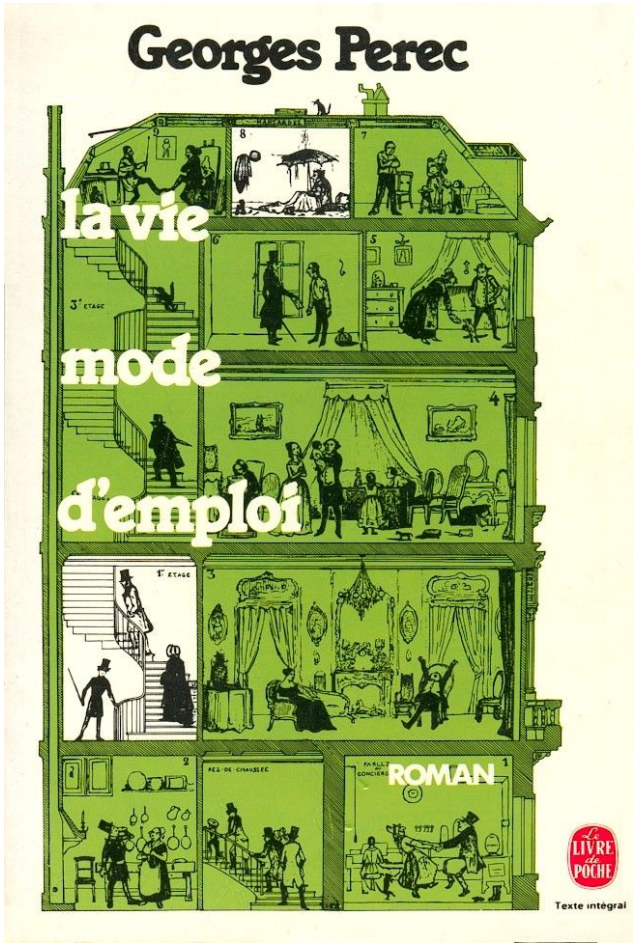
Remedium Codes

Codes correcteurs avec des carrés latins

La vie : mode d'emploi

Roman de **Georges Perec** publié en **1978**
Prix Médicis 1978

Georges Perec
(1936 - 1982)



Qu'est-ce qu'un carré latin ?

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

Groupe de Klein

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{bmatrix}$$

Groupe cyclique

Carrés latins orthogonaux d'ordre 4

Les carrés latins à travers les âges

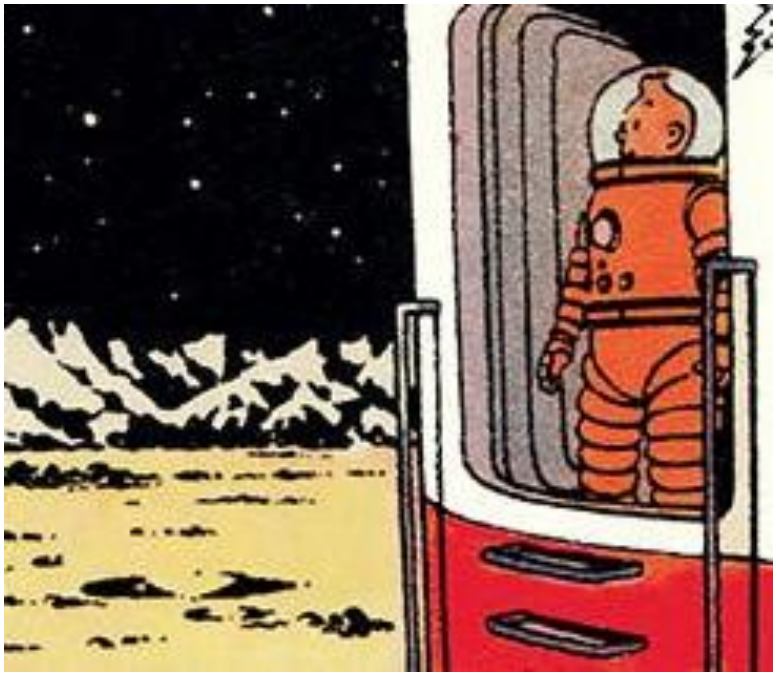
Leonhard Euler
(1707-1783)

Gaston Tarry
(1843 - 1913)

