Nom : Ibrahim Lahlou

Partie 1 :

1. comment varie mse dans les 5 premières itérations? comment justifier ce résultat? il s'agit de quel problème?

nous constatons que mse diminue brusquement au début mais il arrive a atteindre une valeur minimal de 13.9 . Cela peut être dû à un learning rate élevé, ce qui peut provoquer des sauts importants dans l'optimisation des paramètres.

2. Qu'est-ce qui se passe si on augmente le nombre d'itérations (500)? Expliquer le résultat obtenu.

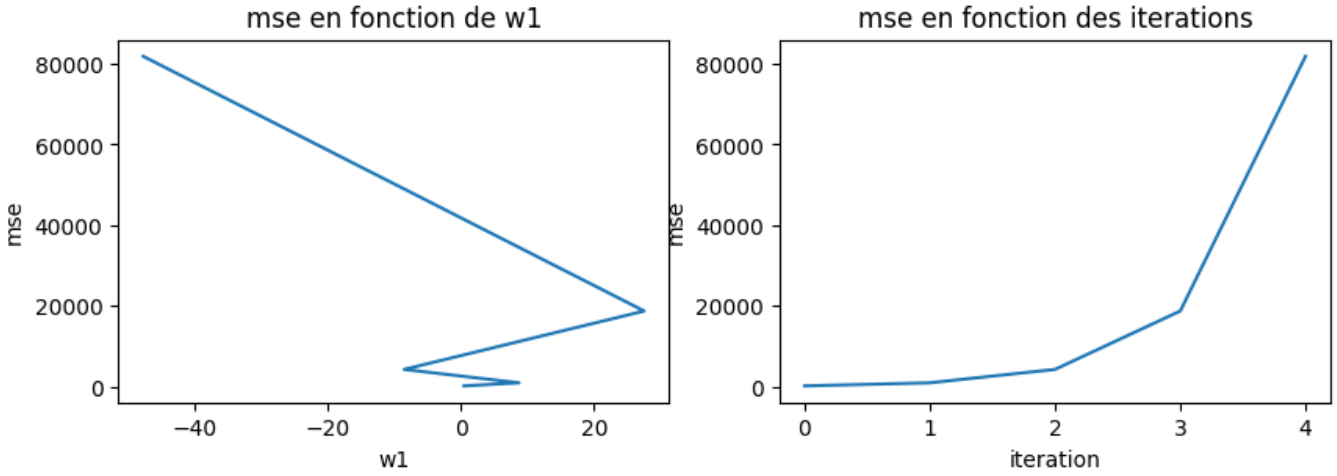
le modèle continuera à s'entraîner pendant un bout de temps et peux convergé a une solution optimal mais cette augmentation pourra juste avoi un impact minimal sur les performances du modèles

3. Quelles sont les recommandations à faire pour corriger ce problème?

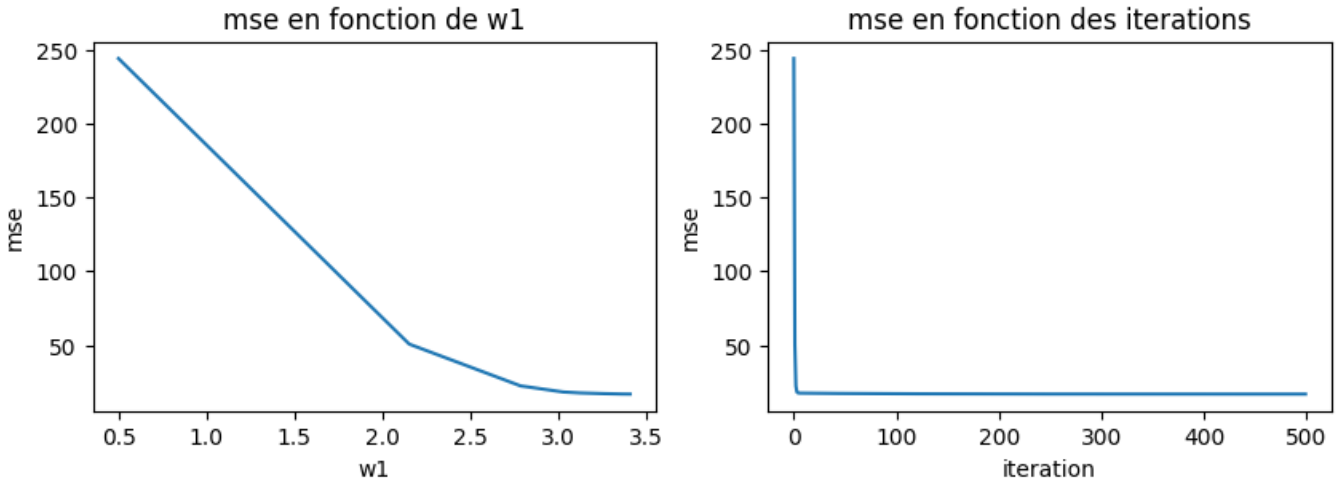
- Utilisé un bon learning rate souvent on utilisant des algorithme d'hyperparameter tunning
- Surveillé la courbe de MSE au fils du temps pour optimisé l'erreur

