Les codes correcteurs d'erreurs

Rocchia Sylvain

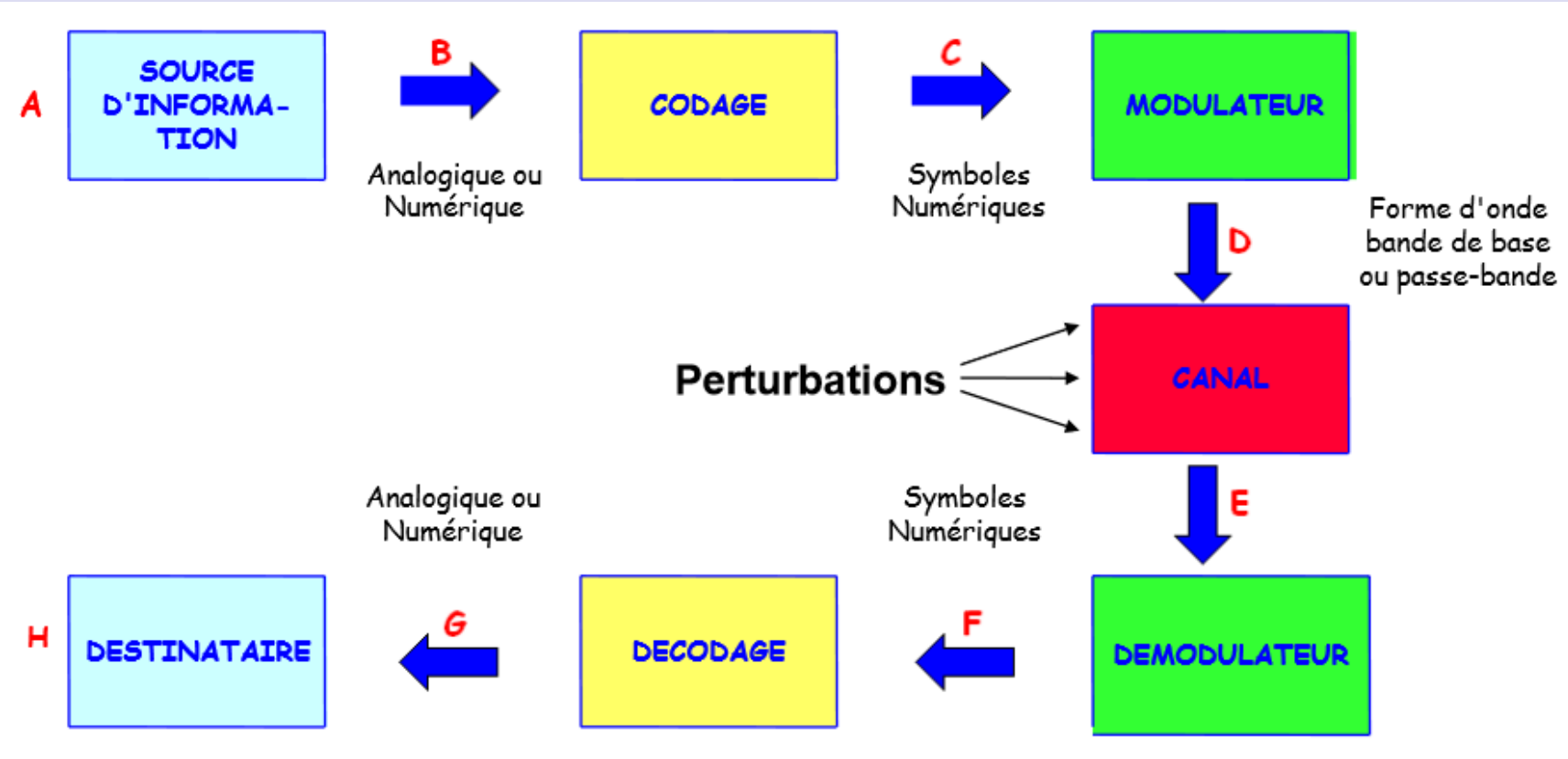
Numéro de candidat : 51610

21 Septembre 2020

Sommaire

Introduction

Principe



Introduction

De nombreux domaines d'application

- Présents dans les CD et DVD
- Utiles dans la transmission d'informations en général

Problématique :

Comment fonctionnent-ils et comment les implémenter ? Quelles sont leurs limites ?

