MTH 8302 - Modèles de Régression et d'Analyse de Variance

Leçon 3 : Méthodes de Sélection de Modèles et Généralisation en Apprentissage Statistique

Polytechnique Montréal - Hiver 2025

Chiheb Trabelsi

March 26, 2025

Table des Matières

Théorie de la Décision et Compromis Biais-Variance

Théorie de la Décision et Compromis Biais-Variance

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- On considère un modèle de régression, ou tout autre modèle de prédiction où l'on a une variable cible Y est reliée à une entrée $\mathbf{X}$ selon l'équation:

$$Y = f(\mathbf{X}) + \boldsymbol{\epsilon}, \quad \text{où} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_n).$$

- L'objectif est d'analyser comment l'estimateur $\hat{f}$, obtenu à partir d'un jeu de données d'entraînement $\mathcal{D}$, se comporte lorsqu'on fait varier $\mathcal{D}$.
- Nous évaluons la distribution des prédictions $\hat{f}(\boldsymbol{x})$ pour une observation $\boldsymbol{x}$ donnée en moyenne sur différents ensembles $\mathcal{D}$.

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- **Définition :** Dans le contexte d¡un problème de prédiction, un estimateur ponctuel $\hat{\boldsymbol{\theta}}$ D'un paramètre $\boldsymbol{\theta}$, est défini comme :

$$\begin{aligned}\hat{\boldsymbol{\theta}} = g(\mathcal{D}) &= g(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n, Y_1, \ldots, Y_n) \\ &= g(\{(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \ldots, (\mathbf{X}_n, Y_n)\}) \\ &= g(\mathbf{X}, \mathbf{Y}), \quad \text{où :}\end{aligned}$$

  - g est une fonction des données de l'échantillon des données $\mathbf{X}_1, \ldots, \mathbf{X}_n$ de $\mathbf{X}$ et de leur cibles correspondantes $\mathbf{Y} = (Y_1, \ldots, Y_n)^\top$.
  - $\mathcal{D}$ est la paire du jeux de données (*dataset*) constitué par la $(\mathbf{X}, \mathbf{Y})$
- **Exemple de la Régression Linéaire :** Dans le cas de la régression linéaire multiple, où l'on considère le modèle :

$$\mathbf{Y} = \underbrace{\mathbf{X}\boldsymbol{\beta}}_{f_{\boldsymbol{\beta}}(\mathbf{X})} + \boldsymbol{\epsilon}, \quad \text{où} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_n),$$

  - Le paramètre $\boldsymbol{\theta}$ à estimer est le vecteur $\boldsymbol{\beta} \in \mathbb{R}^n$.
  - $f_{\boldsymbol{\beta}}$ est la fonction de prédiction de $\mathbf{Y}$ tel que $f_{\boldsymbol{\beta}}(\mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$.

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- Les méthodes d'estimation des paramètres du modèle de régression linéaires définissent des fonctions d'estimation de $\boldsymbol{\beta}$ où l'on a :

$$\begin{cases} \hat{\boldsymbol{\beta}}_{\text{MCO}} = g_{\text{MCO}}(\mathbf{X}, \mathbf{Y}) = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y} \\ \hat{\boldsymbol{\beta}}_{\text{MV}} = g_{\text{MV}}(\mathbf{X}, \mathbf{Y}) = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y} \end{cases}$$

- g_{MCO} et g_{MV} sont respectivement les fonctions d'estimation de $\boldsymbol{\beta}$ par la méthode des moindres carrés ordinaires et par la méthode de maximum de vraisemblance.
- La fonction de prédiction $f_{\boldsymbol{\beta}}$ est elle estimée par $f_{\hat{\boldsymbol{\beta}}_{\text{MCO}}}$ et d'une manière équivalente par $f_{\hat{\boldsymbol{\beta}}_{\text{MV}}}$ où l'on a :

$$\hat{f}_{\boldsymbol{\beta}}(\mathbf{X}) = f_{\hat{\boldsymbol{\beta}}_{\text{MCO}}}(\mathbf{X}) = f_{\hat{\boldsymbol{\beta}}_{\text{MV}}}(\mathbf{X}) = \underbrace{\mathbf{X}\hat{\boldsymbol{\beta}}_{\text{MCO}}}_{\hat{\mathbf{Y}}_{\text{MCO}}} = \underbrace{\mathbf{X}\hat{\boldsymbol{\beta}}_{\text{MV}}}_{\hat{\mathbf{Y}}_{\text{MV}}}.$$

- Comme nous pouvons le constater, $\hat{f}_{\boldsymbol{\beta}}$, l'estimateur de $f_{\boldsymbol{\beta}}$ dépend de la variable cible $\mathbf{Y}$ et des données de l'échantillon $\mathbf{X}_1, \ldots, \mathbf{X}_n$ de $\mathbf{X}$.

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- Pour une réalisation donnée $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ de l'échantillon aléatoire $\mathbf{X}_1, \ldots, \mathbf{X}_n$ de $\mathbf{X}$, $\hat{f}_{\boldsymbol{\beta}}$ **dépend de ces réalisations et des observations cibles correspondantes** $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$**.** On aura :

$$\begin{aligned}\hat{\boldsymbol{\beta}}_{\text{MCO}} = \hat{\boldsymbol{\beta}}_{\text{MV}} &= g_{\text{MCO}}(\mathcal{D} = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}, \boldsymbol{y}) \\ &= g_{\text{MV}}(\mathcal{D} = \{(\boldsymbol{x_1}, y_1), \ldots (\boldsymbol{x_n}, y_n)\}) \\ &= (\mathbf{X}_{\text{train}}{}^\top \mathbf{X}_{\text{train}})^{-1} \mathbf{X}_{\text{train}}{}^\top \boldsymbol{y}, \quad \text{où :}\end{aligned}$$

- $\mathbf{X}_{\text{train}} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})^\top \in \mathbb{R}^{n \times (p+1)}$ ici dénote la matrice contenant les réalisation $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ de l'échantillon aléatoire $\mathbf{X} = (\mathbf{X_1}, \ldots, \mathbf{X_n})^\top \in \mathbb{R}^{n \times (p+1)}$.
- L'ensemble $\mathcal{D}_{\text{train}}$ formé par les réalisations et leurs cibles correspondantes $(\mathbf{X}_{\text{train}}, \boldsymbol{y})$ est utilisé pour estimer $\boldsymbol{\beta}$ et obtenir $\hat{\boldsymbol{\beta}}$.
- $\mathcal{D}_{\text{train}} = (\mathbf{X}_{\text{train}}, \boldsymbol{y}) = (\mathbf{X}_{\text{train}}, \boldsymbol{y}_{\text{train}})$ est appelé ensemble d'ajustement ou encore **ensemble d'entraînement**.
- $\Rightarrow$ Nous avons donc $\hat{\boldsymbol{\beta}}_{\text{MCO}} = \hat{\boldsymbol{\beta}}_{\text{MV}} = g_{\text{MCO}}(\mathcal{D}_{\text{train}}) = g_{\text{MV}}(\mathcal{D}_{\text{train}})$.

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- L'objectif est d'évaluer comment l'estimateur $\hat{f}$ se comporte lorsque ses paramètres sont ajustés sur différents jeux de données $\mathcal{D}$.
- Pour une nouvelle entrée $\boldsymbol{x}$, nous étudions la distribution des prédictions $\hat{f}(\boldsymbol{x})$ lorsque le modèle est entraîné avec différents échantillons de données.
- Puisque $\hat{f}$ dépend du jeu de données d'entraînement $\mathcal{D}$, nous analysons sa performance moyenne en étudiant ses prédictions lorsqu'il est ajusté sur différents ensembles $\mathcal{D}$.
- Nous cherchons à comprendre si $\hat{f}$ est biaisé et/ou si ses prédictions varient trop en fonction des données d'entraînement.
- Pour une nouvelle observation $\boldsymbol{x}$, nous nous intéressons alors à la moyenne et à la dispersion des prédictions $\hat{f}(\boldsymbol{x})$ obtenues sur plusieurs jeux de données.

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- On considère le modèle de régression suivant :

$$Y = f(\mathbf{X}) + \epsilon, \quad \text{où} \quad \epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_n).$$

- On considère les modèles de régression linéaires et tout autre modèle de prédiction.
  - $\mathbf{X} \in \mathbb{R}^{n \times (p+1)}$: Matrice de conception contenant les n observations.
  - $\boldsymbol{x} \in \mathbb{R}^{p+1}$: Un vecteur représentant une observation donnée.
  - $f(\mathbf{X})$: La vraie relation inconnue entre $\mathbf{X}$ et Y.
  - $\hat{f}(\boldsymbol{x})$: L'estimateur de $f(\boldsymbol{x})$ obtenu à partir d'un ensemble d'entraînement.
- Objectif : Étudier la performance de $\hat{f}(\boldsymbol{x})$ en variant l'ensemble d'entraînement $\mathcal{D}$ utilisé pour estimer f.

Rappel : Propriétés des Estimateurs

- **Biais :** Le biais d'un estimateur est la différence entre l'espérance de l'estimateur et le vrai paramètre qu'il est censé estimer.
- La fonction de prédiction
  - $\text{Biais}(\hat{\theta}) = \mathbb{E}[\hat{\theta}] - \theta$
  - Un estimateur est **non biaisé** si son biais est nul: $\text{Biais}(\hat{\theta}) = 0 \Rightarrow \mathbb{E}[\hat{\theta}] = \theta$, pour toutes les tailles d'échantillon.
- **Variance :** La variance d'un estimateur mesure la dispersion de l'estimation d'un échantillon à l'autre.
  - $\text{Var}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]$
  - Un estimateur avec une faible variance a tendance à donner des résultats plus précis.
- **Dans le contexte de la Régression Linéaire :**
  - Nous cherchons à approximer une fonction inconnue f par une fonction estimée $\hat{f}$, obtenue à partir d'un jeu de données d'entraînement $\mathcal{D}_{\text{train}}$.

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- Formellement, le biais en un point donné $\boldsymbol{x} \in \mathbb{R}^{(p+1)}$ est donné par :

$$\text{Biais}(\hat{f}(\boldsymbol{x})) = \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] - f(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}[\hat{y}] - y.$$

- Un estimateur est **non biaisé** si $\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] = f(\boldsymbol{x})$ **pour tout** $\boldsymbol{x}$. C'est à dire, la moyenne des prédictions sur des échantillons de données différents $\mathbb{E}_{\mathcal{D}}[y]$ est égale à la vraie observation y.
- **Variance :** La variance d'un estimateur $\hat{f}$ mesure la sensibilité de l'estimation aux variations des échantillons de données d'entraînement.
  - Formellement, la variance en un point $\boldsymbol{x}$ est définie par :

$$\text{Var}(\hat{f}(\boldsymbol{x})) = \mathbb{E}_{\mathcal{D}}\left[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2\right] = \mathbb{E}_{\mathcal{D}}\left[(\hat{y} - \mathbb{E}_{\mathcal{D}}[\hat{y}])^2\right] = \text{Var}(\hat{y}).$$

  - Un estimateur avec une variance élevée produit **des prédictions très différentes pour des échantillons de données différents.**

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- On peut démontrer que **l'Erreur Quadratique Moyenne (EQM)** (En anglais *Mean Squared Error (MSE)*) en un point donné $\boldsymbol{x}$ peut être décomposée comme suit : **(voir Annexe pour la preuve détaillée)**

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[(y-\hat{f}(\boldsymbol{x}))^2] &= \underbrace{(f(\boldsymbol{x})-\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2}_{\text{Biais}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x})-\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]}_{\text{Variance}} + \sigma^2 \\ &= (y-\mathbb{E}_{\mathcal{D}}[\hat{y}])^2 + \mathbb{E}_{\mathcal{D}}[\hat{y}-\mathbb{E}_{\mathcal{D}}[\hat{y}]] + \sigma^2 \\ &= \mathbb{E}_{\mathcal{D}}[(y-\hat{y})^2], \quad \text{où } \sigma^2 \text{ la variance de } \epsilon \text{ est irréductible.}\end{aligned}$$

- **Interprétation : 1. Modèle à Biais Élevé (Sous-ajustement)** :
  - Un modèle trop simple ne capture pas la complexité des données.
  - **Exemple :** une régression linéaire sur un problème intrinsèquement non linéaire. $\Rightarrow$ L'estimateur $\hat{f}$ est sous-ajusté aux données d'entraînement (*problème de sous-apprentissage, en anglais underfitting*).

Théorie de la Décision et Compromis Biais-Variance de $\hat{f}$

- **Interprétation : 2. Modèle Bien Ajusté** :
  - Un bon compromis entre biais et variance.
  - **Exemple :** une régression polynomiale de degré adapté au problème.
  - $\Rightarrow$ L'estimateur $\hat{f}$ capture bien la structure des données et généralise bien.
- **Interprétation : 3. Modèle à Variance Élevée (Sur-ajustement)** :
  - Un modèle trop complexe apprend trop bien les données d'entraînement, y compris le bruit.
  - **Exemple :** une régression polynomiale de degré trop élevé.
  - $\Rightarrow$ L'estimateur $\hat{f}$ varie fortement en fonction des données (*problème de sur-apprentissage, en anglais overfitting*).

Résumé : Interprétation du Compromis Biais-Variance

- Le compromis biais-variance peut être illustré à travers 3 types de modèles :
  - **Modèle à Biais Élevé (Sous-ajustement)** :
    - Exemple : régression linéaire sur un problème non linéaire.
    - Biais élevé : le modèle est trop simple pour capturer $f(X)$.
    - Variance faible : les prédictions sont similaires quelle que soit la base d'entraînement.
  - **Modèle Bien Ajusté** :
    - Exemple : régression polynomiale d'un bon degré.
    - Biais faible : le modèle capture bien la structure sous-jacente des données.
    - Variance modérée : le modèle ne s'adapte pas excessivement aux fluctuations des données.
  - **Modèle à Variance Élevée (Sur-ajustement)** :
    - Exemple : polynôme de degré trop élevé.
    - Biais faible : le modèle suit parfaitement les données d'entraînement.
    - Variance élevée : une petite variation dans les données change significativement les prédictions.

Exercice : Illustration du Compromis Biais-Variance

Exercice : Illustration du Compromis Biais-Variance

- **Objectif :**
  - Dans cet exercice, vous allez explorer le compromis biais-variance à travers la régression polynomiale appliquée à un ensemble de données simulé.
  - Vous devrez analyser le comportement d'un modèle sous-ajusté, bien ajusté et sur-ajusté.
- **Instructions :**
  - **Comprendre les données :** Vous disposez d'un jeu de données généré artificiellement qui suit une relation non linéaire avec du bruit.
  - **Créer et entraîner des modèles :** Vous devrez ajuster des modèles polynomiaux de différents degrés et observer leur performance.
  - **Analyser les erreurs d'entraînement et de test :** Vous étudierez l'évolution de l'erreur quadratique moyenne (EQM) en fonction de la complexité du modèle.
  - **Visualiser l'effet du biais et de la variance :** Vous observerez comment le degré du polynôme influence le compromis entre biais et variance.

Compromis Biais-Variance : Code à Remplir

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
import seaborn as sns
# Configuration du style des figures
sns.set_theme()

# 1. Génération des données
np.random.seed(8302)
X = np.linspace(-5, 5, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.cos(X).ravel() ** 2 + np.sin(X).ravel() ** 3 + \
    np.random.normal(0, 0.6, X.shape[0])  # Fonction sinusoïdale avec bruit

# Séparation en jeu d'entraînement et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Visualisation des données
plt.scatter(X_train, y_train, label="Train Data", alpha=0.6)
plt.scatter(X_test, y_test, label="Test Data", alpha=0.6)
plt.legend()
plt.xlabel("X")
plt.ylabel("y")
plt.title("Données simulées avec bruit")
plt.show()
```

Compromis Biais-Variance : Code à Remplir

```python
# 2. Définition de la fonction d'affichage du compromis biais-variance
def plot_bias_variance_tradeoff(degrees):
    """
    Fonction qui entraîne des modèles polynomiaux de différents degrés
    et affiche leurs performances.
    """
    plt.figure(figsize=(12, 5))
    train_errors, test_errors = [], []
    for d in degrees:
        model = make_pipeline(PolynomialFeatures(d), LinearRegression())
        # Compléter l'entraînement du modèle
        # À COMPLETER: Entraînez le modèle sur les données d'entraînement
        model.fit(______, ______)
        # Prédictions sur l'ensemble d'entraînement et de test
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)
        # Calcul de l'erreur quadratique moyenne
        # À COMPLETER: Calculer l'erreur quadratique moyenne
        # pour les ensembles d'entraînement et de test
        train_error = np.mean((_____ - ______) ** 2)
        test_error = np.mean((_____ - ______) ** 2)
        # Stocker les erreurs
        train_errors.append(train_error)
        test_errors.append(test_error)
```

Compromis Biais-Variance : Code à Remplir

```
        # Visualisation des prédictions du modèle
        plt.scatter(X_train, y_train, label="Train Data", alpha=0.3)
        plt.scatter(X_test, y_test, label="Test Data", alpha=0.3)
        # Générer une grille de valeurs pour les prédictions
        X_plot = np.linspace(-5, 5, 100).reshape(-1, 1)
        y_plot = model.predict(X_plot)
        # À COMPLETER: Ajouter la courbe des prédictions du modèle pour le degré d
        plt.plot(______, ______, label=f"Degree {d}")
        plt.xlabel("X")
        plt.ylabel("y")
        plt.legend()
        plt.title(f"Fit du Modèle pour un polynôme de degré {d}")
        plt.show()
    # Affichage des courbes d'erreur
    plt.figure(figsize=(10, 5))
    plt.plot(degrees, train_errors, label="Training Error", marker='o', linestyle='--')
    plt.plot(degrees, test_errors, label="Test Error", marker='o', linestyle='-')
    plt.xlabel("Complexité du Modèle (Degré du polynôme)")
    plt.ylabel("Erreur Quadratique Moyenne (MSE)")
    plt.title("Compromis Biais-Variance")
    plt.legend()
    plt.show()
# 3. Expérimentation : Tester des modèles avec différents degrés de complexité
# À COMPLETER: Choisir une liste de degrés pertinents pour illustrer les trois cas
# sous-ajustement, bon ajustement, et sur-ajustement :
plot_bias_variance_tradeoff([____, ____, ____, ____, ____, ____])
```

Solution de l'Exercice : Illustration du Compromis Biais-Variance

Compromis Biais-Variance : Solution

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
import seaborn as sns
# Configuration du style des figures
sns.set_theme()

# 1. Génération des données
np.random.seed(8302)
X = np.linspace(-5, 5, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.cos(X).ravel() ** 2 + np.sin(X).ravel() ** 3 + \
    np.random.normal(0, 0.6, X.shape[0])  # Fonction sinusoïdale avec bruit

# Séparation en jeu d'entraînement et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Visualisation des données
plt.scatter(X_train, y_train, label="Train Data", alpha=0.6)
plt.scatter(X_test, y_test, label="Test Data", alpha=0.6)
plt.legend()
plt.xlabel("X")
plt.ylabel("y")
plt.title("Données simulées avec bruit")
plt.show()
```

Compromis Biais-Variance : Solution

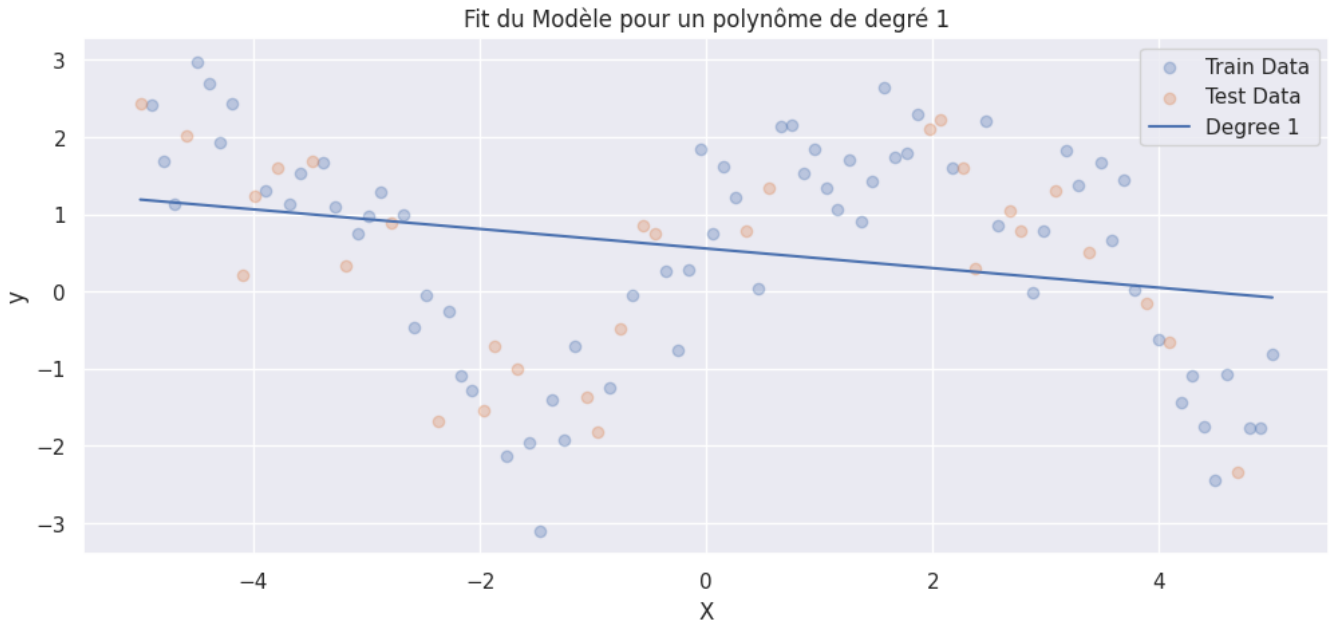
```python
# 2. Définition de la fonction d'affichage du compromis biais-variance
def plot_bias_variance_tradeoff(degrees):
    """
    Fonction qui entraîne des modèles polynomiaux de différents degrés
    et affiche leurs performances.
    """
    plt.figure(figsize=(12, 5))
    train_errors, test_errors = [], []
    for d in degrees:
        model = make_pipeline(PolynomialFeatures(d), LinearRegression())
        # Compléter l'entraînement du modèle
        # À COMPLETER: Entraînez le modèle sur les données d'entraînement
        model.fit(X_train, y_train)
        # Prédictions sur l'ensemble d'entraînement et de test
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)
        # Calcul de l'erreur quadratique moyenne
        # À COMPLETER: Calculer l'erreur quadratique moyenne
        # pour les ensembles d'entraînement et de test
        train_error = np.mean((y_train - y_train_pred) ** 2)
        test_error = np.mean((y_test - y_test_pred) ** 2)
        # Stocker les erreurs
        train_errors.append(train_error)
        test_errors.append(test_error)
```

Compromis Biais-Variance : Code à Remplir

```
            # Visualisation des prédictions du modèle
            plt.scatter(X_train, y_train, label="Train Data", alpha=0.3)
            plt.scatter(X_test, y_test, label="Test Data", alpha=0.3)
            # Générer une grille de valeurs pour les prédictions
            X_plot = np.linspace(-5, 5, 100).reshape(-1, 1)
            y_plot = model.predict(X_plot)
            # À COMPLETER: Ajouter la courbe des prédictions du modèle pour le degré d
            plt.plot(X_plot, y_plot, label=f"Degree {d}")
            plt.xlabel("X")
            plt.ylabel("y")
            plt.legend()
            plt.title(f"Fit du Modèle pour un polynôme de degré {d}")
            plt.show()
    # Affichage des courbes d'erreur
    plt.figure(figsize=(10, 5))
    plt.plot(degrees, train_errors, label="Training Error", marker='o', linestyle='--')
    plt.plot(degrees, test_errors, label="Test Error", marker='o', linestyle='-')
    plt.xlabel("Complexité du Modèle (Degré du polynôme)")
    plt.ylabel("Erreur Quadratique Moyenne (MSE)")
    plt.title("Compromis Biais-Variance")
    plt.legend()
    plt.show()
# 3. Expérimentation : Tester des modèles avec différents degrés de complexité
# À COMPLETER: Choisir une liste de degrés pertinents pour illustrer les trois cas
# sous-ajustement, bon ajustement, et sur-ajustement :
plot_bias_variance_tradeoff([1, 2, 3, 4, 5, 6, 7, 8, 15, 20, 25, 30, 35, 40])
```

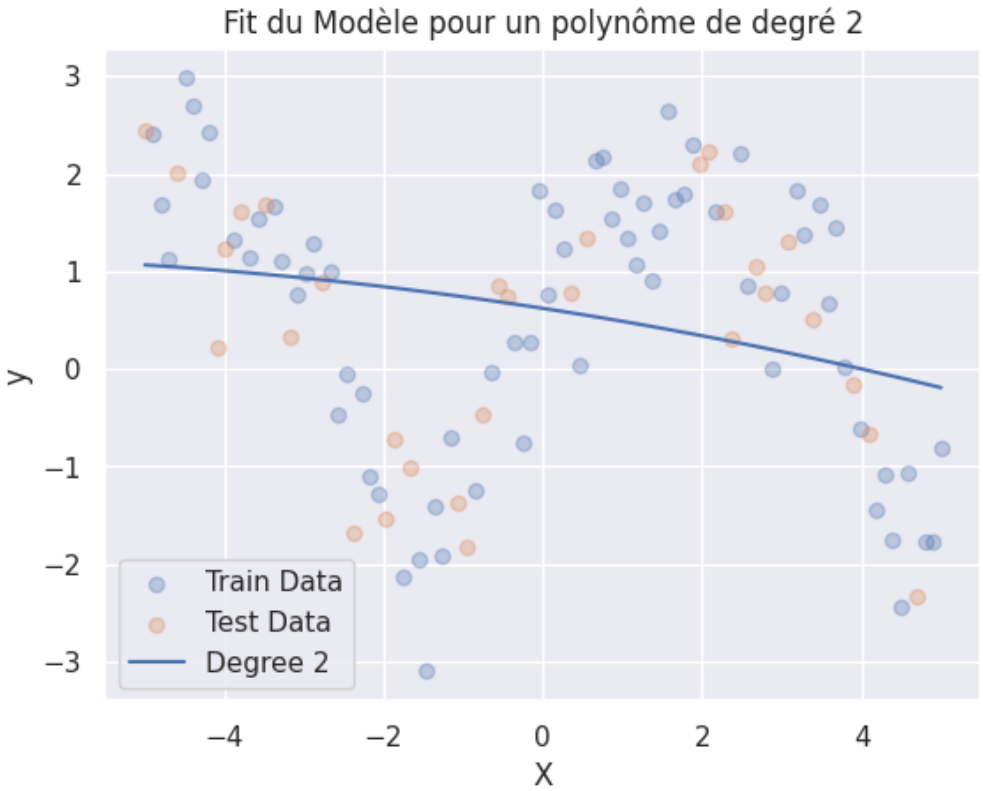
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Biais Élevé (Sous-ajustement)** :
  - Exemple : régression linéaire sur un problème non linéaire.
  - Biais élevé : le modèle est trop simple pour capturer $f(X)$.
  - Variance faible : les prédictions sont similaires quelle que soit la base d'entraînement.



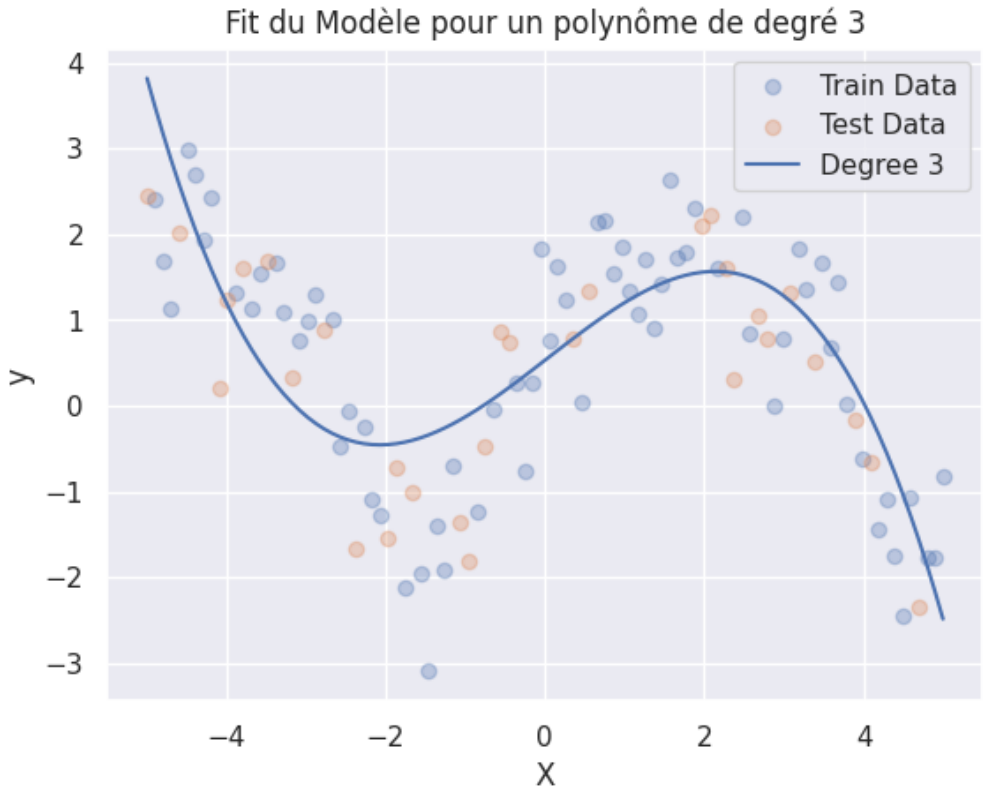
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Biais Élevé (Sous-ajustement)** :
  - Exemple : régression linéaire sur un problème non linéaire.
  - Biais élevé : le modèle est trop simple pour capturer $f(X)$.
  - Variance faible : les prédictions sont similaires quelle que soit la base d'entraînement.



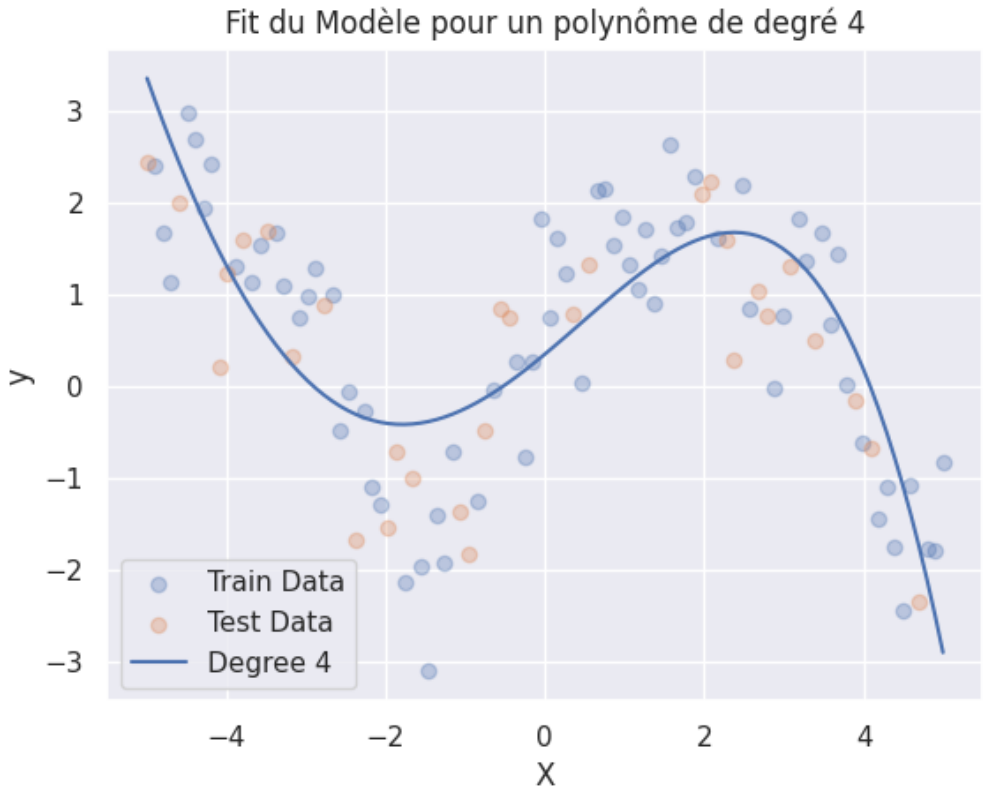
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Exemple : régression polynomiale d'un bon degré.
  - Biais faible : le modèle capture bien la structure sous-jacente des données.
  - Variance modérée : le modèle ne s'adapte pas excessivement aux fluctuations des données.



