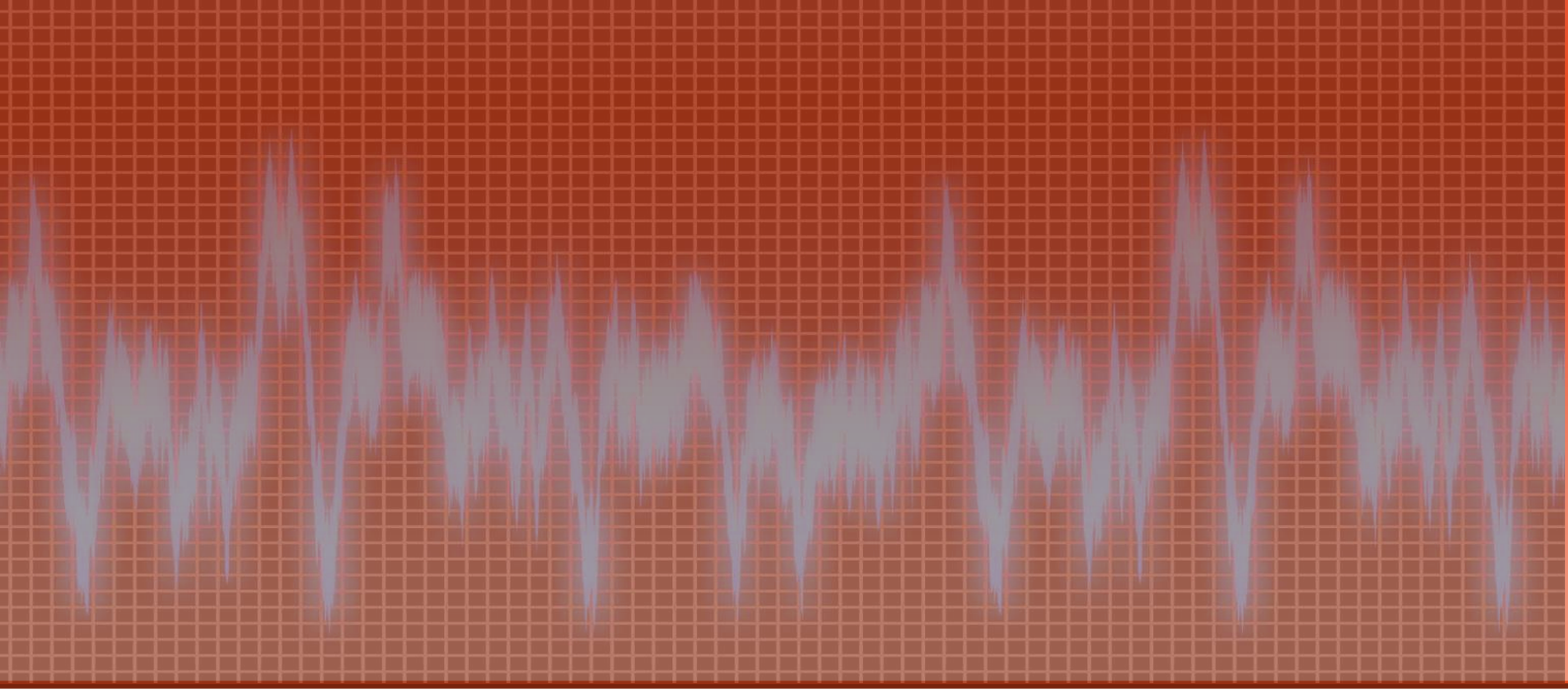


Compte Rendu TP2

Conversion Analogique-Numérique Audio

Réalisé par : Hachem Squalli ElHoussaini N°29

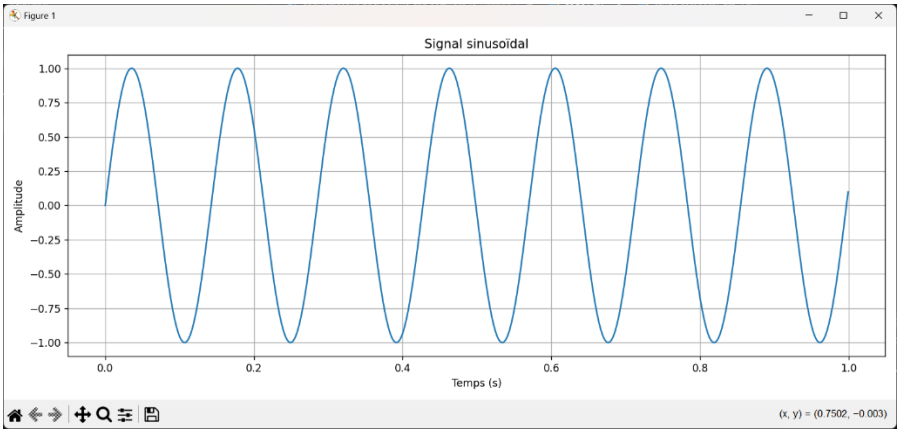
Dirigé par : Pr. H. TOUZANI

LAB 1 :

Exercice :

