Vendredi 19 novembre 2021

Compte-rendu 3 : Méthodes statistiques pour la classification dans les chaînes de Markov cachées

Alexandre Chaussard

Antoine Klein

Introduction : L'objectif de cette dernière partie est de rajouter de la dépendance entre les variables de sorte à diminuer l'erreur moyenne de segmentation tout en maintenant une durée de calculs acceptable. Pour ce faire, on se place dans un modèle de chaînes de Markov cachées. Chaque variable X_i ne dépend que de celle qui la précède, et conditionnellement à une classe X_i, les Y_i sont indépendants. Pour implémenter la segmentation, nous utilisons l'algorithme du forward-backward. Avec l'ajout d'une dépendance entre les variables, on s'attend à un taux d'erreur plus faible qu'avec le critère de totale indépendance.

I) Segmentation d'un signal par les chaînes de Markov cachées

Ayant un signal à deux classes, nous plaçons les valeurs successives d'alpha et de beta dans une matrice n*2 où la première colonne donne la valeur pour la première classe, la deuxième colonne pour la deuxième classe.

Voici un exemple de la segmentation d'un signal avec une chaîne de Markov cachée :

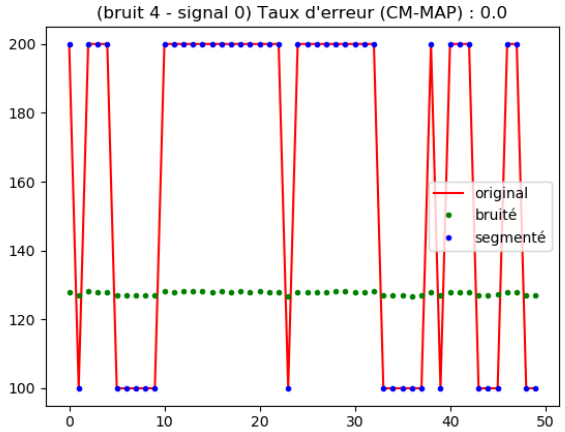


Figure 1 : Segmentation du signal 0 avec le bruit 4 par la méthode des chaînes de Markov cachée avec un critère MAP

Pour un signal quelconque, nous devons estimer les probabilités de transition, au nombre de 4, définie simplement à partir d'une probabilité par ligne car la matrice de transition est de taille 2*2. Il nous faut aussi déterminer la graine, c'est-à-dire la distribution de la première distribution. Nous la fixons uniforme (probabilité d'apparition de la première classe égale à 0.5).

Voici nos résultats de segmentation de 5 chaînes de Markov par le critère MAP et par une chaîne de Markov cachée générées aléatoirement selon une matrice de transition A définie par ses coefficients A[0][0] et A[1][1]:

	Bruit 1	Bruit 2	Bruit 3	Bruit 4	Bruit 5
A[0][0] = 0.1 **A[1][1] = 0.9**	CM : 0% MAP : 0%	CM : 13% MAP : 20%	CM : 25% MAP : 30%	CM : 0% MAP : 0%	CM : 32% MAP : 39%
A[0][0] = 0.2 **A[1][1] = 0.8**	CM : 0% MAP : 0%	CM : 17% MAP : 21%	CM : 26% MAP : 27%	CM : 0% MAP : 0%	CM : 31% MAP : 35%
A[0][0] = 0.3 **A[1][1] = 0.7**	CM : 0% MAP : 0%	CM : 20% MAP : 20%	CM : 26% MAP : 26%	CM : 0% MAP : 0%	CM : 33% MAP : 30%
A[0][0] = 0.4 **A[1][1] = 0.6**	CM : 0% MAP : 0%	CM : 21% MAP : 20%	CM : 23% MAP : 24%	CM : 0% MAP : 0%	CM : 26% MAP : 30%
A[0][0] = 0.5 **A[1][1] = 0.5**	CM : 0% MAP : 0%	CM : 24% MAP : 19%	CM : 24% MAP : 22%	CM : 0% MAP : 0%	CM : 28% MAP : 25%

Figure 2 : Tableau récapitulatif des erreurs moyennes pour chaque méthode de segmentation, pour chaque signal et chaque bruit

On constate alors une nette supériorité de la qualité de la segmentation par les chaînes de Markov par rapport au MAP pour des probabilités de transition A[0][0] et A[1][1] fortement distinctes.

A l'inverse lorsque celles-ci tendent à s'égaliser, les résultats sont de moins en moins probants et les deux modèles semblent comparables. Voire, certains cas font apparaître une moins bonne estimation par les chaînes de Markov que par le MAP, peut-être car l'on tend vers une loi uniforme pour chaque transition et que la modélisation de dépendance entre X_i et X_{i-1} devient de moins en moins correcte de fait.

Par ailleurs, pour les 6 signaux fournis en annexe, nous obtenons les segmentations suivantes :

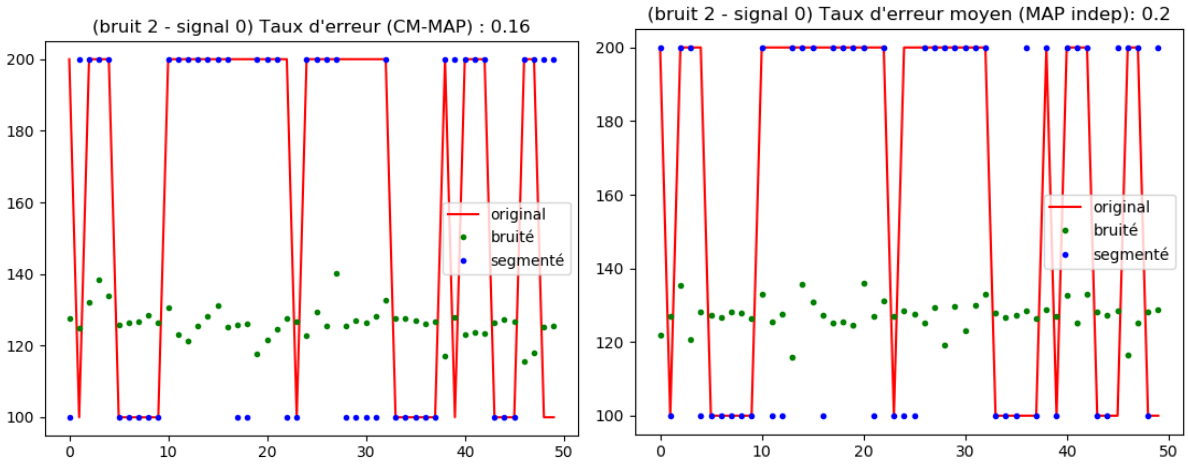
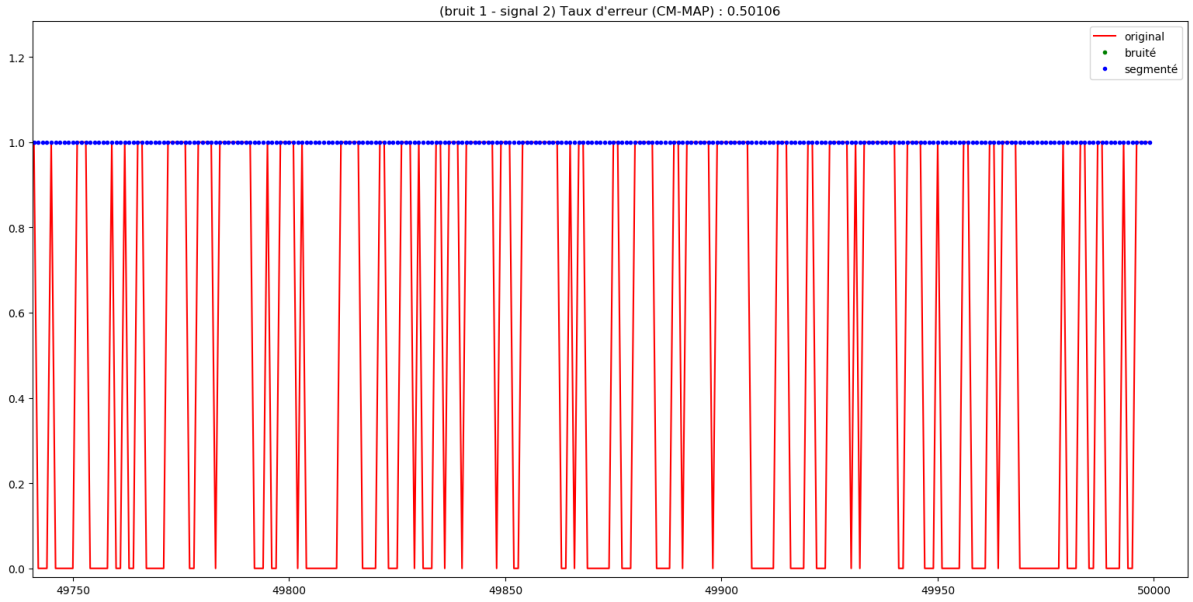


Figure 3 : Segmentation du signal 0 avec le bruit 2 par la méthode des chaînes de Markov cachée avec un critère MAP (gauche) et par la méthode MAP (droite)



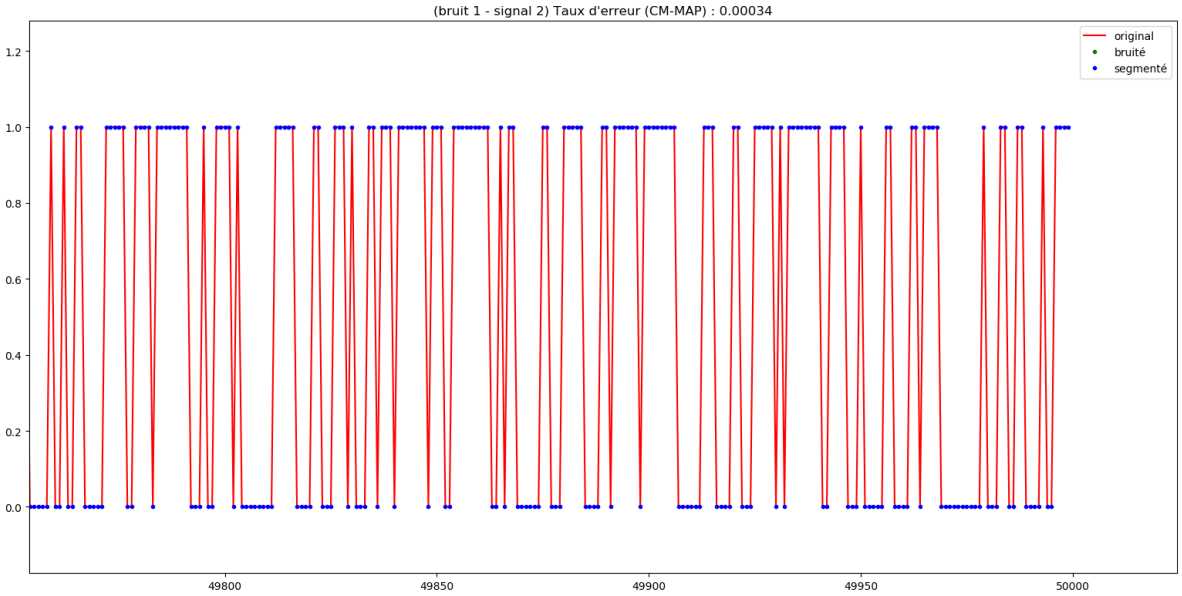
<u>Figure 4</u> : Segmentation du signal 2 avec le bruit 1 par la méthode des chaînes de Markov cachée avec un critère MAP sans rescaling

A cette étape de la programmation, alpha et beta sont obtenus comme produit de facteurs strictement plus petits que 1, et aucune étape de rescaling n'est mise en place. A mesure que la chaîne de Markov s'allonge, ils tendent vers 0. Si théoriquement cela ne pose aucun problème, lorsque l'on passe à un calcul numérique, le produit alpha*beta est arrondi à 0 ; rendant impossible toute segmentation.

C'est ce que l'on observe sur la figure 4 : la classe de sortie est constante car alpha et beta sont nuls, ce qui donne une sortie constante.

Pour résoudre ce problème numérique, nous procédons à un rescaling ; c'est-à-dire que nous multiplions chaque alpha par un terme de même ordre de grandeur que alpha lui-même de sorte que le terme ne soit ni trop loin de 1, ni trop prêt. En quelque sorte, nous divisons alpha par son ordre de grandeur.