Introduction

Différents types de codes correcteurs d'erreurs

- Les codes en blocs linéaires
  - Complexité en temps polynomial à la longueur du bloc à coder/décoder
  - Basés sur des structures d'espaces vectoriels, de corps finis et de polynômes
- Les codes convolutifs
  - Complexité en temps exponentiel

Modélisation et premières constatations

- Passage d'un texte et/ou d'une image, dans un canal ayant une probabilité p de changer la valeur d'un bit lors de la transmission
- Un caractère = un octet tandis qu'un pixel = 3 octets (RGB)
- On étudiera principalement le code de Hamming (7,4) et on expliquera le fonctionnement du code de Reed-Solomon RS(255,239)

Modélisation et premières constatations

```
>>> textetobin("deux")
[[0, 1, 1, 0, 0, 1, 0, 0], [0, 1, 1, 0, 0, 1, 0, 1], [0, 1, 1, 1, 0, 1, 0, 1]
, [0, 1, 1, 1, 1, 0, 0, 0]]

>>> ord("d")
100

>>> dectobin(100)
[0, 1, 1, 0, 0, 1, 0, 0]
```

Figure: Lettre → Octet

```
>>> generr2([0,1,1,0,0,1,0,0])
[1, 1, 1, 0, 0, 1, 0, 0]
```

```
>>> generr2([0,1,1,0,0,1,0,0])
[0, 0, 1, 0, 0, 1, 1, 0]
```

Figure: Génération d'erreurs

Le code de Hamming (7,4)

Explication et modélisation

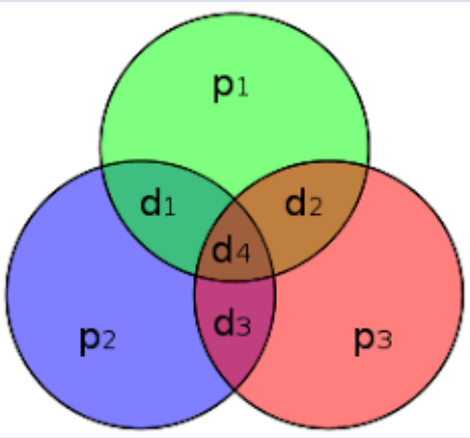


Figure: Bits du mot encodé

```
>>> decomp([0,1,1,0,1,0,0,1])
[[0, 1, 1, 0], [1, 0, 0, 1]]
```

Figure: Octet → Deux codes de longueur 4

```
>>> Hamming74([0,1,0,0])
[1, 0, 0, 1, 1, 0, 0]
```

Figure: Encodage

Le code de Hamming (7,4)

Explication et modélisation

```
>>> generr2([1, 0, 0, 1, 1, 0, 0])
[0, 0, 0, 1, 1, 0, 1]

>>> detec([0, 0, 0, 1, 1, 0, 1])
('Erreur sur le bit', 6, [0, 0, 0, 1, 1, 1, 1])

>>> detec([0, 0, 0, 1, 1, 0, 0])
('Erreur sur le bit', 1, [1, 0, 0, 1, 1, 0, 0])
```

Figure: Détection et correction

Le code de Hamming (7,4)

Résultats et observations : p = 0.05

```
>>> t
"Ceci est un texte de test, il a été écrit à des fins s
cientifiques et expérimentales et est cependant sujet à
erreur, mais l'algorithme utilisé sur lui est censé le
renvoyer identique à l'original. Sa taille est fixe au
cours du programme."
```

Figure: Texte transmis, l = 239

Le code de Hamming (7,4)

Résultats et observations : p = 0.05

```
>>> ter
'Ceci"est un\xa0texôe fe decv,!il A ù|á ècsyt à`dås fi*
s"scieouiNaeues mt e|përame~talqs et es4\x00gepanda.ô$r
Ujed!à ebrdur, íaks l\'alg/pithme"}tédyqé òer lui\x02es
T #i~sé le renæOydp é$ajtiaum à l\'orégi~ah. Óa!ôaille
ect bixe"a5 cours"$q$Prograkmå,'
```

Figure: Texte avec erreurs, pourcentage d'erreur : 36.8%

Le code de Hamming (7,4)

Résultats et observations : p = 0.05

```
>>> tc
'Ceci est un texte de test, il a étá écRit à des fins"s
cienqif`ques et e|périmentales et es$ cependant sujet#à
errlur, ýais l\'algorithme utidisé sur lui est cinsé le
renäoyer identiaue à l\'original. Sa taille est `ixe au
cours du progracmå.'
```

Figure: Texte corrigé, pourcentage d'erreur : 7.1%

Le code de Hamming (7,4)

