

Capacité numérique n° 2

Transformée de Fourier discrète

Le fichier FFT.py propose deux procédures.

Procédure spectre

Retourne la représentation de la transformée de Fourier discrète d'un signal calculée par l'algorithme de la transformée de Fourier rapide.

```

1 def spectre(S,F_ech,duree,*args):
2     t = np.arange(0,duree,1/F_ech)           # tableau des instants d'échantillonnage
3     ech = S(t)                               # on construit l'échantillon
4     tfd = fft(ech)                           # on calcule le spectre par FFT
5     N = len(tfd)                             # on construit le tableau des fréquences
6     freq = np.zeros(N)
7     for k in range(N):
8         freq[k] = 1./duree*k
9     spectre = np.absolute(tfd)*2./N          # composantes du spectre, en normalisant
10    spectre[0] = .5*spectre[0]                # normalisation de la composante continue
11    if len(args)>0:                            # argument optionnel : on trace le spectre
12        f_max = args[0]                       # jusqu'à la fréquence f_max
13    else:
14        f_max = F_ech
15    N_max = int(N*f_max/F_ech)
16    plt.stem(freq[:N_max],spectre[:N_max],markerfmt=".",basefmt=" ") # tracé du spectre
17    plt.xlabel("fréquence")
18    plt.ylabel("amplitude")
19    plt.title("spectre calculé par FFT avec  $F_{\mathrm{éch}}=\$"+str(F_ech))
20    plt.show()$ 
```

Les arguments sont :

s fonction définissant le signal.

F_ech fréquence d'échantillonnage.

duree durée totale de l'acquisition.

F_max (optionnel) fréquence maximale de tracé du spectre (sur $[0, F_{\max}]$).

Procédure spectre_fenetre

Même principe que la procédure précédente, mais en appliquant un fenêtrage.

```

1 def spectre_fenetre(signal,F_ech,duree,fenetre,*args):
2     t = np.arange(0,duree,1/F_ech)           # on construit le tableau des instants d'échantillonnage
3     ech = signal(t)                          # on construit l'échantillon
4     M = len(ech)                             # nombre de points de l'échantillon
5     if fenetre == 1:
6         window = np.hamming(M)               # fenêtre de Hamming
7         nom = "avec fenêtre de Hamming"
8     elif fenetre == 2:
9         window = np.hanning(M)              # fenêtre de Hanning
10        nom = "avec fenêtre de Hanning"
11    elif fenetre == 3:
12        window = np.blackman(M)             # fenêtre de Blackman
13        nom = "avec fenêtre de Blackman"
14    elif fenetre == 4:
15        window = np.bartlett(M)             # fenêtre de Bartlett
16        nom = "avec fenêtre de Bartlett"
17    elif fenetre == 5:
18        window = np.kaiser(M,14)           # fenêtre de Kaiser (coeff. modifiable)
19        nom = "avec fenêtre de Kaiser"
20    else:
21        window = 1.

```

```

22     nom = "sans fenêtre"
23     ech_fenetre = window*ech
24     tfd = fft(ech_fenetre)           # on calcule le spectre par FFT
25     N = len(tfd)                    # on construit le tableau des fréquences
26     freq = np.zeros(N)
27     for k in range(N):
28         freq[k] = 1./duree*k
29     spectre = np.absolute(tfd)*2./N # norme des composantes du spectre, en normalisant
30     spectre[0] = .5*spectre[0]      # normalisation de la composante continue
31     if len(args)>0:                  # argument optionnel : on trace le spectre
32         f_max = args[0]             # jusqu'à la fréquence f_max
33     else:
34         f_max = F_ech
35     N_max = int(N*f_max/F_ech)
36     plt.stem(freq[:N_max],spectre[:N_max],markerfmt=".",basefmt=" ") # tracé du spectre
37     plt.xlabel("fréquence")
38     plt.ylabel("amplitude")
39     plt.title("spectre calculé par FFT "+str(nom))
40     plt.show()

```

s fonction définissant le signal.

