

Réception et décodage d'images issues de satellites météorologiques en orbite polaire

Le protocole APT des satellites NOAA

par **Thomas LAVARENNE**
Lycée Jean Rostand - 93420 Villepinte
thomas.lavarenne@ac-creteil.fr

CET ARTICLE se situe dans la continuité de ceux publiés en juillet 2017 et en juillet 2018 [1-2], et concerne l'utilisation d'une clé TNT équipé du chipset RTL2832u, un dispositif particulièrement pratique et très peu onéreux qui, associé à la puissance du logiciel libre GNU Radio, offre de nombreuses possibilités expérimentales dans le domaine des radiofréquences (télécommande de porte de garage, RDS de la radio FM, signaux ADS-B issus des avions...). Cette fois-ci, nous allons nous intéresser aux images fournies par certains satellites météorologiques, les satellites NOAA-15, 18 et 19. Nous allons utiliser la clé TNT et le logiciel GNU Radio pour réceptionner et écouter le signal APT (Automatic Picture Transmission) issu des satellites NOAA-15, 18 ou 19 et l'enregistrer au format wav. Puis nous chercherons à construire et améliorer un script Python dans le but d'afficher l'image reçue. De la construction de l'antenne à l'affichage de l'image à l'aide du langage de programmation Python, les différentes étapes sont assez simples et illustrent de manière expérimentale de nombreux thèmes de physique (propagation d'une onde électromagnétique, antennes, mouvement d'un satellite, filtrage, images numériques...).

1. LES SATELLITES NOAA ET LA NORME APT

Les satellites NOAA (*National Oceanic and Atmospheric Administration*) sont des satellites américains à orbite polaire évoluant à une altitude d'environ 850 km de la Terre. Ils ont pour mission principale l'observation des phénomènes météorologiques à la surface de la Terre. Le premier d'entre eux a été lancé en 1970 (NOAA-1) et le dernier en 2009 (NOAA-19). La période de révolution de ces satellites est voisine de cent minutes.

Ils sont équipés du capteur-imageur AVHRR (*Advanced Very High Resolution Radiometer*) qui peut délivrer deux types d'images, soit en utilisant le mode APT (*Automatic Picture Transmission*), soit en mode HRPT (*High Résolution Picture Transmission*). Dans cet article, nous allons nous intéresser uniquement au mode APT.

Depuis 2013, les seuls satellites NOAA diffusant encore en utilisant le mode APT sont les NOAA-15, 18 et 19.

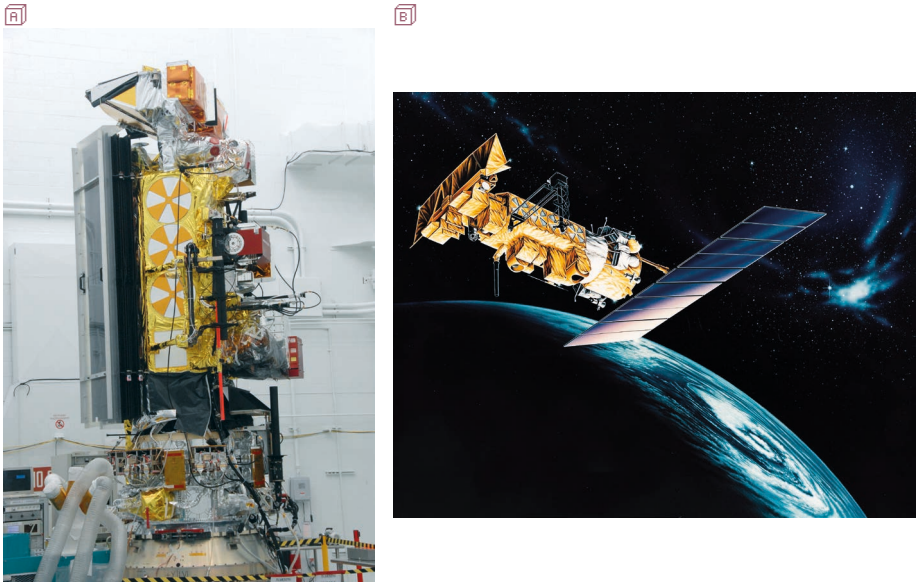


Figure 1 - **A**) Le satellite NOAA-19 avant son lancement - **B**) Le même satellite en orbite (vue d'artiste)⁽¹⁾.

Le mode APT est un système de transmission analogique qui module en amplitude les données du capteur sur une fréquence audio de 2400 Hz. Le signal modulé en amplitude est envoyé vers les stations sur Terre grâce à une modulation de fréquence (137,620 MHz pour NOAA-15 ; 137,9125 MHz pour NOAA-18 et 137,100 MHz pour NOAA-19).

Le capteur fonctionne dans les domaines visible et infrarouge. Les deux images sont envoyées sous la forme d'une seule ligne horizontale toutes les 0,5 seconde. Le capteur rajoute également des données de synchronisation et de télémétrie. Toutes les informations utiles concernant ce protocole se trouvent aisément sur Internet ou dans la documentation [3-4].

2. RÉCEPTION ET ÉCOUTE DU SIGNAL

2.1. Quelle antenne choisir ?

La documentation sur la réception des signaux APT est assez fournie. Le point délicat est la construction de l'antenne adaptée à la réception du signal à 137 MHz

(1) Source : Wikimedia Commons, les images proviennent de la NASA.

polarisé circulairement. L'antenne la plus simple à réaliser, donnant des résultats satisfaisants, est l'antenne dipôle. Commençons par déterminer la longueur d'onde du signal à réceptionner :

$$\lambda = \frac{c}{f} = \frac{3,00 \cdot 10^8}{137 \cdot 10^6} = 2,19 \text{ m.}$$

Partons sur un dipôle de longueur totale égale à la moitié de la longueur d'onde auquel nous appliquons un angle de 120° afin de réaliser une meilleure adaptation d'impédance :

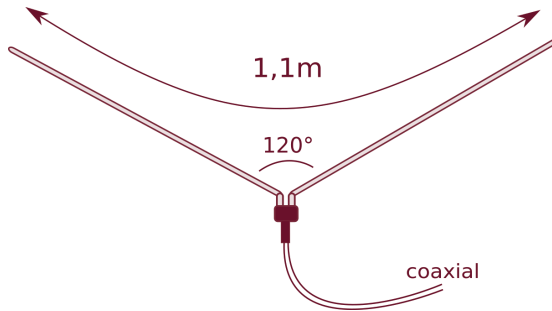


Figure 2 - Réalisation de l'antenne dipôle en V.

