[EEC3600-001] 수치해석		
소속: 전기전자공학부	학번: 12191529	이름: 장준영
Term Project		**Prob #4**

1. Problem

a. 문제

Problem 4. [*Least-squares (programming)*] Consider the power conversion relation

$$P_{\text{out}} = P_{\text{in}} + P_{\text{loss}} = \eta_{\text{effc}} P_{\text{in}}$$

where $P_{\text{out}} = P_{\text{mech}}$ is the mechanical power delivered (for $P_{\text{mech}} > 0$) by a motor or the mechanical power supplied (for $P_{\text{mech}} < 0$) to a motor, and $P_{\text{in}} = P_{\text{elec}}$ is the electrical power supplied to a motor from a battery (for $P_{\text{elec}} > 0$) or generated by a motor and supplied to a battery (for $P_{\text{elec}} < 0$).

The power conversion efficiency, from electrical power to mechanical power when $P_{\text{elec}} > 0$ (traction mode) and from mechanical power to electrical power when $P_{\text{elec}} < 0$ (generator mode), $\eta_{\text{effc}} \in (0, 1)$ is considered to be a function of motor speed ω_m [rad/sec] and T_m [N·m]. In other words,

$$\eta_{\text{effc}} = g(\omega_m, T_m)$$

for some function $g : \mathbb{R} \times \mathbb{R} \to (0, 1)$.

Consider the data file

`motor_efficiency_data.mat`

There are two variables,

- `Data_regen_braking`$\in \mathbb{R}^{170\times 3}$ and
- `Data_traction`$\in \mathbb{R}^{180\times 3}$.

In the variables `Data_regen_braking` and `Data_traction`, the first column is the power conversion efficiency η_{effc}, the second column is the motor speed ω_m [rad/sec], and the third column is the motor torque T_m [N·m].

In this problem, we want to determine a (regression) model for P_{elec} as a function of (ω_m, T_m). In other words,

$$P_{\text{elec}} = f(\omega_m, T_m) = \begin{cases} \dfrac{T_m \omega_m}{\eta_{\text{effc}}} & \text{for } T_m > 0\,, \\ T_m \omega_m \eta_{\text{effc}} & \text{for } T_m < 0\,, \end{cases}$$

where $P_{\text{mech}} = T_m \omega_m$.

(a) Consider a parameterized linear regression with the following set of basis functions $\{b_k(\omega_m, T_m)\}_{k=1}^{7}$:

$$\{1, \omega_m, \omega_m^2, \omega_m T_m, T_m^2, \omega_m T_m^2, \omega_m^2 T_m^2\}\,.$$

Determine the parameters of approximation models

$$\hat{f}^+(\omega_m, T_m) = \sum_{k=1}^{7} c_k^+ b_k(\omega_m, T_m) \approx \frac{T_m \omega_m}{\eta_{\text{effc}}(\omega_m, T_m)} \quad \text{for } T_m > 0$$

and

$$\hat{f}^-(\omega_m, T_m) = \sum_{k=1}^{7} c_k^- b_k(\omega_m, T_m) \approx T_m \omega_m \eta_{\text{effc}}(\omega_m, T_m) \quad \text{for } T_m < 0\,,$$

via the least-squares method:

$$c^+ := \arg\min \sum_{i=1}^{N_s^+} \left| \hat{f}^+(\omega_m^{(i)}, T_m^{(i)}) - \frac{T_m^{(i)} \omega_m^{(i)}}{\eta_{\text{effc}}(\omega_m^{(i)}, T_m^{(i)})} \right|^2$$

and

$$c^- := \arg\min \sum_{i=1}^{N_s^-} \left| \hat{f}^-(\omega_m^{(i)}, T_m^{(i)}) - T_m^{(i)} \omega_m^{(i)} \eta_{\text{effc}}(\omega_m^{(i)}, T_m^{(i)}) \right|^2 ,$$

where $N_s^+ = 180$ and $N_s^- = 170$ are the number of experimental data obtained for the traction mode and the regenerative braking mode, respectively. (They are indeed obtained from a real electric motor of a commercial PHEV, Hyundai Ioniq PHEV.)

