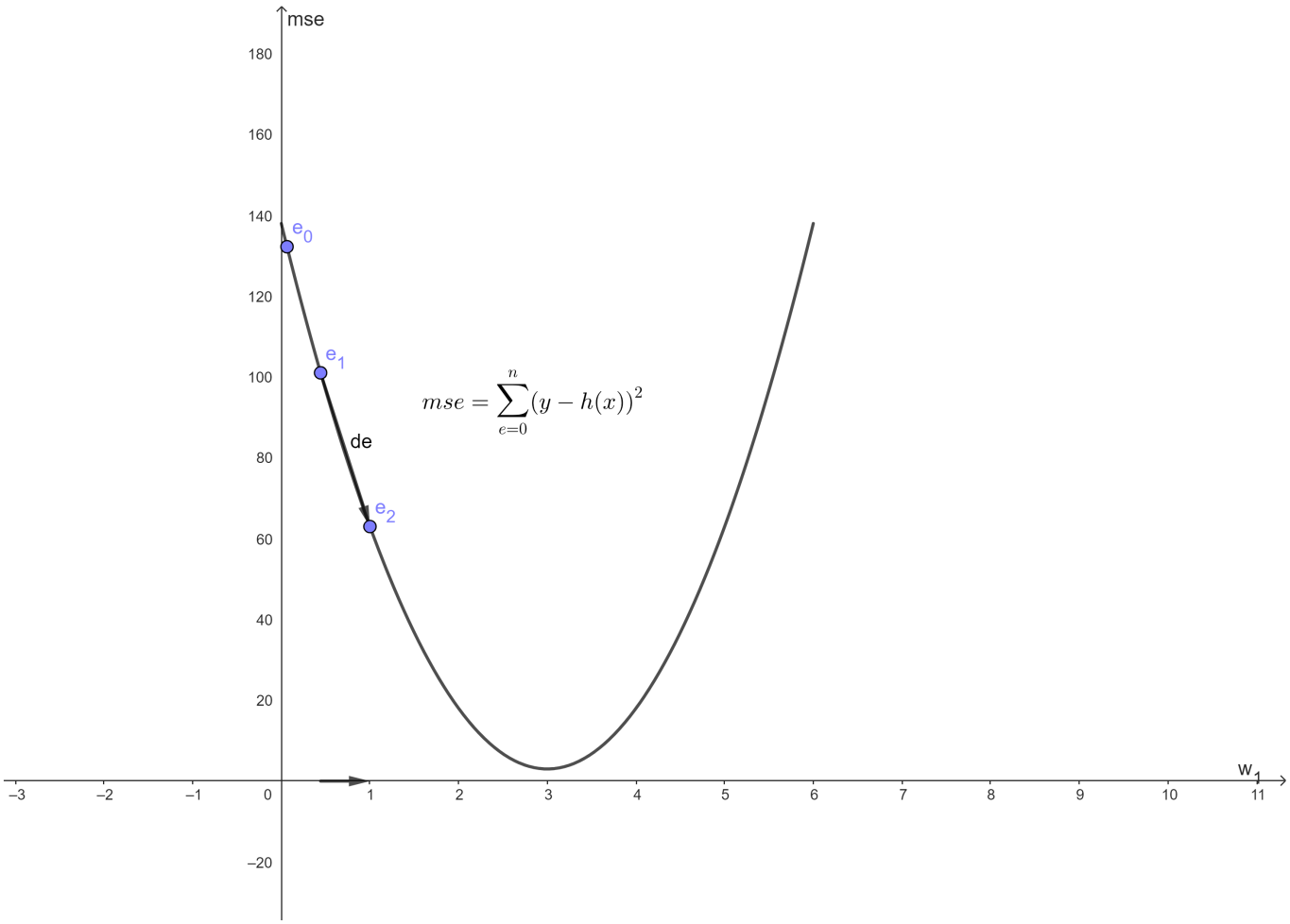
Nom : Ibrahim Lahlou

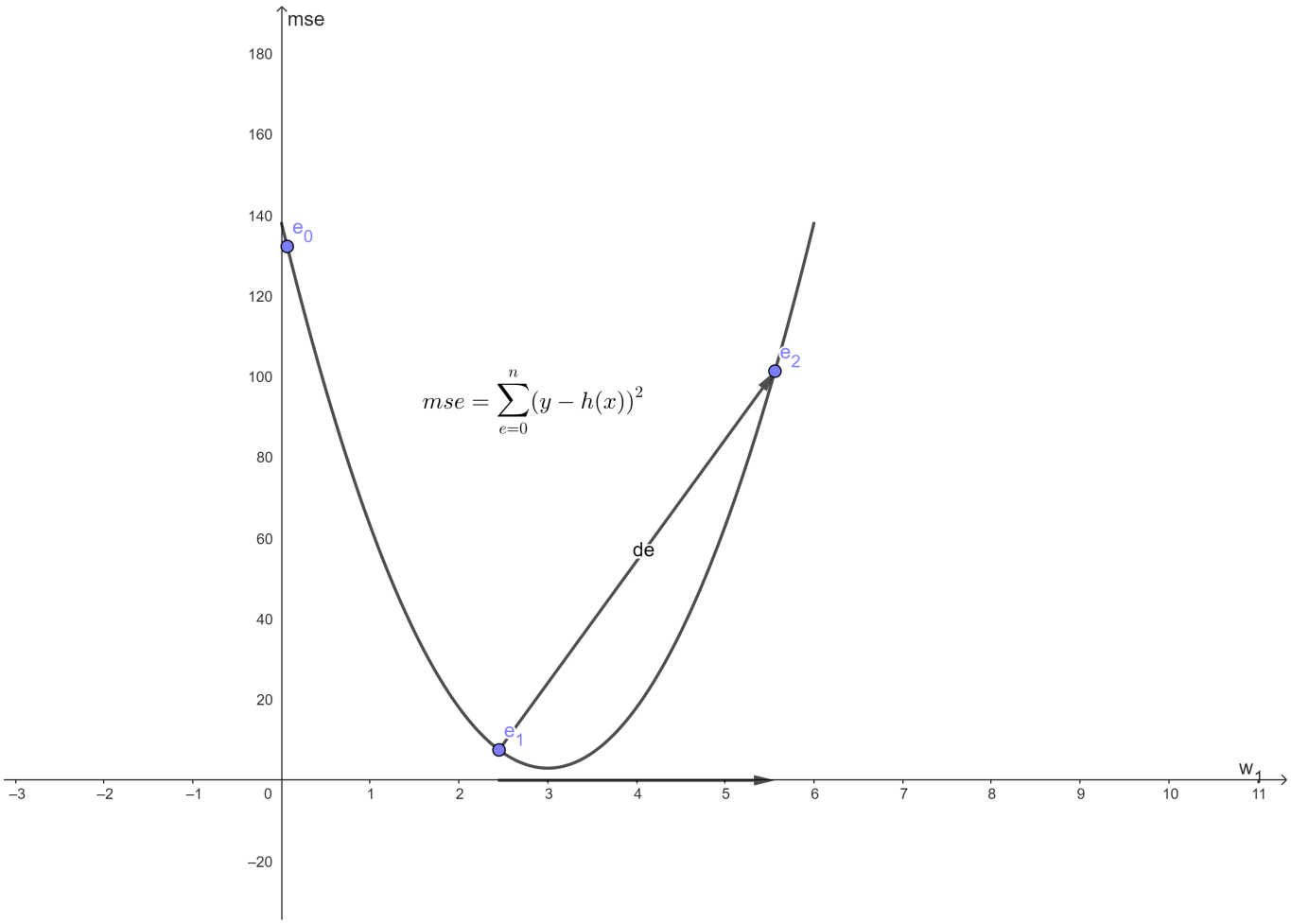
1. Comment varie mse à partir de l'itération 0 à l'iération 30?

En utilisant l'algorithme Gradient Descent , l'ajustement w_1 au début commence a diminuer progressivement le mse jusqu'a 30



2. Comment varie `mse` à partir de l'itération 30 à l'iération 59?

À partir de l'itération 30 , MSE peut commencer à fluctuer ou à augmenter car le pas d'apprentissage dw_1 est assez grands , et peut aussi entraîner une divergence à l'itération 59

