[15.33969071]]
[[14.21792263]]
[[13.12633469]]
[[12.06775019]]
[[11.04532548]]
[[10.06246901]]
[[9.12272314]]
[[8.22963764]]
[[7.38666376]]
```

## 1D-문제



## Sklearn 패키지 활용

```
import numpy as np
from sklearn.linear_model import LogisticRegression
```

```
#데이터 분리
x_data = data[:,0].reshape(-1,1) #row data 형태로 만들어 줄
y_data = data[:,1]

#model 생성
model = LogisticRegression(random_state=0, solver='lbfgs')

#Logistic regression
solution = model.fit(x_data,y_data)

#결과
print("분류 결과: \n", solution.predict(x_data))
print("분류 점수: \n", solution.score(x_data,y_data))
```



분류 결과:

[0 0 0 0 1 1 1 1 1 1]

분류 점수:

1.0

## 다차원 문제

- Tensorflow + softmax 이용

```
#x_data: (10 by 4)
x_data = np.array([[0., 0., 0., 0.],
                   [1., 1., 0., 1.],
                   [1., 0., 1., 0.],
                   [0., 1., 0., 1.],
                   [0., 0., 1., 0.],
                   [1., 0., 0., 1.],
                   [1., 1., 1., 0.],
                   [0., 0., 1., 0.],
                   [1., 0., 0., 1.],
                   [0., 0., 0., 1.]], dtype='f')
```

```
#y_data: (10 by 3)
y_data = np.array([[1.,0.,0.],
                   [0.,1.,0.],
                   [1.,0.,0.],
                   [0.,0.,1.],
                   [1.,0.,0.],
                   [0.,1.,0.],
                   [0.,0.,1.],
                   [1.,0.,0.],
                   [0.,0.,1.],
                   [0.,0.,1.]], dtype='f')
```

## 다차원 문제

- Tensorflow + softmax 이용

```
#Model
x = tf.placeholder(tf.float32, [None, 4], name="input")
y = tf.placeholder(tf.float32, [None, 3], name="output")
W = tf.Variable(tf.random_normal([4,3]))
b = tf.Variable(tf.random_normal([3]))
L = tf.matmul(x, W)
y_predict = tf.nn.softmax(L)

#Objective function (softmax이므로 cross-entropy cost function 쓰자.)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y, logits=L))

#파라미터 찾기
train = tf.train.GradientDescentOptimizer(learning_rate=rate).minimize(cost)

#세션 생성
sess1 = tf.Session()

#세션 초기화 #(*)필수
init1 = tf.global_variables_initializer()
sess1.run(init1)

#학습 (최적 파라미터 찾기)
for i in range(100001):
    sess1.run(train, feed_dict={x: x_data, y: y_data})

    if i%10000 == 0:
        print(i, sess1.run(cost, feed_dict={x: x_data, y: y_data}))

sess1.close()
```

Run 횟수: 0 Error cost: 0.98095685  
Run 횟수: 10000 Error cost: 0.2510945  
Run 횟수: 20000 Error cost: 0.2497911  
Run 횟수: 30000 Error cost: 0.24935699  
Run 횟수: 40000 Error cost: 0.24914011  
Run 횟수: 50000 Error cost: 0.24901009  
Run 횟수: 60000 Error cost: 0.24892339  
Run 횟수: 70000 Error cost: 0.24886158  
Run 횟수: 80000 Error cost: 0.24881515  
Run 횟수: 90000 Error cost: 0.24877906  
Run 횟수: 100000 Error cost: 0.24875025

## 모델링 심화: Ensemble learning

2023.01

Engineering Development Research Center (EDRC)

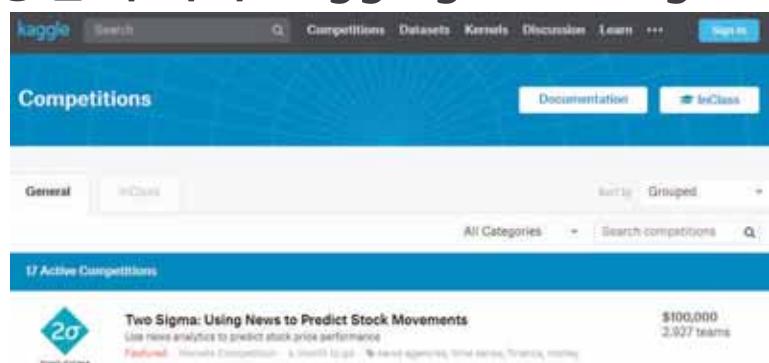
정동휘

# 개요

1. Ensemble learning이란?
2. 장점
3. Bagging (Bootstrap Aggregating)
4. Boosting

## Ensemble learning 이란?

- 여러 모델의 결과를 혼합하여 최종 결론을 낸다.
- 예 1) 다수결
- 예 2) 가중치 준 평균
- 여러 모델을 만드는 방법에 따라 Bagging/Boosting 등으로 나뉜다.
- [www.kaggle.com](http://www.kaggle.com)



## 장점

- 모델 하나만 사용했을 때 보다 예측 성능이 향상 됨.

#Ensemble learning 기초: 왜 하나의 모델보다 여러 모델을 사용하는 것이 유리한가?

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score
from sklearn.ensemble import BaggingClassifier
```

```
X, Y = make_moons(n_samples=600, noise=0.5, random_state=30)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=30)
```

#local model 하나만 사용했을 때

```
result_1 = DecisionTreeClassifier(random_state=30)
result_1.fit(X_train, Y_train)
Y_pred_1 = result_1.predict(X_test)
print("하나의 local model만 사용했을 때의 정확도: %f" % accuracy_score(Y_test, Y_pred_1))
```

하나의 local model만 사용했을 때의 정확도:

0.7866666666666666

여러개의 local model을 사용하여 ensemble learning 했을 때의 정확도:

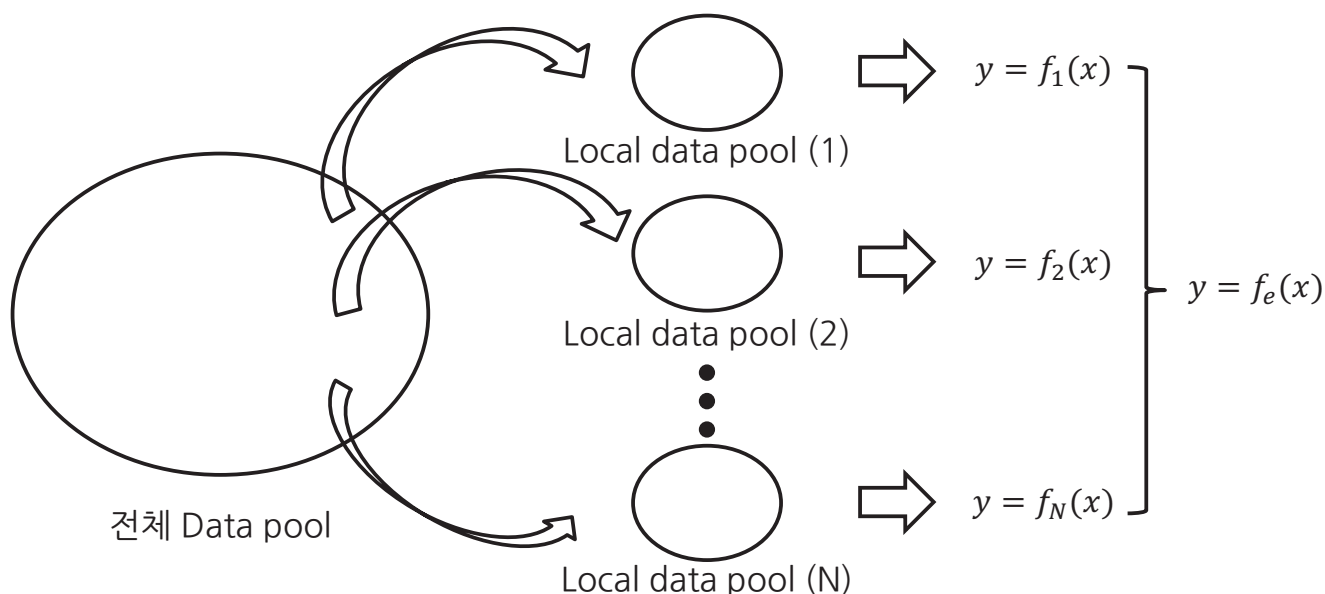
0.8266666666666667

#local model을 여러개 사용했을 때 (Ensemble)

```
result_2 = BaggingClassifier(DecisionTreeClassifier(random_state=30), n_estimators=100, max_samples=100,
                             bootstrap=True, n_jobs=-1, random_state=30)
result_2.fit(X_train, Y_train)
Y_pred_2 = result_2.predict(X_test)
print("여러개의 local model을 사용하여 ensemble learning 했을 때의 정확도: %f" % accuracy_score(Y_test, Y_pred_2))
```

## Bagging (Bootstrap Aggregating)

- 반복적인 복원 추출을 통한 여러 모델 생성



# Bagging (Bootstrap Aggregating)

#Bagging (Bootstrap aggregating): 중복을 허용한 랜덤 추출

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score
from sklearn.ensemble import BaggingClassifier
```



하나의 local model만 사용했을 때의 정확도:

0.7866666666666666

여러개의 local model을 사용하여 ensemble learning 했을 때의 정확도:

0.8266666666666667

#local model 하나만 사용했을 때

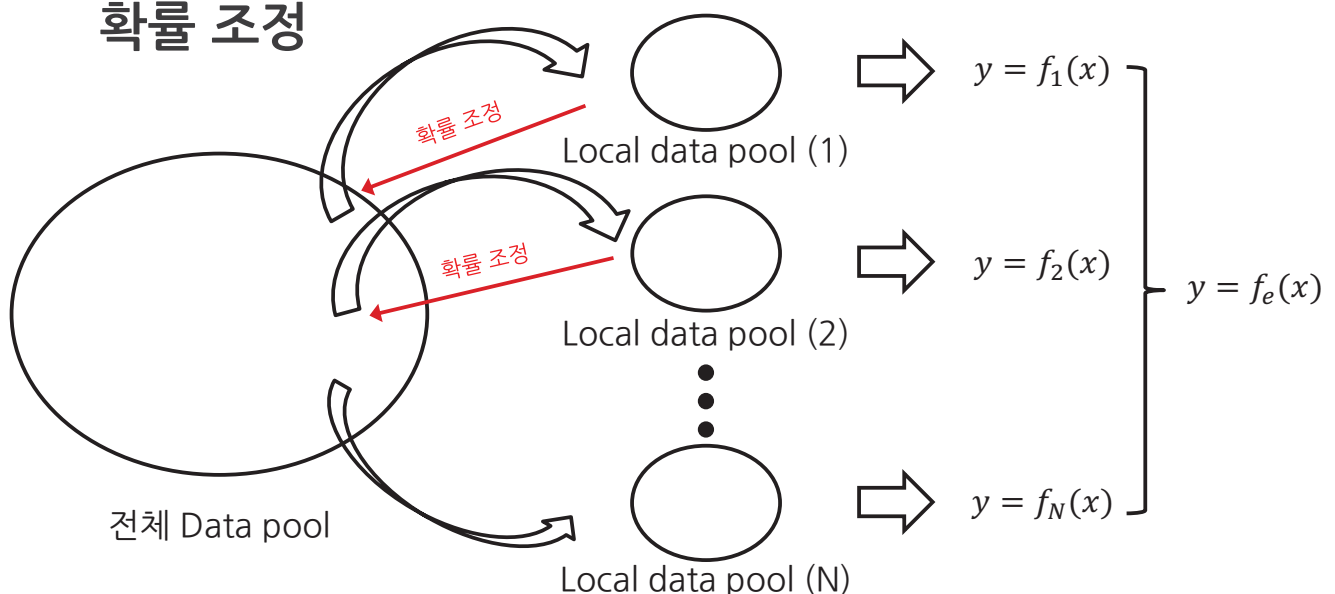
```
result_1 = DecisionTreeClassifier(random_state=30)
result_1.fit(X_train, Y_train)
Y_pred_1 = result_1.predict(X_test)
print("하나의 local model만 사용했을 때의 정확도: %f" % accuracy_score(Y_test, Y_pred_1))
```

#local model을 여러개 사용했을 때 (Ensemble)

