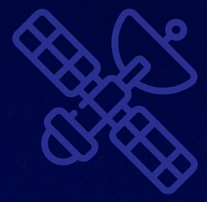
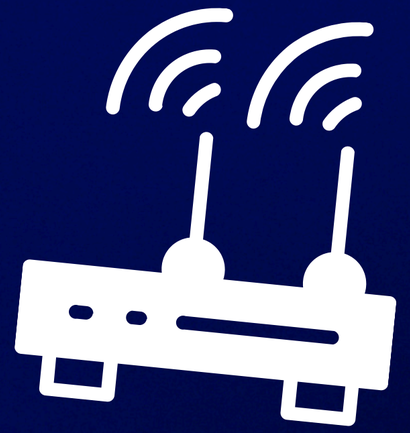
UVSQ
université PARIS-SACLAY
IUT de Vélizy-Rambouillet
CAMPUS DE VÉLIZY-VILLACOUBLAY



CHAÎNE DE TRANSMISSION NUMÉRIQUE

TP OFDM
N MODULATEURS COMPLEXES

Module R402

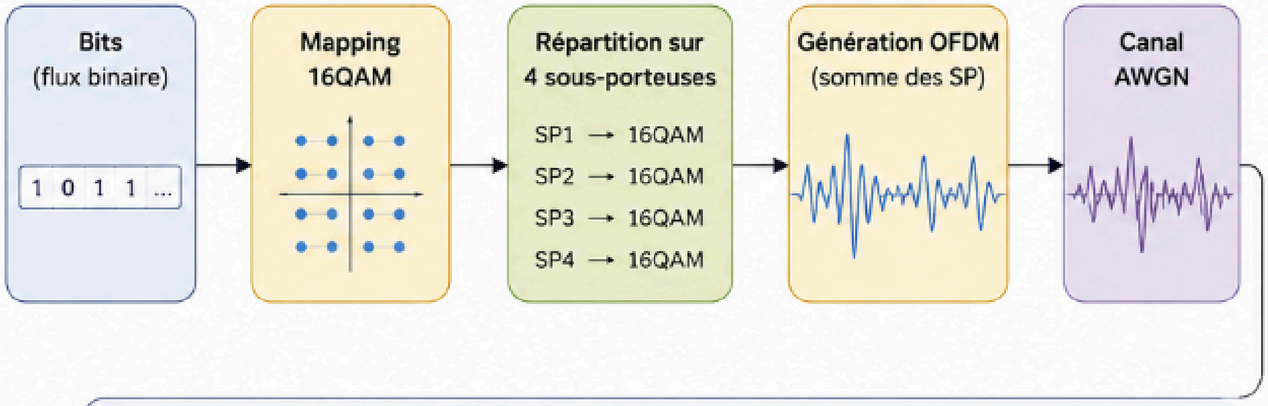


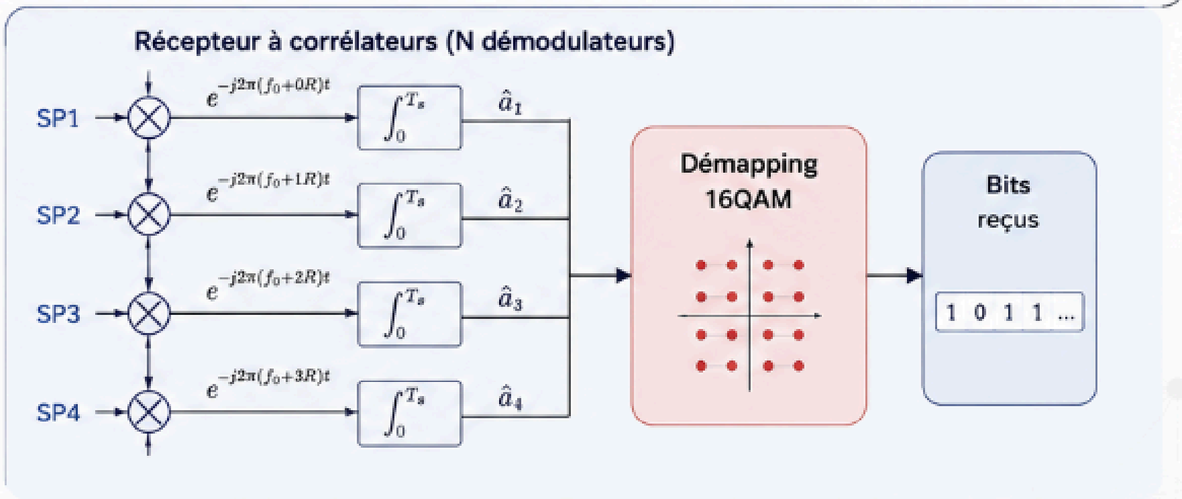
RIGAUX Xavier

OBJECTIF

- **Comprendre** le principe d'une transmission OFDM
- **Répartir** un flux binaire sur 4 sous-porteuses orthogonales
- **Mapper** les données en 16QAM sur chaque sous-porteuse
- **Générer** le signal OFDM par somme des sous-porteuses modulées
- **Ajouter** un canal bruité AWGN (Additive White Gaussian Noise)
- **Récupérer** les symboles avec un récepteur à corrélateurs
- **Observer** l'impact d'un défaut d'orthogonalité entre sous-porteuses

Chaîne de transmission OFDM (N = 4 sous-porteuses)

