Compte rendu TP4

ROUBIA Akram

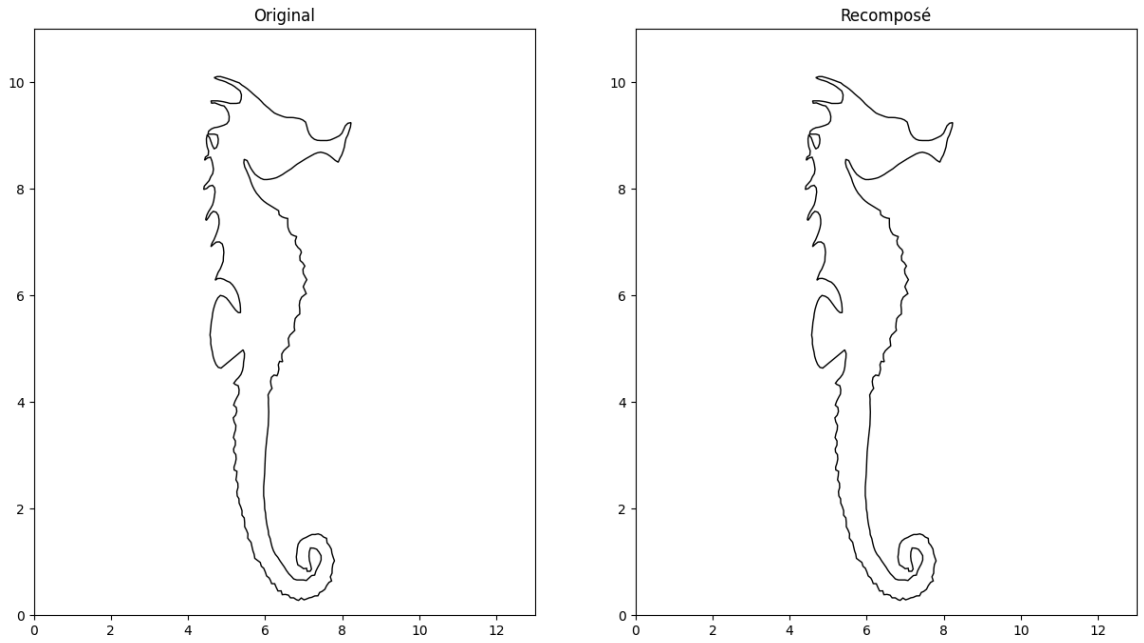
Décomposition / Recomposition de Chaikin :