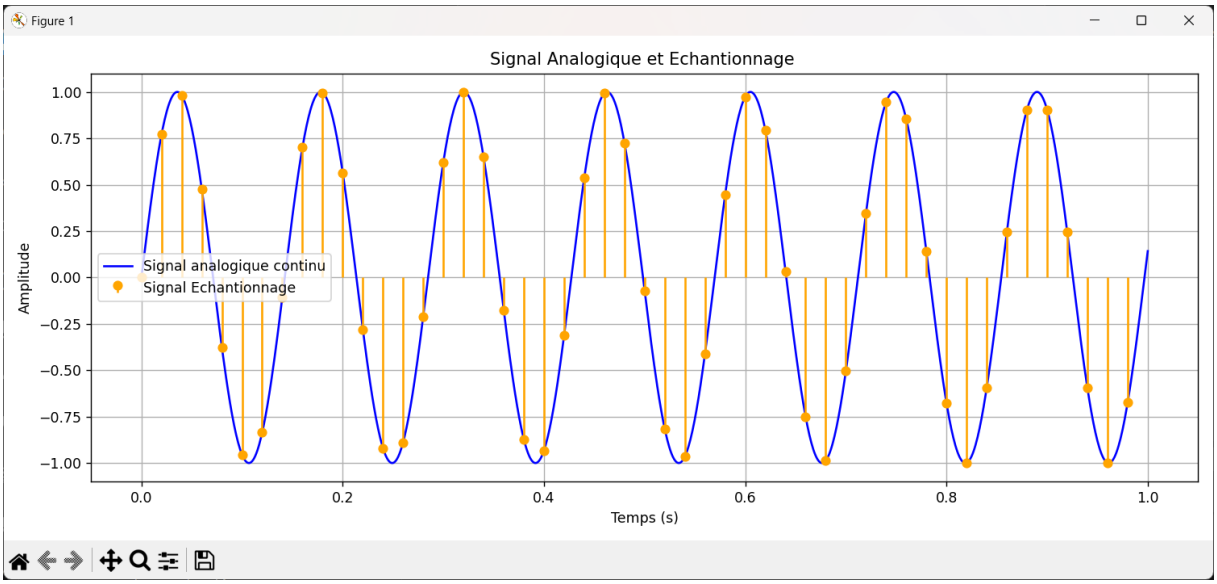
1. Génération et visualisation d'un signal sinusoïdal (5 Hz, échantillonné à 1 kHz).

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter


f = 5
Fe = 1000
Te = 1/Fe
t = np.arange(0,1,Te)

signal = np.sin(2**np.pi*f*t)