4. Donner à learning_rate la valeur 0.01 au lieu de 0.1 et refaire l'exécution. Qu'est-ce que vous constatez? Comment expliquer le résultat obtenu?

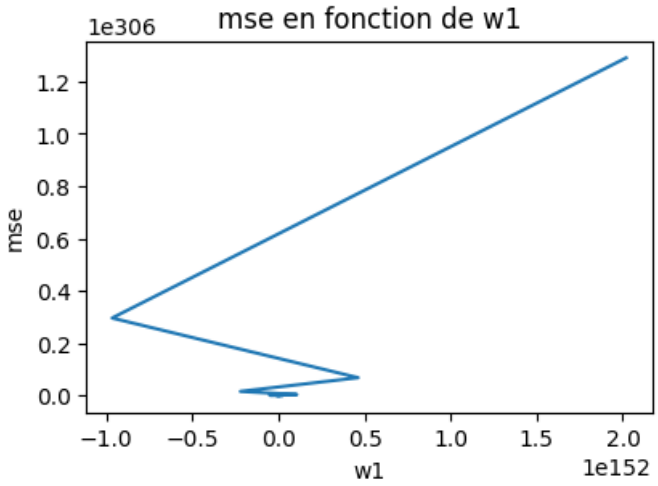
On constate que l'apprentissage du modèle se fait d'une manière brusque de tels façon qu'elle créer une oscillations

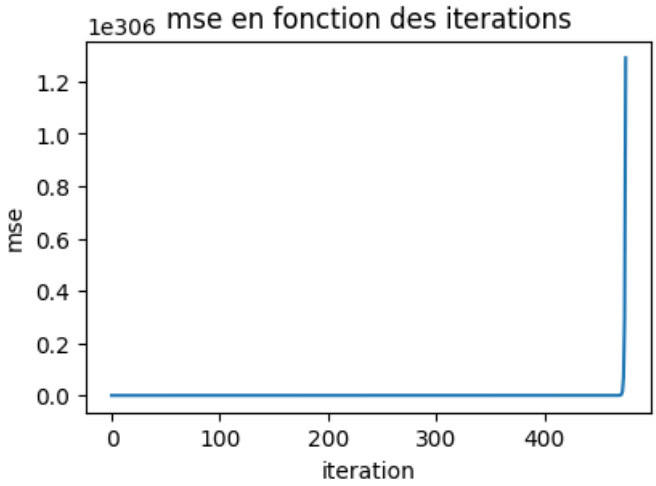


5. Augmenter le nombre d'epochs à 500, puis tracer les graphes à chaque 50 itération.
   expliquer les résultats obtenus
   avec un learning rate de 0.01 nous aurons deja optimisé le mse au début



avec un learning rate de 0.05 nous aurons un divergence accrue au niveau de mse vers l'ordre de 10^{306}





cela dit on aura pas besoin de faire autant d'itération pour savoir si l'algorithme Gradient Descent s'optimise

---

Partie 2:

1. Donner la forme du modèle à entraîner

le modèle est écrit ainsi

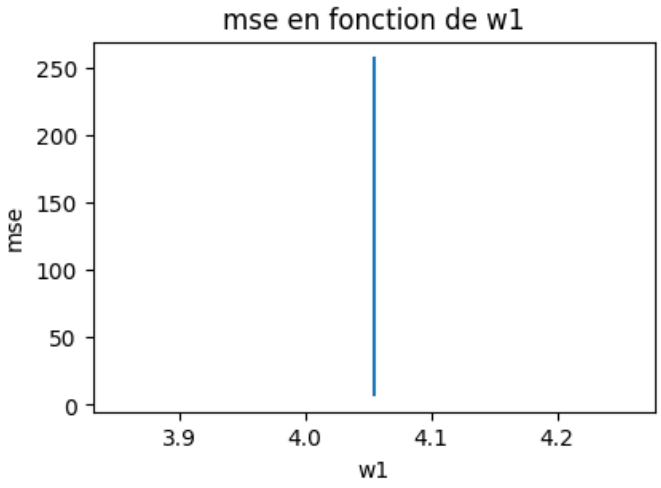
$$h(x_1, x_2, x_3) = bias + \sum_{i=1}^{3} w_i x_i$$

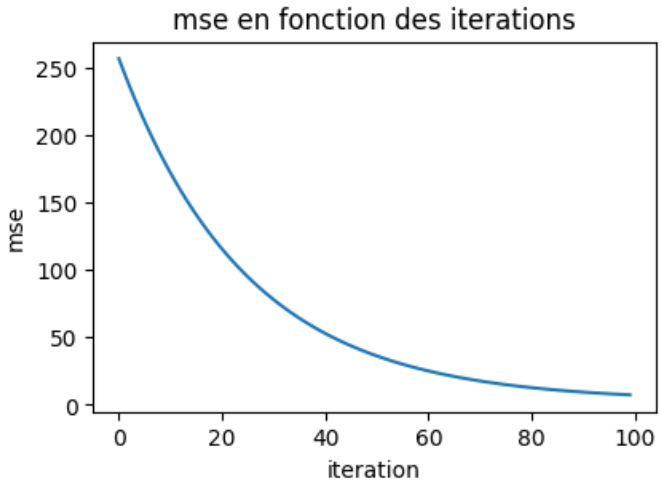
Le modèle `h` prédit les ventes en fonction de trois variables d'entrée : x_1 : `TV`, x_2 : `Radio`, et x_3 : `Newspaper`, avec les paramètres `bias`, `w1`, `w2`, et `w3`.

2. en reprenant la fonction train2, compléter la fonction train3 afin de trouver le modèle adéquat.

```python
x1,x2,x3=X[:,0],X[:,1],X[:,2]

def train3(x1, x2, x3, y):

    learning_rate = 0.01
    epochs = 100

    w = np.array([0.0, 0.0, 0.0])
    bias = 0.0
```

```
    #----------------------
    n = len(x1)
    global mseList
    global wList
    mseList = []
    wList = []

    # Training
    for i in range(epochs):
        yhat = bias + np.dot(np.array([x1, x2, x3]).T, w)

        error = y - yhat
        squared_error = error**2
        mse = (1/n) * np.sum(squared_error)
        mseList.append(mse)
        wList.append(w)

        dw = (-2/n) * np.dot(np.array([x1, x2, x3]), error)
        dbias = (-2/n) * np.sum(error)

        w -= learning_rate * dw
        bias -= learning_rate * dbias

        if i % 10 == 0:
            print("Epoch:", i, "MSE:", mse)

    return bias, w, mseList, wList

bias, w, mse_list, w_list = train3(x1,x2,x3, y)
tracer_graphes([w[0] for w in w_list], mse_list, [i for i in range(100)])
```





3. Tester le modèle. pourquoi le modèle ne donne pas les résultats obtenus? il s'agit de quel problème? quels sont changement à faire pour éviter ce problème?

