Patryk Rakowski 51626

Implementacje w kodzie

A-law

```python
def a_law_encode(signal, A=87.6):  2 usages  new *
    """A-law encoding."""
    abs_signal = np.abs(signal)
    encoded_signal = np.where(abs_signal < 1/A, A * abs_signal / (1 + np.log(A)), (1 + np.log(A * abs_signal)) / (1 + np.log(A)))
    encoded_signal = np.sign(signal) * encoded_signal
    return encoded_signal

def a_law_decode(encoded_signal, A=87.6):  2 usages  new *
    """A-law decoding."""
    abs_encoded_signal = np.abs(encoded_signal)
    decoded_signal = np.where(abs_encoded_signal < 1 / (1 + np.log(A)), abs_encoded_signal * (1 + np.log(A)) / A,
                              np.exp(abs_encoded_signal * (1 + np.log(A)) - 1) / A)
    decoded_signal = np.sign(encoded_signal) * decoded_signal
    return decoded_signal
```

μ-law

```python
def u_law_encode(signal, mu=255.0):  2 usages  new *
    """u-law encoding."""
    abs_signal = np.abs(signal)
    encoded_signal = np.sign(signal) * np.log(1 + mu * abs_signal) / np.log(1 + mu)
    return encoded_signal

def u_law_decode(encoded_signal, mu=255.0):  2 usages  new *
    """u-law decoding."""
    abs_encoded_signal = np.abs(encoded_signal)
    decoded_signal = np.sign(encoded_signal) * (1 / mu) * ((1 + mu) ** abs_encoded_signal - 1)
    return decoded_signal
```

Patryk Rakowski 51626

DPCM bez predykcji

```python
def dpcm_encode(signal, bit=8):  2 usages  new *
    """DPCM encoding without prediction."""
    encoded_signal = np.zeros(signal.shape)
    e = 0
    for i in range(0, signal.shape[0]):
        encoded_signal[i] = quantize_signal(signal[i] - e, bit)
        e += encoded_signal[i]
    return encoded_signal

def dpcm_decode(encoded_signal):  2 usages  new *
    """DPCM decoding without prediction."""
    decoded_signal = np.zeros(encoded_signal.shape)
    e = 0
    for i in range(0, encoded_signal.shape[0]):
        decoded_signal[i] = encoded_signal[i] + e
        e = decoded_signal[i]
    return decoded_signal
```

DPCM z predykcją np.mean (lub dowolną inną jako argument) oraz dla wskazanego n (domyślnie = 1)

```python
def dpcm_encode_with_prediction(signal, bit=8, predictor=np.mean, n=1):  1 usage  new *
    """DPCM encoding with linear prediction (1 element)."""
    encoded_signal = np.zeros(signal.shape)
    xp = np.zeros(signal.shape)
    e = 0
    for i in range(1, signal.shape[0]):
        encoded_signal[i] = quantize_signal(signal[i] - e, bit)
        xp[i] = encoded_signal[i] + e
        idx = (np.arange(i - n, i, 1, dtype=int) + 1)
        idx = np.delete(idx, idx < 0)
        e = predictor(xp[idx])
    return encoded_signal

def dpcm_decode_with_prediction(encoded_signal, predictor=np.mean, n=1):  1 usage  new *
    """DPCM decoding with linear prediction (1 element)."""
    decoded_signal = np.zeros(encoded_signal.shape)
    xp = np.zeros(encoded_signal.shape)
    decoded_signal[0] = encoded_signal[0]  # Set the first value to the first value of the encoded signal
    e = decoded_signal[0]
    for i in range(1, encoded_signal.shape[0]):
        decoded_signal[i] = encoded_signal[i] + e
        xp[i] = decoded_signal[i]
        idx = (np.arange(i - n, i, 1, dtype=int) + 1)
        idx = np.delete(idx, idx < 0)
        e = predictor(xp[idx])
    return decoded_signal
```