(b) Apply the regularized least-squares method for the problem stated in part (a). Compare the results with varying weights for the least-norm criterion $\lambda \|c\|_2^2$[2].

(c) Apply the kernel methods (non-parametric linear regression). Compare the results with the ones obtained in Parts (a) and (b).

[2]Since the elements of the coefficient vector c have different units and scales, you might apply the following normalized least-norm criterion: For example, let $\hat{c}(0)$ be the optimal solution for the unregularized (i.e., $\lambda = 0$) least-squares problem and use the least-norm criterion $\lambda \|(c - \hat{c}(0))/\hat{c}(0)\|_2^2$ where the operator / implies the elementwise division.

Appendix: scipy.io를 사용하여 python에서 mat 데이터 파일을 읽어서 사용하는 방법
MATLAB 데이터 파일(.mat)을 Python에서 읽어오는 방법은 다음과 같다.

```
>>> from scipy import io
>>> data1 = io.loadmat('engine_efficiency_data') # 엔진 효율 데이터 → data1
>>> data2 = io.loadmat('motor_efficiency_data') # 모터 효율 데이터 → data2
```

변수 이름을 사용하여 각 변수 데이터를 array로 사용할 수 있도록 하기 위해서는 다음의 과정이 필요하다.

```
>>> data1_var1 = data1['변수명11']
>>> data1_var2 = data1['변수명12']
>>> data2_var1 = data2['변수명21']
>>> data2_var2 = data2['변수명22']
```

위의 command를 통해서 array 타입 변수인 data1_var1, ..., data2_var2 이 생성되었음을 확인하고, 이 변수들을 사용하여 원하는 수치연산을 수행하면 된다.

본 문제에서는 모터의 Electrical Input Power (P_{elec})와 Mechanic Output Power(P_{mech}) 사이 관계에서, 효율 $\eta_{eff} \in (0,1)$ 를 바탕으로 두 동작 모드에 대해 다음의 식으로 모델링한다:

- **Traction mode ($T_m > 0$):**

$$P_{elec} = \frac{T_m \omega_m}{\eta_{eff}}$$

- **Regenerative braking mode ($T_m < 0$):**

$$P_{elec} = T_m \omega_m \eta_{eff}$$

여기서 T_m 은 토크, ω_m 은 모터 속도이고, 이 두 변수에 따라 효율이 결정된다:

$$\eta_{eff} = g(\omega_m, T_m)$$

2. Solution (a)

■ .mat file 로부터 데이터 로드

```
# ------------------------------------------------------------
# Step 1: Load data from .mat file
# ------------------------------------------------------------
mat = io.loadmat("C:/Users/SAMSUNG/OneDrive/Desktop/대학/Solution 모음/25-1/수치해석/Project/prob4/prob4_codes/motor_efficiency_data.mat")
data_traction = mat['Data_traction']           # shape: (180, 3)
data_braking = mat['Data_regen_braking']       # shape: (170, 3)
```

- .mat 파일로부터 모터 효율 실험 데이터를 불러온다.
- Data_traction: 토크 $T_m > 0$, traction mode 의 데이터 (180 개 샘플).
- Data_regen_braking: 토크 $T_m < 0$, regenerative braking mode 의 데이터 (170 개 샘플).
- 각 행 = $[\eta_{eff},\ \omega_m,\ T_m]$: 효율, 모터 속도, 모터 토크.