En effet, en utilisant les données simulées issues de [5] on peut visualiser (cf. figure 3) les variations du gain et du taux d'onde stationnaire en fonction de l'angle donné au dipôle et ainsi constater qu'un angle de 120° est un bon compromis en terme de gain et d'adaptation d'impédance.

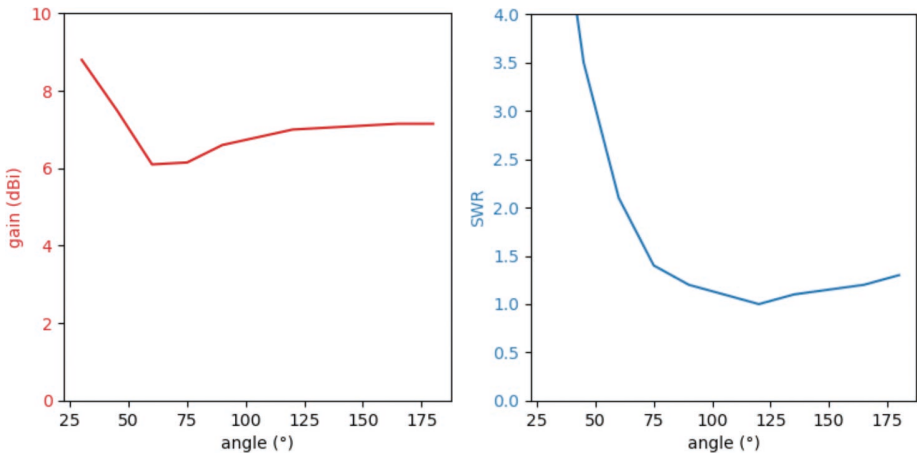


Figure 3 - Variations du gain et du taux d'onde stationnaire en fonction de l'angle donné au dipôle (d'après [5]).

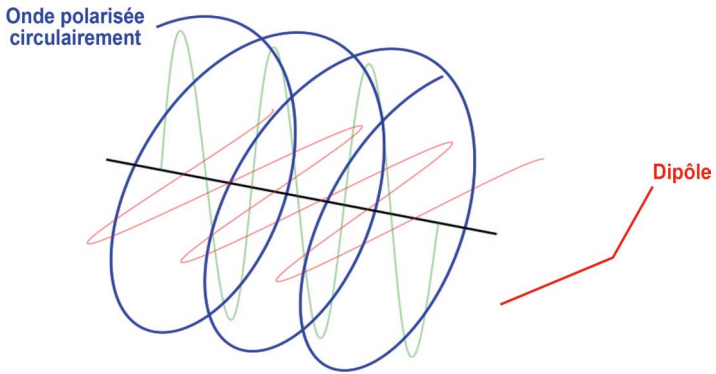


Figure 4 - L'onde incidente est polarisée circulairement alors que le dipôle est polarisé rectilignement.

A



B

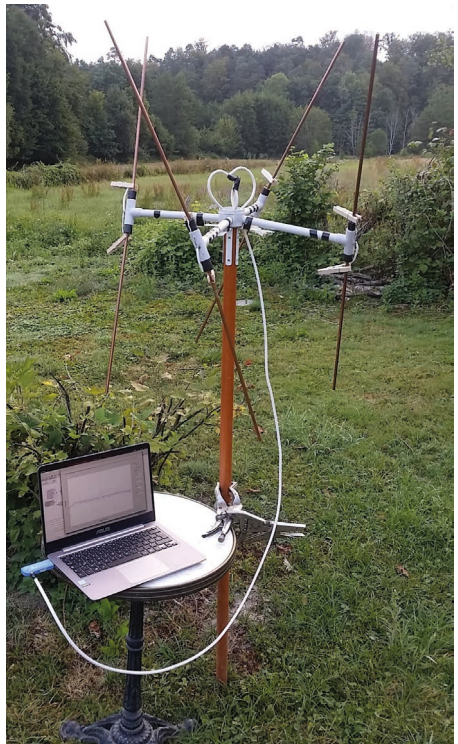


Figure 5 - A Antenne dipôle en V, simple et efficace ! - B « Double cross antenna » qui a peu servie de par son encombrement et le faible gain apporté au final.

Le signal à réceptionner étant polarisé circulairement, il peut sembler étrange de choisir un dipôle, adapté aux polarisations rectilignes. En effet, cela n'est pas optimal puisque nous n'allons réceptionner qu'une seule des deux composantes du signal incident, ce qui se traduira par une perte de 3 dB du simple fait du choix de l'antenne (cf. figure 4, page ci-contre).

Une autre possibilité, bien plus complexe à construire, et nous permettant de récupérer ces 3 dB (dans l'idéal), est une antenne constituée de quatre dipôles positionnés en forme de croix (« double cross antenna ») (cf. figure 5B, page ci-contre). En pratique, concernant la réception, l'antenne dipôle n'a pas à rougir face à sa concurrente, tout en étant beaucoup plus simple à réaliser et à manier. Dans la suite de cet article, la majorité des images présentées ont été réalisées avec l'antenne dipôle.

2.2. Bilan de liaison

La documentation du protocole APT annonce une puissance rayonnée équivalente (ERP) de 37 dBm pour l'émission au niveau du satellite. Avant de tenter quoi que ce soit, nous allons voir si, avec les informations dont nous disposons, nous avons une chance de récupérer un signal suffisamment important grâce à notre antenne. Si c'est bien la puissance rayonnée équivalente ERP qui est donnée (donc relative à un dipôle et pas relative à l'antenne isotrope), il faut commencer par ajouter 2,15 dB pour convertir cette valeur d'émission en puissance isotrope rayonnée équivalente (EIRP). On aurait donc pour l'émission :

$$EIRP \text{ (dBm)} = ERP \text{ (dBm)} + 2,15$$

$$EIRP \simeq 39 \text{ dBm.}$$

Les pertes en espace libre (« Free Space Loss », FSL) sont :

$$FSL \text{ (dB)} = 20 \log \left(\frac{\lambda}{4\pi d} \right)$$

$$FSL \text{ (dB)} = 20 \log \left(\frac{2,19}{4\pi \times 850\,000} \right) \simeq -134 \text{ dB.}$$

Pour notre antenne dipôle, d'après la figure 3, on peut espérer un gain en réception G_r d'environ 7 dBi, ce qui au final ferait pour la puissance reçue (sans oublier de retrancher les 3 dB du fait de la mauvaise polarisation de l'antenne) :

$$Pr \text{ (dBm)} = EIRP + FSL + G_r - 3 = 39 - 134 + 7 - 3 = -91 \text{ dBm.}$$

Nous n'avons pas trouvé de valeur bien définie pour la sensibilité du récepteur utilisé (la clé TNT RTL2832u). Certaines sources vont jusqu'à affirmer une sensibilité de -130 dBm, valeur qui nous paraît un peu optimiste. D'autres sources indiquent une sensibilité plutôt autour de -110 dBm, ce qui nous paraît plus réaliste. Au final, si on considère cette dernière valeur pour la sensibilité du récepteur, avec notre étude conduisant à une puissance au niveau de la réception de -91 dBm, le signal pourrait se trouver à une vingtaine de décibels au-dessus du bruit, ce qui est plutôt une bonne nouvelle pour la suite et pourrait valider notre démarche.

