# Arithmetic Codes

## *Arithmetic Codes*

➢ ***Basic principles:***

- the code is associated to the sequence of symbols $m_i$ (messages), not to every symbol in the sequence.

- coding of intervals of type $[c_i, d_i[$ for each symbol.

- iteration on the selected interval for the next symbol of the sequence

- one codes the sequence of symbols with a real value on [0, 1 [.

Arithmetic codes allow you to encode a sequence of events by using estimates of the probabilities of the events. The arithmetic code assigns one codeword to each possible data set. This technique differs from Huffman codes, which assign one codeword to each possible event.
The codeword assigned to one sequence is a real number which belongs to the half-open unit interval [0, 1 [. Arithmetic codes are calculated by successive subdivisions of this original unit interval. For each new event in the sequence, the subinterval is refined using the probabilities of all the individual events.

Finally we create a half-open subinterval of [0, 1[, so that any value in this subinterval encodes the original sequence.

# *Arithmetic Codes*

➢ ***Definitions:***

- Let $S = \{s_1, s_2, \ldots, s_N\}$ be a source and $p_k = Pr(s_k)$

- $[Ls_k, Hs_k[$ is the interval corresponding to the symbol $s_k$
  with: $Hs_k - Ls_k = p_k$

➢ ***Encoding algorithm:***

1) Initialization: $L_c = 0$ ; $H_c = 1$
2) Calculate code sub-intervals
3) Get next input symbol $s_k$
4) Update the code sub-interval
5) Repeat from step 2 until all the sequence has been encoded

Let $S = \{s_1, \ldots, s_N\}$ be a source which can produce N source symbols. The probabilities of the source symbols are denoted by: $\forall \in [1, N]$, $P\{s_k\} = p_k$. Here is the basic algorithm for the arithmetic coding of a sequence $s_M = \{s_{\alpha 1}, s_{\alpha 2}, \ldots, s_{\alpha M}\}$ of M source symbols ($s_{\alpha k}$ stands for the k-th source symbol that occurs in the sequence we want to encode):

- Step1:

Let us begin with a current half-open interval $[L_c, H_c[$ initialized to $[0, 1[$ (this interval corresponds to the probability of choosing the first source symbol $s_{\alpha 1}$). The length of the current interval is thus defined by: length = $H_c - L_c$.

- Step 2:

For each source symbol in the sequence, we subdivide the current interval into half-open subintervals $[Ls_k, Hs_k[$, one for each possible source symbol $s_k$. The size of a symbol's subinterval $[Ls_k, Hs_k[$ depends on the probability $p_k$ of the symbol $s_k$. The subinterval length ( $Ls_k - Hs_k$ ) is defined so that: $H_k - L_k = p_k$, then:

$$Ls_k = L_c + \text{length} \times \sum_{i=1}^{k-1} p_i \quad \text{et } Hs_k = L_c + \text{length} \times \sum_{i=1}^{k} p_i \quad .$$

- Step 3:

We select the subinterval corresponding to the source symbol $s_k$ that occurs next and make it the new current interval $[L_c, H_c[$: $\begin{cases} L_c = L_c + length \times Ls_k \\ H_c = L_c + length \times Hs_k \end{cases}$

- Step 4 :

This new current interval is subdivided again as described in Step 2.

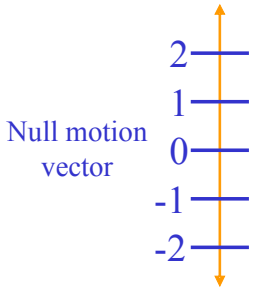
- Step 5 :

Repeat the steps 2, 3, and 4 until the whole sequence of source symbols has been encoded.