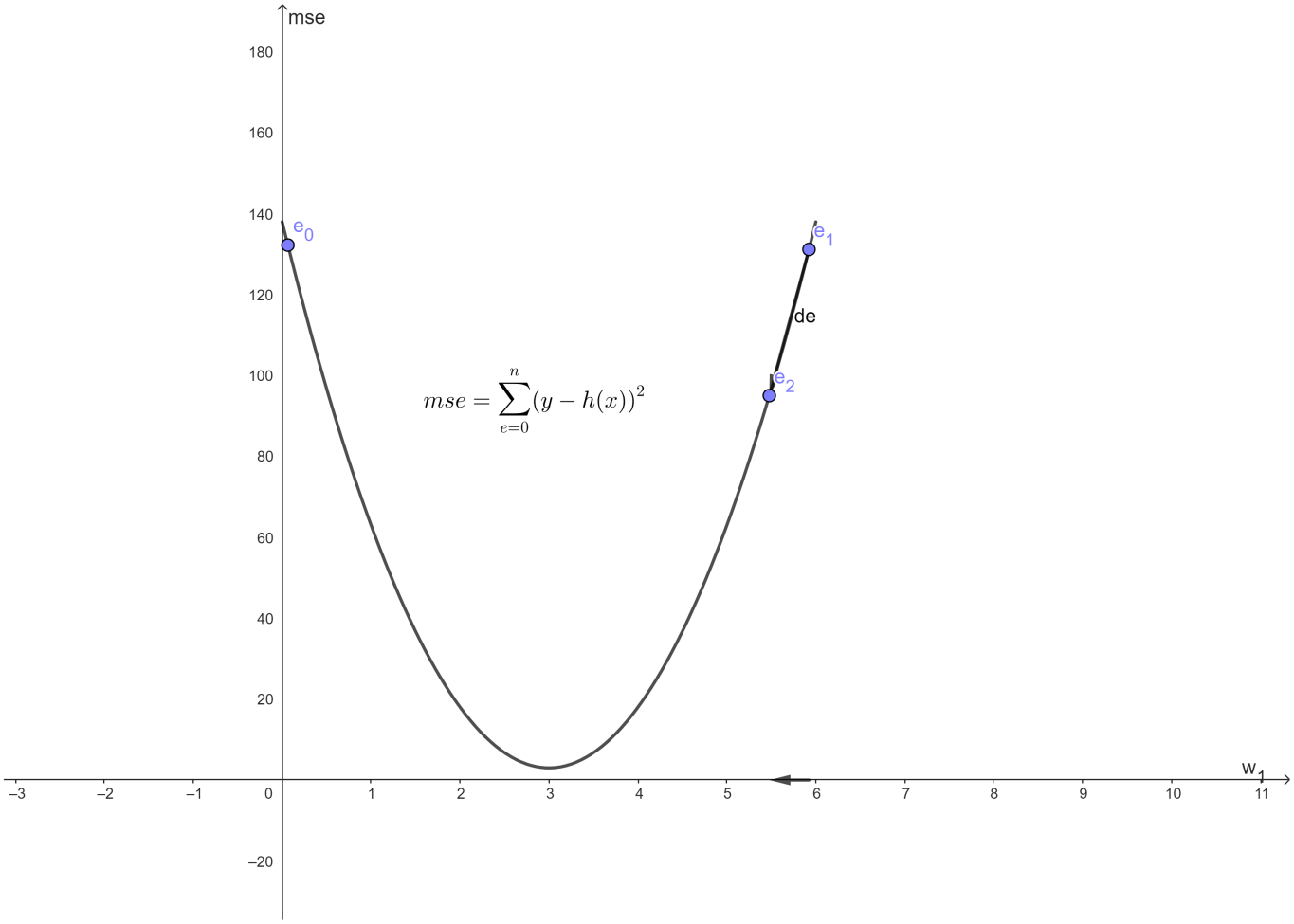


3. À quelle itération, `mse` est minimal?