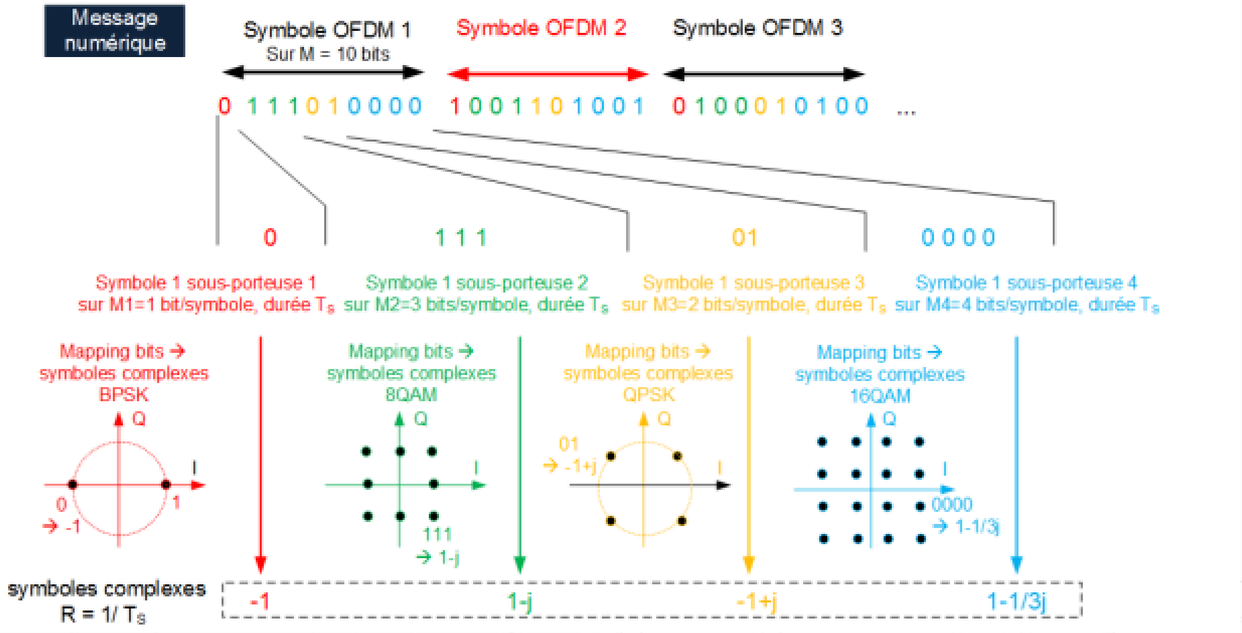




Condition d'orthogonalité : $\Delta f = R = \frac{1}{T_s}$ (R : rapidité de modulation)

ÉMETTEUR

schéma bloc simplifié



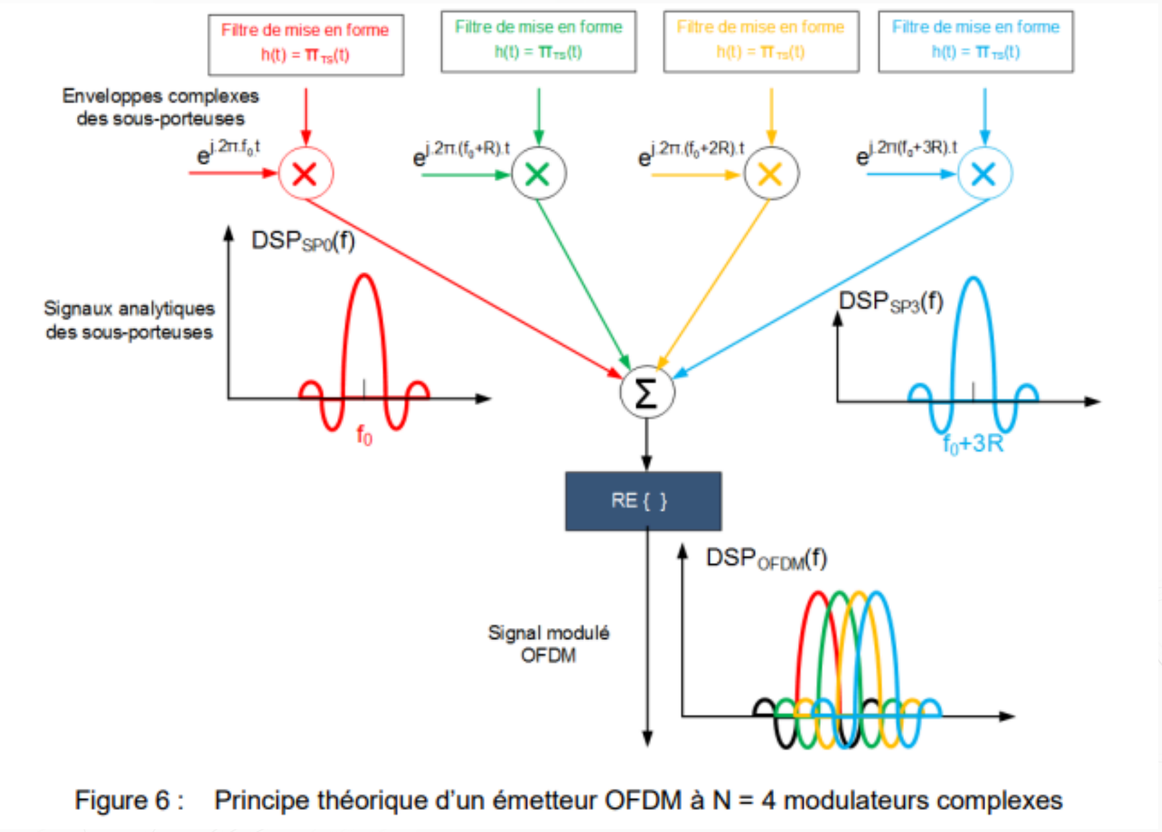


Figure 6 : Principe théorique d'un émetteur OFDM à N = 4 modulateurs complexes

ÉMETTEUR

Mapping 16QAM et mise en parallèle

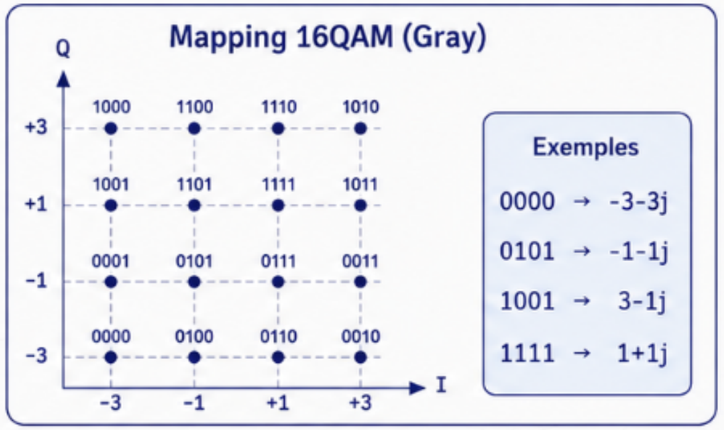
On regroupe les bits par paquets de 4 bits car chaque sous-porteuse utilise une modulation 16QAM.