S. S. Shrikhande
(1917 - 2020)

Présentation du problème

Correction de données endommagées lors
d'un voyage spatial



Variables de l'algorithme

On entre un code de taille m^2.

h est le nombre de carrés latins orthogonaux de taille m :

$$h = min(p_i^{e_i} - 1)$$

Le code correcteur peut corriger t erreurs.

$$t \leq \frac{h}{2} + 1,\ t \in \mathbb{N}^*$$

Construction de la matrice de contrôle

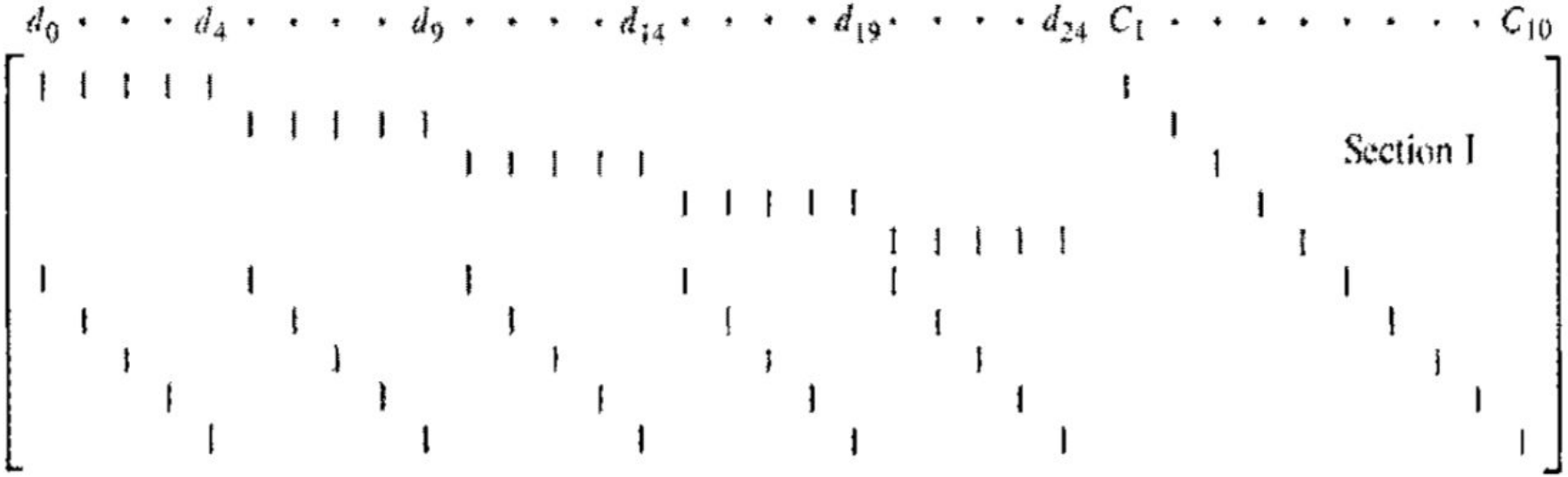
$$H = \begin{pmatrix} M_1 & \\ M_2 & \\ M_3 & I2tm \\ \dots & \\ M_{2t} & \end{pmatrix}$$