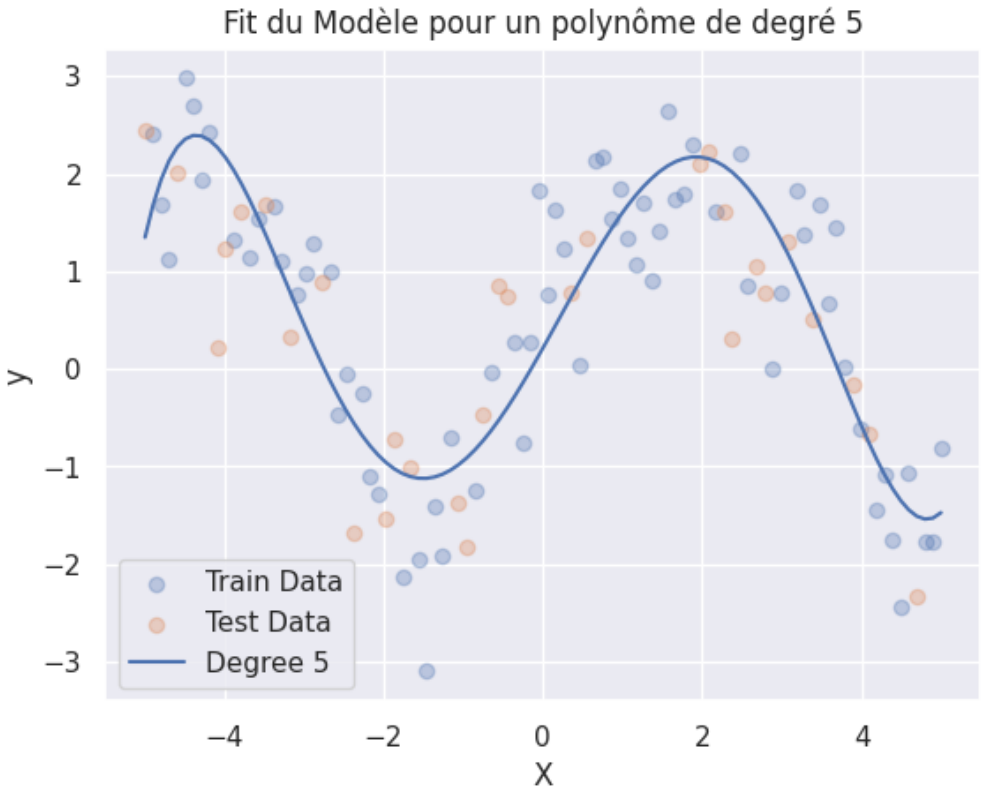
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Exemple : régression polynomiale d'un bon degré.
  - Biais faible : le modèle capture bien la structure sous-jacente des données.
  - Variance modérée : le modèle ne s'adapte pas excessivement aux fluctuations des données.



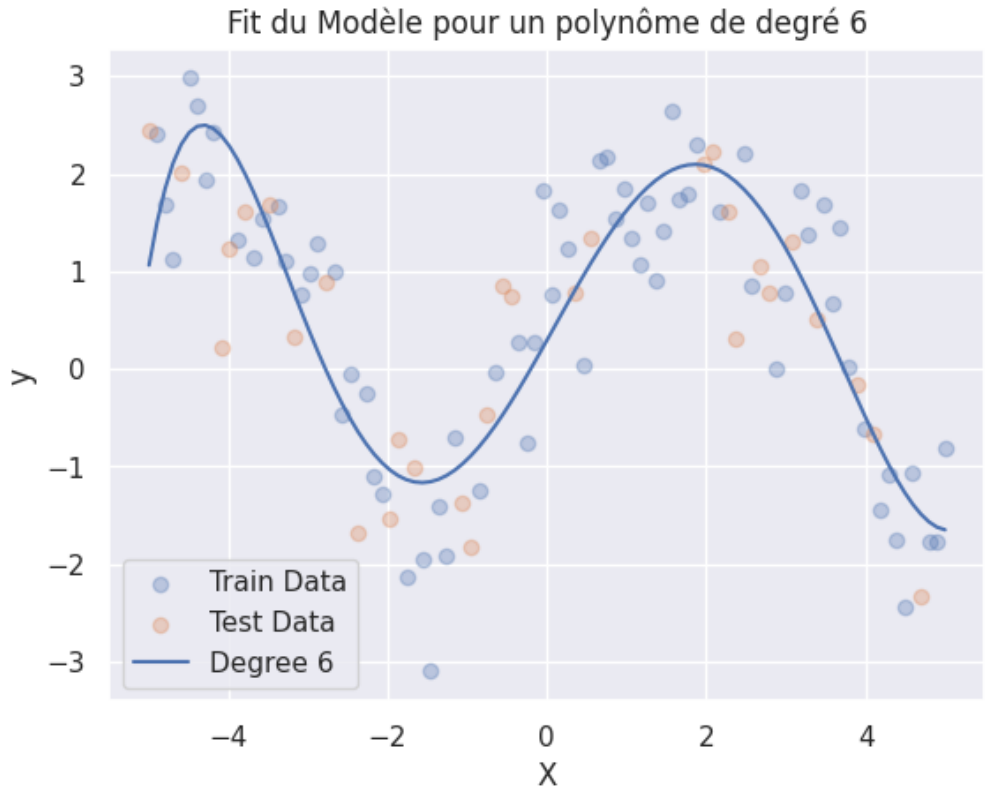
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Exemple : régression polynomiale d'un bon degré.
  - Biais faible : le modèle capture bien la structure sous-jacente des données.
  - Variance modérée : le modèle ne s'adapte pas excessivement aux fluctuations des données.



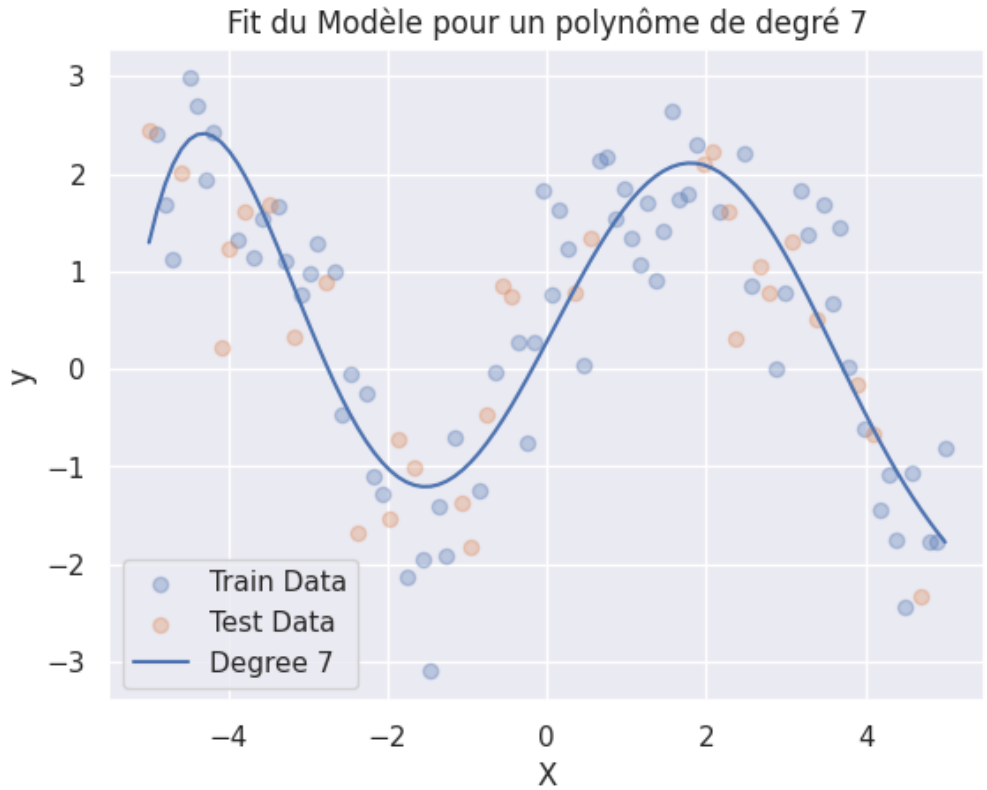
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Exemple : régression polynomiale d'un bon degré.
  - Biais faible : le modèle capture bien la structure sous-jacente des données.
  - Variance modérée : le modèle ne s'adapte pas excessivement aux fluctuations des données.



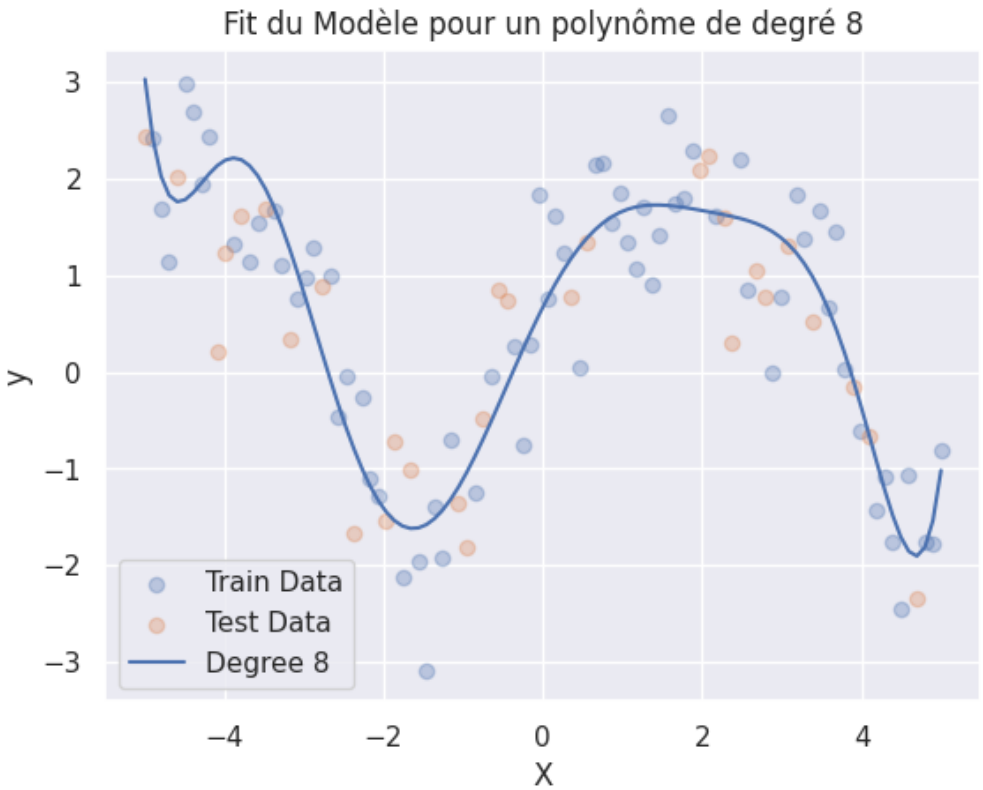
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Exemple : régression polynomiale d'un bon degré.
  - Biais faible : le modèle capture bien la structure sous-jacente des données.
  - Variance modérée : le modèle ne s'adapte pas excessivement aux fluctuations des données.



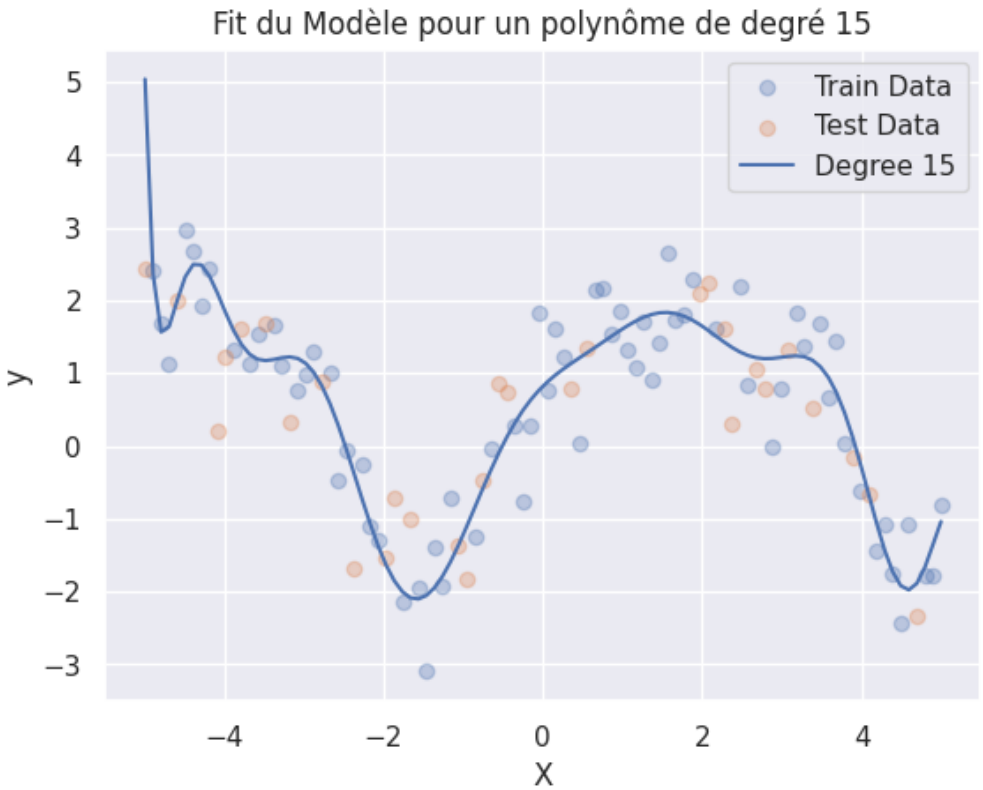
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : le modèle suit parfaitement les données d'entraînement.
  - Variance élevée : une petite variation dans les données change significativement les prédictions.



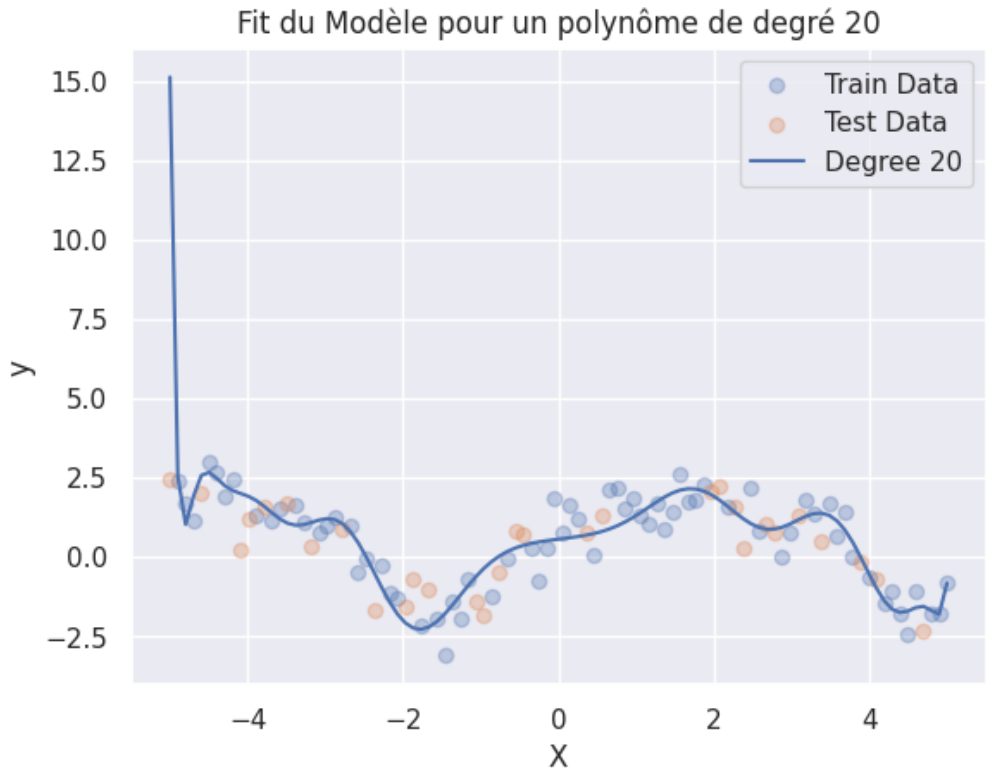
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : le modèle suit parfaitement les données d'entraînement.
  - Variance élevée : une petite variation dans les données change significativement les prédictions.



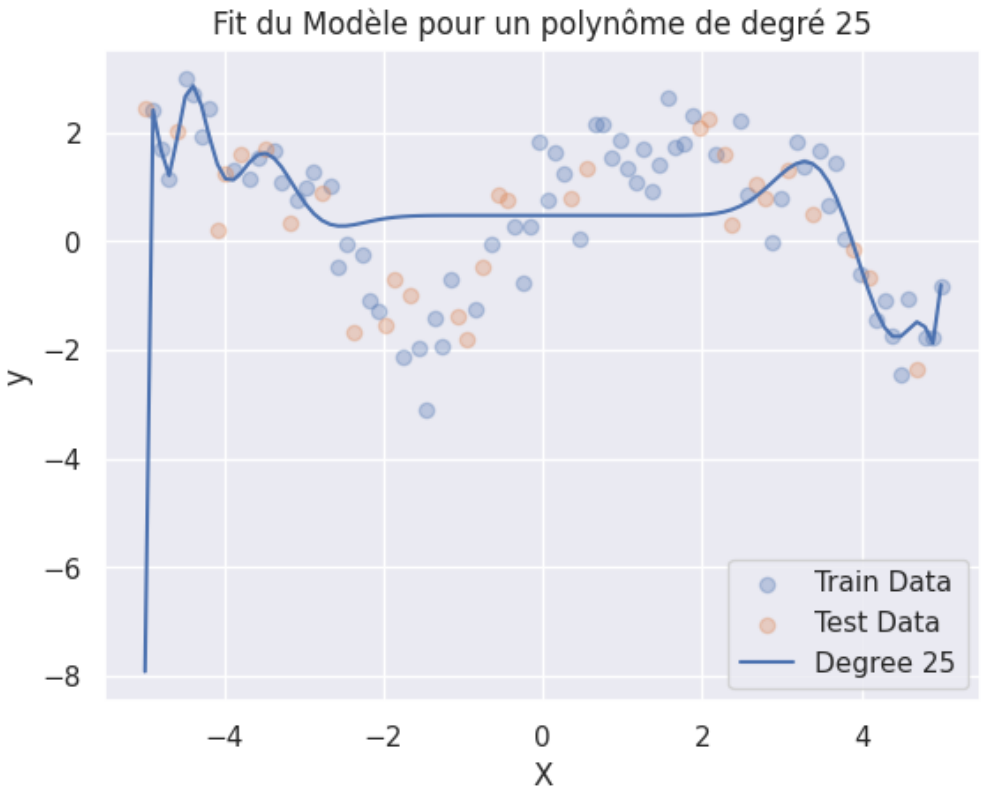
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : le modèle suit parfaitement les données d'entraînement.
  - Variance élevée : une petite variation dans les données change significativement les prédictions.



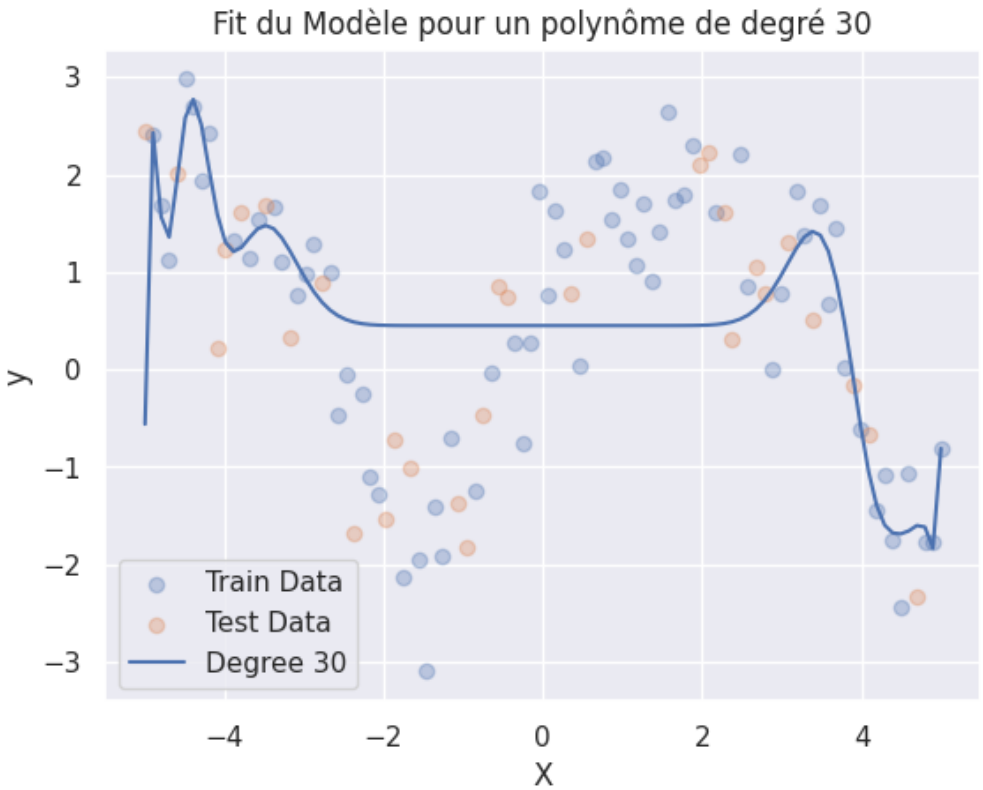
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : le modèle suit parfaitement les données d'entraînement.
  - Variance élevée : une petite variation dans les données change significativement les prédictions.



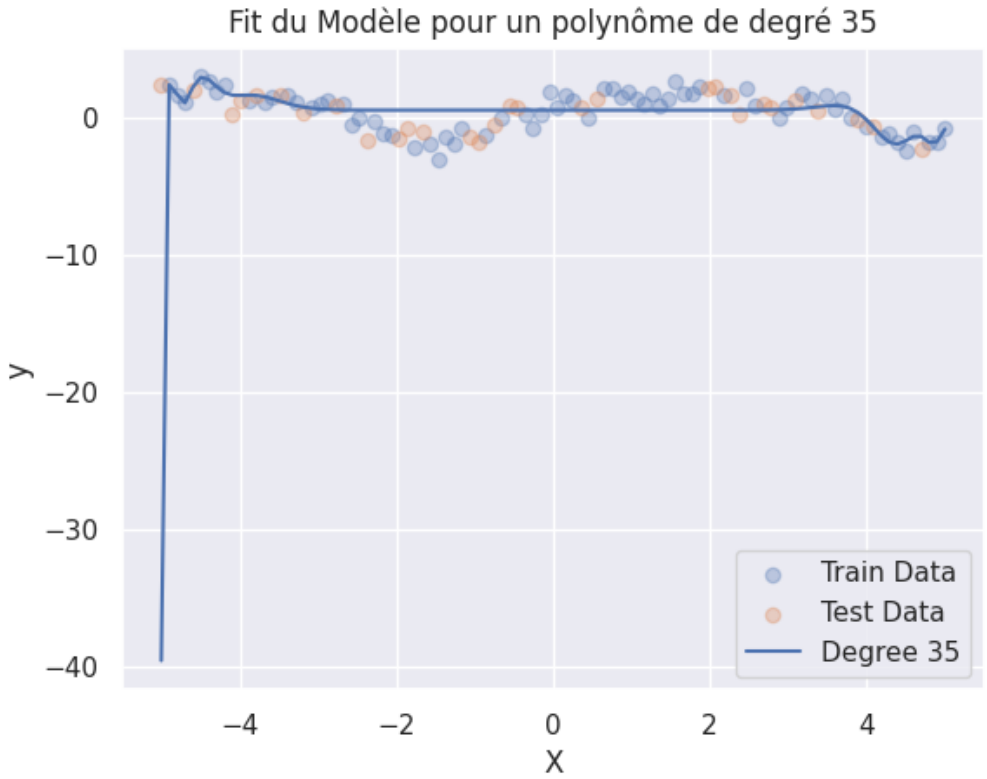
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : le modèle suit parfaitement les données d'entraînement.
  - Variance élevée : une petite variation dans les données change significativement les prédictions.



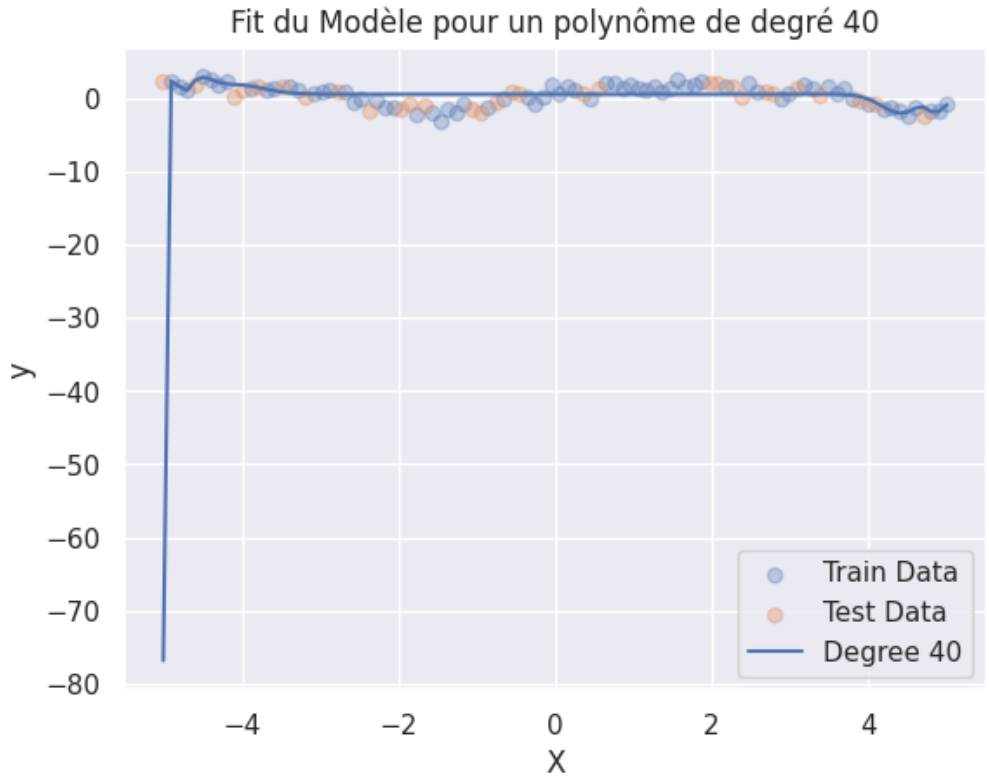
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : le modèle suit parfaitement les données d'entraînement.
  - Variance élevée : une petite variation dans les données change significativement les prédictions.



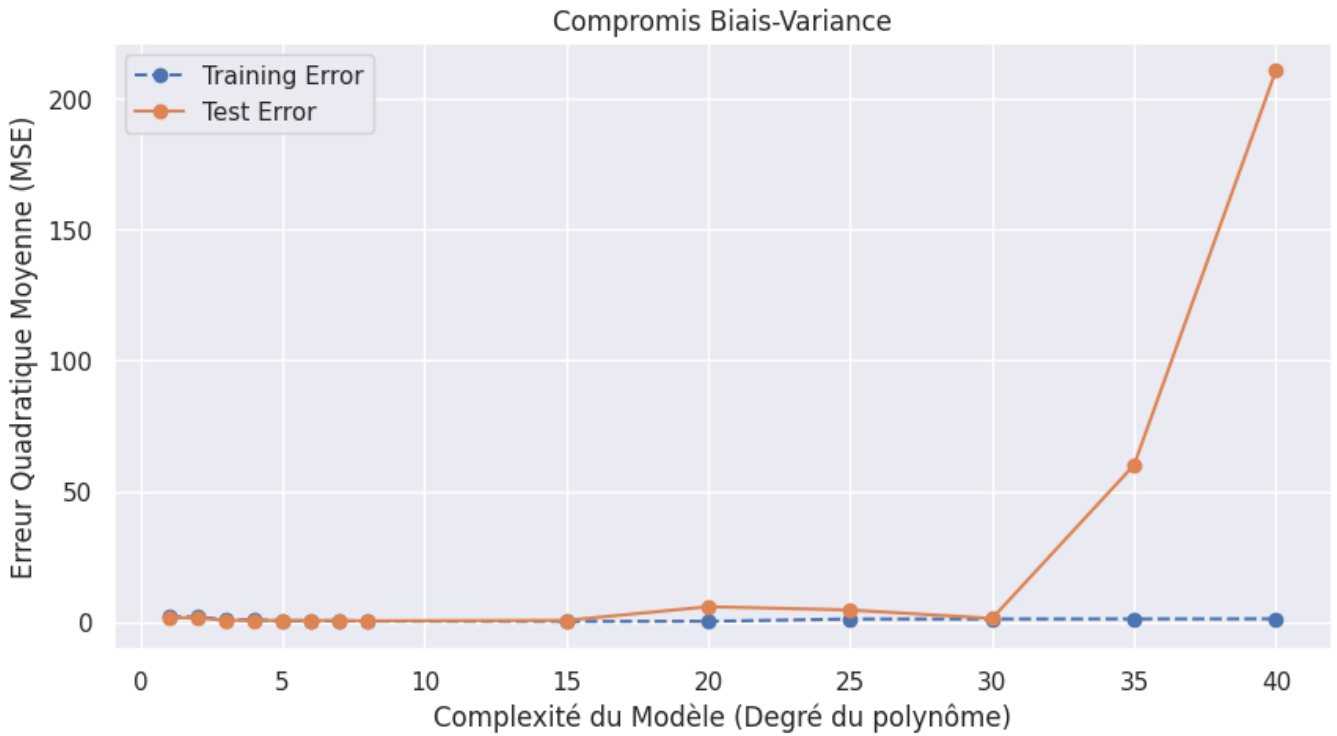
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : le modèle suit parfaitement les données d'entraînement.
  - Variance élevée : une petite variation dans les données change significativement les prédictions.



Solution : Analyse du Compromis Biais-Variance

- On observe une décroissance de l'erreur d'entraînement à mesure que le degré du polynôme augmente.
- Mais l'erreur de test diminue jusqu'à un certain seuil (degré 8), puis augmente rapidement, ce qui reflète le sur-ajustement.
- À partir de degré 25-40, la courbe devient chaotique : le modèle est trop complexe, ce qui conduit à une explosion de l'erreur de test.