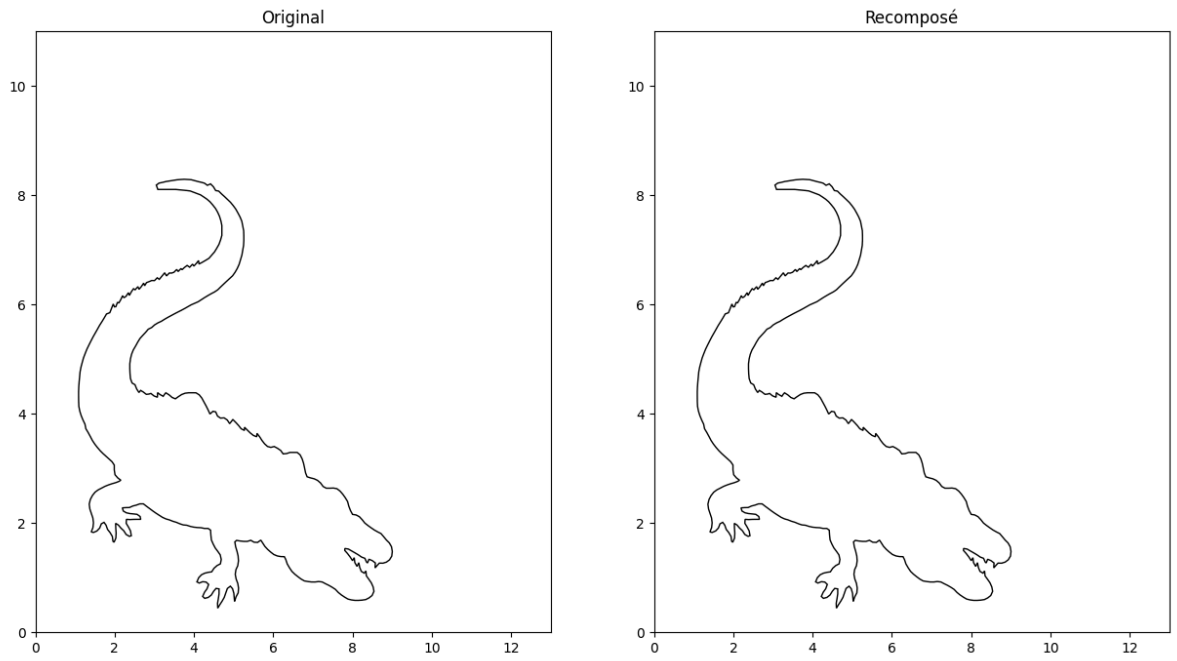
```
def decomposition_chaikin(points):
    """
    Décompose un ensemble de points en moyennes et détails selon
Chaikin.

    Formules:
    x_i^n = 1/4 * (-x_{2i-2}^{n+1} + 3*x_{2i-1}^{n+1} + 3*x_{2i}^{n+1} -
x_{2i+1}^{n+1})
    y_i^n = 1/4 * (x_{2i-2}^{n+1} - 3*x_{2i-1}^{n+1} + 3*x_{2i}^{n+1} -
x_{2i+1}^{n+1})

    Returns: (moyennes, details) où moyennes sont les points décomposés
et details les détails associés après une étape de décomposition
    """
    n = len(points)
    n_moy = n // 2

    moyennes = np.zeros((n_moy, 2))
    details = np.zeros((n_moy, 2))

    for i in range(n_moy):
        # Indices avec gestion circulaire
        i1 = (2*i - 2) % n
        i2 = (2*i - 1) % n
        i3 = (2*i) % n
        i4 = (2*i + 1) % n

        # Calcul des moyennes (composante X et Y)
        moyennes[i] = 0.25 * (-points[i1] + 3*points[i2] + 3*points[i3]
- points[i4])

        # Calcul des détails
        details[i] = 0.25 * (points[i1] - 3*points[i2] + 3*points[i3] -
points[i4])

    return moyennes, details
```

```python
def recomposition_chaikin(moyennes, details):
    """
    Recompose un ensemble de points à partir des moyennes et détails
selon Chaikin.

    Formules:
    x_{2i}^{n+1} = x_i^n - y_i^n
    x_{2i+1}^{n+1} = x_i^n + y_i^n

    Returns: points reconstitués après une étape de recomposition
    """
    if len(moyennes) != len(details):
        raise ValueError("Les tailles des moyennes et des détails
doivent être identiques.")

    n_moy = len(moyennes)
    n = n_moy * 2

    points = np.zeros((n, 2))

    for i in range(n_moy):
        i_next = (i + 1) % n_moy
        points[2*i] = 0.25 * ((3*(moyennes[i] + details[i])) +
(moyennes[i_next] - details[i_next]))
        points[2*i + 1] = 0.25 * ((moyennes[i] + details[i]) +
3*(moyennes[i_next] - details[i_next]))

    return points

def decomposition_totale(points):
    """
    Effectue une décomposition complète en moyennes et détails jusqu'à
la plus petite résolution.
    Returns: (moyenne, détails) où moyenne est un point et détails est
une liste de tableaux de détails à chaque niveau.
    """
    moyenne = points[:]
    details = []
    while len(moyenne) > 2:
        moy, det = decomposition_chaikin(moyenne)
        details.append(det)
        moyenne = moy
    return moyenne, details
```

```python
def recomposition_totale(moyenne, details):
    """
    Recompose les points à partir de la décomposition complète en
moyennes et détails.
    Returns: points reconstitués
    """
    n = len(moyenne)
    if n != 2:
        raise ValueError("La moyenne doit être de taille 2 pour la
recomposition totale.")
    current_points = moyenne
    for det in reversed(details):
        current_points = recomposition_chaikin(current_points, det)
    return current_points
```