Résultats et observations : p = 0.05

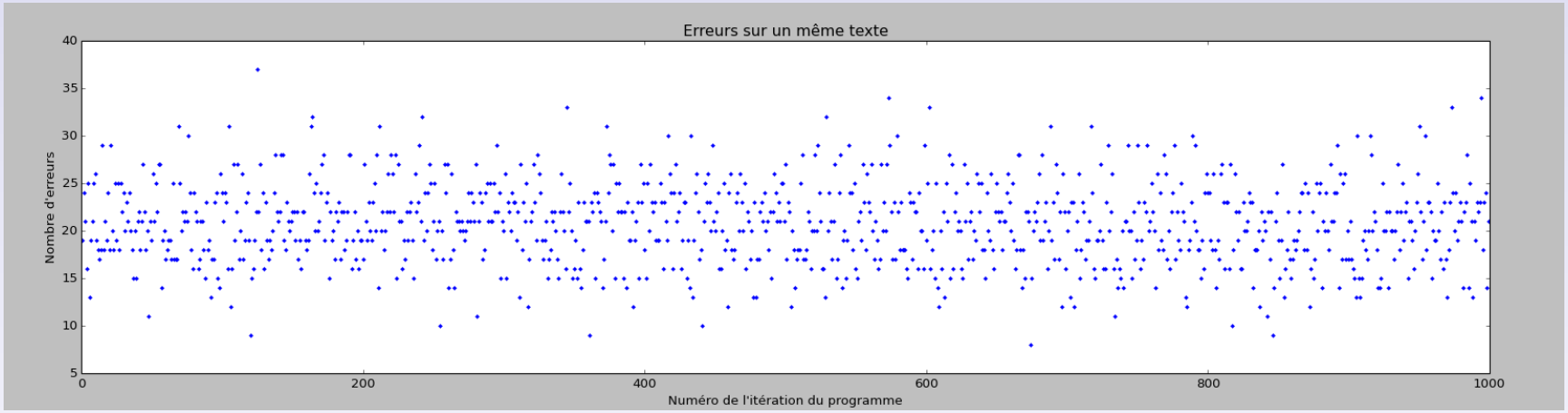


Figure: Nombre d'erreurs restantes après la correction pour un même texte de longueur l, en moyenne 8.7% (pour 33.6% d'erreurs avant)

Le code de Hamming (7,4)

Résultats et observations : p = 0.05

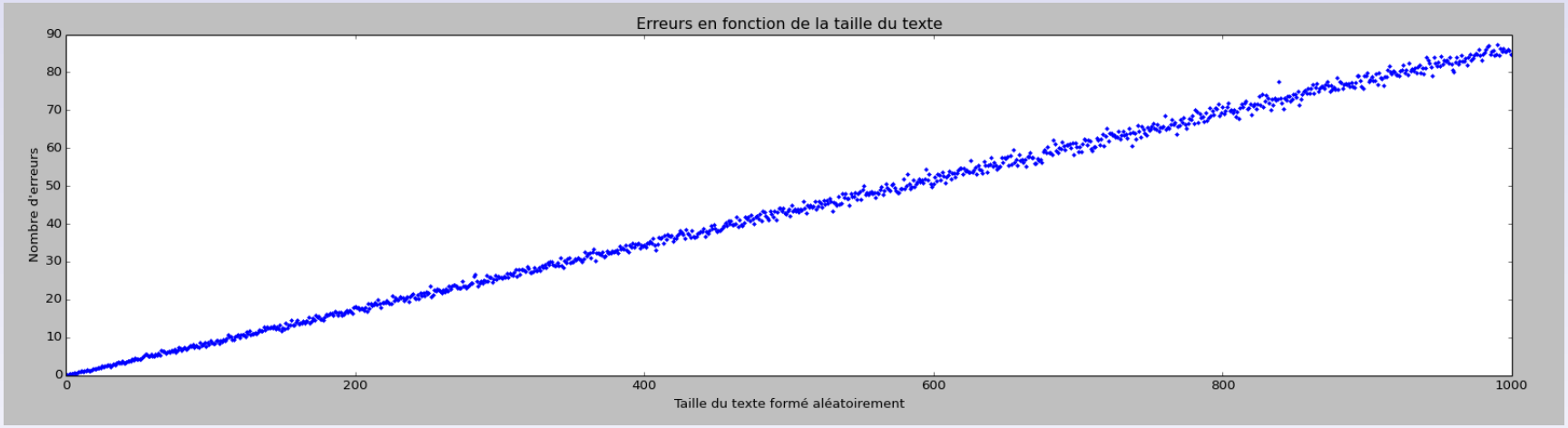


Figure: Moyenne du nombre d'erreurs restantes après le programme pour 50 textes aléatoires de taille en abscisse

Le code de Hamming (7,4)

Résultats et observations : p = 0.05

Figure: Image transmise

Le code de Hamming (7,4)

