Université Djilali Liabès de Sidi Bel Abbès
Faculté des Sciences Exactes
Département d'Informatique

Année Universitaire 2025-2026

Module : Machine Learning

4ème Année Ingénieur - Intelligence Artificielle - Groupe 2

Rapport TP 1 : Régression Linéaire et Polynomiale

Réalisé par :

Abdelhakem Abdelhak
Harmel Rayan

24 octobre 2025

Table des matières

Table des figures

Listings

1 Introduction

Ce rapport présente les solutions et les analyses des exercices réalisés lors du TP 1 du module de Machine Learning. L'objectif principal de ce TP est d'explorer et d'implémenter différents types de modèles de régression : la régression linéaire simple (par moindres carrés ordinaires et descente de gradient), la régression linéaire multiple et la régression polynomiale. Les exercices utilisent le langage Python avec les bibliothèques NumPy, Pandas, Scikit-learn et Matplotlib pour les calculs, les comparaisons et les visualisations.

2 Exercice 1 : Régression Linéaire Simple (Prix vs Ventes)

2.1 Objectif

Analyser la relation entre le prix de vente d'un produit (variable explicative X) et le nombre d'unités vendues (variable expliquée Y) dans huit régions différentes. Implémenter un modèle de régression linéaire simple $Y = \beta_0 + \beta_1 X + \epsilon$ en utilisant la méthode des moindres carrés (OLS) et la descente de gradient. Évaluer le modèle à l'aide de la somme des carrés résiduels (RSS) et du coefficient de détermination R^2.

2.2 Données et Visualisation

Les données d'entraînement fournies sont :
— Prix (X) : [420, 380, 350, 400, 440, 380, 450, 420]
— Ventes (Y) : [5.5, 6.0, 6.5, 6.0, 5.0, 6.5, 4.5, 5.0]

2.2.1 Code Python : Importations et Données

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

# 1. Donnees d'entrainement
X = np.array([420, 380, 350, 400, 440, 380, 450, 420], dtype=float)
Y = np.array([5.5, 6.0, 6.5, 6.0, 5.0, 6.5, 4.5, 5.0], dtype=float)
n = len(X)
```

Listing 1 – Importations et définition des données d'entraînement (exo1.ipynb)

2.2.2 Code Python et Graphe : Visualisation du Nuage de Points

```python
# 2. representation nuage
plt.scatter(X, Y, color='blue', s=80)
plt.title("Nuage de points : Prix vs Ventes")
plt.xlabel("Prix (X)")
plt.ylabel("Ventes (Y)")
plt.grid(True)
plt.show()
#quand le prix augmente, les ventes diminuent
```

Listing 2 – Code pour la représentation du nuage de points (exo1.ipynb)

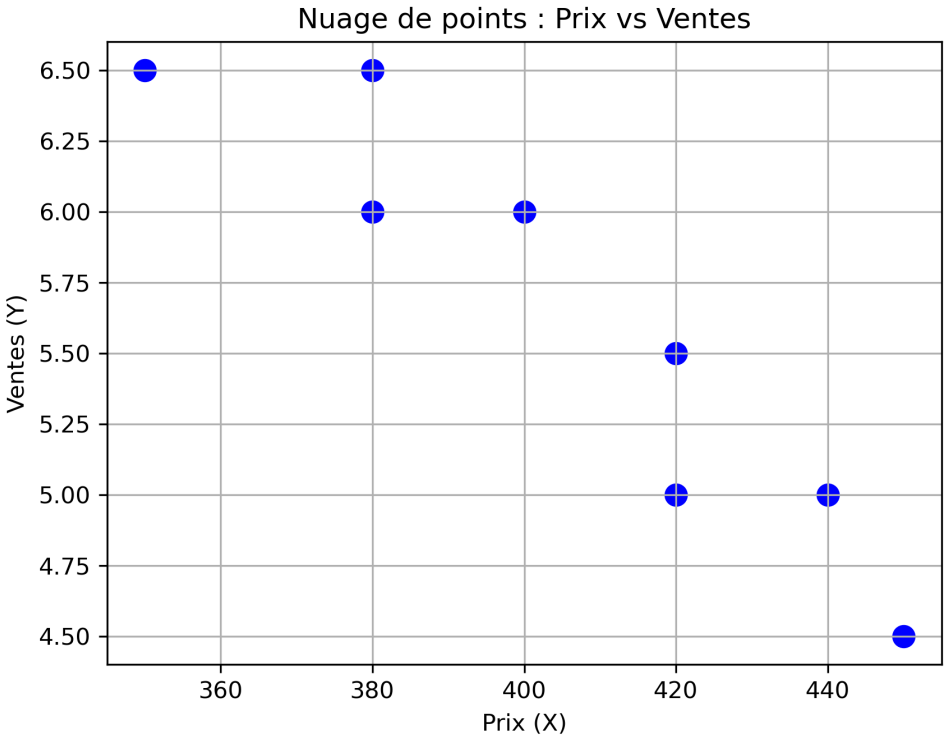

FIGURE 1 – Nuage de points : Prix vs Ventes (Données d'entraînement)

Le nuage de points (Figure 1) montre une tendance générale : lorsque le prix augmente, le nombre de ventes tend à diminuer. Cela suggère qu'une relation linéaire négative pourrait être appropriée pour modéliser ces données.

2.3 Méthode des Moindres Carrés (OLS)

2.3.1 Théorie

La méthode OLS cherche à minimiser la somme des carrés des erreurs entre les valeurs observées et les valeurs prédites. Les coefficients β_0 (ordonnée à l'origine) et β_1 (pente) sont calculés analytiquement par :

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

où $\bar{x}$ et $\bar{y}$ sont les moyennes respectives de X et Y.

2.3.2 Code Python : Calcul OLS

