Step 1: Understanding Locally Weighted Regression (LWR)

Unlike ordinary linear regression, which fits a **global model** to the entire dataset, LWR fits a **local model** around each query point. This is done using **weighted least squares**, where closer points to the query point get **higher weights** and farther points get **lower weights**.

Mathematical Formulation

LWR finds the parameters θ that minimize:

$$J(\theta) = \sum_{i=1}^{m} w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$$

where:

- $w^{(i)}$ is a weight function giving **higher** weights to points near the query point x_q.
- $y^{(i)}$ is the output value of the training data.
- $x^{(i)}$ is the feature value of the training data.
- θ is the regression parameter.

The **weight function** is typically chosen as a Gaussian kernel:

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x_q)^2}{2\tau^2}\right)$$

where τ is the **bandwidth parameter**, controlling how much influence distant points have.

Why is the Weight Function in LWR Typically Chosen as a Gaussian Kernel?

The weight function in **Locally Weighted Regression (LWR)** determines how much influence each training point has when making a prediction at a given query point x_{query}. The most commonly used weight function is the **Gaussian kernel**:

$$w_i = \exp\left(-\frac{(X_i - x_{\text{query}})^2}{2\tau^2}\right)$$

where τ (bandwidth) controls how much nearby points influence the prediction.

1 Smoothness and Differentiability

- The **Gaussian function is smooth and infinitely differentiable**, making it ideal for **continuous, smooth predictions**.
- Other weight functions (e.g., step or triangular functions) might introduce **sharp transitions**, causing instability.

2 Naturally Assigns Higher Weight to Nearby Points

- Gaussian weights **decay exponentially**, meaning **closer points get higher weight** and **farther points get negligible weight**.
- This property ensures that **locality** is well-respected.

◆ **Example:**

If $x_{\text{query}} = 5$, the weight for each X_i might look like this:

X_i	Distance from 5	Weight w_i
5.0	0	1.0000
4.9	0.1	0.9950
4.5	0.5	0.8825
3.0	2.0	0.1353
1.0	4.0	0.0003

This ensures that **far-off points contribute almost nothing**.

3 Exponential Decay Avoids Abrupt Cutoffs

- Some alternative weight functions like **step functions** (uniform weight inside a fixed window) or **triangular functions** abruptly cut off contributions after a certain distance.
- Gaussian weights **decay gradually**, making the transition between influential and non-influential points **smooth and continuous**.

◆ **Comparison:**

- **Gaussian**: Smoothly decreasing influence.
- **Step Function**: Hard cutoff (discontinuous).
- **Triangular**: Linear decay, but still sharp.

4 Well-Behaved Mathematically

- The Gaussian kernel ensures the weight matrix W is **always positive definite**, preventing numerical instability when computing:

$$\theta = (X^T W X)^{-1} X^T W y$$

- Some other kernels might create singular or near-singular matrices, leading to computational issues.

5️⃣ Close Relationship to Maximum Likelihood Estimation (MLE)

- Gaussian weighting resembles **MLE under a normal distribution assumption**.
- If errors in regression follow a Gaussian distribution, then using a Gaussian kernel is **statistically justified**.

6️⃣ Universal Approximation Capability

- The Gaussian kernel allows LWR to approximate **any smooth function**.
- With an appropriate τ, LWR can behave like **global linear regression** (large τ) or **nearest-neighbor regression** (small τ).

🔹 Conclusion: Why Gaussian?

✅ **Smooth, differentiable**, preventing sudden changes in predictions.
✅ **Assigns higher weight to closer points**, making LWR work effectively.
✅ **Exponential decay avoids abrupt cutoffs** (unlike step or triangular kernels).
✅ **Ensures numerical stability** when solving for θ.
✅ **Closely related to MLE**, making it statistically sound.

Step 2: Simple Dataset

Let's create a **small dataset** for demonstration:

x	y
1	1
2	2
3	1.3
4	3.75
5	2.25

Now, let's fit a locally weighted regression model **at** $x_q = 3$.