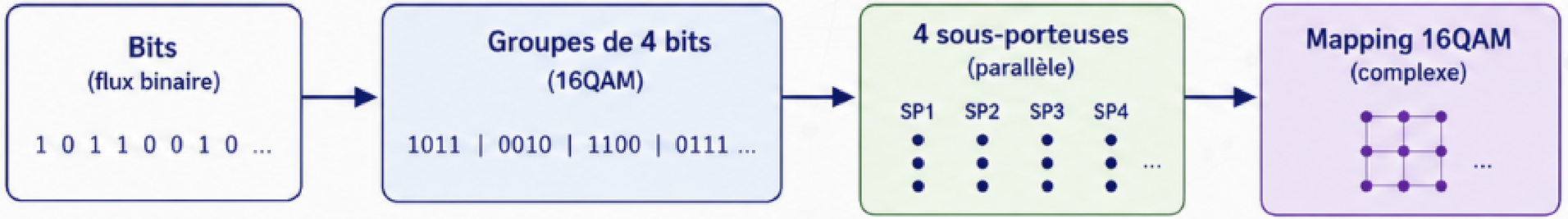
Avec 4 sous-porteuses, un symbole OFDM transporte donc 16 bits.

Les bits sont ensuite réorganisés en parallèle avant le mapping complexe.

```python
# Réorganisation des bits : en parallèle (4 sous-porteuses)
symbs_num_para = bits.reshape(nb_symb_ofdm, nb_sp, bits_par_symb_sp)
```

```python
# Mapping 16QAM sur chaque sous-porteuse
symbs_mod_para = Ofdm.mapping(symbs_num_para, mapping_table)
```



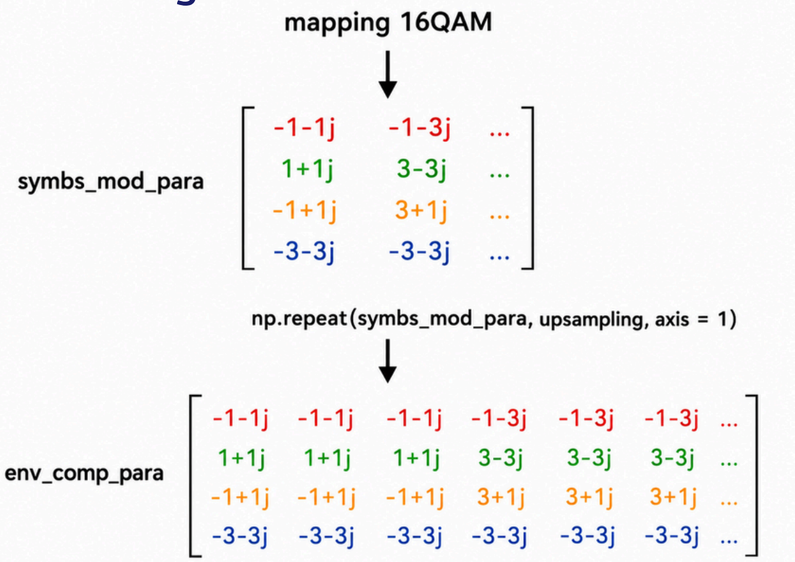


ÉMETTEUR

Filtre de mise en forme rectangulaire

```
env_comp_para = np.repeat(
    symbs_mod_para,
    upsampling,
    axis=1
)

N = env_comp_para.shape[1]
```



En Théorie :

$$x_{EC,n}(t) = \sum_k a_{k,n}\, h(t - kT_s)$$

$$h(t) = \Pi_{T_s}(t)$$

ÉMETTEUR

Génération des sous-porteuses OFDM

```
N = env_comp_para.shape[1]
# 1) Signaux analytiques de chaque sous-porteuse
exp_para = Ofdm.exp_para(nb_sp, f0, df, te, N)
sig_ana_sp = env_comp_para * exp_para
# 2) Signal analytique OFDM = somme des sous-porteuses
signal_ana_OFDM = np.sum(sig_ana_sp, axis=0)
# 3) Signal OFDM réel transmis
signal_OFDM = np.real(signal_ana_OFDM)
```

$$x_{A,n}(t) = x_{EC,n}(t)\, e^{j2\pi(f_0+n\Delta f)t}$$

$$x_{OFDM}(t) = \mathrm{Re}\left\{\sum_{n=0}^{N-1} x_{A,n}(t)\right\}$$

$$\Delta f = R = \frac{1}{T_s}$$

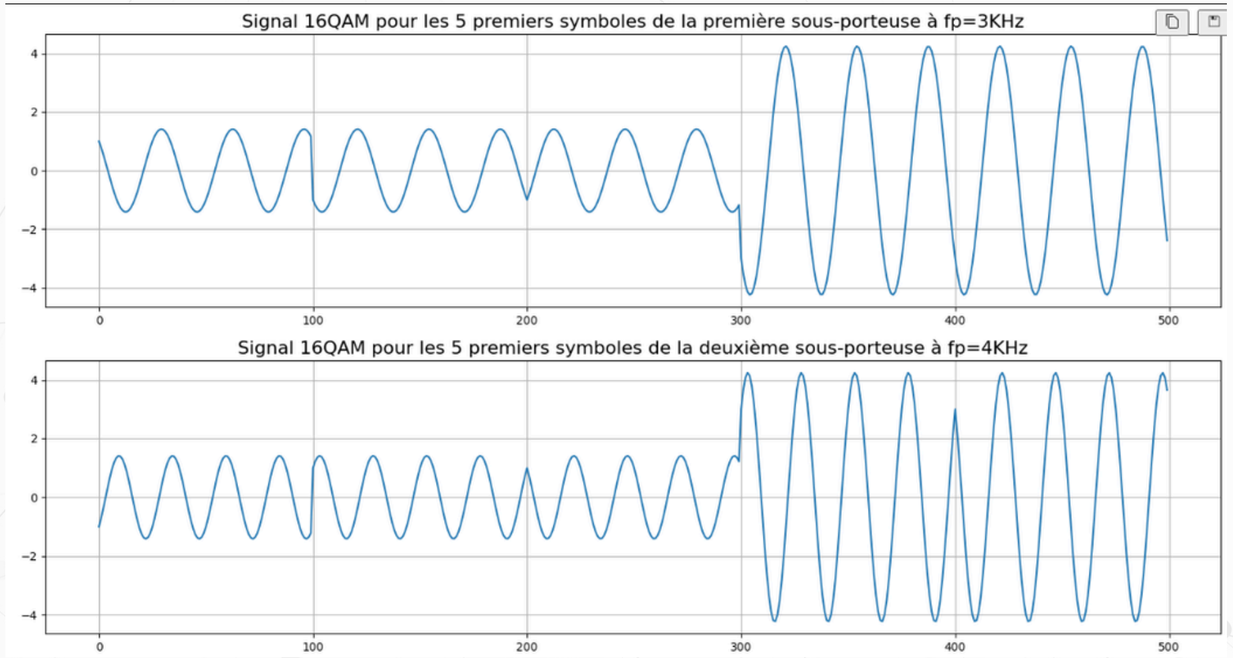
Chaque sous-porteuse est générée par multiplication
avec une exponentielle complexe.