Résultats et observations : p = 0.05

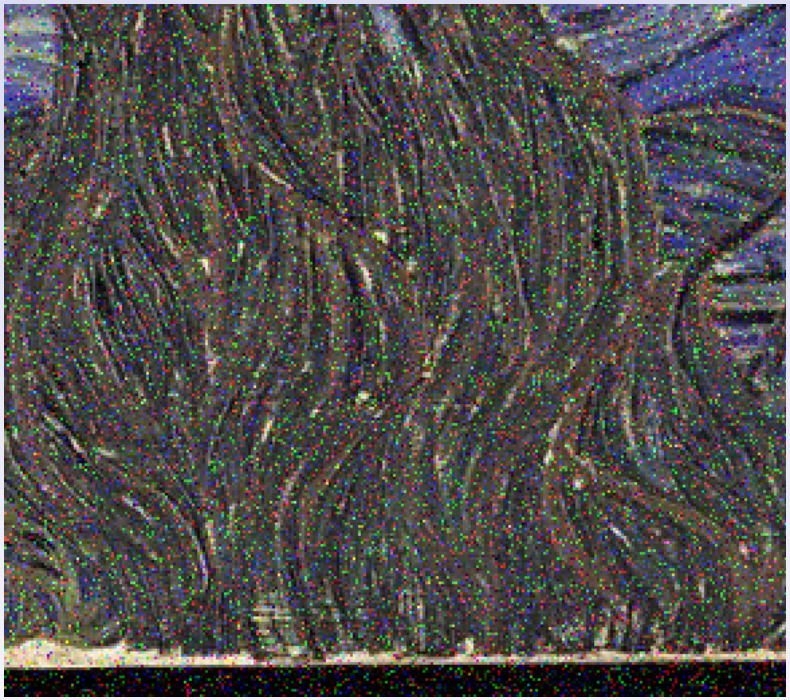


Figure: Image avec erreurs

Figure: Image corrigée

Le code de Hamming (7,4)

Résultats et observations : p = 0.08

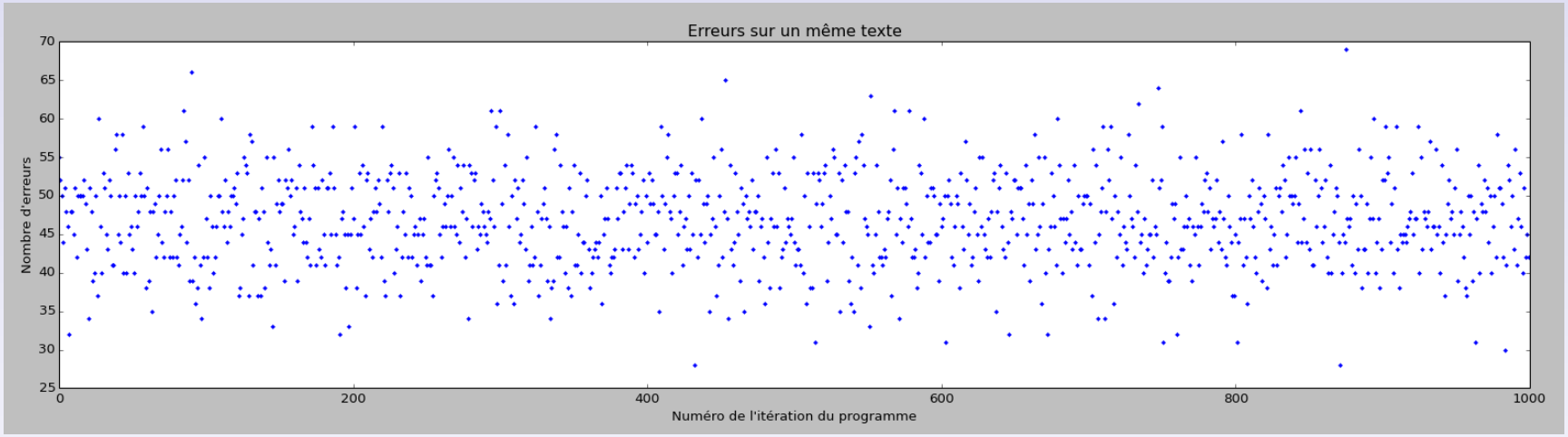


Figure: Nombre d'erreurs restantes après la correction pour un même texte de longueur l, en moyenne 19.5% (pour 48.5% d'erreurs avant)

Le code de Hamming (7,4)

