

MATLAB FUNCTIONS FOR SOURCE CODING

Communication System Toolbox offer several functions for source coding. We consider here only Huffman and arithmetic coding

Contents

- Huffman coding
- Create a Huffman Code Dictionary in MATLAB
- Create and Decode a Huffman Code Using MATLAB
- Arithmetic Coding
- Represent Arithmetic Coding Parameters
- Create and Decode an Arithmetic Code Using MATLAB

Huffman coding

Huffman coding offers a way to compress data. The average length of a Huffman code depends on the statistical frequency with which the source produces each symbol from its alphabet. A Huffman code dictionary, which associates each data symbol with a codeword, has the property that no codeword in the dictionary is a prefix of any other codeword in the dictionary. The `huffmandict`, `huffmanenco`, and `huffmandeco` functions support Huffman coding and decoding.

Note For long sequences from sources having skewed distributions and small alphabets, arithmetic coding compresses better than Huffman coding. To learn how to use arithmetic coding, see Arithmetic Coding.

Create a Huffman Code Dictionary in MATLAB

Huffman coding requires statistical information about the source of the data being encoded. In particular, the `p` input argument in the `huffmandict` function lists the probability with which the source produces each symbol in its alphabet. For example, consider a data source that produces 1s with probability 0.1, 2s with probability 0.1, and 3s with probability 0.8. The main computational step in encoding data from this source using a Huffman code is to create a dictionary that associates each data symbol with a codeword. The commands below create such a dictionary and then show the codeword vector associated with a particular value from the data source.

```
symbols = [1 2 3]; % Data symbols
p = [0.1 0.1 0.8]; % Probability of each data symbol
```

The output below shows that the most probable data symbol, 3, is associated with a one-digit codeword, while less probable data symbols are associated with two-digit codewords. The output also shows, for example, that a Huffman encoder receiving the data symbol 1 should substitute the sequence 11.

```
dict = huffmandict(symbols,p) % Create the dictionary.
dict{1,:} % Show one row of the dictionary.
```

```
dict =

    [1]    [1x2 double]
    [2]    [1x2 double]
    [3]    [          0]
```

```
ans =
```

```
    1
```

```
ans =
```

```
    1    1
```

Create and Decode a Huffman Code Using MATLAB

The example below performs Huffman encoding and decoding, using a source whose alphabet has three symbols. Notice that the `huffmanenco` and `huffmandeco` functions use the dictionary that `huffmandict` created.

```
sig = repmat([3 3 1 3 3 3 3 2 3],1,50); % Data to encode
symbols = [1 2 3]; % Distinct data symbols appearing in sig
p = [0.1 0.1 0.8]; % Probability of each data symbol
dict = huffmandict(symbols,p); % Create the dictionary.
hcode = huffmanenco(sig,dict); % Encode the data.
dhsig = huffmandeco(hcode,dict); % Decode the code.
whos
```

Name	Size	Bytes	Class	Attributes
ans	1x2	16	double	
code	1x470	3760	double	

counts	1x3	24	double
dhsig	1x500	4000	double
dict	3x2	424	cell
dseq	1x500	4000	double
hcode	1x600	4800	double
p	1x3	24	double
seq	1x500	4000	double
sig	1x500	4000	double
symbols	1x3	24	double

Arithmetic Coding

Arithmetic coding offers a way to compress data and can be useful for data sources having a small alphabet. The length of an arithmetic code, instead of being fixed relative to the number of symbols being encoded, depends on the statistical frequency with which the source produces each symbol from its alphabet. For long sequences from sources having skewed distributions and small alphabets, arithmetic coding compresses better than Huffman coding.

The `arithenco` and `arithdeco` functions support arithmetic coding and decoding.

Represent Arithmetic Coding Parameters

Arithmetic coding requires statistical information about the source of the data being encoded. In particular, the counts input argument in the `arithenco` and `arithdeco` functions lists the frequency with which the source produces each symbol in its alphabet. You can determine the frequencies by studying a set of test data from the source. The set of test data can have any size you choose, as long as each symbol in the alphabet has a nonzero frequency.

For example, before encoding data from a source that produces 10 x's, 10 y's, and 80 z's in a typical 100-symbol set of test data, define

```
counts = [10 10 80];
```

Alternatively, if a larger set of test data from the source contains 22 x's, 23 y's, and 185 z's, then define

```
counts = [22 23 185];
```

Create and Decode an Arithmetic Code Using MATLAB

The example below performs arithmetic encoding and decoding, using a source whose alphabet has three symbols.

```
seq = repmat([3 3 1 3 3 3 3 2 3],1,50);  
counts = [10 10 80];  
code = arithenco(seq,counts);  
dseq = arithdeco(code,counts,length(seq));  
whos
```

Name	Size	Bytes	Class	Attributes
ans	1x2	16	double	
code	1x470	3760	double	
counts	1x3	24	double	
dhsig	1x500	4000	double	
dict	3x2	424	cell	
dseq	1x500	4000	double	
hcode	1x600	4800	double	
p	1x3	24	double	
seq	1x500	4000	double	
sig	1x500	4000	double	
symbols	1x3	24	double	