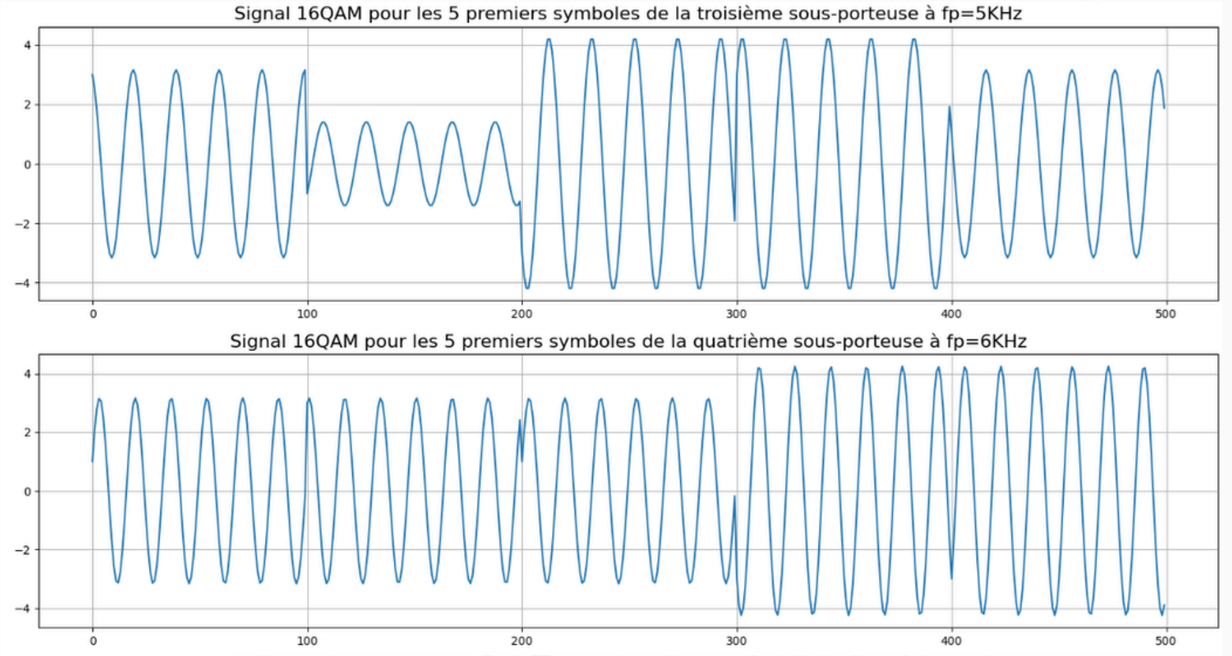
Les fréquences utilisées sont :
3 kHz, 4 kHz, 5 kHz et 6 kHz.

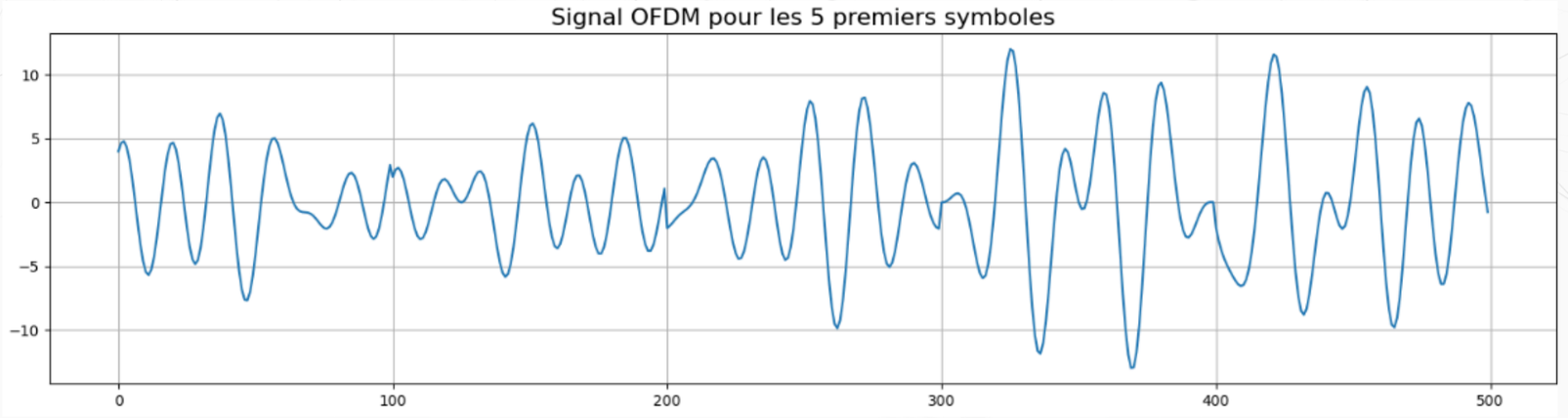
Les 4 sous-porteuses sont ensuite sommées
pour former le signal OFDM.