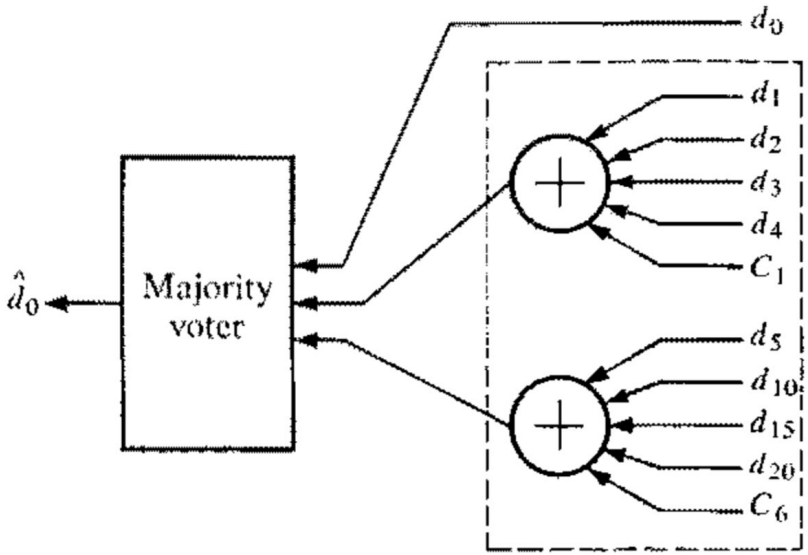
$$L = [l_{i,j}]m \times m$$

$$V_k = [\delta_{l_{ij},k}]$$

$$M_i = \begin{pmatrix} V_1 \\ V_2 \\ \dots \\ V_m \end{pmatrix}$$

Structure de la matrice





Aspect théorique du code correcteur

Le code correcteur peut corriger **t** erreurs parmi $\mathbf{m^2}$ bits.

Exemple pour ***m = 3*** *et* ***t = 2*** *:*

A_1 = [0, 1, 0, 1, 1, 1, 1, 0, 0]

de longueur $\mathbf{m^2 = 9}$

A_2 = [0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1]