Pour le cas de la figure 4, on obtient désormais cette segmentation, beaucoup plus adaptée :



<u>Figure 5</u> : Segmentation du signal 2 avec le bruit 1 par la méthode des chaînes de Markov cachée avec un critère MAP avec rescaling

Pour l'ensemble des signaux, on obtient alors les résultats suivants récapitulés dans ce tableau :

	Bruit 1	**Bruit 2**	**Bruit 3**	**Bruit 4**	**Bruit 5**
Signal 0	CM : 0% MAP : 0%	CM : 22% MAP : 28%	CM : 24% MAP : 26%	CM : 0% MAP : 0%	CM : 46% MAP : 36%
Signal 1	CM : 0% MAP : 0%	CM : 18% MAP : 21%	CM : 28% MAP : 36%	CM : 0% MAP : 0%	CM : 37% MAP : 38%
Signal 2	CM : 0% MAP : 0%	CM : 16% MAP : 18%	CM : 28% MAP : 31%	CM : 0% MAP : 0%	CM : 36% MAP : 38%
Signal 3	CM : 0% MAP : 0%	CM : 16% MAP : 17%	CM : 29% MAP : 31%	CM : 0% MAP : 0%	CM : 36% MAP : 38%
Signal 4	CM : 0% MAP : 0%	CM : 17% MAP : 18%	CM : 31% MAP : 31%	CM : 0% MAP : 0%	CM : 38% MAP : 38%
Signal 5	CM : 0% MAP : 0%	CM : 17% MAP : 18%	CM : 31% MAP : 31%	CM : 0% MAP : 0%	CM : 38% MAP : 38%

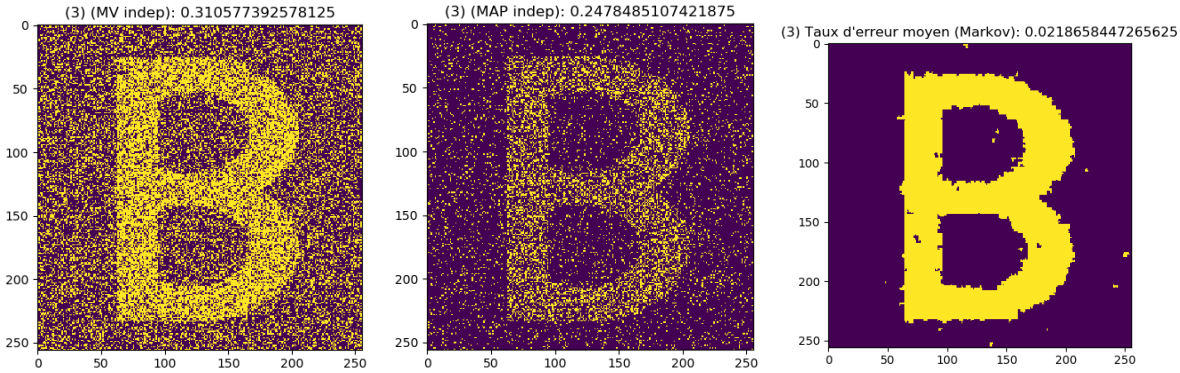
<u>Figure 6</u> : Tableau récapitulatif des segmentations des signaux donnés en annexe sous les cinq bruits par la méthode des chaînes de Markov cachée avec un critère MAP avec rescaling et une segmentation par MAP

On note alors des résultats globalement meilleurs avec la segmentation par CMC + MAP par rapport à une MAP simple, sauf dans certains cas où les performances sont identiques voire légèrement moins bonne. Ce dernier cas peut être lié à un problème de vectorisation du code qui nous a imposé de réduire drastiquement notre calcul d'erreur moyenne pour obtenir des résultats en un temps raisonnable.

II) Application de notre modèle à la segmentation d'image

Une image étant en 2D nous devons la convertir en une chaîne de Markov 1D. Pour cela, nous utilisons le parcours de Hilbert-Peano qui est adapté aux images ayant des contrastes rectangulaires. On importe donc la librairie fournie.

Voici les résultats de segmentation sur les images fournies (autres segmentations fournies en annexe) :



<u>Figure 7</u> : Segmentation de l'image Beee2 par les méthodes de Maximum de Vraisemblance (MV indep), MAP (MAP indep), puis chaîne de Markov cachées (Markov)

Voici un tableau récapitulatif pour l'ensemble des images segmentées :

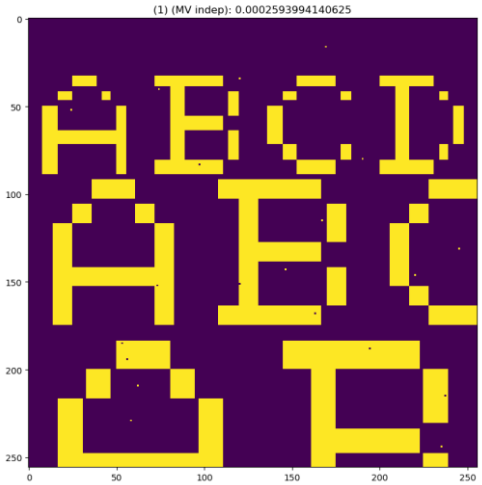
	Bruit 1	**Bruit 2**	**Bruit 3**	**Bruit 4**	**Bruit 5**
Alfa2	CM : 0% MAP : 0% MV : 0%	CM : 2% MAP : 10% MV : 12%	CM : 5% MAP : 22% MV : 31%	CM : 0% MAP : 0% MV : 0%	CM : 7% MAP : 22% MV : 29%
Beee2	CM : 0% MAP : 0% MV : 0%	CM : 1% MAP : 12% MV : 13%	CM : 2% MAP : 24% MV : 31%	CM : 0% MAP : 0% MV : 0%	CM : 3% MAP : 26% MV : 31%
Cible2	CM : 0% MAP : 0% MV : 0%	CM : 2% MAP : 12% MV : 13%	CM : 4% MAP : 26% MV : 31%	CM : 0% MAP : 0% MV : 0%	CM : 7% MAP : 27% MV : 32%
City2	CM : 0% MAP : 0% MV : 0%	CM : 9% MAP : 15% MV : 16%	CM : 20% MAP : 30% MV : 31%	CM : 0% MAP : 0% MV : 0%	CM : 25% MAP : 34% MV : 35%
Country2	CM : 0% MAP : 0% MV : 0%	CM : 4% MAP : 14% MV : 14%	CM : 9% MAP : 28% MV : 31%	CM : 0% MAP : 0% MV : 0%	CM : 12% MAP : 30% MV : 33%

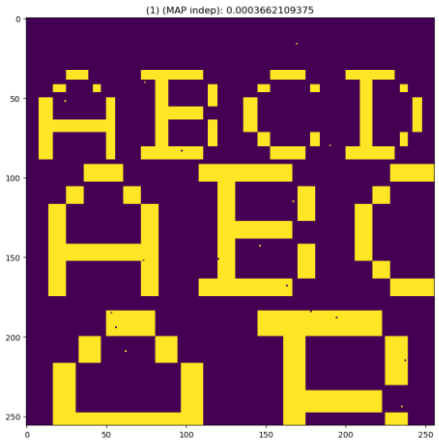
<u>Figure 8</u> : Tableau des taux d'erreurs moyens sur chaque image vis-à-vis de chaque méthode et pour chaque bruit

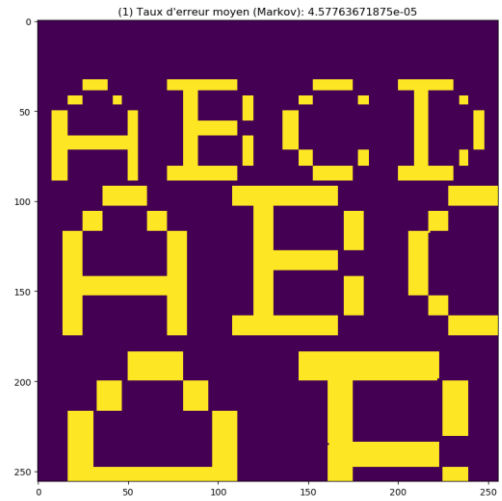
On constate une nette supériorité de la méthode des chaînes de Markov cachées dans le cas de la segmentation des images précédentes. En effet, celle-ci élimine le bruit de segmentation visible sur l'image en considérant une dépendance avec les pixels proches, ce qui limite le cas de pixels isolés qui perturbent la qualité de la segmentation dans les autres méthodes. On notera comme précédemment la supériorité du MAP sur le MV également pour les raisons évoquées lors du CR 2 : l'apport d'information sur la proportion des classes dans l'image permet d'accroître la qualité de la segmentation.

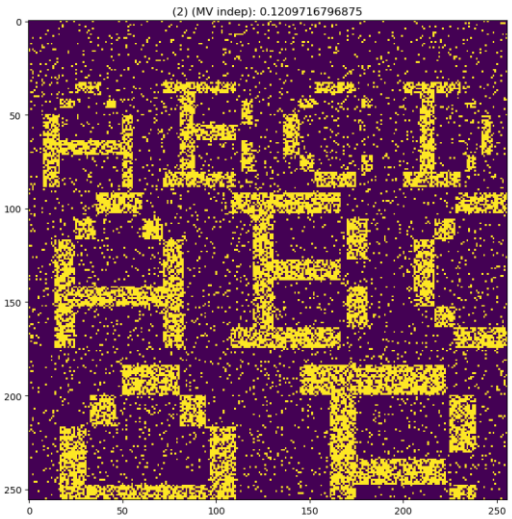
Conclusion : Nous avons vu différentes manières de segmenter un signal en différentes classes à partir de son observation bruitée. Tout d'abord, le critère du maximum de vraisemblance qui ne tient pas compte des fréquences d'apparition des classes a priori. Ensuite le critère MAP qui lui apporte l'information d'une loi d'apparition des classes. Enfin, la segmentation par les chaines de Markov cachées qui apporte un cran de dépendance entre les variables X_i et X_{i-1}. A chaque fois, l'apport d'information, et donc de dépendance entre les variables, diminue l'erreur moyenne tout en restant calculable dans des temps respectables. Si les deux premières méthodes donnent des segmentations peu acceptables lors de forts bruits, la segmentation markovienne donne des résultats probants pour l'œil nu ; même pour le bruit le plus fort. La segmentation par les chaînes de Markov cachées peut donc être privilégiée pour des jeux de données adaptées à la modélisation de dépendance entre X_i et X_{i-1}.

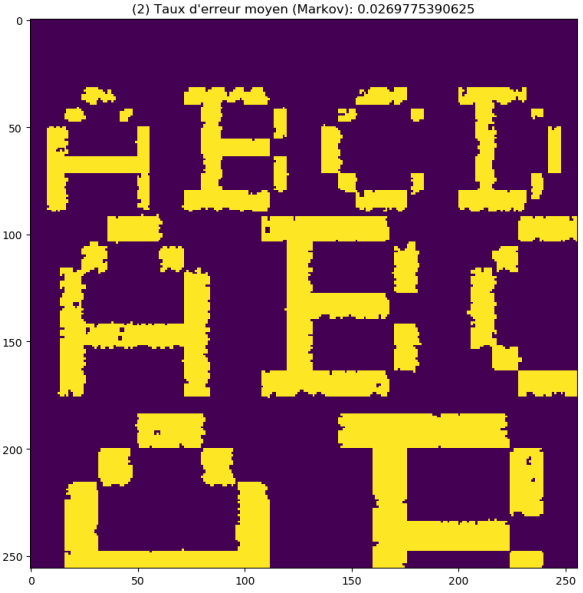
Annexes : Nos images segmentées par les différentes méthodes de segmentation