```
result_2 = BaggingClassifier(DecisionTreeClassifier(random_state=30), n_estimators=100, max_samples=100,
                             bootstrap=True, n_jobs=-1, random_state=30)
result_2.fit(X_train, Y_train)
Y_pred_2 = result_2.predict(X_test)
print("여러개의 local model을 사용하여 ensemble learning 했을 때의 정확도: %f" % accuracy_score(Y_test, Y_pred_2))
```

# Boosting

- 전 단계에서 잘 못 맞춘 데이터가 더 잘 뽑히도록 추출 확률 조정



# Boosting

#Boosting: 먼저 만들어진 local model의 단점을 보완하는 다른 local model

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```
#local model 하나만 사용했을 때
result_1 = DecisionTreeClassifier(random_state=30)
result_1.fit(X_train, Y_train)
Y_pred_1 = result_1.predict(X_test)
print("하나의 local model만 사용했을 때의 정확도: %f" % accuracy_score(Y_test, Y_pred_1))
```

```
#1. AdaBoost (Adaptive boosting)
result_2 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=100, learning_rate=1.,
                              algorithm='SAMME.R', random_state=30)
result_2.fit(X_train, Y_train)
Y_pred_2 = result_2.predict(X_test)
print("Adaptive Boosting ensemble learning 했을 때의 정확도: %f" % accuracy_score(Y_test, Y_pred_2))
```

```
#2. Gradient boosting
result_3 = GradientBoostingClassifier(loss='deviance', learning_rate=0.01, n_estimators=200, random_state=30)
result_3.fit(X_train, Y_train)
Y_pred_3 = result_3.predict(X_test)
print("Gradient Boosting ensemble learning 했을 때의 정확도: %f" % accuracy_score(Y_test, Y_pred_3))
```

하나의 local model만 사용했을 때의 정확도:

0.7866666666666666

Adaptive Boosting ensemble learning 했을 때의 정확도:

0.8266666666666667

Gradient Boosting ensemble learning 했을 때의 정확도:

0.8066666666666666



# Tensorflow 이용한 모델링 및 파라미터 추정

2023.01

Engineering Development Research Center (EDRC)

정동휘

## 개요

1. Tensorflow 패키지 설치
2. 상수와 변수
3. Placeholder
4. 세션
5. 모델링
6. 파라미터 추정

## Tensorflow 패키지 설치

- 패키지 설치: tensorflow 검색 후 install

The screenshot shows the PyCharm IDE interface. The 'File' menu is open, and 'Settings' is selected. The 'Settings' dialog is open, showing the 'Project: Practice\_1' settings. The 'Project Interpreter' tab is active, displaying a table of installed packages. A search bar at the bottom contains the text 'tensorflow'. The search results show the following packages:

Package	Version	Latest version
tensorflow	2.0.0	2.0.0
pip	19.1.1	19.1.1
python	3.6.8	3.7.3
setuptools	41.0.1	41.0.1
sqlite	3.28.0	3.28.0
vc	14.1	14.1
vs2015_runtime	14.15.26706	14.15.26706
wheel	0.33.4	0.33.4
wincertstore	0.2	0.2

The 'Install Package' button is highlighted with a red box.



# 상수

## • 상수 지정

```
#tensorflow 기본 개념 1

import tensorflow as tf

#상수 지정
con1 = tf.constant(10)
con2 = tf.constant(20, name='con2')
```

## • 세션 생성

```
#session 생성
sess1 = tf.Session()
```

# 상수의 연산

```
#상수 연산
result1 = tf.add(con1,con2)
print(result1)
print("Tensorflow 덧셈 결과: ", sess1.run(result1))

result2 = tf.subtract(con1,con2)
print("Tensorflow 뺄셈 결과: ", sess1.run(result2))

result3 = tf.multiply(con1,con2)
print("Tensorflow 곱셈 결과: ", sess1.run(result3))

result4 = tf.truediv(con1, con2)
print("Tensorflow 나눗셈 결과: ", sess1.run(result4))
```



```
Tensor("Add:0", shape=(), dtype=int32)
Tensorflow 덧셈 결과: 30
Tensorflow 뺄셈 결과: -10
Tensorflow 곱셈 결과: 200
Tensorflow 나눗셈 결과: 0.5
```

```
#상수 행렬 연산
con_mat1 = tf.constant([[10, 20]], dtype=tf.float32)
con_mat2 = tf.constant([[30],[40]],dtype=tf.float32)
mat_multiply1 = tf.matmul(con_mat1, con_mat2)
mat_multiply2 = tf.matmul(con_mat2, con_mat1)
print("Tensorflow 행렬 곱_1: \n", sess1.run(mat_multiply1))
print("Tensorflow 행렬 곱_2: \n", sess1.run(mat_multiply2))
```



```
Tensorflow 행렬 곱_1:
[[1100.]]
Tensorflow 행렬 곱_2:
[[300. 600.]
 [400. 800.]]
```

## 변수

```
#변수 설정 (예를 들면 placeholder로 받은 데이터로 찾아야하는 파라미터 값)
var1 = tf.Variable([3], dtype=tf.float32)
var2 = data1*var1
sess2 = tf.Session()
init1 = tf.global_variables_initializer() #(*)필수
sess2.run(init1) #(*)필수
result6 = sess2.run(var2, feed_dict={data1: input_data})
print("Variable 예제의 결과: ", result6)
```



Variable 예제의 결과: [3. 6. 9.]

```
#변수 행렬 연산
var_mat1 = tf.Variable([[10, 20]], dtype=tf.float32)
var_mat2 = tf.matmul(var_mat1, con_mat2)
sess3 = tf.Session()
init2 = tf.global_variables_initializer()
sess3.run(init2)
print("Variable 행렬 연산의 결과: ", sess3.run(var_mat2))
```



Variable 행렬 연산의 결과: [[1100.]]

## Placeholder

```
#설정할 값을 넣을 수 있는 그릇: placeholder
input_data=[1,2,3]
data1 = tf.placeholder(dtype=tf.float32)
data2 = data1**2
result5 = sess1.run(data2, feed_dict={data1:input_data})
print("Placeholder 예제의 결과: ", result5)
```



Placeholder 예제의 결과: [1. 4. 9.]

```
#Placeholder 행렬 연산
input_data_mat = [[10, 20], [30, 40]]
data_mat1 = tf.placeholder(dtype=tf.float32, shape=[2,2])
data_mat2 = tf.matmul(data_mat1, data_mat1)
print("Placeholder 사용한 행렬 곱: ", sess1.run(data_mat2,
                                                feed_dict={data_mat1: input_data_mat}))
```



Placeholder 사용한 행렬 곱:  
[[ 700. 1000.]  
[1500. 2200.]]

## 세션

- 문제 풀이의 시작 (가스 불 켜는 것)

```
init = tf.global_variables_initializer()

sess = tf.Session()

sess.run(init)
```

## 모델링

- 다차원 선형 모델 예제

```
x1_data = [ 2.,  1.,  3.,  0.,  7.]
x2_data = [ 0.,  2.,  3.,  4.,  5.]
y_data = [8.3, -3.5,  0, -15, 6.]

W1 = tf.Variable([3], dtype=tf.float32)
W2 = tf.Variable([4], dtype=tf.float32)
b = tf.Variable([1.], dtype=tf.float32)

hypothesis = W1*x1_data + W2*x2_data + b
```

## 파라미터 추정

- Error cost 함수 설정

```
cost = tf.reduce_mean(tf.square(hypothesis - y_data))

rate = tf.Variable(0.01)
optimizer = tf.train.GradientDescentOptimizer(rate)
train = optimizer.minimize(cost)
```

- Gradient descent algorithm

```
Error_cost=[]

for step in range(101):

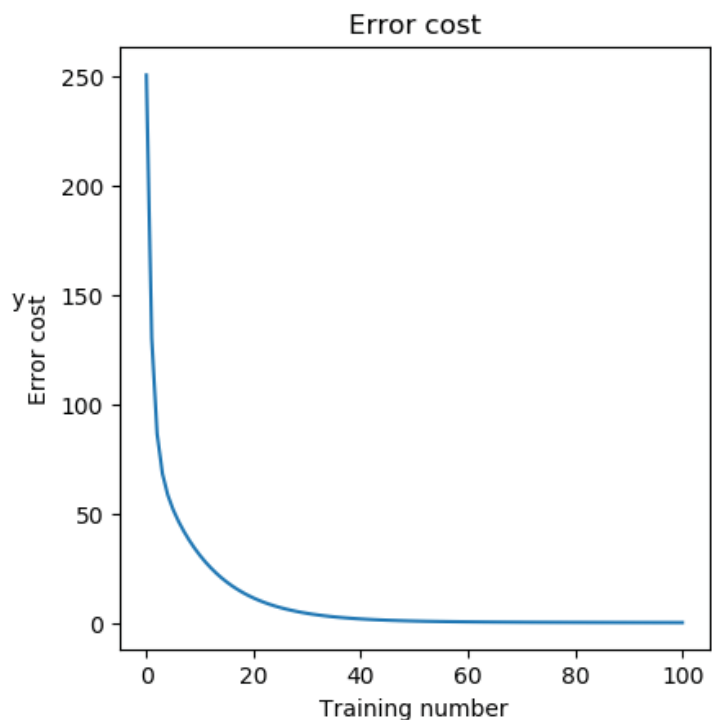
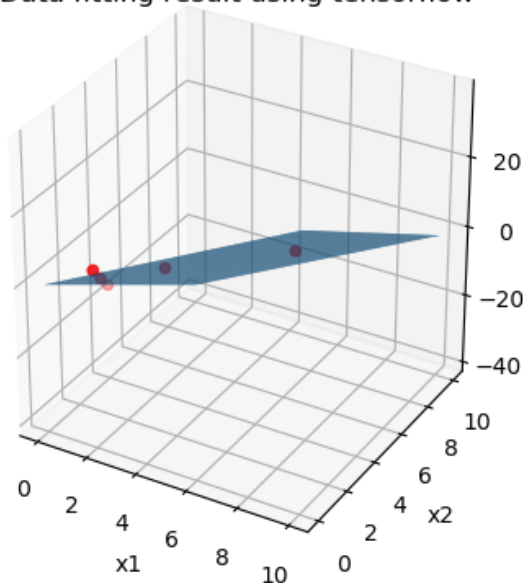
    sess.run(train)
    Error_cost=np.append(Error_cost,sess.run(cost))

    if step%10 == 0:
        print(step, sess.run(cost), sess.run(W1),
              sess.run(W2), sess.run(b))

W1_final = sess.run(W1)
W2_final = sess.run(W2)
b_final = sess.run(b)
```

## 결과

Data fitting result using tensorflow



# 모델링 심화: First principle model

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

---

모델링 심화: First principle model

## 개요

1. White box model
2. 각종 balance equations
3. Ordinary differential equations (ODE)

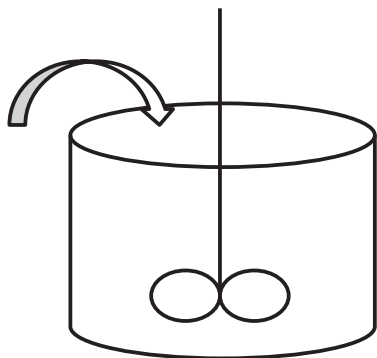
COSMETIC

## White-box model

- 기존의 공정 지식에 기반
- 형태에 대한 정보가 있음
- 이상 기체 방정식
- Empirical equations
- 각종 balance equations = time-dependent

