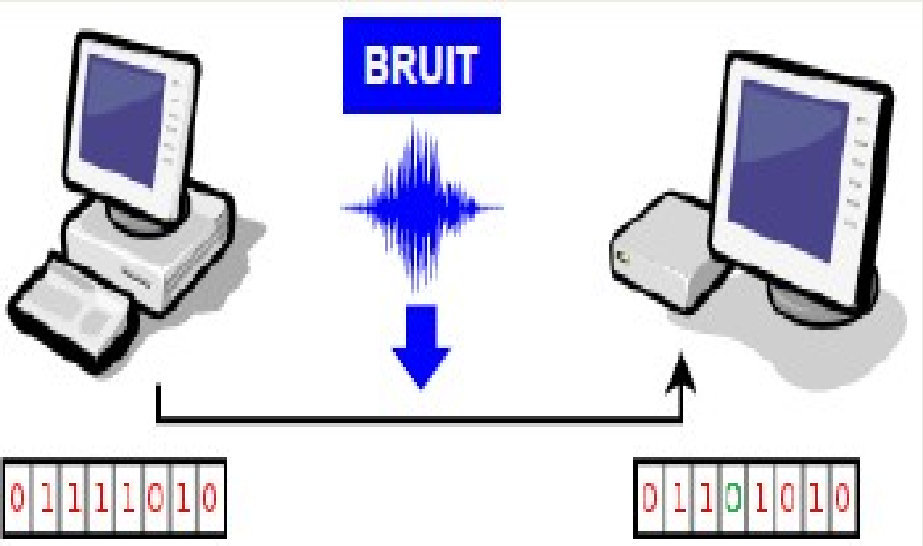
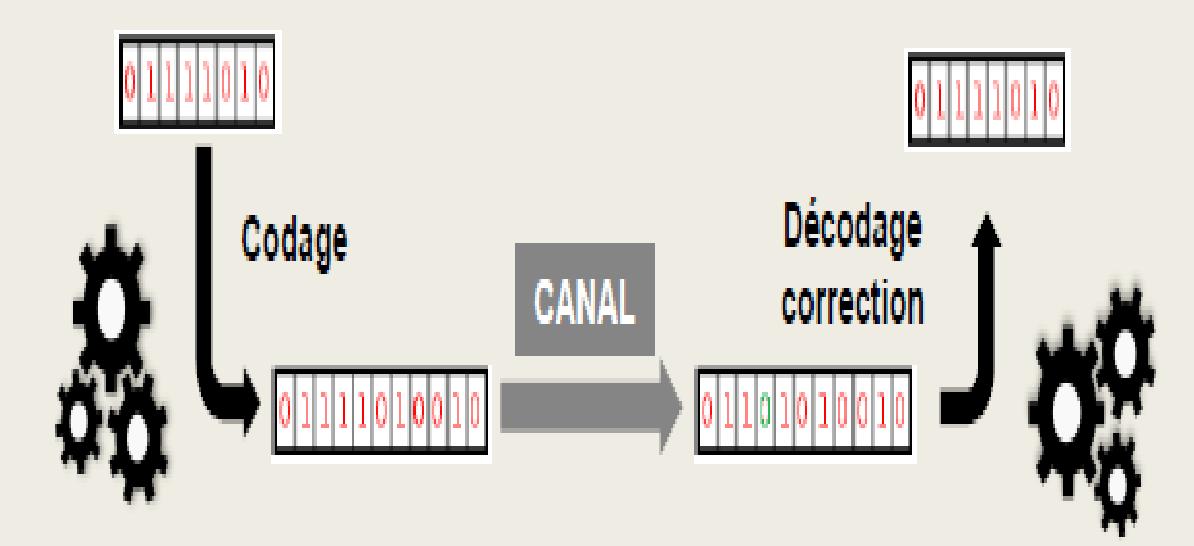
PROJET INFORMATIQUE PT





Pradal Thibault
Pfister Matthias
Dabachil Ali

2018/2019

But du Projet

- **Simulation d’un canal de transmission physique d’un signal binaire**

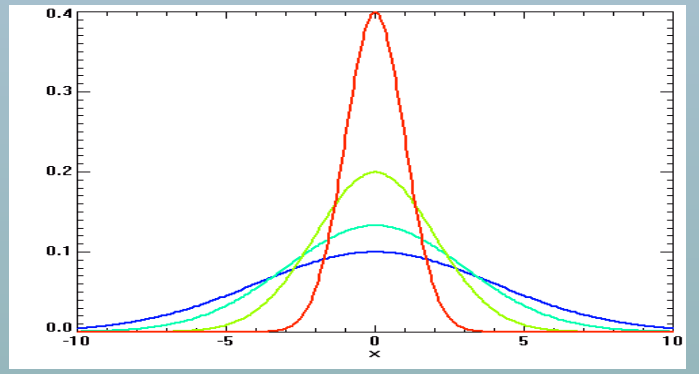
- **Émission d’un signal binaire étant à l’origine une image pixelisée noir/blanc**