de longueur $\mathbf{m^2+2*m*t = 21}$

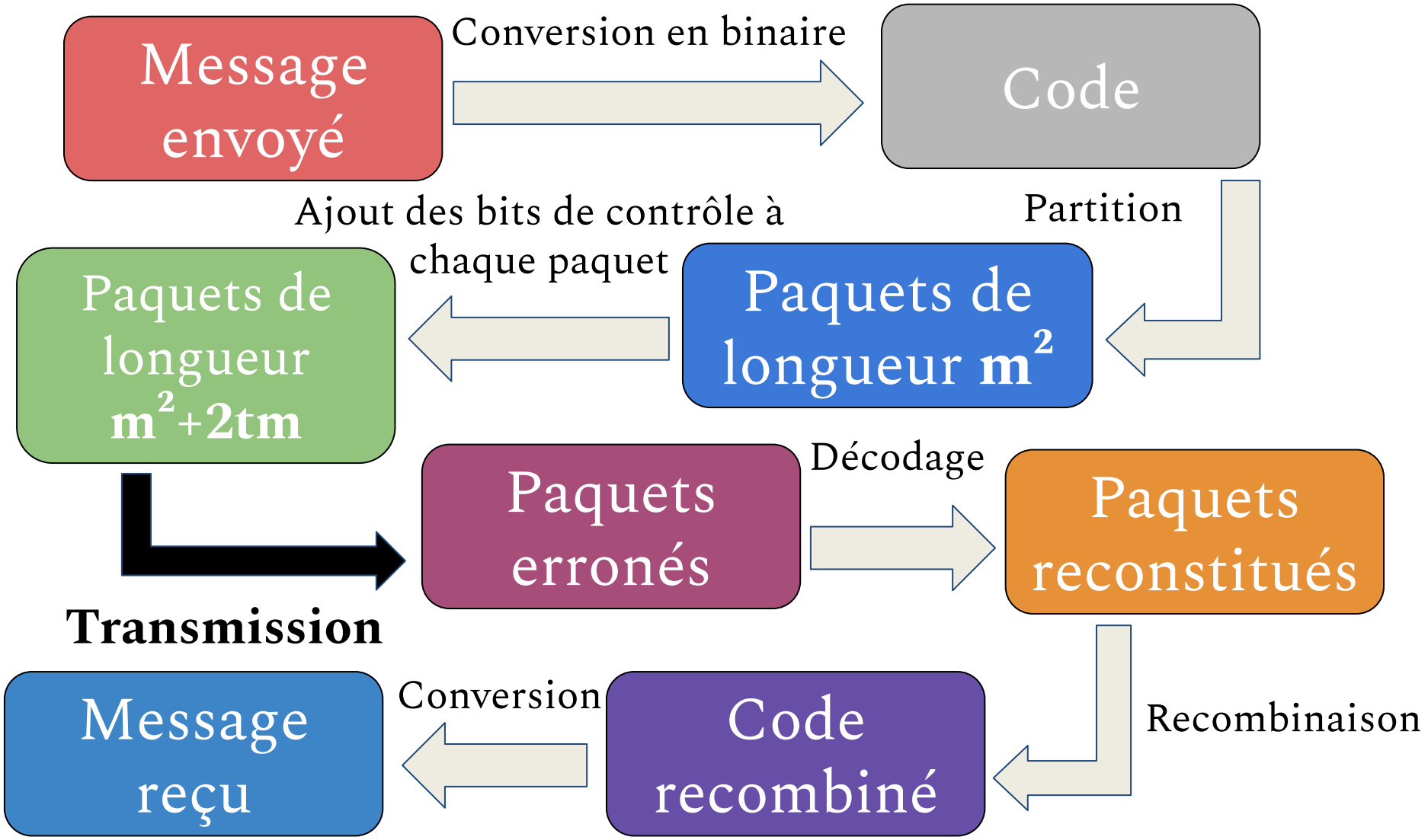
Transmissions : deux erreurs aléatoires

A_3 = [0, 1, 0, 1, 1, **0**, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, **1**, 1, 1, 1]

Décodage :

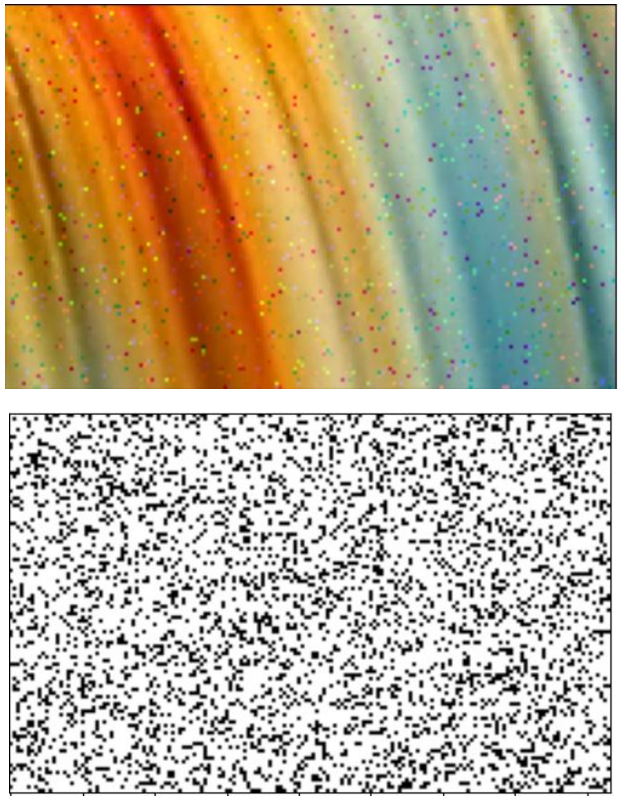