## 각종 balance equations

- Accumulation = In - Out + Generation
- 예시) Batch reactor



# Ordinary differential equations (ODE)

- 목표: 식을  $t=[0,10]$  구간에서 적분하기

$$\frac{dx}{dt} = -ax$$

$$a = 2$$

$$x(0) = 1$$

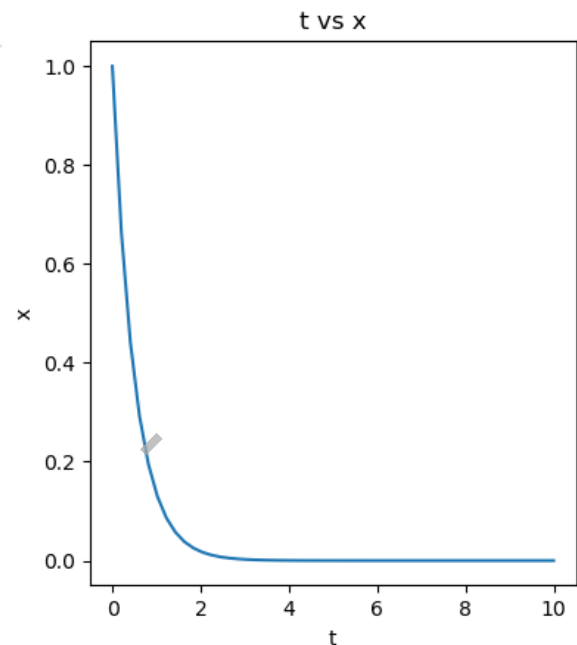
```
#Ordinary differential equation (ODE)
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

#ODE example 1
def ode_model1(x,t):
    a=2
    dxdt = -a*x
    return dxdt

#Initial value
x_initial=1

#time range
t = np.linspace(0,10)

#ODE solve
x = odeint(ode_model1, x_initial, t)
```



# Ordinary differential equations (ODE)

- 목표: 적분할 때 파라미터 바꾸기

$$\frac{dx}{dt} = -ax + p$$

$$a = 2$$

$$x(0) = 1$$

$$p = 1, 1.1$$

```
#Ordinary differential equation (ODE) 풀이 기호
#Parameter
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

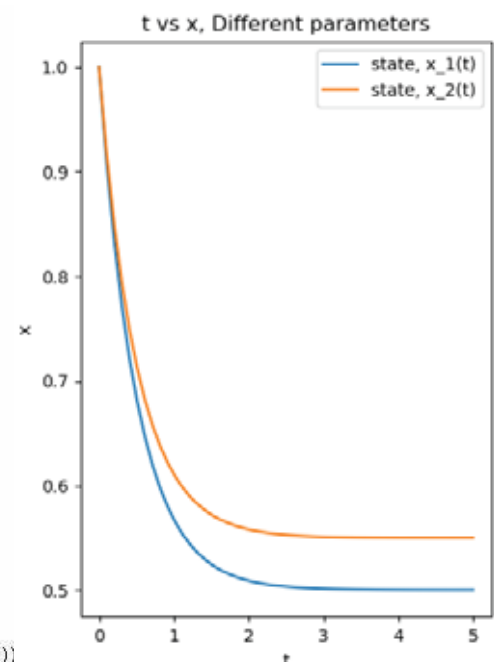
#ODE example 2
def ode_model_3(x,t,p):
    a = 2 #internal parameter
    dxdt = -a*x + p
    return dxdt

#External parameter
p_1=1
p_2=1.1

#Initial value
x_initial=1

#time range
t = np.linspace(0,5)

#ODE solve
x_1 = odeint(ode_model_3, x_initial, t, args=(p_1,))
x_2 = odeint(ode_model_3, x_initial, t, args=(p_2,))
```



# Ordinary differential equations (ODE)

- 목표: 적분할 때 input 넣기

$$\frac{dx}{dt} = -ax + u$$

$$a = 2$$

$$x(0) = 1$$

```
#Ordinary differential equation (ODE) 풀이 기초
#input design 1 (continuous)

import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

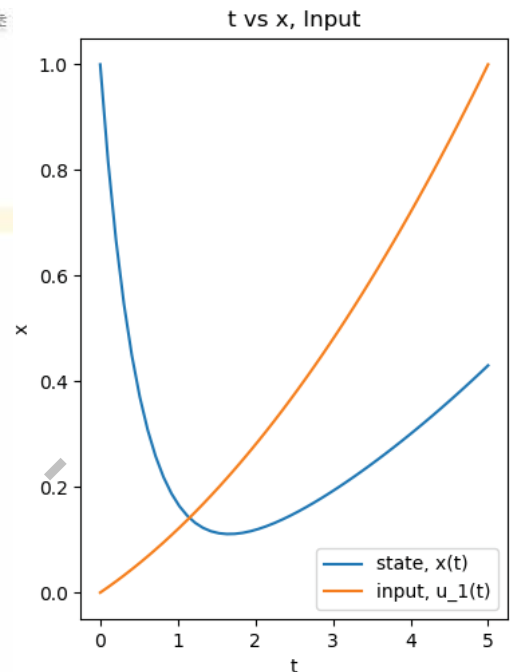
#ODE example 2
def ode_model_2(x,t):
    a=2
    u=input_1(t)
    dxdt = -a*x+u
    return dxdt

#input design
def input_1(t):
    u=0.1*t + 0.02*t**2
    return u

#initial value
x_initial=1

#time range
t = np.linspace(0,5)

#ODE solve
x = odeint(ode_model_2, x_initial, t)
u_1 = input_1(t)
```



# Ordinary differential equations (ODE)

- 목표: 시간 별로 나눠서 적분하기

$$\frac{dx}{dt} = -ax + p$$

$$a = 2$$

$$x(0) = 0.1$$

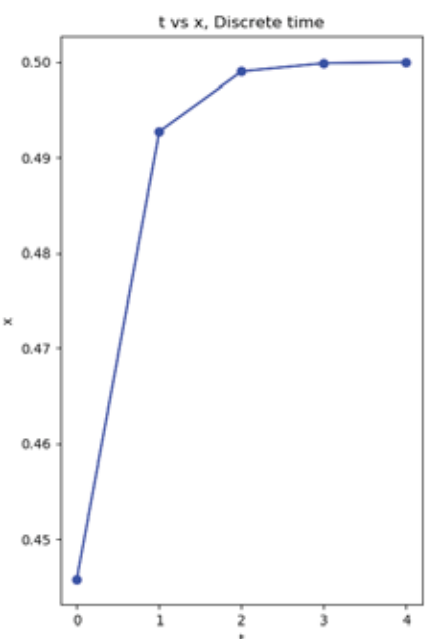
$$p = 1$$

```
#ODE example 2
def ode_model_3(x,t,p):
    a = 2 #internal parameter
    dxdt = -a*x + p
    return dxdt

#External parameter
p_1=1

def ode_model_3_value1(x_initial, start_time, end_time):
    time = np.linspace(start_time, end_time)
    x=odeint(ode_model_3, x_initial, time, args=(p_1,))
    return x

#각 시간 구간을 나눠서 적분
x_=[0.1]
time=[]
for i in range(5):
    new_start_time = i
    new_end_time = i+1
    time=np.append(time,i)
    x_result=ode_model_3_value1(x_[-1], new_start_time, new_end_time)
    x_=np.append(x_, x_result[-1])
```





# 데이터 처리: 모델링을 위한 데이터 분류

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

---

데이터 처리: 모델링을 위한 데이터 분류

## 개요

1. Test/Validation data 분류
2. K-fold cross validation

## Train/Validation 데이터 분류

- Train: 모델이 학습되는 데이터
- Validation: 학습된 모델이 얼마나 정확한지 판단
- 보통 전체 데이터에서 일정 비율 (예: 20%)을 Validation으로 random 추출하고 나머지를 Train으로 사용
- 주의 사항: 둘을 섞으면 안 됨.

## Train/Validation 데이터 분류

```
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn.datasets import load_boston
import pandas as pd
import numpy as np
```

#전체 데이터 중에서 80%만 뽑아서 train 시키고 나머지 20%는 검증에 사용하자.

```
N = len(df)
ratio = 0.8
np.random.seed(0)
idx_train = np.random.choice(np.arange(N), np.int(ratio * N))
idx_test = list(set(np.arange(N)).difference(idx_train))

df_train = df.iloc[idx_train]
df_test = df.iloc[idx_test]

model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.feature_names), data=df_train)
result = model.fit()
print(result.summary())
```

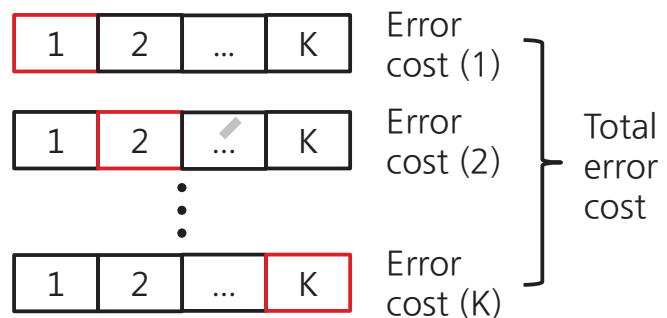


OLS Regression Results							
Dep. Variable:	MEDV	R-squared:	0.735				
Model:	OLS	Adj. R-squared:	0.726				
Method:	Least Squares	F-statistic:	83.27				
Date:	Mon, 08 Jul 2019	Prob (F-statistic):	8.55e-104				
Time:	11:30:25	Log-Likelihood:	-1222.4				
No. Observations:	404	AIC:	2473.				
Df Residuals:	390	BIC:	2529.				
Df Model:	13						
Covariance Type:	nonrobust						
	coef	std err	t	P> t	[0.025	0.975]	
Intercept	43.1078	6.429	6.706	0.000	30.469	55.747	
CRIM	-0.0855	0.036	-2.344	0.020	-0.157	-0.014	
ZN	0.0465	0.015	3.156	0.002	0.018	0.075	
INDUS	0.1047	0.075	1.388	0.166	-0.044	0.253	
CHAS	3.3925	1.030	3.295	0.001	1.368	5.417	
NOX	-21.3887	4.662	-4.588	0.000	-30.554	-12.223	
RM	3.3497	0.506	6.621	0.000	2.355	4.344	
AGE	0.0052	0.015	0.338	0.735	-0.025	0.036	
DIS	-1.6281	0.245	-6.649	0.000	-2.110	-1.147	
RAD	0.3141	0.079	3.968	0.000	0.158	0.470	
TAX	-0.0100	0.004	-2.270	0.024	-0.019	-0.001	
PTRATIO	-1.1411	0.161	-7.101	0.000	-1.457	-0.825	
B	0.0123	0.003	3.692	0.000	0.006	0.019	
LSTAT	-0.6015	0.059	-10.237	0.000	-0.717	-0.486	
Omnibus:	142.270	Durbin-Watson:	1.953				
Prob(Omnibus):	0.000	Jarque-Bera (JB):	568.640				
Skew:	1.522	Prob(JB):	3.32e-124				
Kurtosis:	7.951	Cond. No.	1.59e+04				

## K-fold cross validation

- Train 데이터를 K 개의 데이터 뭉치로 나눈다.
- (K-1)개의 Train 데이터 뭉치로 모델 만든다.
- K번째 데이터 뭉치로 검증한다.
- 이 과정을 K 번 반복한다.

- 최종 Error cost 계산.