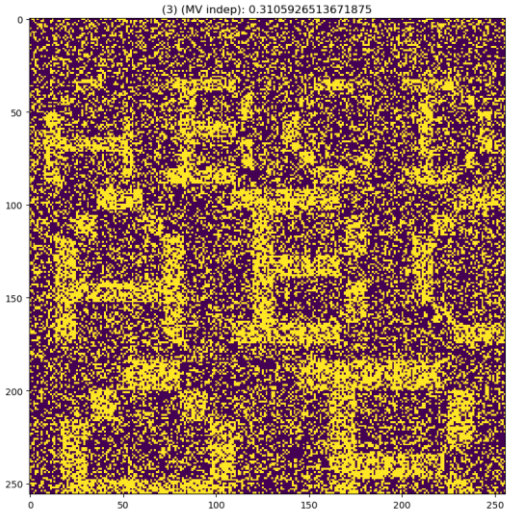


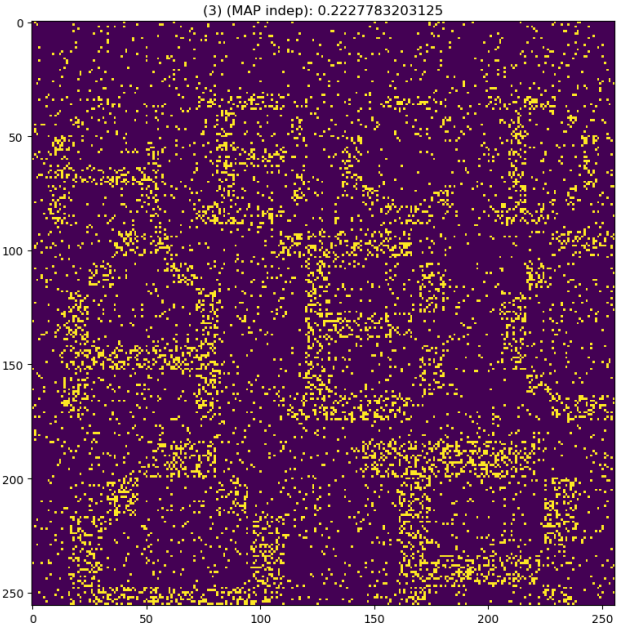


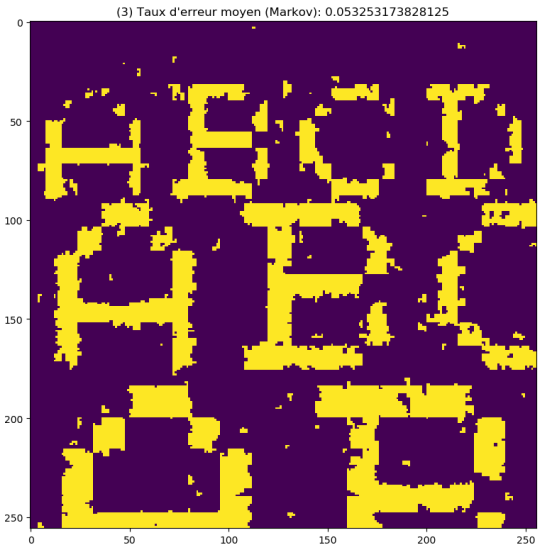






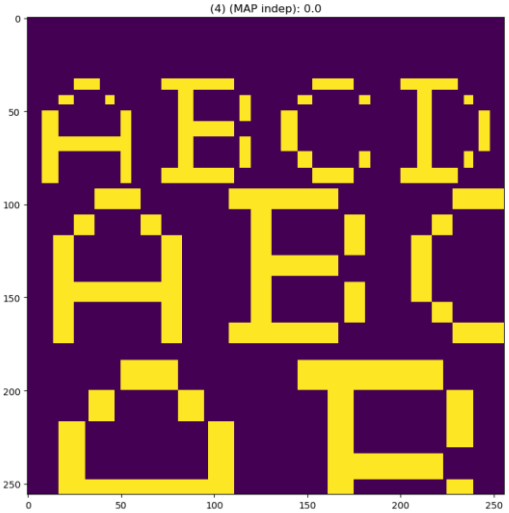

(3) (MV indep): 0.3105926513671875


(3) (MAP indep): 0.2227783203125

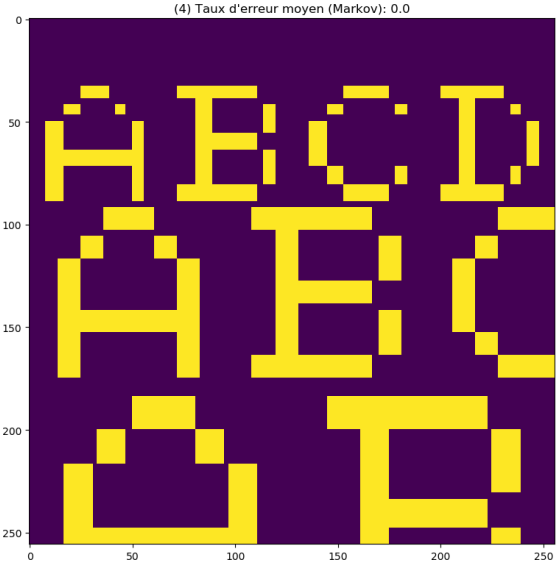

(3) Taux d'erreur moyen (Markov): 0.053253173828125


(4) (MV indep): 0.0

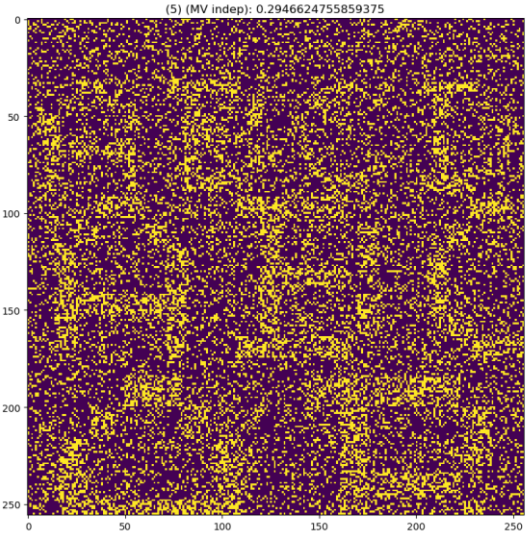
(4) (MAP indep): 0.0



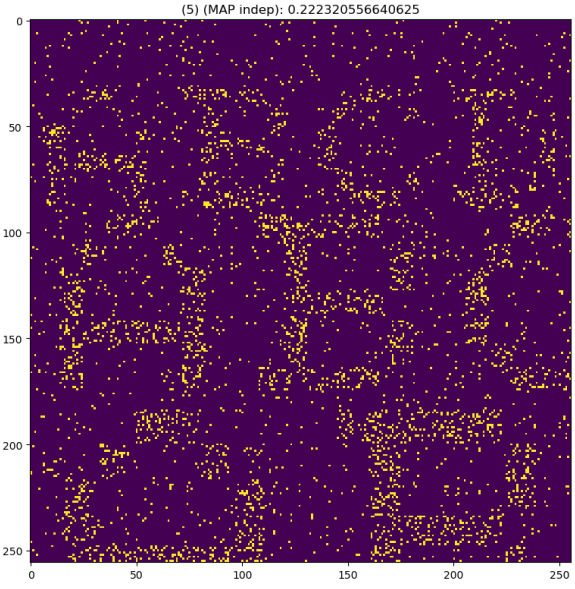
(4) Taux d'erreur moyen (Markov): 0.0



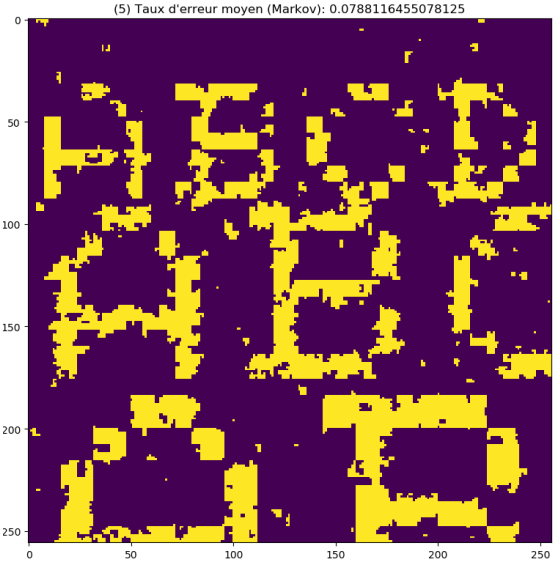
(5) (MV indep): 0.2946624755859375

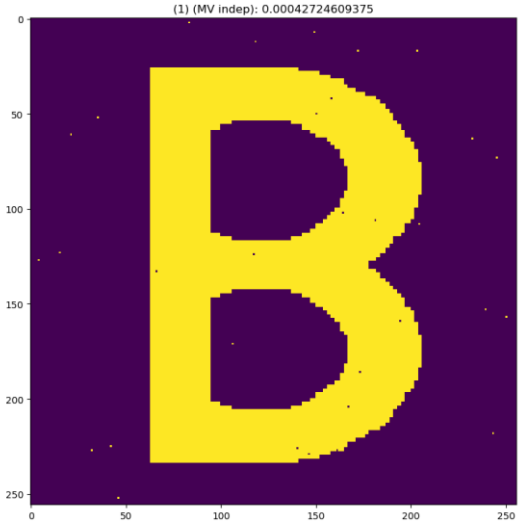


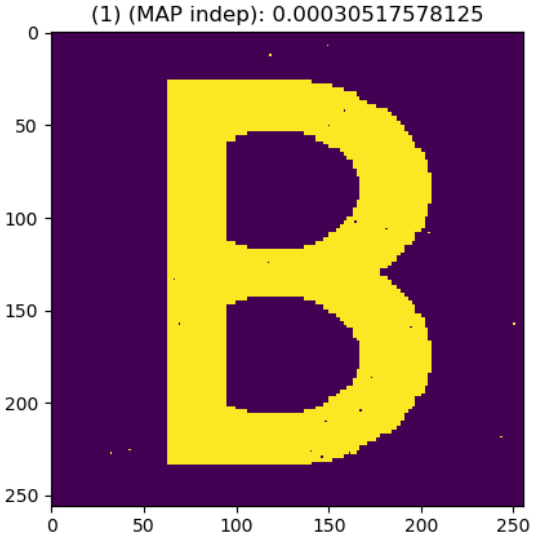
(5) (MAP indep): 0.222320556640625

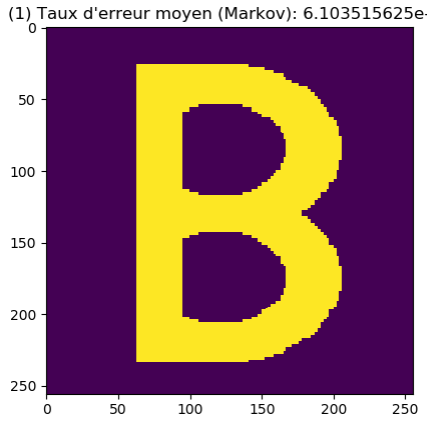


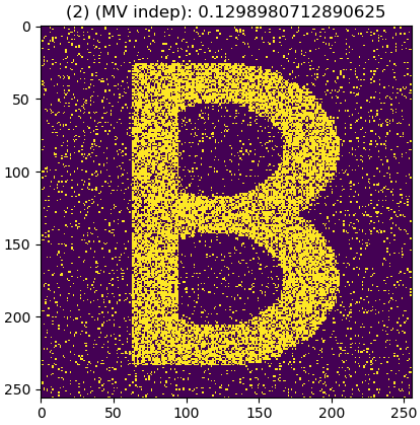
(5) Taux d'erreur moyen (Markov): 0.0788116455078125