Solution : Analyse du Compromis Biais-Variance

- **Figures concernées :** Polynômes de degré 1 et 2.
- **Caractéristiques :**
  - La courbe d'ajustement est quasiment linéaire et ne suit pas du tout la structure non linéaire des données.
  - **Biais élevé** : Le modèle est trop simpliste et ne capture pas les tendances sous-jacentes des données.
  - **Variance faible** : Les prédictions sont similaires pour différents jeux de données d'entraînement.
  - **Performance** : L'erreur quadratique moyenne (EQM) est élevée aussi bien sur les données d'entraînement que de test.
- **Interprétation :**
  - Le modèle est trop rigide et ne permet pas d'exploiter correctement la structure de la relation $y = f(X)$.
  - Cela correspond à un **modèle sous-ajusté (underfitting)**.

Solution : Analyse du Compromis Biais-Variance

- **Figures concernées :** Polynômes de degré 3 à 7.
- **Caractéristiques :**
  - Le modèle capture bien la structure des données sans suivre excessivement le bruit.
  - **Biais faible** : La courbe s'aligne bien sur la tendance réelle des données.
  - **Variance modérée** : Une légère fluctuation dans les données d'entraînement ne modifie pas fortement les prédictions.
  - **Performance** : L'erreur quadratique moyenne est raisonnablement basse et équilibrée entre l'entraînement et le test.
- **Interprétation :**
  - Ce modèle représente un bon **compromis entre biais et variance**.
  - Il est capable de généraliser correctement à de nouvelles données.
  - C'est le type de modèle que l'on recherche en pratique.

Solution : Analyse du Compromis Biais-Variance

- **Figures concernées :** Polynômes de degré 8 à 40.
- **Caractéristiques :**
  - La courbe suit presque parfaitement les points d'entraînement, avec de fortes oscillations.
  - **Biais faible** : La courbe passe par la plupart des points de l'ensemble d'entraînement.
  - **Variance élevée** : La moindre variation dans les données d'entraînement entraîne de grands changements dans la prédiction.
  - **Performance** : Très faible erreur sur l'entraînement, mais forte erreur sur le test (voir la courbe de l'EQM).
- **Interprétation :**
  - Le modèle mémorise les données d'entraînement au lieu de généraliser.
  - Il est trop sensible aux fluctuations du jeu d'entraînement, ce qui le rend mauvais en généralisation.
  - C'est un **modèle sur-ajusté (overfitting)**.

Lien entre le Compromis Biais-Variance et les Intervalles de Confiance et de Prédiction

Lien avec les Intervalles de Confiance et Prédiction

- Le compromis biais-variance concerne la capacité d'un modèle à généraliser à de nouvelles données.
- Cela se manifeste dans :
  - **Les intervalles de confiance et de prédiction :**
    - L'**intervalle de confiance** (**IC**) pour $E(Y_0)$ reflète l'incertitude sur la moyenne prédite, qui diminue à mesure que le modèle est bien ajusté.
    - L'**intervalle de prédiction** (**IP**) pour une nouvelle observation est toujours plus large que l'IC, car il prend en compte la variance des erreurs.
    - Si le modèle est trop flexible, càd avec une faible erreur d'entraînement mais forte variance, l'IP peut être très large, ce qui est une indication de surajustement.
- **En résumé :**
  - **Modèle trop simple** (biais élevé) : IC et IP **très larges**, car le modèle sous-exploite l'information des données.
  - **Modèle trop complexe** (variance élevée) : IC et IP plus grand car le modèle surexploite les données d'entraînement (la largeur des intervalles augmente où il y a moins de données et plus d'incertitude).

Lien avec les Intervalles de Confiance et Prédiction

- La sélection d'un modèle et la construction des intervalles de confiance et de prédiction sont liées à la théorie de la décision.
- Risque et choix du modèle :
  - Un modèle est sélectionné pour minimiser l'erreur quadratique moyenne (EQM).
  - La largeur de l'intervalle de prédiction est un indicateur clé :
    - Un modèle avec une variance élevée aura des IP plus grands, signifiant une incertitude plus grande sur les nouvelles observations.
- Utilisation des intervalles dans la prise de décision :
  - Si intervalle de prédiction est trop large, le modèle est peu fiable pour la prise de décision.
  - Un intervalle de confiance étroit sur $\mathbb{E}(Y_0)$ est un bon signe : cela signifie que l'estimation du modèle est précise.

Rappel : Intervalles de Confiance et de Prédiction

- **Somme des carrés des écarts :**

$$S_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2$$

- **Estimateur de la variance des résidus :**

$$s^2 = \frac{1}{n-p-1} \sum (y_i - \hat{y}_i)^2, \quad \text{où :}$$

  - n est la taille de l'échantillon,
  - p est le degré du polynôme.
- **Valeurs Critique :** $\pm t_{\alpha/2,\, n-p-1}$ tel que $P(T > t_{\alpha/2,\, n-p-1}) = \frac{\alpha}{2}$ où T suit une loi de Student à $n-p-1$ degrés de liberté.
- **Intervalle de Confiance :** (incertitude sur la moyenne prédite) $IC = \hat{y} \pm t \cdot s \cdot \sqrt{\frac{1}{n} + \frac{(x_0 - \bar{X})^2}{S_{xx}}}$
- **Intervalle de Prédiction :** (incertitude sur une nouvelle observation) $IP = \hat{y} \pm t \cdot s \cdot \sqrt{1 + \frac{1}{n} + \frac{(x_0 - \bar{X})^2}{S_{xx}}}$.

Exercice : Lien entre le Compromis Biais-Variance et les Intervalles de Confiance et Prédiction

Exercice : Lien avec les IC et IP

- **Objectif :**
  - Explorer le compromis biais-variance à travers la régression polynomiale appliquée à un ensemble de données simulé.
  - Comprendre et interpréter les intervalles de confiance et de prédiction.
  - Analyser le comportement d'un modèle sous-ajusté, bien ajusté et sur-ajusté.
- **Instructions :**
  - **Comprendre les données :** Un jeu de données généré artificiellement suivant une relation non linéaire avec du bruit.
  - **Créer et entraîner des modèles :** Ajuster des modèles polynomiaux de différents degrés et observer leur performance.
  - **Analyser les erreurs d'entraînement et de test :** Étudier l'évolution de l'erreur quadratique moyenne (EQM) en fonction de la complexité du modèle.
  - **Visualiser les intervalles :** Observer l'effet de la complexité du modèle sur la largeur des intervalles de confiance et de prédiction.

Exercice : Lien avec les IC et IP (Code à Compléter)

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
# Configuration du style des figures
sns.set_theme()

# 1. Génération des données
np.random.seed(8302)
X = np.linspace(-5, 5, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.cos(X).ravel() ** 2 + np.sin(X).ravel() ** 3 + \
    np.random.normal(0, 0.6, X.shape[0])  # Fonction sinusoïdale avec bruit

# Séparation en jeu d'entraînement et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Visualisation des données
plt.scatter(X_train, y_train, label="Données d'entraînement", alpha=0.6)
plt.scatter(X_test, y_test, label="Données de test", alpha=0.6)
plt.legend()
plt.xlabel("X")
plt.ylabel("y")
plt.title("Données simulées avec bruit")
plt.show()
```

Exercice : Lien avec les IC et IP (Code à Compléter)

```python
from scipy import stats
def plot_model_with_intervals(degree):
    """
    Fonction qui entraîne un modèle polynomial et affiche les intervalles
    de confiance et de prédiction.
    """
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    model.fit(X_train, y_train)

    X_plot = np.linspace(-5, 5, 100).reshape(-1, 1)
    y_pred = model.predict(X_plot)

    # Transformation polynomiale
    poly = PolynomialFeatures(degree)
    X_train_poly = poly.fit_transform(X_train)
    X_plot_poly = poly.transform(X_plot)

    #  À remplir: Calculer le facteur critique de Student
    n = ______
    p = ______.shape[1]
    dof = ______  # À compléter
    t_critical = stats.t.ppf(0.975, df=______)  # À compléter

    # À remplir: Calculer les résidus et l'erreur standard
    residuals = y_train - model.predict(X_train)
    mse = np.sum(residuals**2) / (________)  # À compléter : degrés de liberté
    sigma_hat = np.sqrt(mse)
```

Exercice : Lien avec les IC et IP (Code à Compléter)

```
X_mean = np.mean(X_train_poly, axis=0)
S_xx = np.sum((________ - ________) ** 2, axis=0) # À remplir: Calculer S_xx
# IMPORTANT : Ajout d'un epsilon pour éviter la division par zéro
S_xx[S_xx == 0] += 1e-10
# À remplir : Calculer les intervalles de confiance et de prédiction
SE_confidence = sigma_hat * np.sqrt(
    1/n + np.sum((X_plot_poly - X_mean) ** 2 / S_xx, axis=1))
SE_prediction = sigma_hat * np.sqrt(
    1 + 1/n + np.sum((X_plot_poly - X_mean) ** 2 / S_xx, axis=1))
ci_upper = y_pred + _______ * _______
ci_lower = y_pred - _______ * _______
pi_upper = y_pred + _______ * _______
pi_lower = y_pred - _______ * _______
# Visualisation
plt.figure(figsize=(8, 5))
plt.scatter(X_train, y_train, label="Données d'entraînement", alpha=0.3)
plt.scatter(X_test, y_test, label="Données de test", alpha=0.3)
plt.plot(X_plot, y_pred, label=f"Modèle degré {degree}", color='red')
plt.fill_between(X_plot.ravel(), ci_lower, ci_upper, color='blue',
                 alpha=0.2, label='Intervalle de Confiance (95%)')
plt.fill_between(X_plot.ravel(), pi_lower, pi_upper, color='green',
                 alpha=0.2, label='Intervalle de Prédiction (95%)')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.title(f"Modèle de degré {degree} : Intervalles de Confiance et de Prédiction")
plt.show()
```

Exercice : Lien avec les IC et IP (Code à Compléter)

```
# À remplir Tester pour différents degrés de polynômes
degrees = [_______, _______, _______, _______, _______, ...,  _______]
for d in degrees:
    plot_model_with_intervals(d)
# Comparaison des erreurs
train_errors, test_errors = [], []

for d in degrees:
    model = make_pipeline(PolynomialFeatures(d), LinearRegression())
    model.fit(X_train, y_train)

    train_error = np.mean((y_train - model.predict(X_train)) ** 2)
    test_error = np.mean((y_test - model.predict(X_test)) ** 2)

    train_errors.append(train_error)
    test_errors.append(test_error)

# Visualisation des erreurs
plt.figure(figsize=(8, 5))
plt.plot(degrees, train_errors, marker='o', linestyle='--', label="Erreur d'entraînement")
plt.plot(degrees, test_errors, marker='o', linestyle='-', label="Erreur de test")
plt.xlabel("Degré du polynôme")
plt.ylabel("Erreur Quadratique Moyenne (MSE)")
plt.title("Compromis Biais-Variance et Erreur de Test")
plt.legend()
plt.show()
```

Questions d'Analyse

- Quel est l'effet de l'augmentation du degré du polynôme sur l'erreur d'entraînement et l'erreur de test ?
- Comment la largeur des intervalles de confiance et de prédiction évolue-t-elle avec la complexité du modèle ?
- Pourquoi un modèle sur-ajusté a-t-il des intervalles de confiance plus serrés mais des intervalles de prédiction plus larges ?
- En quoi cette analyse peut-elle aider à choisir un modèle optimal en pratique ?

Solution de l'Exercice : Lien entre le Compromis Biais-Variance et les Intervalles de Confiance et Prédiction

Exercice : Lien avec les IC et IP (Solution)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split

# Configuration du style des figures
sns.set_theme()

# 1. Génération des données
np.random.seed(8302)
X = np.linspace(-5, 5, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.cos(X).ravel() ** 2 + np.sin(X).ravel() ** 3 + \
    np.random.normal(0, 0.6, X.shape[0])  # Fonction sinusoïdale avec bruit

# Séparation en jeu d'entraînement et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Visualisation des données
plt.scatter(X_train, y_train, label="Données d'entraînement", alpha=0.6)
plt.scatter(X_test, y_test, label="Données de test", alpha=0.6)
plt.legend()
plt.xlabel("X")
plt.ylabel("y")
plt.title("Données simulées avec bruit")
```

Exercice : Lien avec les IC et IP (Solution)

```python
from scipy import stats

def plot_model_with_intervals(degree):
    """
    Fonction qui entraîne un modèle polynomial et affiche les intervalles
    de confiance et de prédiction.
    """
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    model.fit(X_train, y_train)

    X_plot = np.linspace(-5, 5, 100).reshape(-1, 1)
    y_pred = model.predict(X_plot)

    # Transformation polynomiale
    poly = PolynomialFeatures(degree)
    X_train_poly = poly.fit_transform(X_train)
    X_plot_poly = poly.transform(X_plot)

    # Calcul du facteur critique de Student
    n = len(y_train)
    p = X_train_poly.shape[1]
    dof = n - p
    t_critical = stats.t.ppf(0.975, df=dof)

    # Calcul des résidus et de l'erreur standard
    residuals = y_train - model.predict(X_train)
    mse = np.sum(residuals**2) / dof
```

Exercice : Lien avec les IC et IP (Solution)

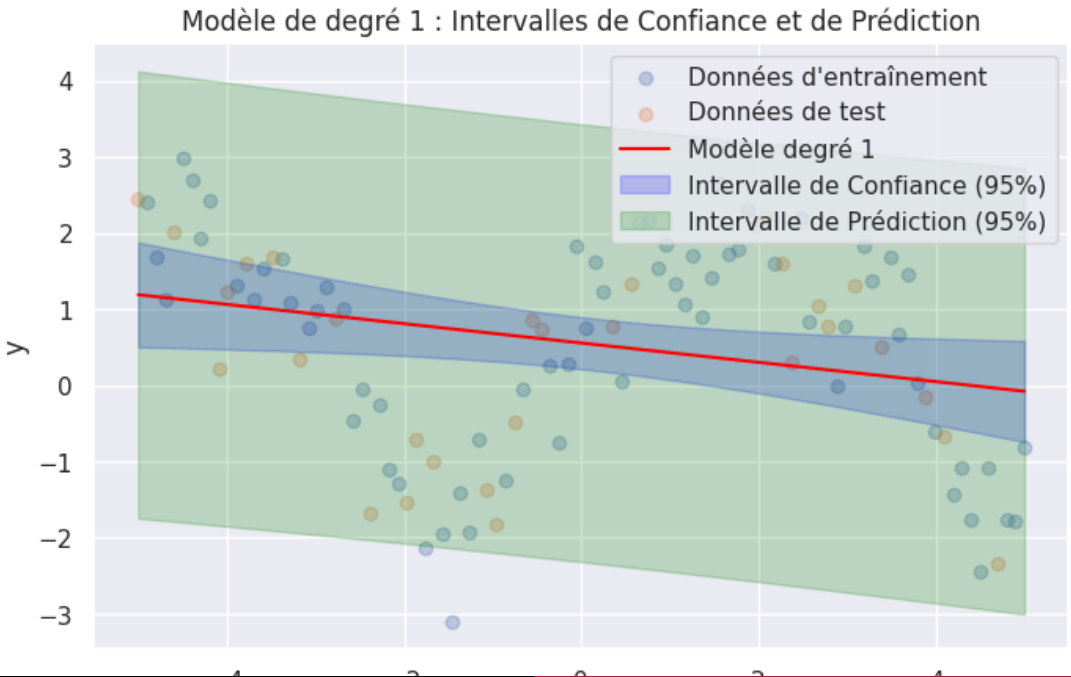
```
# Calcul de S_xx
X_mean = np.mean(X_train_poly, axis=0)
S_xx = np.sum((X_train_poly - X_mean) ** 2, axis=0)
# IMPORTANT : Ajout d'un epsilon pour éviter la division par zéro
S_xx[S_xx == 0] += 1e-10
# Calcul des intervalles de confiance et de prédiction
SE_confidence = sigma_hat * np.sqrt(
    1/n + np.sum((X_plot_poly - X_mean) ** 2 / S_xx, axis=1))
SE_prediction = sigma_hat * np.sqrt(
    1 + 1/n + np.sum((X_plot_poly - X_mean) ** 2 / S_xx, axis=1))
ci_upper = y_pred + t_critical * SE_confidence
ci_lower = y_pred - t_critical * SE_confidence
pi_upper = y_pred + t_critical * SE_prediction
pi_lower = y_pred - t_critical * SE_prediction
# Visualisation
plt.figure(figsize=(8, 5))
plt.scatter(X_train, y_train, label="Données d'entraînement", alpha=0.3)
plt.scatter(X_test, y_test, label="Données de test", alpha=0.3)
plt.plot(X_plot, y_pred, label=f"Modèle degré {degree}", color='red')
plt.fill_between(X_plot.ravel(), ci_lower, ci_upper, color='blue',
                 alpha=0.2, label='Intervalle de Confiance (95%)')
plt.fill_between(X_plot.ravel(), pi_lower, pi_upper, color='green',
                 alpha=0.2, label='Intervalle de Prédiction (95%)')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.title(f"Modèle de degré {degree} : Intervalles de Confiance et de Prédiction")
```

Exercice : Lien avec les IC et IP (Solution)

```
# Tester pour différents degrés de polynômes
degrees = [1, 2, 3, 4, 5, 6, 7, 8, 15, 20, 25, 30, 35, 40]
for d in degrees:
    plot_model_with_intervals(d)

# Comparaison des erreurs
train_errors, test_errors = [], []

for d in degrees:
    model = make_pipeline(PolynomialFeatures(d), LinearRegression())
    model.fit(X_train, y_train)

    train_error = np.mean((y_train - model.predict(X_train)) ** 2)
    test_error = np.mean((y_test - model.predict(X_test)) ** 2)

    train_errors.append(train_error)
    test_errors.append(test_error)

# Visualisation des erreurs
plt.figure(figsize=(8, 5))
plt.plot(degrees, train_errors, marker='o', linestyle='--', label="Erreur d'entraînement")
plt.plot(degrees, test_errors, marker='o', linestyle='-', label="Erreur de test")
plt.xlabel("Degré du polynôme")
plt.ylabel("Erreur Quadratique Moyenne (MSE)")
plt.title("Compromis Biais-Variance et Erreur de Test")
plt.legend()
plt.show()
```

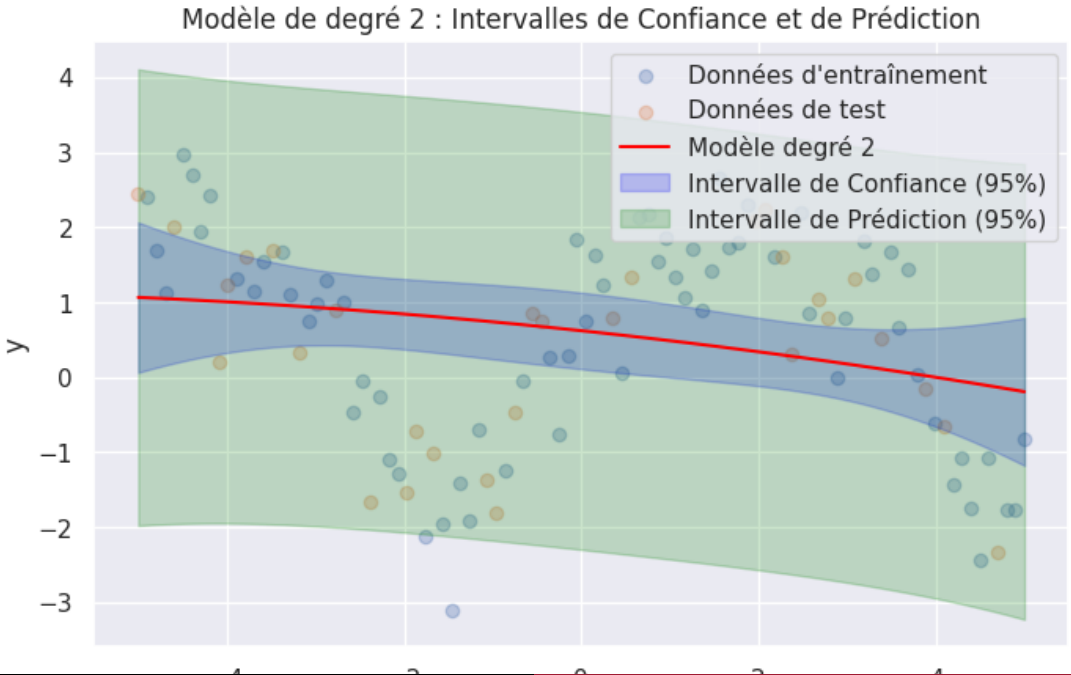
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Biais Élevé (Sous-ajustement)** :
  - Biais élevé : le modèle est trop simple pour capturer $f(X)$.
  - Les intervalles de confiance (bleu) et de prédiction (vert) sont très larges, indiquant une incertitude élevée.
  - Le modèle ne suit pas bien la tendance réelle des données, ce qui est un indicateur de biais élevé.
  - Variance faible : les prédictions sont similaires quelle que soit la base d'entraînement.



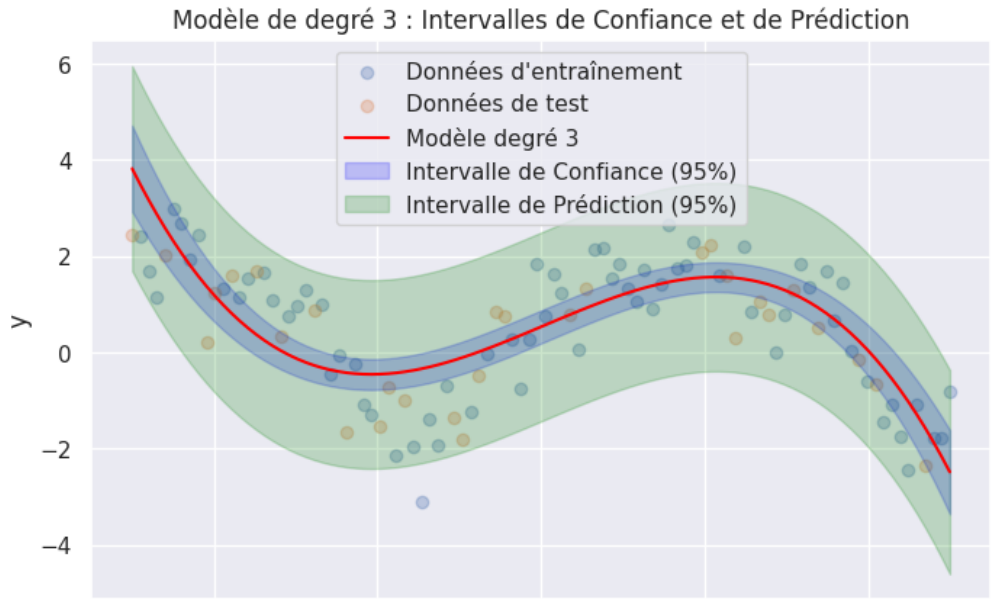
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Biais Élevé (Sous-ajustement)** :
  - Biais élevé : le modèle est trop simple pour capturer $f(X)$.
  - Les intervalles de confiance (bleu) et de prédiction (vert) sont très larges, indiquant une incertitude élevée.
  - Le modèle ne suit pas bien la tendance réelle des données, ce qui est un indicateur de biais élevé.
  - Variance faible : les prédictions sont similaires quelle que soit la base d'entraînement.



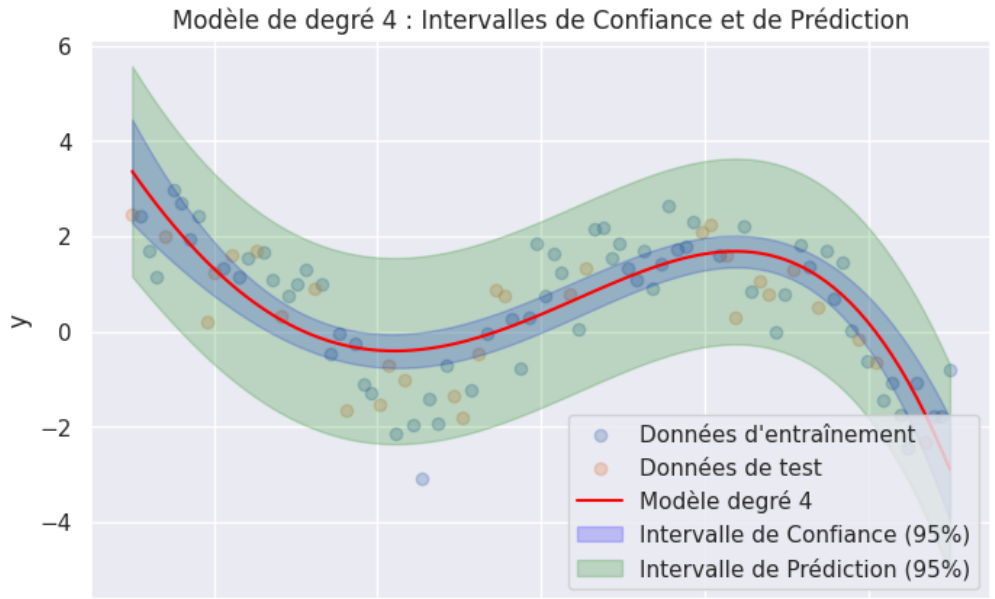
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Régression polynomiale d'un bon degré. Ici, le modèle semble offrir un bon compromis entre biais et variance.
  - Biais faible : La courbe rouge commence à mieux épouser la structure des données. Les intervalles de confiance deviennent plus resserrés, indiquant une meilleure estimation de la moyenne.
  - L'intervalle de prédiction reste large mais raisonnable, ce qui est normal car il prend en compte la variabilité des nouvelles observations.



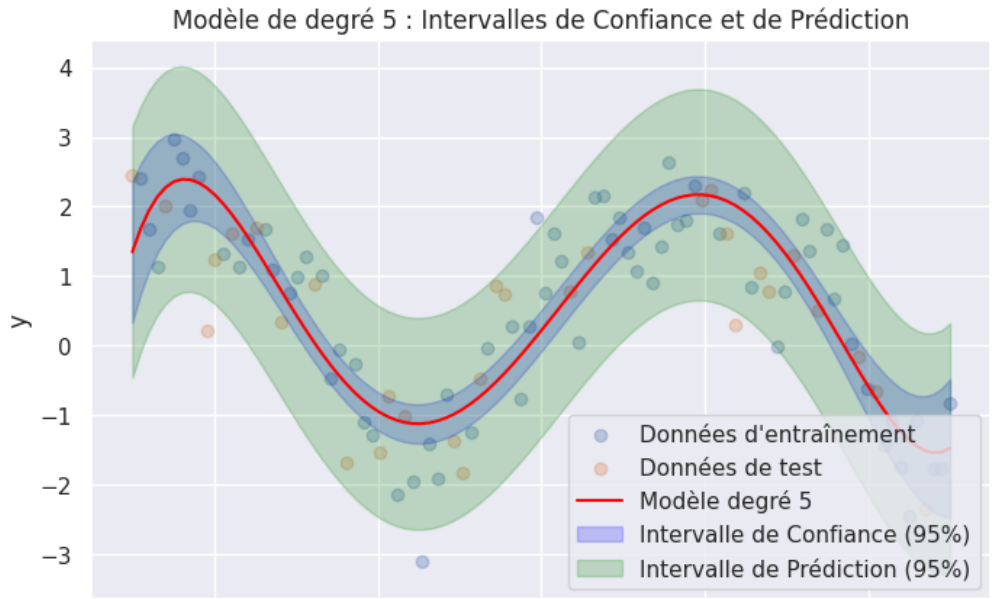
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Régression polynomiale d'un bon degré. Ici, le modèle semble offrir un bon compromis entre biais et variance.
  - Biais faible : La courbe rouge commence à mieux épouser la structure des données. Les intervalles de confiance deviennent plus resserrés, indiquant une meilleure estimation de la moyenne.
  - L'intervalle de prédiction reste large mais raisonnable, ce qui est normal car il prend en compte la variabilité des nouvelles observations.



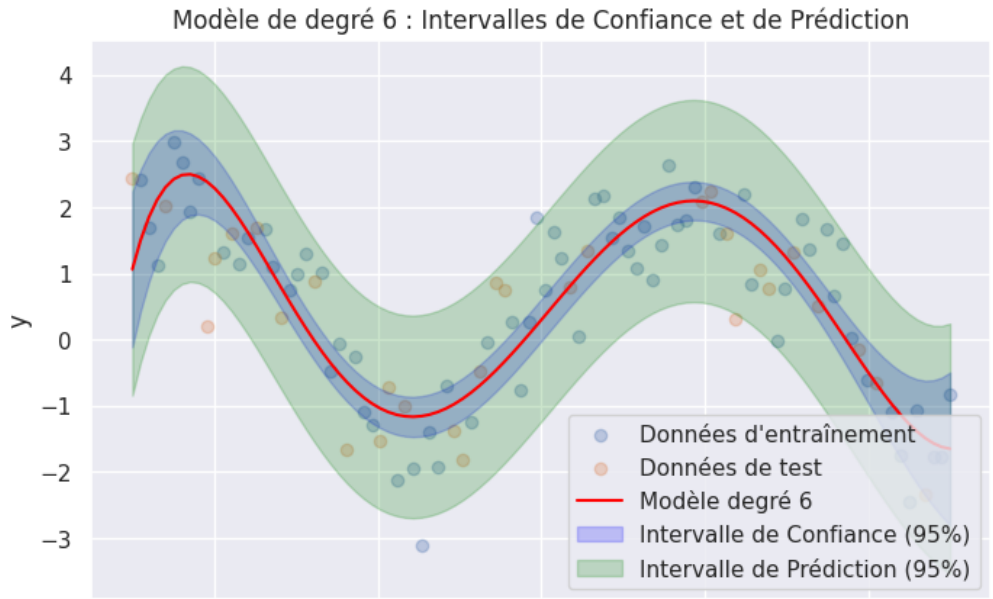
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Régression polynomiale d'un bon degré. Ici, le modèle semble offrir un bon compromis entre biais et variance.
  - Biais faible : La courbe rouge commence à mieux épouser la structure des données. Les intervalles de confiance deviennent plus resserrés, indiquant une meilleure estimation de la moyenne.
  - L'intervalle de prédiction reste large mais raisonnable, ce qui est normal car il prend en compte la variabilité des nouvelles observations.