■ 기저 함수 정의

```
# ------------------------------------------------------------
# Step 2: Define basis function builder
# ------------------------------------------------------------
def basis_functions(omega, torque):
    """
    Construct design matrix B using 7 basis functions:
    b1 = 1, b2 = w_m, b3 = w_m^2, b4 = w_m*T_m, b5 = T_m^2, b6 = w_m*T_m^2, b7 = w_m^2*T_m^2
    """
    return np.vstack([
        np.ones_like(omega),                # b1
        omega,                              # b2
        omega**2,                           # b3
        omega * torque,                     # b4
        torque**2,                          # b5
        omega * torque**2,                  # b6
        omega**2 * torque**2                # b7
    ]).T  # Shape: (N, 7)
```

- 효율 함수 $\eta(\omega_m,\ T_m)$ 를 표현하기 위한 7 개의 basis function 을 정의한다.
- 각 basis function 은 비선형 조합을 포함하여 입력 변수의 다양성과 상호작용을 모델링할 수 있게 돕는다.
- 이 함수는 회귀를 위한 design matrix $B \in \mathbb{R}^{N\times 7}$를 생성한다.

■ 회귀 행렬 생성

```
# ------------------------------------------------------------
# Step 3: Build regression matrices
# ------------------------------------------------------------
def build_regression_matrix(data, mode='traction'):
    eta = data[:, 0]
    omega = data[:, 1]
    torque = data[:, 2]
    B = basis_functions(omega, torque)

    if mode == 'traction':
        y = (torque * omega) / eta
    elif mode == 'braking':
        y = torque * omega * eta
    else:
        raise ValueError("mode must be 'traction' or 'braking'")

    return B, y
```

- 각 모드(traction/braking)에 대해:
  - basis 행렬 B는 ω_m, T_m을 기반으로 구성.
  - target 벡터 y는 다음과 같이 정의:
    - Traction: $y = \frac{T_m \cdot \omega_m}{\eta} = P_{elec}$.
    - Braking: $y = T_m \cdot \omega_m \cdot \eta_{eff} = P_{elec}$.
- 즉, 입력 B와 출력 y를 구성해 회귀 문제 $Bc \approx y$를 만들기 위한 함수.

■ 최소제곱 해 구하기

```
48  # ------------------------------------------------------
49  # Step 4: Solve least-squares for both modes
50  # ------------------------------------------------------
51  B_plus, y_plus = build_regression_matrix(data_traction, mode='traction')
52  c_plus = np.linalg.lstsq(B_plus, y_plus, rcond=None)[0]
53
54  B_minus, y_minus = build_regression_matrix(data_braking, mode='braking')
55  c_minus = np.linalg.lstsq(B_minus, y_minus, rcond=None)[0]
```

- 최소제곱 해법을 통해 계수 c^+, c^-를 추정한다.
- 이 계수들은 각각의 basis function 이 출력(electrical power)에 얼마나 영향을 주는지를 나타냄.
- Numpy 의 "lstsq()"는 수치적으로 안정된 pseudo-inverse 방법을 사용한다.

■ 계수 출력

```
57  # ------------------------------------------------------
58  # Step 5: Output the coefficients
59  # ------------------------------------------------------
60  print("Estimated coefficients for traction mode (c+):")
61  for i, val in enumerate(c_plus, start=1):
62      print(f"c+_{i} = {val:.6f}")
63
64  print("\nEstimated coefficients for braking mode (c-):")
65  for i, val in enumerate(c_minus, start=1):
66      print(f"c-_{i} = {val:.6f}")
```

- Regression 결과로 얻은 7 개의 계수를 출력한다.
- 각 c_k는 basis function b_k에 대한 regression coefficient 로, 예측 모델 $\hat{P}_{elec} = \sum c_k \cdot b_k$에서 사용된다.

[결과 분석]

1) Traction Mode ($T_m > 0$)

```
Estimated coefficients for traction mode (c+):
c+_1 = 7.667962
c+_2 = 1.012452
c+_3 = 0.000101
c+_4 = 1.005214
c+_5 = 0.055164
c+_6 = 0.000640
c+_7 = -0.000000
```

$$\hat{f}^{+}(\omega_m, T_m) = \sum_{k=1}^{7} c_k^{+} \cdot b_k(\omega_m, T_m) \approx \frac{T_m \omega_m}{\eta_{eff}}$$