l'algorithme train2 et train3 sont conçu pour une regression linéaire simple c'est a dire pour une seule caractéristique , on pourra implementer le meme algorithme en utilisant la notation matriciel pour les 3 features avec la regression linéaire multiple

4. écrire une fonction train4 en reprenant le code de train3 de manière à utiliser matrices/vecteurs

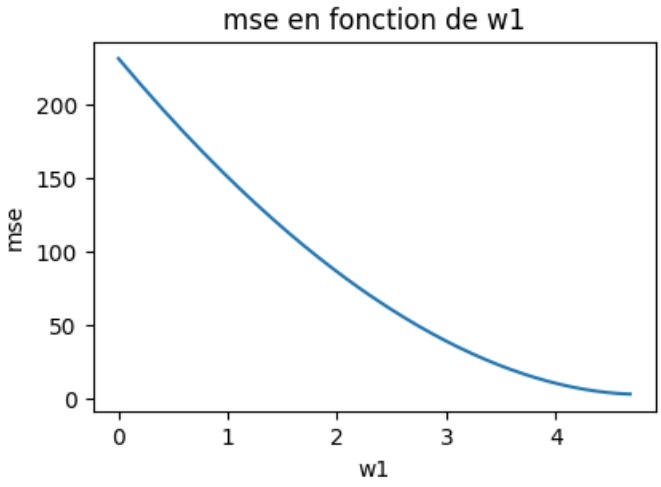
```
$$h(x)=X.\mathbf{W}+\mathbf{b}$$
```

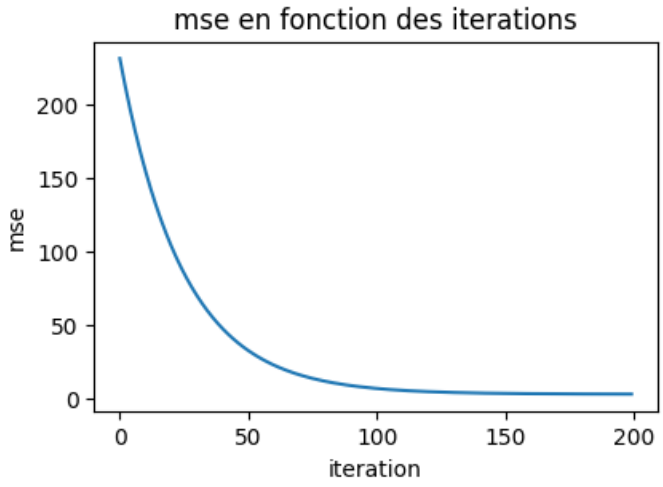
$$\mathbf{W} = \mathbf{W} - \alpha \cdot \frac{1}{m} \cdot \mathbf{X}^\top \cdot (\mathbf{X} \cdot \mathbf{W} + \mathbf{b} - \mathbf{y})$$