```python
# 3.calcul de beta0 et beta1 analytiquement
X_mean, Y_mean = np.mean(X), np.mean(Y)
beta1 = np.sum((X - X_mean)*(Y - Y_mean)) / np.sum((X - X_mean)**2)
beta0 = Y_mean - beta1 * X_mean

print(f"       = {beta0:.4f}")
print(f"       = {beta1:.6f}")
```

Listing 3 – Calcul analytique de et (exo1.ipynb)

2.3.3 Résultats OLS

Les calculs effectués donnent les coefficients suivants :

```
      = 13.9781
      = -0.020625
```

L'équation de la droite de régression estimée est donc :

$$\hat{Y} = 13.9781 - 0.020625X$$

2.3.4 Code Python et Graphe : Tracé de la droite OLS

```python
# 4. Tracer la droite de regression obtenue
Y_pred = beta0 + beta1 * X

plt.scatter(X, Y, color='blue', label='Donn es r elles')
plt.plot(X, Y_pred, color='red', label='Droite de r gression')
plt.title("Droite de r gression lin aire")
plt.xlabel("Prix (X)")
plt.ylabel("Ventes (Y)")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 4 – Code pour le tracé de la droite de régression OLS (exo1.ipynb)

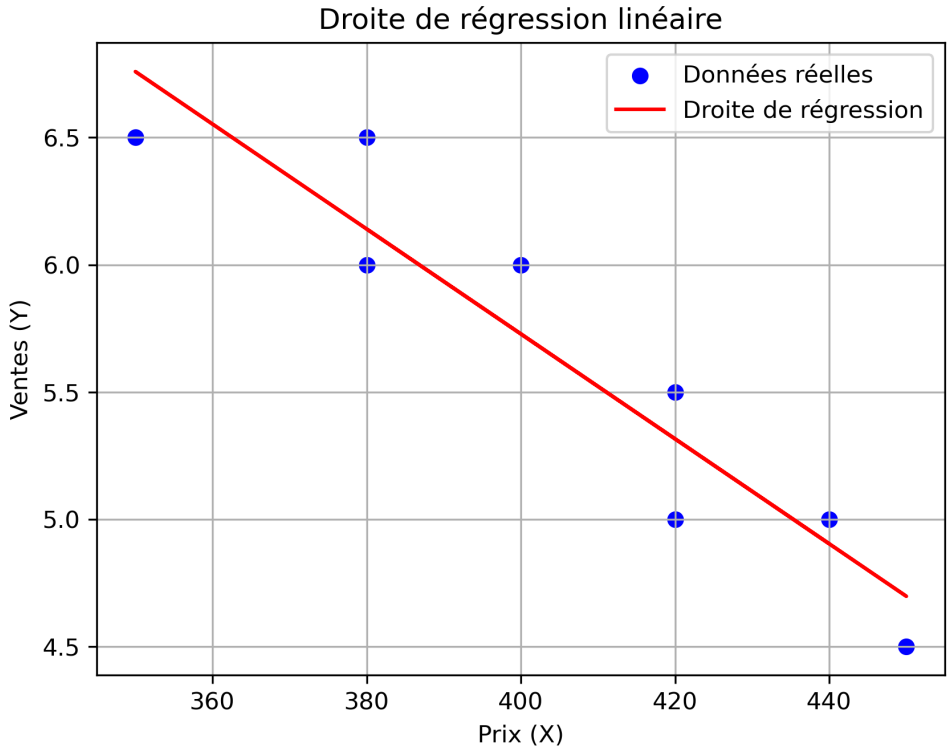

FIGURE 2 – Droite de régression linéaire (OLS) ajustée aux données d'entraînement.

La droite de régression (Figure 2) visualise cet ajustement linéaire aux données d'entraînement.

2.4 Descente de Gradient

2.4.1 Théorie

La descente de gradient est un algorithme itératif pour minimiser une fonction de coût $J(\beta)$. Pour la régression linéaire, la fonction de coût (erreur quadratique moyenne divisée par 2) est :

$$J(\beta_0, \beta_1) = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 = \frac{1}{2n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_i - y_i)^2$$

Les paramètres sont mis à jour itérativement selon les règles :

$$\beta_0 := \beta_0 - \alpha \frac{\partial J}{\partial \beta_0} = \beta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

$$\beta_1 := \beta_1 - \alpha \frac{\partial J}{\partial \beta_1} = \beta_1 - \alpha \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_i$$

où α est le taux d'apprentissage. L'implémentation utilise des données normalisées (X_{norm}) et Y centré ($Y_{centered}$) pour améliorer la convergence.