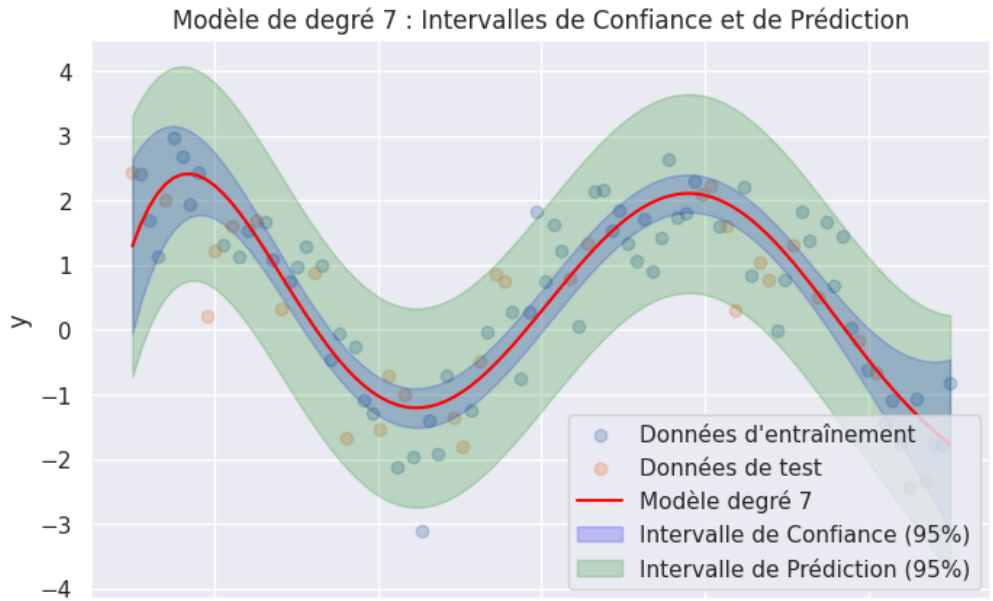
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Régression polynomiale d'un bon degré. Ici, le modèle semble offrir un bon compromis entre biais et variance.
  - Biais faible : La courbe rouge commence à mieux épouser la structure des données. Les intervalles de confiance deviennent plus resserrés, indiquant une meilleure estimation de la moyenne.
  - L'intervalle de prédiction reste large mais raisonnable, ce qui est normal car il prend en compte la variabilité des nouvelles observations.



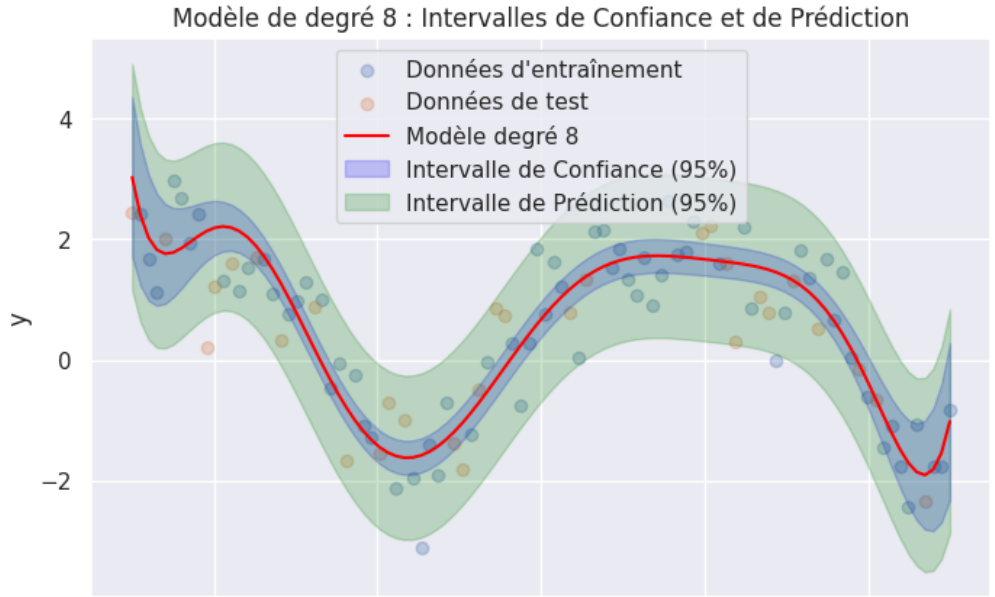
Solution : Interprétation du Compromis Biais-Variance

- **Modèle Bien Ajusté** :
  - Régression polynomiale d'un bon degré. Ici, le modèle semble offrir un bon compromis entre biais et variance.
  - Biais faible : La courbe rouge commence à mieux épouser la structure des données. Les intervalles de confiance deviennent plus resserrés, indiquant une meilleure estimation de la moyenne.
  - L'intervalle de prédiction reste large mais raisonnable, ce qui est normal car il prend en compte la variabilité des nouvelles observations.



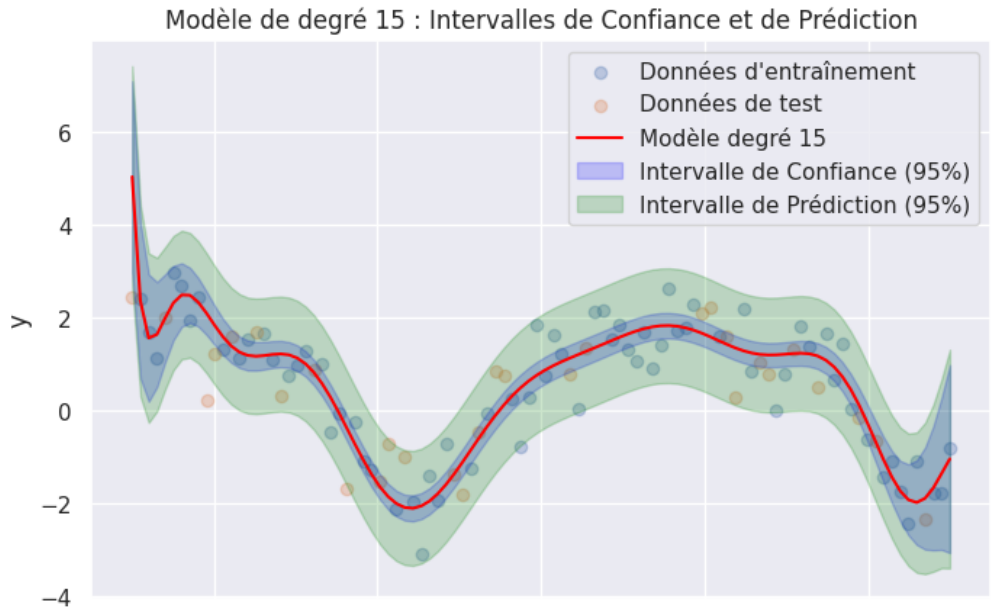
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : La courbe rouge épouse trop fidèlement les points d'entraînement, capturant même le bruit. Les intervalles de confiance sont parfois instables, en particulier aux extrémités.
  - Variance élevée : une petite variation dans les données change significativement les prédictions. Le modèle s'adapte trop aux variations du jeu d'entraînement et risque de mal généraliser sur de nouvelles données.



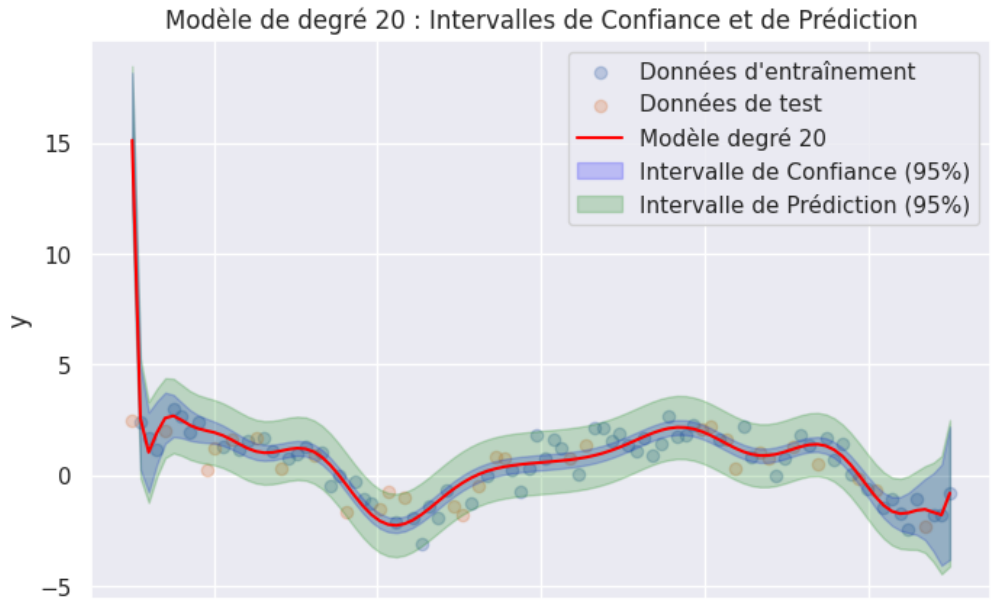
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : La courbe rouge épouse trop fidèlement les points d'entraînement, capturant même le bruit. Les intervalles de confiance sont parfois instables, en particulier aux extrémités.
  - Variance élevée : une petite variation dans les données change significativement les prédictions. Le modèle s'adapte trop aux variations du jeu d'entraînement et risque de mal généraliser sur de nouvelles données.



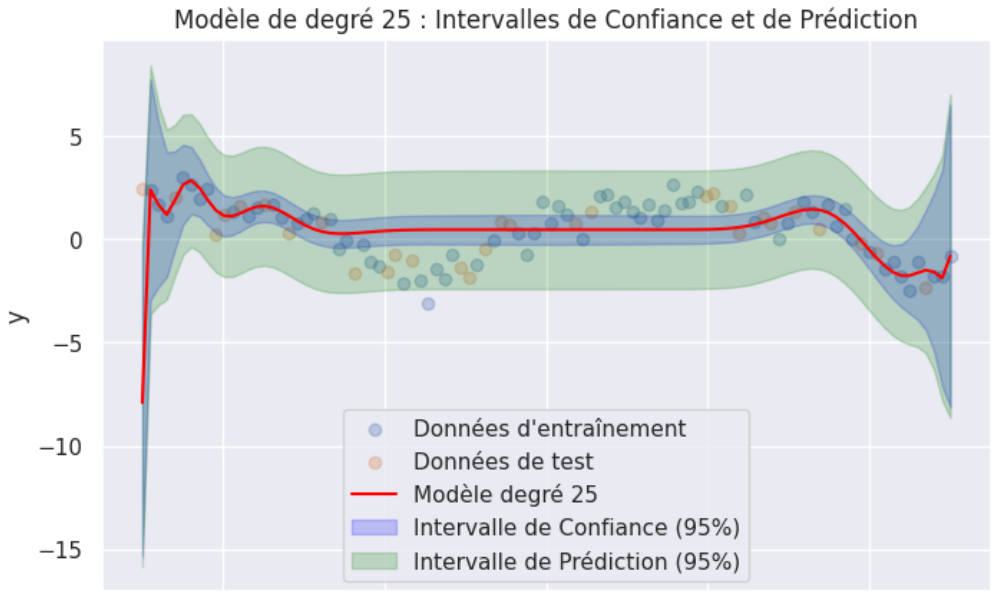
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : La courbe rouge épouse trop fidèlement les points d'entraînement, capturant même le bruit. Les intervalles de confiance sont parfois instables, en particulier aux extrémités.
  - Variance élevée : une petite variation dans les données change significativement les prédictions. Le modèle s'adapte trop aux variations du jeu d'entraînement et risque de mal généraliser sur de nouvelles données.



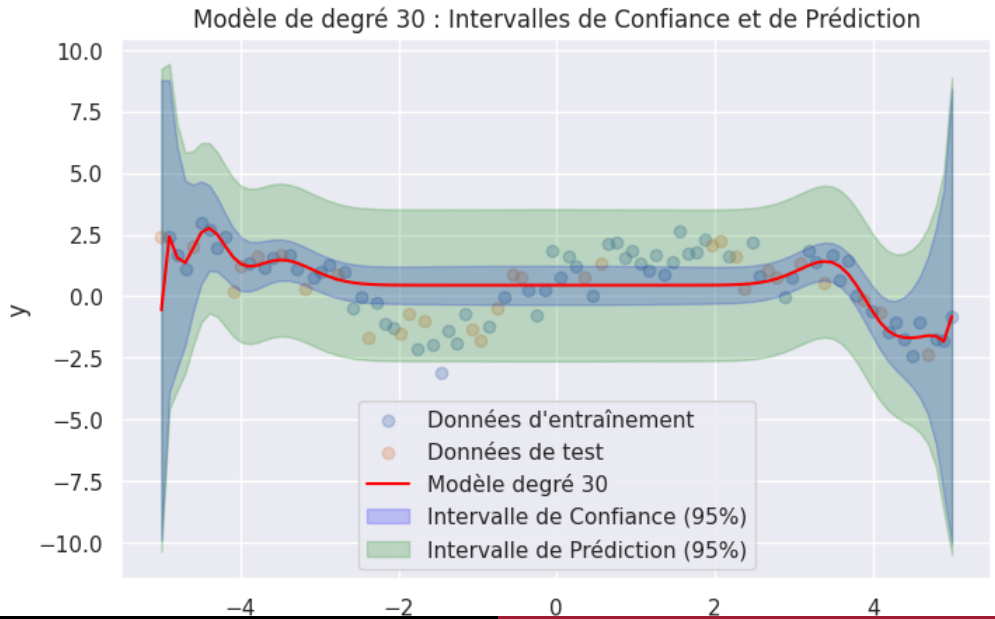
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : La courbe rouge épouse trop fidèlement les points d'entraînement, capturant même le bruit. Les intervalles de confiance sont parfois instables, en particulier aux extrémités.
  - Variance élevée : une petite variation dans les données change significativement les prédictions. Le modèle s'adapte trop aux variations du jeu d'entraînement et risque de mal généraliser sur de nouvelles données.



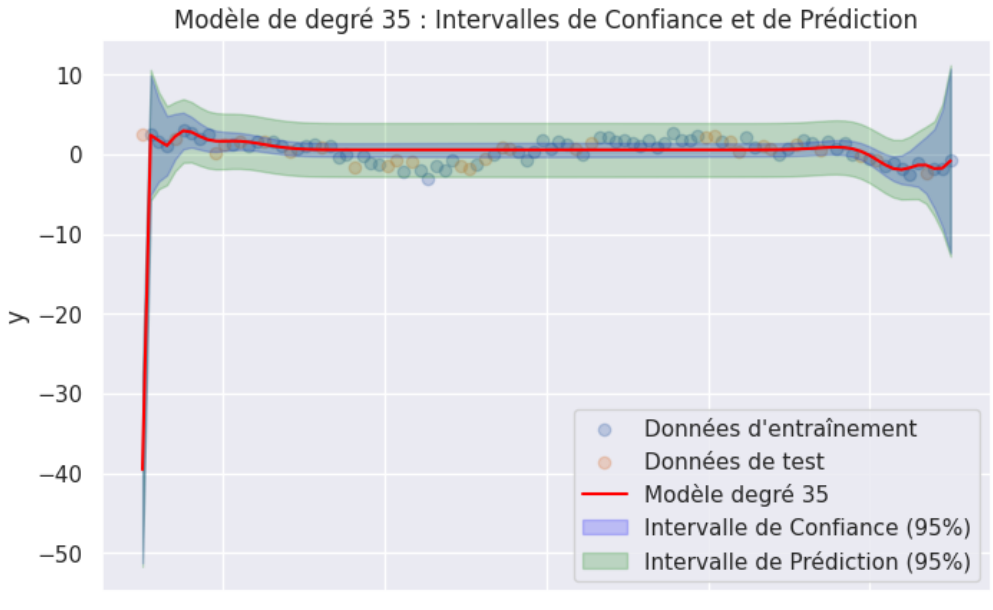
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : La courbe rouge épouse trop fidèlement les points d'entraînement, capturant même le bruit. Les intervalles de confiance sont parfois instables, en particulier aux extrémités.
  - Variance élevée : une petite variation dans les données change significativement les prédictions. Le modèle s'adapte trop aux variations du jeu d'entraînement et risque de mal généraliser sur de nouvelles données.



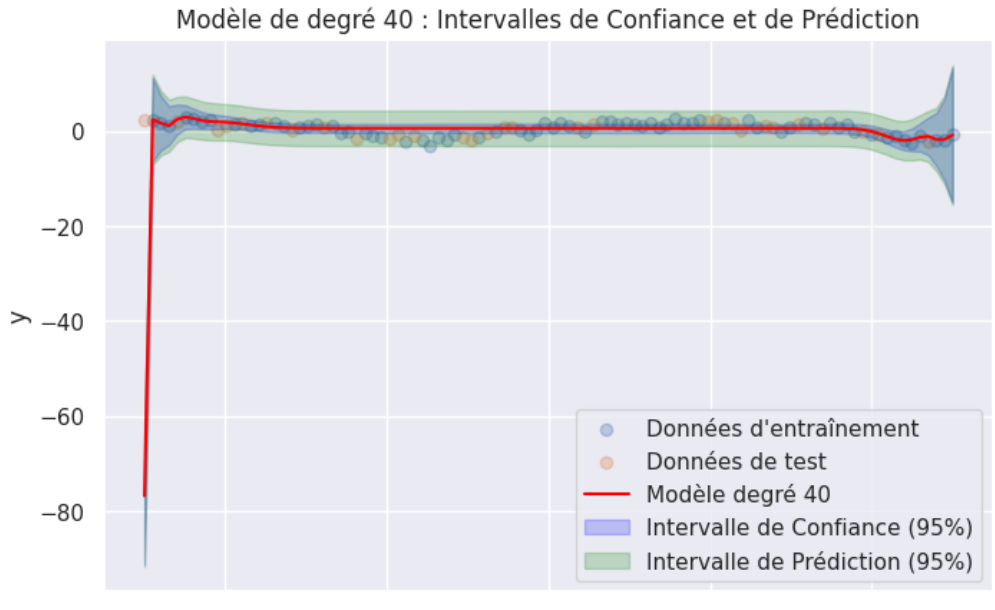
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : La courbe rouge épouse trop fidèlement les points d'entraînement, capturant même le bruit. Les intervalles de confiance sont parfois instables, en particulier aux extrémités.
  - Variance élevée : une petite variation dans les données change significativement les prédictions. Le modèle s'adapte trop aux variations du jeu d'entraînement et risque de mal généraliser sur de nouvelles données.



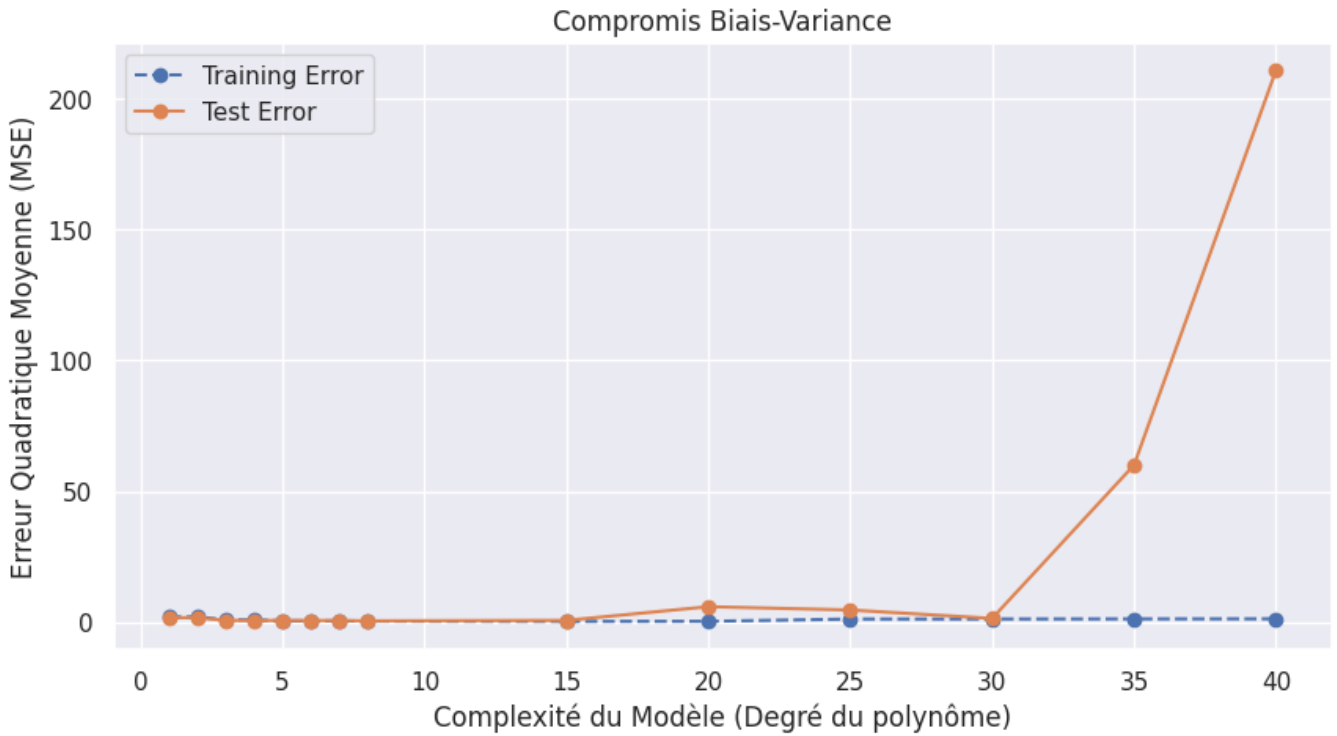
Solution : Interprétation du Compromis Biais-Variance

- **Modèle à Variance Élevée (Sur-ajustement)** :
  - Exemple : polynôme de degré trop élevé.
  - Biais faible : La courbe rouge épouse trop fidèlement les points d'entraînement, capturant même le bruit. Les intervalles de confiance sont parfois instables, en particulier aux extrémités.
  - Variance élevée : une petite variation dans les données change significativement les prédictions. Le modèle s'adapte trop aux variations du jeu d'entraînement et risque de mal généraliser sur de nouvelles données.



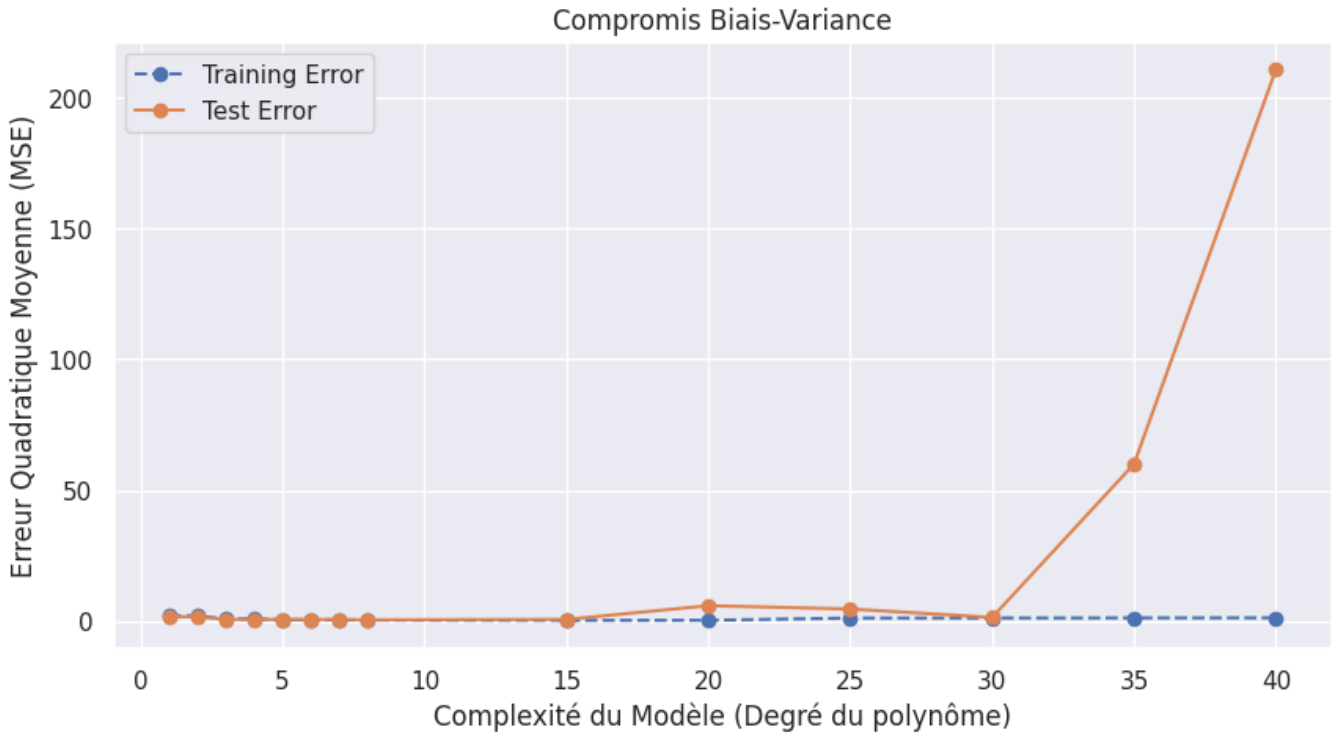
Solution : Analyse du Compromis Biais-Variance

- On observe une décroissance de l'erreur d'entraînement à mesure que le degré du polynôme augmente.
- Mais l'erreur de test diminue jusqu'à un certain seuil (degré 8), puis augmente rapidement, ce qui reflète le sur-ajustement.
- À partir de degré 25-40, la courbe devient chaotique : le modèle est trop complexe, ce qui conduit à une explosion de l'erreur de test.



Solution : Analyse du Compromis Biais-Variance

- Solutions Potentielles aux sur-ajustement :
  - **L'utilisation de validation croisée** permet d'identifier la meilleure complexité de modèle sans sur-ajuster.
  - **Ajouter de la régularisation** (ex : Ridge, Lasso) peut aussi aider à éviter le sur-ajustement.



Lien entre Estimateur Consistant et Erreur Moyenne Quadratique (EMQ)

Consistance et Erreur Moyenne Quadratique (EMQ)

- **Définition de la consistance**
  Un estimateur $\hat{\theta}_n$ d'un paramètre θ est consistant si, lorsque la taille de l'échantillon n tend vers l'infini, il converge en probabilité vers le paramètre réel θ, c'est-à-dire :

$$\hat{\theta}_n \xrightarrow{\mathbb{P}} \theta$$

  Autrement dit, pour tout $\varepsilon > 0$:

$$\lim_{n \to \infty} P(|\hat{\theta}_n - \theta| > \varepsilon) = 0.$$

- **Erreur Moyenne Quadratique (EMQ) pour un estimateur ponctuel**
  L'erreur moyenne quadratique d'un estimateur $\hat{\theta}_n$ est définie par :

$$EMQ(\hat{\theta}_n) = \mathbb{E}[(\hat{\theta}_n - \theta)^2] = \text{Var}(\hat{\theta}_n) + \text{Biais}^2(\hat{\theta}_n),$$

  où :
  - Le biais est $\text{Biais}(\hat{\theta}_n) = \mathbb{E}[\hat{\theta}_n] - \theta$.
  - La variance est $\text{Var}(\hat{\theta}_n) = \mathbb{E}[(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n])^2]$.

Lien entre Consistance et Erreur Moyenne Quadratique (EMQ)

- **Erreur Moyenne Quadratique (EMQ) pour un modèle de prédiction**
  Dans le cadre d'un modèle de prédiction, l'EMQ inclut un terme supplémentaire lié à l'erreur irréductible σ^2 :

$$\mathbb{E}_{\mathcal{D}}[(y - \hat{f}(\boldsymbol{x}))^2] = \text{Biais}^2 + \text{Variance} + \sigma^2,$$

  où σ^2 représente la variance du bruit aléatoire ϵ.
- Si un estimateur $\hat{\theta}_n$ est consistant, alors :
  - Son biais tend vers 0 : $\lim_{n\to\infty} \mathbb{E}[\hat{\theta}_n] = \theta.$
  - Sa variance tend vers 0 : $\lim_{n\to\infty} \text{Var}(\hat{\theta}_n) = 0.$
  - Donc, l'EMQ tend vers 0 : $\lim_{n\to\infty} EMQ(\hat{\theta}_n) = 0.$
- **Distinction pour les modèles de prédiction**
  Contrairement aux estimateurs ponctuels, dans un modèle de prédiction : $\lim_{n\to\infty} \mathbb{E}_{\mathcal{D}}[(y - \hat{f}(\boldsymbol{x}))^2] = \sigma^2$. Même si $\text{Biais}^2 \to 0$ et $\text{Variance} \to 0$, l'erreur irréductible σ^2 persiste (même avec un modèle parfait, il y a une erreur résiduelle due au bruit des données.).

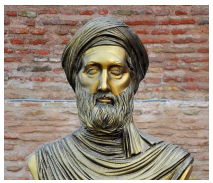
Le Principe de Parcimonie (Rasoir d'Occam)

Principe de Parcimonie : Rasoir D'Occam

- Le principe de parcimonie, ou rasoir d'Occam, est un principe fondamental en modélisation statistique et en apprentissage machine. Il stipule que :
  - **Parmi plusieurs modèles expliquant aussi bien un phénomène, il faut privilégier le plus simple** (celui qui élimine les hypothèses inutiles, d'où *la notion de rasoir.*)
- Ce principe se retrouve sous diverses formulations :

Guillaume d'Occam (XIVe siècle, 1319) :

"La multiplicité ne doit pas être posée sans nécessité."



Ibn Khaldoun (XIVe siècle, Al-Muqaddima, 1377) :

"La nature ne délaisse pas le chemin le plus court dans ses actions pour en prendre un plus compliqué et plus difficile."

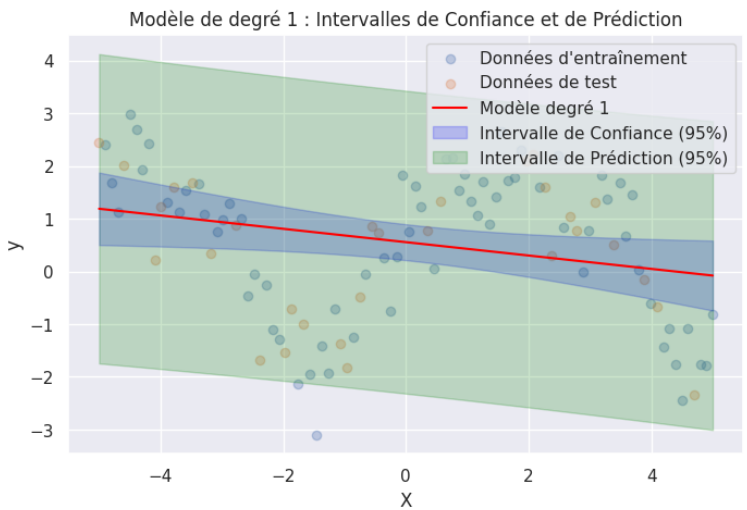
- Un modèle plus simple, **sans excès de complexité**, capturant l'essence du phénomène naturel, a souvent une meilleure capacité de généralisation.

Le Principe de Parcimonie (Rasoir d'Occam)

- Un modèle trop simple, **avec trop peu de paramètres**, ne capture pas bien la structure des données et a une performance médiocre sur l'entraînement et le test (**sous-ajustement : Biais élevé, variance faible**).
- Un modèle plus simple, sans excès de complexité, capturant l'essence du phénomène naturel, a souvent une meilleure capacité de généralisation(**un bon compromis biais-variance**).
- Un modèle plus complexe, **avec trop de paramètres,** risque de s'ajuster parfaitement aux données d'entraînement mais de mal généraliser aux nouvelles données (**surajustement : Biais faible, variance élevée**).