## K-fold cross validation

```
from sklearn.model_selection import KFold
import statsmodels.api as sm
from sklearn.datasets import load_boston
import pandas as pd
import numpy as np

#K-fold split
scores = []
cv = KFold(10, shuffle=True, random_state=0) #크래치 생성
```

```
for i, (idx_train, idx_test) in enumerate(cv.split(df)):
    df_train = df.iloc[idx_train]
    df_test = df.iloc[idx_test]
```

```
model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.feature_names), data=df_train) #model
result = model.fit()
```

```
pred = result.predict(df_test)
rss = ((df_test.MEDV - pred) ** 2).sum()
tss = ((df_test.MEDV - df_test.MEDV.mean()) ** 2).sum()
rsquared = 1 - rss / tss
```

```
scores = np.append(scores, rsquared)
print(i, "번 째 Fold", "학습 R square 값 = {:.8f}, 검증 R square 값 = {:.8f}".format(result.rsquared, rsquared))
```

0 번째 Fold 학습 R square 값 = 0.76292630, 검증 R square 값 = 0.51496621  
1 번째 Fold 학습 R square 값 = 0.74874971, 검증 R square 값 = 0.63966634  
2 번째 Fold 학습 R square 값 = 0.73384697, 검증 R square 값 = 0.79072521  
3 번째 Fold 학습 R square 값 = 0.73622538, 검증 R square 값 = 0.77004336  
4 번째 Fold 학습 R square 값 = 0.75165392, 검증 R square 값 = 0.53405472  
5 번째 Fold 학습 R square 값 = 0.73733265, 검증 R square 값 = 0.76694329  
6 번째 Fold 학습 R square 값 = 0.73598053, 검증 R square 값 = 0.77290348  
7 번째 Fold 학습 R square 값 = 0.75823881, 검증 R square 값 = 0.58757414  
8 번째 Fold 학습 R square 값 = 0.71555404, 검증 R square 값 = 0.85613019  
9 번째 Fold 학습 R square 값 = 0.73522808, 검증 R square 값 = 0.79378366

# 데이터 처리: Preprocessing

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

---

데이터 처리: Preprocessing

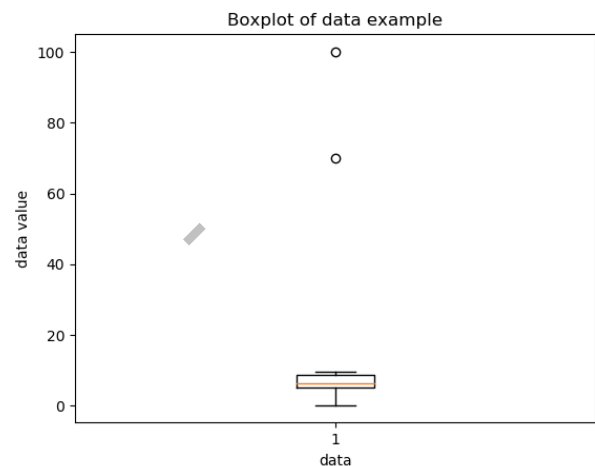
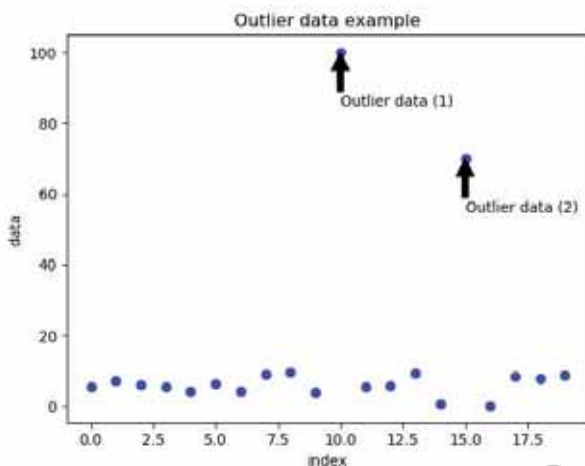
---

## 개요

1. Outlier 데이터 처리
2. Normalize (정규화)

# Outlier 데이터 처리

- 기존의 trend와 맞지 않는 데이터
- 모델 업데이트의 시작
- 해석하기 나름



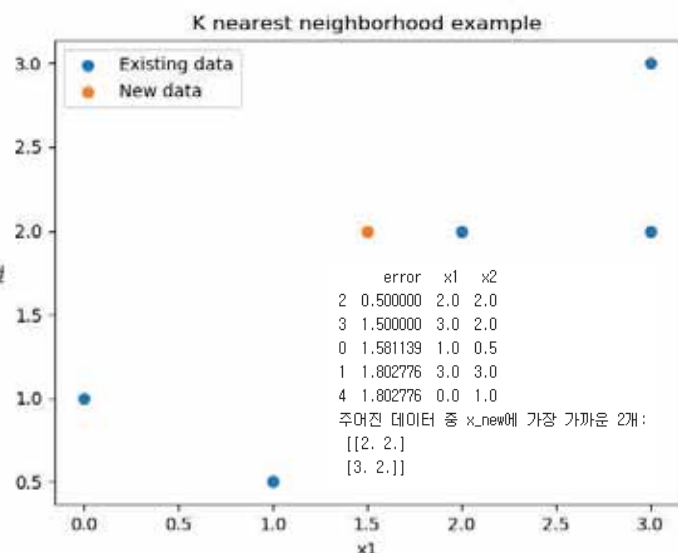
# Outlier 데이터 처리

- KNN (K-Nearest Neighbor) algorithm
- 새로 들어오는 데이터와 가장 유사한 상위 K 개의 벡터를 뽑는다.

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

def distance_sort(x_new, x_data):
    error=np.array(x_new-x_data)
    error_norm=np.linalg.norm(error,axis=1)
    x_data_new_pandas=pd.DataFrame({'error':error_norm,
                                    'x1':x_data[:,0],
                                    'x2':x_data[:,1]})
    x_data_sort=x_data_new_pandas.sort_values(by='error')
    return x_data_sort

def KNN_example(x_new,x_data,K):
    x_data_sort=distance_sort(x_new,x_data)
    K_nearest_x_data_pandas=x_data_sort.head(K)
    K_nearest_x_data=K_nearest_x_data_pandas[["x1", "x2"]].values
    return K_nearest_x_data
```



# Normalize (정규화)

- 학습을 더 효율적으로

$$X_n = \frac{X - \text{mean}(X)}{\text{std}(X)}$$

$$\text{mean}(X) = \frac{1}{N} \sum_{i=1}^N X_i$$

$$\text{std}(X) = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (X_i - \text{mean}(X))^2} = \sqrt{\text{var}(X)}$$

- 특성:  $\text{mean}(X_n) = 0, \text{std}(X_n) = 1$

# Normalize (정규화)

```
#Data normalize
#X_ = (X-mean(X))/std(X)
#mean(X_)=0
#std(X_)=1

import pandas as pd
import numpy as np
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler

#데이터셋 예제 데이터 생성 (바꿔야 할, pandas로 엑셀 파일 읽어오는 연습 함)
X=np.array([[1, 3, 5, 7],
            [2, 1, 4, 0],
            [5, 1, 3, 3]])

#데이터 표준화 (Normalize: mean=0, standard deviation = 1)
X_ = StandardScaler().fit_transform(X)
print("Raw data: \n", X)
print("Mean value of X's columns: \n", np.mean(X,axis=0)) #column 방향으로의 평균
print("Standard deviation value of X's columns: \n", np.std(X,axis=0)) #column 방향으로의 표준편차

print("Normalized data: \n", X_)
print("Mean value of X_: \n", np.mean(X_,axis=0)) #column 방향으로의 평균
print("Standard deviation value of X_'s columns: \n", np.std(X_,axis=0)) #column 방향으로의 표준편차
```

➡

Raw data:

```
[[1 3 5 7]
 [2 1 4 0]
 [5 1 3 3]]
```

Mean value of X's columns:

```
[2.66666667 1.66666667 4.          3.33333333]
```

Standard deviation value of X's columns:

```
[1.69967317 0.94280904 0.81649658 2.86744176]
```

Normalized data:

```
[[-0.98058068  1.41421356  1.22474487  1.27872403]
 [-0.39223227 -0.70710678  0.          -1.16247639]
 [ 1.37281295 -0.70710678 -1.22474487 -0.11624764]]
```

Mean value of X\_:

```
[ 7.40148683e-17 -7.40148683e-17  0.00000000e+00 -6.01370805e-17]
```

Standard deviation value of X\_'s columns:

```
[1. 1. 1. 1.]
```

# 데이터 처리: Data reduction by PCA, PLS

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

---

데이터 처리: Data reduction by PCA, PLS

## 개요

1. Data reduction 개념
2. Principal component analysis (PCA)
3. Partial least squares (PLS)

## Data reduction 개념

- 데이터는 정보를 가지고 있음
- ‘거의’ 동일한 정보를 가지고 있는 두 데이터
- 크기가 매우 다르다면 작은 데이터를 선택하면 효율 좋음
- 필터링 효과

## Principal component analysis (PCA)

- X끼리의 variance 최대한 보존하면서 차원축소

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

#데이터 표준화 (Normalize)

```
X_ = StandardScaler().fit_transform(X)
```

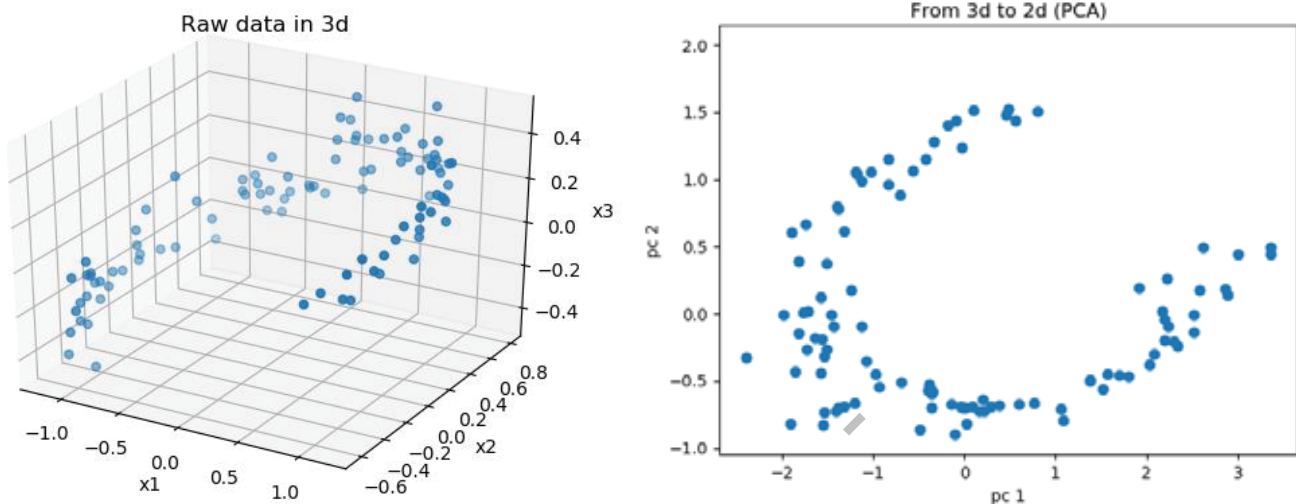
```
pca_result2 = PCA(n_components=2) #객체 생성
pca_result2.fit(X_) #실행
print("PCA 결과 각 요소가 차지하는 variance 비율: ",
      pca_result2.explained_variance_ratio_)
X_pca = pca_result2.transform(X_)
print(X_pca)
print(pca_result2.components_)
```



PCA 결과 각 요소가 차지하는 variance 비율:  
[0.77882381 0.16772713]



# Principal component analysis (PCA)



# Partial least squares (PLS)

- X와 Y의 covariance 최대화

```
from sklearn.cross_decomposition import PLSRegression
```

```
#PLS
pls2 = PLSRegression(n_components=2) #객체 생성
pls2.fit(X, Y) #PLS로 loading, score, weight and coefficient matrix
print("X block loading matrix: ", pls2.x_loadings_)
print("X block score matrix: ", pls2.x_scores_)
print("X block weights matrix: ", pls2.x_weights_)
print("-----")
print("Y block loading matrix: ", pls2.y_loadings_)
print("Y block score matrix: ", pls2.y_scores_)
print("Y block weights matrix: ", pls2.y_weights_)
print("-----")
print("The coefficient matrix (Y_pls = X*coeff): ", pls2.coef_)