On utilise decomposition_chaikin() et recomposition_chaikin() (qui font une étape de la décomposition / recomposition respectivement) dans decomposition_totale() / recomposition_totale(). Quand on teste cela sur croco.d et hyppo.d on remarque que la décomposition/recomposition est identique au dessin d'origine

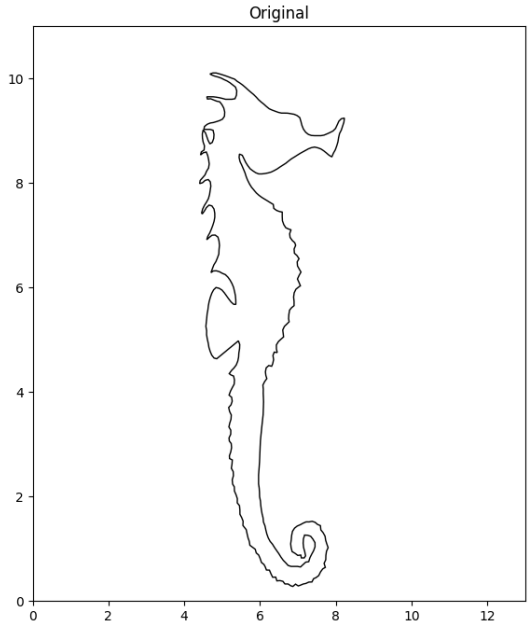


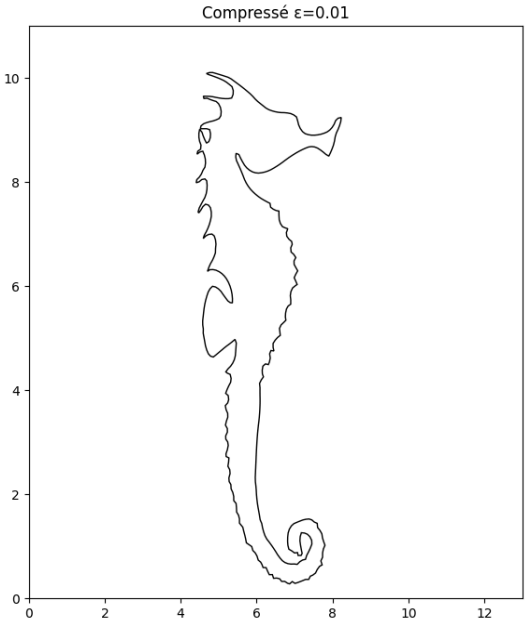


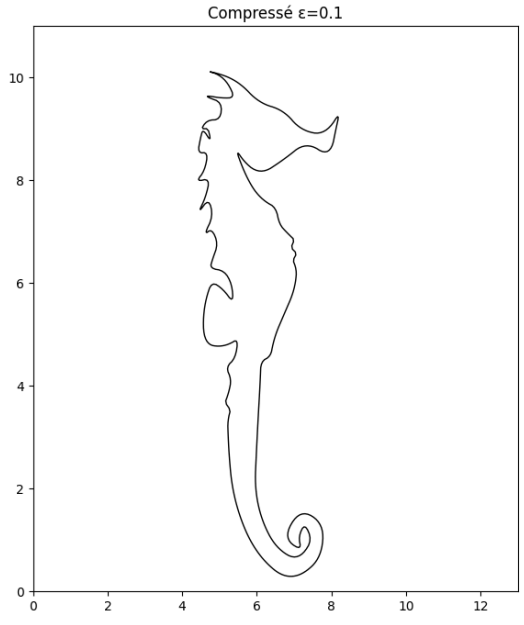
Multi-résolution :

```
def compression_et_recomposition(moyenne, details, epsilon):
    """
    Realise une reconstitution compressée en mettant à zéro les détails de
norme inférieure à epsilon.
    Returns: points reconstitués après compression
    """
    for detail in details:
        for elem in detail:
            if np.linalg.norm(elem) < epsilon:
                elem[:] = 0.0
    return recomposition_totale(moyenne, details)
```