2.4.2 Code Python : Normalisation et Descente de Gradient

```
# 6. Normalisation (centrage-reduction)
X_std = np.std(X)
Y_std = np.std(Y)

X_norm = (X - X_mean) / X_std
Y_norm = (Y - Y_mean) / Y_std # Y_norm n'est pas utilis  par la descente de
    gradient fournie

# Coefficients normalis s  calcul s analytiquement (pour info)
beta1_prime = beta1 * (X_std / Y_std)
beta0_prime = 0

print(f"         (analytique) = {beta0_prime:.4f}")
print(f"         (analytique) = {beta1_prime:.4f}")
```

Listing 5 – Normalisation des données (exo1.ipynb)

```
# 7. Descente de gradient pour estimer       et
Y_centered = Y - Y_mean # Utilisation de Y centr
alpha = 0.1       # taux d'apprentissage
max_iter = 5000
tol = 1e-8

b0, b1 = 0.0, 0.0 # Initialisation pour Y_centered = b0 + b1 * X_norm
J_history = []

for i in range(max_iter):
    Y_pred_gd = b0 + b1 * X_norm
    error = Y_pred_gd - Y_centered # Erreur par rapport    Y centr
    grad_b0 = (1/n) * np.sum(error)
    grad_b1 = (1/n) * np.sum(error * X_norm)
    b0 -= alpha * grad_b0
    b1 -= alpha * grad_b1
    J = (1/(2*n)) * np.sum((Y_centered - (b0 + b1*X_norm))**2) # Co t par
    rapport    Y centr
    J_history.append(J)
    if i > 0 and abs(J_history[-2] - J_history[-1]) < tol:
```

```
        print(f"Convergence atteinte      l  itration   {i}")
        break

print(f" b   (GD pour Y_centered) = {b0:.6f}") # b0 devrait converger vers 0
print(f" b   (GD pour Y_centered) = {b1:.6f}") # b1 correspond          * X_std
    / std(Y) -> non, juste       * X_std
```

Listing 6 – Implémentation de la Descente de Gradient (exo1.ipynb)

2.4.3 Code Python et Graphe : Visualisation de la fonction de coût

```
# 8. Visualisation de la fonction de cout J
plt.plot(J_history, color='purple')
plt.title(" volution  de la fonction de co t J")
plt.xlabel("It rations")
plt.ylabel("J( b  ,  b  )")
plt.grid(True)
plt.show()
```

Listing 7 – Code pour l'évolution de la fonction de coût J (exo1.ipynb)

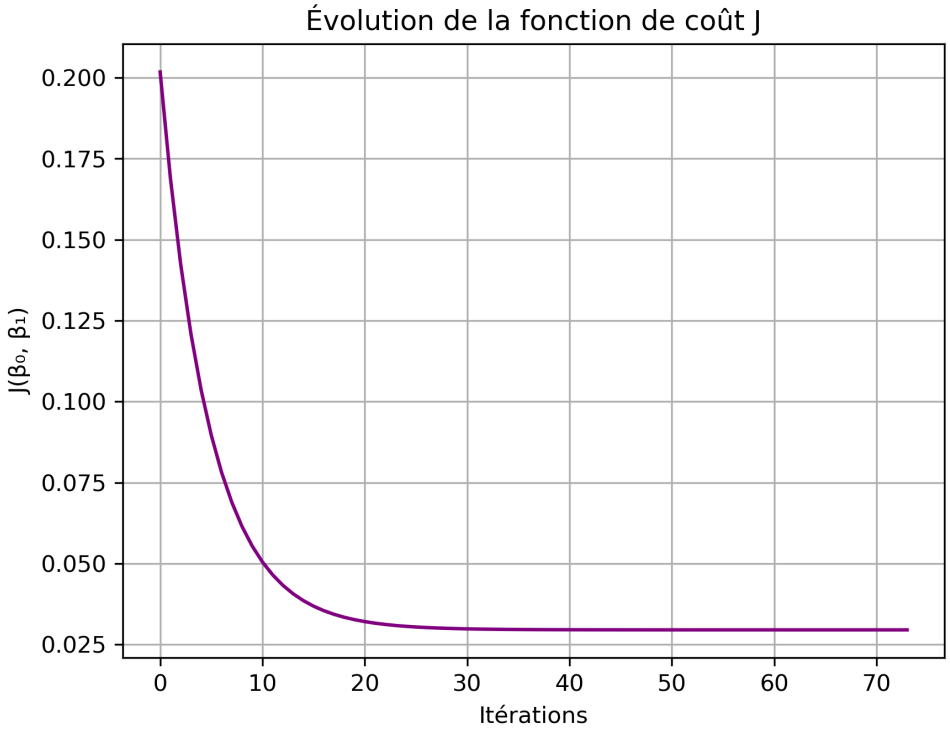

FIGURE 3 – Évolution de la fonction de coût J pendant la descente de gradient.

2.4.4 Résultats de la Descente de Gradient

Avec $\alpha = 0.1$, tolérance 10^{-8}, initialisation à 0 :

— Convergence atteinte à l'itération 73.

— Coefficients estimés pour $Y_{centered} \approx b_0 + b_1 X_{norm}$:

```
b   (GD pour Y_centered) = -0.000000
b   (GD pour Y_centered) = -0.651952
```

— L'évolution de la fonction de coût J (Figure 3) montre une convergence rapide.

2.4.5 Code Python : Comparaison OLS vs GD

```
# 9. Comparaison Analytique vs Gradient

# Conversion des coefficients GD vers  chelle  r elle (  _gd  ,  _gd  )
beta1_gd_real = b1 / X_std
beta0_gd_real = Y_mean - beta1_gd_real * X_mean

# Recalcul du RSS avec les coefficients GD  l' chelle  r elle
Y_pred_gd_real = beta0_gd_real + beta1_gd_real * X
RSS_gd_real =  np.sum((Y - Y_pred_gd_real)**2)

print(" COMPARAISON ")
print(f"Analytique :      ={beta0:.4f},      ={beta1:.6f}, RSS={RSS:.4f}") # RSS
    d fini plus haut
print(f"Gradient    :      ={beta0_gd_real:.4f},      ={beta1_gd_real:.6f}, RSS
   ={RSS_gd_real:.4f}")
```

Listing 8 – Comparaison des coefficients et RSS (exo1.ipynb)

2.4.6 Résultats Comparaison

```
 COMPARAISON
Analytique :      =13.9781,      =-0.020625, RSS=0.4719
Gradient    :      =13.9781,      =-0.020625, RSS=0.4719
```

Après conversion à l'échelle réelle, les coefficients et le RSS obtenus par descente de gradient sont très proches de ceux obtenus par la méthode OLS analytique.