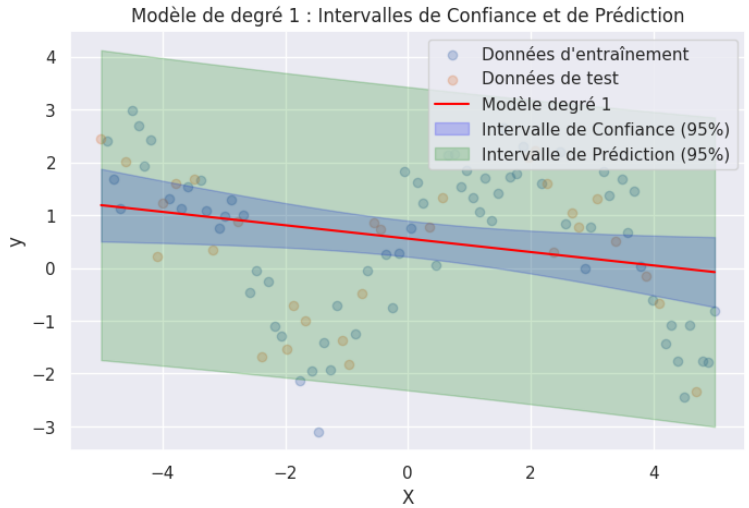
Le Principe de Parcimonie (Rasoir d'Occam)

- **Sous-ajustement - Biais élevé :** Les hypothèses émises concernant la nature de l'estimateur $\hat{f}$ sont **trop restrictives** de façon à ce que $\hat{f}$ ne peut pas s'adapter aux différentes données d'entraînement pour estimer f. On a alors un Biais$^2 = (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2$ élevé.
- **Exemple :** Pour le problème de régression polynomiale testé, les modèles linéaire et quadratique (degrés 1 et 2) ne s'ajustent pas aux données d'entraînement (sous-ajustement). Les hypothèses de linéarité et de relation quadratique sont trop restrictives pour modéliser adéquatement la tendance des données.



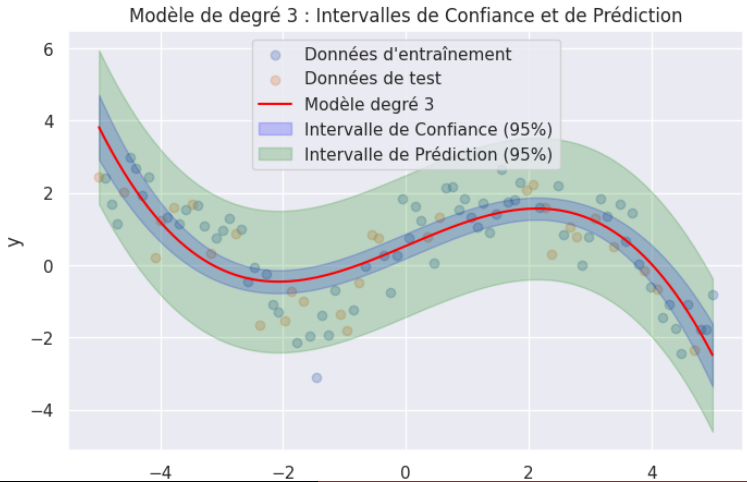
Le Principe de Parcimonie (Rasoir d'Occam)

- **Sous-ajustement - Variance faible :** Les hypothèses émises concernant la nature de l'estimateur $\hat{f}$ sont **trop restrictives** de façon à ce que $\hat{f}$ reste relativement "*rigide*" lorsque l'on change les données d'entraînement. On a alors une faible sensibilité aux variations des données : Var $= \mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]$ faible.
- **Exemple :** Dans le problème de régression polynomiale testé, les prédictions des modèles linéaire et quadratique (degrés 1 et 2) varient peu lorsqu'on modifie les données d'entraînement. Cette rigidité les empêche de s'ajuster correctement à la structure sous-jacente des données, ce qui entraîne un sous-ajustement.



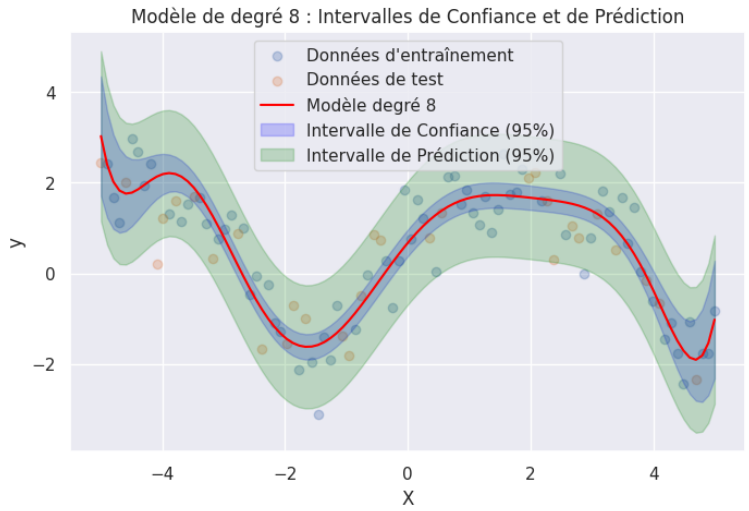
Le Principe de Parcimonie (Rasoir d'Occam)

- **Bon Compromis** - **Équilibre entre biais et variance :** Les hypothèses émises concernant la nature de l'estimateur $\hat{f}$ sont **suffisemment flexibles et robustes** de façon à ce que $\hat{f}$:
  - S'ajuste à la structure générale sous-jacente des données d'entraînement sans "absorber" le bruit aléatoire qui vient avec $\Rightarrow$ Biais$^2 = (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2$ modéré.
  - Ne varie pas de manière excessive aux variations et fuctuations des données d'entraînement $\Rightarrow$ Var $= \mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]$ modéré, ce qui garantie une bonne généralisation sur de nouvelles données.
- **Exemple :** Un modèle cubique (degré 3) représente un bon compromis : il suit la tendance des données sans les surajuster.



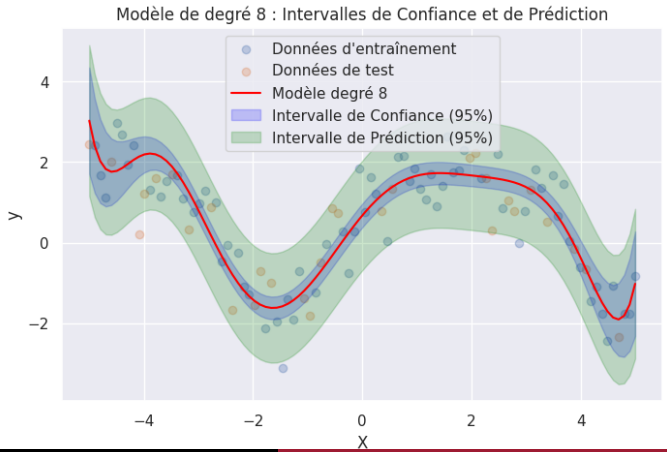
Le Principe de Parcimonie (Rasoir d'Occam)

- **Surajustement - Biais faible :** Les hypothèses émises concernant la nature de l'estimateur $\hat{f}$ sont **trop flexibles** de façon à ce que $\hat{f}$ s'ajuste excessivement au moindre détail et particularité des différentes données d'entraînement. Cela signifie que $\hat{f}$ **apprend** non seulement la tendance générale, mais aussi **le bruit aléatoire des données.** $\Rightarrow \text{Biais}^2 = (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2 \approx 0$
- Cette flexibilité excessive se fait au détriment de la généralisation.
- **Exemple :** Dans la régression polynomiale, un Les polynôme de degré ≥ 8 ont un biais faible puisqu'ils s'ajustent aux fluctuations des données.



Le Surajustement (Overfitting) - Variance Élevée

- **Surajustement - Variance élevée :** Les hypothèses émises concernant la nature de l'estimateur $\hat{f}$ sont **trop flexibles** de façon à ce que $\hat{f}$ est trop sensible aux variations des données d'entraînement. Cela veut dire que lorsqu'on change légèrement les données d'entraînement, la prédiction $\hat{f}(\boldsymbol{x})$ change drastiquement pour s'ajuster aux particularités des données
$\Rightarrow$ Var $= \mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]$ élevé. **manque de robustesse.**
- **Exemple :** Dans la régression polynomiale, la prédiction des polynômes de degré ≥ 8, $\hat{f}(\boldsymbol{x})$ varie considérablement lorsque l'on modifie les données d'entraînement $\mathcal{D}$, ce qui nuit à leurs capacités respectives de généraliser.



Le Principe de Parcimonie et Lien avec l'EMQ

- L'erreur moyenne quadratique (EMQ) se décompose en trois termes :

$$\underbrace{\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2]}_{EMQ} = \underbrace{(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2}_{\text{Biais}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Erreur Irréductible}}.$$

où :

- **Biais**^{2} : Erreur systématique due à la mauvaise spécification du modèle.
- **Variance** : Sensibilité du modèle aux variations de l'échantillon d'entraînement.
- σ^2 (**erreur irréductible**) : Bruit aléatoire dans les données.

Impact sur chaque Régime de Complexité

- **A. Si le modèle est sous-ajusté**
  - $EMQ = \underbrace{(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2}_{\text{Biais}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Erreur Irréductible}}$.
  - Un modèle sous-ajusté fait des hypothèses trop restrictives et son biais (Biais^2) reste **élevé**, même quand $n \to \infty$

$$\lim_{n \to \infty} (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2 \neq 0.$$

  - Son biais ne disparaît pas, car il est dû à une mauvaise spécification du modèle.
  - Sa variance est faible à cause de la régidité des hypothèses émises concernant $\hat{f}$ et elle tend vers 0. Le modèle cependant n'apprend pas la structure réelle des données à cause de sa régidité.
- **Conséquence** : Même avec une infinité de données, le modèle sous-ajusté reste médiocre, **car l'erreur quadratique moyenne EMQ est dominée par un Biais^2 élevé.**

Impact sur chaque Régime de Complexité

- **B. Si le modèle est bien équilibré (Bon Compromis Biais-Variance)**
  - $EMQ = \underbrace{(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2}_{\text{Biais}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Erreur Irréductible}}$.
  - Un bon modèle voit son biais (Biais^2) et sa variance **diminuer** au fur et à mesure que $n \to \infty$.

$$\begin{cases} \lim_{n\to\infty} (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2 = 0. \\ \lim_{n\to\infty} \mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2] = 0. \end{cases}$$

  - Son erreur totale se stabilise autour de σ^2 (**l'erreur irréductible**).
  - C'est le comportement idéal, car il montre que le modèle apprend bien la structure des données et généralise bien.
- **Conséquence** : Le modèle généralisera toujours mieux avec plus de données et atteindra la limite optimale $\lim_{n\to\infty} EMQ = \sigma^2$.

Impact sur chaque Régime de Complexité

- **C. Si le modèle est surajusté**
  - $EMQ = \underbrace{(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2}_{\text{Biais}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Erreur Irréductible}}$.
  - Le modèle s'ajuste trop bien aux petites variations et fluctuations des données d'entraînement de façon à ce que le Biais^2 diminue au fur et à mesure que $n \to \infty$.
  - La variance peut rester élevée, même quand $n \to \infty$, si le modèle est trop complexe puisque cette complexité cause la sensibilité aux petites variations des données d'entraînement.
  - La flexibilité et complexité des hypothèses du modèles sont au détriment de la généralisation.

$$\begin{cases} \lim_{n\to\infty}(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2 = 0. \\ \lim_{n\to\infty} \mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2] \neq 0. \end{cases}$$

- **Conséquence** :
  - Si la complexité est trop élevée, le modèle ne généralisera jamais.
  - Si le modèle est bien régularisé, la variance diminuera avec $n \to \infty$ et l'erreur totale tendra EMQ vers σ^2.

Le Principe de Parcimonie (Rasoir d'Occam) : Conclusion

- La parcimonie consiste à :
  - Expliquer les données avec un minimum de complexité.
  - Cela tout en assurant une bonne capacité de généralisation.
- Un bon modèle cherche un équilibre entre ajustement (précision) et simplicité, ce qui se fait en combinant plusieurs approches :
  - **Les méthodes de régularisation**, qui pénalisent la variance excessive et limitent ainsi la complexité du modèle.
  - **L'évaluation de la performance sur des données autres que celles d'entraînement** à l'aide de la validation, ou même, la validation croisée (*k-fold cross-validation*).
  - **L'utilisation de critères de sélection de modèles**, tels que :
    - AIC (Akaike Information Criterion), qui pénalise la complexité excessive tout en favorisant un bon ajustement aux données.
    - BIC (Bayesian Information Criterion), qui impose une pénalisation plus stricte pour éviter le surajustement.

Le principe de Parsimonie en IA Aujourd'hui

- Traditionnellement en statistique, on cherche à limiter la complexité des modèles pour éviter le surajustement.
- Avec l'avènement des lois d'échelle neuronales *Neural Scaling Laws*, on observe que des modèles plus grands généralisent souvent mieux, **à condition d'être entraînés sur de vastes quantités de données.**
- Aujourd'hui, la parcimonie signifie qu'il faudrait **optimiser l'efficacité des modèles complexes** :
  - **Techniques de régularisation** dans les éléments constitutifs des modèles (*dropout*, *weight decay*, *batch normalization*) pour limiter la complexité inutile.
  - **Compression post-entraînement** (*pruning*, *quantization*) pour réduire la taille sans perte de performance.
  - **Fine-tuning efficace** pour adapter un modèle massif à des tâches spécifiques sans en réentraîner un nouveau.
- Ainsi, la **théorie de la décision en IA évolue** : On accepte la complexité si elle améliore systématiquement la généralisation et la robustesse.

Table des Matières

Méthodes de Régularisation

Méthodes de Régularisation : Introduction

Méthodes de Régularisation : Introduction

- La régularisation agit directement sur la **variance de l'estimateur** $\hat{f}$, en limitant l'espace de recherche des estimateurs $\hat{\boldsymbol{\beta}}$.
- Elle impose une **contrainte sur la norme** l_p **des coefficients** $\boldsymbol{\beta}$ afin d'empêcher le modèle d'être trop sensible aux fluctuations des données d'entraînement.

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2$$

sous la contrainte :

$$\|\boldsymbol{\beta}\|_p^p = \sum_{j=1}^{p} |\beta_j|^p \leq C,$$

où C est un hyperparamètre qui contrôle la complexité du modèle.

- Le choix de la norme l_p influence la structure de $\hat{\boldsymbol{\beta}}$:
  - $p = 2$ (Ridge) impose une pénalisation quadratique et empêche des valeurs extrêmes.
  - $p = 1$ (LASSO) impose une pénalisation absolue, favorisant la sélection de variables en mettant certains coefficients à 0.

Méthodes de Régularisation : Introduction - Rappel sur les Normes

Rappel : Visualisation des Normes ℓ_1, ℓ_2 et ℓ_∞

- Les normes ℓ_1, ℓ_2, et ℓ_∞ peuvent être visualisées sous forme de sphères unités (ensemble des points $\mathbf{x} \in \mathbb{R}^2$ tels que $\|\mathbf{x}\| = 1$) :
  - La norme ℓ_1 privilégie les déplacements alignés avec les axes, ce qui se traduit par un losange.
  - La norme ℓ_2, la plus courante, mesure les distances euclidiennes. La sphère d'unité est lisse et circulaire, représentant un poids égal dans toutes les directions (cercle).
  - La norme ℓ_∞ prend la valeur maximale des composantes, donnant lieu à un carré.

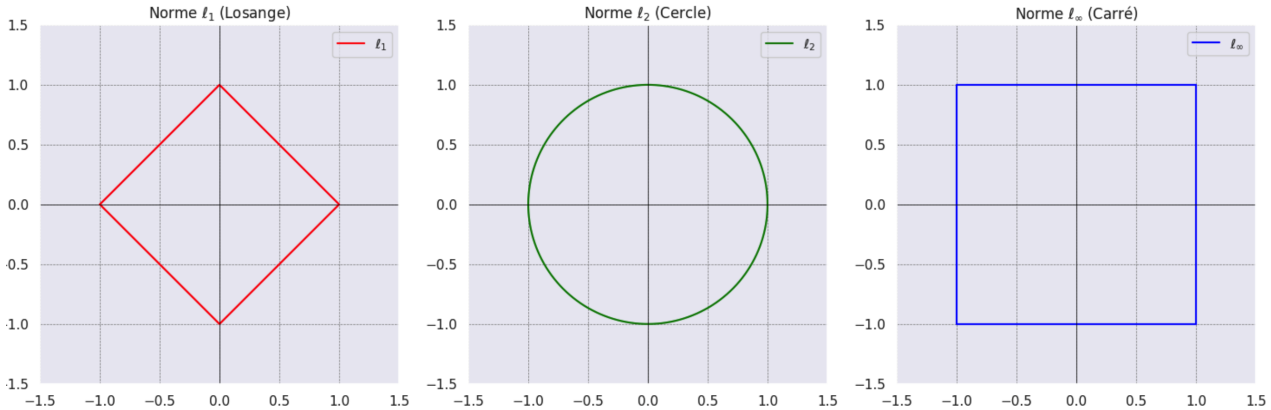


Figure: Représentation des normes ℓ_1 (losange), ℓ_2 (cercle), et ℓ_∞ (carré).

Rappel : Norme ℓ_1 (Norme Manhattan)

- **Norme ℓ_1 (Norme Manhattan)** :
  - **Interprétation** : Mesure la distance comme si l'on se déplaçait à travers un réseau orthogonal, comme les rues d'une ville (Manhattan).
  - **Application** : Utilisée dans le modèle de régression Lasso pour exprimer la régularisation L_1. Elle permet d'obtenir des paramètres éparses.
  - **Propriété** : Robuste aux données éparses (sparse data), elle encourage les solutions où de nombreux paramètres sont nuls. Elle effectue une sélection des caractéristiques en éliminant celles qui contribuent le moins au modèle.

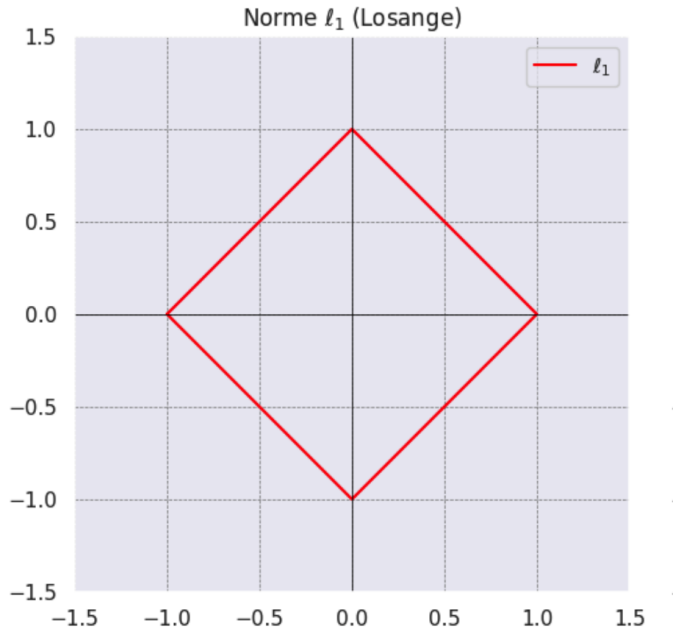


Figure: Norme ℓ_1 : Sphère unité (forme de losange)

Rappel : Norme ℓ_2 (Norme Euclidienne)

- **Norme ℓ_2 (Norme Euclidienne)** :
  - **Interprétation** : mesure la distance droite (ou longueur) entre deux points dans un espace euclidien. C'est la norme classique que l'on utilise intuitivement pour mesurer une distance.
  - **Application** : Utilisée dans la régression Ridge utilise pour exprimer régularisation L_2 pour pénaliser les valeurs extrêmes des estimateurs.
  - **Propriété** : Utilisée pour la régularisation L_2, elle ne réalise pas de sélection de caractéristiques, mais équilibre les contributions de toutes les caractéristiques en les réduisant proportionnellement.

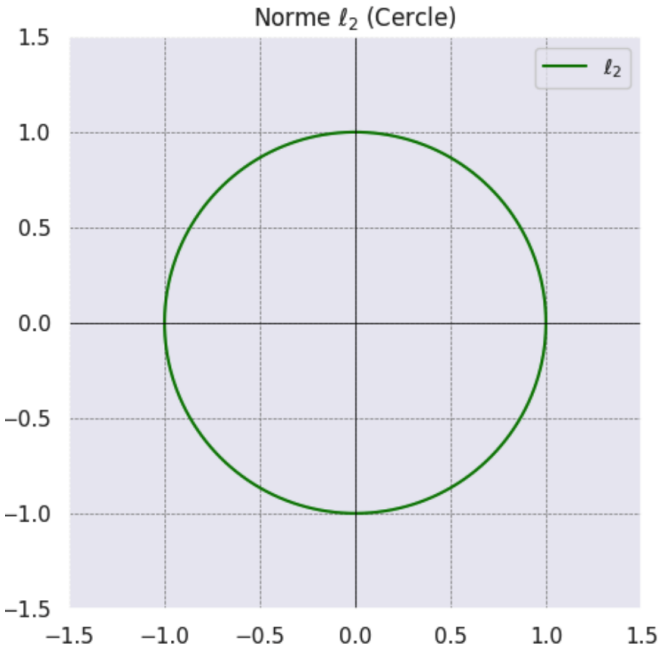


Figure: Norme ℓ_2 : Sphère unité (forme de cercle)

Méthodes de Régularisation : Introduction - Fin du Rappel sur les Normes

Interprétation Géométrique de la Régularisation

- L'ajout d'une contrainte sur la norme l_p des coefficients $\boldsymbol{\beta}$ modifie la manière dont les solutions optimales sont trouvées.
- Cette contrainte influence directement l'espace des solutions admissibles, impactant ainsi la sélection des paramètres et la structure du modèle.
- Sans contrainte de régularisation, on cherche la solution qui minimise l'erreur quadratique :

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2.$$

- **Interprétation géométrique** :
  - La solution optimale correspond à la projection orthogonale de $\mathbf{Y}$ sur le sous-espace engendré par les colonnes de $\mathbf{X}$.
  - Si $\mathbf{X}^T\mathbf{X}$ est inversible :

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}.$$

  - **Problème** : Lorsque les colonnes de $\mathbf{X}$ sont fortement corrélées (*multicolinéarité*), $\mathbf{X}^T\mathbf{X}$ devient mal conditionnée, ce qui entraîne une solution instable avec une variance élevée.

Interprétation Géométrique de la Régularisation

- L'ajout d'une contrainte sur la norme l_p des coefficients $\boldsymbol{\beta}$ modifie la manière dont les solutions optimales sont trouvées.
- Cette contrainte influence directement l'espace des solutions admissibles, impactant ainsi la sélection des paramètres et la structure du modèle.
- Sans contrainte de régularisation, on cherche la solution qui minimise l'erreur quadratique :

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2.$$

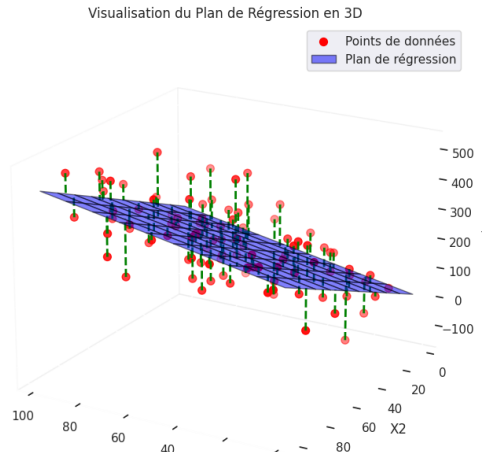
- **Interprétation géométrique** :
  - La solution optimale correspond à la projection orthogonale de $\mathbf{Y}$ sur le sous-espace engendré par les colonnes de $\mathbf{X}$.
  - Si $\mathbf{X}^T\mathbf{X}$ est inversible :

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}.$$

  - La solution des moindres carrés ordinaires (OLS) $\hat{\boldsymbol{\beta}}$ permet d'obtenir une estimation de $\mathbf{Y}$ qui appartient toujours à $\mathcal{C}(\mathbf{X})$, c'est-à-dire l'espace engendré par les colonnes de $\mathbf{X}$.

Rappel - Interprétation Géométrique des MCO

- **Interprétation Géométrique :**
  - **Projection orthogonale sur le sous-espace engendré par les prédicteurs:**
  - L'hyperplan en bleu représente la **régression linéaire multiple**, c'est-à-dire le sous-espace sur lequel les valeurs prédites $\hat{y}$ sont projetées ($\hat{\boldsymbol{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$).
  - Les **lignes verticales vertes pointillées** représentent les **résidus** $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$, c'est-à-dire la différence entre les points de données réels (rouges) et leurs projections sur l'hyperplan (prédictions).
  - La **minimisation des moindres carrés** ajuste cet hyperplan de manière à minimiser la somme des carrés des longueurs de ces segments.



Interprétation Géométrique de la Régularisation

- **Interprétation géométrique** :
  - Le sous-espace engendré par ces colonnes est l'ensemble de toutes les combinaisons linéaires possibles de ces vecteurs (colonnes) :

$$\mathcal{C}(\mathbf{X}) = \left\{ \mathbf{X}\boldsymbol{\beta} \mid \boldsymbol{\beta} \in \mathbb{R}^{p+1} \right\}.$$

  - Autrement dit, on projette $\mathbf{Y}$ sur cet espace pour minimiser l'erreur quadratique :

$$\hat{\mathbf{Y}} = \mathbf{P}_X \mathbf{Y} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}.$$

  - Cette projection est la meilleure approximation possible de $\mathbf{Y}$ dans $\mathcal{C}(\mathbf{X})$ en termes de norme L_2, c'est-à-dire qu'elle minimise l'erreur quadratique : $\|\mathbf{Y} - \hat{\mathbf{Y}}\|^2$.
  - **Intuition géométrique** :
    - Si les colonnes de $\mathbf{X}$ sont linéairement indépendantes, alors $\mathcal{C}(\mathbf{X})$ est un espace vectoriel de dimension égale au nombre de colonnes de $\mathbf{X}$.
    - Si elles sont dépendantes, alors certaines colonnes sont redondantes, et $\dim(\mathcal{C}(\mathbf{X}))$ est strictement inférieure au nombre de colonnes de $\mathbf{X}$.
  - **Problème** : Lorsque les colonnes de $\mathbf{X}$ sont fortement corrélées (*multicolinéarité*), $\mathbf{X}^T\mathbf{X}$ devient mal conditionnée, ce qui entraîne une solution instable avec **une variance élevée.**

Interprétation Géométrique de la Régularisation

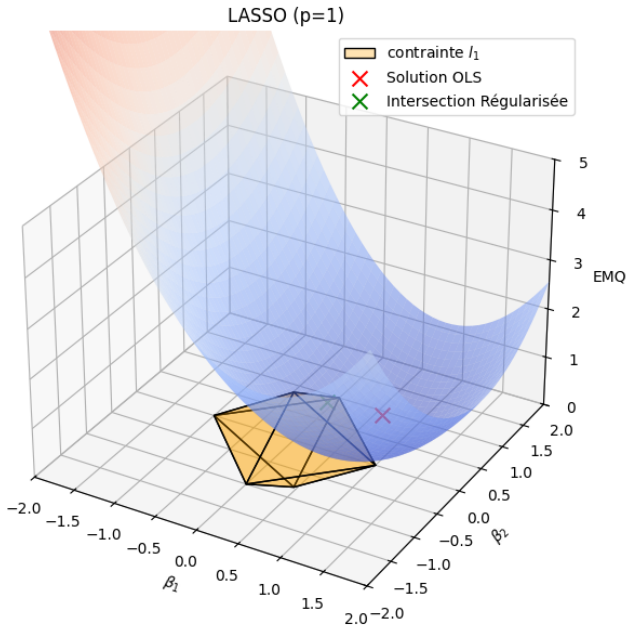
- En ajoutant une contrainte sur la norme l_p, la solution optimale doit être contenue dans une région géométrique spécifique :

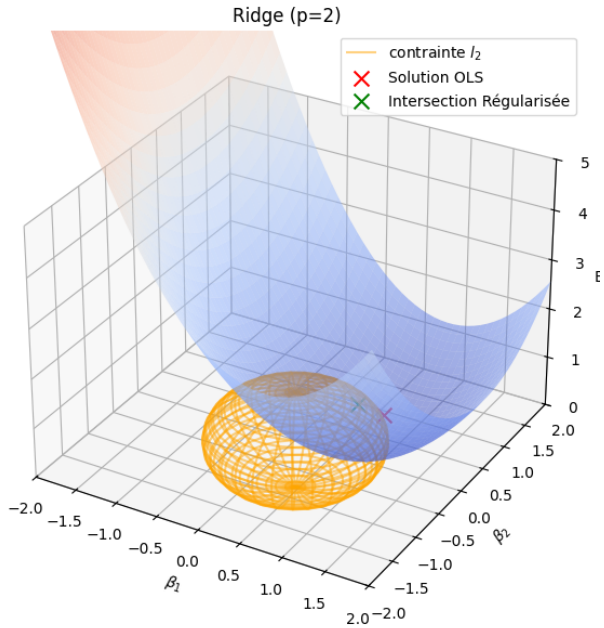
$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2, \quad \text{sous la contrainte} \quad \|\boldsymbol{\beta}\|_p^p \leq C.$$

- **Interprétation géométrique** :
  - La contrainte définit une région admissible où la solution doit se situer.
  - La solution optimale est à l'intersection entre l'ellipse des niveaux de l'erreur quadratique et la région admissible $\|\boldsymbol{\beta}\|_p^p \leq C$.

Interprétation Géométrique de la Régularisation

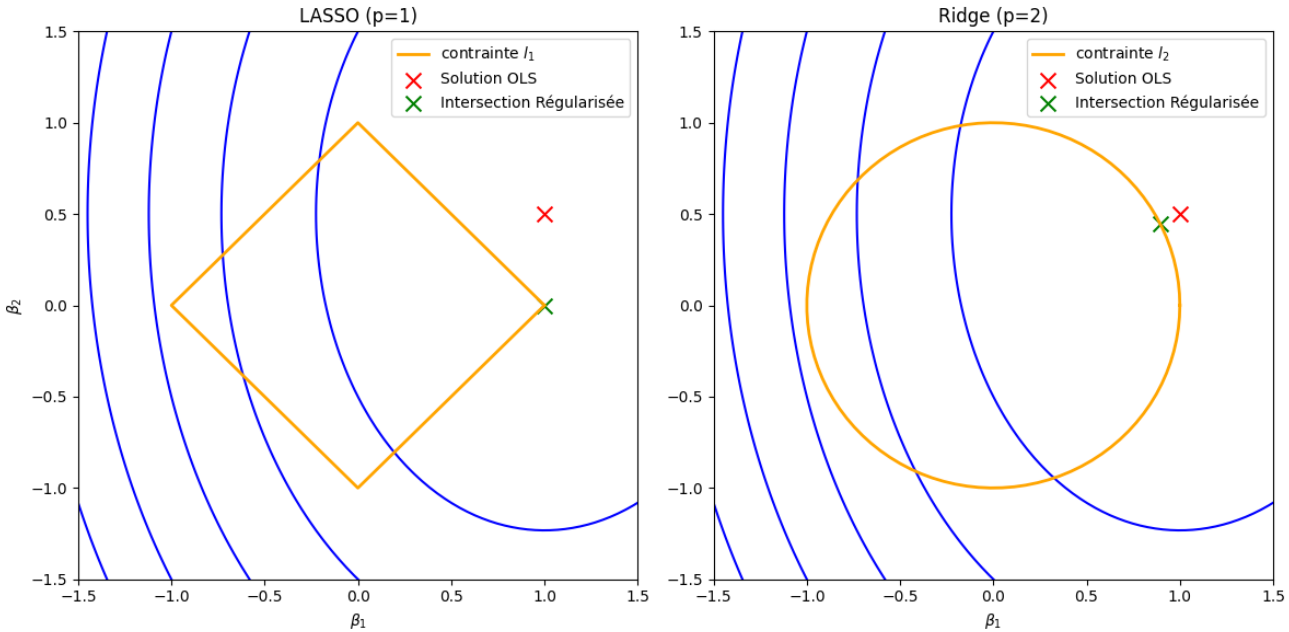
- La contrainte l_1 définit une région admissible sous la forme d'un **octaèdre** (généralisation du losange en 3D).
- L'intersection entre la fonction de coût et cet octaèdre se fait souvent sur un **sommet**, ce qui entraîne la mise à zéro de certains coefficients β_j (Les coins du losange sont les premiers points d'intersection).
- **Conséquence** : Le modèle sélectionne automatiquement certaines variables en supprimant les autres, conduisant à une solution **éparse**.