Y_pred = pls2.predict(X)
print(Y_pred)
```

```
X block loading matrix: [[ 0.53214779  0.95309339]
 [ 0.60988553 -0.28245141]
 [ 0.58738169 -0.54861287]]
X block score matrix: [[-1.39700475 -0.56509355]
 [-1.19678754  0.34913038]
 [ 0.56032252  0.55035779]
 [ 2.03346977 -0.33439462]]
X block weights matrix: [[ 0.52188779  0.81112973]
 [ 0.617029  -0.57267793]
 [ 0.58898926 -0.11877943]]
```

```
Y block loading matrix: [[ 0.61778462 -0.06611187]
 [ 0.61520881 -0.07612656]]
Y block score matrix: [[-1.40620988  0.09323305]
 [-1.10143324 -0.85118366]
 [ 0.37266921  1.64933181]
 [ 2.13497391 -0.8913812 ]]
```

```
Y block weights matrix: [[ 0.61778462 -0.06611187]
 [ 0.61520881 -0.07612656]]
```

```
The coefficient matrix (Y_pls = X*coeff): [[1.47187317 1.47090503]
 [2.29380865 2.39915766]
 [2.03491986 2.10573249]
 [ 0.26087869  0.15302213]
 [ 0.60667302  0.45634164]
 [ 6.46856199  6.48931562]
 [11.7638863 12.00132061]]
```

# 최적화 기초: 개념 설명

2023.01

Engineering Development Research Center (EDRC)

정 동 휘

최적화 기초: 개념 설명

## 개요

1. 최적화 개념
2. 응용 분야
3. 모델식 없을 때 vs 모델식 있을 때
4. 문제 구성 요소
5. 풀이 구성 요소
6. 해답 구성 요소
7. 최적화 분류

## 최적화 개념

- 원하는 범위 내에서 수학적으로 가장 좋은 해를 찾는 것.
- 수학적으로 가장 좋은: Objective function (min/max)
- 해: Optimal solution
- 찾는 것: 최적화 알고리즘
- 원하는 범위: 제약 조건 (Constraint)

## 응용 분야

- 새로운 데이터를 이용한 모델 업데이트
- 최적 operation input design
- 최적 설계
- 스케줄링

## 모델식 없을 때 vs 있을 때

- 모델식 없어도 최적화 가능
- 모델식 있을 때의 장점 (= 모델식 없을 때의 단점)
- 모델식 활용한 최적화에서 주의할 점

## 문제 구성 요소

$$\min_u \text{obj}(x, u, p)$$

$$\text{subject to } G(x, u, p) \leq 0$$

$$lb_x \leq x \leq ub_x$$

$$lb_u \leq u \leq ub_u$$

예시

$$\min_u (x - 100)^2 + (u + 20)^2$$

$$\text{s.t. } x + u - 100 \leq 0$$

$$u^2 - 20 \leq 0$$

$$10 \leq x \leq 150$$

$$-50 \leq u \leq 50$$

## 풀이 구성 요소

- 알고리즘 종류 선택
- 알고리즘에 따라 시작점 등 하이퍼 파라미터 선택

예시

$$\min_u (x - 100)^2 + (u + 20)^2$$

$$\text{s.t. } x + u - 100 \leq 0$$

$$u^2 - 20 \leq 0$$

$$10 \leq x \leq 150$$

$$-50 \leq u \leq 50$$

```
from scipy.optimize import minimize

#Convex objective function model
def obj1(x):
    return (x[0]-100)**2 + (x[1]+20)**2

#Convex nonlinear constraint (o(x)>=0)
def Nonlinear_con1(x):
    return [x[0] + x[1] - 100, x[1]**2 - 20]

bnds = ((10,150),(-50,50))

ineq_cons = {'type': 'ineq',
             'fun': Nonlinear_con1}

#초기값 (Local optimizing algorithm)으로
initial = np.array([100, 100], dtype='float64')

result_1 = minimize(obj1, initial, method='SLSQP', bounds=None, constraints=ineq_cons)
```

## 해답 구성 요소

- 최적 input, state
- 최적 objective, constraint function value

예시

$$\min_u (x - 100)^2 + (u + 20)^2$$

$$\text{s.t. } x + u - 100 \leq 0$$

$$u^2 - 20 \leq 0$$

$$10 \leq x \leq 150$$

$$-50 \leq u \leq 50$$

```
print("Optimal state: %n", result_1.x[0]), print("_____")
print("Optimal input: %n", result_1.x[1]), print("_____")
print("Optimal output: %n", result_1.fun), print("_____")
print("Constraint output: %n", Nonlinear_con1(result_1.x))
```

Optimal state:

100.0

Optimal input:

4.47213595369085

Optimal output:

598.8854381359283

Constraint output:

[4.472135953690852, -1.1705633795600079e-08]

## 최적화 분류

- LP: Linear
- QP: Quadratic
- NLP: Nonlinear
- IP: Integer
- 최종적으로 혼합된 MINLP

## 최적화 기초: Derivative-free

---

2023.01

Engineering Development Research Center (EDRC)

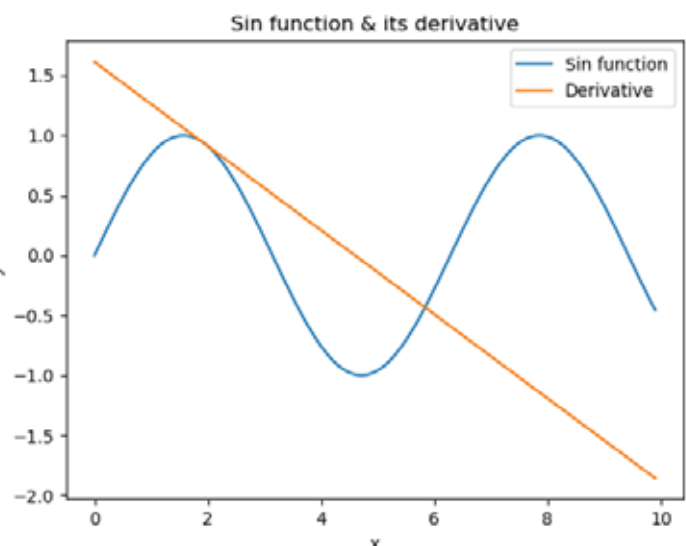
정동휘

## 개요

1. Derivative 정보란?
2. Derivative-free 최적화 알고리즘 개념
3. Scipy 패키지 활용
4. Nelder-Mead (amoeba) method
5. Differential evolution algorithm

## Derivative 정보란?

- 쉽게 생각하면 기울기
- 조작 변수를 unit size 만큼 움직였을 때의 종속 변수의 변화량
- 미분
- 다변수일 때: Gradient



## Derivative-free 최적화 알고리즘 개념

- 데이터를 뿌려서 각각의 경우에 objective 값을 확인한다.
- 이 정보를 바탕으로 다음 데이터 뿌릴 경우를 계산한다.
- 위 과정을 반복한다.

## Scipy 패키지 활용

### • 설치 방법 1

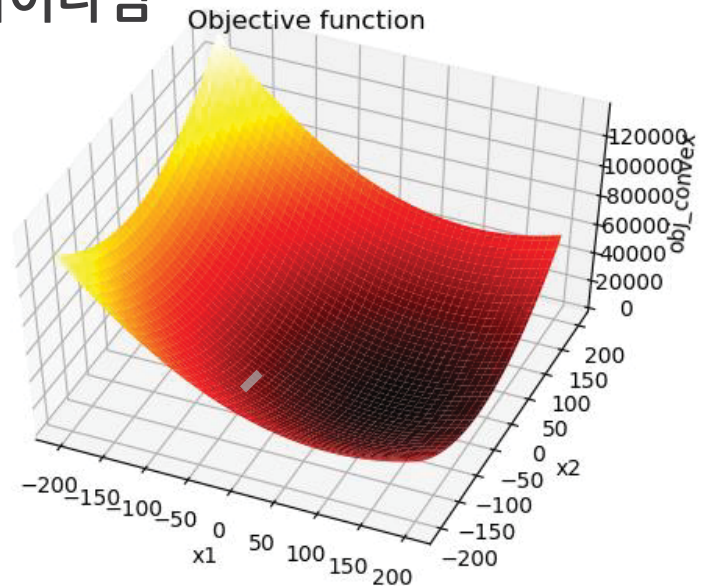
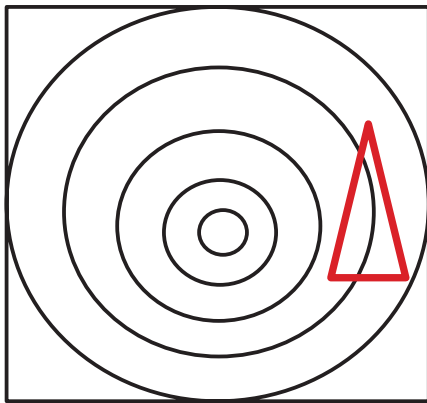
The screenshot shows the PyCharm IDE interface. On the left, the 'File' menu is open, and 'Settings...' is selected. The 'Settings' dialog is open, showing the 'Project: Practice\_1' settings. The 'Project Interpreter' tab is selected, showing a list of installed packages. The 'scipy' package is highlighted, and a red box is drawn around the '+' icon in the rightmost column, indicating the option to add or update the package. Below the package list, there is a search bar with 'scipy' entered and an 'Install Package' button.

Package	Version	Latest version
certifi	2019.6.16	2019.6.16
pip	19.1.1	19.1.1
python	3.6.8	3.7.3
setuptools	41.0.1	41.0.1
sqlite	3.28.0	3.28.0
vc	14.1	14.1
vs2015_runtime	14.15.26706	14.15.26706
wheel	0.33.4	0.33.4
wincertstore	0.2	0.2



## Nelder-Mead (amoeba) method

- 원리: N 차원에서 N+1개의 점을 찍어서 다음 스텝
- 아메바처럼 N 차원을 기어다님



## Nelder-Mead (amoeba) method

```
import numpy as np
from scipy.optimize import minimize
from Optimization_objective_1 import obj_convex
import matplotlib.pyplot as plt
```

*#초기값 (Local optimizing algorithm이므로)*

```
initial = np.array([0, 0])
```

*#최적화*

```
result1 = minimize(obj_convex, initial,
                   method='nelder-mead', #simplex)
```

*#결과*

```
print("종합 정보: %n", result1)
print("Optimal input: %n", result1.x)
print("Optimal output: %n", result1.fun)
```



종합 정보:

```
final_simplex: (array([[100.00002526, -20.00000997],
 [ 99.99993935, -19.9999796 ],
 [ 99.99995474, -20.00007598]]), array([[7.37323265e-10, 4.09456540e-09, 7.82168395e-09]]))
fun: 7.373232649554163e-10
message: 'Optimization terminated successfully.'
nfev: 216
nit: 112
status: 0
success: True
x: array([100.00002526, -20.00000997])
Optimal input:
[100.00002526 -20.00000997]
Optimal output:
7.373232649554163e-10
```

# Differential evolution algorithm

- 원리: 유전 알고리즘
- “최후에 살아남는 좋은 변화에 가장 잘 적응한 종”
- 과정: 초기화-선택-교차-변이-대치-반복

# Differential evolution algorithm

```
import numpy as np
from scipy.optimize import differential_evolution
from Optimization_objective_1 import obj_convex

#최적화 1 (bound 없을 때)
bounds1=[(-10,10),(-10,10)] #boundary
result_1 = differential_evolution(obj_convex, bounds1)

#결과
print("종합 정보: \n", result_1), print("-----")
print("Optimal input: \n", result_1.x), print("-----")
print("Optimal output:\n", result_1.fun), print("-----")

#최적화 2 (bound 있을 때)
bounds2 = [(-200,200), (-200, 200)] #boundary
result_2 = differential_evolution(obj_convex, bounds2)
print("Optimal input: \n", result_2.x)
print("Optimal output:\n", result_2.fun)
```

종합 정보:

```
fun: 8200.0
jac: array([-180.00027922,  19.99997039])
message: 'Optimization terminated successfully.'
nfev: 426
nit: 13
success: True
x: array([ 10., -10.])
```

-----

Optimal input:  
[ 10. -10.]

-----

Optimal output:  
8200.0

-----

Optimal input:  
[100. -20.]

-----

Optimal output:  
1.262177448353619e-29

# 최적화 기초: Derivative-based

---

2023.01

Engineering Development Research Center (EDRC)

정 동 휘

---

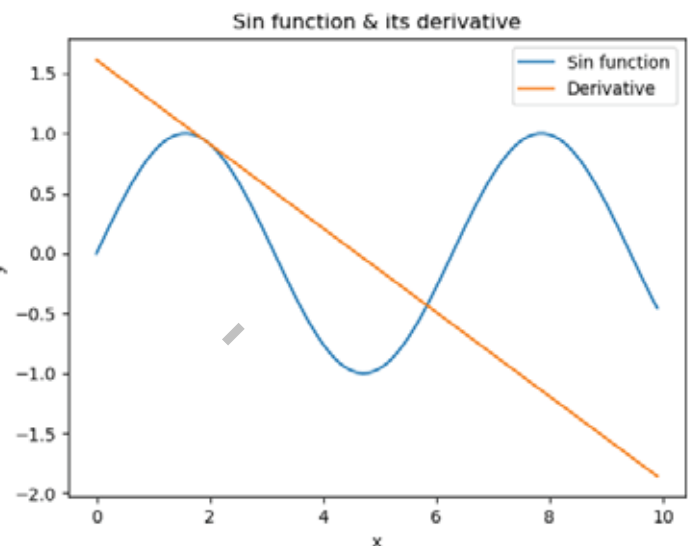
최적화 기초: Derivative-based

## 개요

1. Derivative 정보란?
2. Derivative-based 최적화 알고리즘 개념
3. Gradient descent method
4. Line search
5. (Quasi-) Newton method
6. Trust region

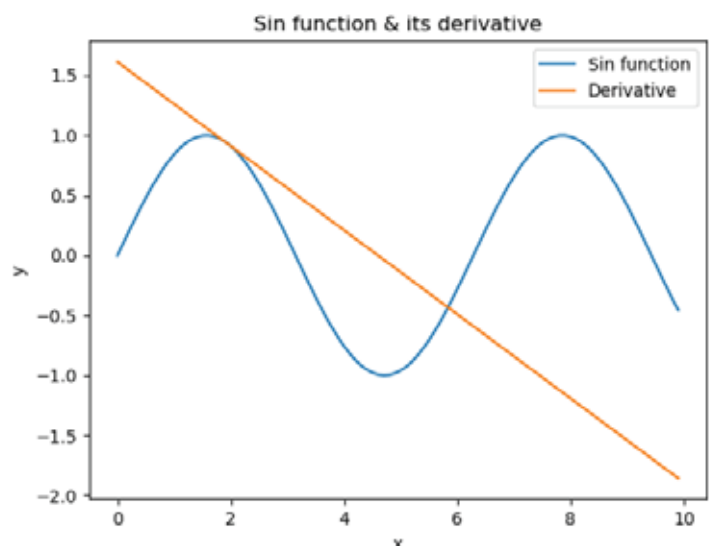
## Derivative 정보란?

- 쉽게 생각하면 기울기
- 조작 변수를 unit size 만큼 움직였을 때의 종속 변수의 변화량
- 미분
- 다변수일 때: Gradient
- 부호: 방향



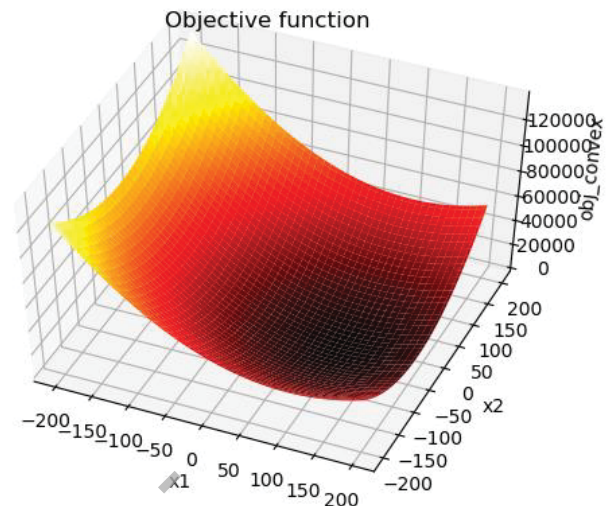
## Derivative-based 최적화 알고리즘 개념

- 개념: 줄어드는 (늘어나는) 방향으로 업데이트 계속
- 결국에는 골짜기 (혹은 정상)에 도달
- 업데이트의 방향
- 업데이트의 크기



# Gradient descent algorithm

- 1차 근사
- $u_{i+1} = u_i - a * \frac{\partial(Obj)}{\partial u} |_{u_i}$
- 업데이트의 방향:  $\frac{\partial(Obj)}{\partial u} |_{u_i}$
- 업데이트의 크기:  $a$



```
#Derivative of the objective function (Numerically)
def derivative_obj1(x):
    h = 10**(-5)
    der = np.zeros_like(x)
    der[0] = (obj_convex([x[0]+h, x[1]])-obj_convex([x[0]-h, x[1]]))/2/h
    der[1] = (obj_convex([x[0], x[1]+h])-obj_convex([x[0], x[1]-h]))/2/h
    return der
```

# Gradient descent algorithm

```
#초기값 (Local optimizing algorithm)으로
initial = np.array([0, 0])
```

```
#Gradient descent method로 최적화
x=np.array([[2,1]])
obj_result=np.array(obj_convex(x[0,:]))
step_size=0.001
```

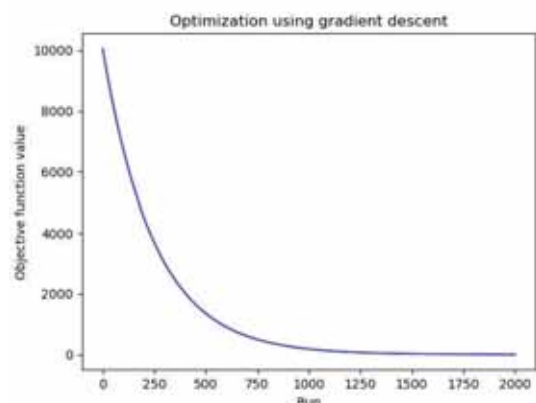
```
for i in range(2001):
    new_input = x[i,: ]
    new_input = new_input - step_size*derivative_obj1(new_input)
    x=np.vstack((x, new_input))
    obj_result=np.vstack((obj_result,obj_convex(new_input)))

print("Gradient descent method에 따라 계산되는 input 값: \n", x)

plt.figure(1)
plt.plot(obj_result, 'b-')
plt.xlabel('Run')
plt.ylabel('Objective function value')
plt.title('Optimization using gradient descent')
plt.show()
```

Gradient descent method에 따라 계산되는 input 값:

```
[[ 2.         1.         ]
 [ 2.196      0.958      ]
 [ 2.391608   0.916084   ]
 ...
 [ 98.20865965 -19.61614135]
 [ 98.21224233 -19.61690907]
 [ 98.21581785 -19.61767525]]
```



## Line search

- $u_{i+1} = u_i - a * \frac{\partial(Obj)}{\partial u} |_{u_i}$
- 업데이트의 크기:  $a$
- Gradient descent의 문제: 너무 느리다.
- Line search:  $a$  를 일단 무조건 크게 해주고 나중에 작게 만들어서 빠른 업데이트를 해보자.

## (Quasi-) Newton method

- 2차 근사
- $u_{i+1} = u_i - \left( \frac{\partial^2(Obj)}{\partial u^2} |_{u_i} \right)^{-1} * \frac{\partial(Obj)}{\partial u} |_{u_i}$
- 업데이트: 2차 근사의 극점

```
import numpy as np
from scipy.optimize import minimize
from Optimization_objective_1 import obj_convex

result_2 = minimize(obj_convex, initial, method='BFGS', options={'disp':True})
print("종합 정보: %n", result_2)
print("Optimal input: %n", result_2.x)
print("Optimal output: %n", result_2.fun)
```

Optimization terminated successfully.  
 Current function value: 0.000000  
 Iterations: 4  
 Function evaluations: 32  
 Gradient evaluations: 8

종합 정보:

fun: 5.2154444178801176e-12  
 hess\_inv: array([[0.51922895, 0.09614947],  
 [0.09614947, 0.98077105]])  
 jac: array([ 4.49303766e-06, -8.84026605e-07])  
 message: 'Optimization terminated successfully.'  
 nfev: 32  
 nit: 4  
 njev: 8  
 status: 0  
 success: True  
 x: array([100.00000224, -20.00000045])

Optimal input:  
 [100.00000224 -20.00000045]  
 Optimal output:  
 5.2154444178801176e-12

## Trust region

- $u_{i+1} = u_i - \left( \frac{\partial^2(Obj)}{\partial u^2} \Big|_{u_i} \right)^{-1} * \frac{\partial(Obj)}{\partial u} \Big|_{u_i}$
- (Quasi-) Newton method의 단점: 불안정
- Trust region: 극점으로 업데이트 하지 않고 일정 범위 내에서만 업데이트 한다.

## 최적화 기초: 제약 조건 있을 때

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

## 개요

1. 제약 조건 (Constraint) 개념
2. 응용 분야
3. 풀이 방법
4. 예시

## 제약 조건 (Constraint) 개념

- (In)equality:  $G(x, u, p) \leq 0$

예시

$$\min_u (x - 100)^2 + (u + 20)^2$$

$$\text{s.t. } x + u - 100 \leq 0$$

$$u^2 - 20 \leq 0$$

$$10 \leq x \leq 150$$

$$-50 \leq u \leq 50$$

- Bounds:  $lb_x \leq x \leq ub_x$

- 실제로는 feasible 해를 잘 못 찾을 수 있기 때문에 soft constraint 개념을 도입하기도 한다.



## 응용 분야

- “대부분의” 화학/생물 공정
- 모델링의 이유
- 예시: First principle model (ODE)로 표현된 모델이 있을 때 Productivity를 최대화하는 input을 계산

## 풀이 방법

- 제약 조건 없을 때: 기존에 배웠던 알고리즘들 사용
- 제약 조건 있을 때: 제약 조건 없을 때와 비슷하게 형태를 만들어서 제약 조건 없을 때처럼 풀이
- 제약 조건의 비선형성 등은 국소 근사 함수 적용
- 국소 근사 함수: (해당 영역에서) LP 혹은 QP
- 그러므로 빠른 해 찾기 가능

## 예시 1

### • Linear constraint 가지는 최적화 문제

```
from scipy.optimize import minimize
from scipy.optimize import BFGS
from scipy.optimize import SR1
from scipy.optimize import LinearConstraint
from scipy.optimize import NonlinearConstraint
from scipy.optimize import Bounds
#Convex objective function model
def obj1(x):
    return (x[0]-100)**2 + (x[1]+20)**2

#Convex linear constraint
linear_con = LinearConstraint([[1,0], [0,1]], [-100000, -10000], [10000, 10000])

#초기값 (Local optimizing algorithm)으로
initial = np.array([100, 100], dtype='float64')

#최적화 1 using Trust-Region Constrained Algorithm (Linear constraint만 있을 때)
result_1 = minimize(obj1, initial, method='trust-constr', jac="2-point", hess=SR1(),
                    constraints=[linear_con],
                    options={'verbose': 1}, bounds=None)
```

(Convex obj, 선형 con) Optimal input:  
[ 99.99999925 -19.99999985]

(Convex obj, 선형 con) Optimal output:  
5.777504171800091e-13

## 예시 2

### • Nonlinear constraint 가지는 최적화 문제