계수	값	해석
c_1^+	7.66796	출력과 무관한 오프셋 또는 Idle 손실. 클수록 저속 시 손실 유의미함.
c_2^+	1.01245	속도 증가 시 전기 파워 증가 경향. 거의 이상적 수준.
c_3^+	0.000101	속도 제곱 항. 아주 작아 거의 선형 모델에 가까움.
c_4^+	1.00521	T_m, ω_m 항의 계수. 기계적 출력과 거의 일치 → 물리적으로 정확함.
c_5^+	0.05516	토크 비선형 영향. 낮은 수준의 토크 손실 또는 saturation 반영.
c_6^+	0.000640	높은 토크&고속 구간에서의 추가 전력 소비.
c_7^+	-0.000000	고차항은 거의 무시된다. 모델이 과도하게 비선형일 필요는 없음을 보여준다.

전반적으로 선형성이 강한 모델이며, 모터의 물리적 특성에 잘 부합한다. 가장 핵심인 $c_4^+ \approx 1$이라는 결과는 신뢰도가 매우 높다.

2) RMS 오차

```
Estimated coefficients for braking mode (c-):
c-_1 = -13.132316
c-_2 = 1.991050
c-_3 = -0.000628
c-_4 = 1.009579
c-_5 = 0.052935
c-_6 = 0.000631
c-_7 = -0.000000
```

$$\hat{f}^{-}(\omega_m, T_m) = \sum_{k=1}^{7} c_k^{-} \cdot b_k(\omega_m, T_m) \approx T_m \cdot \omega_m \cdot \eta_{eff}$$

계수	값	해석
c_1^-	-13.13231	매우 큰 음의 상수항. 역방향 회생 시 비효율 or 전력 손실을 의미한다.
c_2^-	1.99105	속도 항의 계수가 매우 큼. 고속 회생 시 전력 생성량 증가를 반영하였다.
c_3^-	-0.000628	음의 곡률: 너무 높은 속도에서는 오히려 효율이 저하될 수 있다.
c_4^-	1.00958	기계 동력 항. traction 과 마찬가지로 거의 1 로 정확하다.
c_5^-	0.05293	회생 제동에서의 토크 제곱 효과. 고토크 회생에서 비선형 증가를 반영하였다.
c_6^-	0.000631	복합 항. 고속 고토크에서 추가 전기 흐름 포착 가능성이 존재한다.
c_7^-	-0.000000	고차항은 거의 무시된다. 모델이 과도하게 비선형일 필요는 없음을 보여준다.

회생 제동 모드는 약간 더 비선형적인 경향을 보인다. 큰 음의 상수항과 c_2^- 의 큰 값은 저속-고속 간의 출력 차이가 크다는 점을 나타낸다. 물리적 일관성은 유지되며, $c_4^- \approx 1$이 도출된다.

3. Solution (b)

■ .mat file 로부터 데이터 로드

```
# Step 1: Load data
mat = loadmat("C:/Users/SAMSUNG/OneDrive/Desktop/대학/Solution 모음/25-1/수치해석/Project/prob4/prob4_codes/motor_efficiency_data.mat")
data_traction = mat["Data_traction"]
data_braking = mat["Data_regen_braking"]
```

- .mat 파일로부터 모터 효율 실험 데이터를 불러온다.
- Data_traction: 토크 $T_m > 0$, traction mode 의 데이터 (180 개 샘플).
- Data_regen_braking: 토크 $T_m < 0$, regenerative braking mode 의 데이터 (170 개 샘플).
- 각 행 = $[\eta_{eff},\ \omega_m,\ T_m]$: 효율, 모터 속도, 모터 토크.

■ 기저 함수 정의

```
# Step 2: Define basis functions
def basis_functions(omega, torque):
    return np.vstack([
        np.ones_like(omega),
        omega,
        omega**2,
        omega * torque,
        torque**2,
        omega * torque**2,
        omega**2 * torque**2
    ]).T
```

- 효율 함수 $\eta(\omega_m,\ T_m)$를 표현하기 위한 7 개의 basis function 을 정의한다.
- 각 basis function 은 비선형 조합을 포함하여 입력 변수의 다양성과 상호작용을 모델링할 수 있게 돕는다.
- 이 함수는 회귀를 위한 design matrix $B \in \mathbb{R}^{N\times 7}$를 생성한다.