A_4 = [0, 1, 0, 1, 1, 1, 1, 0, 0] Ouf !

Structure de l'algorithme

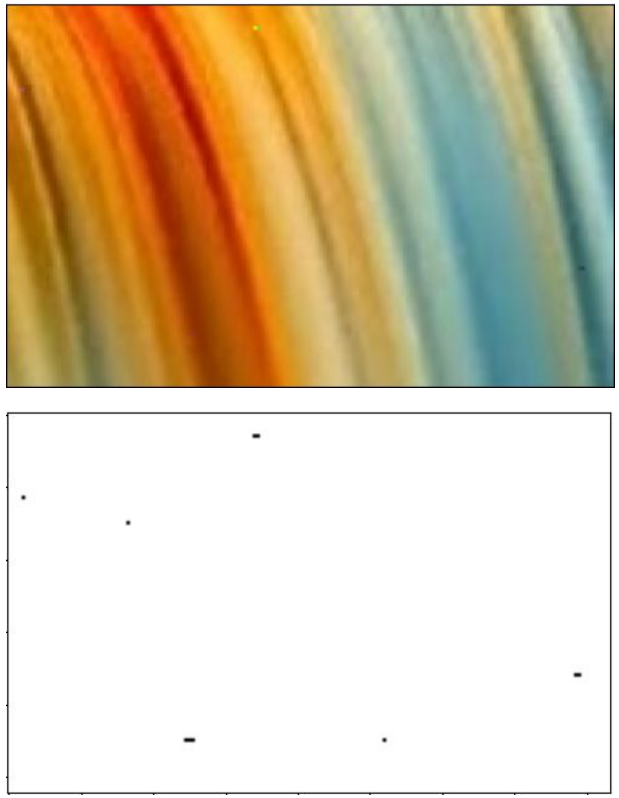


Application : transmission d'une image

Transmission d'une image avec et sans contrôle
Probabilité d'erreur de transmission : **1 %**