2.3. Réception du signal

Après la théorie, même si notre développement précédent contient beaucoup d'approximations, il est grand temps de passer à la pratique en toute confiance. Pour tester notre dispositif, il faut attendre qu'un satellite NOAA-15, 18 ou 19 se présente⁽²⁾ et, dans un premier temps, utiliser le graphe *GNU Radio*⁽³⁾ suivant (cf. figure 6).

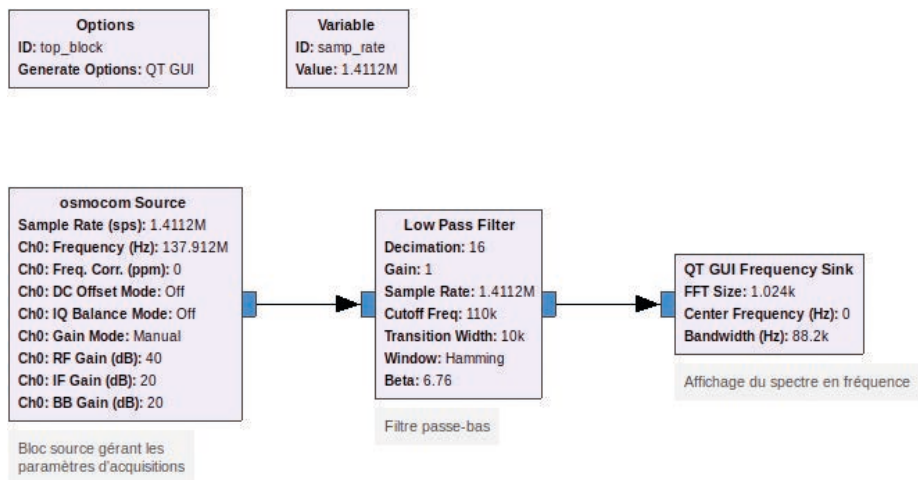


Figure 6 - Graphe *GNU Radio* permettant d'afficher le spectre en fréquence du signal reçu après filtrage et décimation.

Le bloc « Osmocom Source » gère les paramètres du système d'acquisition (la clé TNT) :

- ◆ Frequency : 137,9125 MHz (ici pour NOAA-18) ;
- ◆ RF Gain : 40 dB (pas loin du maximum possible de la clé utilisée – à ajuster selon la qualité de réception).

La variable « samp_rate » (fréquence d'échantillonnage) a été choisie à 1,4112 MHz, valeur que nous justifierons plus en détail dans la suite (remarquons déjà qu'il s'agit d'un multiple entier de 44,1 kHz, fréquence de travail de la carte son utilisée dans notre étude). Lorsque le satellite est à proximité et l'antenne correctement orientée, nous obtenons la représentation fréquentielle suivante (cf. figure 7).

(2) Le site <https://www.n2yo.com> s'est avéré bien utile pour notre étude.

(3) On rappelle que le logiciel *GNU Radio* est un logiciel libre, gratuit et disponible pour Linux, Mac et Windows.

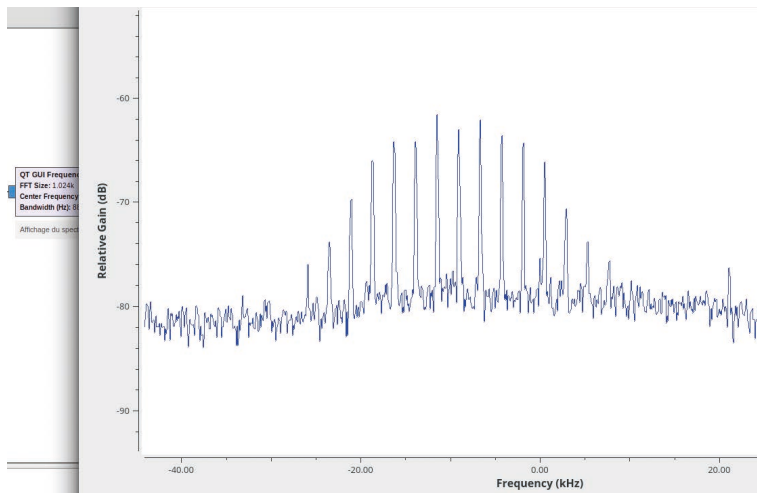


Figure 7 - Représentation fréquentielle du signal reçu.

Nous reconnaissons la distribution spectrale caractéristique d'une modulation de fréquence, les pics étant séparés de 2400 Hz, fréquence du signal audio transmis contenant les informations concernant l'image. Chaque passage dure environ quinze minutes, et nous pouvons passer au décodage lorsque nous sommes capables d'obtenir un signal suffisamment stable durant une dizaine de minutes au moins.

2.4. Écoute du signal

Un attrait particulier de cette étude est le fait que le signal réceptionné est audible. Il y a quelque chose de très satisfaisant lorsque, d'abord noyé dans le bruit de fond, on commence à distinguer le signal bien reconnaissable à 2400 Hz émis par le satellite et constater ainsi que de la théorie à la pratique, tout fonctionne exactement comme prévu !

Pour ce faire, nous allons encore exploiter un peu les fonctionnalités de *GNU Radio*. En sortie du bloc «Low Pass Filter», nous obtenons un signal modulé en fréquence échantillonné à 88,2 kHz. Ce cas est analogue à celui de la radio FM et *GNU Radio* possède un bloc «WBFM Receive» permettant de le démoduler simplement⁽⁴⁾. Le flux arrivant étant de 88,2 kHz, il suffira d'ajouter une décimation d'un facteur 2 pour obtenir un flux audio à 44,1 kHz à envoyer directement à la carte son à travers le bloc «Audio Sink». Si la carte son est paramétrée à 48 kHz ou une autre fréquence,

(4) Le bloc «WBFM Receive» avait déjà été utilisé lors de l'étude du signal RDS («Radio Data System») de la radio FM [1].

il suffit d'adapter le paramètre d'échantillonnage en fonction de cette valeur et nous comprenons maintenant pourquoi nous avons réglé le paramètre «*samp_rate*» à 1,4112 MHz lors de l'acquisition.