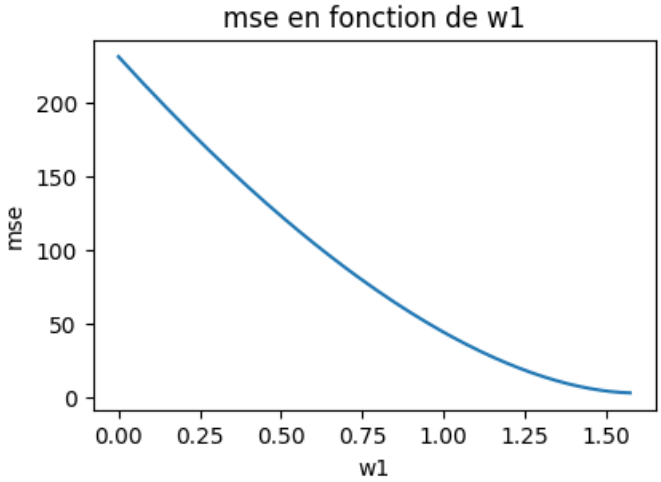
PYTHON

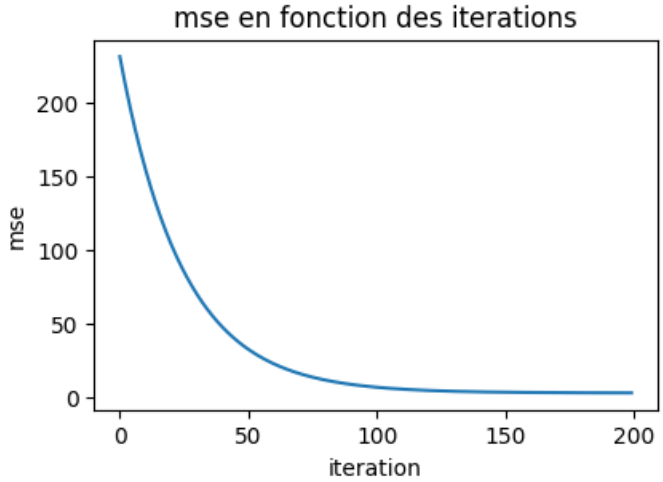
```python
import pandas as pd

mean = X.mean(axis=0)
std = X.std(axis=0)
X = (X - mean) / std


def train4(X, y, learning_rate=0.01, epochs=200):
    n = len(X)
    num_features = X.shape[1]
    w = np.zeros(num_features)
    bias = 1
    mse_list = []
    w_list = []

    for i in range(epochs):
        y_pred = bias + np.dot(X, w)

        error = y - y_pred
        squared_error = error ** 2
        mse = np.mean(squared_error)
        mse_list.append(mse)
        w_list.append(w.copy())

        grad_w = (-2 / n) * np.dot(X.T, error)
        grad_bias = (-2 / n) * np.sum(error)


        w -= learning_rate * grad_w
```

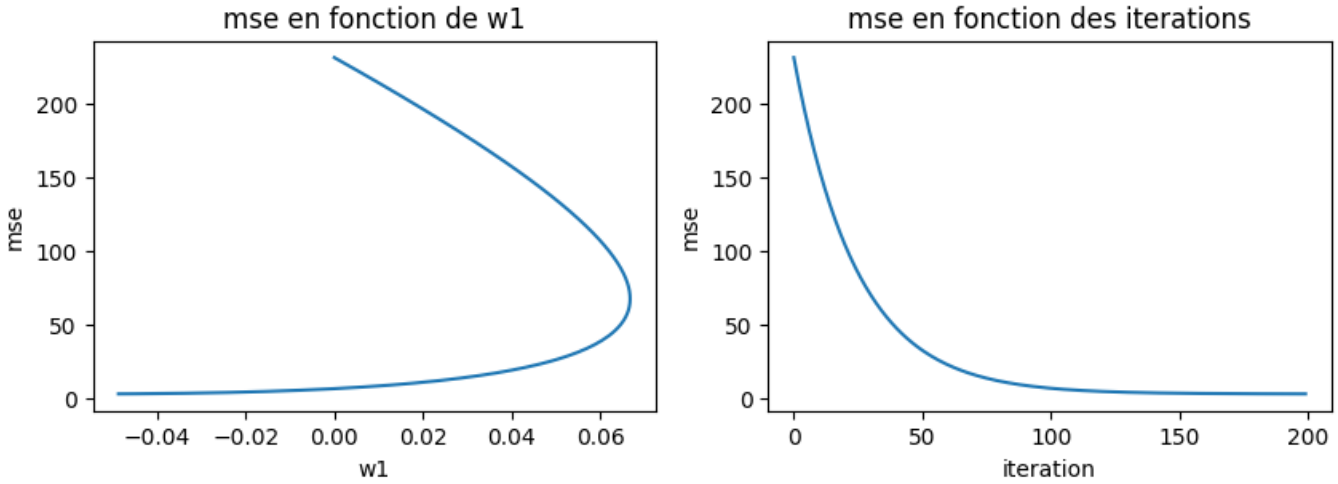
```
        bias -= learning_rate * grad_bias


    w_df = pd.DataFrame(w_list, columns=[f'w_{i}' for i in range(num_features)])

    return bias, w_df, mse_list

bias, w_df, mse_list = train4(X_train, y_train, learning_rate=0.01, epochs=200)



tracer_graphes(w_df['w_0'], mse_list, [i for i in range(200)])
tracer_graphes(w_df['w_1'], mse_list, [i for i in range(200)])
tracer_graphes(w_df['w_2'], mse_list, [i for i in range(200)])
```