Résultats et observations : p = 0.08

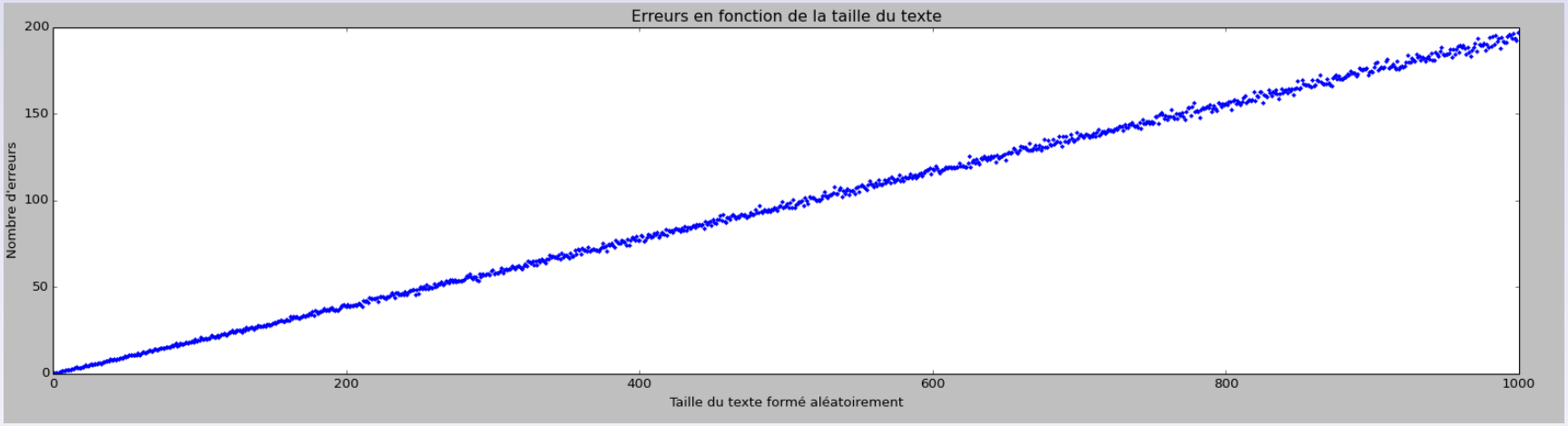


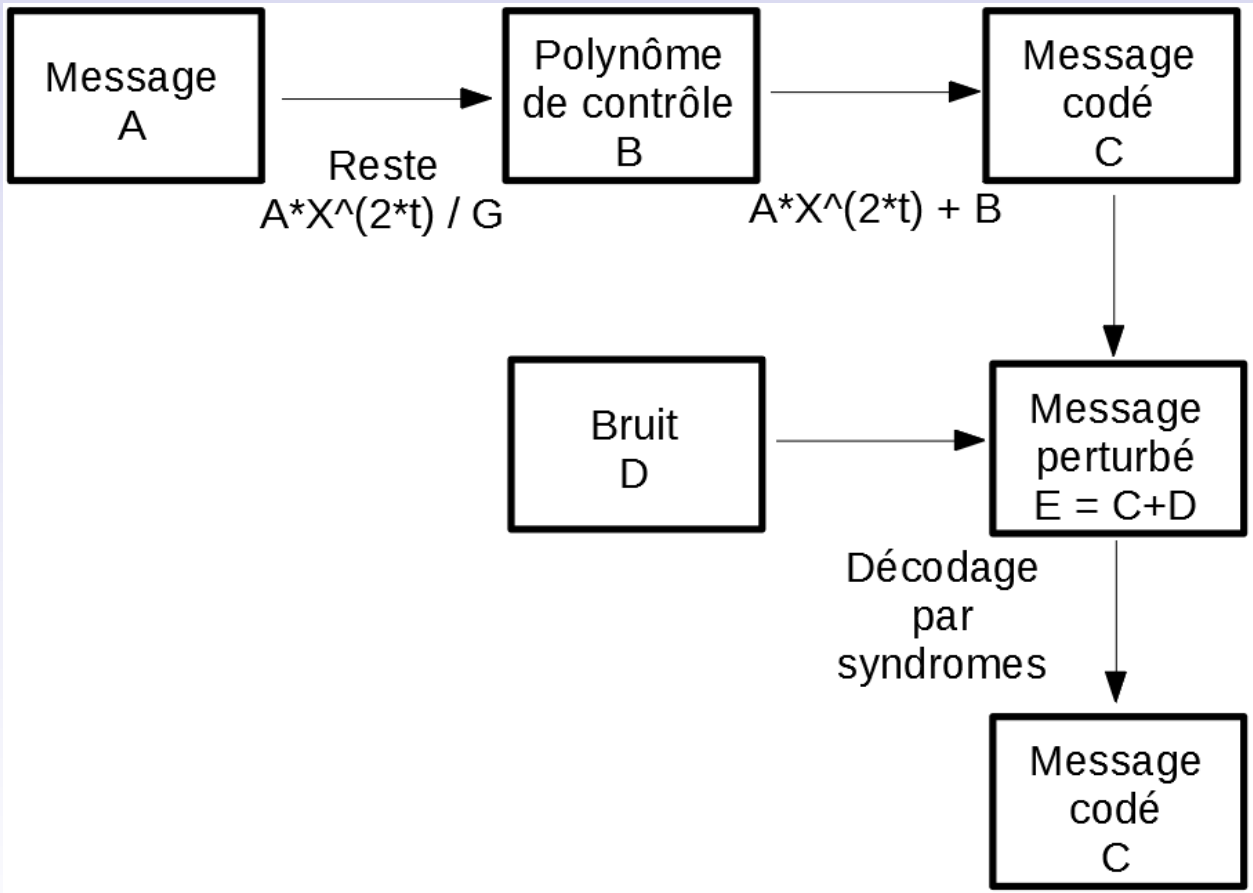
Figure: Moyenne du nombre d'erreurs restantes après le programme pour 50 textes aléatoires de taille en abscisse