■ Design Matrix & Target Vector 생성

```
# Step 3: Construct design matrix and target vector
def build_matrix(data, mode):
    eta = data[:, 0]
    omega = data[:, 1]
    torque = data[:, 2]
    B = basis_functions(omega, torque)

    if mode == 'traction':
        y = (torque * omega) / eta
    elif mode == 'braking':
        y = torque * omega * eta
    else:
        raise ValueError("mode must be 'traction' or 'braking'")

    return B, y
```

- 각 모드(traction/braking)에 대해:
  - basis 행렬 B는 ω_m, T_m을 기반으로 구성.
  - target 벡터 y는 다음과 같이 정의:
    - Traction: $y = \frac{T_m \cdot \omega_m}{\eta} = P_{elec}$.
    - Braking: $y = T_m \cdot \omega_m \cdot \eta_{eff} = P_{elec}$.
- 즉, 입력 B와 출력 y를 구성해 회귀 문제 $Bc \approx y$를 만들기 위한 함수.

■ Lidge Regression 해 구하기

```
38    # Step 4: Ridge regression solver
39    def ridge_regression(B, y, lam):
40        n = B.shape[1]
41        A = B.T @ B + lam * np.eye(n)
42        return np.linalg.solve(A, B.T @ y)
```

- 정규화된 Least-squares 문제 (ridge regression)를 해석적으로 풀기 위한 함수이다.
- 목적 함수는 다음과 같다:

$$\hat{c} = arg\min_{c} \|Bc - y\|^2 + \lambda \|c\|^2$$

- 이를 통해 회귀 계수 c 가 과도하게 커지는 것을 억제하여, 모델의 일반화 성능을 높인다..

■ Multiple Lambda value 들에 대한 회귀 계산

```
44    # Step 5: Run for multiple lambda values
45    lambdas = [0, 0.01, 0.1, 1, 10, 100, 1000]
46    norms_traction, norms_braking = [], []
47
48    B_traction, y_traction = build_matrix(data_traction, "traction")
49    B_braking, y_braking = build_matrix(data_braking, "braking")
50
51    print("Lambda     ||c||^2 (Traction)     ||c||^2 (Braking)")
52    print("-----------------------------------------------------")
53    for lam in lambdas:
54        c_trac = ridge_regression(B_traction, y_traction, lam)
55        c_brake = ridge_regression(B_braking, y_braking, lam)
56        norm_trac = np.linalg.norm(c_trac)**2
57        norm_brake = np.linalg.norm(c_brake)**2
58        norms_traction.append(norm_trac)
59        norms_braking.append(norm_brake)
60        print(f"{lam:<10} {norm_trac:<24.6f} {norm_brake:<.6f}")
```

- 다양한 정규화 계수 λ에 대해 회귀를 반복 수행한다.

- 각 λ에 대해 계산된 회귀 계수 벡터 c 의 크기를 2-norm ($\|c\|^2$)으로 측정한다.
- 이 값은 모델 복잡도를 정량적으로 나타내며, λ가 커질수록 작아지는 경향을 보인다.
- 두 모드 각각에 대해 결과를 리스트로 저장한다.