2.4.7 Code Python et Graphe : Visualisation Finale

```
# 10. Visualisation finale
plt.figure(figsize=(10,5))
plt.scatter(X, Y, color='blue', label='Donn es')
plt.plot(X, beta0 + beta1*X, color='red', label='Analytique (OLS)')
plt.plot(X, beta0_gd_real + beta1_gd_real*X, color='green', linestyle='--',
    label='Gradient')
plt.title("Comparaison des droites de r gression")
plt.xlabel("Prix (X)")
plt.ylabel("Ventes (Y)")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 9 – Code pour la comparaison visuelle des droites de régression (exo1.ipynb)

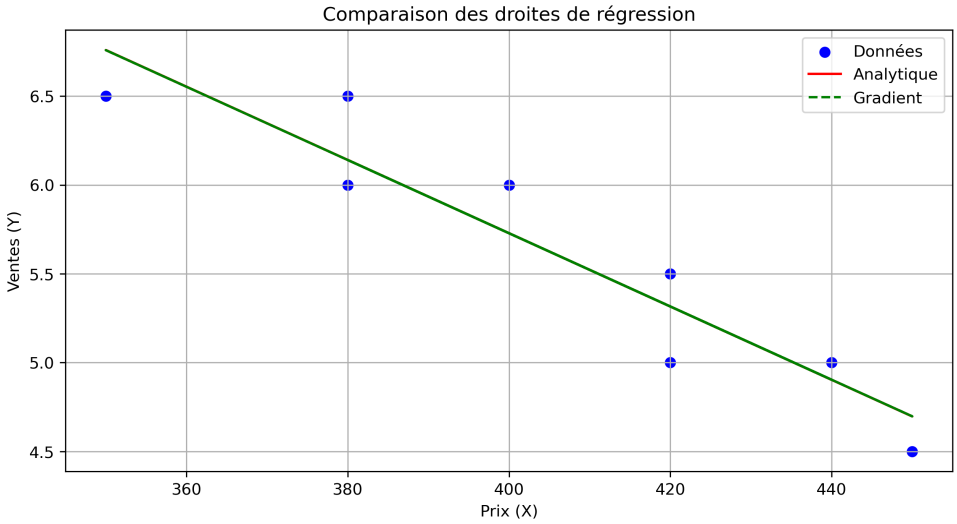

FIGURE 4 – Comparaison visuelle des droites de régression OLS et Descente de Gradient.

Le graphique (Figure 4) confirme visuellement la quasi-superposition des droites obtenues par les deux méthodes.

2.5 Évaluation du Modèle (Suite)

2.5.1 Erreur Résiduelle (RSS) sur l'entraînement

$$RSS_{OLS} = \sum_{i=1}^{n}(y_i - \hat{y}_{OLS,i})^2 \approx 0.4719$$

```python
# 5. Calcul des performances du modele (RSS)
RSS_train = np.sum((Y - Y_pred)**2) # Y_pred from OLS
print(f"RSS (entra nement) = {RSS_train:.4f}")
```

Listing 10 – Calcul du RSS sur l'entraînement (exo1.ipynb)

2.5.2 Test du modèle et Coefficient de Détermination R^2

Le modèle est testé sur un nouvel ensemble de données :
— Prix (X_{test}) : [360, 380, 400, 420, 440, 450]
— Ventes (Y_{test}) : [6.6, 6.04, 5.73, 5.4, 4.84, 4.6]
Le coefficient de détermination R^2 mesure la proportion de la variance de Y expliquée par le modèle X.

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Le notebook `exo1.ipynb` recalcule β_0 et β_1 *sur l'ensemble de test* puis calcule le R^2 sur ce même ensemble de test.

```python
# tester le modele ensemble de test
Xprime = np.array([360,380,400,420,440,450],dtype=float)
Yprime= np.array([6.6 , 6.04 , 5.73  ,5.4  , 4.84 , 4.6],dtype=float)
m = len(Xprime)

# --- Visualisation Test Set ---
plt.scatter(Xprime, Yprime, color='green', s=80)
plt.title("Nuage de points : Prix vs Ventes (Test Set)")
```

```
plt.xlabel("Prix (X)")
plt.ylabel("Ventes (Y)")
plt.grid(True)
plt.show()

# Recalcul des coefficients sur le test set
xprime_mean= np.mean(Xprime)
yprime_mean  =np.mean(Yprime)
bet1_test = np.sum((Xprime-xprime_mean)*(Yprime - yprime_mean)) / np.sum((Xprime - xprime_mean)**2)
bet0_test = yprime_mean - bet1_test * xprime_mean

print(f"beta0 (test) = { bet0_test: .4f}")
print(f"beta1 (test) = { bet1_test : .6f}")