Conclusion partielle

- Diminution satisfaisante du nombre d'erreurs
- Manque de rapidité pour une exécution à grande échelle
- Correction fausse pour 2 erreurs ou plus sur un même bloc, pas de correction si un terme est supprimé

Le code de Reed-Solomon RS(255,239)

Explication et modélisation



Le code de Reed-Solomon RS(255,239)

Explication et modélisation

```
>>> coder.encode([ord(x) for x in t])
'Ceci est un texte de test, il a été écrit à des fins scientifique
s et expérimentales et est cependant sujet à erreur, mais l\'algor
ithme utilisé sur lui est censé le renvoyer identique à l\'origina
l. Sa taille est fixe au cours du programme.H\x14þ$Ar´n".\x12ÙýS`\
x86'
```

Figure: Message encodé

Le code de Reed-Solomon RS(255,239)

Explication et modélisation

```
>>> coder.decode('\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0e de test, il a
été écrit à des fins scientifiques et expérimentales et est cepend
ant sujet à erreur, mais l\'algorithme utilisé sur lui est censé l
e renvoyer identique à l\'original. Sa taille est fixe au cours du
programme.H\x14þ$Ar´n".\x12ÙýS`\x86', erasures_pos = [0,1,2,3,4,5,
6,7,8,9,10,11,12,13,14,15], only_erasures = True)
("Ceci est un texte de test, il a été écrit à des fins scientifiqu
es et expérimentales et est cependant sujet à erreur, mais l'algor
ithme utilisé sur lui est censé le renvoyer identique à l'original
. Sa taille est fixe au cours du programme.", 'H\x14þ$Ar´n".\x12Ùý
S`\x86')
```

Figure: Correction d'effacements (jusque 16)

Le code de Reed-Solomon RS(255,239)

Explication et modélisation

```
>>> coder.decode('\0\0\0\0\0\0\0 un texte de test, il a été écri
t à des fins scientifiques et expérimentales et est cependant suje
t à erreur, mais l\'algorithme utilisé sur lui est censé le renvoy
er identique à l\'original. Sa taille est fixe au cours du program
me.H\x14þ$Ar´n".\x12ÙýS`\x86')
("Ceci est un texte de test, il a été écrit à des fins scientifiqu
es et expérimentales et est cependant sujet à erreur, mais l'algor
ithme utilisé sur lui est censé le renvoyer identique à l'original
. Sa taille est fixe au cours du programme.", 'H\x14þ$Ar´n".\x12Ùý
S`\x86')
```

Figure: Correction d'erreurs (jusque 8)

Le code de Reed-Solomon RS(255,239)

Résultats et observations

- Correction totale de toutes les erreurs sous certaines conditions
- Limite de bits corrigés faible
- Correction des salves d'erreurs et des suppressions

Conclusions

- Correction non négligeable
- Limites : complexité, rapport informations/redondance
- Pistes : turbo-codes, codes BCH et CRC

Annexe

```
## Constantes - Importations
from copy import *
import time as tm
from random import randint
import math as m
import matplotlib.pyplot as plt
import numpy as np

N = 1000
p = 0.05
alphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZéèûîôâêùàäëüïöÿ' -`#~_|([(?,.;:/!§^¨%*£
¤$=+}°)]@ç{&²*µ"
t = "Ceci est un texte de test, il a été écrit à des fins scientifiques et expérimentales et est
cependant sujet à erreur, mais l'algorithme utilisé sur lui est censé le renvoyer identique à
l'original. Sa taille est fixe au cours du programme."

## Fonctions

def genchar():
    d = randint(0,255)
    return chr(d)

def gentxt(n):
    tex = ""
    for k in range(n):
        o = genchar()
        tex = tex + o
    return tex

def genmot(n):
    l = []
    for i in range(n):
        l.append(randint(0, 1))
    return l
```

Annexe

```python
def somme(liste):
    n = len(liste)
    s = 0
    for i in range(n):
        s += liste[i]
    return s

def changer(liste, x):
    if liste[x] == 0:
        liste[x] = 1
        return liste
    else:
        liste[x] = 0
        return liste

def parite(x):
    if x % 2 == 0:
        return 0
    else:
        return 1

def echange(x,y, mot):
    l = [mot[x], mot[y]]
    mot[x] = l[1]
    mot[y] = l[0]
    return mot
```

Annexe

```python
def Hamming74(mot):
    par1 = parite(mot[0])
    par2 = parite(mot[1])
    par3 = parite(mot[2])
    par4 = parite(mot[3])
    p1 = parite(par1+par2+par4)
    p2 = parite(par1+par3+par4)
    p3 = parite(par2+par3+par4)
    mot.insert(0, p1)
    mot.insert(1, p2)
    mot.insert(3, p3)
    return mot

def maxi(L):
    max = L[0]
    for i in range(len(L)):
        if max <= L[i]:
            max = L[i]
        else:
            max = max
    return max

def mini(L):
    min = L[0]
    for i in range(len(L)):
        if min >= L[i]:
            min = L[i]
        else:
            min = min
    return min

def moy(L):
    s = somme(L)
    if s == 0:
        return 0
    return s/len(L)
```