Avant d'enregistrer le signal réceptionné avec le bloc «*Wav File Sink*», nous allons réduire l'échantillonnage à la valeur de 20 800 Hz afin de réduire sa taille et de faciliter son traitement ultérieur (la largeur de l'image étant de 2 080 pixels). Pour ce faire, nous utilisons le bloc «*Rational Resampler*» en appliquant un facteur 208/441. Le graphe complet est donné ci-dessous (cf. figure 8).

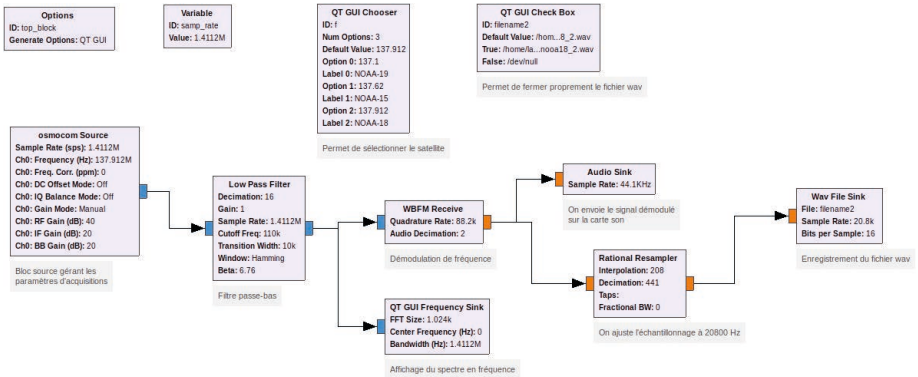


Figure 8 - Graphe GNU Radio complet pour cette étude.

Le bloc «*QT GUI Chooser*» est optionnel et permet d'afficher un champ déroulant pour sélectionner plus facilement la fréquence du satellite (NOAA-15, 18 ou 19). Le bloc «*QT GUI Check Box*» permet de fermer proprement le fichier audio à la fin du passage en redirigeant le flux audio vers `/dev/null`⁽⁵⁾ lorsqu'on décoche la case (pas forcément nécessaire). En effet, il s'avère que dans certains cas, le fichier reste «*vide*» sur le disque si on ferme la fenêtre du programme sans avoir arrêté proprement l'enregistrement. La fenêtre qui s'affiche lors du lancement du programme complet est donnée sur la figure 9 (cf. page ci-contre). Si l'ordinateur utilisé possède peu de ressources, il est tout à fait possible de désactiver le bloc «*QT GUI Frequency Sink*», l'essentiel ici étant l'enregistrement du fichier audio.

3. DÉCODAGE ET AFFICHAGE DE L'IMAGE

Il existe déjà des logiciels permettant de décoder le fichier audio et d'afficher l'image («*wxtoimg*» par exemple), mais il est bien plus intéressant et formateur d'écrire

(5) Tous les chemins présentés ici sont valables pour des systèmes d'exploitation de type Linux. À adapter éventuellement selon le système d'exploitation utilisé.

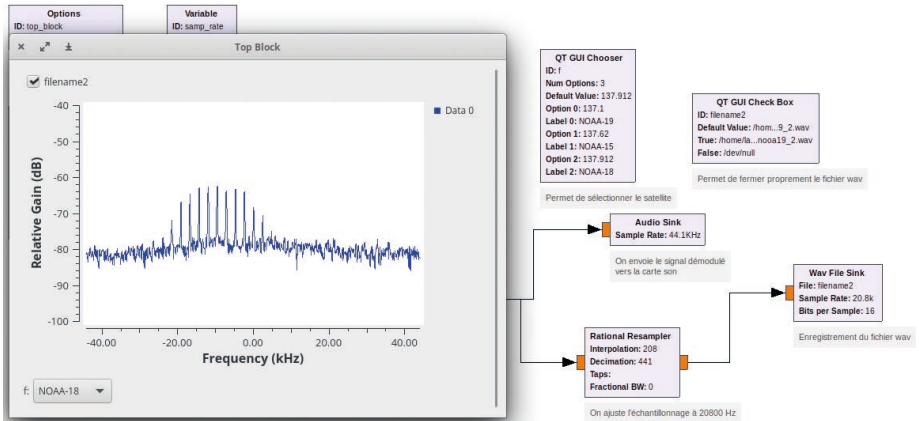


Figure 9 - Lancement et fonctionnement du programme complet.

notre propre script Python⁽⁶⁾. Nous allons détailler maintenant chaque étape. Pour commencer, il faut importer les modules `scipy.io.wavfile` et `matplotlib.pyplot` qui permettent respectivement de lire les données d'un fichier wav et d'afficher des graphiques :

```
import scipy.io.wavfile
import matplotlib.pyplot as plt
import PIL
from PIL import Image
from scipy import signal
filename = 'chemin/du/fichier/wav'
fe, signal = scipy.io.wavfile.read(filename)
plt.plot(signal)
```

Cette avant-dernière ligne permet d'extraire la fréquence d'échantillonnage et les données du fichier wav puis de les stocker dans un array nommé 'signal'. En exécutant ce petit programme, nous obtenons le résultat suivant (cf. figure 10, page ci-après).

La fréquence porteuse du signal est à 2400 Hz et les données correspondant à l'image sont modulées en amplitude. La méthode `hilbert()` permet de démoduler le signal en affichant son enveloppe. Avant cette étape, nous prenons soin de conserver un nombre de points multiples de la fréquence d'échantillonnage (`len()` renvoie le nombre de points et l'opérateur `//` effectue la division entière) :

```
signal = signal[:fe * (len(signal) // fe)]      #on garde un nbr de points cohérent
signal_demodule = abs(hilbert(signal))         #abs(hilbert()) pour détection d'enveloppe
```

(6) Pour écrire ce script nous nous sommes basés sur le décodeur python indiqué [8] de la bibliographie ainsi que sur le script *Octave* écrit par J.-M. Friedt [7].