ÉMETTEUR

Visualisation temporelle des sous-porteuses

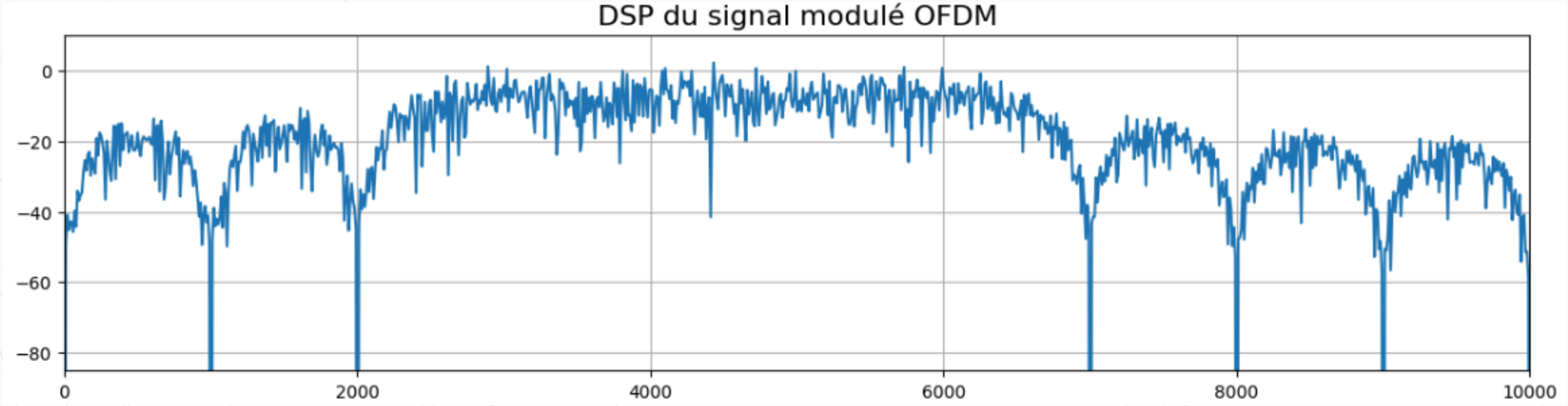






ÉMETTEUR

Analyse spectrale des sous-porteuses



La DSP permet d'observer la répartition fréquentielle du signal OFDM.

Les sous-porteuses sont espacées de :

$$\Delta f = R = \frac{1}{T_s}$$

df = R = 1000 Hz

Cet espacement permet de conserver l'orthogonalité entre les sous-porteuses.

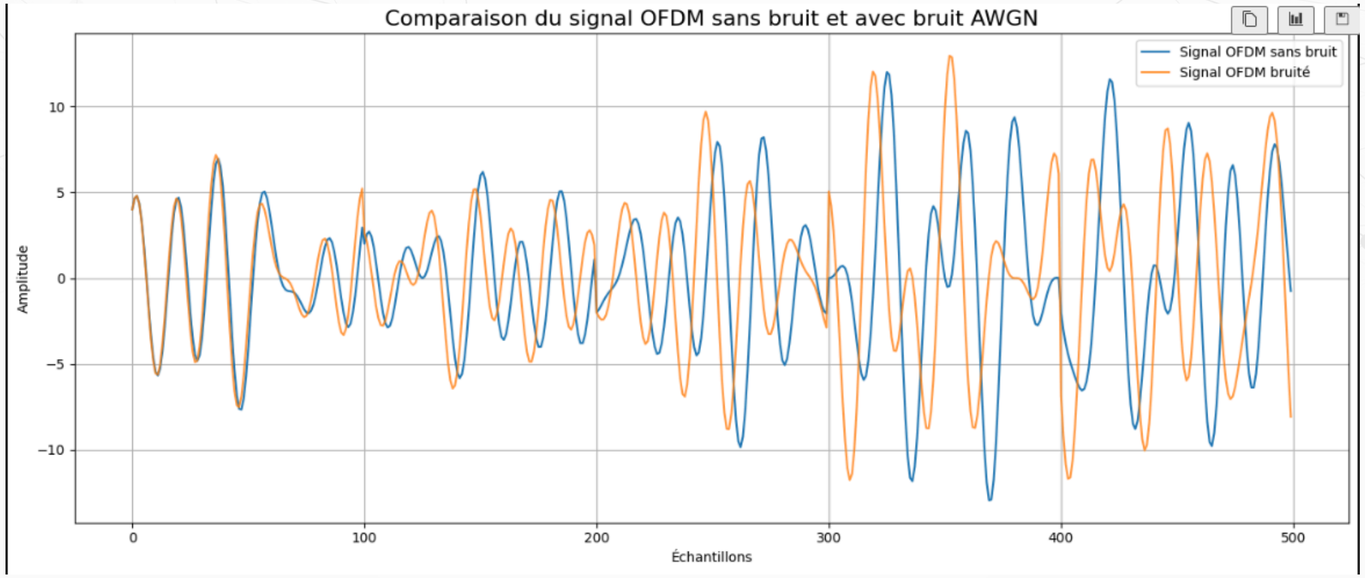
CANAL AWGN

Simulation bruit blanc

```python
@staticmethod
def awgn(signal, mean, std) :
    num_samples = len(signal)
    noise = np.random.normal(mean, std, size=num_samples)
    signal_bruite=signal+noise
    return(signal_bruite)
```