# Calcul R^2 sur le test set en utilisant les coefficients du test set
Yprime_pred_test = bet0_test + bet1_test*Xprime
Rss_test = np.sum((Yprime - Yprime_pred_test)**2)
tss_test = np.sum((Yprime-yprime_mean)**2)
r2_test = 1 - Rss_test /tss_test

print(f"RSS (test)={ Rss_test :.6f}",f"TSS (test)={tss_test:.6f}",f"R2 (test)={ r2_test :.6f}")
```

Listing 11 – Calcul des coefficients et R^2 sur l'ensemble de test (exo1.ipynb)

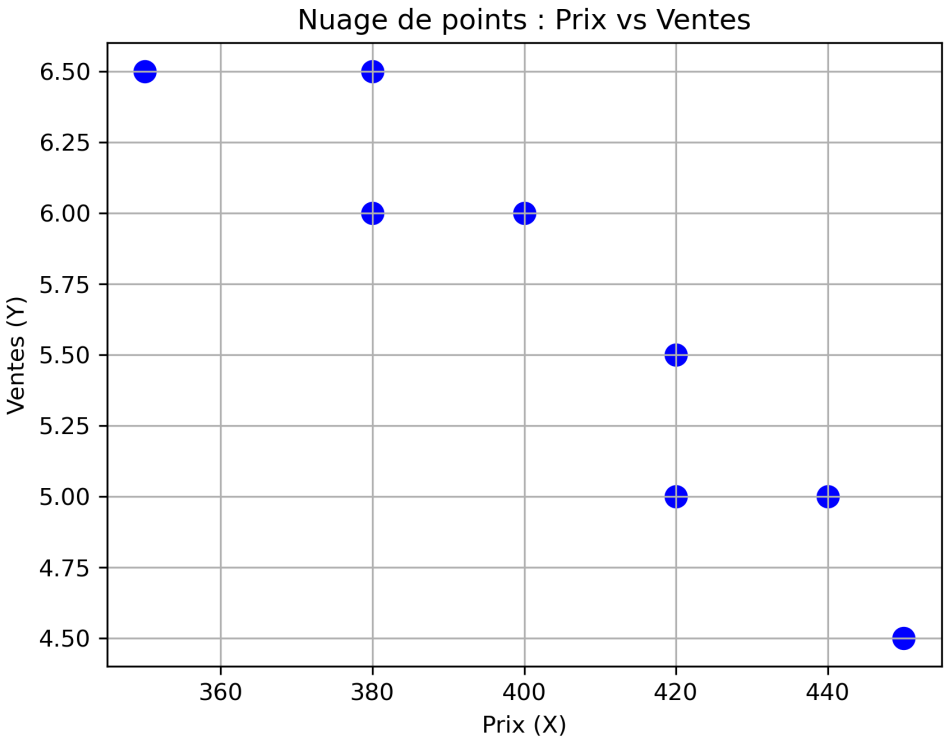

FIGURE 5 – Nuage de points : Prix vs Ventes (Données de test).

Résultats sur l'ensemble de test (avec coefficients recalculés sur le test) :

```
beta0 (test) =  14.2577
beta1 (test) = -0.021362
RSS(test)=0.026804 TSS(test)=2.802750 R2(test)=0.990436
```

Le R^2 très élevé (0.9904) indique un excellent ajustement du modèle linéaire *sur ces données de test spécifiques*. L'évaluation standard consisterait à utiliser les β_0, β_1 de l'entraînement pour

prédire sur le test set et calculer le R^2.

2.5.3 Fonctions prédéfinies

Oui, des bibliothèques Python comme Scikit-learn (‘sklearn.linear$_m$odel.LinearRegression‘)ouStatsmodelsp

3 Exercice 2 : Régression Linéaire Multiple

3.1 Objectif

Expliquer la note moyenne d'un étudiant (Y) en fonction du nombre d'heures de révision (X_1) et du nombre d'heures de sommeil (X_2). Ajuster un modèle de régression linéaire multiple $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$ par la méthode OLS matricielle en utilisant NumPy. Comparer avec les fonctions prédéfinies de Scikit-learn, évaluer avec R^2 et RMSE, et faire une prédiction.

3.2 Données

Les données observées sur trois étudiants sont :

Étudiant	X_1 (heures de révision)	X_2 (sommeil)	Y (note)
1	4	6	10
2	2	5	7
3	3	7	9

3.2.1 Code Python : Importations et Données

```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# 1. Definition des donnees
Y = np.array([10, 7, 9])
X1 = np.array([4, 2, 3])
X2 = np.array([6, 5, 7])
```

Listing 12 – Importations et définition des données (exo2.ipynb)

3.3 Méthode des Moindres Carrés Matricielle

3.3.1 Théorie

Le modèle s'écrit sous forme matricielle $Y = X\beta + \epsilon$, où Y est le vecteur des notes, X est la matrice de conception, et β est le vecteur des coefficients.

$$Y = \begin{pmatrix} 10 \\ 7 \\ 9 \end{pmatrix}, \quad X = \begin{pmatrix} 1 & 4 & 6 \\ 1 & 2 & 5 \\ 1 & 3 & 7 \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix}$$

La solution OLS est donnée par :

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

3.3.2 Code Python : Construction Matrice X et Calcul OLS

```
# 2. Construction de la Matrice de Conception X
X_design = np.column_stack((np.ones(len(Y)), X1, X2))
print("Vecteur des Notes (Y):\n", Y)
print("\nMatrice de Conception (X_design):\n", X_design)
```

Listing 13 – Construction de la matrice de conception (exo2.ipynb)