À l'iteration 30 , On constate que `mse` est minimal

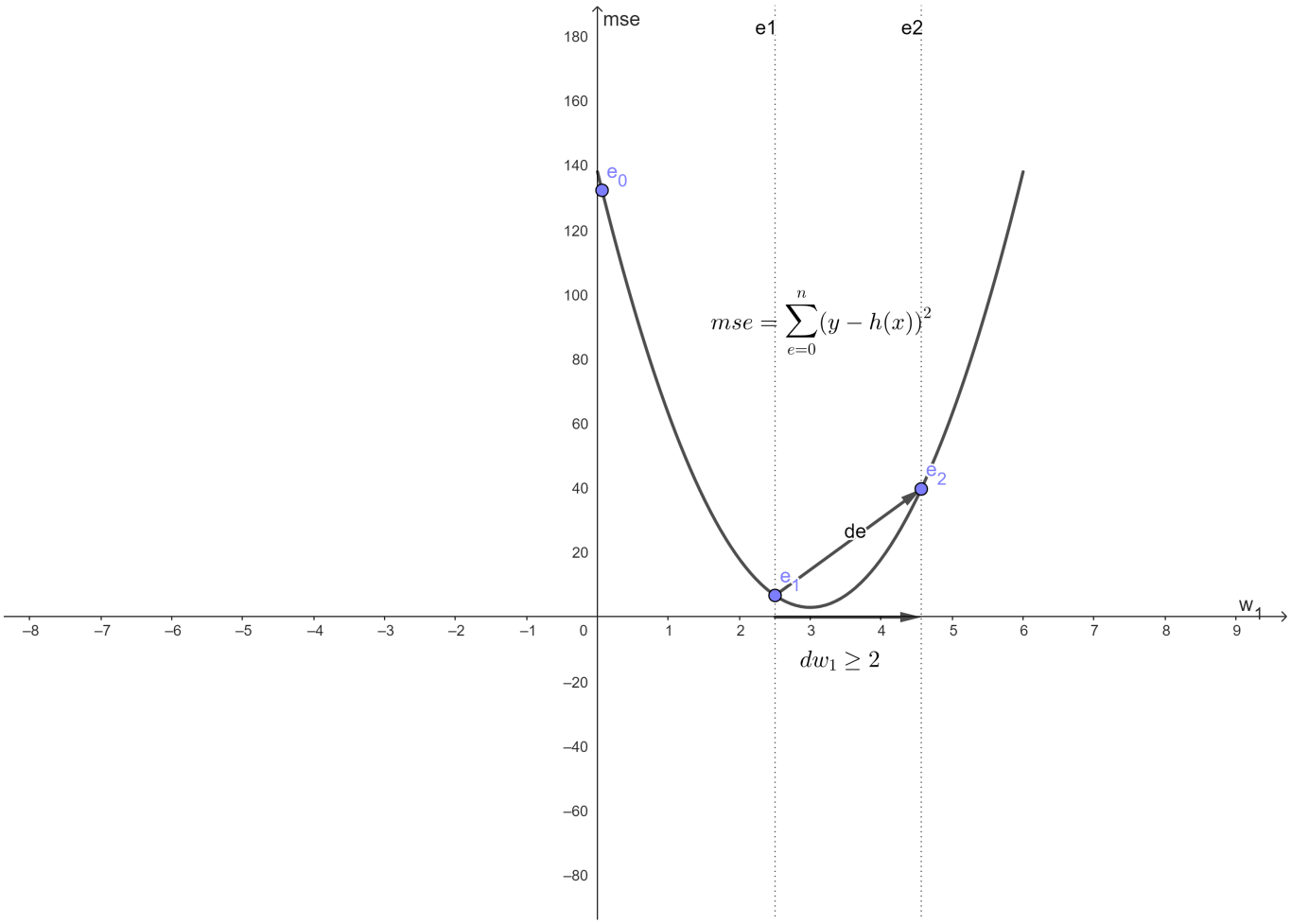
4. Comment se comporte l'algorithme si on initialise w1 à 6?

le modification de poids va être au sens inverse que si on initialisez à 0



5. Comment se comporte l'algorithme si $dw_1 = 2$? (ou autre valeur plus grande que 2

On constat que le pas change trés considérablement et v



7. Comment la dérivée partielle ($\frac{dmse}{dw_1}$) permet de régler le problème en 4)? (Algorithme du gradient)

il modifiera w_1 ça en inversant le signe vu que l'ordre est changé entre y_i et $\hat{y}_i$

$$w_1^{e+1} = w_1^e - \alpha \frac{dmse}{dw_1}$$

$$w_1^{e+1} = w_1^e - \alpha \frac{1}{n} \sum_{i=1}^{n} 2(y_i - \hat{y}_i) x_{i1}$$

8. Comment le learning rate permet de régler le problème en 5

Le learning rate va ajouté une précision au modification que l'apporte le gradient descent pour ne pas avoir des changement brusque ou minim a w_1

9. Justifier la valeur de `mse` trouvé?

Une `mse` plus basse indique généralement un meilleur ajustement du modèle aux données , c'est pour cela qu'on constate que 3 est le meilleur poids a mettre pour un modèle optimal

Cependant, il est essentiel de s'assurer que le modèle généralise bien aux données non vues (c'est-à-dire, des données de test) pour éviter le surajustement.

10. Modifier l'algorithme de manière à assurer de trouver les bons paramètres

Pour garantir que le modèle

11. Compléter le code source de la classe NN_One_Neurone

```python
class NN_One_Neurone:
    def __init__(self, learning_rate=0.01, epochs=1000):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.bias = 0
        self.w1 = 0

    def activation_function(self, x):
        return self.bias + self.w1 * x

    def fit(self, X, y):
        n = len(X)
        for _ in range(self.epochs):
            yhat = self.activation_function(X)
            error = y - yhat
            mse = (1/n) * np.sum(error**2)
            dw1 = (-2/n) * np.sum(X * error)
            dbias = (-2/n) * np.sum(error)

            self.w1 -= self.learning_rate * dw1
            self.bias -= self.learning_rate * dbias

    def predict(self, X):
        return self.activation_function(X)
```

12. Tester les fonctions de la classes

```python
model = NN_One_Neurone(learning_rate=0.01, epochs=60)

model.fit(X, y)

model.predict(5)
```