Patryk Rakowski 51626

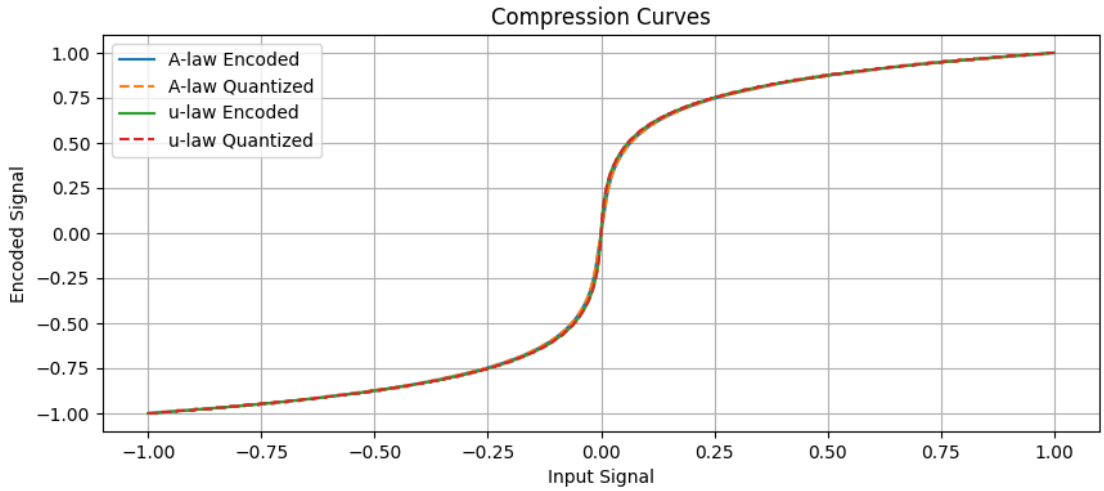
Metoda quantize_signal

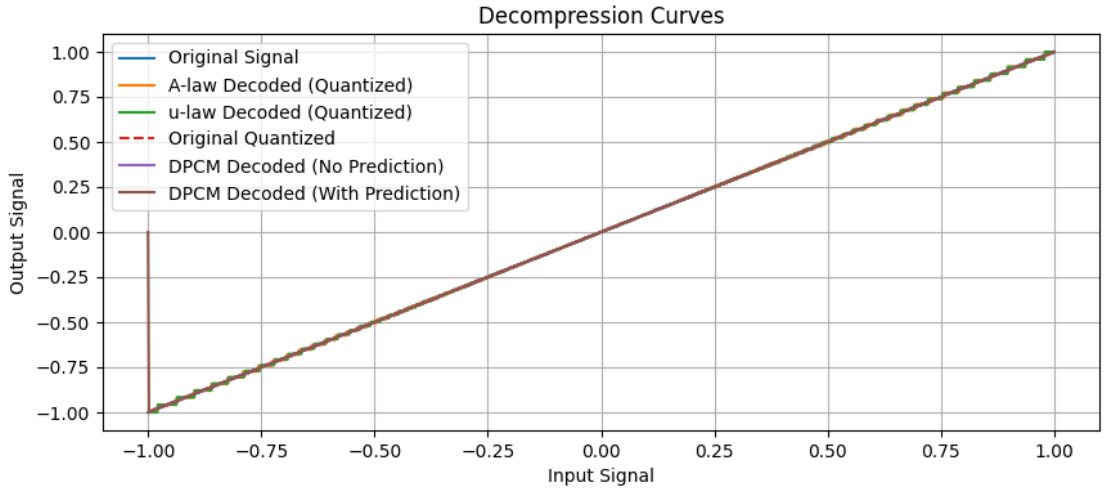
```python
def quantize_signal(signal, num_bits=8):  7 usages  new *
    """Quantize the signal to the specified number of bits."""
    # Calculate the number of quantization levels
    num_levels = 2 ** num_bits

    # Scale the signal to the range [0, num_levels - 1]
    scaled_signal = (signal + 1) * (num_levels / 2)

    # Quantize the signal
    quantized_signal = np.round(scaled_signal)

    # Scale the quantized signal back to the range [-1, 1]
    quantized_signal = (quantized_signal / (num_levels / 2)) - 1

    return quantized_signal
```