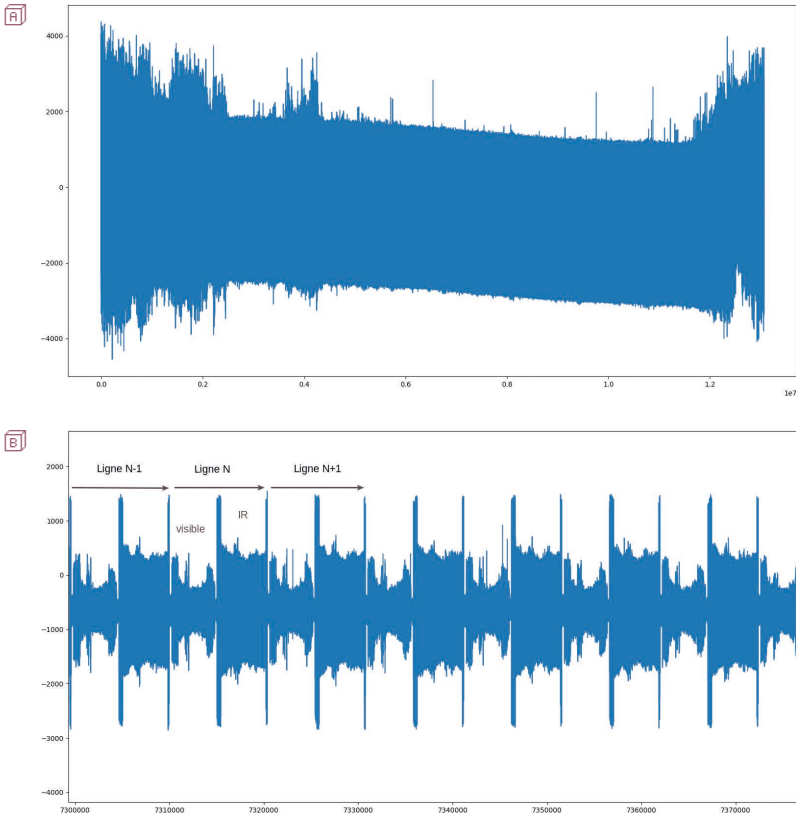


Figure 10 - Signal extrait **A** complet et **B** zoomé. Nous constatons qu'il y a trois lignes envoyées en environ 30 000 points, en tenant compte de la valeur de la fréquence d'échantillonnage de 20 800 Hz nous retrouvons bien que chaque ligne est émise pendant : $30000/3/20800 \approx 0,5$ s.

Ajoutons ensuite un filtre passe-bas (fréquence de coupure f_c) pour obtenir les résultats de la figure 11 (cf. page ci-contre) :

```
fc=1100
```

```
b, a = signal.butter(5, fc/(fe/2), 'low') #filtre passe-bas d'ordre 5
```

```
signal_demodule = signal.filtfilt(b, a, signal_demodule)
```

Le choix de 1100 Hz pour la fréquence de coupure permet d'éliminer efficacement les parasites résiduels à 2400 Hz sans trop atténuer la partie correspondant aux stries blanches et noires au début de chaque image (dont la fréquence est environ 1000 Hz).

Nous allons maintenant garder un point sur cinq afin d'afficher $20800/2/5 = 2080$

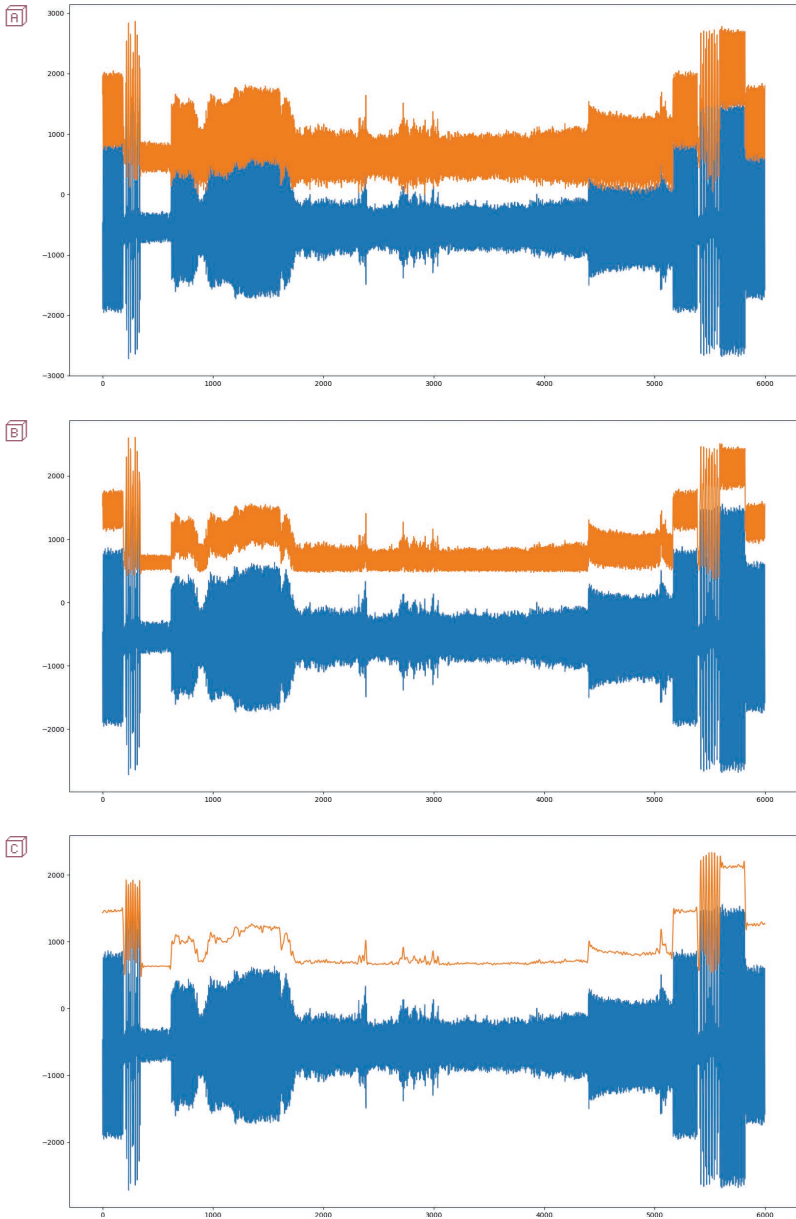


Figure 11 - Le signal brut est en bleu, son enveloppe démodulée est en orange.
De haut en bas : **A**) Aucun filtrage - **B**) $f_c = 2400$ Hz - **C**) $f_c = 1100$ Hz.

