Łukasz Turowski, 45136, TD_20A

LAB_09

1.

Funkcja do słowa:

```
def S2BS(s):
    result = []
    for c in s:
        b = bin(ord(c))[2:]
        if len(b) < 8:
            b = '0' + b
        result.extend([int(x) for x in b])
    return result
```

Dane wejściowe:

```
x = S2BS('Hi')
```

Słowo w ciągu:

```
# [0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1]
```

2.

Funkcja do kodowania Hamminga:

```
def Hamming_encoder(d):
    shape = d.shape[0]
    enc = np.zeros((shape, 7))
    for i in range(shape):
        enc[i] = np.dot(G, d[i]) % 2
    return enc
```

Dane zabezpieczone kodem Hamminga:

```
# [[1. 0. 0. 1. 1. 0. 0.] [1. 1. 1. 0. 0. 0. 0.] [1. 1. 0. 0. 1. 1. 0.]
 [0. 0. 1. 1. 0. 0. 1.]]
```

3. Zakodowanie Manchesterem.

Funkcja kodowania Manchester:

```
def Manchester(s, f, ft):
    encode = []
    temp = np.arange(0, 1 / f / 2 / ft)
    for i in s:
        if i == 1:
            for j in temp:
                encode.append(-1)
            for j in temp:
                encode.append(1)
        else:
            for j in temp:
                encode.append(1)
            for j in temp:
                encode.append(-1)
    return encode
```

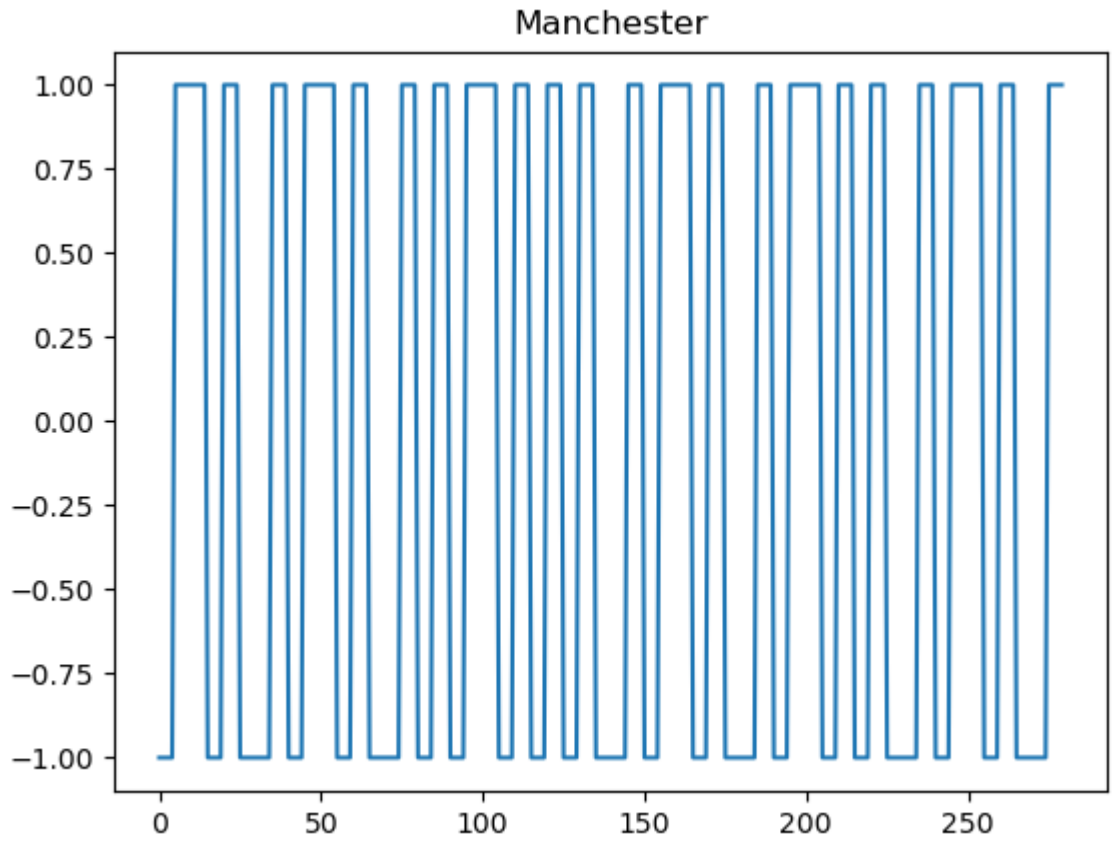
Dane zakodowane:

```
# [-1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1,
1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -
1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -
1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -
1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1,
1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1,
1]
```

Parametry do wykresu:

```
f = 10
ft = 0.01
```

Wykres Manchesteru:



4.

Funkcja dekodera Manchester:

```
def dekoManchester(s, f, ft):
    decode = []
    k = 0
    isLast = False
    last = s[0]
    for i in s:
        k += 1
        if last < i:
            isLast = True
        last = i
        if k == 1 / f / ft:
            if isLast:
                decode.append(1)
            else:
                decode.append(0)
            k = 0
            last = 1
            isLast = False
    return decode
```

Odkodowane dane:

```
# [1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0,
0, 1]
```

5.

Funkcja dekodera Hamminga:

```
def Hamming_decoder(d):
    shape = d.shape[0]
    dec = np.zeros((shape, 4))
    for i in range(shape):
        p = np.dot(H, d[i].T) % 2
        h = int(p[0] * 2 ** 0 + p[1] * 2 ** 1 + p[2] * 2 ** 2)
        if h > 0:
            d[i, h - 1] = flipbit(d[i, h - 1])
            print(i, h)
        dec[i] = [d[i, 2], d[i, 4], d[i, 5], d[i, 6]]
    return dec
```

Zdekodowany strumień binarny:

```
# [[0. 1. 0. 0.][1. 0. 0. 0.][0. 1. 1. 0.][1. 0. 0. 1.]]
```

Funkcja zamieniająca bity na string:

```
def Bit2S(bits):
    chars = []
    for b in range(len(bits) / 8):
        byte = bits[b*8:(b+1)*8]
        chars.append(chr(int(''.join([str(bit) for bit in byte]), 2)))
    return ''.join(chars)
```

Wynik:

```
# Hi
```