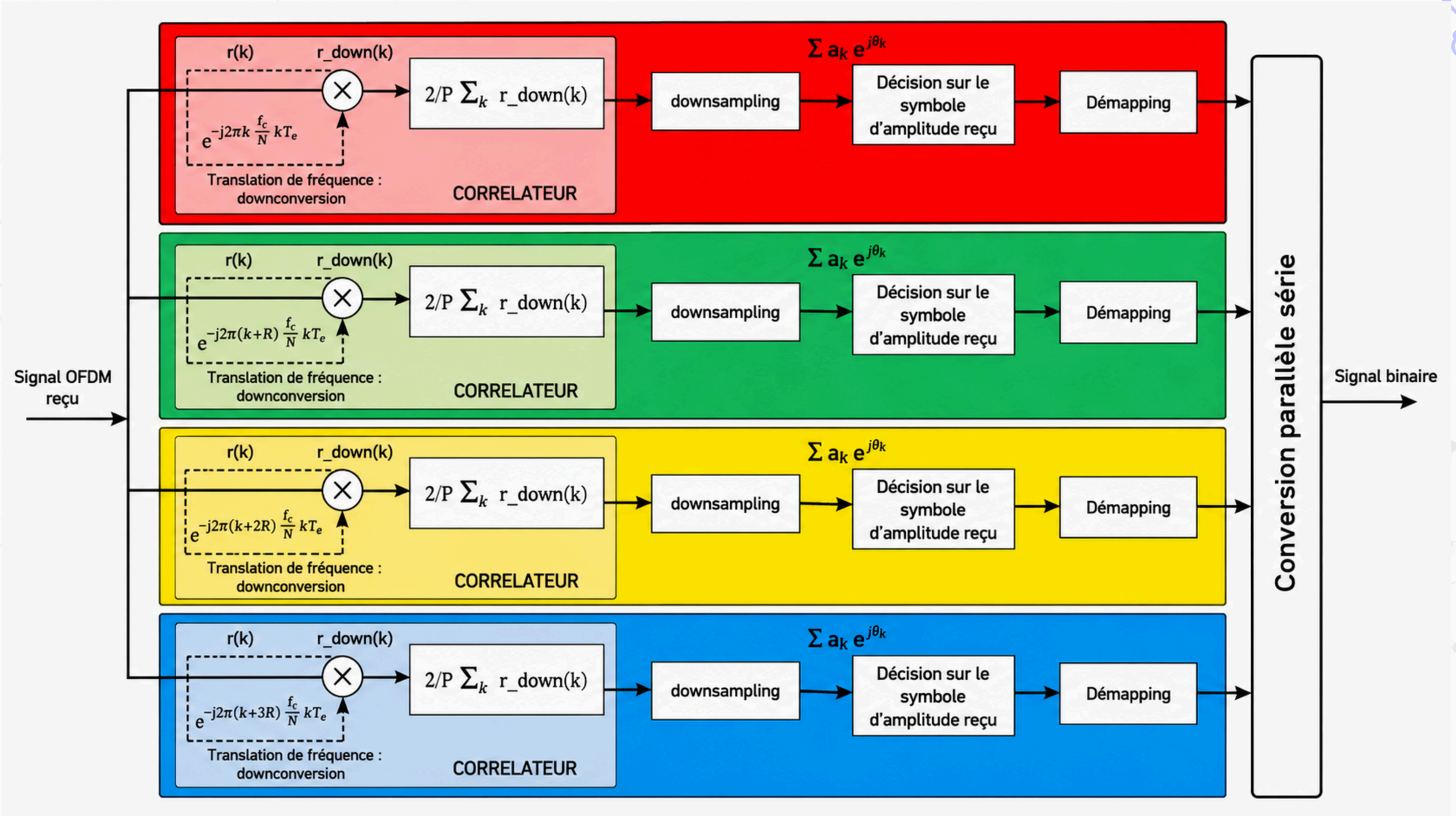
```python
signal_OFDM_bruit = mycanal.awgn(signal_OFDM, 0, 0.4)
```

Le canal AWGN ajoute au signal OFDM un bruit aléatoire gaussien.
Dans commNumv4, le paramètre 0.4 correspond à l'écart-type du bruit.