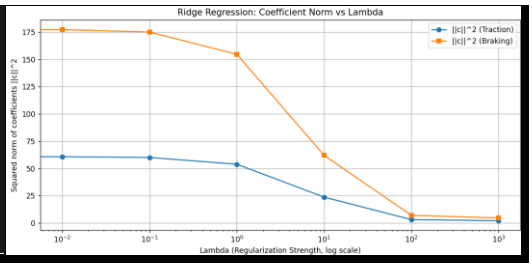
■ **Plot 출력**

```
62  # Step 6: Plot results
63  plt.figure(figsize=(10, 5))
64  plt.semilogx(lambdas, norms_traction, 'o-', label="||c||^2 (Traction)")
65  plt.semilogx(lambdas, norms_braking, 's-', label="||c||^2 (Braking)")
66  plt.xlabel("Lambda (Regularization Strength, log scale)")
67  plt.ylabel("Squared norm of coefficients ||c||^2")
68  plt.title("Ridge Regression: Coefficient Norm vs Lambda")
69  plt.grid(True)
70  plt.legend()
71  plt.tight_layout()
72  plt.show()
```

- x 축: λ(log scale)
- y 축: $\|c\|^2$

1) 시뮬레이션 결과 및 그래프 해석

Lambda	||c||^2 (Traction)	||c||^2 (Braking)
0	60.836197	177.444068
0.01	60.760412	177.190299
0.1	60.084904	174.931324
1	53.927069	154.589210
10	23.800047	62.045914
100	3.195507	7.130014
1000	2.151905	4.670616



본 실험에서는 문제 4-(a)에서 수행한 최소제곱 회귀에 정규화 항 $\lambda\|c\|^2$을 추가하여 Ridge Regression 을 적용하였다. 정규화 계수 λ 의 값에 따라 회귀 계수 벡터 c 의 제곱 노름 $\lambda\|c\|^2$이 어떻게 변하는지를 분석함으로써, 모델 복잡도와 과적합 억제 효과를 정량적으로 평가할 수 있었다.

다양한 λ 값에 대해 실험한 결과는 다음과 같다. λ = 0 에서는 일반 최소제곱법 결과로, traction 모드의 $\|c\|^2$은 60.84, braking 모드의 $\|c\|^2$은 177.44 로 상당히 큰 값을 보였다. 이는 특히 회생 제동 모드에서 회귀 계수의 크기가 매우 커져 과적합(overfitting)이 발생할 수 있음을 시사한다.

λ 값을 증가시킴에 따라 두 모드 모두에서 $\|c\|^2$ 값이 점진적으로 감소하는 경향을 확인할 수 있었다. λ = 10 부터는 감소 폭이 두드러졌으며, λ = 100 이상의 구간에서는 두 모드 모두 안정적인 수준으로 수렴하였다. 예를 들어, λ = 1000 에서는 traction 모드에서 $\|c\|^2$= 2.15, braking 모드에서 $\|c\|^2$= 4.67 로 매우 낮은 수준의 계수 크기를 유지하였다.

이러한 결과는 정규화 항이 모델의 복잡도를 효과적으로 제어하며, 특히 braking 모드처럼 과적합 경향이 강한 경우에 더욱 유의미하게 작용함을 보여준다. 일반적으로 λ 가 커질수록 모델의 복잡도는 감소하고 일반화 성능은 증가할 수 있으나, 과도한 정규화는 예측 정확도 저하를 유발할 수 있으므로, λ는 적절히 선택되어야 한다. 본 실험에서는 λ = 10 ~ 100 구간이 예측력과 안정성 간의 균형점을 제공하는 것으로 판단된다.

4. Solution (c)

1) Kernel Regression

예측값 $\hat{y}(x)$ 를 주변 training samples 의 y_i 값의 가중 평균으로 계산하는 방식이다:

$$\hat{y}(x) = \frac{\sum_{i=1}^{N} K(x, x_i) y_i}{\sum_{i=1}^{N} K(x, x_i)}$$

여기서:

- $x = (\omega_m, T_m)$: 예측할 입력.
- x_i: Training samples.
- $K(x, x_i)$: Kernel function (일반적으로 Gaussian Radial Basis Function(RBF))

$$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|^2}{2h^2}\right) \qquad (h: Bandwidth, smoothing\ 정도를\ 조절)$$