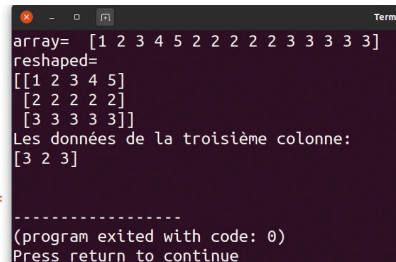
points par ligne (la résolution maximale). 20800 correspond à la fréquence d'échantillonnage du signal et le facteur 2 au fait qu'une ligne est envoyée toutes les 0,5 s. Pour ce faire utilisons la méthode `reshape()` avec la ligne suivante :

```
reshaped = signal_demodule.reshape(len(signal_demodule)//5,5)
```

L'opérateur `//` étant la division entière, cette ligne réarrange les données sous la forme d'un tableau à cinq colonnes dont on n'en conservera qu'une seule pour la suite. Observons un exemple afin de bien comprendre comment fonctionne la méthode `reshape()` (cf. figure 12). Pour sélectionner une seule colonne, la troisième par exemple (dont l'index est le numéro 2), il suffit d'écrire `reshaped[:,2]`, comme le montre l'exemple de la figure 12.

```
decim=reshaped[:,2]
low=300
high=2000
data = (255 * (decim - low) / (high - low))
```

```
import numpy as np
#on définit un array
data = np.array([1,2,3,4,5,2,2,2,2,2,3,3,3,3,3])
#on redéfinit cet array sous forme d'un tableau
# de 5 colonnes et 3 lignes
reshaped=data.reshape(3,5)
print("array= ", data)
print("reshaped= ")
print(reshaped)
#Si on ne veut garder que la troisième colonne (index 2):
print("Les données de la troisième colonne:")
print(reshaped[:,2])
```



```
array= [1 2 3 4 5 2 2 2 2 2 3 3 3 3 3]
reshaped=
[[1 2 3 4 5]
 [2 2 2 2 2]
 [3 3 3 3 3]]
Les données de la troisième colonne:
[3 2 3]
-----
(program exited with code: 0)
Press return to continue
```

Figure 12 - Exemple de réarrangement en cinq colonnes avec la méthode `reshape()`.

Les paramètres 'low' et 'high' sont à ajuster afin d'obtenir le meilleur contraste. En première approche, nous pouvons les estimer en observant le minimum et le maximum de la courbe orange sur la figure 11c (cf. page précédente).

Pour un meilleur affichage, on définit un paramètre 'offset' qui permet d'ajuster le début de l'image à gauche :

```
pix_lignes = int(fe/5/2)           #nombre de pixels par ligne
lines = int(len(data) / pix_lignes) #nombre total de lignes
offset=1810                        #valeur à ajuster pour afficher à partir du début
```

Et enfin, nous générons l'image en réarrangeant notre array en un tableau de 2080 colonnes et puis nous affichons l'image:

```
matrix = data[offset:(lines-1)*pix_lignes+offset].reshape((lines-1, pix_lignes))
image = PIL.Image.fromarray(matrix)
image.show()
```

Arrivé à cette étape, nous voyons s'afficher le résultat suivant (cf. figure 13, page ci-contre).

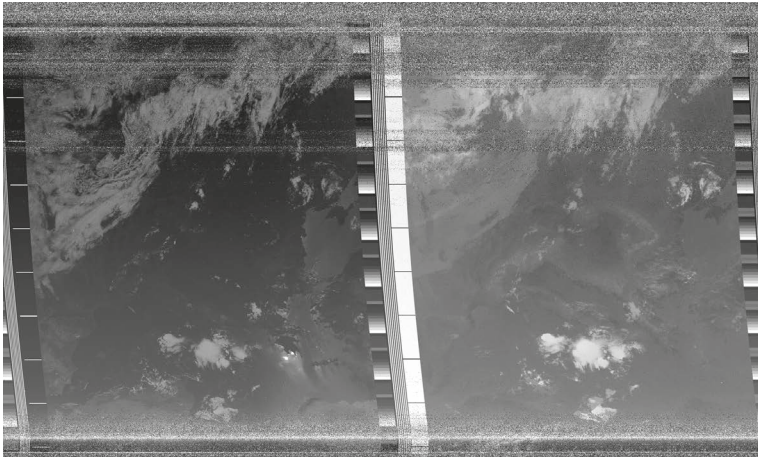


Figure 13 - Résultat brut issu du script complet : à gauche l'image en lumière visible et à droite en infrarouge.

```

1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3  import PIL
4  from scipy import signal
5  from scipy.signal import *
6  import scipy.io.wavfile
7
8  #extraire donnee:
9  (fe, donnee) = scipy.io.wavfile.read('noaa18_20_aout_2018_9h30.wav')
10 donnee = donnee[:fe * (len(donnee) // fe)]
11 signal_demodule=abs(hilbert(donnee))
12
13 fc=1100 #filtre passe-bas
14 b, a = signal.butter(5, fc/(fe/2), 'low')
15 signal_demodule = signal.filtfilt(b, a, signal_demodule)
16
17 reshaped = signal_demodule.reshape(len(signal_demodule)//5,5) #Réarrangement array sur 5 colonnes
18 decim=reshaped[:,2] #on ne conserve que la troisième valeur (par ex)
19 low=300 #réglage du contraste
20 high=2200
21 data = (255 * (decim - low) / (high - low)) #on convertit sur une échelle de 0 à 255 la valeur de chaque pixel
22
23 pix_lignes = int(fe/5/2) #nombre de pixels par ligne
24 lines = int(len(data) / pix_lignes) #nombre total de lignes
25 offset=1810 #valeur à ajuster pour afficher à partir du début
26
27 matrix = data[offset:(lines-1)*pix_lignes+offset].reshape((lines-1, pix_lignes)) #Affichage de l'image
28 image = PIL.Image.fromarray(matrix)
29 image.show()
30

```

Figure 14 - Script complet.

4. ANALYSE DE L'IMAGE ET AMÉLIORATIONS

4.1. Analyse

Il est intéressant de comparer avec l'image satellite fournie par Météociel⁽⁷⁾ le même jour à la même heure. Nous reconnaissons bien les nuages sur la Sicile et à côté

(7) <http://www.meteociel.fr>

de la Sardaigne. En observant attentivement notre image nous distinguons assez bien la Grèce, la Turquie, la méditerranée et une partie de l'Espagne. Remarquons également que la France est coupée en deux par d'épais nuages.