Patryk Rakowski 51626

Przypadki testowe

x = np.linspace(-1, 1, 1000), y = x

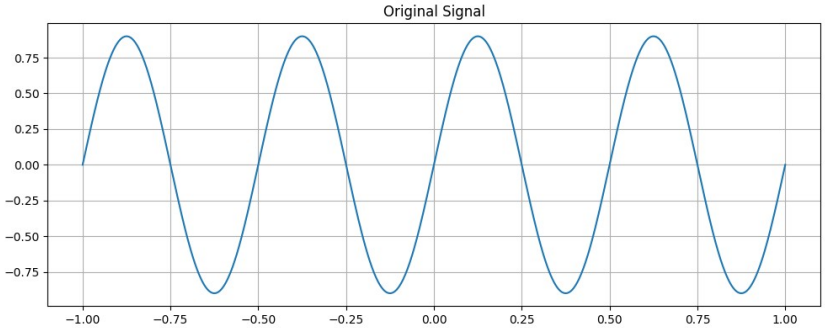


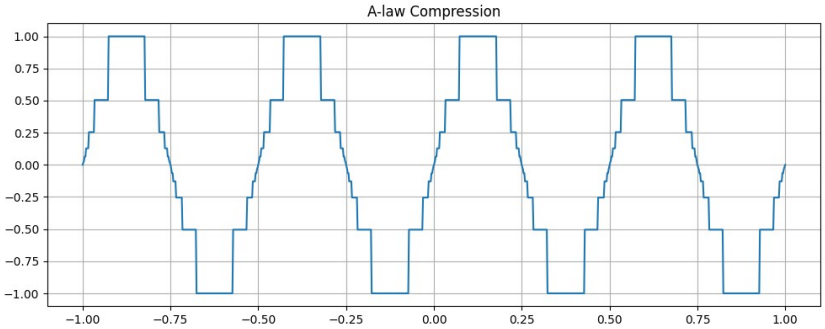


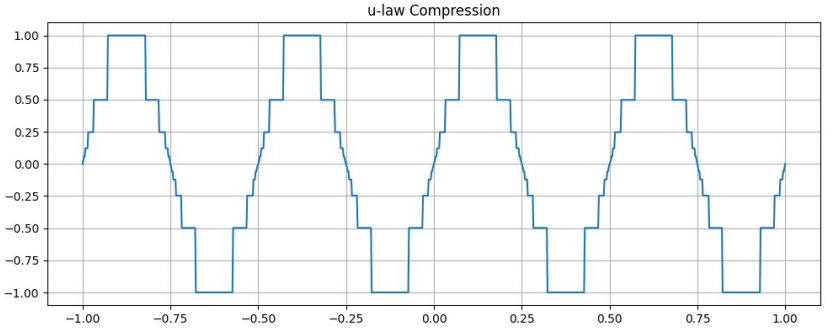
Patryk Rakowski 51626

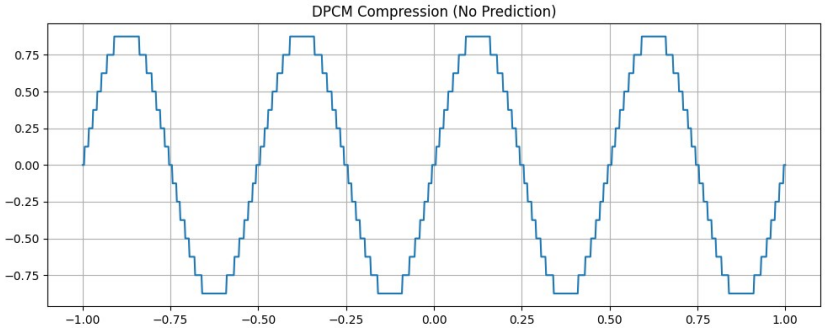
*x = np.linspace(-1, 1, 1000), y = 0.9*np.sin(np.pi*x*4)*

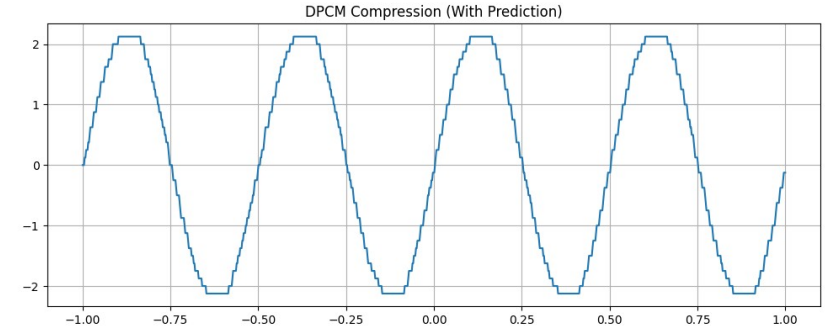
Bit=4, predictor = np.mean, n = 4











Patryk Rakowski 51626

Wnioski

Porównanie a-law i u-law vs dcpm

A-law i u-law: Używają logarytmicznego kompandowania do kompresji zakresu dynamicznego sygnału, redukując błąd kwantyzacji dla sygnałów o szerokim zakresie dynamicznym.

DPCM: Koduje różnice między próbkami, co może być dodatkowo zoptymalizowane za pomocą predykcji, aby zmniejszyć zakres dynamiczny i poprawić efektywność kwantyzacji.

Każda metoda ma swoje zalety i jest odpowiednia do różnych zastosowań, przy czym A-law i u-law są bardziej powszechne w systemach telekomunikacyjnych, a DPCM jest używane w różnych zastosowaniach kompresji audio i wideo.

Badanie plików dźwiękowych

Dla sing_hig1.wav w a-law i u-law mocniej słychać zniekształcenia dźwięku niż dla DPCM.

Dla sing_low1.wav najgorzej wypada a-law, potem dpcm i u-law. A-law najwięcej zniekształceń, dpcm najwięcej szumu.

Dla sing_medium1.wav próbki brzmią porównywalnie do siebie.

	8	7	6	5	4	3	2
sing_low1.wav	Bardzo dobrze może lekki szum	Identycznie jak wcześniej	Mocniejszy szum lekkie zniekształcenia	Gorsza jakość	Dużo gorzej brzmi ale wciąż raczej da się rozpoznać	Do rozpoznania, jakby bardzo blisko mikrofonu nagrać w słabej jakości	Z grubsza buczenie, można nie rozpoznać, bardzo zniekształcone
sing_medium1.wav	Dobrze	Dobrze + minimalny szum	Nieźle + szum	Gorsza jakość dźwięki	Zła jakość duży szum	Ciężej rozpoznać ciężko się słucha	Strasznie głośno i straszny dźwięk
sing_high1.wav	Dobrze, krzyk	Nieźle, trochę jak czajnik	Bardziej jak lokomotywa	Lokomotywa gorsza jakość dźwięku	Express polarny mocne zniekształcenia	Mocno nieprzyjemny dźwięk trudno rozpoznać	Jak wcześniej i dużo głośniej