Annexe

```
def generr1(mot):
    max = maxi(mot)
    min = mini(mot)
    if max == min:
        max, min = 1, 0
    p2 = m.floor(100*p)
    k = randint(1,100)
    i = randint(0, len(mot)-1)
    if k <= p2:
        new = randint(min, max)
        while new == mot[i]:
            new = randint(min, max)
        mot[i] = new
    return mot

def generr2(mot):
    max = maxi(mot)
    min = mini(mot)
    if max == min:
        max, min = 1, 0
    p2 = m.floor(100*p)
    for i in range(len(mot)):
        k = randint(1,100)
        if k <= p2:
            new = randint(min, max)
            while new == mot[i]:
                new = (randint(min, max))
            mot[i] = new
    return mot
```

Annexe

```
def generr3(mot):
    max = maxi(mot)
    min = mini(mot)
    if max == min:
        max, min = 1, 0
    p2 = m.floor(100*p)
    for i in range(len(mot)):
        k = randint(1,100)
        if k <= p2:
            mot = changer(mot, i)
    return mot

def poids(mot):
    po = 0
    for i in range(len(mot)):
        po = po + mot[i]
    return po

def detecpoids(mot, p2):
    s = poids(mot)
    if s == p2:
        print("Aucune erreur n'a été détectée")
    else:
        print("Une erreur a été détectée")

def reconvHam(mot):
    l = []
    l.append(mot[2])
    l.append(mot[4])
    l.append(mot[5])
    l.append(mot[6])
    return l
```

Annexe

```python
def decomp(mot):
    n = len(mot)
    p = n//4
    K = []
    c = 0
    while c != p:
        l = []
        for i in range(4):
            l.append(mot[c*4 + i])
        K.append(l)
        c += 1
    return K

def recon(mot):
    n = len(mot)
    m = []
    for p in range(n):
        k = len(mot[p])
        for i in range(k):
            m.append(mot[p][i])
    return m

def bintodec(liste):
    n = len(liste)
    d = 0
    for i in range(n):
        d += liste[n-1-i]*(2**i)
    return d

def dectobin(float):
    f = int(float)
    b = []
    for i in range(8):
        if f - 2**(7-i) >= 0:
            b.append(1)
            f = f - 2**(7-i)
        else:
            b.append(0)
    return b
```

Annexe

```
def detec(mot):
    c1 = mot[0] + mot[2] + mot[6] + mot[4]
    c2 = mot[1] + mot[2] + mot[6] + mot[5]
    c4 = mot[3] + mot[4] + mot[5] + mot[6]
    cc1 = parite(c1)
    cc2 = parite(c2)
    cc4 = parite(c4)
    pos = bintodec([cc4, cc2, cc1])
    if pos == 0:
        return "Pas d'erreur", pos, mot
    else:
        return "Erreur sur le bit", pos, changer(mot, pos-1)

def indicezero(liste):
    n = len(liste)
    ind = []
    for i in range(n):
        if liste[i] == 0:
            ind.append(i)
    return ind

def textetobin(texte):
    b = []
    for char in texte:
        b.append(dectobin(ord(char)))
    return b

def btt(liste):
    t = ""
    for i in range(len(liste)):
            t = t + chr(bintodec(liste[i]))
    return t
```

Annexe

```
def fusion(l1, l2):
    l = []
    for i in range(len(l1)):
        l.append(l1[i])
    for i in range(len(l2)):
        l.append(l2[i])
    return l

def nberr(l1, l2):
    if len(l1) == len(l2):
        e = 0
        for i in range(len(l1)):
            if l1[i] != l2[i]:
                e = e + 1
        return e
    else:
        return "Entrez des listes de tailles identiques"
```

Annexe

```
def testplot(n):
    nb = []
    for j in range(n):
        b = textetobin(t)
        bsep = []
        for i in range(len(b)):
            bsep.append(decomp(b[i]))
        bham = []
        for i in range(len(bsep)):
            bham.append(Hamming74(bsep[i][0]))
            bham.append(Hamming74(bsep[i][1]))
        berr = []
        for i in range(len(bham)):
            berr.append(generr2(bham[i]))
        bcorr = []
        for i in range(len(berr)):
            bcorr.append(detec(berr[i])[2])
        bcrec = []
        for i in range(len(bcorr)):
            bcrec.append(reconvHam(bcorr[i]))
        bcfus = []
        for k in range(len(bcrec)//2):
            bcfus.append(fusion(bcrec[2*k],bcrec[2*k+1]))
        tc = btt(bcfus)
        nb.append(nberr(t, tc))
    return nb
```