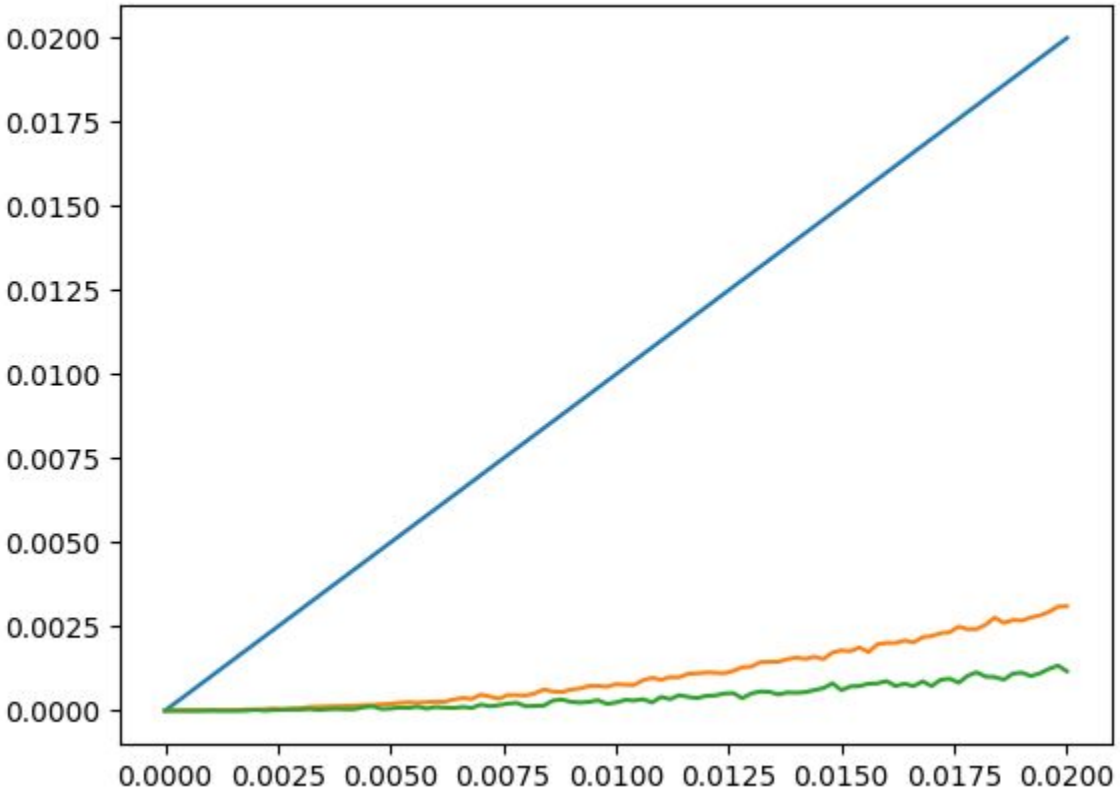
Sans contrôle



Avec contrôle *(m=7, t=4)*

Correction pour des faibles taux d'erreurs

Part d'erreurs lors de la réception



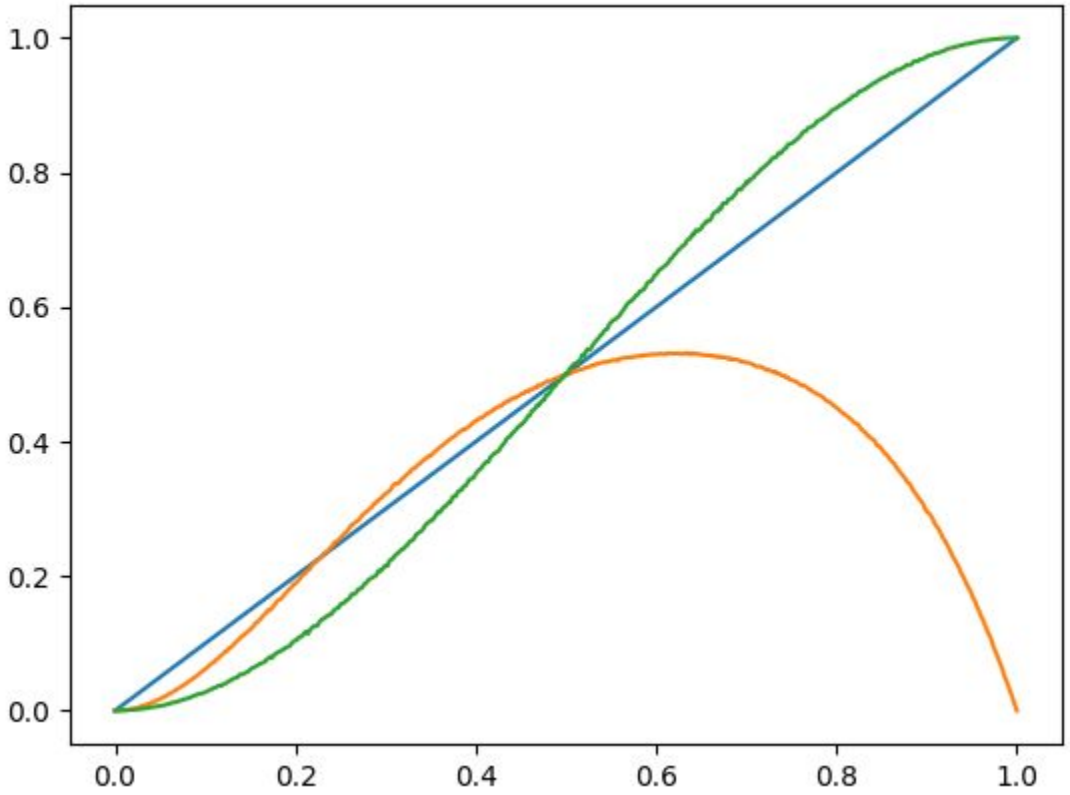
Part d'erreurs lors de la transmission

Fonctions de correction :

- Sans contrôle
- Contrôle avec les carrés latins (m=2, t=1)
- Trois transmissions successives

Correction pour toutes les taux d'erreurs

Part d'erreurs lors de la réception



Part d'erreurs lors de la transmission

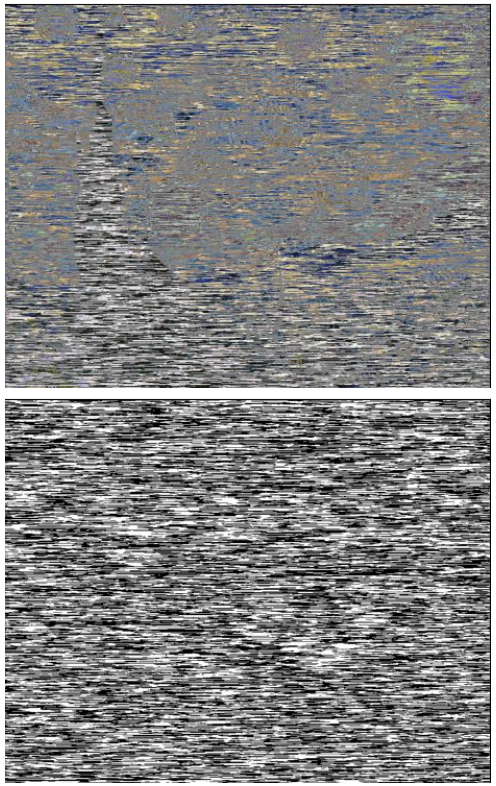
Fonctions de correction :

- Sans contrôle
- Contrôle avec les carrés latins (m=2, t=1)
- Trois transmissions successives