```
from scipy.optimize import minimize
from scipy.optimize import BFGS
from scipy.optimize import SR1
from scipy.optimize import LinearConstraint
from scipy.optimize import NonlinearConstraint
from scipy.optimize import Bounds
#Convex objective function model
def obj1(x):
    return (x[0]-100)**2 + (x[1]+20)**2

#Convex linear constraint
linear_con = LinearConstraint([[1,0], [0,1]], [-100000, -10000], [10000, 10000])

nonlinear_con = NonlinearConstraint(Nonlinear_con1, -np.inf, 0, jac='2-point', hess=BFGS())

result_2 = minimize(obj1, initial, method='trust-constr', jac="2-point", hess=SR1(),
                    constraints=[nonlinear_con],
                    options={'verbose': 1}, bounds=None)
```

(Convex obj, 비선형 convex con) Optimal input:  
[99.99999926 4.47213596]

(Convex obj, 비선형 convex con) Optimal output:  
598.8854382511838

## 예시 3

### • Nonconvex objective 가지는 최적화 문제

```
import numpy as np
from scipy.optimize import minimize
from scipy.optimize import BFGS
from scipy.optimize import SR1
from scipy.optimize import LinearConstraint
from scipy.optimize import NonlinearConstraint
from scipy.optimize import Bounds
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#Nonconvex objective function model
def obj2(x):
    return -(x[1] + 20) * np.sin(np.sqrt(abs(x[0]/2 + (x[1] + 20)))) - x[0] * np.sin(np.sqrt(abs(x[0] - (x[1] + 20))))

#Bounds
bounds = Bounds([-np.inf, -10000], [np.inf, 10000])

#Convex linear constraint
linear_con = LinearConstraint([[1,0], [0,1]], [-100000, -10000], [10000, 10000])

#Convex nonlinear constraint (lb<=o(x)<=ub)
def Nonlinear_con1(x):
    return [-x[0] - x[1] + 100., -x[1]**2 + 20.]

nonlinear_con = NonlinearConstraint(Nonlinear_con1, -np.inf, 0, jac='2-point', hess=BFGS())
```

## 예시 3

### • Nonconvex objective 가지는 최적화 문제

```
#최적화 1 using Trust-Region Constrained Algorithm + starting point: initial_1
initial_1 = np.array([100, 100], dtype='float64') #초기값 (Local optimizing algorithm)으로
result_1 = minimize(obj2, initial_1, method='trust-constr', jac="2-point", hess=SR1(),
                    constraints=[linear_con, nonlinear_con],
                    options={'verbose': 1}, bounds=None)
print("Optimal input (starting point is initial_1): %n", result_1.x)
print("Optimal output (starting point is initial_1): %n", result_1.fun)
```

Optimal input (starting point is initial\_1):  
[180.8233012 98.65760174]

Optimal output (starting point is initial\_1):  
-293.29583236112785

```
#최적화 2 using Trust-Region Constrained Algorithm + starting point: initial_2
initial_2 = np.array([50, 200], dtype='float64') #초기값 (Local optimizing algorithm)으로
result_2 = minimize(obj2, initial_2, method='trust-constr', jac="2-point", hess=SR1(),
                    constraints=[linear_con, nonlinear_con],
                    options={'verbose': 1}, bounds=None)
print("Optimal input (starting point is initial_2): %n", result_2.x)
print("Optimal output (starting point is initial_2): %n", result_2.fun)
```

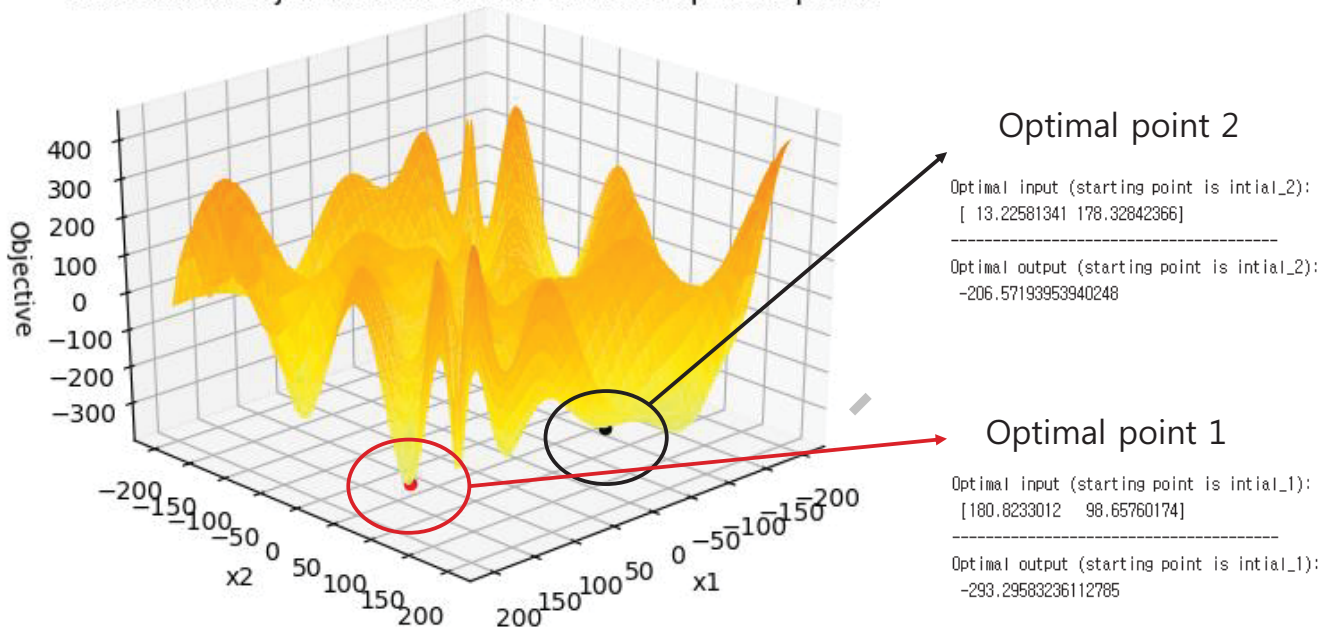
Optimal input (starting point is initial\_2):  
[13.22581341 178.32842366]

Optimal output (starting point is initial\_2):  
-206.57193953940248

## 예시 3

- Nonconvex objective 가지는 최적화 문제

Nonconvex objective function & its local optimal point



# 최적화 심화: 파라미터 추정

2023.01

Engineering Development Research Center (EDRC)

정동휘

## 개요

1. 파라미터란?
2. 응용 분야
3. 파라미터 추정 방법: “배웠던” 최적화 기법들 적용
4. White-box model 파라미터 추정
5. Black-box model 파라미터 추정
6. 꿀팁

## 파라미터란?

- 모델 형태 정하면 데이터를 통해서 구해야 하는 값
- 응용 분야: 모델의 완성

## 파라미터 추정 방법

- 앞서 배웠던 여러 가지 최적화 기법 응용
- 제약 조건 가지는 최적화
- 일반적으로 Gradient based local optimization
- Global approach 써도 됨 (initial point 찾기 용도)

$$\min_p \text{Cost}_{\text{Error}}(y_{\text{data}} - y_{\text{model}})$$

$$\text{s.t. } G_{\text{model}}(x, p) = 0$$

$$y_{\text{model}} = f(x, p)$$

## White-box model 파라미터 추정

- ODE fitting

```
#ODE 함수의 parameter fitting
#local optimizing algorithm (quasi-Newton method)

#Ordinary differential equation (ODE) 풀이 기법

from scipy.optimize import minimize
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

#ODE example 2
def ode_model_3(x, t, a, p):
    dxdt = -a*x + p
    return dxdt

def ode_model_3_value(x_initial, start_time, end_time, a, p_1):
    time = np.linspace(start_time, end_time)
    x = odeint(ode_model_3, x_initial, time, args=(a, p_1))
    return x
```

```
#Cost (실험 결과 모델 추정 값의 차이 제곱 합)
x_data = np.array([0.1, 0.4, 0.42, 0.45, 0.49, 0.50]) #Data
def cost(p, x_data):
    a = p[0]
    p_1 = p[1] #parameters to be estimated

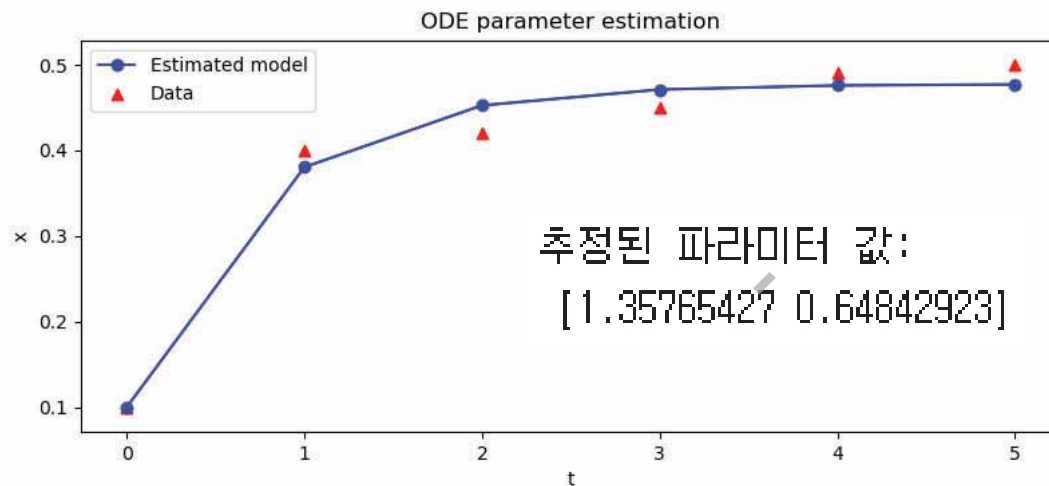
    x_ = [0.1] # initial
    time = [0]
    for i in range(5):
        new_start_time = i
        new_end_time = i + 1
        time = np.append(time, i + 1)
        x_result = ode_model_3_value(x_[-1], new_start_time, new_end_time, a, p_1)
        x_ = np.append(x_, x_result[-1])

    Cost=0
    for j in range(len(x_)):
        err_ = x_data[j] - x_[j]
        Cost = Cost + err**2
    Cost = Cost/len(x_)
    return Cost
```

# White-box model 파라미터 추정

## • ODE fitting

```
#Optimization for parameter estimation (Using local optimizing algorithm)
initial=[0.1,0.1]
result = minimize(cost, initial, method='BFGS', args=(x_data), options={'disp':True})
print("추정된 파라미터 값:", result.x)
```



# White-box model 파라미터 추정

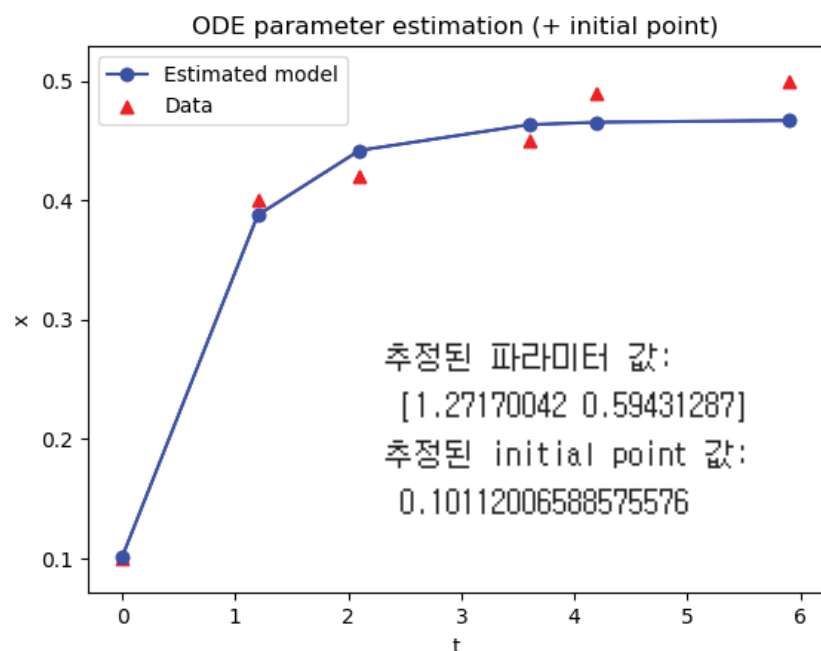
## • ODE fitting + initial point of x