Step 3: Compute Weights

Using the **Gaussian kernel function**, let's compute the weights for each point with $\tau = 1$.

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x_q)^2}{2\tau^2}\right)$$

$x^{(i)}$	$w^{(i)} = \exp\left(-\frac{(x^{(i)}-3)^2}{2(1)^2}\right)$
1	$\exp(-2) = 0.1353$
2	$\exp(-0.5) = 0.6065$
3	$\exp(0) = 1.0000$
4	$\exp(-0.5) = 0.6065$
5	$\exp(-2) = 0.1353$

Step 4: Solve Weighted Linear Regression

The normal equation for weighted linear regression is:

$$\theta = (X^T W X)^{-1} X^T W y$$

where:

- X is the design matrix (including an intercept),
- W is the diagonal weight matrix,
- y is the output vector.

Design Matrix X

Since we are fitting a linear model $y = \theta_0 + \theta_1 x$, we construct:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}$$

Weight Matrix W

$$W = \begin{bmatrix} 0.1353 & 0 & 0 & 0 & 0 \\ 0 & 0.6065 & 0 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 0.6065 & 0 \\ 0 & 0 & 0 & 0 & 0.1353 \end{bmatrix}$$

Output Vector y

$$y = \begin{bmatrix} 1 \\ 2 \\ 1.3 \\ 3.75 \\ 2.25 \end{bmatrix}$$

Step 5: Compute Parameters θ

$$\theta = (X^T W X)^{-1} X^T W y$$

Compute $X^T W X$

$$X^T W X = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix} \times \begin{bmatrix} 0.1353 & 0 & 0 & 0 & 0 \\ 0 & 0.6065 & 0 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 0.6065 & 0 \\ 0 & 0 & 0 & 0 & 0.1353 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \\ 4 & 4 \\ 5 & 5 \end{bmatrix}$$

After computation:

$$X^T W X = \begin{bmatrix} 2.4836 & 7.4508 \\ 7.4508 & 24.924 \end{bmatrix}$$

Compute $X^T W y$

$$X^T W y = \begin{bmatrix} 7.4508 \\ 24.924 \end{bmatrix}$$

Compute θ

$$\theta = (X^T W X)^{-1} X^T W y$$

Using matrix inversion and multiplication, we get:

$$\theta = \begin{bmatrix} 0.504 \\ 0.621 \end{bmatrix}$$

So the locally weighted linear model around $x_q = 3$ is:

$$y = 0.504 + 0.621x$$

Predicted y at $x_q = 3$:

$$y(3) = 0.504 + (0.621 \times 3) = 2.367$$

Step 6: Conclusion

- We computed the **weighted linear regression model** around $x_q = 3$.
- The **predicted value at $x_q = 3$ is 2.367**, different from the global linear regression model.

Let us look at the code and its meaning :

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

def gaussian_kernel(x, x_query, tau):
    return np.exp(- (x - x_query) ** 2 / (2 * tau ** 2))

def locally_weighted_regression(X, y, x_query, tau):
    X_b = np.c_[np.ones(len(X)), X]  # Add bias term (Intercept)
    x_query_b = np.array([1, x_query])  # Query point with bias term

    W = np.diag(gaussian_kernel(X, x_query, tau))  # Compute weights

    # Compute theta using pseudo-inverse to avoid singular matrix error
    theta = np.linalg.pinv(X_b.T @ W @ X_b) @ X_b.T @ W @ y

