LELEC1930 - Introduction aux télécommunications
Séance 4 - Implémentation Python

Prof. : Jérôme Louveaux Assist. : Jérome Eertmans

Rappel

En télécommunications, la majorité, si pas l'entièreté, des calculs sont effectués sur un ordinateur. Le but de cette séance est donc de vous familiariser avec l'implémentation de concepts vus au cours à l'aide du langage de programmation `Python`.

Afin de pouvoir mener à bien cette séance, on vous demandera d'**installer les packages** `numpy` et `scipy`. En plus de ces derniers, il vous est conseillé d'installer les packages `matplotlib` et `notebook`. Le premier permet de tracer des graphiques scientifiques très aisément, tandis que le second offre un environnement de programmation propice aux tests.

En bref, vous pouvez tout installer via une seule commande dans votre terminal : `pip install numpy scipy matplotlib notebook`.

Note : *La plupart de ces packages sont installés par défaut si vous utilisez Anaconda.*

```python
import numpy as np
import matplotlib.pyplot as plt

# Exemple de plot

x = np.linspace(-5, 5, 25)  # 25 valeurs entre -5 et 5 (compris)
y = x ** 2

plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Une jolie parabole")
plt.show()  # Affiche le graphique, non nécessaire dans un Notebook
```

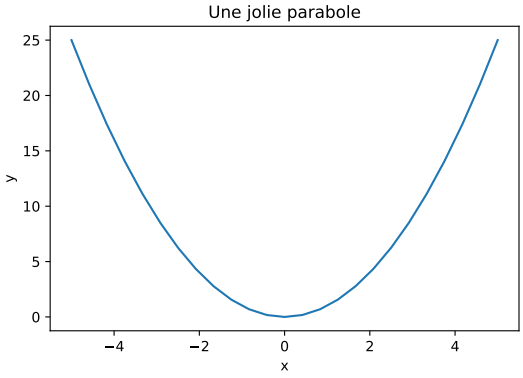

FIGURE 1 – Résultat obtenu avec le code ci-dessus.

Pour toute autre question concernant le code, ne l'oubliez jamais : Google est votre ami ! L'assistant aussi par ailleurs ;-)

Exercices

Exercice 1 : Calcul du BER d'un QAM-4

L'objectif de cet exercice est de réussir à calculer le Bit Error Rate (BER), taux d'erreur en français, d'un signal et de voir son évolution en fonction du niveau de bruit, ou plutôt du Signal to Noise Ratio SNR.
On considère donc la transmission d'une séquence de $N = 1 \times 10^6$ symboles QAM-4 :

$$x[k] = \pm 1 \pm j, \tag{1}$$

avec $k = 0, ..., N$.
Lors de l'envoi, le canal de transmission ajoute un bruit sur le signal de telle manière que le signal reçu, y, soit de la forme :

$$y[k] = x[k] + n[k], \tag{2}$$

avec $n[k]$ le bruit à l'instant k.
Ici, on modélise le bruit comme étant un bruit blanc gaussien, c.-à-.d une variable aléatoire suivant une distribution normale de moyenne nulle. Les symboles étant complexes, le bruit l'est tout autant. On notera donc σ_n^2 la variance du bruit complexe, et $\sigma_n^2/2$ la variance des parties réelles et imaginaires.
Au récepteur, chaque symbole est décodé comme étant celui le plus proche de $y[k]$. On notera l'estimation du signal original $\hat{x}[k]$. Dans un monde idéal, sans bruit, on aura donc que $x[k] = \hat{x}[k]$ pour tout k. Le BER du signal est donc simplement le nombre de symboles erronés dans $\hat{x}[k]$ sur le nombre total de symboles.
Finalement, comme la constellation a une variance de 2, on définit le SNR comme :

$$\text{SNR} = \frac{2}{\sigma_n^2}. \tag{3}$$

1. Générez la séquence $x[k]$ de manière aléatoire (voir `numpy.random.randint`).
2. Générez la séquence $n[k]$ de bruit avec $\sigma_n^2 = 1 \times 10^{-2}$ (voir `numpy.random.randn`).
3. Calculez les échantillons $y[k]$ et afficher les sur le diagramme de la constellation.
4. Estimez le symbole d'origine $\hat{x}[k]$ sur base de cet échantillon.
5. Calculez le BER.
6. Refaites toutes ces étapes pour différents niveaux de bruits (conseil : utilisez `np.logspace(-2, 1, 25)`).
7. Tracez la courbe du BER versus le SNR (conseil : utilisez `plt.semilogy(10 * np.log10(SNR), BER)`) et vérifiez que vous obtenez des résultats cohérents.