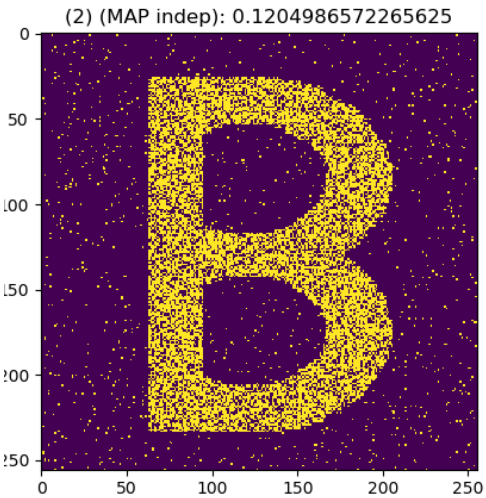


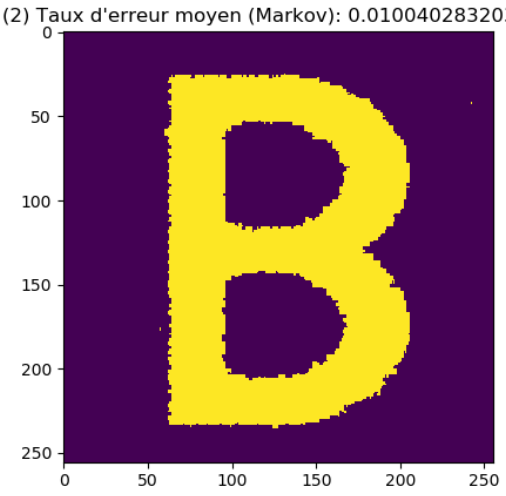

(1) (MV indep): 0.00042724609375

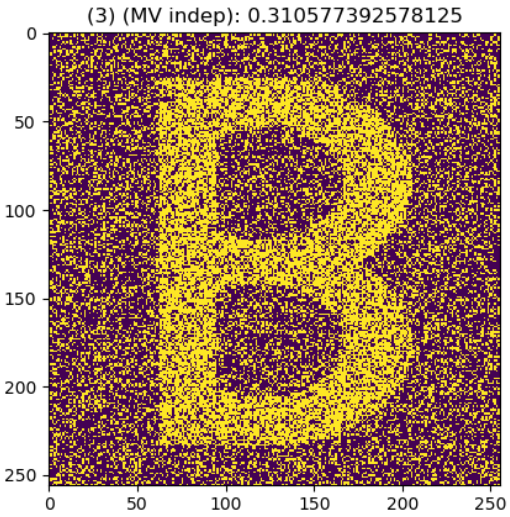

(1) (MAP indep): 0.00030517578125

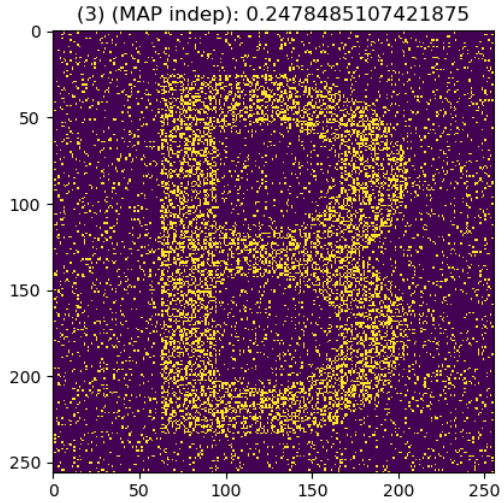

(1) Taux d'erreur moyen (Markov): 6.103515625e-05

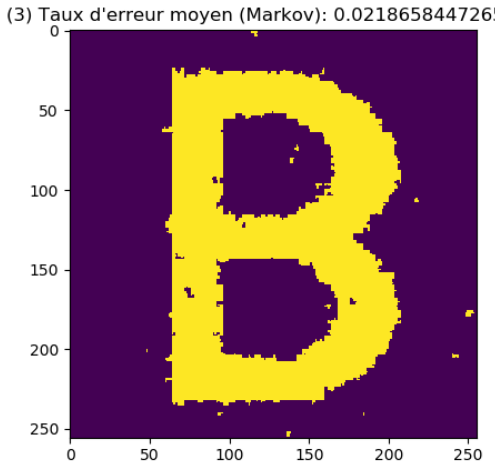

(2) (MV indep): 0.1298980712890625

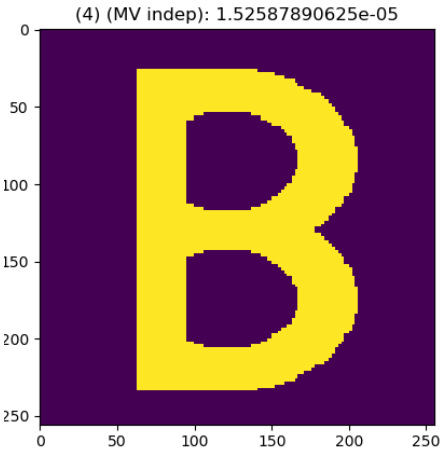

(2) (MAP indep): 0.1204986572265625

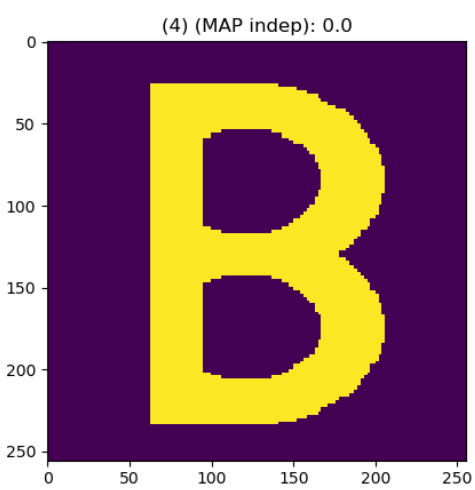

(2) Taux d'erreur moyen (Markov): 0.010040283203125

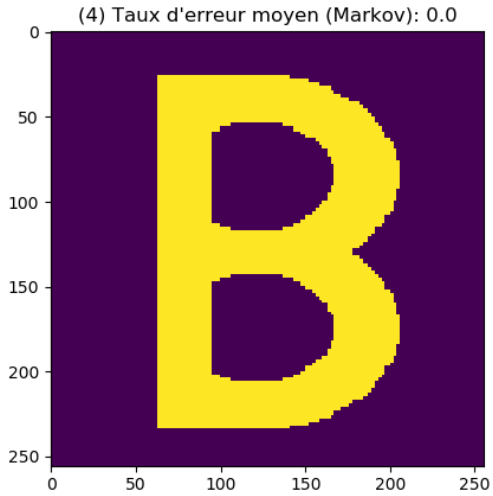

(3) (MV indep): 0.310577392578125

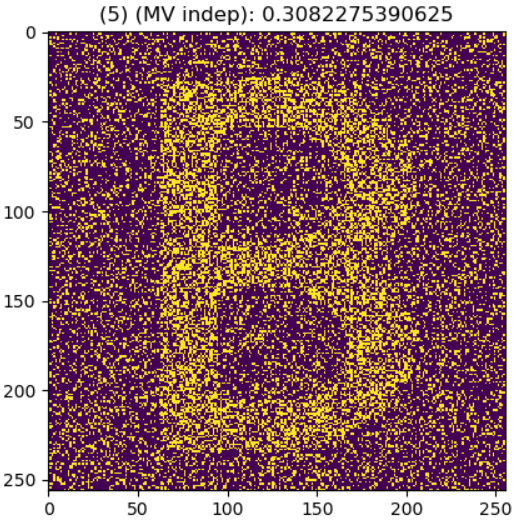

(3) (MAP indep): 0.2478485107421875

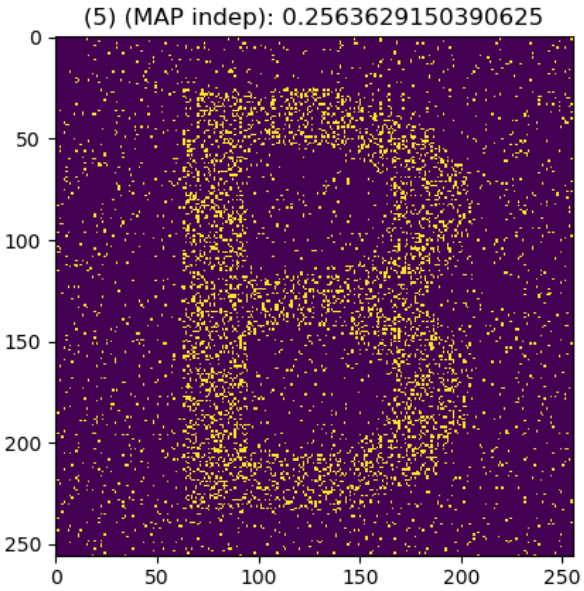

(3) Taux d'erreur moyen (Markov): 0.0218658447265625

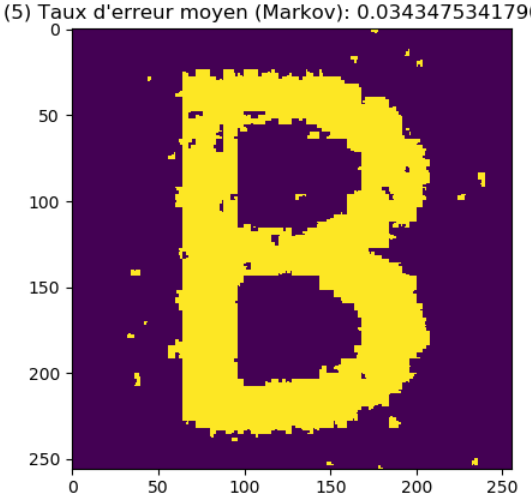

(4) (MV indep): 1.52587890625e-05

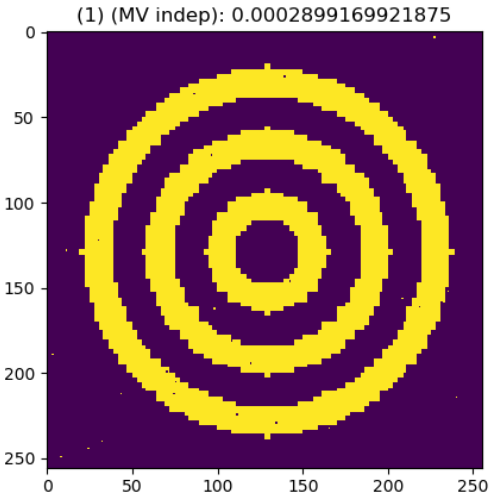

(4) (MAP indep): 0.0

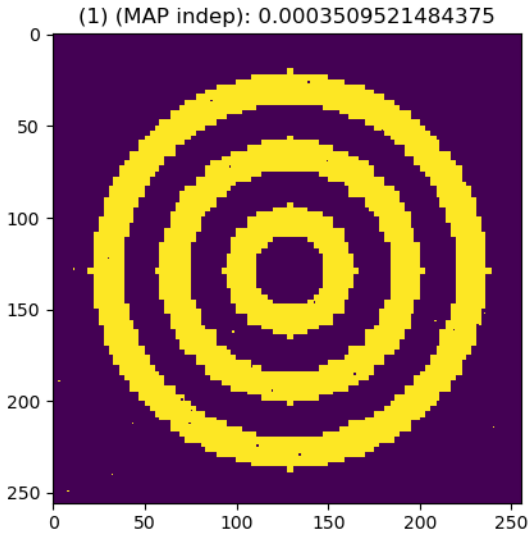

(4) Taux d'erreur moyen (Markov): 0.0

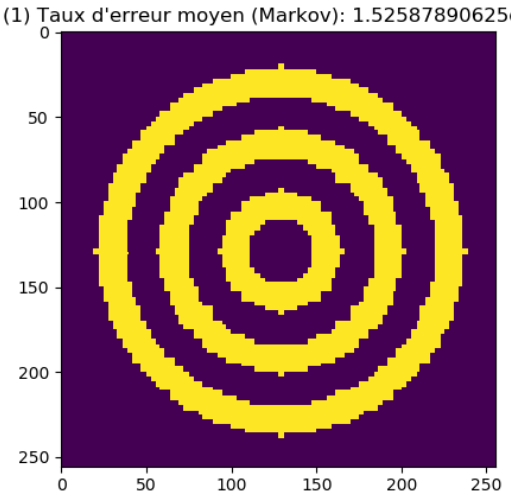

(5) (MV indep): 0.3082275390625
0
50
100
150
200
250
0
50
100
150
200
250

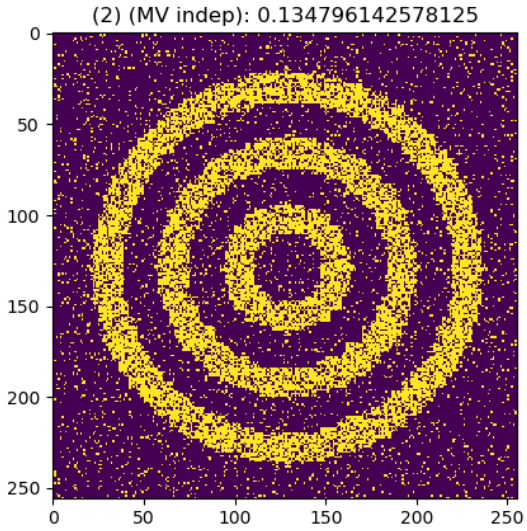

(5) (MAP indep): 0.2563629150390625
0
50
100
150
200
250
0
50
100
150
200
250

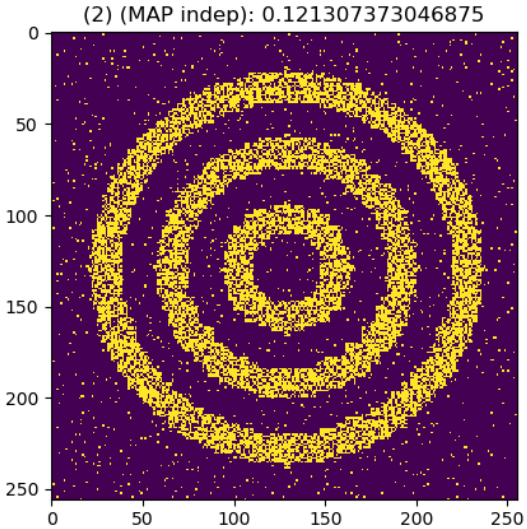

(5) Taux d'erreur moyen (Markov): 0.0343475341796875
0
50
100
150
200
250
0
50
100
150
200
250