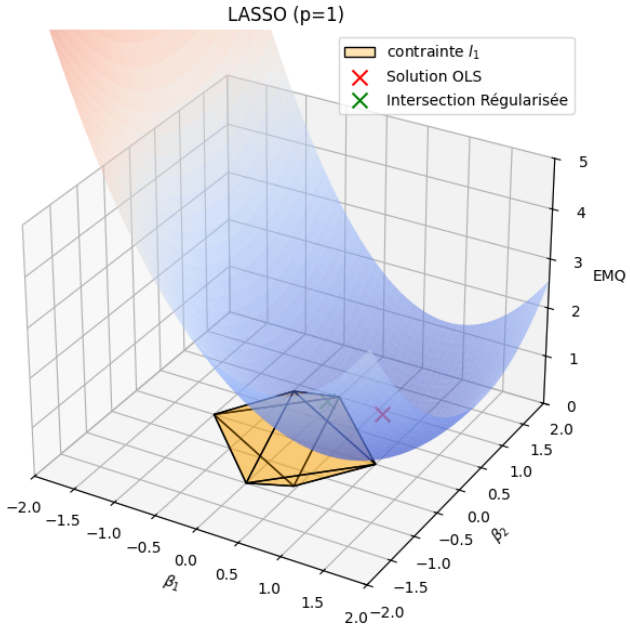
Interprétation Géométrique de la Régularisation

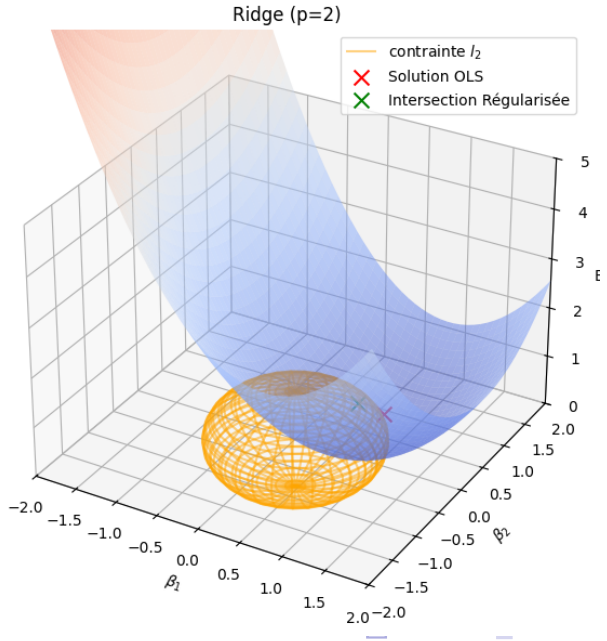
- La contrainte l_1 définit une région admissible sous la forme d'un **octaèdre** (généralisation du losange en 3D).
- L'intersection entre la fonction de coût et cet octaèdre se fait souvent sur un **sommet**, ce qui entraîne la mise à zéro de certains coefficients β_j (Les coins du losange sont les premiers points d'intersection).
- **Conséquence** : Le modèle sélectionne automatiquement certaines variables en supprimant les autres, conduisant à une solution **éparse**.



Interprétation Géométrique de la Régularisation

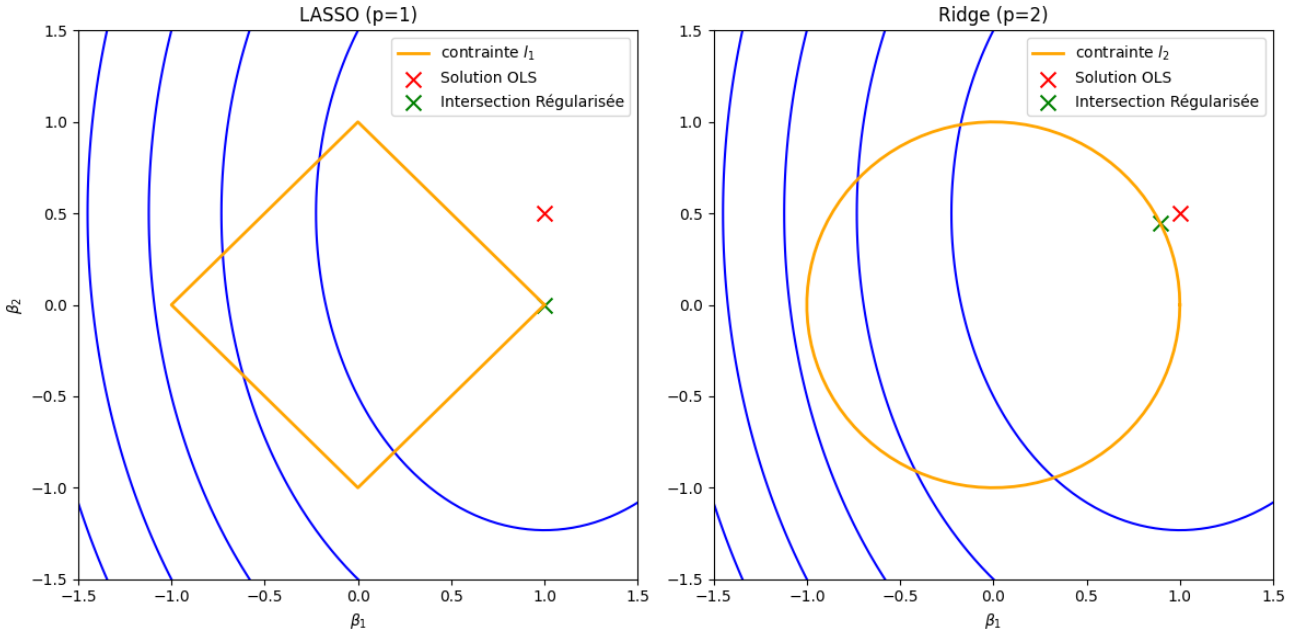
- La contrainte l_2 impose une région admissible en forme de **sphère**.
- L'intersection entre la sphère et l'ellipsoïde des moindres carrés se fait en un point quelconque de la sphère, **sans annulation stricte** des coefficients.
- **Conséquence** : La norme l_2 réduit tous les coefficients sans en mettre certains exactement à zéro, ce qui permet de réduire la variance du modèle sans pour autant effectuer de sélection de variables.





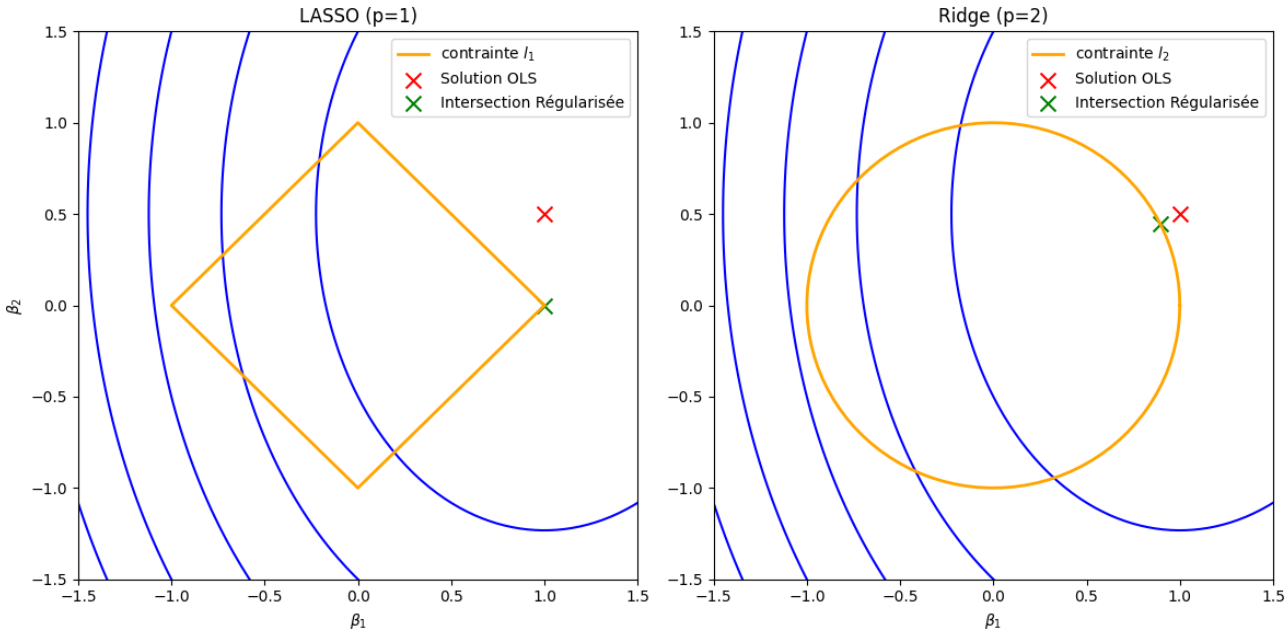
Interprétation Géométrique de la Régularisation

- La contrainte l_2 impose une région admissible en forme de **sphère**.
- L'intersection entre la sphère et l'ellipsoïde des moindres carrés se fait en un point quelconque de la sphère, **sans annulation stricte** des coefficients.
- **Conséquence** : La norme l_2 réduit tous les coefficients sans en mettre certains exactement à zéro, ce qui permet de réduire la variance du modèle sans pour autant effectuer de sélection de variables.



Interprétation Géométrique de la Régularisation

- La régularisation contraint la solution des moindres carrés (point rouge) à s'ajuster à la contrainte définie (surface jaune).
- La solution optimale régularisée (point vert) est déplacée par rapport à l'OLS en fonction de la contrainte.
- **LASSO** favorise les solutions **éparses** (*sparse learning*) en supprimant certaines variables inutiles.
- **Ridge** préserve toutes les variables en leur appliquant une **réduction de l'amplitudes des coefficients**.



Problème Dual et Multiplicateurs de Lagrange

- Considérons un problème d'optimisation sous contrainte :

$$\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2, \quad \text{sous la contrainte} \quad \|\boldsymbol{\beta}\|_p^p \leq C.$$

- La contrainte $\|\boldsymbol{\beta}\|_p^p \leq C$ **limite l'espace des solutions admissibles**.
- Une alternative est d'ajouter un **terme de pénalisation** qui pénalise les solutions qui violent cette contrainte.
- L'idée clé est d'utiliser la **dualité forte** pour reformuler ce problème sous une forme équivalente **sans contrainte**.

Problème Dual et Multiplicateurs de Lagrange

- La **dualité forte** stipule que, sous certaines conditions (convexité, contraintes qualifiées), le problème sous contrainte et sa formulation duale ont la **même solution optimale**.
- Nous introduisons un **multiplicateur de Lagrange** $\lambda \geq 0$ pour transformer la contrainte en un terme de pénalisation :

$$\mathcal{L}(\boldsymbol{\beta}, \lambda) = \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda(\|\boldsymbol{\beta}\|_p^p - C).$$

- Deux cas possibles :
  - Si $\|\boldsymbol{\beta}\|_p^p < C$, alors $\lambda = 0$.
  - Si $\|\boldsymbol{\beta}\|_p^p = C$, alors $\lambda > 0$ agit comme un facteur de pénalisation pour éviter que $\|\boldsymbol{\beta}\|_p^p$ dépasse C.
- La **condition de complémentarité** de Karush-Kuhn-Tucker (KKT) :

$$\lambda \cdot (\|\boldsymbol{\beta}\|_p^p - C) = 0.$$

Problème Dual et Multiplicateurs de Lagrange

- La dualité forte permet de reformuler le problème comme suit :

$$\max_{\lambda \geq 0} \min_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}, \lambda).$$

- **Interprétation** :
  - Pour chaque λ, on trouve la **meilleure solution** $\boldsymbol{\beta}$ qui minimise $\mathcal{L}(\boldsymbol{\beta}, \lambda)$.
  - Ensuite, on résout le problème dual en maximisant la fonction obtenue par rapport à λ.
  - Cela permet de trouver un équilibre entre **minimisation de l'erreur quadratique** et **régularisation**.
- Résoudre le problème dual en maximisant $\min_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}, \lambda)$ par rapport à λ revient à absorber la contrainte dans la fonction objectif :

$$\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_p^p.$$

Problème Dual et Multiplicateurs de Lagrange

- Lorsque nous résolvons : $\max_{\lambda \geq 0} \min_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}, \lambda)$,
- Nous ajustons le compromis entre :
  - **Minimisation de l'erreur d'apprentissage** : $\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2$.
  - **Régularisation** : $\|\boldsymbol{\beta}\|_p^p$.
- **Effet du choix de λ :**
  - λ **faible** : peu de régularisation, solution proche de OLS.
  - λ **élevé** : forte pénalisation des coefficients, valeurs de $\boldsymbol{\beta}$ réduites.
- La régularisation permet de réduire le **surajustement** en limitant la complexité du modèle.
- En théorie, la dualité forte nous indique que la meilleure valeur de λ est obtenue en maximisant la fonction duale.
- En pratique, λ est sélectionné par **validation croisée** pour optimiser la généralisation.

Méthodes de Régularisation : La Régularisation L_2 Ridge

Exercice : Solution Analytique de la Régularisation L_2 Ridge

La Régularisation L_2 Ridge : Exercice

- La régression Ridge est une extension de la régression linéaire classique où l'on ajoute un terme de régularisation L_2 pour éviter le surajustement et stabiliser l'estimation des coefficients.
- On cherche à estimer le vecteur de paramètres $\boldsymbol{\beta}$ en minimisant la fonction de coût suivante :

$$C(\boldsymbol{\beta}) = \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2.$$

où :

- $\mathbf{X} \in \mathbb{R}^{n\times(p+1)}$ est la matrice des variables explicatives (avec une colonne de 1 pour le biais),
- $\mathbf{Y} \in \mathbb{R}^n$ est le vecteur des observations,
- $\boldsymbol{\beta} \in \mathbb{R}^{(p+1)}$ est le vecteur des paramètres à estimer,
- $\lambda \geq 0$ est un hyperparamètre qui contrôle le degré de régularisation.

La Régularisation L_2 Ridge : Exercice

1. **Développement de la fonction de coût**
   - Exprimez $\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$ sous forme matricielle et développez son expression.
   - Exprimez également le terme de régularisation $\lambda\|\boldsymbol{\beta}\|_2^2$ sous forme matricielle.
   - Écrivez l'expression complète de la fonction de coût $C(\boldsymbol{\beta})$.
2. **Calcul du gradient**
   - Rappelez la règle de dérivation pour une forme quadratique $\mathbf{x}^\top \mathbf{A}\mathbf{x}$ où $\mathbf{A}$ est une matrice symétrique.
   - En utilisant cette règle, calculez le gradient de $C(\boldsymbol{\beta})$ par rapport à $\boldsymbol{\beta}$.
   - Posez l'équation du gradient à zéro pour obtenir une condition d'optimalité.
   - En déduire l'expression explicite de $\boldsymbol{\beta}_{\text{Ridge}}$.

- On rappelle les règles suivantes de dérivation :
  - Si $\mathbf{A}$ est une matrice symétrique, alors : $\frac{\partial}{\partial \mathbf{x}} \left(\mathbf{x}^\top \mathbf{A}\mathbf{x}\right) = 2\mathbf{A}\mathbf{x}$.
  - Le carré de la norme l_2 d'un vecteur $\boldsymbol{\alpha}$ peut s'écrire sous forme matricielle : $\|\boldsymbol{\alpha}\|_2^2 = \boldsymbol{\alpha}^\top \boldsymbol{\alpha}$.
  - Si $\mathbf{A}$ est une matrice, alors $\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}^\mathrm{T}\mathbf{A}\mathbf{b}) = \mathbf{A}\mathbf{b}$

La Régularisation L_2 Ridge : Exercice

Régression Ridge : Solution

La Régularisation L_2 Ridge : Solution

- La Ridge Regression est une extension de la régression linéaire avec un terme de régularisation L_2.
- Son objectif est de minimiser la fonction de coût suivante :

$$C(\boldsymbol{\beta}) = \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2.$$

- **Développement du terme d'erreur quadratique** :

$$\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}).$$

- En développant :

$$(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{Y}^\top\mathbf{Y} - 2\boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{Y} + \boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta}.$$

- **Développement du terme de régularisation** :

$$\lambda\|\boldsymbol{\beta}\|_2^2 = \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta}.$$

- **Expression complète de la fonction de coût** :

$$C(\boldsymbol{\beta}) = \mathbf{Y}^\top\mathbf{Y} - 2\boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{Y} + \boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta}.$$

La Régularisation L_2 Ridge : Solution

- Pour obtenir la solution analytique, nous dérivons la fonction de coût par rapport à $\boldsymbol{\beta}$ et posons le gradient à zéro.
- **Calcul des dérivées des termes** :
  - Premier terme $\mathbf{Y}^\top \mathbf{Y}$:

$$\frac{\partial}{\partial \boldsymbol{\beta}} (\mathbf{Y}^\top \mathbf{Y}) = 0.$$

  - Deuxième terme $-2\boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{Y}$:

$$\frac{\partial}{\partial \boldsymbol{\beta}} (-2\boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{Y}) = -2\mathbf{X}^\top \mathbf{Y}.$$

  - Troisième terme $\boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\beta}$:

$$\frac{\partial}{\partial \boldsymbol{\beta}} (\boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\beta}) = 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\beta}.$$

  - Quatrième terme $\lambda\boldsymbol{\beta}^\top \boldsymbol{\beta}$:

$$\frac{\partial}{\partial \boldsymbol{\beta}} (\lambda\boldsymbol{\beta}^\top \boldsymbol{\beta}) = 2\lambda\boldsymbol{\beta}.$$

La Régularisation L_2 Ridge : Solution

- En combinant les termes :

$$2\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^\top\mathbf{Y} + 2\lambda\boldsymbol{\beta} = 0.$$

- On simplifie en divisant par 2 :

$$\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta} = \mathbf{X}^\top\mathbf{Y}.$$

- Nous factorisons $\boldsymbol{\beta}$:

$$(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta} = \mathbf{X}^\top\mathbf{Y}.$$

- Où $\mathbf{I}$ est la matrice identité de dimension $(p+1) \times (p+1)$.
- On inverse la matrice pour obtenir la solution analytique :

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{Y}.$$

La Régularisation Ridge Assure l'Absence de Multicolinéarité

Ridge Assure l'Absence de Multicolinéarité

- La matrice $\mathbf{X}^\top\mathbf{X}$ est symétrique semi-définie positive (SDP), donc elle admet une décomposition spectrale :

$$\mathbf{X}^\top\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^\top, \quad \text{où :}$$

  - $\mathbf{U}$ est une matrice orthogonale ($\mathbf{U}^\top\mathbf{U} = \mathbf{I} = \mathbf{U}\mathbf{U}^\top$).
  - $\boldsymbol{\Sigma}$ est une matrice diagonale contenant les valeurs propres $\sigma_1, \sigma_2, \ldots, \sigma_p$ de $\mathbf{X}^\top\mathbf{X}$.
- Considérons la matrice modifiée :

$$\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}.$$

- En utilisant la base propre $\mathbf{U}$:

$$\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^\top + \lambda\mathbf{U}\mathbf{I}\mathbf{U}^\top.$$

- Puisque $\mathbf{U}^\top\mathbf{U} = \mathbf{I} = \mathbf{U}\mathbf{U}^\top$, on peut réécrire :

$$\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I} = \mathbf{U}(\boldsymbol{\Sigma} + \lambda\mathbf{I})\mathbf{U}^\top.$$

Ridge Assure l'Absence de Multicolinéarité

- On a $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I} = \mathbf{U}(\mathbf{\Sigma} + \lambda\mathbf{I})\mathbf{U}^\top$.
- Comme $\mathbf{\Sigma}$ est diagonale, alors $\mathbf{\Sigma} + \lambda\mathbf{I}$ l'est aussi.
- Les nouvelles valeurs propres deviennent :

$$\sigma_i' = \sigma_i + \lambda, \quad \forall i.$$

- Toutes les valeurs propres de $\mathbf{X}^\top\mathbf{X}$ sont donc augmentées de λ, ce qui évite qu'elles soient trop petites.
- **Amélioration du conditionnement** :

$$\kappa(\mathbf{X}^\top\mathbf{X}) = \frac{\sigma_{\max}}{\sigma_{\min}}, \quad \kappa(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}) = \frac{\sigma_{\max} + \lambda}{\sigma_{\min} + \lambda}.$$

- Si $\lambda > 0$, alors les petites valeurs propres sont augmentées, ce qui assure que $\sigma_{\min} + \lambda > 0$ et améliore donc la stabilité numérique.
- **Stabilisation de l'inversion et absence de multicolinéarité** : Toutes les valeurs propres de $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ sont postives. $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ est alors de plein rang rendant l'inversion plus stable.

La Régularisation Ridge Réduit la Variance des Coefficients

La Régularisation Ridge Réduit la Variance des Coefficients

- Nous cherchons à comparer la variance de l'estimateur Ridge avec celle de l'estimateur des Moindres Carrés Ordinaires (OLS).
- En rappelant que $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ avec $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$, nous avons :

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{Y}.$$

- En substituant $\mathbf{Y}$:

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}).$$

- En développant :

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} + (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\boldsymbol{\epsilon}.$$

La Régularisation Ridge Réduit la Variance des Coefficients

- La variance de $\hat{\boldsymbol{\beta}}_{\text{Ridge}}$ est donnée par :

$$\text{Var}(\hat{\boldsymbol{\beta}}_{\text{Ridge}}) = \text{Var}\left[(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\boldsymbol{\epsilon}\right].$$

- En utilisant $\text{Var}(\boldsymbol{\epsilon}) = \sigma^2\mathbf{I}$ et $\text{Var}(\mathbf{AZ}) = \mathbf{A}\text{Var}(\mathbf{Z})\mathbf{A}^\top$ où $\mathbf{Z}$ est un vecteur aléatoire, nous obtenons :

$$\text{Var}(\hat{\boldsymbol{\beta}}_{\text{Ridge}}) = \sigma^2(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}.$$

- Pour l'estimateur OLS, on sait que :

$$\text{Var}(\hat{\boldsymbol{\beta}}_{\text{OLS}}) = \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}.$$

La Régularisation Ridge Réduit la Variance des Coefficients

- En utilisant la décomposition spectrale de $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ on a :

$$(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1} = \mathbf{U}(\mathbf{\Sigma} + \lambda\mathbf{I})^{-1}\mathbf{U}^\top.$$

- Ainsi, la variance devient :

$$\text{Var}(\hat{\boldsymbol{\beta}}_{\text{Ridge}}) = \sigma^2\mathbf{U}(\mathbf{\Sigma} + \lambda\mathbf{I})^{-1}\mathbf{\Sigma}(\mathbf{\Sigma} + \lambda\mathbf{I})^{-1}\mathbf{U}^\top.$$

- Dans la base propre de $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$, nous avons :

$$\text{Var}(\hat{\beta}_{i,\text{Ridge}}) = \sigma^2\frac{\sigma_i}{(\sigma_i + \lambda)^2}.$$

- Pour OLS, la variance est : $\text{Var}(\hat{\beta}_{i,\text{OLS}}) = \sigma^2\frac{1}{\sigma_i}$.
- Nous avons toujours : $\frac{\sigma_i}{(\sigma_i+\lambda)^2} < \frac{1}{\sigma_i}$ car $\frac{\sigma_i^2}{(\sigma_i+\lambda)^2} < 1 \quad$ pour $\lambda > 0$.
- nous avons bien alors : $\text{Var}(\hat{\beta}_{i,\text{Ridge}}) < \text{Var}(\hat{\beta}_{i,\text{OLS}})$.

Conclusion : Ridge Réduit la Var et Plus Stable

- L'ajout de $\lambda \mathbf{I}$ réduit la variance de chaque coefficient β_i.
- L'effet est plus prononcé lorsque σ_i est petit, ce qui signifie que Ridge régularise davantage les directions associées à des faibles valeurs propres.
- Cette réduction de variance est ce qui permet à Ridge Regression d'améliorer la stabilité numérique et d'éviter le surajustement, contrairement à OLS.
- Cependant, cette réduction de variance s'accompagne d'une augmentation du biais, illustrant le compromis biais-variance.

Méthodes de Régularisation : La Régularisation L_1 Lasso

La Régularisation L_1 Lasso

- La régression Lasso (*Least Absolute Shrinkage and Selection Operator*) est une extension de la régression linéaire classique où l'on ajoute un terme de régularisation L_1 pour favoriser la parcimonie des coefficients.
- On cherche à estimer le vecteur de paramètres $\boldsymbol{\beta}$ en minimisant la fonction de coût suivante :

$$C(\boldsymbol{\beta}) = \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^{p+1} |\beta_j|.$$

où :

- $\mathbf{X} \in \mathbb{R}^{n \times (p+1)}$ est la matrice des variables explicatives (avec une colonne de 1 pour le biais),
- $\mathbf{Y} \in \mathbb{R}^n$ est le vecteur des observations,
- $\boldsymbol{\beta} \in \mathbb{R}^{(p+1)}$ est le vecteur des paramètres à estimer,
- $\lambda \geq 0$ est un hyperparamètre qui contrôle le degré de régularisation.

La Régularisation L_1 Lasso

- La fonction de coût contient un terme de régularisation L_1 qui introduit une pénalisation absolue sur les coefficients $\boldsymbol{\beta}$.
- Nous devons calculer la dérivée de :

$$\frac{\partial}{\partial \beta_j} \left(\lambda \sum_{j=1}^{p+1} |\beta_j| \right).$$

- Cette dérivée est non différentiable en $\beta_j = 0$. Elle est donnée par :

$$\frac{\partial}{\partial \beta_j} |\beta_j| = \begin{cases} +1, & \text{si } \beta_j > 0, \\ -1, & \text{si } \beta_j < 0, \\ \text{indéfini}, & \text{si } \beta_j = 0. \end{cases}$$

- Cela signifie que le gradient du terme L_1 est : $\lambda \cdot \text{sign}(\beta_j)$, où :

$$\text{sign}(\beta_j) = \begin{cases} +1, & \text{si } \beta_j > 0, \\ -1, & \text{si } \beta_j < 0, \\ \in [-1, 1], & \text{si } \beta_j = 0. \end{cases}$$

La Régularisation L_1 Lasso

- En regroupant les termes, le gradient complet de la fonction de coût $C(\boldsymbol{\beta})$ est :

$$\frac{\partial C}{\partial \boldsymbol{\beta}} = 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^\top \mathbf{Y} + \lambda \cdot \text{sign}(\boldsymbol{\beta}).$$

- En annulant le gradient pour obtenir une condition d'optimalité :

$$2\mathbf{X}^\top \mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^\top \mathbf{Y} + \lambda \cdot \text{sign}(\boldsymbol{\beta}) = 0.$$

- Ou encore :

$$\mathbf{X}^\top \mathbf{X}\boldsymbol{\beta} - \mathbf{X}^\top \mathbf{Y} + \frac{\lambda}{2} \cdot \text{sign}(\boldsymbol{\beta}) = 0.$$

- Contrairement à Ridge Regression, où l'on peut directement inverser la matrice pour obtenir une solution analytique, Lasso n'a pas de solution analytique explicite.
- La fonction de coût est non différentiable en $\beta_j = 0$, ce qui rend impossible l'utilisation de la méthode des dérivées classiques.

Descente de Gradient pour La Régularisation L_1 Lasso

Descente de Gradient pour Lasso

- On peut recourir à la descente de gradient pour la régularisation Lasso.
- La mise à jour des paramètres en descente de gradient est donnée par :

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \alpha \frac{\partial C}{\partial \beta_j}$$

- On applique la mise à jour spécifique pour Lasso :

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \alpha \left(2\mathbf{X}_j^\top (\mathbf{X}\boldsymbol{\beta} - \mathbf{Y}) + \lambda \cdot \text{sign}(\beta_j)\right).$$

- Ce qui montre que la mise à jour des coefficients inclut une pénalisation proportionnelle à leur signe, ce qui encourage l'éparsité dans la solution.

Descente de Gradient pour Lasso : Pseudo-Code

- **Initialisation :**
  - On fixe tous les coefficients $\boldsymbol{\beta}$ à zéro au début.
- **Boucle de mise à jour :**
  - On calcule le gradient de la fonction de coût, qui contient :
    - Un terme quadratique : $2\mathbf{X}^\top(\mathbf{X}\boldsymbol{\beta} - \mathbf{Y})$
    - Un terme de régularisation L_1 : $\lambda \cdot \text{sign}(\boldsymbol{\beta})$ qui applique une pénalité sur les coefficients.
  - Chaque coefficient β_j est mis à jour par une descente de gradient :

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \alpha \left(2\mathbf{X}_j^\top (\mathbf{X}\boldsymbol{\beta}^{(t)} - \mathbf{Y}) + \lambda \cdot \text{sign}(\beta_j^{(t)}) \right).$$

- **Retour du vecteur optimal $\boldsymbol{\beta}$ après convergence.**

Coordinate Descent pour La Régularisation L_1 Lasso

Coordinate Descent pour Lasso

- L'algorithme *Coordinate Descent* est une approche efficace pour résoudre le problème Lasso.
- Contrairement à la descente de gradient qui met à jour tous les paramètres simultanément, *Coordinate Descent* met à jour un paramètre à la fois, en maintenant les autres fixes.
- Chaque coefficient β_j est mis à jour en minimisant directement la fonction de coût selon cette seule variable.
- L'optimisation pour chaque β_j s'écrit :

$$\beta_j^{(t+1)} = S\left(\frac{\mathbf{X}_j^\top(\mathbf{Y} - \mathbf{X}_{\setminus j}\boldsymbol{\beta}_{\setminus j})}{\|\mathbf{X}_j\|^2}, \frac{\lambda}{2\|\mathbf{X}_j\|^2}\right).$$

où $S(z, \tau)$ est l'opérateur de *soft-thresholding* défini par :

$$S(z, \tau) = \text{sign}(z) \cdot \max(|z| - \tau, 0).$$

- Ce mécanisme impose naturellement l'éparsité en mettant à zéro les coefficients dont la mise à jour est inférieure au seuil τ.

Coordinate Descent pour La Régularisation L_1 Lasso

- Dans l'algorithme **Coordinate Descent**, à chaque itération, nous mettons à jour un seul coefficient β_j, en supposant que tous les autres restent constants.
- $\mathbf{X}_{\setminus j}$: représente la matrice $\mathbf{X}$ **sans la colonne** correspondant à $\mathbf{X}_j$. Cela signifie qu'on prend en compte toutes les variables explicatives sauf celle associée à β_j.
- $\boldsymbol{\beta}_{\setminus j}$: représente le vecteur $\boldsymbol{\beta}$ **sans le coefficient** β_j. Autrement dit, c'est l'ensemble des coefficients sauf celui qu'on met à jour.
- Lorsqu'on veut mettre à jour β_j, on calcule :

$$z_j = \frac{\mathbf{X}_j^\top (\mathbf{Y} - \mathbf{X}_{\setminus j}\boldsymbol{\beta}_{\setminus j})}{\|\mathbf{X}_j\|^2}.$$

- Cela signifie qu'on regarde l'erreur résiduelle entre les vraies valeurs $\mathbf{Y}$ et la prédiction actuelle sans utiliser β_j.
- Ensuite, on met à jour β_j en le réajustant pour mieux expliquer cette erreur.
- **L'idée principale** : Évaluer combien β_j doit être modifié indépendamment des autres coefficients (dimensions).