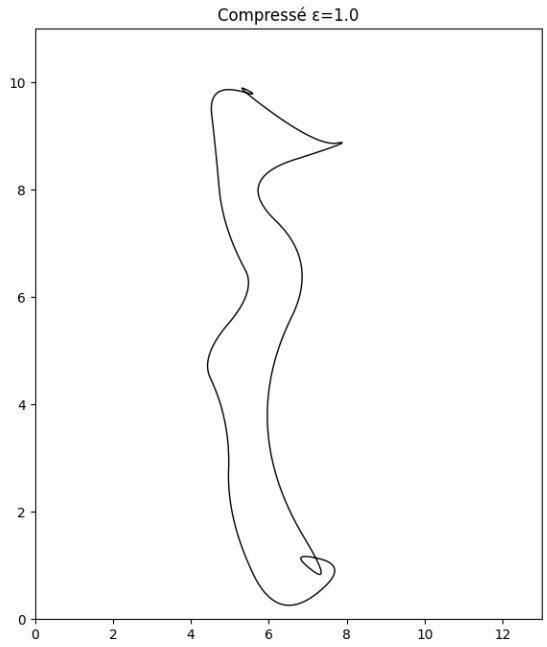
Vu que la compression implique de mettre à zéro les détails inférieurs au seuil, on boucle sur les details pour enlever tous les éléments qui respectent cette condition, et puis on effectue la recomposition avec la liste modifiée.
Cela nous donne les résultats suivants :