RÉCEPTEUR

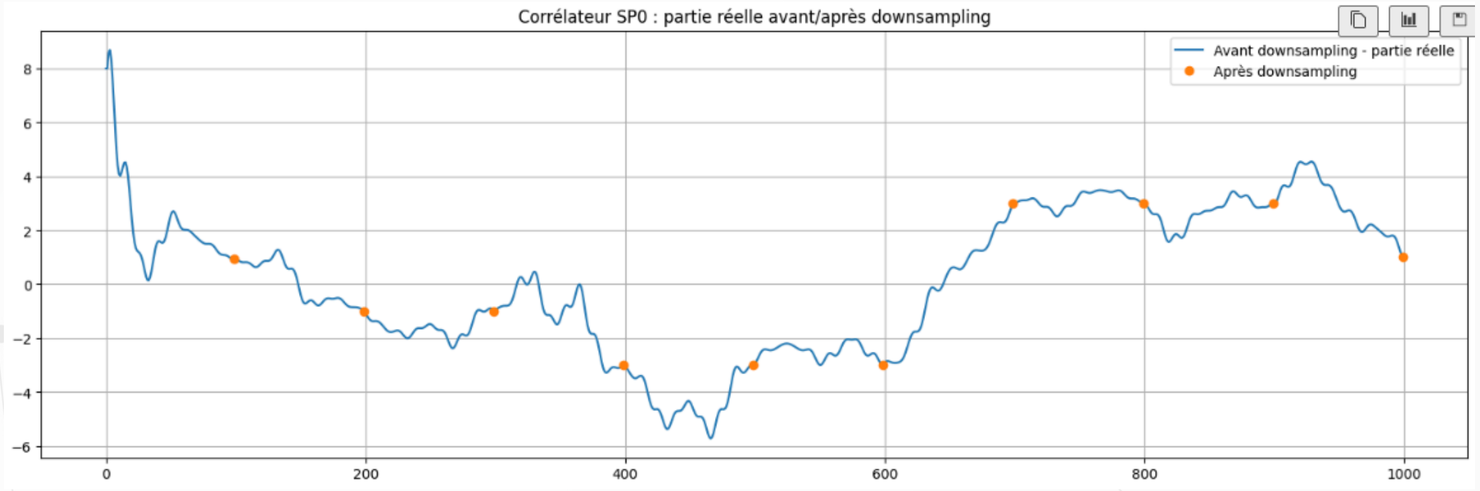
schéma bloc simplifié

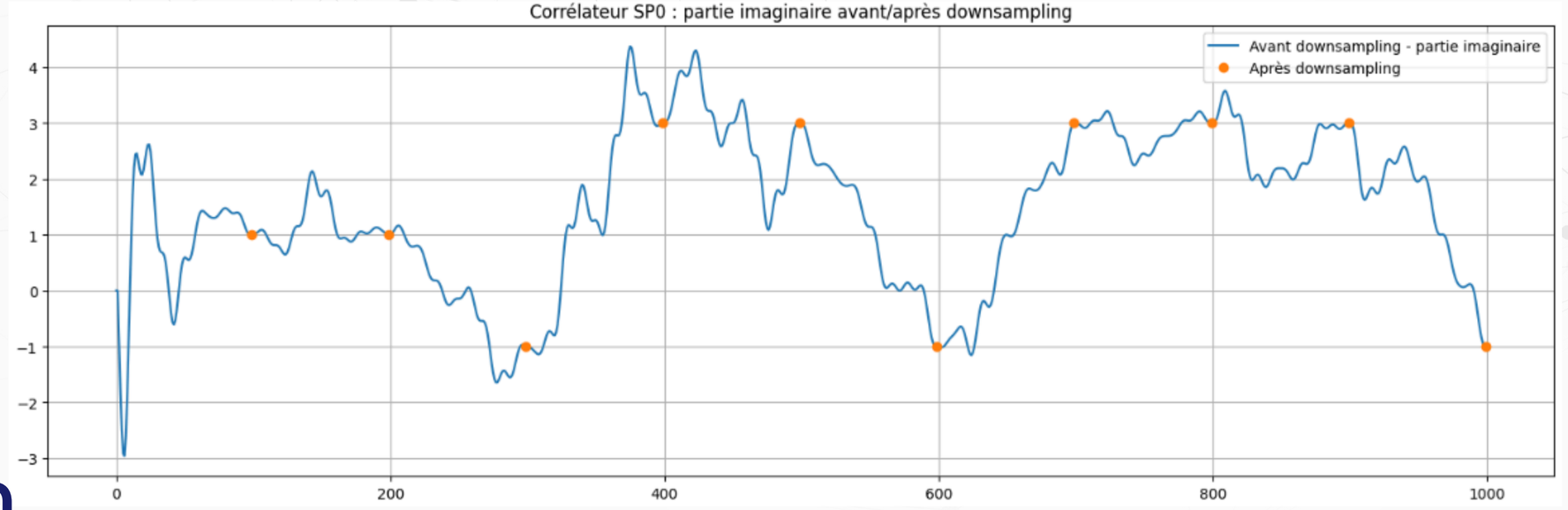


RÉCEPTEUR

Corrélateur sur la première sous-porteuse

```python
fp0 = f0
decalage = upsampling - 1
signal_OFDM_down_sp0 = signal_OFDM * Ofdm.exp_comp(
    -fp0,
    te,
    len(signal_OFDM)
)
env_comp_rcv_sp0 = 2 * Ofdm.moy_glissante(
    signal_OFDM_down_sp0,
    upsampling
)
```





$$r_{down,0}(t) = r(t)\, e^{-j2\pi f_0 t}$$

$$\hat{x}_0(t) = 2 \cdot \frac{1}{P} \sum r_{down,0}(k)$$

RÉCEPTEUR

Boucle sur les 4 sous-porteuses

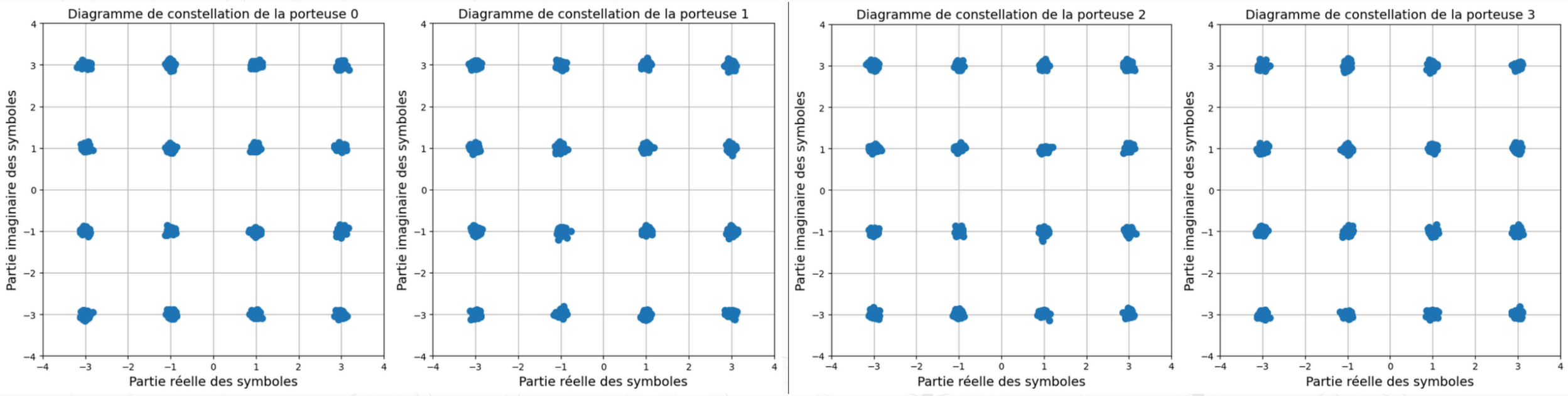
```
for i in range(nb_sp):
    fp_i = f0 + i * df
    signal_OFDM_down = signal_OFDM_bruit * Ofdm.exp_comp(-fp_i, te, len(signal_OFDM_bruit))
    env_comp_rcv = 2 * Ofdm.moy_glissante(signal_OFDM_down, upsampling)
    symbs_mod_para[i, :] = Ofdm.downsample(env_comp_rcv, upsampling, decalage)
    symbs_detect = Ofdm.detection(symbs_mod_para[i, :], mapping_table)
    bits_rcv_para[i, :] = Ofdm.demapping(symbs_detect, mapping_table)
```

Le même traitement est appliqué aux 4 sous-porteuses.

Pour chaque sous-porteuse :
downconversion → moyenne glissante → downsampling
→ détection 16QAM → démapping

RÉCEPTEUR

Constellations des sous-porteuses



Après downconversion, moyenne glissante et downsampling,
on récupère les symboles complexes de chaque sous-porteuse.

Les points forment une constellation 16QAM.

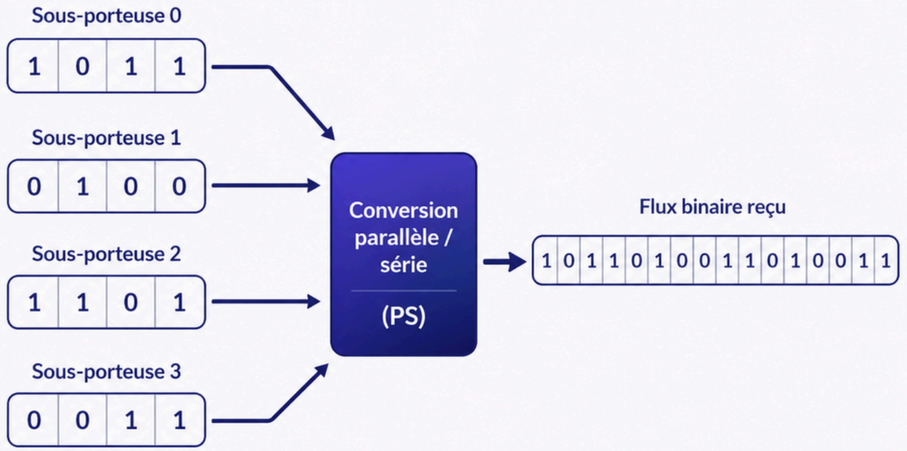
Le bruit AWGN crée un léger nuage autour des positions idéales.