Annexe

```
def plottaille(N):
    nb = []
    for j in range(N):
        to = gentxt(j)
        b = textetobin(to)
        bsep = []
        for i in range(len(b)):
            bsep.append(decomp(b[i]))
        bham = []
        for i in range(len(bsep)):
            bham.append(Hamming74(bsep[i][0]))
            bham.append(Hamming74(bsep[i][1]))
        berr = []
        for i in range(len(bham)):
            berr.append(generr2(bham[i]))
        bcorr = []
        for i in range(len(berr)):
            bcorr.append(detec(berr[i])[2])
        bcrec = []
        for i in range(len(bcorr)):
            bcrec.append(reconvHam(bcorr[i]))
        bcfus = []
        for k in range(len(bcrec)//2):
            bcfus.append(fusion(bcrec[2*k],bcrec[2*k+1]))
        tco = btt(bcfus)
        nb.append(nberr(to, tco))
    return nb
```

Annexe

```
def testerr(n):
    nb = []
    for j in range(n):
        b = textetobin(t)
        bsep = []
        for i in range(len(b)):
            bsep.append(decomp(b[i]))
        bham = []
        for i in range(len(bsep)):
            bham.append(Hamming74(bsep[i][0]))
            bham.append(Hamming74(bsep[i][1]))
        berr = []
        for i in range(len(bham)):
            berr.append(generr2(bham[i]))
        terr = []
        for i in range(len(berr)):
            terr.append(reconvHam(berr[i]))
        bcerr = []
        for k in range(len(terr)//2):
            bcerr.append(fusion(terr[2*k],terr[2*k+1]))
        tcerr = btt(bcerr)
        nb.append(nberr(t, tcerr))
    return [nb, moy(nb)/len(t)]


def Tmoy(k):
    A = []
    for i in range(k):
        print(i)
        A.append(np.array(plottaille(N)))
    B = np.array([0]*len(A[1]))
    for i in range(len(A)):
        B = B + A[i]
    return list(np.array(B)/k)
```

Annexe

```
def testgen(n):
    k = 0
    for j in range(n):
        to = gentxt(1)
        b = textetobin(to)[0]
        dec = decomp(b)
        ham = [Hamming74(dec[0]),Hamming74(dec[1])]
        err = [generr2(ham[0]),generr2(ham[1])]
        err2 = [reconvHam(err[0]), reconvHam(err[1])]
        derr = bintodec(fusion(err2[0],err2[1]))
        cher = chr(derr)
        if cher != to:
            k += 1
    return k
```

Annexe

```
## Main


b = textetobin(t)
bsep = []
for i in range(len(b)):
    bsep.append(decomp(b[i]))
bham = []
for i in range(len(bsep)):
    bham.append(Hamming74(bsep[i][0]))
    bham.append(Hamming74(bsep[i][1]))
berr = []
for i in range(len(bham)):
    berr.append(generr2(bham[i]))
terr = []
for i in range(len(berr)):
    terr.append(reconvHam(berr[i]))
bcerr = []
for k in range(len(terr)//2):
    bcerr.append(fusion(terr[2*k],terr[2*k+1]))
bcorr = []
for i in range(len(berr)):
    bcorr.append(detec(berr[i])[2])
bcrec = []
for i in range(len(bcorr)):
    bcrec.append(reconvHam(bcorr[i]))
bcfus = []
for k in range(len(bcrec)//2):
    bcfus.append(fusion(bcrec[2*k],bcrec[2*k+1]))
tc = btt(bcfus)
ter = btt(bcerr)

nerr = nberr(t,ter)
perc = nerr/len(t)

nerrc = nberr(t,tc)
percc = nerrc/len(t)
```

Annexe

```
Y = testplot(N)
T = Tmoy(50)
X = [i for i in range(N)]
plt.subplot(2,1,1)
plt.plot(X, Y, ".")
plt.title("Erreurs sur un même texte")
plt.xlabel("Numéro de l'itération du programme")
plt.ylabel("Nombre d'erreurs")
plt.subplot(2,1,2)
plt.plot(X, T, ".")
plt.title("Erreurs en fonction de la taille du texte")
plt.xlabel("Taille du texte formé aléatoirement")
plt.ylabel("Nombre d'erreurs")
plt.show()

moyappc = moy(Y) #moyenne d'erreurs par texte après correction
percappc = moyappc/len(t)
moyavc = moy(testerr(N)[0]) #moyenne d'erreurs par texte avant correction
percavc = moyavc/len(t)
```

Annexe

```python
import unireedsolomon as rs
from PIL import Image

coder = rs.RSCoder(255,239)
t = "Ceci est un texte de test, il a été écrit à des fins scientifiques et expérimentales et est
cependant sujet à erreur, mais l'algorithme utilisé sur lui est censé le renvoyer identique à
l'original. Sa taille est fixe au cours du programme."
e = coder.encode([ord(x) for x in t])
r = "\0"*8 + e[8:]
d = coder.decode(r)
print(d)
```