```
import numpy as np
from typing import List, Union, Dict
import string
import random
import matplotlib.pyplot as plt


def convert_characters_to_ascii(string: str) -> np.ndarray:
  return np.asarray(
      [ord(char) for char in string],
      dtype=np.uint8
  )

def normalize_bits(array: np.ndarray) -> np.ndarray:
  return np.where(array, array, -1)

def add_awgn(signal: np.ndarray, mean: float = 0, sigma: float = 1.):
  awgn = np.random.normal(mean, sigma, signal.shape[0])
  return signal + awgn

def encoder(message: str, bit_repetition: int = 8) -> np.ndarray:
  ascii_string = convert_characters_to_ascii(message)
  bits = normalize_bits(
    np.unpackbits(ascii_string)
  )

  return np.repeat(bits, bit_repetition)


encoder_output = encoder(
    message = "oi baralho :)"
)

encoder_output_with_noise = add_awgn(
    signal=encoder_output
)

encoder_output_with_noise.shape
```

```
(832,)
```

```
encoder_output_with_noise.reshape(-1, 8).mean(axis=1)
```

```
array([-1.22611512,  1.15166698,  1.29879111, -1.19941177,  0.73797397,
        1.41878893,  0.81342296,  1.23666555, -1.24443809,  0.91257587,
        1.19574114, -1.05834834,  1.31429791, -0.69565449, -1.08260792,
        0.78774843, -1.34711529, -0.73699983,  0.57647433, -0.71232658,
       -1.4096707 , -0.49768776, -1.04296972, -0.78024812, -1.64636166,
        0.93626761,  1.11877173, -1.32759818, -1.03174064, -0.63184939,
        1.22162553, -0.89600158, -1.10629635,  0.99059344,  1.51739183,
       -0.70562885, -0.87454761, -1.07539734, -0.86760401,  1.05587523,
       -0.6941442 ,  0.74007426,  0.34694815,  0.98762481, -0.79581356,
       -1.67270716,  0.82387293, -1.27523147, -1.03313526,  0.93143321,
        0.85967565, -1.62372807, -0.7328833 , -1.1742258 , -0.68028971,
```

```
               1.37028224, -1.53305035,  0.06295952,  0.81696589, -0.27540215,
               1.26865233,  0.85651869, -0.61098268, -1.08249377, -1.43946374,
               1.11281794,  1.44732772, -0.83275732,  1.02121519, -1.15751128,
              -1.15630639, -1.32142779, -0.88509231,  0.88133668,  1.30609507,
              -1.09730182,  0.84302988,  1.34184595,  1.22137785,  0.35594392,
              -0.89651072, -1.55609903,  1.21738895, -0.62080933, -1.19035433,
              -0.86883545, -1.15358523, -0.43991691, -1.37874543, -1.37511132,