# Affichage
plt.figure(figsize=(12, 5))
plt.plot(t, signal)
plt.xlabel('Temps (s)')
plt.ylabel('Amplitude')
plt.title('Signal sinusoïdal')
plt.grid(True)

plt.tight_layout()
plt.show()
```



2. Visualisation d'un signal sinusoïdal analogique (5Hz) et de son échantillonnage à 50Hz, mettant en évidence l'effet de discrétisation temporelle.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter

# Paramètres
f = 5
Fe = 50 #Frequence d'echantillonnage en Hz sans utilser les comparateurs
Te = 1/Fe #periode d'echantillonnage en s
T=1 #Duree de la signal en s
t = np.arange(0,T,Te) #Temps en s

# Signal analogique
signal_analogique = np.sin(2**np.pi*f*t)
t_fine = np.linspace(0, T, 1000) # Points de temps plus fins pour la courbe
signal_analogique_fine = np.sin(2**np.pi*f*t_fine)
# Affichage
plt.figure(figsize=(12, 5))
plt.stem(t_fine, signal_analogique_fine,label='Signal analogique continu',color='blue')
plt.stem(t,signal_analogique,basefmt=" ",linefmt='orange',markerfmt='o',label='Signal Echantionnage')
plt.title("Signal Analogique et Echantionnage ")
plt.grid(True)

plt.xlabel('Temps (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.tight_layout()
plt.show()
```



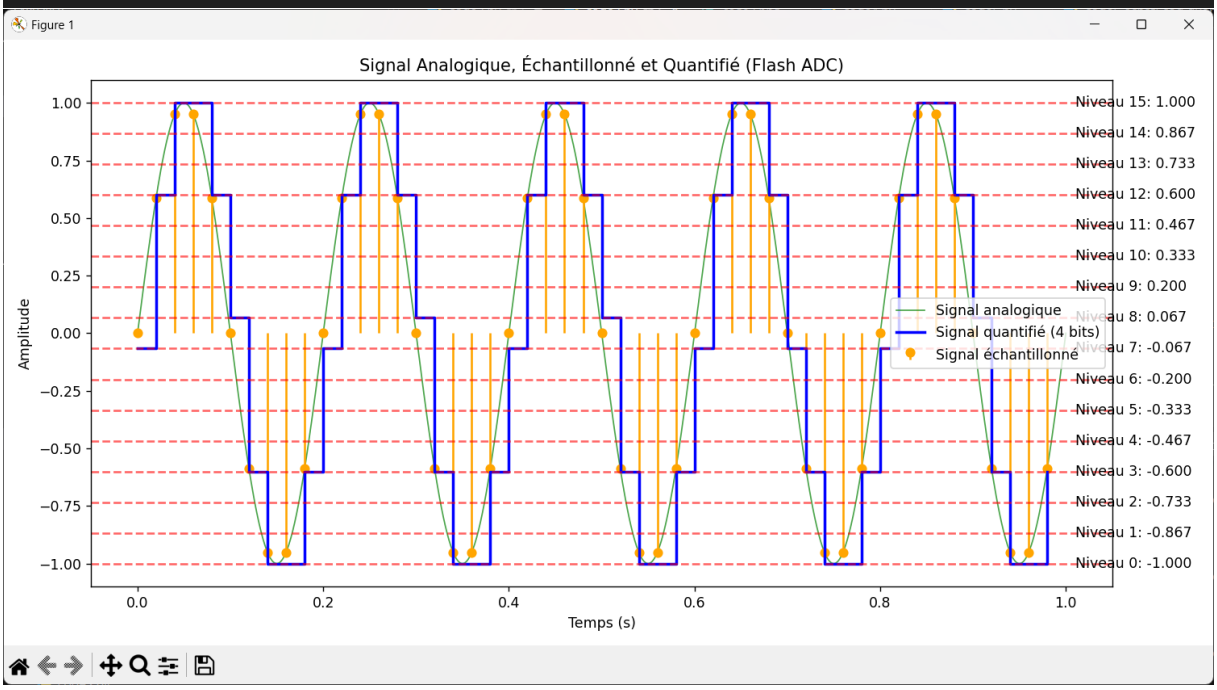
3. Simulation d'un convertisseur Flash ADC (n bits) appliqué à un signal sinusoïdal (5Hz), montrant l'échantillonnage à 50Hz et la quantification avec ses niveaux de référence.
(Processus CAN complet : échantillonnage temporel + quantification amplitude, 8 niveaux de quantification, plage [-1,1])

Points clés :

- Visualisation des trois étapes : signal analogique → échantillonné → quantifié
- Niveaux de quantification matérialisés (lignes rouges)
- Méthode Flash ADC par approximation directe aux niveaux les plus proches