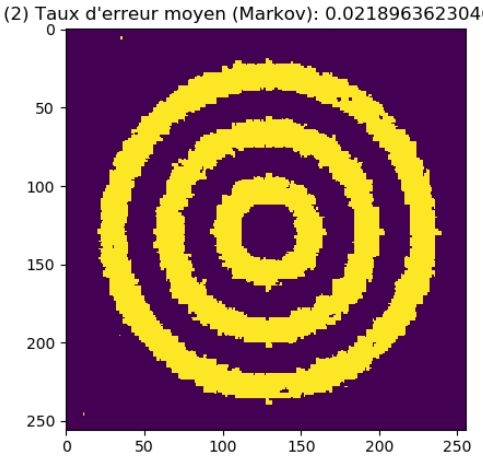

(1) (MV indep): 0.0002899169921875


(1) (MAP indep): 0.0003509521484375

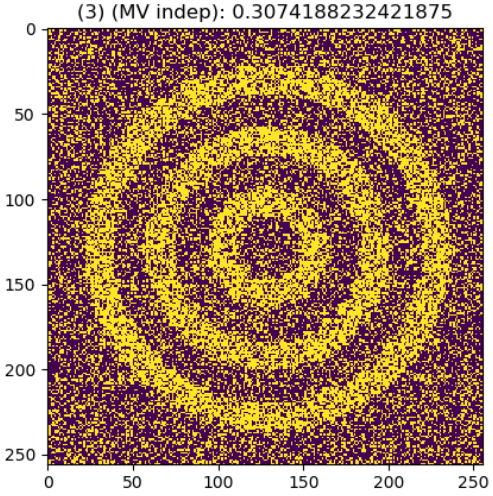

(1) Taux d'erreur moyen (Markov): 1.52587890625e-05


(2) (MV indep): 0.134796142578125

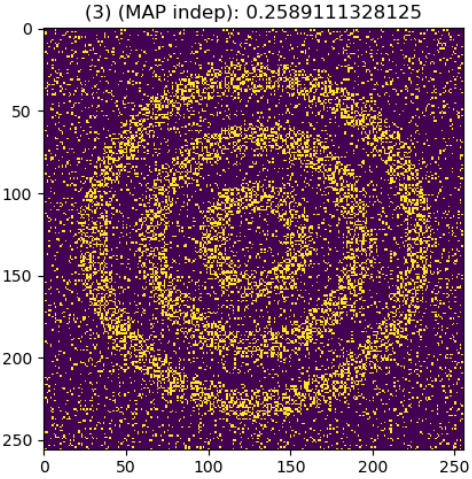

(2) (MAP indep): 0.121307373046875


(2) Taux d'erreur moyen (Markov): 0.0218963623046875

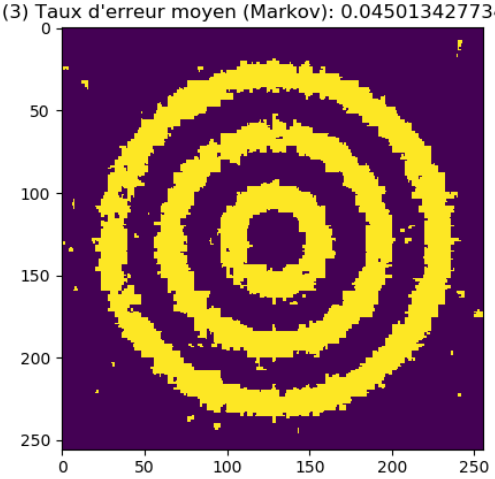
(3) (MV indep): 0.3074188232421875



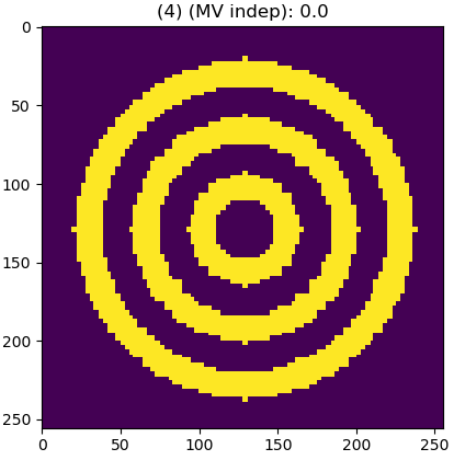
(3) (MAP indep): 0.2589111328125



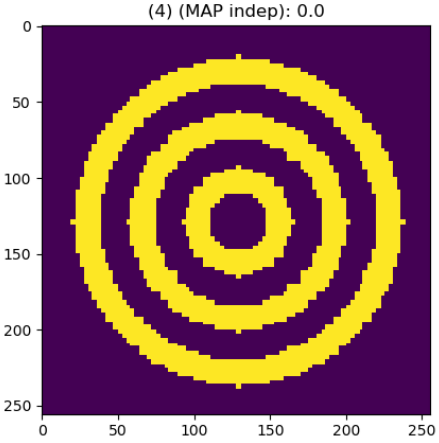
(3) Taux d'erreur moyen (Markov): 0.045013427734375



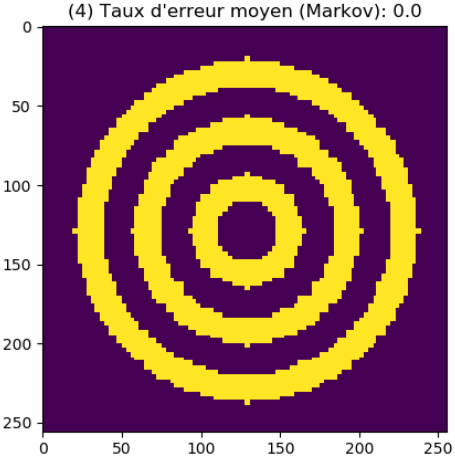
(4) (MV indep): 0.0

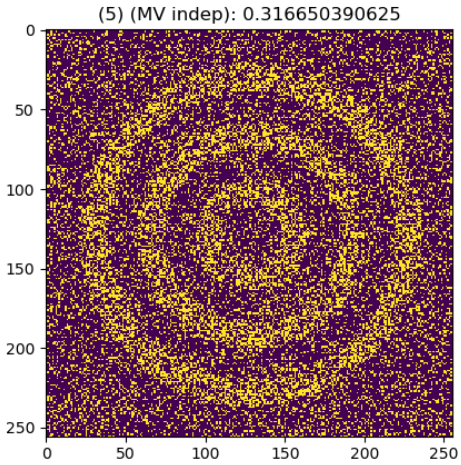


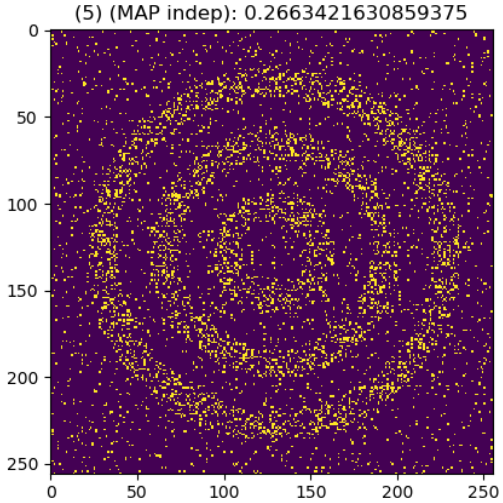
(4) (MAP indep): 0.0

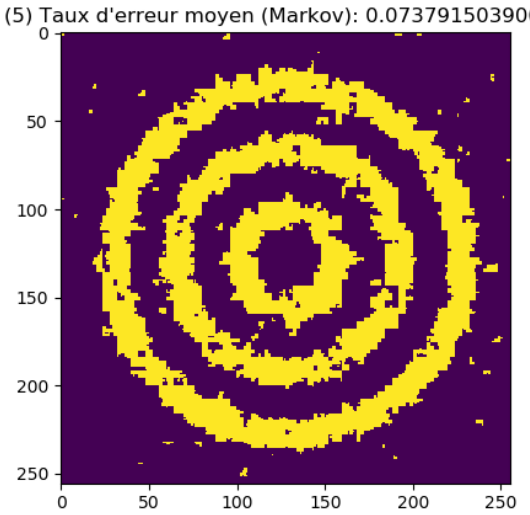


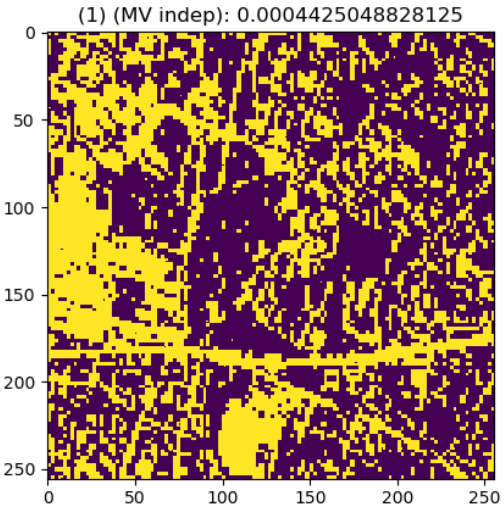
(4) Taux d'erreur moyen (Markov): 0.0

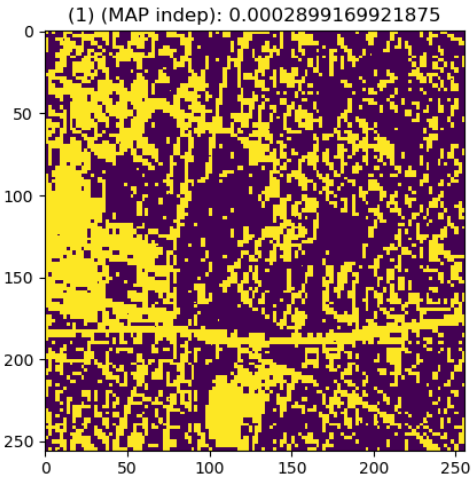


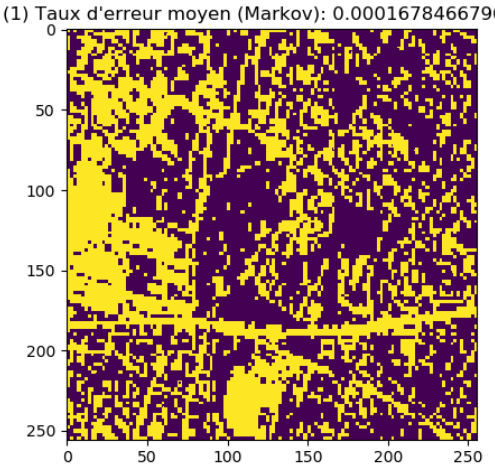

(5) (MV indep): 0.316650390625
0
50
100
150
200
250
0
50
100
150
200
250

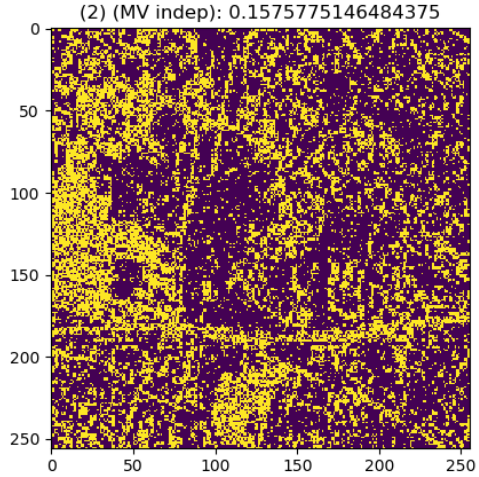

(5) (MAP indep): 0.2663421630859375
0
50
100
150
200
250
0
50
100
150
200
250

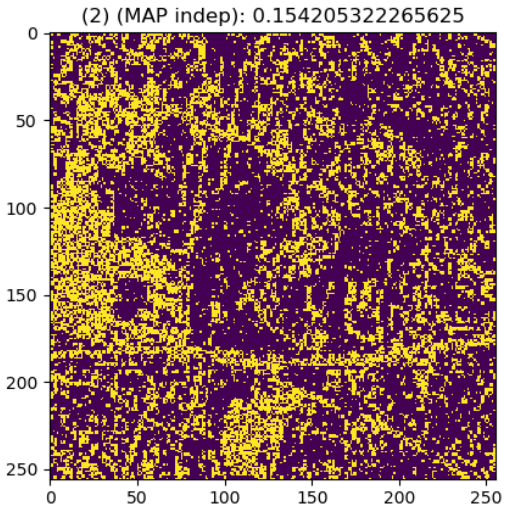

(5) Taux d'erreur moyen (Markov): 0.07379150390625
0
50
100
150
200
250
0
50
100
150
200
250