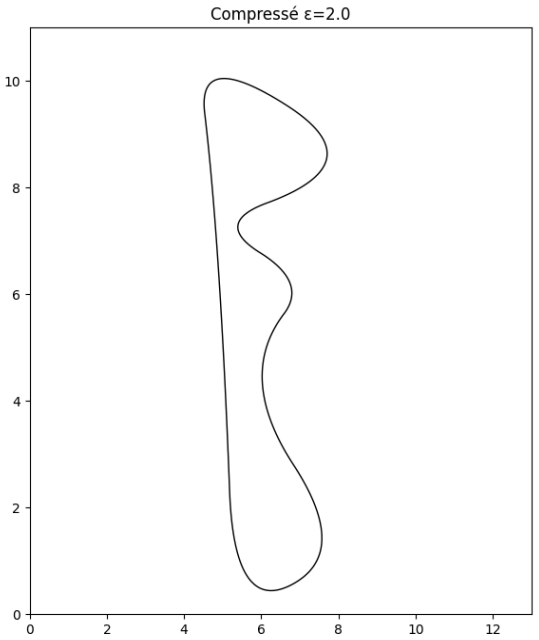

Original
0
2
4
6
8
10
0
2
4
6
8
10
12


Compressé ε=0.01
0
2
4
6
8
10
0
2
4
6
8
10
12


Compressé ε=0.1
0
2
4
6
8
10
0
2
4
6
8
10
12


Compressé ε=0.5
0
2
4
6
8
10
0
2
4
6
8
10
12



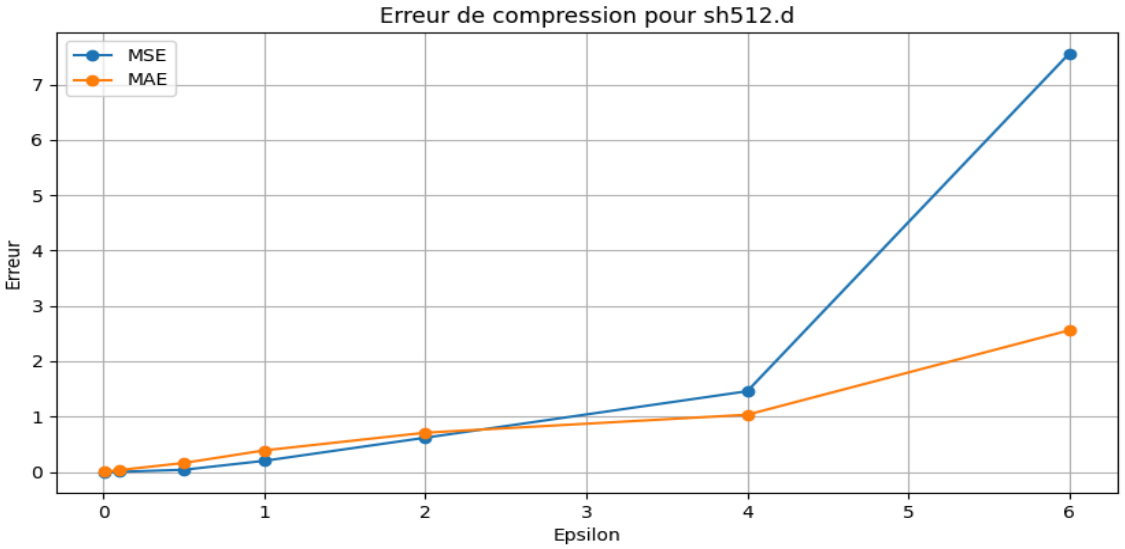


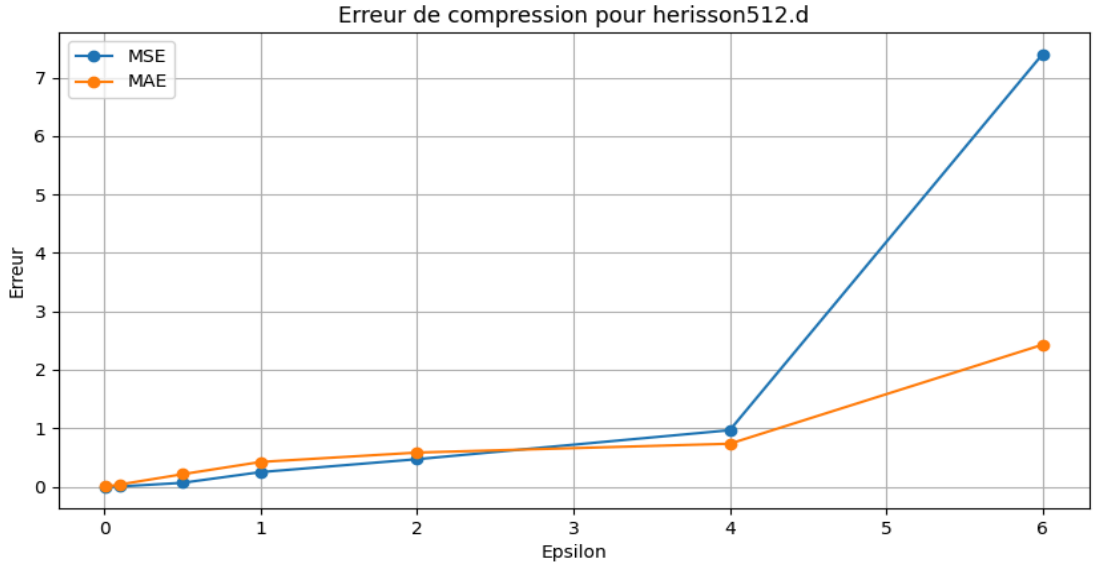
On remarque la même chose pour les autres dessins, ce qui est normal : Plus notre epsilon est élevé, plus nous perdons en détails et plus le dessin est “abstrait”.