2) Python 코드 리뷰

■ .mat file 로부터 데이터 로드

```
6   # Step 1: Load data
7   mat = loadmat("C:/Users/SAMSUNG/OneDrive/Desktop/대학/Solution 모음/25-1/수치해석/Project/prob4/prob4_codes/motor_efficiency_data.mat")
8   data_traction = mat["Data_traction"]
9   data_braking = mat["Data_regen_braking"]
```

- .mat 파일로부터 모터 효율 실험 데이터를 불러온다.
- Data_traction: 토크 $T_m > 0$, traction mode 의 데이터 (180 개 샘플).
- Data_regen_braking: 토크 $T_m < 0$, regenerative braking mode 의 데이터 (170 개 샘플).
- 각 행 = $[\eta_{eff},\ \omega_m,\ T_m]$: 효율, 모터 속도, 모터 토크.

■ (x, y)쌍 추출

```
11  # Step 2: Extract (x, y) pairs
12  def extract_xy(data, mode):
13      eta = data[:, 0]
14      omega = data[:, 1]
15      torque = data[:, 2]
16      X = np.stack((omega, torque), axis=1)
17
18      if mode == 'traction':
19          y = (torque * omega) / eta
20      elif mode == 'braking':
21          y = torque * omega * eta
22      else:
23          raise ValueError("Mode must be 'traction' or 'braking'")
24      return X, y
```

- 각 모드(traction/braking)에 대해:
  - basis 행렬 B는 ω_m, T_m을 기반으로 구성.
  - target 벡터 y는 다음과 같이 정의:
    - Traction: $y = \frac{T_m \cdot \omega_m}{\eta} = P_{elec}$.
    - Braking: $y = T_m \cdot \omega_m \cdot \eta_{eff} = P_{elec}$.
- 즉, 입력 B와 출력 y를 구성해 회귀 문제 $Bc \approx y$를 만들기 위한 함수.

■ Gaussian Kernel

```
26   # Step 3: Gaussian Kernel
27   def gaussian_kernel(x, x_i, h):
28       diff = x - x_i
29       return np.exp(-np.dot(diff, diff) / (2 * h**2))
```

- 커널 함수 $K(x, x_i)$는 두 입력 벡터 간 거리 기반의 가중치를 부여한다.
- $Bandwidth\ h$는 근처 이웃에 얼마나 민감하게 반응할지 조절한다.

■ Kernel regression with Leave-One-Out

```
31   # Step 4: Kernel regression prediction with leave-one-out
32   def kernel_regression_LOO(X, y, h):
33       N = len(y)
34       y_pred = np.zeros(N)
35
36       for i in range(N):
37           x_i = X[i]
38           numer = 0.0
39           denom = 0.0
40           for j in range(N):
41               if i == j:
42                   continue  # leave-one-out
43               w = gaussian_kernel(x_i, X[j], h)
44               numer += w * y[j]
45               denom += w
46           y_pred[i] = numer / denom if denom > 0 else 0.0
47       return y_pred
```

- Leave-One-Out: 각 데이터 샘플을 대상으로, 자신을 제외한 나머지 데이터를 이용하여 예측값을 계산한다.
- 예측값은 주변 데이터들의 가중 평균으로 결정되며, 거리가 가까운 샘플일수록 더 높은 영향을 준다.

■ Run over multiple bandwidth

```
# Step 5: Test over multiple bandwidths
bandwidths = [0.1, 0.5, 1, 2, 5]
rms_traction = []
rms_braking = []

X_trac, y_trac = extract_xy(data_traction, 'traction')
X_brake, y_brake = extract_xy(data_braking, 'braking')

print("Bandwidth   RMS Error (Traction)   RMS Error (Braking)")
print("-----------------------------------------------------")

for h in bandwidths:
    y_trac_pred = kernel_regression_LOO(X_trac, y_trac, h)
    y_brake_pred = kernel_regression_LOO(X_brake, y_brake, h)

    rms_t = np.sqrt(np.mean((y_trac_pred - y_trac) ** 2))
    rms_b = np.sqrt(np.mean((y_brake_pred - y_brake) ** 2))

    rms_traction.append(rms_t)
    rms_braking.append(rms_b)
```

- 여러 $bandwidth\ h$ 값에 대해 연산을 반복 수행하며, 예측 결과의 RMS error 를 계산한다.
- 두 모드 각각에 대해 결과를 리스트로 저장한다.