               0.96345529,  1.43954335,  0.72864059, -0.4473711 ,  1.16937641,
              -1.49642298, -0.6374043 , -0.86054348,  0.90704826, -0.97855343,
               0.83820846, -0.97685379, -0.92050005,  0.70610567])


def convert_ascii_to_string(ascii: np.array) -> str:
  return ''.join([chr(i) for i in ascii])

def remove_repetition(signal: np.array, bit_repetition) -> np.array:
  return signal.reshape(-1, bit_repetition)

def convert_bits_to_ascii(bits: np.array):
  return np.packbits(bits)

def decision_rule(bits: np.array) -> np.array:
  return np.where(bits < 0, 0, 1)

def decoder(signal: np.array, bit_repetition: int = 8) -> str:
  signal_mean = remove_repetition(signal, bit_repetition).mean(axis=1)
  bits = decision_rule(signal_mean)
  ascii = convert_bits_to_ascii(bits)
  return convert_ascii_to_string(ascii), bits


decoded_message, decoded_bits = decoder(
    encoder_output_with_noise
)


decoded_message
```

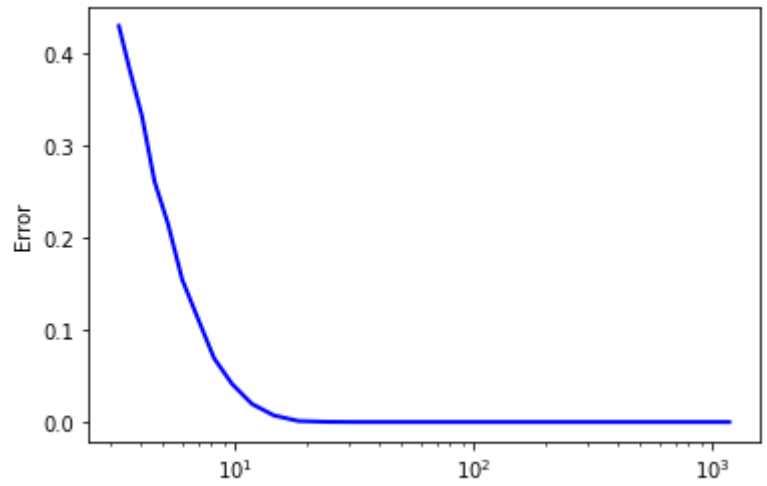
```
     'oi baralho :)'
```

```
encoder_output = encoder(
    message = "oi baralho :) vc eh mto foda e passou em redes"
)

encoder_output_with_noise = add_awgn(
    signal=encoder_output
)
decoded_message, decoded_bits = decoder(
    encoder_output_with_noise
)
decoded_message
```

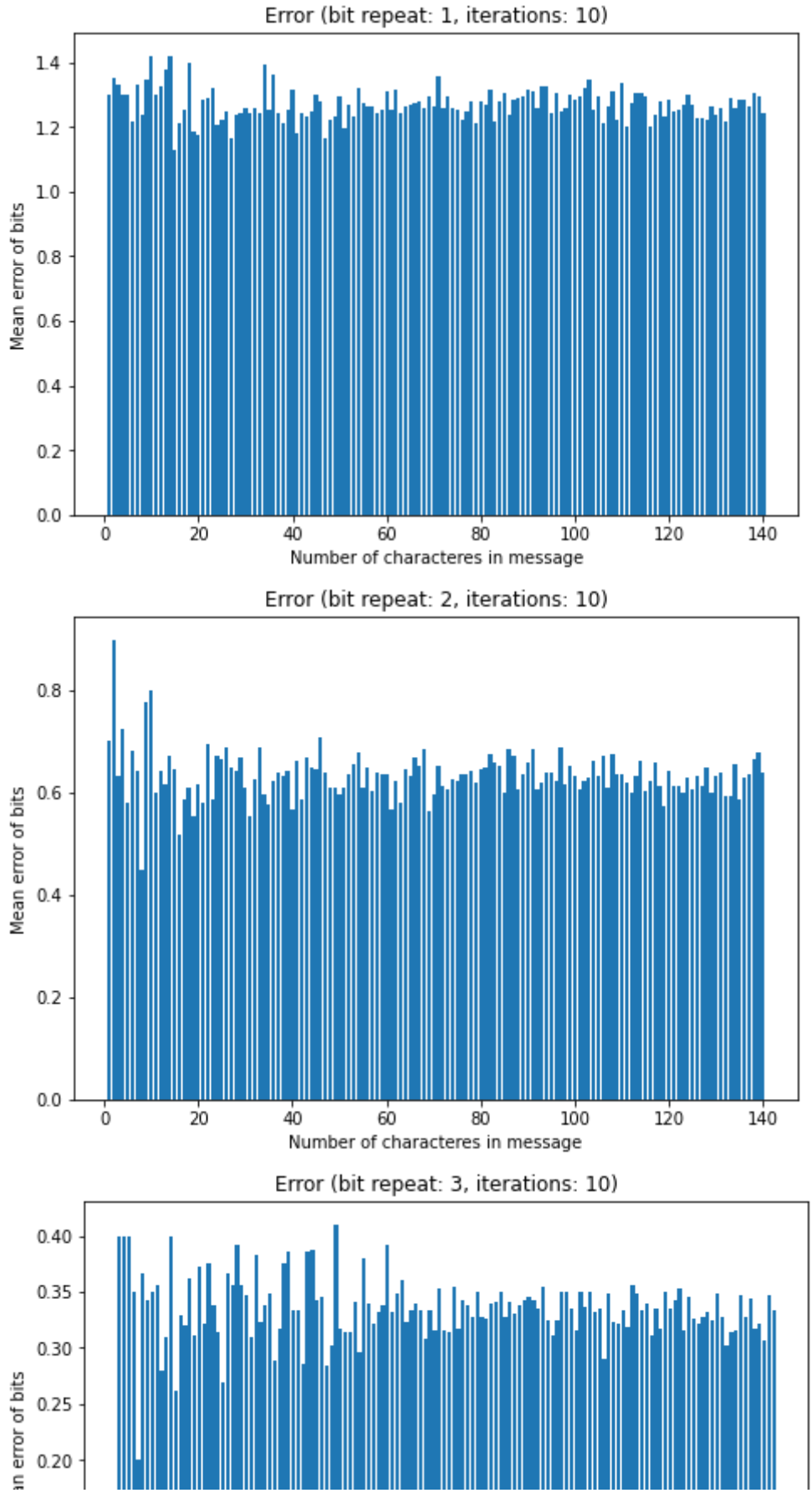
```
     'oi baralho :) vc eh mto foda e passou em redes'
```

```
def channel(message: str, bit_repetition, awgn_configuration: Dict[str, float] = {}) -> Un
```

```
        encoder_output = encoder(
            message = message,
            bit_repetition = bit_repetition
        )

        encoder_output_with_noise = add_awgn(
            signal=encoder_output,
            **awgn_configuration
        )
        decoded_message, decoded_bits = decoder(
            encoder_output_with_noise,
            bit_repetition = bit_repetition
        )

        return decision_rule(encoder_output), encoder_output_with_noise, decoded_bits, decoded

    def calculate_error(string_1: str, string_2: str):
      s1 = np.asarray(list(string_1))
      s2 = np.asarray(list(string_2))

      return ((s1 != s2).sum() / s1.shape[0])


    sigma_values = np.arange(0.1, 2, 0.1)
    message = "a" * 140

    runned_times = 100

    errors = np.zeros((sigma_values.shape[0], runned_times))