# *Arithmetic Coding*

## *Example*

The source S= {-2, -1, 0, 1, 2} is a set of 5 possible motion vector values (used for video coding)



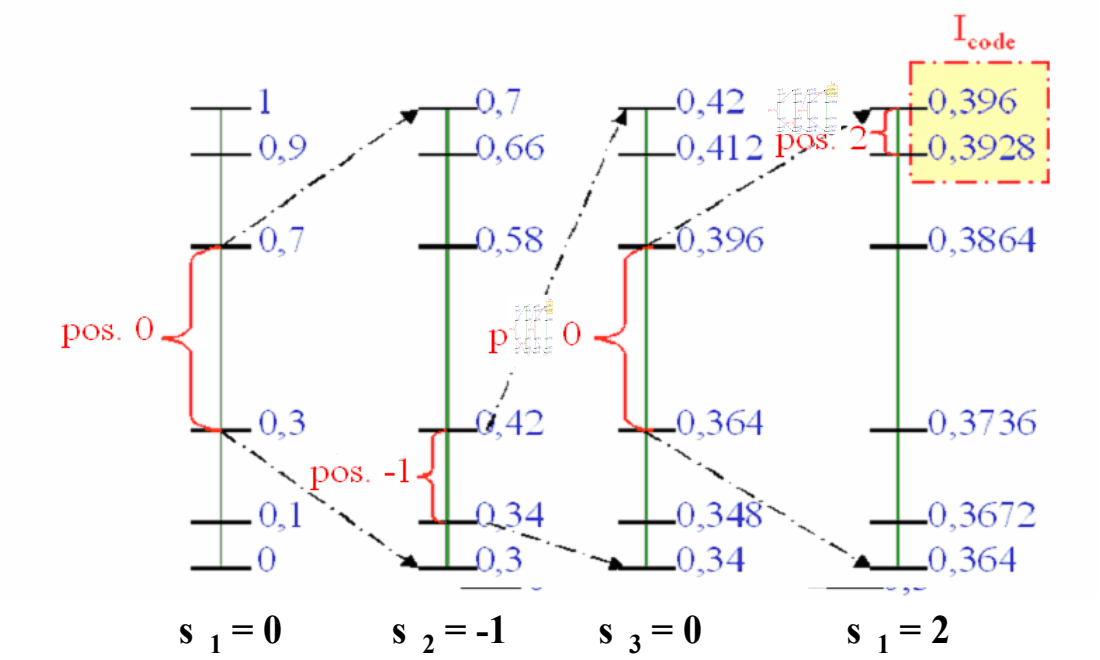

- 'Y' is the random variable associated to the motion vector values
- With the following probabilities:
  - $\Pr\{Y = -2\} = p_1 = 0{,}1$
  - $\Pr\{Y = -1\} = p_2 = 0{,}2$
  - $\Pr\{Y = 0\} = p_3 = 0{,}4$
  - $\Pr\{Y = 1\} = p_4 = 0{,}2$
  - $\Pr\{Y = 2\} = p_5 = 0{,}1$

We want to encode the motion vector sequence (0, -1, 0, 2)

Here is an example for encoding motion vectors of a video signal with an arithmetic code.

# *Arithmetic Coding*

## Subdivisions of the current sub-intervals




To encode the sequence $\{s_{\alpha 1}, s_{\alpha 2}, s_{\alpha 3}, s_{\alpha 4}\} = \{s_3, s_2, s_3, s_5\} = \{0, -1, 0, 2\}$, we subdivide the unit current interval [0, 1[ into 5 half-open intervals, then we select the subinterval corresponding to the first motion vector that occurs in the sequence (value 0). This subinterval is the new current interval. We subdivide it and we select the subinterval corresponding to the next event (the motion vector -1).

We repeat these steps for each source symbol of the sequence (here these source symbols are the motion vectors). Consequently, we can encode the sequence (0, -1, 0, 2) of vertical motion vectors by any value in the half-open range [0.3928, 0.396[. The value 0.3945 encodes this sequence; therefore we need 8 bits to encode this sequence:

$$0.3945 = 0\times2^{-1} + 2^{-2} + 2^{-3} + 0\times2^{-4} + 0\times2^{-5} + 2^{-6} + 0\times2^{-7} + 2^{-8}$$

So we need 8/5 = 1.6 bits/symbol.