Exercice 2 : Transformée en cosinus discrète

La transformée en cosinus discrète (DCT) est une opération mathématique sans perte qui permet d'exprimer un signal en fonction de son contenu fréquentiel. Similaire à la transformée de Fourier,

la DCT est très utilisée lors de la compression d'images (exemple : JPEG).
Ici, vous allez travailler avec des images de taille 8×8. Comme les images sont des matrices de pixels, et non de vecteurs, il faut appliquer deux fois la DCT : une fois pour les lignes, une fois pour les colonnes. On appellera cette fonction `dct2` et son inverse `idct2`. Pour une matrice $N \times N$, on définit la DCT2 de X comme :

$$Y(u,v) = \frac{2C(u)C(v)}{N} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} X(m,n) \cos\frac{(2m+1)u\pi}{2N}) \cos\frac{(2v+1)v\pi}{2N}, \tag{4}$$

avec $C(0) = 1/\sqrt{2}$ et $C(k) = 1$ pour $1 \leq k \leq N-1$.

```python
import numpy as np
from scipy.fftpack import dct, idct

def dct2(X):
    return dct(dct(X.T, norm="ortho").T, norm="ortho")

def idct2(Y):
    return idct(idct(Y.T, norm="ortho").T, norm="ortho")

blocks = np.load("seance4.npy", allow_pickle=True).item()
```

Le code ci-dessus montre comment définir la DCT sur une matrice à partir de la DCT sur un vecteur. La variable `blocks` est un dictionnaire contenant toutes les matrices nécessaires à l'exercice. Vous trouverez le fichier `seance4.npy` sur Moodle.
Pour finir, vous pouvez afficher le contenu d'une matrice de la manière suivante.

```python
import matplotlib.pyplot as plt

def plot_image(X, ax=None):
    if ax is None:  # Si on ne fournit pas d'axes, on en crée
        _, ax = plt.subplots()

    im = ax.imshow(X, origin="lower")
    bar = plt.colorbar(im, ax=ax)

    ax.set_xlabel("m / u")
    ax.set_ylabel("n / v")
    bar.set_label("Valeur pixel")
```

1. Observez les trois premiers blocs (`"img1"`, `"img2"` et `"img3"`) et calculez les valeurs de leurs coefficients DCT. Essayez d'interpréter les résultats.
2. On observe que le coefficient principal est toujours non nul. À quoi correspond-il ? Que faudrait-il faire pour obtenir un coefficient principal nul ?
3. Observez les blocs 4 et 5, qui sont définis entre -1 et 1, et leur DCT. Interprétez les résultats. Puis comparez avec le bloc 6 (défini entre 0 et 1).
4. Observez les blocs 7 et 8, et tentez de prédire quels coefficients DCT seront majoritairement présents.
5. Observez les valeurs du bloc 9, et tentez de prédire les valeurs exactes des coefficients DCT de celui-ci, à partir des résultats obtenus pour les blocs 1 et 3.

6. Finalement, vous pouvez utiliser le bloc 10 pour faire un petit test de compression. La matrice `"Q"` fournie contient les coefficients de quantification habituels en JPEG. Ils sont calibrés pour des valeurs de pixels entre 0 et 256, vous utiliserez donc plutôt la matrice `Q2 = Q / 256`.
   (a) Appliquez la DCT sur le bloc 10.
   (b) Appliquez la quantification en divisant les coefficients par ceux de `Q2`, puis arrondissez à l'entier le plus proche. Observez les coefficients restants.
   (c) Retrouvez les coefficients compressés en re-multipliant par `Q2`, élément par élément.
   (d) Appliquez la DCT inverse et observer la nouvelle image obtenue. Comparez-la en observant les deux images côte à côte.
   (e) Vous pouvez également tester des quantifications plus ou moins fortes en multipliant toute la matrice `Q2` par une constante.
   (f) Si la DCT est une transformation sans perte, pourquoi la compression JPEG, qui utilise la DCT, est-elle une transformation **avec perte** ?