Coordinate Descent pour La Régularisation L_1 Lasso

- **Lasso** produit des solutions **éparses** (*sparse solutions*), c'est-à-dire où beaucoup de coefficients β_j sont exactement égaux à zéro.
- Pourquoi ?
  - La régularisation L_1 de Lasso agit comme une contrainte de parcimonie sur les coefficients.
  - Lorsqu'on applique le **soft-thresholding** dans Coordinate Descent :

$$\tau = \frac{\lambda}{2\|\mathbf{X}_j\|^2}$$

  - Si $|z_j| < \tau$, alors $\beta_j = 0$.
- Cela signifie que Lasso :
  - **Sélectionne seulement les variables importantes** en mettant les autres coefficients à zéro.
  - **Simplifie le modèle** en conservant uniquement les variables significatives.

Coordinate Descent pour La Régularisation L_1 Lasso

- **Efficacité accrue pour Lasso** :
  - **Coordinate Descent** met à jour un seul coefficient β_j à la fois, ce qui est plus efficace pour la régularisation L_1.
  - Il exploite la structure creuse du problème, contrairement à la descente de gradient qui met à jour tous les coefficients simultanément.
- **Meilleure convergence** :
  - Contrairement à **Gradient Descent**, qui peut osciller autour de la solution optimale, **Coordinate Descent** converge souvent plus vite pour Lasso.
  - Il s'adapte mieux à la non-différentiabilité de la pénalité L_1.
- **Sélection automatique des variables** :
  - Coordinate Descent force naturellement certaines valeurs de β_j à zéro.
  - Cette propriété clé de Lasso permet d'identifier les variables les plus importantes.
  - Cela réduit la complexité du modèle et améliore son interprétabilité.

Coordinate Descent pour La Régularisation L_1 Lasso

- **Sélection automatique des variables** :
  - Si un dataset contient beaucoup de features inutiles, Lasso les élimine naturellement en mettant leurs coefficients à zéro.
- **Modèles plus interprétables** :
  - Moins de variables utilisées $\Rightarrow$ meilleure compréhension des relations entre les variables.
- **Réduction de la complexité computationnelle** :
  - Moins de coefficients actifs $\Rightarrow$ prédictions plus rapides et moins de mémoire utilisée.
- **Meilleure généralisation** :
  - En supprimant les variables inutiles, on évite le **surajustement** (overfitting), rendant le modèle plus robuste.

Coordinate Descent pour Lasso : Résumé

- Coordinate Descent fonctionne en optimisant un seul paramètre β_j à la fois tout en gardant les autres constants.
- Chaque mise à jour repose sur une régression partielle, en ajustant β_j pour expliquer au mieux les résidus $\mathbf{Y} - \mathbf{X}_{\setminus j}\boldsymbol{\beta}_{\setminus j}$.
- L'opérateur de soft-thresholding $S(z, \tau)$ joue un rôle clé :
  - Si $|z| > \tau$, le coefficient β_j est ajusté proportionnellement à z.
  - Si $|z| \leq \tau$, alors $\beta_j = 0$, ce qui encourage l'éparsité.
- Cette approche est bien plus rapide que la descente de gradient car elle profite de la structure creuse des solutions Lasso, ne mettant à jour que les coefficients nécessaires.

Pseudo-code de l'algorithme Coordinate Descent

- **Initialisation :**
  - Initialiser $\boldsymbol{\beta}$ à zéro ou à une estimation de départ.
- **Boucle principale :**
  - Répéter jusqu'à convergence :
    - Pour chaque coefficient β_j, calculer :

$$z_j = \frac{\mathbf{X}_j^\top (\mathbf{Y} - \mathbf{X}_{\setminus j} \boldsymbol{\beta}_{\setminus j})}{\|\mathbf{X}_j\|^2}.$$

    - Appliquer l'opérateur de soft-thresholding :

$$\beta_j = S\left(z_j, \frac{\lambda}{2\|\mathbf{X}_j\|^2}\right).$$

Où $S(z, \lambda)$ est la fonction de **seuil doux** définie comme :

$$S(z, \lambda) = \begin{cases} z - \lambda, & \text{si } z > \lambda \\ z + \lambda, & \text{si } z < -\lambda \\ 0, & \text{sinon} \end{cases}$$

- **Retour de β après convergence.**

Coordinate Descent pour Lasso : Exercice

Complétez les parties manquantes du code Python.

Coordinate Descent pour Lasso : Exercice

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
import seaborn as sns
import warnings

sns.set_theme()
warnings.simplefilter("ignore")

# Génération des données
np.random.seed(8302)
X = np.linspace(-5, 5, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.cos(X).ravel() ** 2 + np.sin(X).ravel() ** 3 +
↪  np.random.normal(0, 0.6, X.shape[0])

# Séparation en jeu d'entraînement et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
↪  random_state=42)

# Transformation polynomiale des features
degree = 20  # Degré du polynôme
poly = PolynomialFeatures(degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
```

Code : Fonction de Seuil Doux (Soft-Thresholding)

```python
# Fonction soft-thresholding
def soft_thresholding(z, lambda_):
    """ Implémente la fonction de seuil doux """
    if z > lambda_:
        return _____  # Complétez ici
    elif z < -lambda_:
        return _____  # Complétez ici
    else:
        return 0
```

Coordinate Descent pour Lasso : Exercice

```python
# Implémentation de Coordinate Descent pour Lasso
def coordinate_descent_lasso(X, y, lambda_, max_iter=50, tol=1e-6):
    """ Implémente l'algorithme Coordinate Descent pour Lasso """
    m, n = X.shape
    beta = np.zeros(n)  # Initialisation des coefficients
    train_errors, test_errors = [], []

    for it in range(max_iter):
        beta_old = beta.copy()
        for j in range(n):
            # 1. Sélectionner la colonne correspondante X_j
            X_j = X[:, j]
            # 2. Calculer le résidu en excluant la contribution actuelle de X_j
            residual = _____  # Complétez ici
            # 3. Calculer z_j : projection du résidu sur X_j
            z_j = _____  # Complétez ici

            # 4. Mettre à jour beta_j en appliquant le seuil doux et en normalisant
            beta[j] = _____  # Complétez ici
        # Critère d'arrêt basé sur la convergence
        if np.linalg.norm(beta - beta_old, ord=1) < tol:
            break
    return beta
```

Coordinate Descent pour Lasso : Exercice

```python
# Paramètres
lambda_ = 0.1  # Coefficient de régularisation
max_iter = 50  # Nombre maximal d'itérations

# Exécution de Coordinate Descent
beta_lasso = coordinate_descent_lasso(X_train_poly, y_train, lambda_, max_iter)

# Prédictions finales
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_poly_range = poly.transform(X_range)
y_pred = X_poly_range @ beta_lasso

# Tracé des résultats
plt.figure(figsize=(6,5))
plt.scatter(X_train, y_train, label="Train Data", alpha=0.6)
plt.scatter(X_test, y_test, label="Test Data", alpha=0.6)
plt.plot(X_range, y_pred, color='red', label="Lasso (Coordinate Descent)")
plt.legend()
plt.xlabel("X")
plt.ylabel("y")
plt.title("Régression Lasso polynomiale avec Coordinate Descent")
plt.show()

print("Coefficients Lasso (Coordinate Descent) :")
print(beta_lasso)
```

Coordinate Descent pour Lasso : Exercice

```
# Tracé des courbes d'erreurs
plt.figure(figsize=(7,5))
plt.plot(range(1, len(train_errors) + 1), train_errors, label="Erreur
↪  d'entraînement", color="blue")
plt.plot(range(1, len(test_errors) + 1), test_errors, label="Erreur de test",
↪  color="red")
plt.xlabel("Itérations")
plt.ylabel("Erreur quadratique moyenne (MSE)")
plt.title("Évolution des erreurs pendant l'entraînement (Coordinate Descent
↪  Lasso)")
plt.legend()
plt.show()
```

Coordinate Descent pour Lasso : Solution

Coordinate Descent pour Lasso : Solution

Coordinate Descent pour Lasso : Solution

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
import seaborn as sns
import warnings

sns.set_theme()
warnings.simplefilter("ignore")

# Génération des données
np.random.seed(8302)
X = np.linspace(-5, 5, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.cos(X).ravel() ** 2 + np.sin(X).ravel() ** 3 +
↪  np.random.normal(0, 0.6, X.shape[0])

# Séparation en jeu d'entraînement et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
↪  random_state=42)

# Transformation polynomiale des features
degree = 20  # Degré du polynôme
poly = PolynomialFeatures(degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
```

Coordinate Descent pour Lasso : Solution

```python
# Fonction soft-thresholding (opérateur de seuil doux)
def soft_thresholding(z, lambda_):
    if z > lambda_:
        return z - lambda_
    elif z < -lambda_:
        return z + lambda_
    else:
        return 0
```

Coordinate Descent pour Lasso : Solution

```python
# Implémentation de Coordinate Descent
def coordinate_descent_lasso(X, y, lambda_, max_iter=50, tol=1e-6):
    """ Implémente l'algorithme Coordinate Descent pour Lasso """
    m, n = X.shape
    beta = np.zeros(n)  # Initialisation des coefficients
    train_errors, test_errors = [], []

    for it in range(max_iter):
        beta_old = beta.copy()

        for j in range(n):
            # Calcul de la contribution des autres coefficients
            X_j = X[:, j]
            residual = y - X @ beta + X_j * beta[j]  # Exclure X_j * beta_j
            z_j = X_j.T @ residual  # Projection du résidu sur X_j
            # Mise à jour de beta_j
            beta[j] = soft_thresholding(z_j, lambda_) / (X_j.T @ X_j + 1e-6)
        # Calcul des erreurs MSE
        train_mse = np.mean((y_train - X_train_poly @ beta) ** 2)
        test_mse = np.mean((y_test - X_test_poly @ beta) ** 2)
        train_errors.append(train_mse)
        test_errors.append(test_mse)
        # Critère d'arrêt basé sur la convergence
        if np.linalg.norm(beta - beta_old, ord=1) < tol:
            break
    return beta, train_errors, test_errors
```

Coordinate Descent pour Lasso : Solution

```
# Paramètres
lambda_ = 0.1  # Coefficient de régularisation
max_iter = 50  # Nombre maximal d'itérations

# Exécution de Coordinate Descent
beta_lasso = coordinate_descent_lasso(X_train_poly, y_train, lambda_, max_iter)

# Prédictions finales
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_poly_range = poly.transform(X_range)
y_pred = X_poly_range @ beta_lasso

# Tracé des résultats
plt.figure(figsize=(6,5))
plt.scatter(X_train, y_train, label="Train Data", alpha=0.6)
plt.scatter(X_test, y_test, label="Test Data", alpha=0.6)
plt.plot(X_range, y_pred, color='red', label="Lasso (Coordinate Descent)")
plt.legend()
plt.xlabel("X")
plt.ylabel("y")
plt.title("Régression Lasso polynomiale avec Coordinate Descent")
plt.show()

print("Coefficients Lasso (Coordinate Descent) :")
print(beta_lasso)
```

Coordinate Descent pour Lasso : Solution

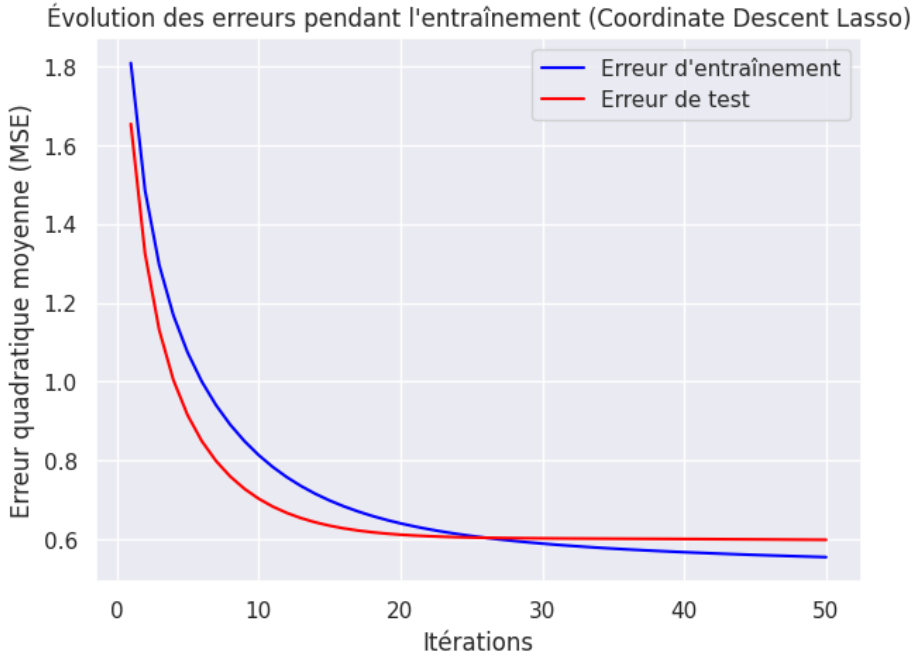
```
# Tracé des courbes d'erreurs
plt.figure(figsize=(7,5))
plt.plot(range(1, len(train_errors) + 1), train_errors, label="Erreur
↪  d'entraînement", color="blue")
plt.plot(range(1, len(test_errors) + 1), test_errors, label="Erreur de test",
↪  color="red")
plt.xlabel("Itérations")
plt.ylabel("Erreur quadratique moyenne (MSE)")
plt.title("Évolution des erreurs pendant l'entraînement (Coordinate Descent
↪  Lasso)")
plt.legend()
plt.show()
```

Résultat Affiché :

```
Coefficients Lasso (Coordinate Descent) :
[ 2.92406796e-01  1.00011315e+00  8.83800614e-02 -8.91818521e-02
 -4.07021225e-04 -6.59136547e-05 -1.42928328e-04  1.56383791e-05
 -3.65523251e-06  5.82578028e-07 -5.64711732e-08  1.90297197e-08
  7.93132958e-11  6.00032183e-10  5.76177254e-11  1.78282914e-11
  3.55447932e-12  4.73993040e-13  1.72311097e-13  9.88245588e-15
  7.69709768e-15]
```

Coordinate Descent pour Lasso : Analyse des Résultats

- L'erreur d'entraînement et l'erreur de test diminuent progressivement au fil des itérations.
- L'erreur de test reste légèrement inférieure à l'erreur d'entraînement, ce qui peut indiquer que le modèle généralise bien et ne souffre pas de sur-apprentissage (overfitting).
- La courbe d'erreur montre une convergence, ce qui indique que l'algorithme Coordinate Descent a bien trouvé une solution stable.



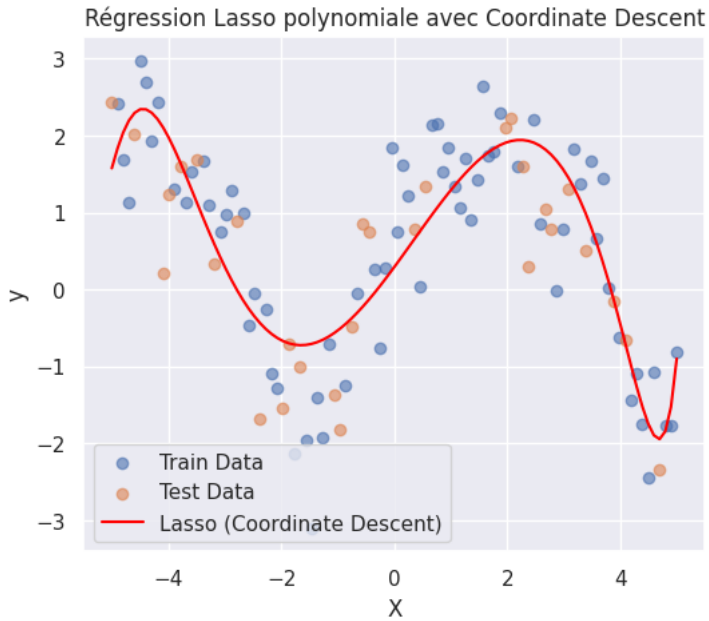
Coordinate Descent pour Lasso : Analyse des Résultats

- Certains coefficients sont proches de zéro, voire exactement égaux à zéro. Cela signifie que Lasso a automatiquement éliminé certaines variables non pertinentes.
- Les premiers coefficients (associés aux puissances les plus basses du polynôme) ont une plus grande magnitude que ceux des puissances plus élevées, indiquant que le modèle privilégie les termes les plus significatifs.
- La décroissance rapide des coefficients vers zéro pour les termes de degré élevé impose une certaine parcimonie, aidant à éviter le sur-ajustement.

$$\hat{\boldsymbol{\beta}} = \begin{bmatrix} 2.92e-01 & 1.00e+00 & 8.83e-02 & -8.91e-02 & -4.07e-04 \\ -6.59e-05 & -1.42e-04 & 1.56e-05 & -3.65e-06 & 5.82e-07 \\ -5.64e-08 & 1.90e-08 & 7.93e-11 & 6.00e-10 & 5.76e-11 \\ 1.78e-11 & 3.55e-12 & 4.73e-13 & 1.72e-13 & 9.88e-15 \\ 7.69e-15 & & & & \end{bmatrix}$$

Coordinate Descent pour Lasso : Analyse des Résultats

- La courbe rouge de prédiction montre un bon ajustement aux données sans sur-ajustement excessif, ce qui est un des objectifs principaux de la régularisation Lasso.
- Lasso aide à contrôler la complexité du modèle en réduisant l'impact des termes de haut degré, rendant la solution plus robuste.
- Comparé à une régression polynomiale classique sans régularisation, le modèle Lasso est plus stable et moins sujet aux oscillations excessives.



Coordinate Descent pour Lasso : Analyse des Résultats

- L'algorithme Coordinate Descent a bien fonctionné, produisant un modèle régularisé qui capture la tendance des données sans être trop complexe.
- Lasso automatise la sélection de variables en mettant certains coefficients à zéro, simplifiant ainsi le modèle et améliorant son interprétabilité.
- Le compromis biais-variance semble être bien géré, permettant une bonne généralisation sur les données de test.

Table des Matières

Méthodes de Sélection de Modèles

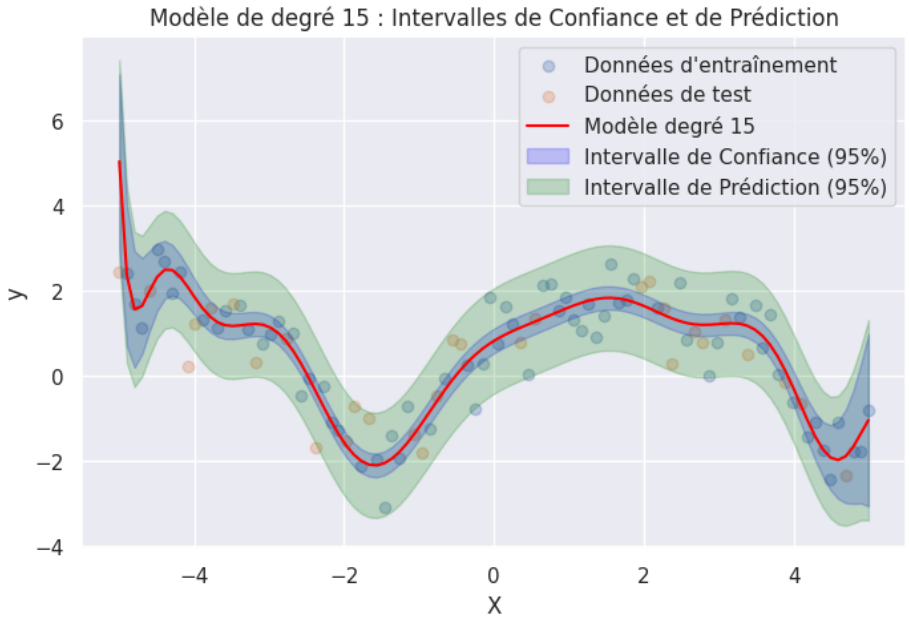
Méthodes de Sélection de Modèles : Critères de Sélection

Méthodes de Sélection de Modèles : Critères de Sélection

- **Pourquoi avons-nous besoin de critères de sélection ?**
- Ajuster un modèle aux données ne suffit pas : ce n'est qu'une première étape.
- Un bon modèle doit :
  - Bien s'ajuster aux données d'entraînement (faible variance $\Rightarrow$ Robustesse).
  - Être capable de généraliser à de nouvelles données (Biais modéré),
  - Être aussi simple et interprétable que possible.
- Juste mesurer l'erreur d'ajustement et appliquer une méthode de régularisation n'est pas suffisant pour juger de la qualité d'un modèle.
- On a besoin de techniques plus rigoureuses pour :
  - Considérer dans la comparaison plusieurs modèles candidats.
  - Décider de façon rigoureuse lequel est le plus adapté aux données.
- Cela nous amène à des critères comme :
  - KL divergence.
  - R^2_{adj}.
  - AIC et BIC.
- **Et à la validation croisée.**

Méthodes de Sélection de Modèles : Critères de Sélection

- **L'erreur sur l'ensemble d'entraînement** n'est pas un bon indicateur de la capacité du modèle à bien généraliser.
  - Un modèle très complexe (Ex : polynôme de degré 15 dans le problème régression polynomiale) s'ajuste parfaitement et sur-apprend. Le modèle s'adapte trop aux variations du jeu d'entraînement et risque de mal généraliser sur de nouvelles données.



Méthodes de Sélection de Modèles : Critères de Sélection

- Lorsqu'on ajuste un modèle statistique ou d'apprentissage automatique, l'objectif n'est pas seulement de prédire correctement sur les données d'entraînement.
- Le vrai but est de trouver un modèle qui **généralise bien** à de nouvelles données jamais vues.
- Un modèle trop complexe peut s'ajuster parfaitement à l'échantillon de données (sur-apprentissage), mais échouer à généraliser.
- Il faut donc un **critère de sélection** qui permette de choisir, parmi plusieurs modèles, celui qui est le plus proche du *vrai modèle générateur*.

Critères de Sélection : KL Divergence

- Avant d'ajuster des modèles, on se pose une question fondamentale :

  À quel point un modèle donné est-il proche de la vraie distribution des données?

- La divergence de Kullback-Leibler (*KL Divergence*) permet de comparer:
  - une distribution de données $f(\mathbf{X})$ (observée ou vraie)
  - à une distribution modélisée $g_{\boldsymbol{\theta}}(\mathbf{X})$ (où $\boldsymbol{\theta}$ les paramètres du modèle).
- Elle mesure **la divergence** entre deux distributions de probabilité

$$D_{\mathsf{KL}}(f \,\|\, g) = \int f(\mathbf{X}) \log \left(\frac{f(\mathbf{X})}{g_{\boldsymbol{\theta}}(\mathbf{X})} \right) d\mathbf{X}$$

- C'est une **quantité statistique** : on compare deux lois de probabilité, pas juste un seul jeu de données.
- Ce n'est pas une distance (non symétrique), mais une mesure de perte d'information.
- Plus $g(X)$ est proche de $f(X)$, plus $D_{\mathsf{KL}}(f \,\|\, g)$ est petit.

$$D_{\mathsf{KL}}(f \| g_{\boldsymbol{\theta}}) = \mathbb{E}_f[\log f(\mathbf{X})] - \mathbb{E}_f[\log g(\mathbf{X} \mid \boldsymbol{\theta})]$$

Critères de Sélection : KL Divergence

- Cela fournit une base rigoureuse pour dire : *Ce modèle est plus proche de la réalité que celui-là.*
- **Exemple** :
  - $f(x)$: distribution empirique des données observées.
  - $g_{\boldsymbol{\theta_1}}(\mathbf{X}), g_{\boldsymbol{\theta_2}}(\mathbf{X}), g_{\boldsymbol{\theta_3}}(\mathbf{X})$: trois modèles candidats.
  - Le meilleur modèle est celui dont la KL-divergence est la plus faible.
- Mis à part l'évaluation, $\Rightarrow$ la divergence de Kullback-Leibler peut être adoptée comme fonction objectif en modélisation statistique.
- Les données $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ sont supposées générées par une distribution vraie f (inconnue).
- On ajuste un modèle paramétrique $g(\mathbf{X} \mid \boldsymbol{\theta})$.
- On souhaite minimiser :

$$D_{\mathsf{KL}}(f \| g_{\boldsymbol{\theta}}) = \mathbb{E}_f[\log f(\mathbf{X})] - \mathbb{E}_f[\log g(\mathbf{X} \mid \boldsymbol{\theta})]$$

- Comme $\mathbb{E}_f[\log f(\mathbf{X})]$ ne dépend pas de $\boldsymbol{\theta}$, on maximise :

$$\mathbb{E}_f[\log g(\mathbf{X} \mid \boldsymbol{\theta})] \approx \frac{1}{n} \sum_{i=1}^{n} \log g(\mathbf{x}_i \mid \boldsymbol{\theta})$$

Critères de Sélection : R^2_{adj}

- D'autres critères théoriques permettent de trouver un compromis entre qualité d'ajustement et complexité :
- Le coefficient R^2 mesure la proportion de variabilité expliquée par le modèle:

$$R^2 = \frac{SC_{reg}}{SC_{totale}} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \overline{y})^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} = 1 - \frac{SC_{res}}{SC_{totale}} = 1 - \frac{\sum_{i=1}^{n}(e_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$$

- Mais R^2 augmente toujours quand on ajoute des variables, même inutiles.
- Solution : le R^2 ajusté pénalise les modèles trop complexes :

$$R^2_{\text{adj}} = 1 - (1 - R^2) \cdot \frac{n-1}{n-p-1}$$

où :

- n : nombre d'observations,
- p : nombre de variables explicatives.

Critères de Sélection : R^2_{adj}

- R^2 ajusté pénalise les modèles trop complexes :

$$R^2_{\text{adj}} = 1 - (1 - R^2) \cdot \frac{n-1}{n-p-1}, \quad \text{où :}$$

  - n : nombre d'observations,
  - p : nombre de variables explicatives.
- **Intuition asymptotique** :
  - Si $n \to \infty$, alors $R^2_{\text{adj}} \to R^2$.
    Cela reflète que quand on a beaucoup de données, R^2 devient un bon indicateur de généralisation.
- **Attention si** $p \gg n$:
  - Le dénominateur $(n - p - 1)$ devient petit ou négatif.
  - Le modèle est alors sur-paramétré : R^2_{adj} peut devenir instable ou non défini.
  - Dans ce cas, il vaut mieux utiliser d'autres critères de sélection (AIC, BIC, régularisation).

Critères de Sélection : AIC & BIC

- Akaike corrige l'optimisme de la log-vraisemblance empirique par un terme de complexité.
- Formule :

$$\text{AIC} = -2 \log \mathcal{L}(\hat{\boldsymbol{\theta}}) + 2k, \quad \text{où :}$$

  - $\log \mathcal{L}$ est la log-vraisemblance maximisée ;
  - k est le nombre de paramètres estimés.
- Le BIC vient d'une approximation bayésienne de la vraisemblance marginale.
- Formule :

$$\text{BIC} = -2 \log \mathcal{L}(\hat{\boldsymbol{\theta}}) + k \log n$$

- Comparé à l'AIC :
  - pénalise plus fortement les modèles complexes quand n est grand ;
  - sélectionne des modèles plus simples.

Critères de Sélection : AIC & BIC

- Soit un modèle paramétrique $g(\mathbf{X} \mid \boldsymbol{\theta})$, et des observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$.
- La **vraisemblance** du modèle est :

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{n} g(\mathbf{x}_i \mid \boldsymbol{\theta})$$

- La **log-vraisemblance** est alors :

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log g(\mathbf{x}_i \mid \boldsymbol{\theta})$$

- On maximise cette fonction pour obtenir l'estimateur du maximum de vraisemblance :

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \log \mathcal{L}(\boldsymbol{\theta})$$

- C'est cette valeur maximisée $\log \mathcal{L}(\hat{\boldsymbol{\theta}})$ qu'on utilise dans les formules de AIC et BIC.

Critères de Sélection : AIC & BIC

- Dans le cadre de la régression linéaire où le bruit est Gaussien $\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_n)$, la log-vraisemblance s'écrit (à une constante près) en fonction de l'erreur quadratique.
- On obtient alors les expressions suivantes pour l'AIC et le BIC :

$$\text{AIC} = n \log \left(\frac{1}{n} \|\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}\|_2^2 \right) + 2k + \text{const}$$

$$\text{BIC} = n \log \left(\frac{1}{n} \|\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}\|_2^2 \right) + k \log n + \text{const}$$

- Où :
  - $\hat{\boldsymbol{\beta}}$ est l'estimateur des moindres carrés ;
  - k est le nombre de paramètres estimés (typiquement $k = p + 1$ si l'on inclut l'ordonnée à l'origine).

Critères de Sélection : AIC & BIC

- AIC $= -2 \log \mathcal{L}(\hat{\boldsymbol{\theta}}) + 2k$.
  - Le premier terme mesure la qualité d'ajustement (meilleur si plus petit).
  - Le second pénalise les modèles avec beaucoup de paramètres k.
- BIC $= -2 \log \mathcal{L}(\hat{\boldsymbol{\theta}}) + k \log n$.
  - Même structure que AIC, mais la pénalisation dépend aussi de la taille n de l'échantillon.
  - Plus sévère quand n est grand : favorise des modèles plus simples.
- **Intuition asymptotique** :
  - Quand $n \to \infty$, la log-vraisemblance s'approxime bien par la log-vraisemblance du vrai modèle.
  - Le BIC devient **consistant** : il sélectionne le vrai modèle avec probabilité 1 si $n \to \infty$.
  - L'AIC reste orienté vers la **prédiction** (même si le modèle n'est pas le vrai).

Critères de Sélection : QCM

Critères de sélection : QCM

Q1. Pourquoi ne peut-on pas se fier uniquement à l'erreur sur les données d'entraînement pour choisir un modèle ?

A) Parce qu'elle est trop lente à calculer.

B) Parce qu'elle ignore la variance du bruit.

C) Parce qu'un modèle trop complexe peut sur-apprendre les données d'entraînement. (Bonne réponse)

D) Parce qu'elle surestime la variance expliquée.

Justification :

- L'erreur d'ajustement est faible pour un modèle qui sur-apprend les données.
- Ce modèle ne généralisera pas bien à de nouvelles données.
- Il faut donc tenir compte de la complexité du modèle, pas seulement de sa performance sur l'entraînement.

QCM – Critères de sélection

Q2. Quel critère mesure la perte d'information entre un modèle et la réalité ?

A) R^2

B) AIC

C) BIC

D) KL divergence (Bonne réponse)

Justification :

- La KL-divergence mesure l'écart entre la vraie distribution $f(\mathbf{X})$ et la distribution du modèle $g(\mathbf{X} \mid \theta)$.
- Elle quantifie la perte d'information lorsqu'on utilise le modèle g à la place de la vérité f.

QCM – R^2_{adj} et surparamétrisation

Q3. Si $p \gg n$, que peut-il se passer pour R^2_{adj} ?

A) Il devient plus fiable

B) Il est identique à R^2

C) Il peut devenir instable ou non défini (Bonne réponse)

D) Il converge vers 0

Justification :

- Le dénominateur de R^2_{adj} contient $(n - p - 1)$, qui devient petit ou négatif quand $p \gg n$.
- Cela rend l'indicateur instable, voire invalide.

QCM – AIC vs BIC

Q4. Dans un problème avec beaucoup d'observations (n grand), lequel des deux critères favorise typiquement les modèles plus simples ?