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

# Fonction de quantification suivant la methode CAN Flash
Cody
def quantification_CAN_flash(signal, n_bits):
    # Nombre niveau de quantification
    N_levels = 2**n_bits

    # Plage de signal analogique (supposée entre -1 et 1)
    min_signal = -1
    max_signal = 1

    # Création des niveaux de référence, uniformément espacés
    levels = np.linspace(min_signal, max_signal, N_levels)

    # Quantification
    quantized_signal = np.zeros_like(signal)
    for i, value in enumerate(signal):
        # Trouver l'indice du niveau le plus proche
        index = np.argmin(np.abs(levels - value))
        # Quantification
        quantized_signal[i] = levels[index]

    return quantized_signal, levels
# Paramètres du signal
f = 5
Fe = 50  # Fréquence d'échantillonnage en Hz
Te = 1/Fe  # période d'échantillonnage en s
T = 1  # Durée du signal en s
t = np.arange(0, T, Te)  # Temps discret pour échantillonnage
t_fine = np.linspace(0, T, 1000)  # Temps continu pour signal analogique

# Génération des signaux
signal_analogique = np.sin(2*np.pi*f*t_fine)  # Signal analogique continu
```

```python
# Génération des signaux
signal_analogique = np.sin(2*np.pi*f*t_fine)  # Signal analogique continu
signal_echanti = np.sin(2*np.pi*f*t)  # Signal échantillonné

n_bits = 4
signal_quantifie ,niveaux= quantification_CAN_flash(signal_echanti, n_bits)

# Affichage
plt.figure(figsize=(12, 6))

# Signal analogique
plt.plot(t_fine, signal_analogique, 'g-', label='Signal analogique', linewidth=1, alpha=0.7)

# Signal échantillonné
plt.stem(t, signal_echanti, basefmt=" ", linefmt='orange', markerfmt='o', label='Signal échantillonné')

# Signal quantifié
plt.step(t, signal_quantifie, 'b-', where='post', label=f'Signal quantifié ({n_bits} bits)', linewidth=2)

Cody
for i, niveau in enumerate(niveaux):
    plt.axhline(y=niveau, color='r', linestyle='--', alpha=0.6)
    # Ajouter des étiquettes pour chaque niveau
    plt.text(1.01, niveau, f'Niveau {i}: {niveau:.3f}', verticalalignment='center')


plt.title("Signal Analogique, Échantillonné et Quantifié (Flash ADC)")
plt.xlabel('Temps (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.tight_layout()
plt.show()
```



4. Simulation complète d'un CAN Flash 4 bits : échantillonnage à 100Hz, quantification uniforme et codage binaire des niveaux, avec affichage des signaux et des codes binaires associés.

Points clés :

- Conversion analogique-numérique en 3 étapes : échantillonnage → quantification → codage
- 16 niveaux de quantification (4 bits) sur la plage [-1,1]
- Affichage des codes binaires pour chaque échantillon temporel
- Visualisation des niveaux de référence et de la quantification par seuillage

```python
def codage_binaire(signal_quantifie, levels, n_bits):
    # Créer un dictionnaire pour mapper les valeurs quantifiées aux codes binaires
    level_to_bin = {level: format(i, '0{}b'.format(n_bits)) for i, level in enumerate(levels)}

    # Conversion du niveau de quantification en code binaire
    signal_code_binaire = [level_to_bin[val] for val in signal_quantifie]

    return signal_code_binaire
```

```python
for i in range(len(t)):
    print("Temps : {:.2f}, signal_quantifie : {:.2f}, Signal codage binaire : {}".format(t[i], signal_quantifie[i], signal_code_binaire[i]))
```