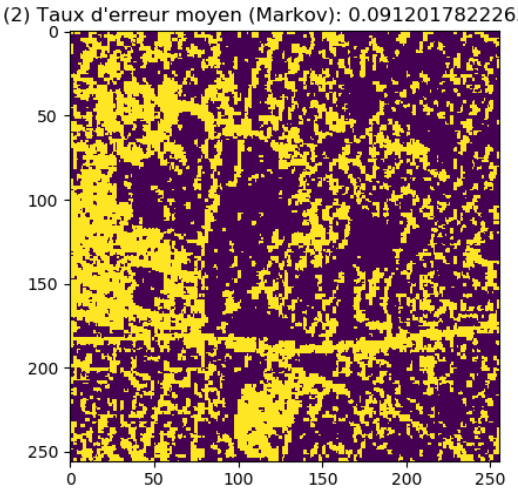

(1) (MV indep): 0.0004425048828125

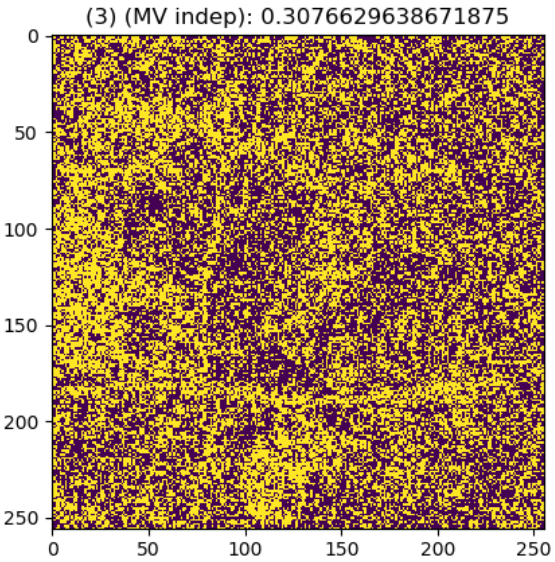

(1) (MAP indep): 0.0002899169921875

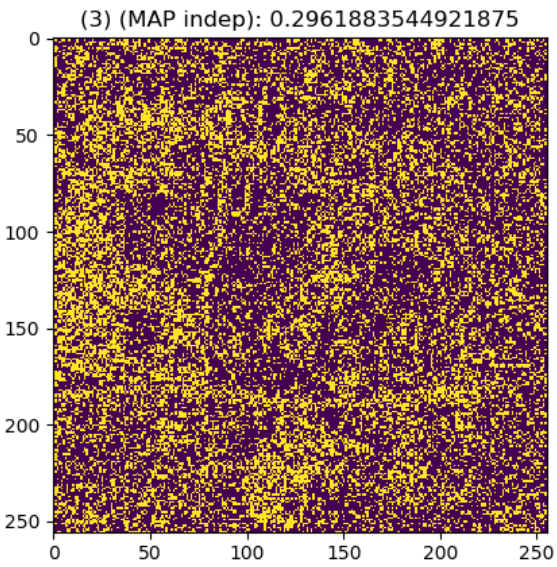

(1) Taux d'erreur moyen (Markov): 0.0001678466796875

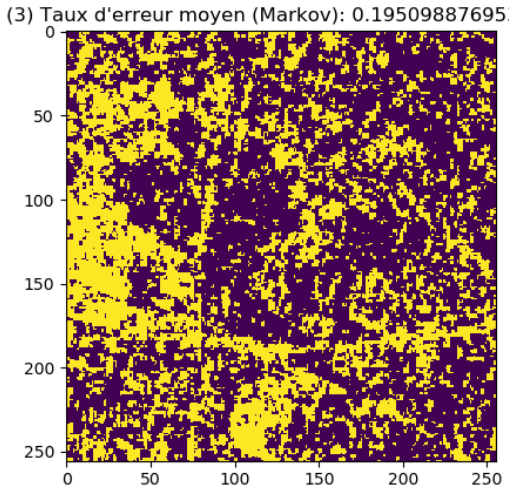

(2) (MV indep): 0.1575775146484375

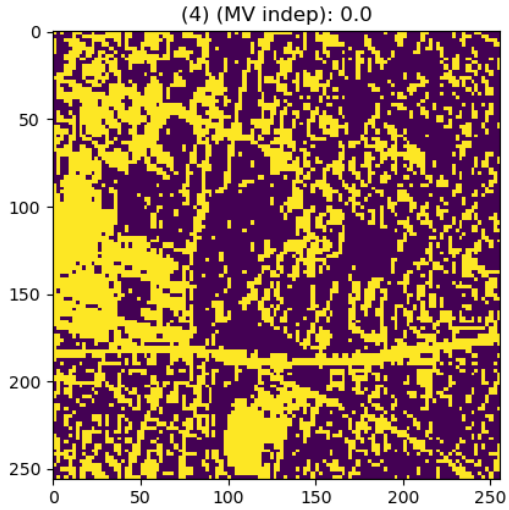

(2) (MAP indep): 0.154205322265625

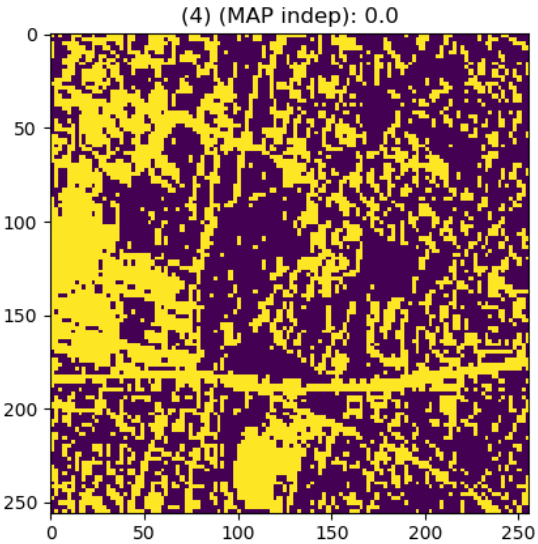

(2) Taux d'erreur moyen (Markov): 0.0912017822265625

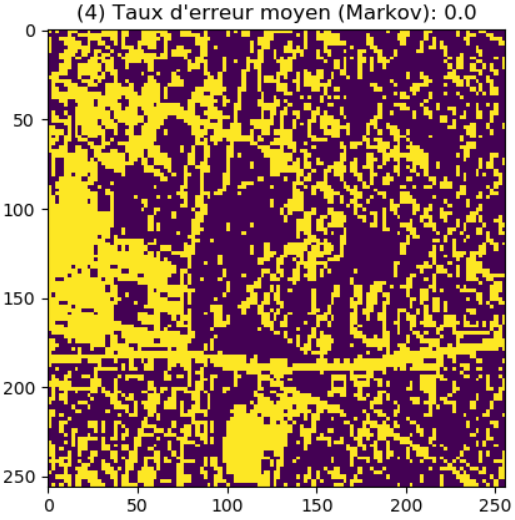

(3) (MV indep): 0.3076629638671875

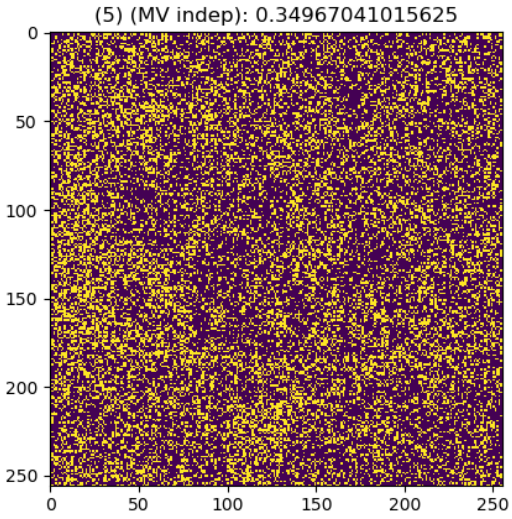

(3) (MV indep): 0.2961883544921875

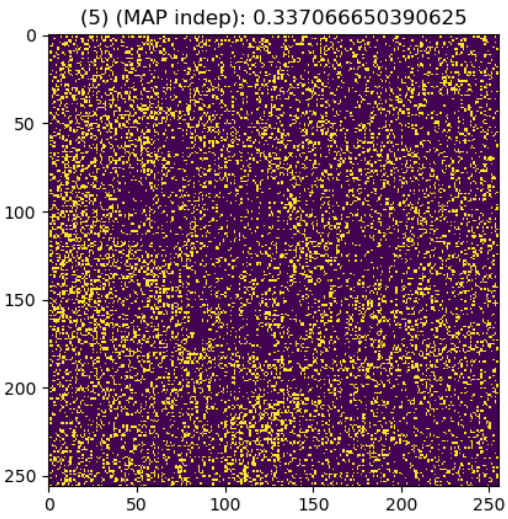

(3) Taux d'erreur moyen (Markov): 0.195098876953125

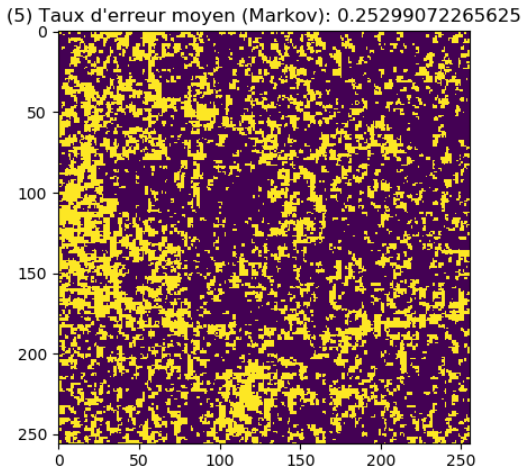

(4) (MV indep): 0.0


(4) (MAP indep): 0.0

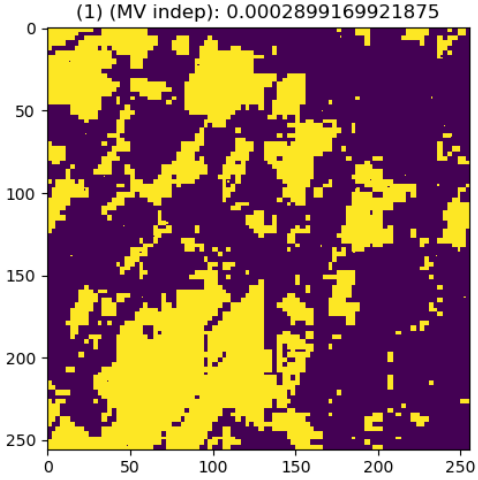

(4) Taux d'erreur moyen (Markov): 0.0


(5) (MV indep): 0.34967041015625
0
50
100
150
200
250
0
50
100
150
200
250

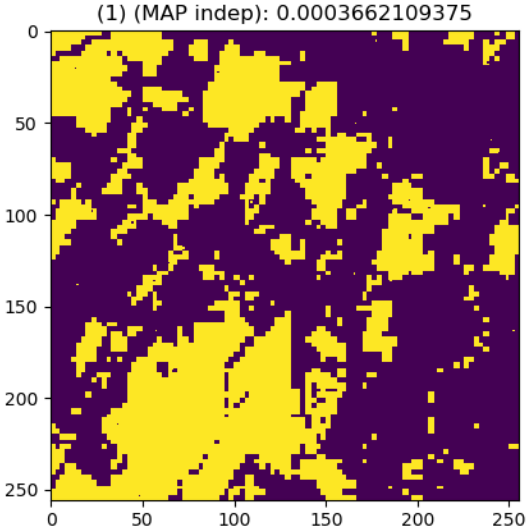

(5) (MAP indep): 0.337066650390625
0
50
100
150
200
250
0
50
100
150
200
250


(5) Taux d'erreur moyen (Markov): 0.25299072265625
0
50
100
150
200
250
0
50
100
150
200
250

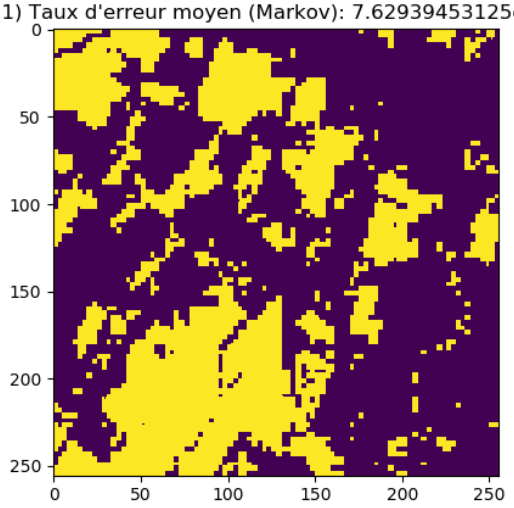
(1) (MV indep): 0.0002899169921875



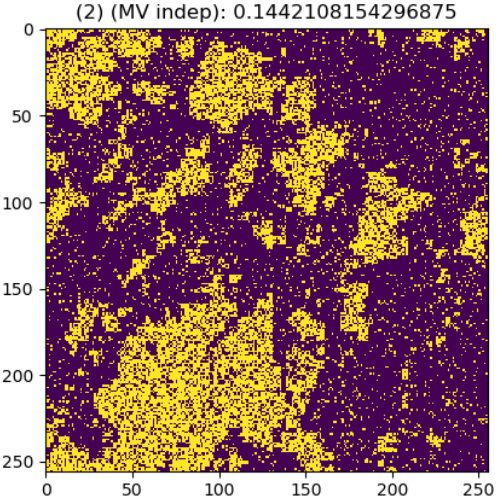
(1) (MAP indep): 0.0003662109375



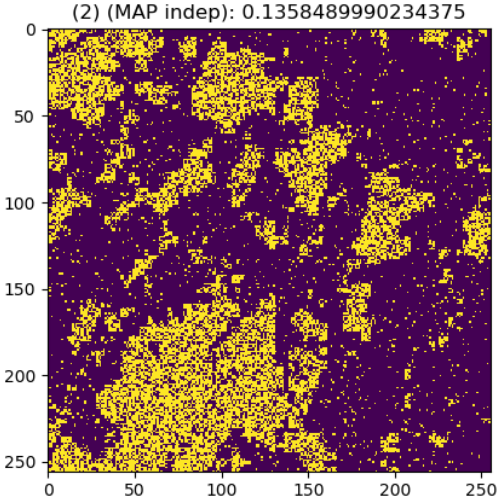
(1) Taux d'erreur moyen (Markov): 7.62939453125e-05