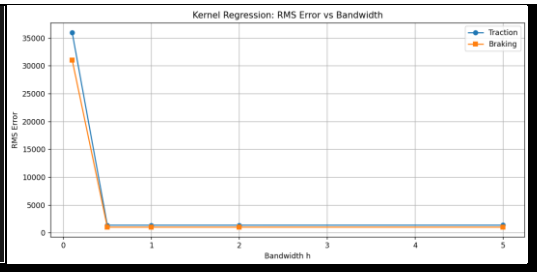
■ RMS Error 시각화

```
# Step 6: Plot RMS error vs bandwidth
plt.figure(figsize=(10, 5))
plt.plot(bandwidths, rms_traction, 'o-', label="Traction")
plt.plot(bandwidths, rms_braking, 's-', label="Braking")
plt.xlabel("Bandwidth h")
plt.ylabel("RMS Error")
plt.title("Kernel Regression: RMS Error vs Bandwidth")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

- 각 h값에 대한 RMS error 결과를 선 그래프로 시각화한다.
- 최적의 bandwidth 를 시각적으로 파악할 수 있게 한다.

3) 시뮬레이션 결과 및 그래프 해석

Bandwidth	RMS Error (Traction)	RMS Error (Braking)
0.1	36001.6177	31050.5292
0.5	1345.5781	990.1676
1	1345.5781	990.1676
2	1345.5781	990.1676
5	1348.5928	992.7116



Gaussian 커널을 활용한 비모수적 회귀(non-parametric kernel regression)를 적용하였다. 이 방식은 문제 4-(a), (b)의 parametric 회귀 모델들과 달리, 미리 정의된 basis function 없이 주어진 입력 데이터의 지역적 유사도만을 기반으로 출력을 예측한다. 실험은 다양한 bandwidth h 값을 설정하여, 커널 함수가 얼마나 넓은 범위의 이웃 샘플을 고려하는지에 따라 예측 정확도가 어떻게 변하는지를 관찰하였다.

실험 결과, bandwidth h=0.1 일 때 RMS 오차는 traction 모드에서 약 36,001.6, braking 모드에서 약 31,050.5 로 매우 높은 수치를 기록하였다. 이는 bandwidth 가 너무 작아 각 샘플이 대부분의 이웃으로부터 영향을 받지 못하고, 사실상 자기 자신만을 반영한 예측을 수행하게 되어 과적합(overfitting)이 심화된 결과로 해석된다.

반면, h=0.5~2.0 범위에서는 RMS 오차가 traction 모드 약 1,345.6, braking 모드 약 990.2 수준으로 급격히 감소하고 안정적인 값을 유지하였다. 이는 이 구간의 bandwidth 가 적절한 양의 이웃 샘플을 고려하여 예측에 활용되었으며, 국소적 평균화를 통해 노이즈를 줄이고 일반화 성능을 확보한 것으로 판단된다.

그러나 h=5.0 에서는 두 모드 모두 RMS 오차가 소폭 상승하는 경향을 보였는데, 이는 bandwidth 가 지나치게 커져 대부분의 샘플이 유사한 가중치를 받게 되면서 예측이 과도하게 평탄해지고, 결국 과소적합(underfitting)이 발생했기 때문으로 볼 수 있다.

종합적으로, kernel regression 은 특정 bandwidth 에서는 안정적인 예측 결과를 보였지만, (a)와 (b)에서의 모수 회귀 모델과 비교하면 RMS 오차 측면에서 성능이 상대적으로 떨어졌다. 이는 커널 회귀가 데이터 수가 많을수록 효과적으로 작동하는 방식이므로, 현재의 데이터 크기에서는 parametric 모델이 더 효율적일 수 있음을 시사한다.