```
#Cost (실험 값과 모델 추정 값의 차이 제곱 합)
```

```
def cost(p,x_data, t_data):
    # parameters to be estimated
    a = p[0]
    p_1 = p[1]
    x0 = p[2]

    x_ = [x0] # initial x value
    time = t_data[0] #initial time
    for i in range(len(t_data)-1):
        new_start_time = t_data[i]
        new_end_time = t_data[i + 1]
        time = np.append(time, t_data[i + 1])
        x_result = ode_model_3.value1(x_[-1],
                                     new_start_time,
                                     new_end_time,
                                     a, p_1)
        x_ = np.append(x_, x_result[-1])

    Cost=0
    for j in range(len(x_)-1):
        err = x_data[j] - x_[j]
        Cost = Cost + err**2
    Cost = Cost/len(x_)
    return Cost
```



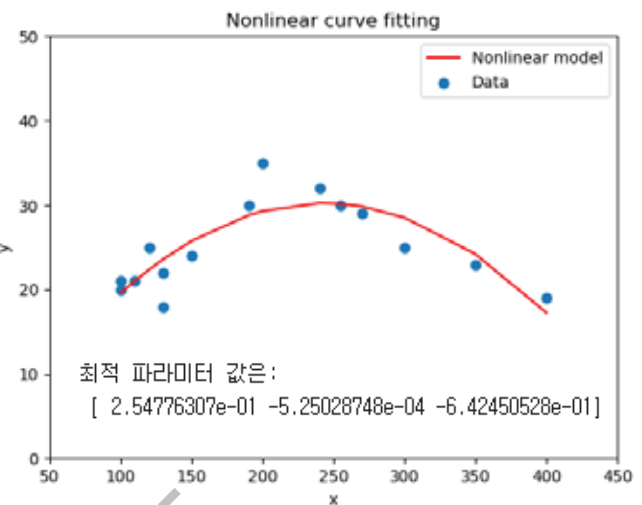
# Black-box model 파라미터 추정

## • Nonlinear model 1

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

#모델 형태 지정 2
def Nonlinear_model(x, coeff1, coeff2, bias):
    return coeff2*x**2 + coeff1*x_data + bias

#Data fitting 2
popt2, pcov2 = curve_fit(Nonlinear_model, x_data, y_data)
print(popt2) #fitting parameters
print(pcov2)
y_predict_nonlinear = Nonlinear_model(x_data, popt2[0], popt2[1], popt2[2])
```



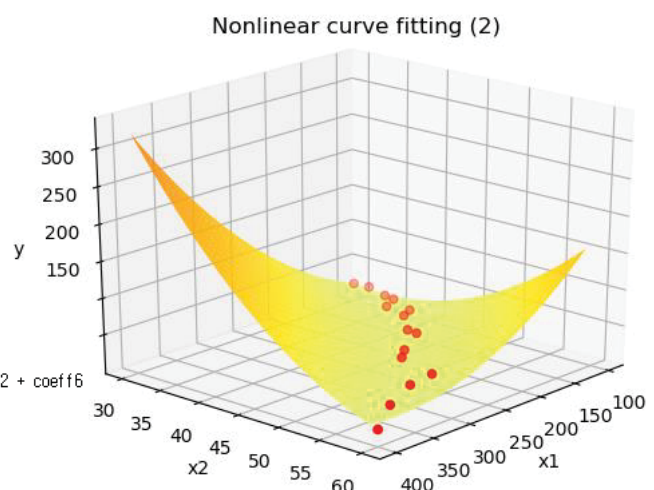
# Black-box model 파라미터 추정

## • Nonlinear model 2

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

#모델 형태 지정 2
def Nonlinear_model(X,coeff1, coeff2, coeff3, coeff4, coeff5, coeff6):
    x1, x2 = X
    return coeff1*x1**2 + coeff2*x2**2 + coeff3*x1*x2 + coeff4*x1 + coeff5*x2 + coeff6

#Data fitting 2
popt2, pcov2 = curve_fit(Nonlinear_model, (x1_data,x2_data), y_data)
print("최적 파라미터 값은: \n", popt2) #fitting parameters
y_predict_nonlinear = Nonlinear_model((x1_data, x2_data), popt2[0],
                                       popt2[1], popt2[2], popt2[3], popt2[4], popt2[5])
```



최적 파라미터 값은:  
[ 1.95826147e-03 2.19553466e-01 -4.95537587e-02 1.50285732e+00  
-9.98026867e+00 1.01711406e+02]



# Black-box model 파라미터 추정

- 시계열 데이터 분석 예제
- ARIMA model

```
from statsmodels.tsa.arima_model import ARIMA
import numpy as np
import matplotlib.pyplot as plt
```

```
data = np.array([1,3,4,6,7,9,11,15,16, 18, 24, 25,
                 30, 31, 32, 33, 28, 26, 22, 20,
                 19, 15, 14,13,9,8])

model = ARIMA(data, order=(0,1,1))
model_fit = model.fit(trend='no', full_output='True', disp=1)
print(model_fit.summary())
model_fit.plot_predict()
plt.show()
```



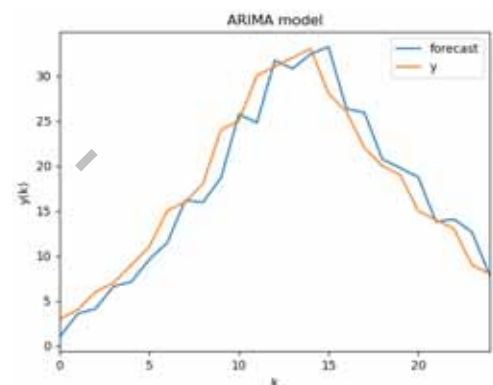
ARIMA Model Results

Dep. Variable:	D.y	No. Observations:	25
Model:	ARIMA(0, 1, 1)	Log Likelihood	-58.774
Method:	csm-ml	S.D. of innovations	2.534
Date:	Tue, 09 Jul 2019	AIC	121.547
Time:	16:20:30	BIC	123.985
Sample:	1	HQIC	122.223

	coef	std err	z	P> z	[0.025	0.975]
ma.L1.D.y	0.3235	0.171	1.887	0.071	-0.013	0.660

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-3.0909	+0.0000j	3.0909	0.5000



## 꿀팁

- 데이터 normalize 할 것
- 혹은 파라미터에 log 씌워서 찾기 (Order of magnitude가 아주 많이 차이나는 경우가 있으므로)
- 파라미터에 대하여 linear하게 생긴 경우 Shortcut method로 파라미터 초기화 할 것

# 최적화 심화: 최적 입력 설계

---

2023.01

Engineering Development Research Center (EDRC)

정동휘

---

최적화 심화: 최적 입력 설계

## 개요

1. 문제 정의
2. 응용 분야: 목표 함수 최적화 (Max or Min)
3. 방법: “배웠던” 최적화 기법들 적용
4. 예시

## 문제 정의

- 시스템의 동적 특성을 특정 ODE로 표현했을 때, 목표로 하는 값을 최적화하는 input을 찾는 것

$$\min_u \text{Obj}(x, u)$$

$$\text{s.t. } \frac{dx}{dt} = f(x, u)$$

$$lb_u \leq u \leq ub_u$$



실제 예제

$$\min_u -100 * x(t_f) + \sum u$$

$$\text{s.t. } \frac{dx}{dt} = -x + u$$

$$0 \leq u \leq 0.2$$

## 응용 분야

- 목표 함수 최적화 (최대화/최소화)
- 제약 조건 만족 (Feasibility)

$$\min_u \text{Obj}(x, u)$$

$$\text{s.t. } \frac{dx}{dt} = f(x, u)$$

$$lb_u \leq u \leq ub_u$$



실제 예제

$$\min_u -100 * x(t_f) + \sum u$$

$$\text{s.t. } \frac{dx}{dt} = -x + u$$

$$0 \leq u \leq 0.2$$

# 예시

## • Input 하나 일 때

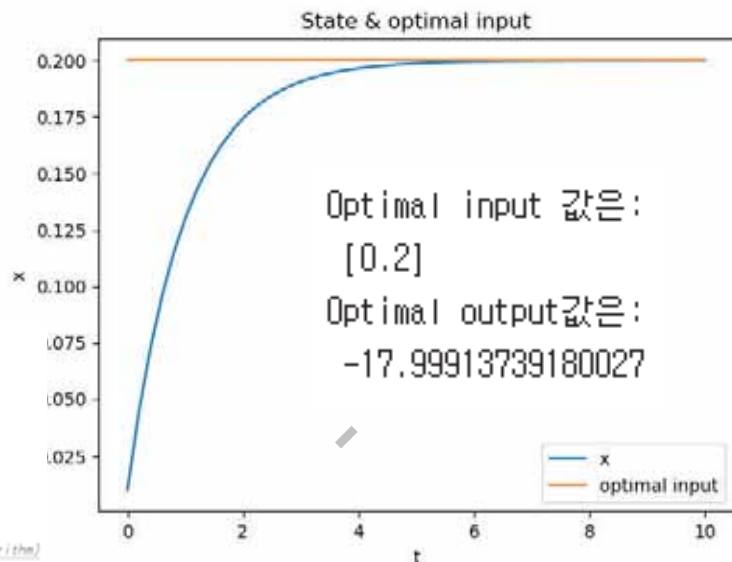
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# ODE example 2
def ode_model_3(x, t, u):
    dxdt = -x + u
    return dxdt

def ode_model_3_value(x_initial, start_time, end_time, u):
    time = np.linspace(start_time, end_time)
    x = odeint(ode_model_3, x_initial, time, args=(u,))
    return x

# Cost
def cost(u):
    t_terminal=10
    x_initial=0.01
    x_result = ode_model_3_value(x_initial, 0,
                                t_terminal, u)
    Cost = -100*x_result[-1] + u*t_terminal
    return Cost

# Optimization for parameter estimation (Using local optimizing algorithm)
initial=[0.1]
result_1 = minimize(cost, initial, method='SLSQP',
                    options={'disp':True}, bounds=((0,0.2),))
print("Optimal input 값은: %n", result_1.x)
print("Optimal output 값은: %n", result_1.fun)
```



# 예시

## • Input 여러 가지로 discretize 했을 때

```
# Cost
def cost(u):
    t_terminal=10
    disc_num=6
    t_data = np.linspace(0, t_terminal,num=disc_num)
    time=[0]
    x_=[0.01]
    input=[]
    for i in range(len(t_data) - 1):
        input=np.append(input,u[i])
        new_start_time = t_data[i]
        new_end_time = t_data[i + 1]
        time = np.append(time, t_data[i + 1])
        x_result = ode_model_3_value(x_[-1], new_start_time,
                                    new_end_time, input[i])
        x_ = np.append(x_, x_result[-1])
    Cost = -100*x_[-1] + np.sum(input)*t_terminal
    return Cost

# Optimization for parameter estimation (Using local optimizing algorithm)
initial=[0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
result_1 = minimize(cost, initial, method='SLSQP',
                    options={'disp':True}, bounds=((0,0.2), (0,0.2), (0,0.2), (0,0.2), (0,0.2)))
print("Optimal input 값은: %n", result_1.x)
print("Optimal output 값은: %n", result_1.fun)
```

Optimal input 값은:  
[9.76677073e-12 9.10063691e-12 7.82698906e-12 2.00000000e-01  
2.00000000e-01]  
Optimal output 값은:  
-15.63373105815717