```
Temps : 0.00, signal_quantifie : -0.07, Signal codage binaire : 0111
Temps : 0.01, signal_quantifie : 0.33, Signal codage binaire : 1010
Temps : 0.02, signal_quantifie : 0.60, Signal codage binaire : 1100
Temps : 0.03, signal_quantifie : 0.87, Signal codage binaire : 1110
Temps : 0.04, signal_quantifie : 1.00, Signal codage binaire : 1111
Temps : 0.05, signal_quantifie : 1.00, Signal codage binaire : 1111
Temps : 0.06, signal_quantifie : 1.00, Signal codage binaire : 1111
Temps : 0.07, signal_quantifie : 0.87, Signal codage binaire : 1110
Temps : 0.08, signal_quantifie : 0.60, Signal codage binaire : 1100
Temps : 0.09, signal_quantifie : 0.33, Signal codage binaire : 1010
Temps : 0.10, signal_quantifie : 0.07, Signal codage binaire : 1000
Temps : 0.11, signal_quantifie : -0.33, Signal codage binaire : 0101
Temps : 0.12, signal_quantifie : -0.60, Signal codage binaire : 0011
Temps : 0.13, signal_quantifie : -0.87, Signal codage binaire : 0001
Temps : 0.14, signal_quantifie : -1.00, Signal codage binaire : 0000
Temps : 0.15, signal_quantifie : -1.00, Signal codage binaire : 0000
Temps : 0.16, signal_quantifie : -1.00, Signal codage binaire : 0000
Temps : 0.17, signal_quantifie : -0.87, Signal codage binaire : 0001
Temps : 0.18, signal_quantifie : -0.60, Signal codage binaire : 0011
Temps : 0.19, signal_quantifie : -0.33, Signal codage binaire : 0101
Temps : 0.20, signal_quantifie : -0.07, Signal codage binaire : 0111
Temps : 0.21, signal_quantifie : 0.33, Signal codage binaire : 1010
Temps : 0.22, signal_quantifie : 0.60, Signal codage binaire : 1100
Temps : 0.23, signal_quantifie : 0.87, Signal codage binaire : 1110
```

5. Application d'un CAN Flash 4 bits sur un signal audio réel, illustrant la quantification et le codage binaire avec visualisation d'un segment temporel significatif.

Points clés :

- Traitement d'un signal audio réel (échantillonné à sr Hz)
- Quantification uniforme sur 16 niveaux (4 bits)
- Visualisation comparative du signal original vs quantifié
- Zoom sur un intervalle temporel révélateur (12ms-20ms)

Affichage des codes binaires associés à chaque échantillon

Analyse :

- L'effet d'escalier de la quantification est clairement visible
- Les niveaux de quantification (lignes rouges) matérialisent la résolution
- Le codage binaire représente fidèlement les paliers de quantification
- La plage d'amplitude réduite (-0.0006 à 0.0002) montre la précision nécessaire pour les signaux audio

Codage binaire plus proche dans le codage_quantification.py

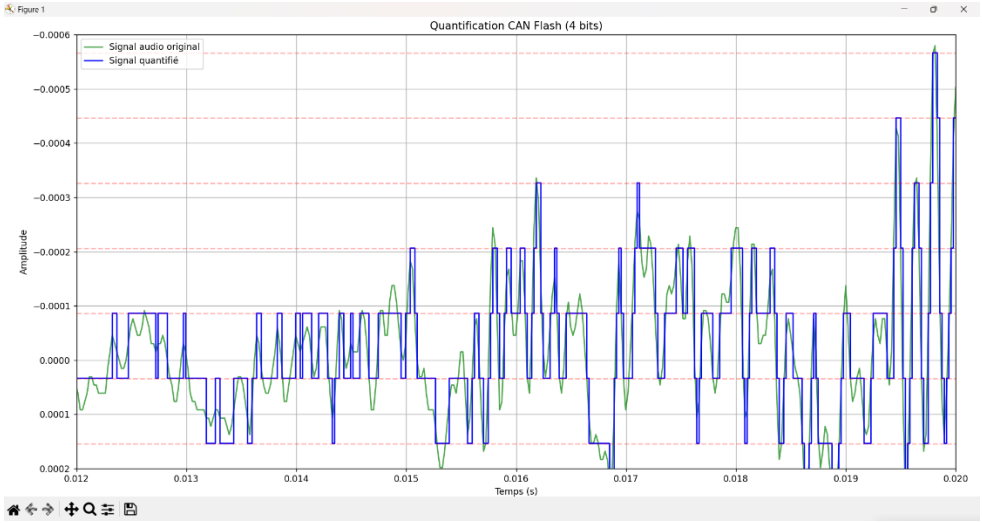
```python
Cody
def codage_binaire(signal_quantifie, levels, n_bits):
    # Créer un dictionnaire pour mapper les valeurs quantifiées aux codes binaires
    level_to_bin = {level: format(i, '0{}b'.format(n_bits)) for i, level in enumerate(levels)}