## *Arithmetic Coding*

➢ ***Decoding algorithm***

1) Initialization: $L_c = 0$ ; $H_c = 1$

2) Calculate the code sub-interval length: length = $H_c - L_c$

3) Find the symbol sub-interval $[Ls_k, Hs_k[$ with $1 \quad k \quad N$

such that: $Ls_k \quad (\text{codeword} - L_c) / \text{length} < Hs_k$

4) Output symbol: $s_k$

5) Update the subinterval:
$$\begin{cases} L_c = L_c + \text{length} \quad Ls_k \\ H_c = L_c + \text{length} \quad Hs_k \end{cases}$$

6) Repeat from step 2 until all the last symbol is decoded

The figure above describes the decoding algorithm of a codeword obtained after having encoded a sequence of source symbols with an arithmetic code. This decoding algorithm is performed for the previous example. Let us consider the codeword $M_c = 0.3945$:

*[ Algorithm beginning ]*

Step 1:

We initialize the current interval $[L_c, H_c[$ : $L_c = 0$ et $H_c = 1$.

Step 2:

We calculate the length L of the current interval: $L = H_c - L_c = 1$.

Step 3:

We calculate the value $( M_c - L_c ) / L = 0.3945$, and we select the subinterval $[Ls_k, Hs_k[$ so that $M_c \in [Ls_k, Hs_k[$. Here, the selected subinterval is [0.3, 0.7[. This subinterval corresponds to the half-open interval $[Ls_3, Hs_3[$.

Step 4:

The ***first symbol*** $s_{\alpha 1}$ of the sequence is thus $s_3$ (***motion vector 0***).

Step 5:

We create the new current interval $[L_c, H_c[$ for encoding the next source symbol:

$$Lc = Lc + L \times Ls3 = 0 + 1 \times 0.3 = 0.3$$
$$Hc = Lc + L \times Hs3 = 0 + 1 \times 0.7 = 0.7$$

Step 2:

We calculate the length L of the current interval: $L = H_c - L_c = 0.7 - 0.3 = 0.4$.

Step 3:

( Mc - Lc ) / L = (0.3945 – 0.3) / 0.4 = 0,2363.
This value belongs to the subinterval [Ls2, Hs2[ = [0.1, 0.3[.

Step 4:

The ***second symbol*** $s_{\alpha 2}$ of the sequence is thus $s_2$ (***motion vector -1***).

Step 5:

Lc = Lc + L × Ls2 = 0.3 + 0.4 × 0.1 = 0.34
Hc = Lc + L × Hs2 = 0.3 + 0.4 × 0.3 = 0.42

Step 2:

We calculate the length L of the current interval: $L = H_c - L_c = 0.42 - 0.34 = 0.08$.

Step 3:

( Mc - Lc ) / L = (0.3945 – 0.34) / 0.08 = 0.6812.
This value belongs to the subinterval [Ls3, Hs3[ = [0.3, 0.7[.

Step 4:

The ***third symbol*** $s_{\alpha 3}$ of the sequence is thus $s_3$ (***motion vector 0***).

Step 5:

Lc = Lc + L × Ls3 = 0.34 + 0.08 × 0.3 = 0.364
Hc = Lc + L × Hs3 = 0.34 + 0.08 × 0.7 = 0.396
Step 2:

We calculate the length L of the current interval: $L = H_c - L_c = 0.396 - 0.364 = 0.032$.

Step 3:

( Mc - Lc ) / L = (0.3945 – 0.364) / 0.032 = 0.9531.
This value belongs to the subinterval [Ls5, Hs5[ = [0.9, 1[.

Step 4:

The ***fourth symbol*** $s_{\alpha 4}$ of the sequence is thus $s_5$ (***motion vector 2***).

*[ Algorithm end ]*

The decoding of the value 0.3945 allows us to rebuild the original sequence $\{s_{\alpha 1}, s_{\alpha 2}, s_{\alpha 3}, s_{\alpha 4}\}$ = $\{s_3, s_2, s_3, s_5\}$ = { 0, -1, 0, 2 }.

Contrary to Huffman codes, arithmetic codes allow you to allocate fractional bits to symbols. The data compression with arithmetic codes is thus more efficient. However arithmetic coding is slower than Huffman. It is not possible to start decoding without the entire sequence of symbols, which is possible in Huffman coding.
The compression rate can also be increased by using probability models which are not static. The probabilities are adapted according to the current and the previous sequences: arithmetic coding can thus handle adaptive coding.