```python
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 24 20:59:09 2024

@author: dnick
"""

#
# main() for CL-scheme implementation
#

import numpy as np
import matplotlib.pyplot as plt
from math import sqrt

import sys
LDPC_PATH = './wimax_ldpc_lib/'
sys.path.append(f'{LDPC_PATH}/python_ldpc')

import ldpc_encoder
import ldpc_decoder

from chance_love_utils import calc_gamma_opt, calc_F, calc_q #,calc_beta_opt

def add_awgn(signal, ebno_db):
    out_data = np.zeros(len(signal), dtype=np.float32)
    ebno = 10.0**(ebno_db/10.0)
    noise_pow = 1/np.sqrt(2*ebno)
    noise = noise_pow * np.random.randn(len(signal))
    out_data = signal + noise
    print(f'signal: {signal[:10]}')
    print(f'out: {out_data[:10]}\n')

    return out_data

#
#
def generate_awgn_from_db(n, ebno_db):
    ebno = 10**(ebno_db/10)
    noise_power = 1/sqrt(2*ebno)
    return noise_power * np.random.randn(n).reshape(-1,1)

#
#
def generate_awgn(n, noise_power):
    return np.random.normal(0,sqrt(noise_power),n).reshape(-1,1)

#
#
def inner_code(symbols, s_2, n, ebno_db, gg):
    rho = 10**(ebno_db/10)
    gamma_opt = calc_gamma_opt(s_2, rho, n)
    beta_opt = np.sqrt((n - 1) / (n+(1+s_2)*n*gamma_opt*rho)) #calc_beta_opt(s_2, n, gamma_opt, rho
    q = calc_q(beta_opt,n)
    F = calc_F(s_2, beta_opt, n)
    outdata = []
    for sym in sqrt(rho)*symbols:
        fwd_noise = generate_awgn(n, 1)
        fbk_noise = generate_awgn(n, s_2)
        y = F@(fwd_noise+fbk_noise) + sym*gg + fwd_noise
        outdata.append((q.T @ y).item())

    return outdata


if __name__=='__main__':
    K = 2304
    NUM_DECODER_ITERS = 10
    num_constellation_symbols = 2
    alist_file = f'{LDPC_PATH}/alist/wimax_{K}_0_5.alist'
    encoder = ldpc_encoder.ldpc_encoder(alist_file, 5, 7, False)
    decoder = ldpc_decoder.ldpc_decoder(alist_file)

    rounds = 10
    ebdb_low, ebdb_high = -15,1
    sigma_2 = .001
    ebno_dBs = np.linspace(ebdb_low, ebdb_high,25)
    ebno_linears = 10**(ebno_dBs/10)

    colors = ['blue','orange','green','red']
    markers = ['o','P','^','s']
    ls = '--'
    fs = 15
    plt.figure()
    plt.yscale('log')
    plt.xlabel('Eb/No (dB)',fontsize=fs)
    plt.ylabel('BER',fontsize=fs)
    plt.title(r'BER for Different Outer Code Lengths: $\sigma^{2}=.001$',fontsize=fs)
    for ndx,N in enumerate([2,5,7]):
        ber_results = []
        g = np.ones(N).reshape(-1,1)/sqrt(N)
        for eb_dB, eb_lin in zip(ebno_dBs, ebno_linears):
            print(round(eb_dB,5))
            errors = 0
            for k in range(rounds):
                data = np.random.randint(0,2,encoder.N//2)
                encoded_data = encoder.encode_data(data)
                modulated_data = -2.0 * encoded_data + 1.0 # cheap BPSK

                # received_data = add_awgn(modulated_data,eb_dB)
                received_data = inner_code(modulated_data, sigma_2, N, eb_dB, g)

                decoded_data = decoder.ldpc_tdmp(received_data, NUM_DECODER_ITERS)
                errors += (decoded_data[0:K//2] != data).sum()

            ber_results.append(errors / (rounds*(decoder.N/2)))

        # Plot the modulated data in the complex plane
        plt.semilogy(ebno_dBs, ber_results, label=f'N = {N}',
                     color=colors[ndx], marker=markers[ndx], ls=ls)
        plt.axis([ebdb_low, 3, 1e-5, 1])
    plt.legend()
    plt.grid()
    plt.show()
```