Transmission d'une image, erreurs en rafale

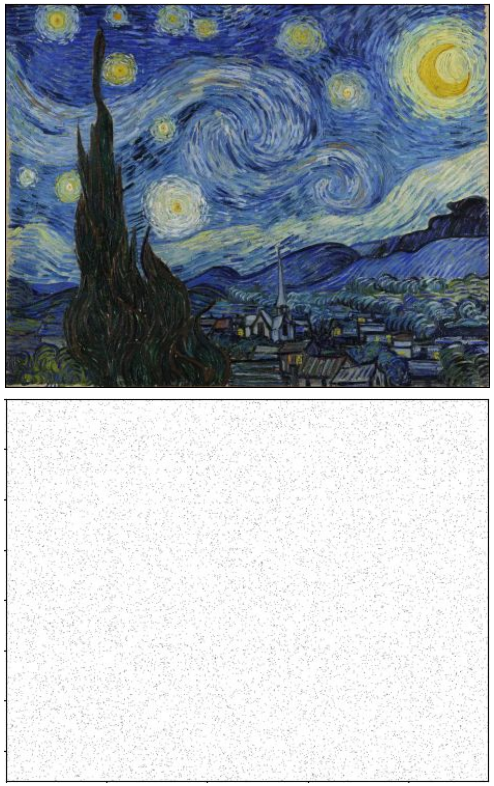
Transmission d'une image avec et sans contrôle
Erreurs en rafale

Image envoyée



Sans contrôle

Pourcentage d'erreurs : **49,7 %**



Avec contrôle

Pourcentage d'erreurs : **0,2 %**

Annexe : Structure du programme

Générations des carrés latins orthogonaux d'ordre **m** :

- `fact_premiere(m)`
- `nb_carres_orth(m)`
- `crea_tous_carres_orth(m)`

Construction de la matrice de contrôle :

- `matM1(m)`
- `matM2(m)`
- `vectVmu(mu,m,L)`
- `matMi(i,m,LAT)`
- `matH(m,t)`

Codage :

- `bits_de_controle(L,m,t)`
- `ajout_bits_de_controle(L,m,t)`

Transmission :

- `transmi_proba(L,p)`
- `transmi_rafale(L,d_min,d_max,n)`
- `trois_transmissions(L,p)`

Décodage :

- `decode(L,m,t)`
- `recombinateur(L,m,t)`

Fonctions auxiliaires :

- `partition(L,dis)`
- `xor(L)`
- `vote(L)`

Traitement de l'image :

- `conv_binaire(n)`
- `convertisseur_jpeg_vers_binaire(I)`
- `liste4_a_1(L)`
- `liste1_a_4(L,n,m)`
- `conv_base_10(N)`
- `convertisseur_binaire_vers_jpeg(L)`