RÉCEPTEUR

Conversion parallèle / série

```python
@staticmethod
def PS(bits_rcv_para, nb_sp, nb_symb_ofdm, bits_par_symb_sp):
    bits_rcv = np.empty((nb_symb_ofdm*nb_sp, bits_par_symb_sp),dtype=int)
    for i in range(nb_symb_ofdm):
        for j in range(nb_sp):
            bits_rcv[i*nb_sp+j,] = bits_rcv_para[j,i*bits_par_symb_sp:(i+1)*bits_par_symb_sp]
    return(bits_rcv.ravel())
```

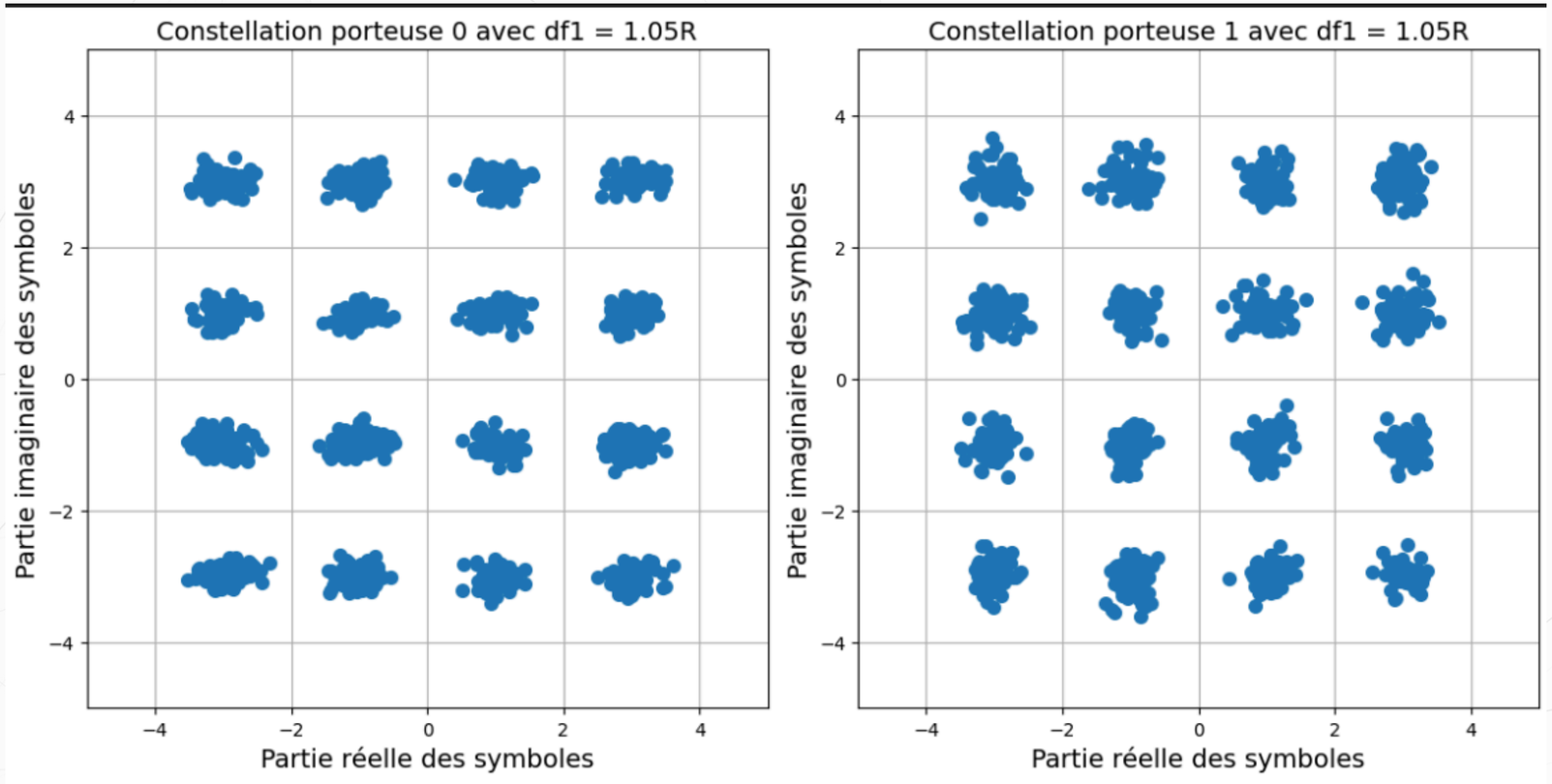
```python
bits_rcv = Ofdm.PS(
    bits_rcv_para,
    nb_sp,
    nb_symb_ofdm,
    bits_par_symb_sp
)
```



Réorganisation des bits reçus par sous-porteuse en un vecteur unique

RÉCEPTEUR

Impact d'un défaut d'orthogonalité



Condition normale :
df = R = 1000 Hz

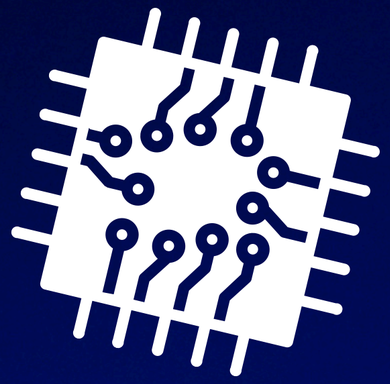
Test réalisé :
df1 = 1.05 × R = 1050 Hz

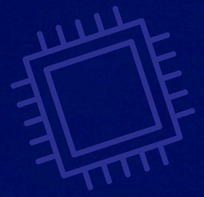
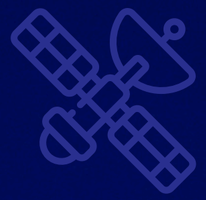
Conséquence :
les sous-porteuses ne sont plus parfaitement orthogonales.

$$\Delta f = R = \frac{1}{T_s}$$

Les points de constellation s'étalent :
chaque corrélateur récupère une partie des autres sous-porteuses.

$$\Delta f \neq R \Rightarrow \text{perte d'orthogonalité}$$



MERCI