    return x_query_b @ theta  # Return prediction
```

Step 3: Compute Weights

Using the **Gaussian kernel function**, let's compute the weights for each point with $\tau = 1$.

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x_q)^2}{2\tau^2}\right)$$

$$\begin{bmatrix} 0.1353 & 0 & 0 & 0 & 0 \\ 0 & 0.6065 & 0 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 0.6065 & 0 \\ 0 & 0 & 0 & 0 & 0.1353 \end{bmatrix}$$

Design Matrix X

Since we are fitting a **linear model** $y = \theta_0 + \theta_1 x$, we construct:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}$$

- X_b is the **augmented input matrix** where we add a column of ones.
- This is done so that we can **include the intercept (bias term) in the regression equation**.

The reason for this transformation is that the regression model is expressed as:

$$y = \theta_0 + \theta_1 x$$

In matrix form:

$$y = X_b\theta$$

where

$$X_b = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

$$\theta = (X^T W X)^{-1} X^T W y$$

2 What is $x_{\text{query},b}$? (Bias-Adjusted Query Point)

When we make a prediction at a new query point x_{query}, we need to transform it **in the same way as** X_b.

```python
x_query_b = np.array([1, x_query])  # Query point with bias term
```

This means that for any query x_{query}, we create:

$$x_{\text{query},b} = \begin{bmatrix} 1 & x_{\text{query}} \end{bmatrix}$$

Why is this needed?

Since θ is computed for the equation:

$$y = X_b\theta$$

To make a prediction:

$$\hat{y} = x_{\text{query},b} \cdot \theta$$

This ensures **consistent dimensionality** between our learned model parameters θ and the new data points.

```python

# Complex Dataset
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array([1, 3, 2, 4, 3.5, 5, 6, 7, 6.5, 8])

# Query points for LWR
X_query = np.linspace(1, 10, 100)

tau_values = [0.1, 0.5, 1.0, 5.0, 10.0]  # Different bandwidth values
```

```python

# Simple Linear Regression
lin_reg = LinearRegression()
X_reshaped = X.reshape(-1, 1)
lin_reg.fit(X_reshaped, y)
y_lin = lin_reg.predict(X_query.reshape(-1, 1))

# Visualizing
plt.figure(figsize=(12, 8))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X_query, y_lin, color='black', linestyle='dashed', label='Simple Linear Regression')

# Plot LWR for different tau values
colors = ['red', 'green', 'purple', 'orange', 'brown']
for tau, color in zip(tau_values, colors):
    y_lwr = np.array([locally_weighted_regression(X, y, x_q, tau) for x_q in X_query])
    plt.plot(X_query, y_lwr, color=color, label=f'LWR (τ={tau})')

plt.title("Effect of Different τ Values in Locally Weighted Regression")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()

```

Used in Locally Weighted Regression (LWR)

python Copy Edit

```python
y_lwr = np.array([locally_weighted_regression(X, y, x_q, tau) f
```

- This **loops over all 100 query points** (`x_q` in `X_query`).
- Each point `x_q` is passed to `locally_weighted_regression()` to **compute a locally weighted prediction.**
- This generates a **smooth regression curve that adapts to local data variations.**

Used for Plotting Results

python Copy Edit

```python
plt.plot(X_query, y_lwr, color=color, label=f'LWR (τ={tau})')
```

- The estimated values from LWR (`y_lwr`) are plotted against `X_query`.
- This shows how the LWR model fits the data differently for different values of τ.

```python
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array([1, 3, 2, 4, 3.5, 5, 6, 7, 6.5, 8])

# Query points for LWR
X_query = np.linspace(1, 10, 100)

tau_values = [0.1, 0.5, 1.0, 5.0, 10.0]  # Different bandwidth values

# Simple Linear Regression
lin_reg = LinearRegression()
X_reshaped = X.reshape(-1, 1)
lin_reg.fit(X_reshaped, y)
y_lin = lin_reg.predict(X_query.reshape(-1, 1))

# Visualizing
plt.figure(figsize=(12, 8))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X_query, y_lin, color='black', linestyle='dashed', label='Simple Linear Regression')

# Plot LWR for different tau values
colors = ['red', 'green', 'purple', 'orange', 'brown']
for tau, color in zip(tau_values, colors):
    y_lwr = np.array([locally_weighted_regression(X, y, x_q, tau) for x_q in X_query])
    plt.plot(X_query, y_lwr, color=color, label=f'LWR (τ={tau})')

plt.title("Effect of Different τ Values in Locally Weighted Regression")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```