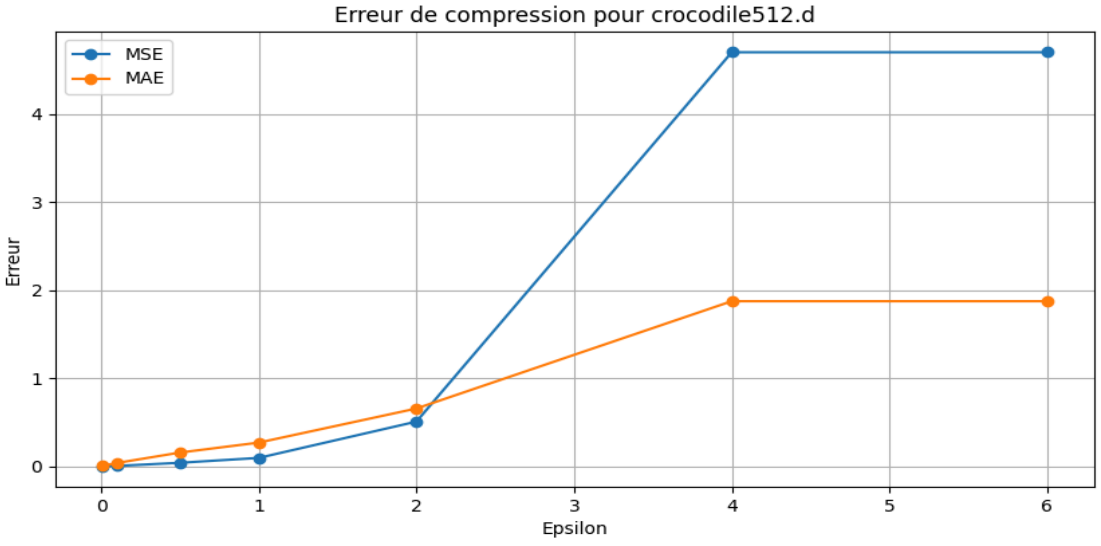
Pour calculer l’erreur en fonction du seuil epsilon, nous utilisons l’erreur absolue et l’erreur quadratique :

```
def erreur_compression(original):
    """
    Calcule les erreurs (quadratique et absolue) entre la courbe
reconstruite et la courbe compressée.
    Returns: erreur quadratique moyenne et erreur absolue moyenne
    """
    mse_list = []
    mae_list = []
    for eps in [0.01, 0.1, 0.5, 1.0, 2.0, 4.0, 6.0]:
        moyenne, details = decomposition_totale(original)
        recomposed = recomposition_totale(moyenne, details)
        compressed = compression_et_recomposition(moyenne, details, eps)
        recomposed = np.array(recomposed)  # Convert to numpy array for
vectorized operations
        compressed = np.array(compressed)
        mse = np.mean(np.linalg.norm(recomposed - compressed, axis=1) ** 2)
        mae = np.mean(np.linalg.norm(recomposed - compressed, axis=1))
        mse_list.append(mse)
        mae_list.append(mae)
    return mse_list, mae_list
```

Cela nous donne les graphes suivants :