A) AIC

B) BIC (Bonne réponse)

C) KL divergence

D) R^2

Justification :

- Le terme de pénalisation dans le BIC est $k \log n$, qui devient plus grand avec n.
- Cela incite à choisir des modèles plus simples, càd, avec moins de paramètres quand on a beaucoup de données.

Sélection de Modèles : Vers une Approche plus Empirique

- Ces critères reposent souvent sur des hypothèses : normalité, indépendance, spécification correcte du modèle...
- **Limite des critères théoriques** : ils reposent sur des hypothèses fortes qui ne sont pas toujours vérifiées.
- **Idée** : Évaluer directement la capacité de généralisation sur des données non vues.
- **Principe de la validation** :
  - Séparer les données en deux parties : entraînement et validation.
  - Ajuster le modèle sur l'ensemble d'entraînement.
  - Évaluer la performance sur l'ensemble de validation.
- Cela mène naturellement à la validation croisée.

Méthodes de Sélection de Modèles : Validation Croisée

Méthodes de Sélection de Modèles : Validation Croisée

- Les critères comme AIC, BIC ou R^2_{adj} reposent sur des hypothèses fortes :
  - Bruit Gaussien
  - Indépendance des observations
  - Bonne spécification du modèle
- Mais en pratique :
  - **Ces hypothèses sont souvent violées.**
  - On veut simplement savoir si notre modèle généralise bien à de nouvelles données.
- **Solution** : évaluer la performance du modèle sur des données non utilisées pour l'ajustement.
- C'est le principe fondamental de la **validation empirique** (croisée).

Méthodes de Sélection de Modèles : Validation Croisée

- En apprentissage statistique (automatique), **les hyperparamètres** sont des paramètres **fixés avant l'entraînement du modèle,** contrairement aux paramètres **appris à partir des données**.
- Ils déterminent le comportement du modèle ou de l'algorithme d'optimisation.
- **Exemples d'hyperparamètres :**
  - En régression polynomiale : le **degré du polynôme** choisi est un hyperparamètre.
  - En régression Ridge ou Lasso : la **constante de régularisation** (souvent notée λ) est un hyperparamètre.
  - En descente de gradient ou coordinate descent : le **pas d'apprentissage** (learning rate) est un hyperparamètre.
- **Problème** : on ne sait pas *a priori* quelle est la meilleure combinaison.
- **Solution simple : Recherche sur une grille** : tester plusieurs combinaisons sur un ensemble de validation.

Méthodes de Sélection de Modèles : Validation Croisée

- Pour évaluer et sélectionner un modèle, on divise généralement les données en différents ensembles.
- **Pour les grands jeux de données :**
  - **Ensemble d'entraînement** $\mathcal{D}_{\text{train}}$: utilisé pour entraîner le modèle.
  - **Ensemble de validation** $\mathcal{D}_{\text{valid}}$: utilisé pour choisir les meilleurs hyper-paramètres (sélection de modèle).
  - **Ensemble de test** $\mathcal{D}_{\text{test}}$: utilisé **une seule fois à la fin**, après toute validation, pour estimer la performance réelle sur des données jamais vues.
- **Méthodes de Sélection de Modèles : Validation Croisée**
  - Il n'est pas toujours possible de réserver une grande partie des données pour la validation ou le test.
  - On utilise alors des techniques comme la **validation croisée à K partitions** (*K-fold cross-validation*) pour réutiliser intelligemment toutes les données.

Pseudocode : Validation Croisée

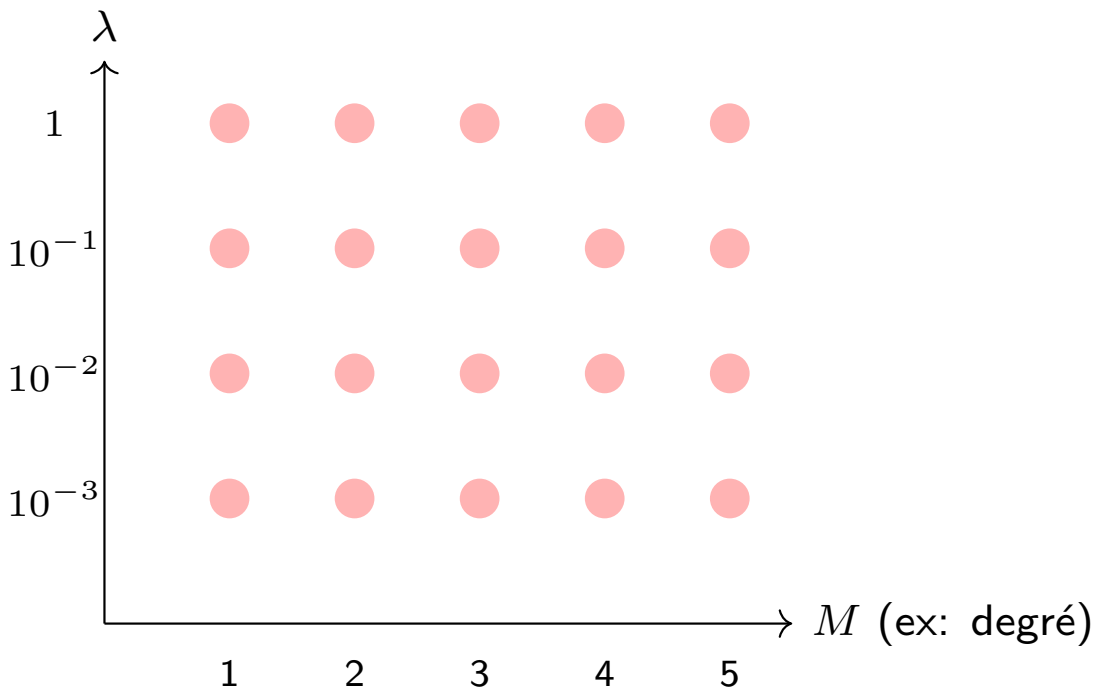
Étapes

- Diviser les données $\mathcal{D}$ en trois parties :
  - Entraînement $\mathcal{D}_{\text{train}}$
  - Validation $\mathcal{D}_{\text{valid}}$
  - Test $\mathcal{D}_{\text{test}}$
- Pour chaque valeur d'hyperparamètre h :
  - Entraîner un modèle sur $\mathcal{D}_{\text{train}}$
  - Évaluer la performance sur $\mathcal{D}_{\text{valid}}$
- Choisir la valeur $h^\star$ qui donne la meilleure performance sur $\mathcal{D}_{\text{valid}}$
- Réentraîner un modèle final sur $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{valid}}$
- Évaluer ce modèle final sur $\mathcal{D}_{\text{test}}$

Méthodes de Sélection de Modèles : Validation Croisée

- La **recherche sur grille** (*grid search*) est une méthode systématique pour trouver les meilleurs hyperparamètres h^*.
- Elle consiste à :
  - Définir une grille de valeurs possibles pour chaque hyperparamètre ;
  - Évaluer les performances pour **chaque combinaison** possible.
- **Exemple** :
  - $M \in \{1, 2\}$, $\lambda \in \{0, 10^{-6}, 10^{-3}\}$
  - Total de $2 \times 3 = 6$ modèles à entraîner et valider
- On garde la combinaison donnant la meilleure performance moyenne (e.g. erreur quadratique moyenne la plus basse).

Méthodes de Sélection de Modèles : Validation Croisée



Chaque point de la grille correspond à une combinaison d'hyperparamètres testée.

Pseudocode : recherche sur grille (grid search)

Objectif

Trouver les meilleurs hyperparamètres à partir d'un ensemble de combinaisons possibles.

Étapes

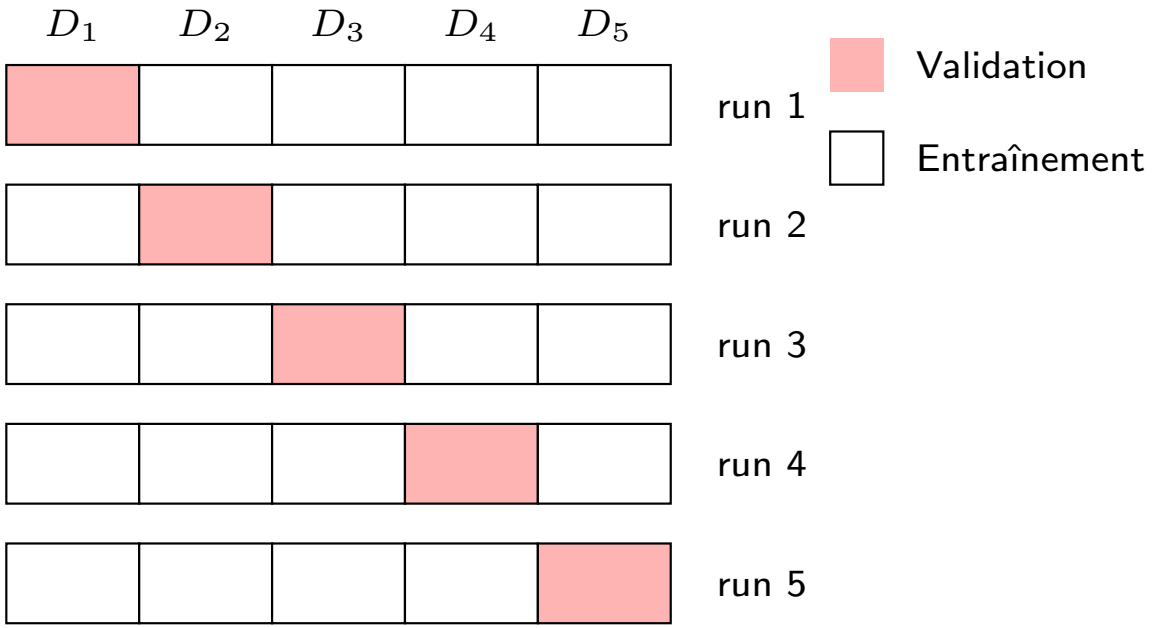
- Définir une liste de valeurs possibles pour chaque hyperparamètre :
  - Exemple : $M \in \{1, 2\}$, $\lambda \in \{0, 10^{-6}, 10^{-3}\}$
- Construire la grille de toutes les combinaisons possibles d'hyperparamètres.
- Pour chaque combinaison (M, λ) dans la grille :
  - Évaluer la performance du modèle à l'aide de la validation croisée
- Choisir la combinaison donnant la meilleure performance moyenne.

Méthodes de Sélection de Modèles : Validation Croisée

- **Ratios courants (indicatifs) :**
  - **Grand jeu de données** : 60% train / 20% valid / 20% test
  - **Moyen jeu de données** : 70% train / 30% test (validation croisée utilisée sur le train)
  - **Petit jeu de données** : 80% train / 20% valid ou validation croisée complète
- Pourquoi ne pas utiliser $\mathcal{D}_{\text{test}}$ pour choisir les hyper-paramètres ?
  - Car cela reviendrait à adapter le modèle à ces données ;
  - Ce ne serait plus un indicateur objectif de généralisation.
  - Utiliser le test trop tôt revient à "tricher" : on ajuste inconsciemment aux données test.
  - Cela donne une fausse impression de généralisation.
- **Limite** : ce découpage reste sensible au hasard de la séparation. D'où l'intérêt de la *K-fold cross-validation*.

Méthodes de Sélection de Modèles : Validation Croisée

- **K-fold Cross-validation** :
  - Divise les données en k parties.
  - Chaque partie est utilisée une fois comme ensemble de validation, les $k-1$ autres pour l'entraînement.
  - On moyenne les performances pour choisir les meilleurs hyper-paramètres.



Pseudocode : Validation croisée à K plis

Étapes

- Diviser les données $\mathcal{D}$ en K sous-ensembles de taille similaire : $\mathcal{D}_1, \ldots, \mathcal{D}_K$
- Pour chaque valeur d'hyperparamètre h :
  - Initialiser une variable pour accumuler les performances
  - Pour chaque pli $k = 1, \ldots, K$:
    - Utiliser $\mathcal{D}_k$ comme validation
    - Utiliser les $K - 1$ autres sous-ensembles comme entraînement
    - Entraîner un modèle et mesurer l'erreur sur $\mathcal{D}_k$
  - Calculer la performance moyenne sur les K plis
- Choisir la valeur $h^\star$ ayant la meilleure performance moyenne
- Réentraîner un modèle final sur l'ensemble complet $\mathcal{D}$ avec $h^\star$

Exercice : Validation Croisée Pour Ridge la Régularisation Ridge

Exercice : Régression Ridge et Validation Croisée

- **Implémenter la régression Ridge manuellement**
  Utiliser la formule fermée avec régularisation :

$$\hat{\boldsymbol{w}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

- **Découper manuellement les données**
  Créez une fonction qui sépare les données en trois ensembles :
  - Entraînement (train) : 60 %
  - Validation : 20 %
  - Test : 20 %

  Utiliser `random_state=8302` pour assurer la reproductibilité.
- **Implémenter la validation croisée simple pour choisir le meilleur hyperparamètre λ**
  - Tester plusieurs valeurs de λ sur les données de validation.
  - Réentraîner sur `train + valid` avec le meilleur λ.
  - Évaluer la performance finale sur l'ensemble de test.

Exercice : Régression Ridge et Validation Croisée

- **Implémenter la validation croisée à K plis (avec $K = 5$)**
  - Moyennez les erreurs de validation sur les 5 plis pour chaque valeur de λ.
  - Sélectionnez le λ optimal à partir de la performance moyenne.
- **Implémenter la recherche sur grille (grid search)**
  - Pour une grille de valeurs de λ, évaluer toutes les combinaisons possibles.
  - Pour chaque combinaison, utiliser une méthode de validation (simple ou K-fold).
  - Identifier la meilleure combinaison selon l'erreur quadratique moyenne (MSE).
- **Valeurs suggérées pour les tests** :
  - `degrees = [3, 5, 7, 10]`
  - `lambdas = [1e-4, 1e-2, 1e-1, 1, 10]`

Code à compléter : Validation croisée et Grid Search

```python
import numpy as np
import matplotlib.pyplot as plt

# 1. Génération des données bruitées
np.random.seed(8302)
X = np.linspace(-5, 5, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.cos(X).ravel()**2 + np.sin(X).ravel()**3
y += np.random.normal(0, 0.6, size=y.shape)

# 2. Matrice de design pour polynôme de degré d
def design_matrix(X, degree):
    return np.hstack([X**i for i in range(_______ + 1)])

# 3. Régression Ridge
def ridge_fit(X, y, lam):
    n_features = X.shape[1]
    I = np.eye(_______)
    return _______

def ridge_predict(X, w):
    return _______
```

Code à compléter : Validation croisée et Grid Search

```python
# 4. Split manuel 60/20/20
def manual_split(X, y, train_ratio=0.6, valid_ratio=0.2, random_state=8302):
    np.random.seed(random_state)
    indices = np.random.permutation(len(X))
    n_train = int(_______ * len(X))
    n_valid = int(_______ * len(X))
    train_idx = indices[:n_train]
    valid_idx = indices[n_train:n_train + n_valid]
    test_idx = indices[n_train + n_valid:]
    return (X[train_idx], y[_______],
            X[valid_idx], y[_______],
            X[test_idx], y[_______])
def simple_validation(X, y, degree, lambdas): # 5. Validation simple
    X_train, y_train, X_valid, y_valid, X_test, y_test = manual_split(X, y)
    Phi_train = design_matrix(X_train, degree)
    Phi_valid = design_matrix(X_valid, degree)
    best_lambda = None
    best_valid_error = float('inf')
    for lam in lambdas:
        w = ridge_fit(Phi_train, y_train, lam)
        y_valid_pred = ridge_predict(Phi_valid, w)
        valid_error = np.mean((_______ - y_valid_pred)**2)
        print(f"lambda = {lam:.4f} | Validation MSE = {valid_error:.4f}")
        if _______ < best_valid_error:
            best_valid_error = _______
            best_lambda = _______
    return best_lambda, best_valid_error
```

Code à compléter : Validation croisée et Grid Search

```python
# 6. Validation croisée K-fold
def k_fold_cv(X, y, degree, lambdas, k=5):
    n = len(X)
    # Création d'un tableau d'indices pour les données (de 0 à n-1)
    indices = np.arange(n)
    # On initialise un tableau de taille k où chaque pli contiendra n // k éléments
    fold_sizes = np.full(k, n // k)
    # S'il reste des points (n % k  0), on ajoute 1 aux premiers plis pour
    ↪  équilibrer
    fold_sizes[:n % k] += 1
    # Mélange aléatoire des indices pour une répartition aléatoire des plis
    np.random.seed(8302)
    np.random.shuffle(indices)
    folds = np.split(indices, np.cumsum(fold_sizes)[:-1])
    avg_errors = []
    # Découpe des indices selon les tailles cumulées définies dans fold_sizes
    # Cela crée une liste de k tableaux, chacun représentant un pli
    folds = np.split(indices, np.cumsum(fold_sizes)[:-1])
    for lam in lambdas:
        errors = []
        for i in range(k):
            valid_idx = folds[i]
            train_idx = np.hstack(folds[:i] + folds[i+1:])
            X_train, y_train = X[train_idx], y[train_idx]
            X_valid, y_valid = _______
            Phi_train = design_matrix(X_train, degree)
            Phi_valid = design_matrix(X_valid, degree)
```

Code à compléter : Validation croisée et Grid Search

```
            w = ridge_fit(_______)
            y_valid_pred = ridge_predict(_______)
            valid_error = np.mean((_______)**2)
            errors.append(valid_error)
        avg_error = _______
        avg_errors.append(avg_error)
        print(f"lambda = {lam:.4f} | Moyenne MSE (k-fold) = {avg_error:.4f}")
    best_idx = np.argmin(avg_errors)
    return lambdas[best_idx], avg_errors[best_idx]
# 7. Recherche sur grille
def grid_search(X, y, degrees, lambdas, validation_fn):
    best_val_error = float('inf')
    best_degree = None
    best_lambda = None
    for degree in degrees:
        print(f"Degré testé : {degree}")
        lam, val_err = validation_fn(X, y, _______, _______)
        print(f"Meilleur lambda = {lam:.4f} | Erreur validation = {val_err:.4f}")
        if val_err < best_val_error:
            best_val_error = val_err
            best_lambda = lam
            best_degree = degree
    print("Résultat final :")
    print(f"Degré optimal = {best_degree}, lambda optimal = {best_lambda}, MSE =
    ↪  {best_val_error:.4f}")
    return best_degree, best_lambda, best_val_error
```

Code à compléter : Validation croisée et Grid Search

```
# 8. Paramètres
degrees = [3, 5, 7, 10]
lambdas = [1e-4, 1e-2, 1e-1, 1, 10]

# 9. Lancements
print("=== Validation simple ===")
grid_search(X, y, _______, _______, simple_validation)

print("\n=== Validation croisée K-fold ===")
grid_search(X, y, _______, _______, lambda X, y, d, l: k_fold_cv(X, y, d, l,
↪  k=_______))
```

Solution de l'Exercice : Validation Croisée Pour Ridge la Régularisation Ridge

Solutions : Validation croisée et Grid Search (1/5)

```python
import numpy as np
import matplotlib.pyplot as plt

# 1. Génération des données bruitées
np.random.seed(8302)
X = np.linspace(-5, 5, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.cos(X).ravel()**2 + np.sin(X).ravel()**3
y += np.random.normal(0, 0.6, size=y.shape)

# 2. Matrice de design pour polynôme de degré d
def design_matrix(X, degree):
    return np.hstack([X**i for i in range(degree + 1)])

# 3. Régression Ridge
def ridge_fit(X, y, lam):
    n_features = X.shape[1]
    I = np.eye(n_features)
    return np.linalg.inv(X.T @ X + lam * I) @ X.T @ y

def ridge_predict(X, w):
    return X @ w
```

Solutions : Validation croisée et Grid Search (2/5)

```python
# 4. Split manuel 60/20/20
def manual_split(X, y, train_ratio=0.6, valid_ratio=0.2, random_state=8302):
    np.random.seed(random_state)
    indices = np.random.permutation(len(X))
    n_train = int(train_ratio * len(X))
    n_valid = int(valid_ratio * len(X))
    train_idx = indices[:n_train]
    valid_idx = indices[n_train:n_train + n_valid]
    test_idx = indices[n_train + n_valid:]
    return (X[train_idx], y[train_idx],
            X[valid_idx], y[valid_idx],
            X[test_idx], y[test_idx])
def simple_validation(X, y, degree, lambdas): # 5. Validation simple
    X_train, y_train, X_valid, y_valid, X_test, y_test = manual_split(X, y)
    Phi_train = design_matrix(X_train, degree)
    Phi_valid = design_matrix(X_valid, degree)
    best_lambda = None
    best_valid_error = float('inf')
    for lam in lambdas:
        w = ridge_fit(Phi_train, y_train, lam)
        y_valid_pred = ridge_predict(Phi_valid, w)
        valid_error = np.mean((y_valid - y_valid_pred)**2)
        print(f"lambda = {lam:.4f} | Validation MSE = {valid_error:.4f}")
        if valid_error < best_valid_error:
            best_valid_error = valid_error
            best_lambda = lam
    return best_lambda, best_valid_error
```

Solutions : Validation croisée et Grid Search (3/5)

```python
# 6. Validation croisée K-fold
def k_fold_cv(X, y, degree, lambdas, k=5):
    n = len(X)
    indices = np.arange(n)
    fold_sizes = np.full(k, n // k)
    fold_sizes[:n % k] += 1
    np.random.seed(8302)
    np.random.shuffle(indices)
    folds = np.split(indices, np.cumsum(fold_sizes)[:-1])
    avg_errors = []
    for lam in lambdas:
        errors = []
        for i in range(k):
            valid_idx = folds[i]
            train_idx = np.hstack(folds[:i] + folds[i+1:])
            X_train, y_train = X[train_idx], y[train_idx]
            X_valid, y_valid = X[valid_idx], y[valid_idx]
            Phi_train = design_matrix(X_train, degree)
            Phi_valid = design_matrix(X_valid, degree)
```

Solutions : Validation croisée et Grid Search (4/5)

```
            w = ridge_fit(Phi_train, y_train, lam)
            y_valid_pred = ridge_predict(Phi_valid, w)
            valid_error = np.mean((y_valid - y_valid_pred)**2)
            errors.append(valid_error)
        avg_error = np.mean(errors)
        avg_errors.append(avg_error)
        print(f"lambda = {lam:.4f} | Moyenne MSE (k-fold) = {avg_error:.4f}")
    best_idx = np.argmin(avg_errors)
    return lambdas[best_idx], avg_errors[best_idx]

# 7. Recherche sur grille
def grid_search(X, y, degrees, lambdas, validation_fn):
    best_val_error = float('inf')
    best_degree = None
    best_lambda = None
    for degree in degrees:
        print(f"Degré testé : {degree}")
        lam, val_err = validation_fn(X, y, degree, lambdas)
        print(f"Meilleur lambda = {lam:.4f} | Erreur validation = {val_err:.4f}")
        if val_err < best_val_error:
            best_val_error = val_err
            best_lambda = lam
            best_degree = degree
    print("Résultat final :")
    print(f"Degré optimal = {best_degree}, lambda optimal = {best_lambda}, MSE =
    ↪  {best_val_error:.4f}")
    return best_degree, best_lambda, best_val_error
```

Solutions : Validation croisée et Grid Search (5/5)

```
# 8. Paramètres
degrees = [3, 5, 7, 10]
lambdas = [1e-4, 1e-2, 1e-1, 1, 10]

# 9. Lancements
print("=== Validation simple ===")
grid_search(X, y, degrees, lambdas, simple_validation)

print("\n=== Validation croisée K-fold ===")
grid_search(X, y, degrees, lambdas, lambda X, y, d, l: k_fold_cv(X, y, d, l, k=5))
```

Résultats : Validation croisée et Grid Search

```
=== Validation simple ===
Démarrage de la recherche sur grille...

Degré testé : 3
lambda = 0.0001 | Validation MSE = 1.5087
lambda = 0.0100 | Validation MSE = 1.5086
lambda = 0.1000 | Validation MSE = 1.5086
lambda = 1.0000 | Validation MSE = 1.5083
lambda = 10.0000 | Validation MSE = 1.5224
Meilleur lambda = 1.0000 pour degré 3, erreur validation = 1.5083

Degré testé : 5
lambda = 0.0001 | Validation MSE = 0.9754
lambda = 0.0100 | Validation MSE = 0.9756
lambda = 0.1000 | Validation MSE = 0.9779
lambda = 1.0000 | Validation MSE = 1.0004
lambda = 10.0000 | Validation MSE = 1.2202
Meilleur lambda = 0.0001 pour degré 5, erreur validation = 0.9754

Degré testé : 7
lambda = 0.0001 | Validation MSE = 1.2624
lambda = 0.0100 | Validation MSE = 1.2620
lambda = 0.1000 | Validation MSE = 1.2588
lambda = 1.0000 | Validation MSE = 1.2377
lambda = 10.0000 | Validation MSE = 1.3242
Meilleur lambda = 1.0000 pour degré 7, erreur validation = 1.2377
```

Résultats : Validation croisée et Grid Search

```
Degré testé : 10
lambda = 0.0001 | Validation MSE = 0.5748
lambda = 0.0100 | Validation MSE = 0.5761
lambda = 0.1000 | Validation MSE = 0.5879
lambda = 1.0000 | Validation MSE = 0.7077
lambda = 10.0000 | Validation MSE = 1.2274
Meilleur lambda = 0.0001 pour degré 10, erreur validation = 0.5748
Résultat final :
 Degré optimal = 10, lambda optimal = 0.0001, Erreur validation = 0.5748
=== Validation croisée à 5 plis ===
Démarrage de la recherche sur grille...
```

Résultats : Validation croisée et Grid Search

```
Degré testé : 3
lambda = 0.0001 | Moyenne MSE (k-fold) = 0.9046
lambda = 0.0100 | Moyenne MSE (k-fold) = 0.9046
lambda = 0.1000 | Moyenne MSE (k-fold) = 0.9044
lambda = 1.0000 | Moyenne MSE (k-fold) = 0.9031
lambda = 10.0000 | Moyenne MSE (k-fold) = 0.9022
Meilleur lambda = 10.0000 pour degré 3, erreur validation = 0.9022

Degré testé : 5
lambda = 0.0001 | Moyenne MSE (k-fold) = 0.6028
lambda = 0.0100 | Moyenne MSE (k-fold) = 0.6028
lambda = 0.1000 | Moyenne MSE (k-fold) = 0.6026
lambda = 1.0000 | Moyenne MSE (k-fold) = 0.6015
lambda = 10.0000 | Moyenne MSE (k-fold) = 0.6323
Meilleur lambda = 1.0000 pour degré 5, erreur validation = 0.6015

Degré testé : 7
lambda = 0.0001 | Moyenne MSE (k-fold) = 0.6410
lambda = 0.0100 | Moyenne MSE (k-fold) = 0.6409
lambda = 0.1000 | Moyenne MSE (k-fold) = 0.6403
lambda = 1.0000 | Moyenne MSE (k-fold) = 0.6363
lambda = 10.0000 | Moyenne MSE (k-fold) = 0.6857
Meilleur lambda = 1.0000 pour degré 7, erreur validation = 0.6363
```

Résultats : Validation croisée et Grid Search

```
Degré testé : 10
lambda = 0.0001 | Moyenne MSE (k-fold) = 0.4677
lambda = 0.0100 | Moyenne MSE (k-fold) = 0.4677
lambda = 0.1000 | Moyenne MSE (k-fold) = 0.4680
lambda = 1.0000 | Moyenne MSE (k-fold) = 0.4789
lambda = 10.0000 | Moyenne MSE (k-fold) = 0.6533
Meilleur lambda = 0.0001 pour degré 10, erreur validation = 0.4677

Résultat final :
 Degré optimal = 10, lambda optimal = 0.0001, Erreur validation = 0.4677
(10, 0.0001, np.float64(0.4676567358352758))
```

Table des Matières

Décomposition en Biais Variance

Preuve détaillée de la décomposition biais-variance

- On considère un modèle de régression avec une relation entre la variable cible y et l'entrée $\boldsymbol{x}$ définie par :

$$y = f(\boldsymbol{x}) + \epsilon, \quad \text{où} \quad \mathbb{E}[\epsilon] = 0, \quad \text{Var}(\epsilon) = \sigma^2.$$

- Notre objectif est d'étudier l'erreur quadratique moyenne (**EQM**) en un point donné $\boldsymbol{x}$:

$$\mathbb{E}_{\mathcal{D}}[(y - \hat{f}(\boldsymbol{x}))^2].$$

Étape 1 : Développement de l'erreur quadratique moyenne

- On remplace y par son expression en fonction de $f(\boldsymbol{x})$ et du bruit ϵ :

$$\mathbb{E}_{\mathcal{D}}[(y - \hat{f}(\boldsymbol{x}))^2] = \mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) + \epsilon - \hat{f}(\boldsymbol{x}))^2].$$

- En développant le carré :

$$\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) + \epsilon)^2].$$

- Décomposons ce carré :

$$\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2 + 2(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))\epsilon + \epsilon^2].$$

Étape 2 : Calcul de l'espérance terme par terme

- Par linéarité de l'espérance :

$$\mathbb{E}_{\mathcal{D}}[(y-\hat{f}(\boldsymbol{x}))^2] = \mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x})-\hat{f}(\boldsymbol{x}))^2]+2\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x})-\hat{f}(\boldsymbol{x}))\epsilon]+\mathbb{E}[\epsilon^2].$$

- Puisque $\mathbb{E}[\epsilon] = 0$ et que ϵ est indépendant de $\hat{f}(\boldsymbol{x})$, on a :

$$\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))\epsilon] = \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})] \cdot \mathbb{E}[\epsilon] = 0.$$

- Il reste donc :

$$\mathbb{E}_{\mathcal{D}}[(y - \hat{f}(\boldsymbol{x}))^2] = \mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2] + \mathbb{E}[\epsilon^2].$$

- Or, la variance du bruit est $\mathbb{E}[\epsilon^2] = \sigma^2$, donc :

$$\mathbb{E}_{\mathcal{D}}[(y - \hat{f}(\boldsymbol{x}))^2] = \mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2] + \sigma^2.$$

Étape 3 : Décomposition du premier terme (biais et variance)

- On ajoute et soustrait $\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})]$ dans $(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))$:

$$f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) = (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})]) + (\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] - \hat{f}(\boldsymbol{x})).$$

- En élevant au carré et en prenant l'espérance :

$$\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2] + 2\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])(\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] - \hat{f}(\boldsymbol{x}))] + \mathbb{E}_{\mathcal{D}}[(\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] - \hat{f}(\boldsymbol{x}))^2].$$

- Le deuxième terme s'annule car :

$$\begin{aligned} &\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])(\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] - \hat{f}(\boldsymbol{x}))] \\ &= (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})]) \cdot \mathbb{E}_{\mathcal{D}}[\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] - \hat{f}(\boldsymbol{x})] \\ &= 0. \end{aligned}$$

- Il reste donc :

$$\mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2] = (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2 + \mathbb{E}_{\mathcal{D}}[(\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] - \hat{f}(\boldsymbol{x}))^2].$$

Conclusion : Décomposition finale

- Ces termes correspondent respectivement :
  - Au biais au carré :

$$\text{Biais}^2 = (f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2.$$

  - À la variance :

$$\text{Variance} = \mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2].$$

- On réinjecte dans l'expression de l'erreur quadratique moyenne :

$$\mathbb{E}_{\mathcal{D}}[(y - \hat{f}(\boldsymbol{x}))^2] = \underbrace{(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2}_{\text{Biais}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Erreur Irréductible}}.$$