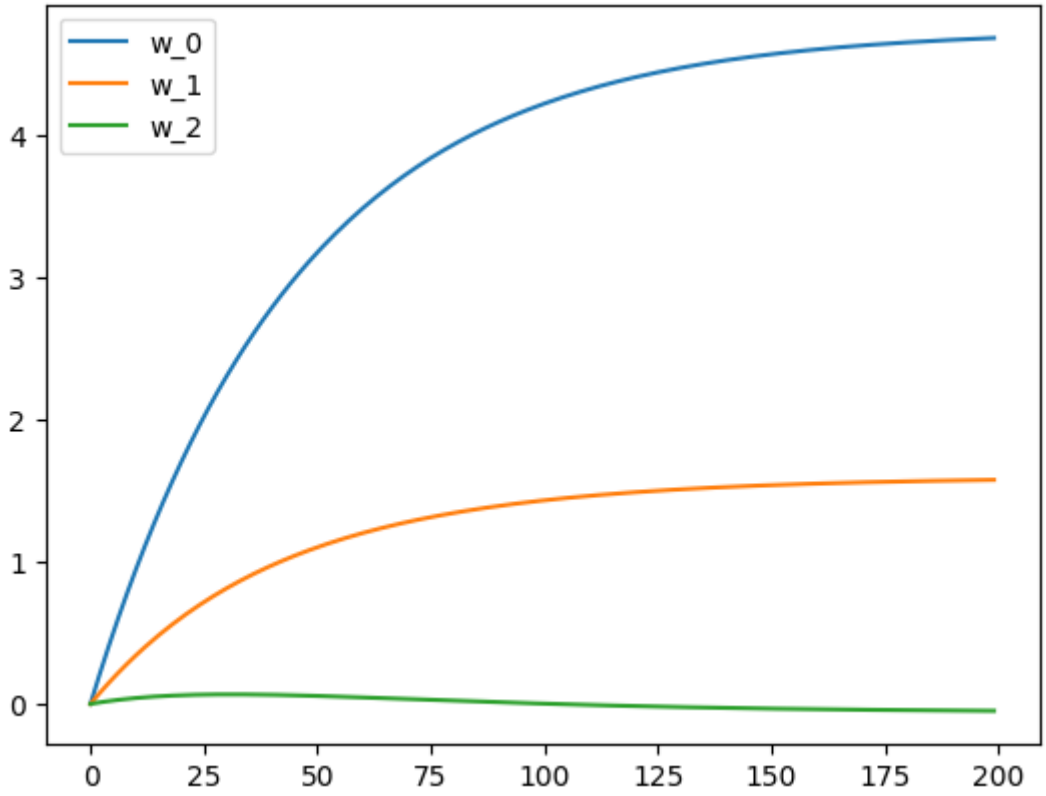












5) écrire une classe NN_One_Neurone avec les fonctions qu'il faut.

PYTHON

```python
class NN_One_Neurone:
    def __init__(self, learning_rate=0.01, epochs=100):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.bias = 0
        self.weights = np.zeros(3)

    def predict(self, X):
        return self.bias + np.dot(X, self.weights)
```

```
    def fit(self, X, y):
        n = len(X)
        for _ in range(self.epochs):
            yhat = self.predict(X)
            error = y - yhat
            mse = np.mean(error ** 2)

            grad_weights = (-2 / n) * np.dot(X.T, error)
            grad_bias = (-2 / n) * np.sum(error)

            self.weights -= self.learning_rate * grad_weights
            self.bias -= self.learning_rate * grad_bias

    def get_weights(self):
        return self.bias, self.weights
```

PYTHON

```
from IPython.display import display


model = NN_One_Neurone(learning_rate=0.01, epochs=100)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
bias, weights = model.get_weights()

display(bias)
display(weights)
```