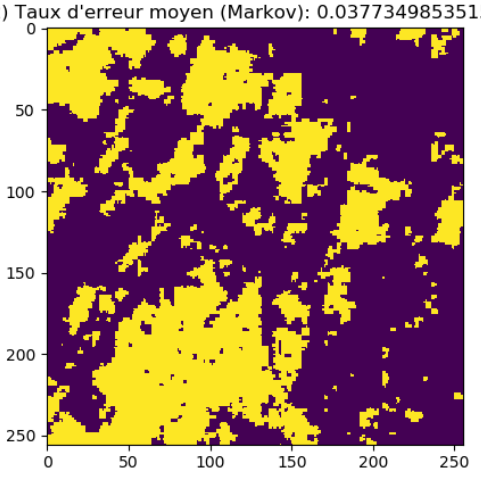
(2) (MV indep): 0.1442108154296875



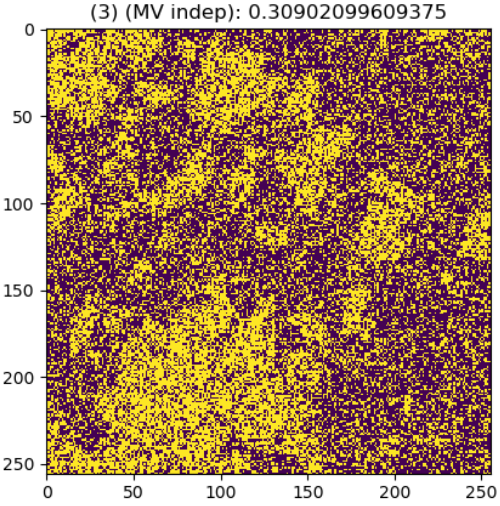
(2) (MAP indep): 0.1358489990234375



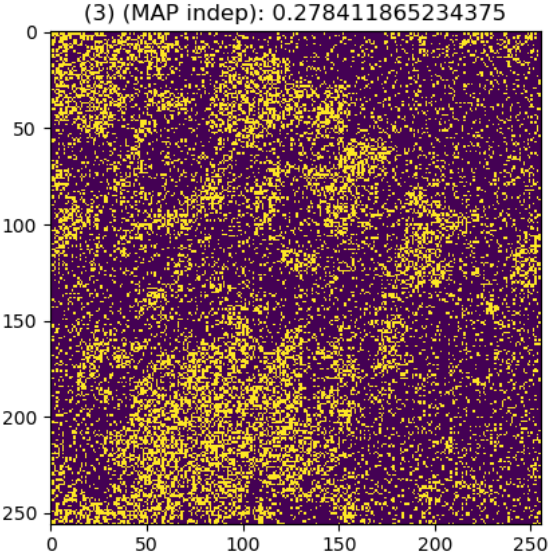
(2) Taux d'erreur moyen (Markov): 0.0377349853515625



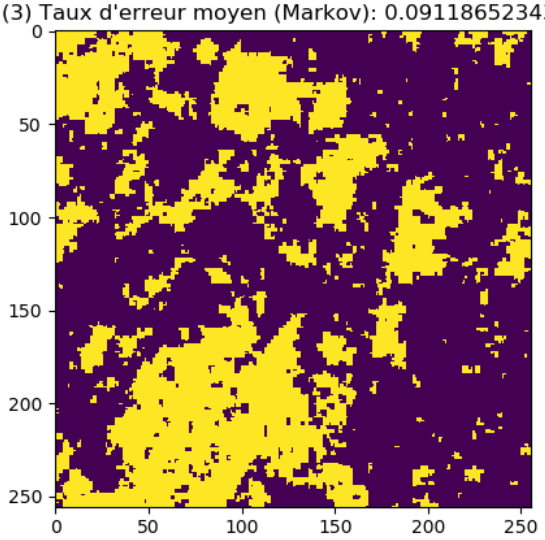
(3) (MV indep): 0.30902099609375



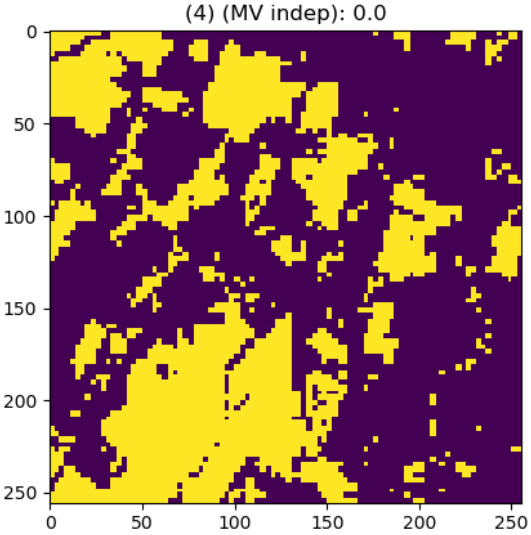
(3) (MAP indep): 0.278411865234375



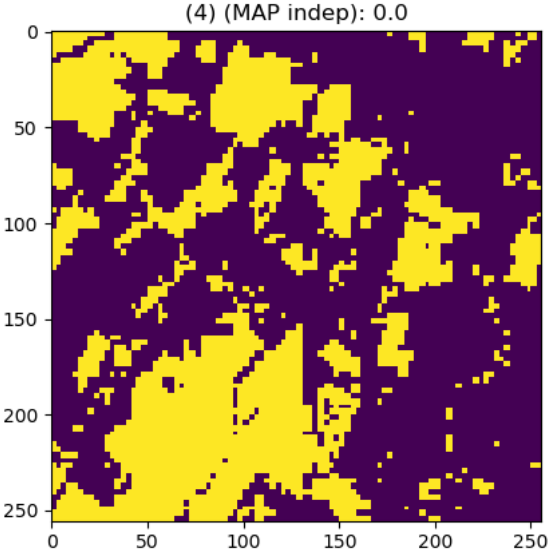
(3) Taux d'erreur moyen (Markov): 0.0911865234375