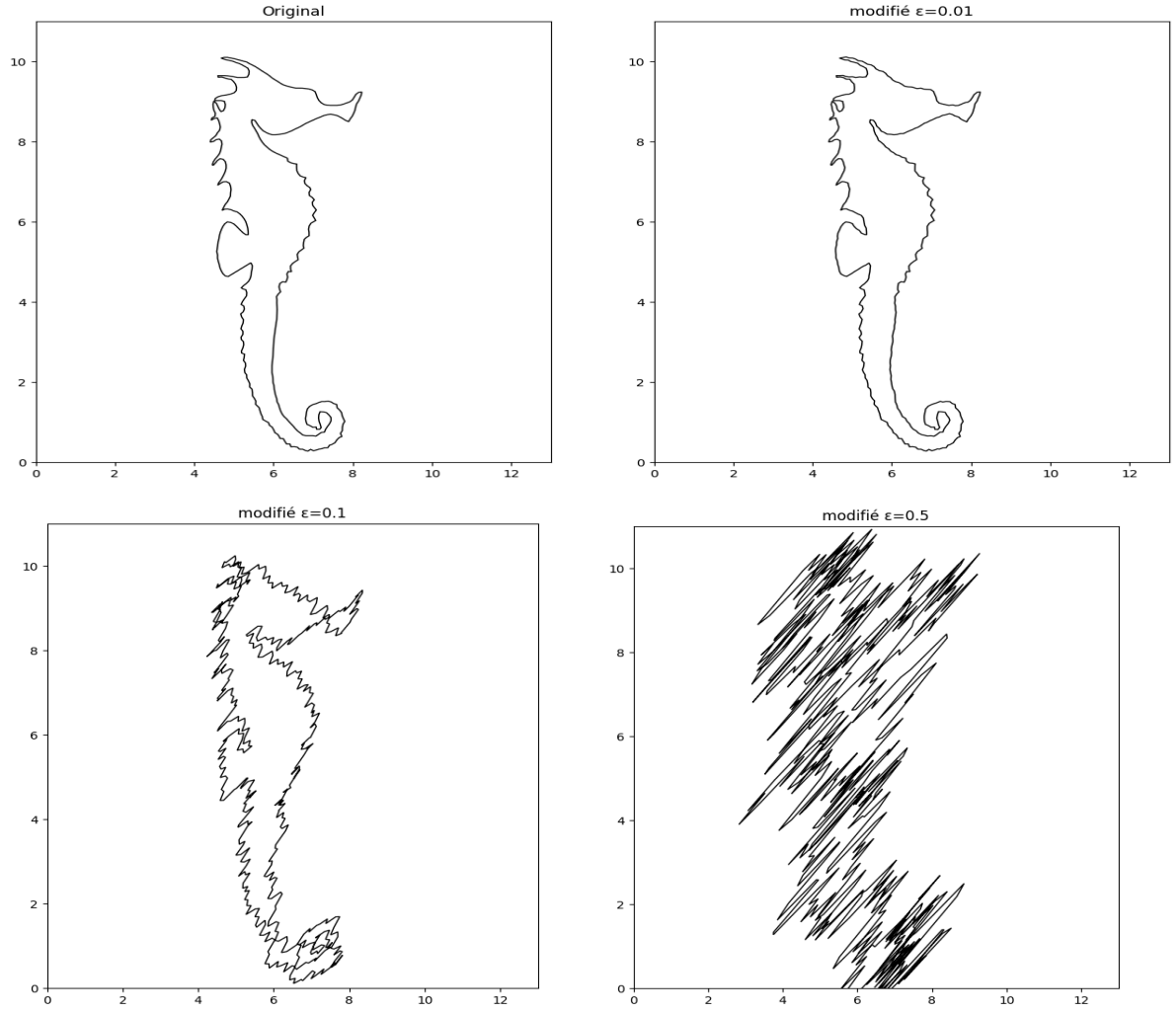




Finalement, nous avons déplacé quelques points en basse résolution en modifiant légèrement les détails en basse résolution :

```
def compression_modification_basse_resolution(moyenne, details, epsilon):
    """
    Modifie la basse résolution en modifiant légèrement les détails de norme
inférieure à epsilon.
    Returns: points reconstitués après modification de la basse résolution
    """
    for detail in details:
        for elem in detail:
            if np.linalg.norm(elem) < epsilon:
                if random.randint(1,10) <= 5:
                    elem += epsilon
                else:
                    elem -= epsilon
    return recomposition_totale(moyenne, details)
```

Cela nous donne les dessins suivants pour l'hippocampe :



Plus on modifie les détails , plus les dessins sont erronés (logique vu qu'on utilise les détails pour monter en résolution)