    for i_sigma_values, sigma in enumerate(sigma_values):

      for i_runned_times in range(runned_times):

        _, _, _, output_message = channel(
            message=message,
            bit_repetition=8,
            awgn_configuration = {
                "sigma": sigma
            }
        )

        errors[i_sigma_values, i_runned_times] = calculate_error(message, output_message)


    message_size = 140
    fig = plt.figure()
    ax = fig.add_subplot()
    ax.plot((message_size ** .5)/(sigma_values ** 2), errors.mean(axis=1), color='blue', lw=2)

    ax.set_xscale('log')

    ax.set_ylabel('Error')
    ax.set_xlabel('Es/(sigma^2)')
    plt.show()
```




```
def error_random_cases(max_message_len: int = 140, max_bit_repeat: int = 8, n_iterations:
  error = [[0 for _ in range(max_message_len)] for _ in range(max_bit_repeat)]
  for _ in range(n_iterations):
    for b in range(max_bit_repeat):
      for s in range(max_message_len):
        encoder_output, encoder_output_with_noise, decoded_bits, decoded_message = channel
        error[b][s] += np.sum(encoder_output.reshape(-1, b+1).mean(axis=1) != decoded_bits
  return error
def mean_error_for_string_length(error: list, max_message_len: int = 140, max_bit_repeat:
  mean_error = []
  for b in range(max_bit_repeat):
    mean_error.append([error[b][i]/(n_iterations*(i+1)) for i in range(max_message_len)])
  return mean_error


max_message_len = 140
max_bit_repeat = 8
n_iterations = 10

error = error_random_cases(
    max_message_len,
    max_bit_repeat,
    n_iterations
)
mean_error_string_length = mean_error_for_string_length(
    error,
    max_message_len,
    max_bit_repeat
)

for b in range(max_bit_repeat):
  fig = plt.figure()
  ax = fig.add_axes([0,0,1,1])
  ax.bar(list(range(1, max_message_len+1)), mean_error_string_length[b])
  plt.title(f"Error (bit repeat: {b+1}, iterations: {n_iterations})")
  plt.xlabel("Number of characteres in message")
  plt.ylabel("Mean error of bits")
  plt.show()
```




```
mean_error_for_bit_repeat = np.array(mean_error_string_length).mean(axis=1)
fig = plt.figure()
```

```
ax = fig.add_axes([0,0,1,1])
ax.bar(list(range(1, max_bit_repeat+1)), mean_error_for_bit_repeat)
plt.title("Mean error for each bit repeat")
plt.xlabel("Bit repeat")
plt.ylabel("Mean error")
plt.show()
```