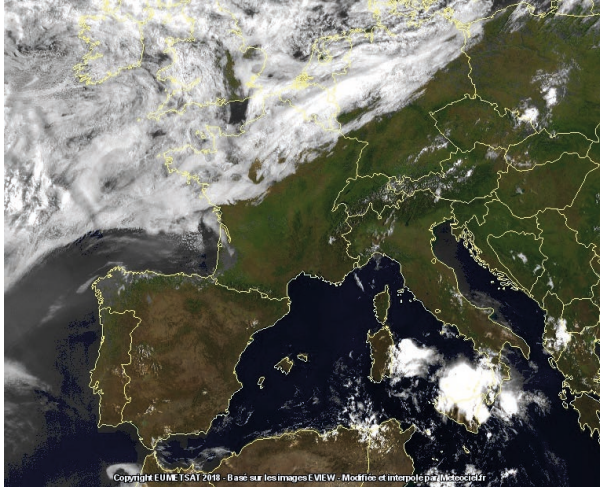


Figure 15 - Image issue du site meteociel.fr le 20 août 2018 à 9 h 30.

Il peut être intéressant d'observer le signal audio (grâce au logiciel *Audacity*) afin de bien visualiser la modulation d'amplitude et de faire le lien avec une ligne de l'image.

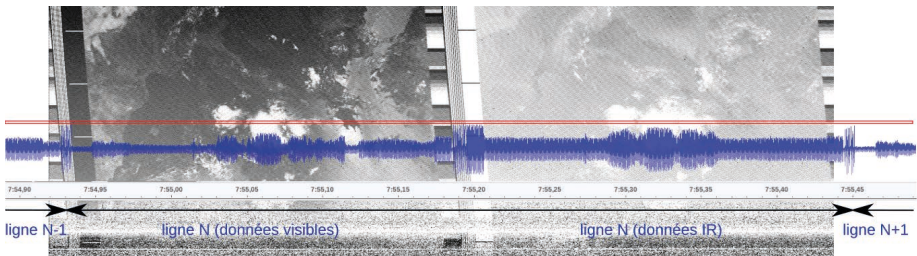


Figure 16 - Mise en évidence de la modulation d'amplitude, plus l'amplitude du signal est grande, plus la couleur est blanche.

Dans le cadre rouge : la ligne de l'image qui correspond à la forme du signal audio modulé (en bleu). Ce qui dépasse à gauche et à droite correspond respectivement aux lignes N-1 et N+1. Sur l'image décodée, nous observons deux effets provoquant une légère déformation de l'image : une dérive linéaire due à un écart entre la fréquence de l'oscillateur local et celui ayant généré le signal, écart non totalement compensé par la boucle à verrouillage de phase (PLL), et une forme courbée due à l'effet Doppler (variation de la fréquence d'environ plus ou moins 3 kHz).

4.2. Améliorations

Obtenir un tel résultat avec un petit script python d'une vingtaine de lignes est tout à fait satisfaisant. Néanmoins, si nous décodons notre fichier audio avec le logiciel *wxtoimg*, nous constatons une qualité nettement meilleure pour le logiciel... Nous allons proposer quelques pistes pour améliorer notre image, sous forme de fonctions.

Premièrement, nous avons constaté sur le graphique de la figure 10A, une dérive linéaire, le signal « descend », ce qui nous pose problème pour le calcul du contraste. On peut corriger cette dérive avec un calcul de régression linéaire :

```
import numpy as np
from scipy import stats

def calcul_droite_regression(donnee):
    ''' effectue la regression linéaire pour éliminer une dérive vers le haut ou le bas,
    permet d'améliorer nettement la démodulation. Renvoie y = aX+b'''
    X=np.arange(len(donnee))
    pente,ord_origine, r_value, p_value, std_err = stats.linregress(X,donnee)
    y=pente*X+ord_origine
    return y
```

Il faudra juste retrancher y en effectuant $donnee = donnee - y$ pour améliorer grandement la démodulation dans ce cas.

Nous pouvons également chercher à automatiser et améliorer le calcul des valeurs 'low' et 'high' pour le calcul du contraste, en cherchant la moyenne des valeurs maximales et minimales pour un certain nombre de ligne au voisinage de celle considérée :

```
def convertGrey(data,l):
    ''' Cette fonction convertit les valeurs en niveau de gris sur l'échelle 0-255 en
    cherchant une «moyenne» des valeurs max et min sur les l lignes précédentes et suivantes '''
    data2=[]
    lines=int(len(data) / pix_lignes)
    for ligne in range(l,lines-1):
        maxi=[]
        mini=[]
        for n in range(-l,l):
            maxi.append(0.7*max(data[(ligne+n)*pix_lignes:(ligne+1+n)*pix_
lignes]))
            #Les coeff 0.7 et 1.1 sont à ajuster pour obtenir
un meilleur résultat
            mini.append(1.1*min(data[(ligne+n)*pix_lignes:(ligne+1+n)*pix_
lignes]))
        minimum=mean(mini)
        maximum=mean(maxi)
        for k in range(pix_lignes):
            data2.append(255 * (data[(ligne-1)*pix_lignes+k] - minimum) /
(maximum - minimum))
            #conversion sur 0-255
```

```

for ligne in range(lines-1,lines): #Pour ajouter la partie manquante
    for k in range(pix_lignes):
        data2.append(255 * (data[ligne*pix_lignes+k] - minimum) / (maximum - minimum))
data2=np.array(data2)
lines = int(len(data2) / pix_lignes)
return data2,lines

```

Pour finir, nous pouvons également chercher à automatiser le calcul de l'offset pour ne pas être obligé de modifier le paramètre à la main pour chaque fichier en essayant de détecter les stries noires et blanches au début de chaque image :

```

def offset(data):
    '''Cette fonction détecte (à peu près bien) les stries blanches et noires vers le milieu de l'image. Elle renvoie la valeur contenue dans offset '''
    i=2*len(data)//5
    while(i<3*len(data)//5):
        if(data[i]>240 and data[i+3]<170 and data[i+5]>230 and data[i+8]<170 and data[i+11]>240 and data[i+13]<170 and data[i+pix_lignes]>240 and data[i+3+pix_lignes]<170 and data[i+5+pix_lignes]>230 and data[i+8+pix_lignes]<170 and data[i+11+pix_lignes]>240 and data[i+13+pix_lignes]<170):
            break
        else:
            i=i+1
    offset=i*pix_lignes
    return offset

```

Ces modifications ne prétendent pas être optimales, et il y a sans doute beaucoup mieux à faire, mais elles ont le mérite de fonctionner et d'améliorer la qualité de l'image pour toutes les acquisitions réalisées jusqu'à présent. Sur les figures 17 et 18, quelques exemples pris parmi les nombreuses images réceptionnées pour cette étude.