    # Conversion du niveau de quantification en code binaire
    signal_code_binaire = []

    for val in signal_quantifie:
        # Trouver le niveau le plus proche au lieu de chercher une correspondance exacte
        closest_level = levels[np.argmin(np.abs(levels - val))]
        signal_code_binaire.append(level_to_bin[closest_level])

    return signal_code_binaire
```

code5.py

```python
import numpy as np
import matplotlib.pyplot as plt
import librosa
from codage_quantification import quantification_CAN_flash, codage_binaire

# Charger l'audio
audio_file = "sample-1.wav"
y, sr = librosa.load(audio_file, sr=None)

# Pour plus de clarté, on prend juste un petit segment de l'audio
# (sinon il y aurait trop de points à afficher)
segment_length = 1000  # Nombre d'échantillons à traiter
Cody
if len(y) > segment_length:
    y_segment = y[:segment_length]
Cody
else:
    y_segment = y

time_segment = np.linspace(0, len(y_segment)/sr, len(y_segment))

# Quantification avec différents nombre de bits
n_bits = 4  # Utiliser moins de bits pour mieux voir l'effet de quantification
signal_quantifie, niveaux = quantification_CAN_flash(y_segment, n_bits)

# Codage binaire
signal_code_binaire = codage_binaire(signal_quantifie, niveaux, n_bits)
```

```
# Affichage
plt.figure(figsize=(14, 10))

# Signal original et quantifié
plt.plot(time_segment, y_segment, 'g-', label='Signal audio original', alpha=0.7)
plt.step(time_segment, signal_quantifie, 'b-', where='post', label='Signal quantifié', linewidth=1.5)

# Afficher les niveaux de quantification
Cody
for niveau in niveaux:
    plt.axhline(y=niveau, color='r', linestyle='--', alpha=0.3)

plt.title(f'Quantification CAN Flash ({n_bits} bits)')
plt.xlabel('Temps (s)')
plt.ylabel('Amplitude')
plt.xlim(0.012, 0.020)
plt.ylim(0.0002, -0.0006)
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

Graphe avec quantification



Codage :

```
# Afficher quelques échantillons et leur code binaire
print("Extrait des résultats:")
Cody
for i, t in enumerate(time_segment):
    if 0.012 <= t <= 0.020:
        print(f"""Temps: {t:.4f}s, Valeur: {y_segment[i]:.4f},
              Quantifiée: {signal_quantifie[i]:.4f}, Code binaire: {signal_code_binaire[i]}""")
```

```
Extrait des résultats:
Temps: 0.0120s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0120s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0121s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0121s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0121s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0121s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0121s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0122s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0122s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0122s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0122s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0122s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0123s, Valeur: 0.0001, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0123s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0123s, Valeur: -0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0123s, Valeur: -0.0000, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0123s, Valeur: -0.0000, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0124s, Valeur: -0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0124s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0124s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0124s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0124s, Valeur: 0.0000, Quantifiée: 0.0000, Code binaire: 0110
Temps: 0.0125s, Valeur: -0.0000, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0125s, Valeur: -0.0001, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0125s, Valeur: -0.0001, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0125s, Valeur: -0.0001, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0126s, Valeur: -0.0000, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0126s, Valeur: -0.0000, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0126s, Valeur: -0.0001, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0126s, Valeur: -0.0001, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0126s, Valeur: -0.0001, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0127s, Valeur: -0.0001, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0127s, Valeur: -0.0000, Quantifiée: -0.0001, Code binaire: 0101
Temps: 0.0127s, Valeur: -0.0000, Quantifiée: -0.0001, Code binaire: 0101
```