F_ech fréquence d'échantillonnage.

duree durée totale de l'acquisition.

N type de fenêtre

F_max (optionnel) fréquence maximale de tracé du spectre (sur $[0, F_{\max}]$).

Le type de fenêtre est sélectionné par un chiffre :

1 : fenêtre de Hamming

2 : fenêtre de Hanning

3 : fenêtre de Blackman

4 : fenêtre de Bartlett

5 : fenêtre de Kaiser

autre valeur : sans fenêtre

Signaux prédéfinis

Quelques signaux usuels ont été prédéfinis :

```

f = 1      # fréquence du signal

def sinus(t):
    return np.sin(2. * np.pi * f * t)

def triangle(t):
    return signal.sawtooth(2. * np.pi * f * t,0.5)

def dent_scie(t):
    return signal.sawtooth(2. * np.pi * f * t,1)

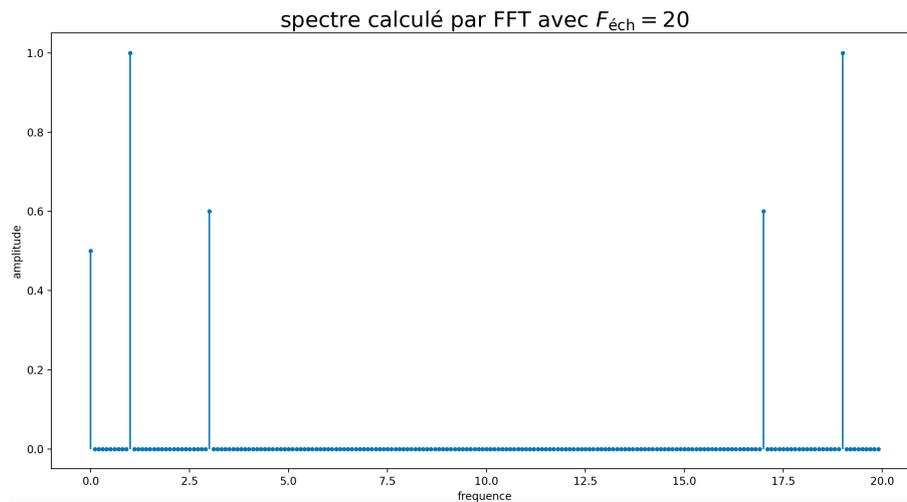
def carre(t):
    return signal.square(2. * np.pi * f * t,.5)

```

► On pourra modifier la valeur .5 de la fonction `carre` pour obtenir un signal créneau de rapport cyclique différent.

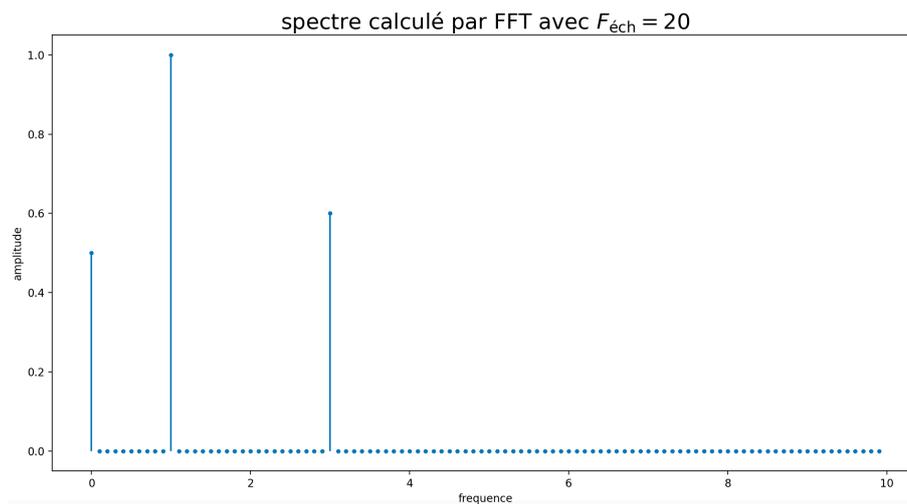
Exemples d'utilisation

spectre(s,20,10)