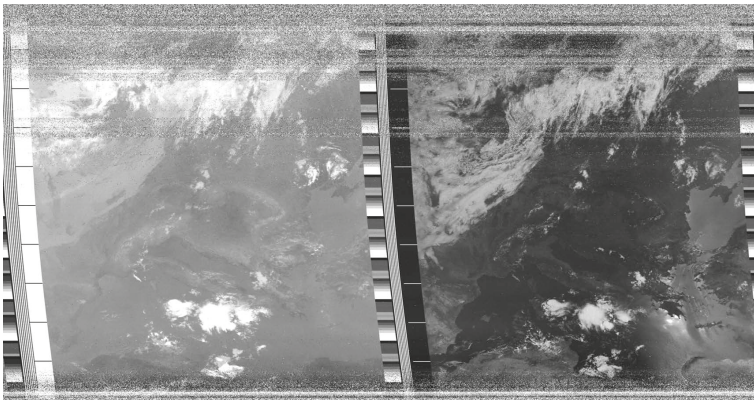


Figure 17 - Image obtenue avec notre script « amélioré ». Le début de l'image est détecté automatiquement et le contraste est amélioré.

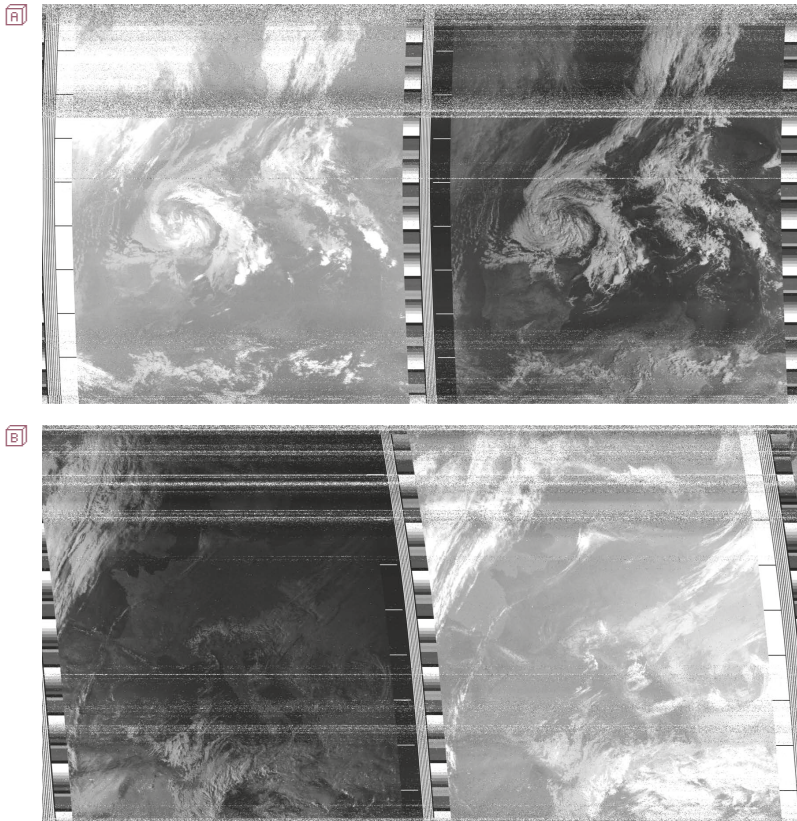


Figure 18 - Deux autres exemples d'acquisitions.

- A** Passage de NOAA-18 le 7 octobre 2018 vers 10h -
- B** Passage de NOAA-19 le 24 février 2019 vers 16h30.

CONCLUSION

Le protocole APT a le mérite d'être simple à décoder et permet de s'entraîner à la réception de signaux venant de satellites en orbite. Il s'agit d'une très belle illustration de notions de physiques telles que la propagation des ondes électromagnétiques dans l'atmosphère ou dans les antennes, les lois de Kepler peuvent être utilisées (période de révolution des satellites), la modulation d'amplitude, la numérisation du signal, les images numériques... L'emploi du langage de programmation Python a également été très enrichissant et a permis de comprendre parfaitement le principe de la modulation et de l'affichage de l'image.

REMERCIEMENTS

Une fois de plus, un grand merci à Jean-Michel Friedt pour l'ensemble de son œuvre !

BIBLIOGRAPHIE ET NETOGRAPHIE

- [1] T. Lavarenne, « Une clé TNT pour l'étude expérimentale de signaux radiofréquences mettant en jeu différentes modulations numériques », *Bull. Un. Prof. Phys. Chim.*, vol. 111, n° 995, p. 721-739, juin 2017.
- [2] T. Lavarenne, « Introduction à la surveillance du trafic aérien : les signaux ACARS et ADS-B », *Bull. Un. Prof. Phys. Chim.*, vol. 112, n° 1005, p. 831-855, juin 2018.
- [3] Des informations sur les satellites NOAA issues du site Éduscol : <https://eduscol.education.fr/orbito/system/noaa/noaa00.htm>
- [4] "The NOAA KLM User's Guide" : <https://www1.ncdc.noaa.gov/pub/data/satellite/publications/podguides/N-15%20thru%20N-19/pdf/0.0%20NOAA%20KLM%20Users%20Guide.pdf>
- [5] "What Happens If... A Dipole is Bent Sideways into a 'V'?", Dick Reid, KK4OBI at QSL.net : <https://www.qsl.net/kk4obi/Center-fed%20V-dipoles%20Lateral.html>
- [6] Instructions for Building a Portable Double Cross Antenna: Great for NOAA/ Meteor Weather Satellites : <https://www.rtl-sdr.com/instructions-for-building-a-double-cross-antenna-great-for-noaameteor-weather-satellites/>
- [7] J.-M. Friedt, « Examen sur la transmission de l'information », 14 mars 2016 : http://jmfriedt.free.fr/examenTI_L3_2016.pdf
- [8] Un script en python de Zac Stewart qui permet de synchroniser le début de chaque image avec le début de chaque ligne (qui a servi de base pour cette étude) : <https://github.com/zacstewart/apt-decoder>

Complément de l'article

Cet article comporte un complément nommé :

◆ NOAA18_20_aout_2018_9h30.wav

Il est disponible sur le site de l'UdPPC sous la forme d'un fichier zippé 10291075.



Thomas LAVARENNE

Enseignant en sciences physiques
Lycée Jean Rostand
Villepinte (Seine-Saint-Denis)