- **Mise en présence d’une perturbation venant modifier la trame du signal d’origine dans le canal de transmission à réaliser**

- **Décodage de la trame + correction par le biais du codage *Hamming***

Cahier des charges

- Seuillage fixé + Attribution d’un coefficient de canal
- Émission d’un bruit ‘additif blanc gaussien‘
- Détermination précise d’un seuil



- Élaboration d’une fonction codage Hamming (7,4)
- Codage de la trame en partant d’une image pixelisée noir/blanc
- Décodage Hamming + Rectification d’une erreur
- Décodage de la trame avec correction de l’erreur rencontrée

Problématique et plan

► Comment parvenir à modéliser un canal physique de transmission d'une image donnée, tout en corrigeant les erreurs rencontrées ?

- I. Simulation du canal de transmission physique

- II. Codage de l'information et correction d'erreurs

I. Simulation du canal de transmission physique

```python
def conversionPHY(message):
    # Convertit un message en signaux de +/-5V

    #signal : liste recueillant chaque élément transformé du message
    signal = []
    #boucle de tri des éléments du message
    for i in range(len(message)):
        if message[i] == 1:
            #ajout et transformation de chaque bit 1 dans signal
            signal.append(5)
        elif message[i] == 0:
            #ajout et transformation de chaque bit 0 dans signal
            signal.append(-5)
    return(signal)
```

Affectation de 2 valeurs distinctes au signal reçu

```python
def coef_attenuation(signal):
    # Multiplication du signal par un coefficient aléatoire

    #L3 : liste recueillant les éléments du signal atténués
    L3 = []
    #création du coefficient inconnu et aleatoire inf. a 1 et positif
    c = (random.randint(1,100))/100
    for k in range(len(signal)):
        #ajout et affectation du coeff a chaque élément du signal dans L3
        L3.append(c*signal[k])
    return(L3)
```

Détermination aléatoire du coefficient de canal relatif au bruit

Simulation d'un bruit additif blanc gaussien et élaboration d'un seuillage

```python
def bruitgauss(ecartT,variance,L3):
    #Calcul du bruit additif gaussien

    #L4 : liste recueillant les éléments de L3 bruités
    L4 = []
    #y : valeur de bruit gaussien inconnue
    y = random.gauss(ecartT,variance)
    for j in range(len(L3)):
        #application du bruit aux éléments de la liste L3 puis ajout dans L4
        L4.append(L3[j]+y)
    return(L4)
```

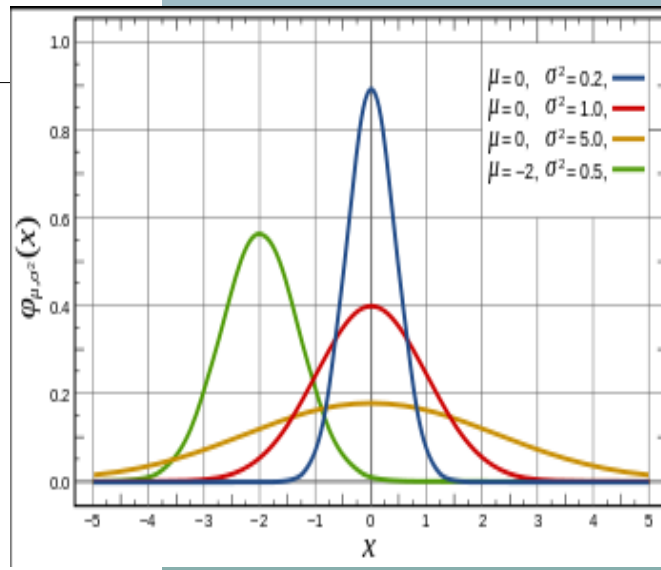
Simulation d'un bruit gaussien

```python
def seuillage(L3,L4):
    #L5 : liste de sortie du canal physique composée de bits
    L5 = []
    #S4/S3 : somme des éléments de L4/L3
    #M4/M3 : moyenne des éléments de L4/L3
    S4,S3,M4,M3 = 0,0,0,0

    for x in L4:
        S4 += x
    for p in L3:
        S3 += p

    M4 = S4/(len(L4))
    M3 = S3/(len(L3))
    #estimation d'un bruit moyen peu importe la variance
    bruitmoy = M4 - M3
    #seuillage du signal physique pour le traduire en bits dans L5
    for x in L4:
        if x-bruitmoy < 0:
            L5.append(0)
        else:
            L5.append(1)
    return(L5)
```

Élaboration d'un seuillage d'image



II. Codage de l'information et correction d'erreurs

```python
def imglist(img) :
    #Transforme une image en liste binaire

    #Argument : le nom d'un fichier image sous forme string
    # Sortie : Liste binaire
    # Exemple :
    # >>> imglist("foot.bmp")
    # [0,1,1,0,...,1,1,0,1]


    #Appel de l'image sous forme de matrice
    I = mpimg.imread(img)
    IF = I[:,:,0]

    #Transformation des pixels non blanc de l'image en pixels noirs
    for i in range(np.shape(IF)[0]) :
        for j in range(np.shape(IF)[1]) :
            if IF[i,j] != 0 :
                IF[i,j] = 1

    #Transformation de la matrice en liste
    LF = []
    for i in range(np.shape(IF)[0]) :
        for j in range(np.shape(IF)[1]) :
            LF.append(int(IF[i,j]))
    return LF
```