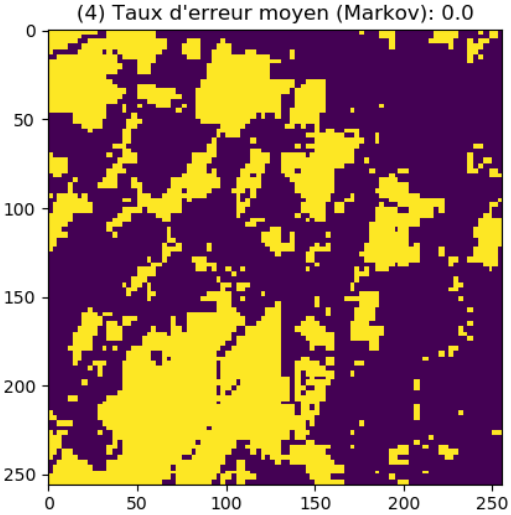
(4) (MV indep): 0.0

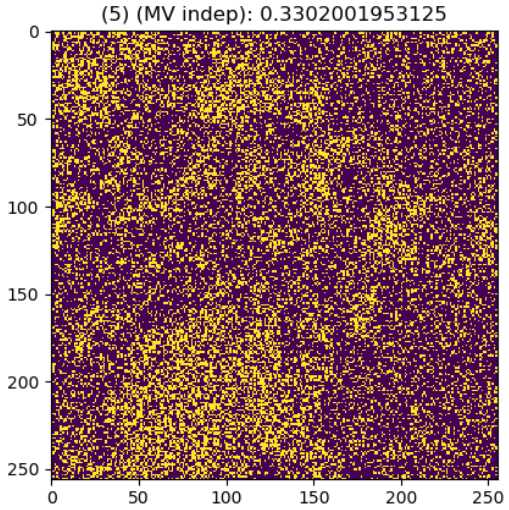


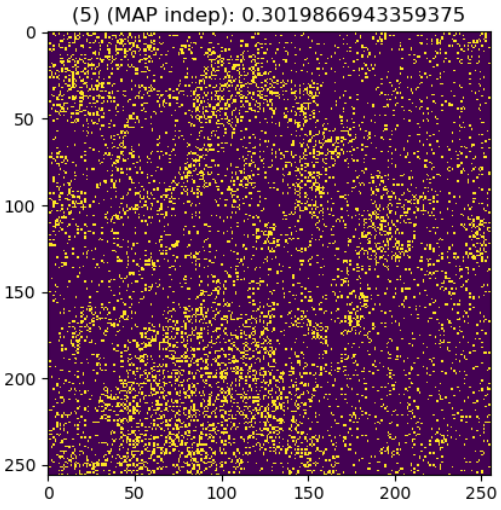
(4) (MAP indep): 0.0

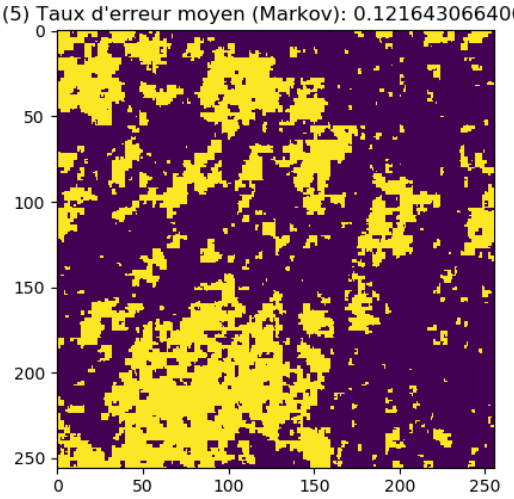


(4) Taux d'erreur moyen (Markov): 0.0




(5) (MV indep): 0.3302001953125
0
50
100
150
200
250


(5) (MAP indep): 0.3019866943359375
0
50
100
150
200
250


(5) Taux d'erreur moyen (Markov): 0.12164306640625
0
50
100
150
200
250

Nos codes Python (fournis en annexe dans le ZIP)

```
## 1

# Construction d'une matrice Mat_f pour le calcul des proba forward et backward
def gauss2(Y, m1, sig1, m2, sig2):
    Mat_f = []

    for y in Y:
        f1 = norm.pdf(y, loc=m1, scale=sig1)
        f2 = norm.pdf(y, loc=m2, scale=sig2)

        Mat_f.append([f1, f2])

    return np.array(Mat_f)


## 2

# Calcul récursif des composantes de la matrice alfa par le processus forward
def forward2(Mat_f, n, A, p10):
    a1 = [Mat_f[0][0] * p10, Mat_f[0][1] * (1 - p10)]
    alfa = [a1]

    for z in range(1, n):

        sumprevious0 = 0
        sumprevious1 = 0
        for j in range(0, len(a1)):
            sumprevious0 = alfa[z - 1][j] * A[j][0] + sumprevious0
            sumprevious1 = alfa[z - 1][j] * A[j][1] + sumprevious1

        sumprevious00 = sumprevious0 * Mat_f[z][0]
        sumprevious11 = sumprevious1 * Mat_f[z][1]

        aBis = [sumprevious00, sumprevious11]
        alfa.append(aBis)

    return alfa


## 3

# Calcul récursif des composantes de la matrice betha par le processus backward
def backward2(Mat_f, n, A, p10):
    bethaN = [1, 1]
    betha = [bethaN]

    for z in range(1, n):

        sumprevious0 = 0
        sumprevious1 = 0
        for j in range(0, len(bethaN)):
            sumprevious0 = betha[z - 1][j] * A[0][j] * Mat_f[n - z][j] + sumprevious0
            sumprevious1 = betha[z - 1][j] * A[1][j] * Mat_f[n - z][j] + sumprevious1

        aBis = [sumprevious0, sumprevious1]
        betha.append(aBis)

    betha.reverse()

    return betha
```

```
## 4

# Segmente le signal par le critère MPM pour les chaines de markov cachées
def MPM_chaine2(Mat_f, n, cl1, cl2, A, p10, rescaling):
    # Liste des sous-éléments segmentés
    S = []

    if rescaling:
        alfa = forward2_rescaled(Mat_f, n, A, p10)
        betha = backward2_rescaled(Mat_f, n, A, p10)
    else:
        alfa = forward2(Mat_f, n, A, p10)
        betha = backward2(Mat_f, n, A, p10)
    for i in range(0, len(alfa)):

        if rescaling:

            normal = alfa[i][0] * betha[i][0] + alfa[i][1] * betha[i][1]

            c = alfa[i][0] * betha[i][0] / normal
            d = alfa[i][1] * betha[i][1] / normal
        else:
            c = alfa[i][0] * betha[i][0]
            d = alfa[i][1] * betha[i][1]

        if c > d:
            S.append(cl1)
        else:
            S.append(cl2)

    return S
```

```
## 5 & 8

def seg_chaines_MPM_super2(n, cl1, cl2, A, p10, m1, sig1, m2, sig2, rescaling, plot=True, forceX=None, index=0):
    if forceX is None:
        X = genere_Chaine2(n=n, cl1=cl1, cl2=cl2, A=A, p10=p10)
    else:
        X = forceX
    Y = bruit_gauss2(X, cl1, cl2, m1, sig1, m2, sig2)

    Mat_f = gauss2(Y, m1, sig1, m2, sig2)

    S = MPM_chaine2(Mat_f, n, cl1, cl2, A, p10, rescaling)

    if plot:
        tau = erreur_moyenne_MAP_chaine2(1, n, A, p10, m1, sig1, m2, sig2, toPlot=False, forceX=X, index=index,
                                         rescaling=rescaling)
        abscisse = []
        for i in range(0, len(X)):
            abscisse.append(i)
        plt.plot(abscisse, X, "r-")
        plt.plot(abscisse, Y, "g.")
        plt.plot(abscisse, S, "b.")
        plt.title("(" + str(index) + ") Taux d'erreur (CM-MAP) : " + str(tau))
        plt.show()

    return S
```

```python
# Calcul moyen du taux d'erreur
def erreur_moyenne_MAP_chaine2(T, n, A, p10, m1, sig1, m2, sig2, toPlot=False, forceX=np.load("signal.npy"), index=0,
                               rescaling=False):
    # Somme du calul de la moyenne
    tau = 0

    # Calcul de l'erreur partielle pour le plot
    erreur_partielle = []
    abscisse = []

    # Chargement des données
    X = forceX

    # Récupération des classes
    cl1, cl2 = np.unique(X)

    for k in range(0, T):
        # Bruitage du signal
        Y = bruit_gauss2(X, cl1, cl2, m1, sig1, m2, sig2)

        # Segmentation
        Mat_f = gauss2(Y, m1, sig1, m2, sig2)

        S = MPM_chaine2(Mat_f, n, cl1, cl2, A, p10, rescaling)

        # Construction de la somme
        tau += taux_erreur(X, S)

        # Ajout du taux d'erreur partiel
        erreur_partielle.append(tau / (k + 1))
        abscisse.append(k + 1)

    if toPlot:
        # Plot de l'erreur partielle
        plt.plot(abscisse, erreur_partielle)
        plt.title(
            "(" + str(index) + ") Taux d'erreur cumulé ((" + str(m1) + "," + str(sig1) + ") -" + " (" + str(
                m2) + "," + str(sig2) + "))")
        plt.show()

    return tau / T
```

```python
def test6(n):
    M1 = [120, 127, 127, 127, 127]
    M2 = [130, 127, 128, 128, 128]
    SIG1 = [1, 1, 1, 0.1, 2]
    SIG2 = [2, 5, 1, 0.1, 3]
    A_ARRAY = []

    PREPARE_A = [0.1, 0.2, 0.3, 0.4, 0.5]
    PREPARE_A2 = [0.5, 0.4, 0.3, 0.2, 0.1]

    cl1, cl2 = 100, 200
    p10 = 0.5

    for j in range(0, len(PREPARE_A)):
        i = PREPARE_A[j]
        k = PREPARE_A2[j]
        A_ARRAY.append(np.array([[i, 1 - i], [k, 1 - k]]))

    abscisse = []
    for i in range(0, n):
        abscisse.append(i)

    for i in range(0, len(A_ARRAY)):

        A = A_ARRAY[i]
        X = genere_Chaine2(n=n, cl1=cl1, cl2=cl2, A=A, p10=p10)

        for j in range(0, len(M1)):
            m1 = M1[j]
            m2 = M2[j]
            sig1 = SIG1[j]
            sig2 = SIG2[j]
            seg_chaines_MPM_super2(n, cl1, cl2, A, p10, m1, sig1, m2, sig2, rescaling=False, plot=True, forceX=X,
                                   index=("bruit " + str(j + 1) + " - signal " + str(i) + " - A[0][0] = " + str(A[0][0])))

            [p1, p2] = calc_probaprio(X, cl1, cl2)

            # Bruitage de l'échantillon
            Y = bruit_gauss2(X, cl1, cl2, M1[j], SIG1[j], M2[j], SIG2[j])

            # Classification
            S = MAP_MPM2(Y, cl1, cl2, p1, p2, M1[j], SIG1[j], M2[j], SIG2[j])

            # Plot
            plt.plot(abscisse, X, 'r-')
            plt.plot(abscisse, Y, 'g.')
            plt.plot(abscisse, S, 'b.')
            plt.title(
                "(" + "bruit " + str(j + 1) + " - signal " + str(i) + " - A[0][0] = " + str(A[0][0]) + ") Taux d'erreur moyer
                    erreur_moyenne_MAP(100, p1, p2, M1[j], SIG1[j], M2[j], SIG2[j], forceX=X)))
            plt.show()
```

```python
# Generateur de la matrice de transition
def calc_transit_prio2(X, n, cl1, cl2):
    metcl1 = False
    metcl2 = False

    indexCl1 = 0
    indexCl2 = 0
    nCl1 = 0
    nCl2 = 0
    taille = len(X)
    for x in X:
        if x == cl1:
            if metcl2:
                metcl2 = False
            if metcl1:
                indexCl1 += 1

            metcl1 = True
            nCl1 += 1
        else:
            if metcl1:
                metcl1 = False

            if metcl2:
                indexCl2 += 1

            metcl2 = True
            nCl2 += 1

    pCl1toCl1 = indexCl1 / nCl1
    pCl2toCl2 = indexCl2 / nCl2
    pCL1toCl2 = 1 - pCl1toCl1
    pCl2toCl1 = 1 - pCl2toCl2

    return np.array([[pCl1toCl1, pCL1toCl2],
                     [pCl2toCl1, pCl2toCl2]])
```

```python
# Test sur les signaux 0 à 5
def testSignals(rescaling):
    M1 = [120, 127, 127, 127, 127]
    M2 = [130, 127, 128, 128, 128]
    SIG1 = [1, 1, 1, 0.1, 2]
    SIG2 = [2, 5, 1, 0.1, 3]
    X_ARRAY_NAME = ["signal.npy", "signal1.npy", "signal2.npy", "signal3.npy", "signal4.npy", "signal5.npy"]
    X_ARRAY = []

    p10 = 0.5

    for name in X_ARRAY_NAME:
        X_ARRAY.append(np.load(name))

    for i in range(0, len(X_ARRAY)):

        X = X_ARRAY[i]
        cl1, cl2 = np.unique(X)
        A = calc_transit_prio2(X, len(X), cl1, cl2)

        abscisse = []
        for k in range(0, len(X)):
            abscisse.append(k)

        for j in range(0, len(M1)):
            m1 = M1[j]
            m2 = M2[j]
            sig1 = SIG1[j]
            sig2 = SIG2[j]
            seg_chaines_MPM_super2(len(X), cl1, cl2, A, p10, m1, sig1, m2, sig2, rescaling, plot=True, forceX=X,
                                   index=("bruit " + str(j + 1) + " - signal " + str(i)))

            [p1, p2] = calc_probaprio(X, cl1, cl2)

            # Bruitage de l'échantillon
            Y = bruit_gauss2(X, cl1, cl2, M1[j], SIG1[j], M2[j], SIG2[j])

            # Classification
            S = MAP_MPM2(Y, cl1, cl2, p1, p2, M1[j], SIG1[j], M2[j], SIG2[j])

            # Plot
            plt.plot(abscisse, X, 'r-')
            plt.plot(abscisse, Y, 'g.')
            plt.plot(abscisse, S, 'b.')
            plt.title(
                "(" + "bruit " + str(j + 1) + " - signal " + str(i) + ") Taux d'erreur moyen (MAP indep): " + str(
                    erreur_moyenne_MAP(1, p1, p2, M1[j], SIG1[j], M2[j], SIG2[j], forceX=X)))
            plt.show()
```

```python
# Programmation du rescaling
# Calcul récursif des composantes de la matrice alfa par le processus forward
def forward2_rescaled(Mat_f, n, A, p10):
    a1 = np.array([Mat_f[0][0] * p10, Mat_f[0][1] * (1 - p10)])
    Am = [a1[0] + a1[1]]
    alfaprime = [a1 / Am[0]]

    for z in range(1, n):

        alfaprimeprime = alfaprime[z - 1]
        An = alfaprimeprime[0] + alfaprimeprime[1]
        Am.append(An)
        alfaprimeValue = np.array(alfaprime[z - 1])

        sumprevious0 = 0
        sumprevious1 = 0
        for j in range(0, len(a1)):
            sumprevious0 = alfaprimeValue[j] * A[j][0] + sumprevious0
            sumprevious1 = alfaprimeValue[j] * A[j][1] + sumprevious1

        sumprevious00 = sumprevious0 * Mat_f[z][0]
        sumprevious11 = sumprevious1 * Mat_f[z][1]

        alfaprimeValue0 = sumprevious00 / An
        alfaprimeValue1 = sumprevious11 / An

        aBis = [alfaprimeValue0, alfaprimeValue1]
        alfaprime.append(aBis)

    return alfaprime
```

```python
## 3

# Calcul récursif des composantes de la matrice betha par le processus backward
def backward2_rescaled(Mat_f, n, A, p10):
    bethaN = np.array([1, 1])
    B = [bethaN[0] + bethaN[1]]
    bethaprime = [bethaN / B[0]]

    for z in range(1, n):

        bethaprimeprime = bethaprime[z - 1]
        Bn = bethaprimeprime[0] + bethaprimeprime[1]
        B.append(Bn)
        bethaprimeValue = np.array(bethaprime[z - 1])

        sumprevious0 = 0
        sumprevious1 = 0
        for j in range(0, len(bethaN)):
            sumprevious0 = bethaprimeValue[j] * A[0][j] * Mat_f[n - z][j] + sumprevious0
            sumprevious1 = bethaprimeValue[j] * A[1][j] * Mat_f[n - z][j] + sumprevious1

        bethaprimeValue0 = sumprevious0 / Bn
        bethaprimeValue1 = sumprevious1 / Bn

        aBis = [bethaprimeValue0, bethaprimeValue1]
        bethaprime.append(aBis)

    bethaprime.reverse()

    return bethaprime
```

```python
##------- TP 5 ----------##

import chain_to_image_functions
from PIL import Image


# 4
# Bruite et segmente une image à deux classes par la méthode MPM en chaîne de Markov cachée
def segmentation_image(image, m1, sig1, m2, sig2):
    X_ch = chain_to_image_functions.image_to_chain(image)

    cl1, cl2 = np.unique(X_ch)
    A = calc_transit_prio2(X_ch, len(X_ch), cl1, cl2)
    p10 = 0.5

    S1 = seg_chaines_MPM_super2(len(X_ch), cl1, cl2, A, p10, m1, sig1, m2, sig2, rescaling=True, plot=False,
                                forceX=X_ch, index=0)

    [p1, p2] = calc_probaprio(X_ch, cl1, cl2)
    # Bruitage de l'échantillon
    Y = bruit_gauss2(X_ch, cl1, cl2, m1, sig1, m2, sig2)
    # Classification MAP
    S2 = MAP_MPM2(Y, cl1, cl2, p1, p2, m1, sig1, m2, sig2)
    # Segmentation MV
    S3 = classif_gauss2(Y, cl1, cl2, m1, sig1, m2, sig2)

    return X_ch, S1, S2, S3
```

```
# Affiche une chaîne de markov en image
def affiche(X, title):
    X_img = chain_to_image_functions.chain_to_image(X)
    plt.title(title)
    plt.imshow(X_img)
    plt.show()


# test final
def finaltest():
    M1 = [120, 127, 127, 127, 127]
    M2 = [130, 127, 128, 128, 128]
    SIG1 = [1, 1, 1, 0.1, 2]
    SIG2 = [2, 5, 1, 0.1, 3]
    X_ARRAY_NAME = [np.asarray(Image.open("alfa2.bmp")),
                    np.asarray(Image.open("beee2.bmp")),
                    np.asarray(Image.open("cible2.bmp")),
                    np.asarray(Image.open("city2.bmp")),
                    np.asarray(Image.open("country2.bmp"))]

    p10 = 0.5

    for i in range(0, len(X_ARRAY_NAME)):

        img = X_ARRAY_NAME[i]

        for j in range(0, len(M1)):
            X, S1, S2, S3 = segmentation_image(img, M1[j], SIG1[j], M2[j], SIG2[j])
            cl1, cl2 = np.unique(X)
            [p1, p2] = calc_probaprio(X, cl1, cl2)
            A = calc_transit_prio2(X, len(X), cl1, cl2)

            error_Map = "(" + str(j + 1) + ") (MAP indep): " \
                        + str(erreur_moyenne_MAP(1,
                                                 p1,
                                                 p2,
                                                 M1[j],
                                                 SIG1[j],
                                                 M2[j],
                                                 SIG2[j],
                                                 forceX=X))

            error_MV = "(" + str(j + 1) + ") (MV indep): " \
                       + str(erreur_moyenne_MV(1, M1[j], SIG1[j], M2[j], SIG2[j], forceX=X))
            error_Markov = "(" + str(j + 1) + ") Taux d'erreur moyen (Markov): " + \
                           str(erreur_moyenne_MAP_chaine2(1, len(X), A, p10, M1[j], SIG1[j], M2[j], SIG2[j],
                                                          toPlot=False, forceX=X, index=j,
                                                          rescaling=True))

            affiche(S3, error_MV)
            affiche(S2, error_Map)
            affiche(S1, error_Markov)
```