```
# 3. calcul du ensemble beta
XT = X_design.T
XTX = XT @ X_design
XTX_inv = np.linalg.inv(XTX)
XTY = XT @ Y
beta_hat = XTX_inv @ XTY

print("Matrice (X^T X) :\n", XTX)
print("\nMatrice (X^T X)^-1 :\n", XTX_inv.round(4))
print("\nVecteur (X^T Y) :\n", XTY)
print("\nCoefficients estim s (beta_hat) :\n", beta_hat.round(4))

print(f"\n quation du mod le estim  :    = {beta_hat[0]:.2f} + {beta_hat
    [1]:.2f}*X1 + {beta_hat[2]:.2f}*X2")
```

Listing 14 – Calcul matriciel des coefficients (exo2.ipynb)

3.3.3 Résultats OLS

Le calcul matriciel donne les coefficients estimés $\hat{\beta}_0 \approx 2.67$, $\hat{\beta}_1 \approx 1.33$, $\hat{\beta}_2 \approx 0.33$. L'équation du modèle estimé est :

$$\hat{Y} = 2.67 + 1.33X_1 + 0.33X_2$$

3.4 Comparaison avec Scikit-learn

3.4.1 Code Python : Comparaison Scikit-learn

```
# 3. Comparaison avec la Fonction de Regression Predefinie
X_data_only = X_design[:, 1:] # Variables X1, X2 sans la colonne de 1

model_sklearn = LinearRegression()
model_sklearn.fit(X_data_only, Y)

beta0_sklearn = model_sklearn.intercept_
beta1_sklearn, beta2_sklearn = model_sklearn.coef_

print("--- Comparaison des r sultats (scikit-learn) ---")
print(f"Intercept ( 0 ) estim  manuellement : {beta_hat[0]:.4f}")
print(f"Intercept ( 0 ) par sklearn : {beta0_sklearn:.4f}")
print(f"\nCoefficient X1 ( 1 ) estim  manuellement : {beta_hat[1]:.4f}")
print(f"Coefficient X1 ( 1 ) par sklearn : {beta1_sklearn:.4f}")
print(f"\nCoefficient X2 ( 2 ) estim  manuellement : {beta_hat[2]:.4f}")
print(f"Coefficient X2 ( 2 ) par sklearn : {beta2_sklearn:.4f}")

# La tr s petite diff rence est due aux erreurs d'arrondi
if np.allclose(beta_hat, np.array([beta0_sklearn, beta1_sklearn, beta2_sklearn])
    ):
    print("\n Les r sultats du calcul matriciel et de scikit-learn sont
    coh rents.")
else:
    print("\n Les r sultats sont diff rents.")
```

Listing 15 – Comparaison avec Scikit-learn (exo2.ipynb)

3.4.2 Résultats Comparaison

Le notebook confirme que les coefficients obtenus via Scikit-learn sont cohérents avec ceux calculés manuellement.

3.5 Évaluation du Modèle

3.5.1 Théorie

Le R^2 et le RMSE sont calculés pour évaluer la qualité de l'ajustement.

$$R^2 = 1 - \frac{SCR}{SCT} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

$$RMSE = \sqrt{\frac{SCR}{n}} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n}}$$

3.5.2 Code Python : Calcul R² et RMSE

```
# 4. Calcul du R^2 et du RMSE
Y_hat = X_design @ beta_hat
Y_mean = np.mean(Y)
SCR = np.sum((Y - Y_hat)**2)
SCT = np.sum((Y - Y_mean)**2)
R_squared = 1 - (SCR / SCT) if SCT != 0 else 1
n = len(Y)
RMSE = np.sqrt(SCR / n)

print("--- Calcul de R   et RMSE ---")
print(f"Pr dictions du mod le (  ) : {Y_hat.round(4)}")
print(f"Moyenne de Y (  ) : {Y_mean:.4f}")
print(f"Somme des Carr s des R sidus (SCR) : {SCR:.4f}")
print(f"Somme Totale des Carr s (SCT) : {SCT:.4f}")
print(f"\nCoefficient de D termination (R  ) : {R_squared:.4f}")
print(f"Erreur Quadratique Moyenne (RMSE) : {RMSE:.4f}")
print("-----------------------------------------\n")

print("--- Discussion sur la Qualit  de l'Ajustement ---")
print(f"Le R   de {R_squared:.4f} indique qu'environ {R_squared*100:.2f}% de la
    variation de la note est expliqu e par les heures de r vision (X1) et de
    sommeil (X2).")
if np.isclose(R_squared, 1.0):
    print("Un R   de 1.0 indique un ajustement parfait aux donn es d'
    entra nement.")
    print("Attention: Avec seulement 3 points et 3 param tres, cela peut
    indiquer un surajustement.")
else:
    print("Un R   proche de 1 est id al. Une valeur plus faible sugg re que le
     mod le n'est pas parfait...")
print(f"Le RMSE de {RMSE:.4f} signifie que l'erreur moyenne de pr diction de la
     note est d'environ {RMSE:.2f} points.")
print("-------------------------------------------------\n")
```

Listing 16 – Calcul de R² et RMSE (exo2.ipynb)

3.5.3 Résultats Évaluation

— Prédictions $\hat{Y} = (10, 7, 9)$
— SCR (RSS) $= 0.0000$
— SCT (TSS) ≈ 4.6667
— $R^2 = 1.0000$
— $RMSE = 0.0000$

3.5.4 Discussion

Un $R^2 = 1.0$ et un $RMSE = 0.0$ indiquent que le modèle ajuste parfaitement les données d'entraînement. Cela signifie que 100% de la variation des notes dans cet échantillon est expliquée par les variables X_1 et X_2. Cependant, avec seulement 3 points de données et 3 coefficients (incluant l'intercept), cet ajustement parfait peut indiquer un surajustement (overfitting), et le modèle pourrait ne pas généraliser correctement sur de nouvelles données.

3.6 Prédiction

3.6.1 Code Python : Prédiction