Les graphiques ont été tracés grâce au module `matplotlib.pyplot`

```
def fact_premiere(n): #Renvoie la liste des facteurs premiers de n pour n<200 avec leur multiplicité
    PRE=[2,3,5,7,9,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,
    101,103,107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,19
    7,199]
    P=[]
    for i in PRE:
        while n%i==0:
            P.append(i)
            n=n//i
    m=len(P)
    L=[[P[0],1]]
    for j in range(1,m):
        if P[j]!=P[j-1]:
            L.append([P[j],0])
    c=0
    for k in range(1,m):
        if P[k]!=P[k-1]:
            c+=1
        L[c][1]+=1
    return L

def nb_carres_orth(n): # min(pi**ei - 1)
    P=fact_premiere(n)
    m=P[0][0]**P[0][1] - 1
    l=len(P)
    for i in range(l):
        if P[i][0]**P[i][1] - 1 < m:
            m=P[i][0]**P[i][1] - 1
    return m
```

```
def crea_tous_carres_orth(n):
    h=nb_carres_orth(n)
    LAT=[]
    for j in range(1,h+1):
        #Creation des h carres
        #Parcours vertical puis horizontal avec à
chaque fois application des actions sur les lignes
ou les colonnes
        L=[]
        for x in range(n):
            L.append([])
            for y in range(n):
                L[x].append((j*x)%n)
        for r in range(n):
            for s in range(n):
                L[s][r]=((L[s][r]+r)%n)
        LAT.append(L)
    return LAT
```

```
def matM1(m): #crée la matrice M1
    M=[]
    for i in range(m):
        M.append([])
        for j in range(m*m):
            if j>=m*i and j<m*(i+1):
                M[i].append(1)
            else:
                M[i].append(0)
    return M

def matM2(m): #crée la matrice M2
    M=[]
    for i in range(m):
        M.append([])
        for j in range(m*m):
            if j%m == i:
                M[i].append(1)
            else:
                M[i].append(0)
    return M

def vectVmu(mu,m,L):
    V=[]
    for j in range(m):
        for k in range(m):
            if L[j][k]==mu:
                V.append(1)
            else:
                V.append(0)
    return V
```

```
def matMi(i,m,LAT):
    L=LAT[i-3]
    Mi=[]
    for mu in range(m):
        Mi.append(vectVmu(mu,m,L))
    return Mi



def matH(m,t): #crée H
    H=[]#listes internes : lignes
    M1=matM1(m)
    M2=matM2(m)
    H=M1+M2
    LAT=crea_tous_carres_orth(m)
    for i in range(3,2*t+1):
        M=matMi(i,m,LAT)
        H=H+M
    for j in range(2*t*m):
        for k in range(2*t*m):
            if k==j:
                H[j].append(1)
            else:
                H[j].append(0)
    return H
```

```
def bits_de_controle(d,m,t):
    if len(d)!=m**2:
        return "Mauvaise taille de liste"
    H=matH(m,t)
    for i in range(2*t*m):
        L=[]
        for j in range(m**2):
            if H[i][j]==1: L.append(d[j])
        d.append(xor(L))
    return d

def ajout_bits_de_controle(L,m,t): #Prend une liste de bits et y
ajoute les bits de contrôle
    P=partition(L,m**2)
    N=[]
    #print(len(P))
    for i in range(len(P)):
        #if i%1000000==0:
            #print(i)
        B=bits_de_controle(P[i],m,t)
        for j in range(len(B)):
            N.append(B[j])
    return N
```

```
def transmi_proba(L,p): #pour chaque bit d'une liste de bits L, il y a une probabilité p qu'il y ait une erreur de transmission
    for j in range(len(L)):
        if random()<p:
            if L[j]==0:
                L[j]=1
            else:
                L[j]=0
    return L

def transmi_rafale(L,d_min,d_max,n):
    for j in range(n):
        d=random.randint(d_min,d_max)
        R=random.randint(0,len(L)-1-24*d)
        for i in range(R,R+24*d):
            L[i]=1-L[i]
```

```
def trois_transmissions(L,p):
    M1=L[:]
    M2=L[:]
    M3=L[:]
    M1=transmi_proba(M1,p)
    M2=transmi_proba(M2,p)
    M3=transmi_proba(M3,p)
    S=[]
    for i in range(len(L)):
        v=vote([M1[i],M2[i],M3[i]])
        S.append(v)
    return S

def decode(d,m,t):
    L=[] #liste correspondant au code corrigé
    H=[d]+matH(m,t)
    for i in range(m**2):
        S=[d[i]] #liste des "suffrages"
        for j in range(1,2*m*t+1):
            X=[] #liste des éléments de la ligne
            if H[j][i]==1:
                for k in range(m**2+2*m*t):
                    if k!=i and H[j][k]==1:
                        X.append(d[k])
                S.append(xor(X))
        L.append(vote(S))
    return L
```

```
def recombinateur(L,m,t):
#L est la liste après transmission_image , la
fonction renvoie une liste décodée de binaires
    P=partition(L,m**2+2*m*t)
    p=len(P)
    CORRI=[]
    for i in range(p):
        CORRI.append(decode(list(P[i]),m,t))
    CORR=[]
    #print('reconstruction après decodage')
    for j in range(len(CORRI)):
        for k in range(len(CORRI[0])):
            CORR.append(CORRI[j][k])
    for l in range(L[1]*L[2]%(m**2)):
        a=CORR.pop()
    return CORR
```

```
def partition(L,dis): #partitionne une liste en bouts de longueur m^2.
Si il y a des éléments en en trop, on rajoute des 0
    n=len(L)
    c=0
    k=0
    M=[[]]
    for i in range(n):
        c+=1
        M[k].append(L[i])
        if c==dis:
            M.append([])
            k+=1
            c=0
    r=n%dis
    for i in range(dis-r):
        M[k].append(0)
    return(M)

def xor(L): #Fonction prenant en entrée une liste de bits, renvoyant
le xor de la liste
    if len(L)<2:
        return "Liste trop petite"
    else:
        B=abs(L[0]-L[1])
        for i in range(2,len(L)):
            B=abs(B-L[i])
        return B
```