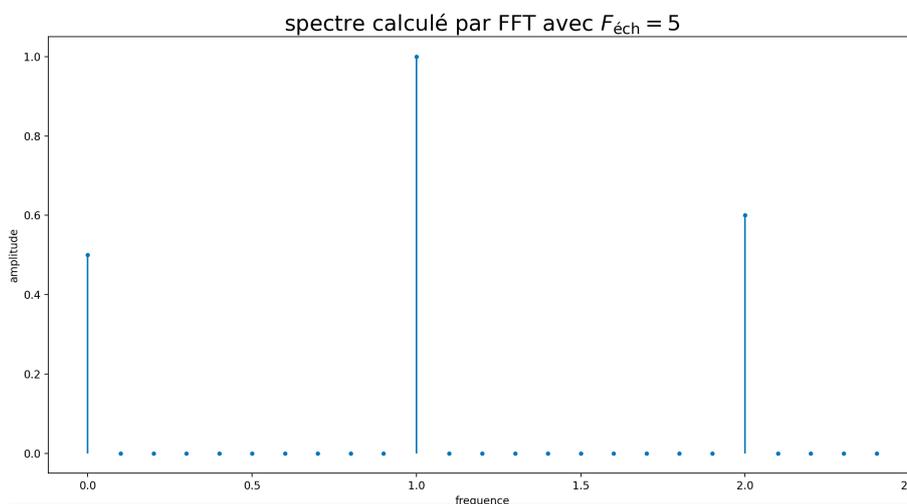
Représentation de la partie utile du spectre ($[0, F_e/2]$) :

spectre(s,20,10,10)



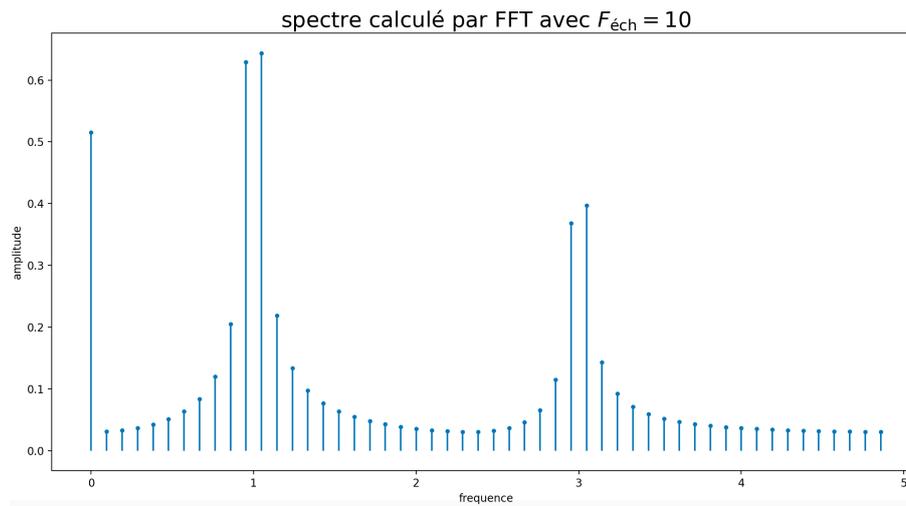
Ici la condition de Shannon n'est pas respectée : on observe le phénomène de repliement de spectre (apparition de la fréquence 2 absente du signal initial).

spectre(s,5,10,2.5)



Cas où la durée de l'échantillon n'est pas un multiple de la période du signal :

`spectre(s,10,10.5,5) # $F_{ech} = 10$, tracé sur $[0, F_{ech}/2]$`



Application d'un fenêtre de Hamming.

`spectre_fenetre(s,10,10.5,1,5)`