```python
# 5. Estimation de la Note Moyenne Prevue
# Nouvelles donnees (X_new = [1, X1_new, X2_new])
X1_new = 3.5
X2_new = 6
X_new = np.array([1, X1_new, X2_new])

# Prediction
Y_predicted = X_new @ beta_hat

print(f"Nouvelles donn es : X1 = {X1_new}, X2 = {X2_new}")
print(f"Note moyenne pr vue (  ) : {Y_predicted:.2f}")
```

Listing 17 – Estimation de la note moyenne prévue (exo2.ipynb)

3.6.2 Résultat Prédiction

Pour un étudiant avec $X_1 = 3.5$ heures de révision et $X_2 = 6$ heures de sommeil, la note moyenne prévue est ≈ 9.33.

4 Exercice 3 : Régression Polynomiale

4.1 Objectif

Modéliser la relation entre deux variables quantitatives X et Y (ici, température et consommation d'énergie) à l'aide d'une régression polynomiale de degré 2 :

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$$

Estimer les coefficients $\beta_0, \beta_1, \beta_2$ par OLS, écrire l'équation du modèle et évaluer la qualité de l'ajustement avec R^2.

4.2 Données Simulées et Visualisation

Le notebook `EXO3.ipynb` utilise un jeu de données simulé :
— X (Température) : [1, 2, 3, 4, 5, 6, 7, 8]
— Y (Consommation) : [4.5, 3.9, 3.0, 2.8, 3.1, 3.6, 4.2, 5.0]

4.2.1 Code Python et Graphe : Données et Visualisation

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

# Donnees : temperature (X) et consommation (Y)
X = np.array([1, 2, 3, 4, 5, 6, 7, 8], dtype=float)
Y = np.array([4.5, 3.9, 3.0, 2.8, 3.1, 3.6, 4.2, 5.0], dtype=float)

plt.scatter(X, Y, color='blue', s=80)
plt.title("Nuage de points : Temp rature vs Consommation")
plt.xlabel("X (Temp rature)")
plt.ylabel("Y (Consommation)")
plt.grid(True)
plt.show()
```

Listing 18 – Importations, données simulées et nuage de points (EXO3.ipynb)

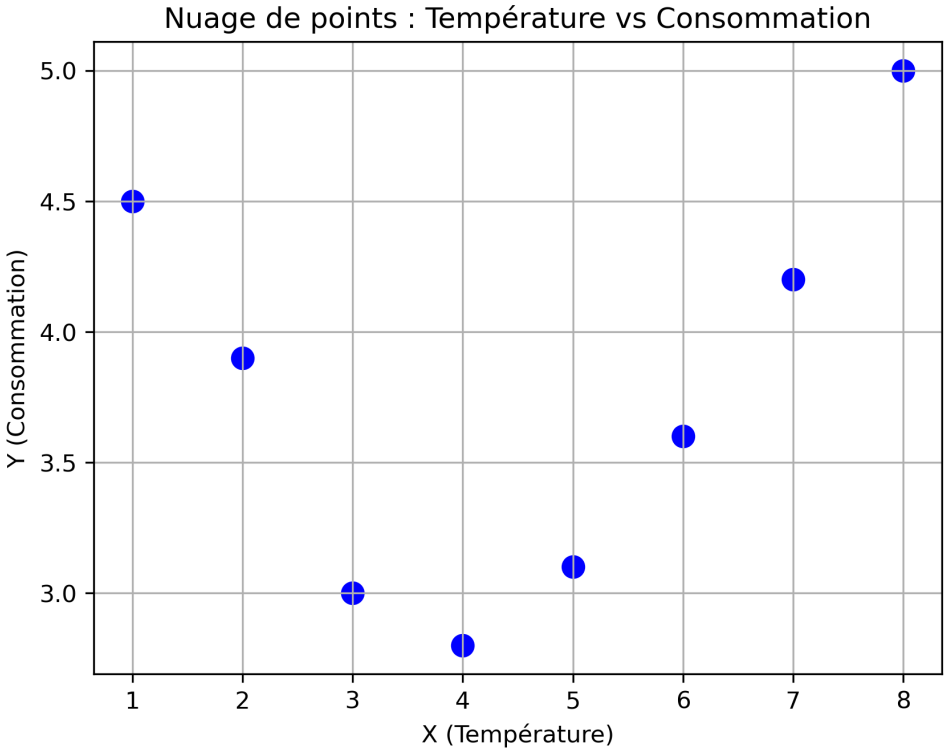

FIGURE 6 – Nuage de points : Température vs Consommation.

Le nuage de points initial (Figure 6) montre une relation non linéaire, possiblement quadratique.

4.3 Méthode OLS pour Régression Polynomiale

4.3.1 Théorie

Ce modèle est traité comme une régression linéaire multiple en créant la variable X^2. La matrice de conception X_{poly} et le calcul de $\hat{\beta} = (X_{poly}^T X_{poly})^{-1} X_{poly}^T Y$ sont utilisés pour estimer les coefficients.