Accès aux informations de l'image via un tableau de valeurs (0,255)

Pixels autre que noir et blanc transformés en pixels noir

Conversion effective de l'image en trame, format Liste

Fonction codage Hamming (7,4)

```
def encodage(img) :
    # Code une liste par le code de Hamming 4/7

    # Argument : Nom d'un fichier image sous forme string
    # Sortie : Liste binaire
    # Exemple :
    # >>>encodage("foot.png")
    # [1,0,0,1,...,1,1,1,1]

    # Transformation de l'image en liste
    listimage = imglist(img)
    # Création de la liste qui contiendra le message final
    msg = []
    # Construction de la matrice génératrice d'encodage
    Ge = np.array([[1,1,0,1],[1,0,1,1],[1,0,0,0],[0,1,1,1],[0,1,0,0],[0,0,1,0],[0,0,0,1]])

    # Boucle de traitement pour chaque groupe de 4 bits
    k = 0
    while int((len(msg)/7)-(len(listimage)/4)) < 0  :

        #Création de la matrice message de 4 bits
        L4 = listimage[k:k+4]
        lnc = np.array(L4)

        #Création du message final
        lc = np.dot(Ge,lnc)
        # Correction des bits supérieurs à 1 dans le message
        for  i in range(len(lc)) :
            if lc[i]%2 == 0 :
                lc[i] = 0
            elif lc[i]%2 > 0 :
                lc[i] = 1
        [msg.append(i) for i in lc]
        k = k+4

    # Rajout des 1 à 3 bits non codables
    if len(listimage)-int(k*7/3) != 0 :
            msg = msg + listimage[(int(k*7/3)):]

    return msg
```

Décodage Hamming avec correction de l'erreur rencontrée

```
def decodage(message) :
    # Décode un message crypté par le codage de Haming 4/7 et corriges les
    # éventuelles erreurs

    # Entrée : Message modifié par le bruit sous forme de liste binaire
    # Sortie : Liste binaire corrigée et décodée
    # Exemple :
    # >>>decodlist("message")
    # [1,1,1,...,0,1,1]

    # Construction de la matrice génératrice de décryptage
    Gd = np.array([[0,0,0,1,1,1,1], [0,1,1,0,0,1,1], [1,0,1,0,1,0,1]])

    mfinal = []

    k = 0
    while k < len(message) :

        L7 = message[k:k+7]
        lnd = np.array(L7)

        # Multiplication de la matrice génératrice et de la matrice 7 bits
        matest = np.dot(Gd,lnd)
        for i in range(len(matest)) :
            if matest[i] > 1 :
                if matest[i]%2 == 0 :
                    matest[i] = 0
                elif matest[i]%2 == 1 :
                    matest[i] = 1

        # Correction d'une potentielle erreur
        if matest[1]+matest[2]+matest[0] != 0 :
            r = matest[0] + 2*matest[1] + 4*matest[2]
            if L7[r-1] == 1 :
                L7[r-1] = 0
            else :
                L7[r-1] = 1

        mfinal.append(L7[2])
        [mfinal.append(L7[i]) for i in range(4,7)]
        k = k+7

    # Rajout des bits non en trop
    if len(message)-int(k*7/3) != 0 :
        mfinal = mfinal + message[int(k*7/3):]

    return mfinal
```

Matrice génératrice de décryptage définie

Détection de l'erreur potentielle

Affichage d'une image grâce à ses dimensions

```python
def listimage_taille(msg,x,y) :
    # Affiche une image grace a une liste et les dimensions de cette image

    # Entrée : Message décodé et taille de l'image initiale
    # Sortie : Une image et le tableau numpy correspondant à cette image
    image = np.zeros((x, y), dtype=np.uint8)
    for i in range(0,x) :
        for j in range(0,y) :
            k = int((i-1)*x+j)
            image[i,j] = msg[k]
    plt.imshow(image)
    return image
```

Programme combiné

```
# Programme :


k = str(input("Nom de l'image.type (par exemple foot.png) : "))

message_code = encodage(k)# Codage de l'image
signal = conversionPHY(message_code)# Conversion en signal electrique
L3 = coef_attenuation(signal)# Atténuation du signal


ET = int(float(input("Ecart type : ")))
Var = int(float(input("Variance : ")))

L4 = bruitgauss(ET,Var,L3)# Application du bruit
message_altere = seuillage(L3,L4)# Message remis en binaire

message_decode = decodage(message_altere)# Décodage du message

img = mpimg.imread(k)
x = len(img)
y = int(len(img[0]))

listimage_taille(message_decode,x,y)# Affichage de l'image
```