4.3.2 Code Python : Construction Matrice et Calcul OLS

```python
# 2.Construction du modele polynomial
# Construction de la matrice du modele polynomial
X_poly = np.column_stack((np.ones(len(X)), X, X**2))
print("Matrice X_poly :\n", X_poly)

# 3.Calcul des coefficients  0 ,  1 ,  2
# Calcul des coefficients par moindres carres
beta = np.linalg.inv(X_poly.T @ X_poly) @ X_poly.T @ Y
beta0, beta1, beta2 = beta
print(f"      = {beta0:.4f}")
print(f"      = {beta1:.4f}")
print(f"      = {beta2:.4f}")
```

Listing 19 – Construction matrice polynomiale et calcul OLS (EXO3.ipynb)

4.3.3 Résultats OLS

Les coefficients estimés sont :

— $\hat{\beta}_0 \approx 5.6411$

— $\hat{\beta}_1 \approx -1.2708$

— $\hat{\beta}_2 \approx 0.1506$

L'équation du modèle polynomial estimé est :

$$\hat{Y} = 5.6411 - 1.2708X + 0.1506X^2$$

4.3.4 Code Python et Graphe : Prédictions et Tracé

```python
# 4.Predictions et trace de la courbe
# Predictions
Y_pred = beta0 + beta1*X + beta2*(X**2)

# Trace
plt.scatter(X, Y, color='blue', label='Donn es  r elles')
plt.plot(X, Y_pred, color='red', label='Mod le polynomial (degr  2)')
plt.title("R gression polynomiale de degr  2")
plt.xlabel("X (Temp rature)")
plt.ylabel("Y (Consommation)")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 20 – Prédictions et tracé de la courbe polynomiale (EXO3.ipynb)

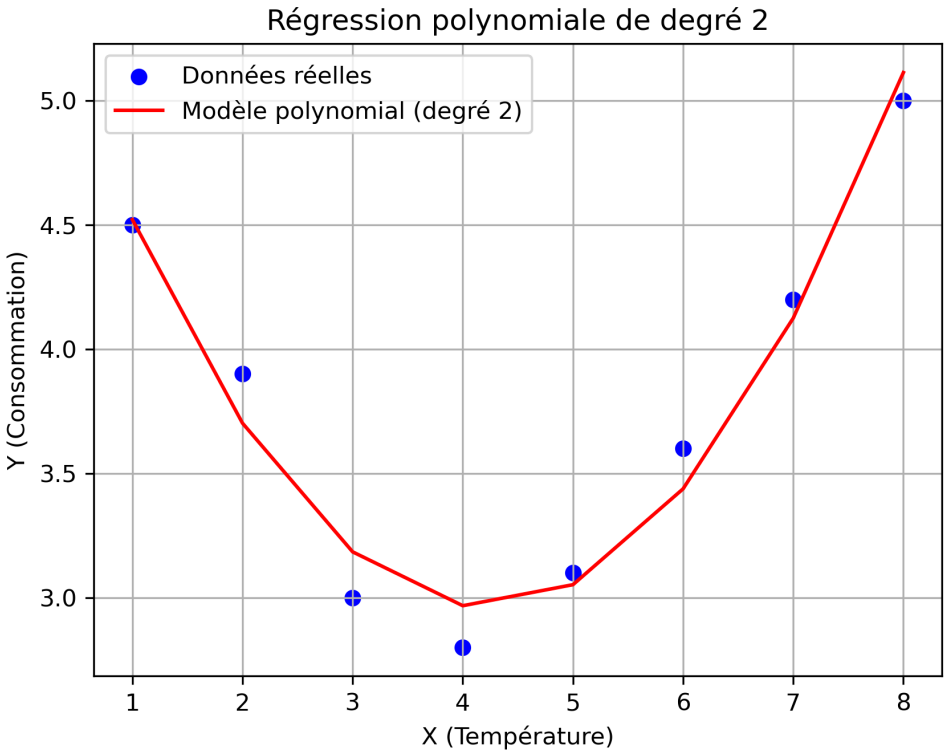

FIGURE 7 – Régression polynomiale de degré 2 ajustée aux données simulées.

Le tracé (Figure 7) montre que la courbe s'ajuste bien aux données.

4.4 Évaluation du Modèle

4.4.1 Code Python : Calcul R²

```python
# 5.Calcul du coefficient de determination R2
RSS = np.sum((Y - Y_pred)**2)
TSS = np.sum((Y - np.mean(Y))**2)
R2 = 1 - (RSS / TSS)

print(f"RSS = {RSS:.4f}")
print(f"TSS = {TSS:.4f}")
print(f"R    = {R2:.4f}")
```

Listing 21 – Calcul du coefficient R^2 (EXO3.ipynb)

4.4.2 Résultats Évaluation

— RSS ≈ 0.1486
— TSS ≈ 4.2588
— $R^2 \approx 0.9651$

Un R^2 de 0.9651 indique que le modèle polynomial de degré 2 explique environ 96.5% de la variance de la consommation d'énergie (Y) basée sur la température (X) dans cet échantillon simulé. C'est un très bon ajustement.

4.4.3 Code Python : Résumé Modèle

```
# 6.Resume du modele
print(f"  quation   du  mod le  :  Y = {beta0:.4f} + {beta1:.4f}X + {beta2:.4f}X  ")
```

Listing 22 – Affichage de l'équation finale (EXO3.ipynb)

5 Conclusion

Ce TP a permis de mettre en pratique les concepts fondamentaux de la régression linéaire simple, multiple et polynomiale. Nous avons implémenté les calculs des coefficients par la méthode analytique des moindres carrés (OLS) et par la méthode itérative de la descente de gradient, en observant la cohérence des résultats. L'évaluation des modèles à l'aide du RSS, du RMSE et du R^2 a permis de quantifier la qualité de l'ajustement aux données, tout en notant le risque de surajustement dans le cas de l'exercice 2. Les comparaisons avec les fonctions prédéfinies de Scikit-learn ont validé les implémentations. L'exercice sur la régression polynomiale a démontré l'application de la régression linéaire pour modéliser des relations non linéaires via la transformation des variables explicatives.