Table of Contents


```
clearvars; clc
```

MAIN

```
fid = fopen('_README.txt');
txt = textscan(fid, '%s', 'whitespace', '');
msg = txt{1};
orig_bytes = msg{1};
compr_bytes = dzip(orig_bytes);
decom_bytes = dunzip(compr_bytes);

fprintf(" * Original length = %d B\n",length(orig_bytes));
fprintf(" * Compressed length = %d B\n",length(compr_bytes));
fprintf(" * Decompressed length = %d B\n",length(decom_bytes));
fprintf(" * I/O identity-check result = %d\n",strcmp(orig_bytes,decom_bytes));
      % '1' means successful
fprintf("\n ------ \n\n")
fprintf(" * Compressed message =\n\n");
fprintf("%c ",compr_bytes);
fprintf("\n\n * Decompressed message =\n\n");
fprintf(decom_bytes);
```

ZIP FUNCTION

```
function Z = dzip(M)
% DZIP - losslessly compress data into smaller memory space
%
% USAGE:
% Z = dzip(M)
%
% VARIABLES:
% M = variable to compress
% Z = compressed output
%
% NOTES: (1) The input variable M can be a scalar, vector, matrix, or
%            n-dimensional matrix
%        (2) The input variable must be a non-complex and full (meaning
%            matrices declared as type "sparse" are not allowed)
%        (3) Permitted input types include: double, single, logical,
%            char, int8, uint8, int16, uint16, int32, uint32, int64,
%            and uint64.
%        (4) In testing, DZIP compresses several megabytes of data per
%            second.
```

```
%        (5) In testing, random matrices of type double compress to about
%            75% of their original size. Sparsely populated matrices or
%            matrices with regular structure can compress to less than
%            1% of their original size. The realized compression ratio
%            is heavily dependent on the data.
%        (6) Variables originally occupying very little memory (less than
%            about half of one kilobyte) are handled correctly, but
%            the compression requires some overhead and may actually
%            increase the storage size of such small data sets.
%            One exception to this rule is noted below.
%        (7) LOGICAL variables are compressed to a small fraction of
%            their original sizes.
%        (8) The DUNZIP function decompresses the output of this function
%            and restores the original data, including size and class type.
%        (9) This function uses the public domain ZLIB Deflater algorithm.
%       (10) Carefully tested, but no warranty; use at your own risk.
%       (11) Michael Kleder, Nov 2005
s = size(M);
c = class(M);
cn = strmatch(c,{'double','single','logical','char','int8','uint8',...
    'int16','uint16','int32','uint32','int64','uint64'});
if cn == 3 || cn == 4
    M=uint8(M);
end
M=typecast(M(:),'uint8');
M=[uint8(cn);uint8(length(s));typecast(s(:),'uint8');M(:)];
f=java.io.ByteArrayOutputStream();
g=java.util.zip.DeflaterOutputStream(f);
g.write(M);
g.close;
Z=typecast(f.toByteArray,'uint8');
f.close;
return
end
```

UNZIP FUNCTION

```
function M = dunzip(Z)
% DUNZIP - decompress DZIP output to recover original data
%
% USAGE:
% M = dzip(Z)
%
% VARIABLES:
% Z = compressed variable to decompress
% M = decompressed output
%
% NOTES: (1) The input variable Z is created by the DZIP function and
%            is a vector of type uint8
%        (2) The decompressed output will have the same data type and
%            dimensions as the original data provided to DZIP.
%        (3) See DZIP for other notes.
%        (4) Carefully tested, but no warranty; use at your own risk.
```

```
%       (5) Michael Kleder, Nov 2005

import com.mathworks.mlwidgets.io.InterruptibleStreamCopier
a=java.io.ByteArrayInputStream(Z);
b=java.util.zip.InflaterInputStream(a);
isc = InterruptibleStreamCopier.getInterruptibleStreamCopier;
c = java.io.ByteArrayOutputStream;
isc.copyStream(b,c);
Q=typecast(c.toByteArray,'uint8');
cn = double(Q(1)); % class
nd = double(Q(2)); % # dims
s = typecast(Q(3:8*nd+2),'double')'; % size
Q=Q(8*nd+3:end);
if cn == 3
    M  = logical(Q);
elseif cn == 4
    M = char(Q);
else
    ct = {'double','single','logical','char','int8','uint8',...
        'int16','uint16','int32','uint32','int64','uint64'};
    M = typecast(Q,ct{cn});
end
M=reshape(M,s);
return
end
```

 * *Original length = 2966 B*
 * *Compressed length = 1266 B*
 * *Decompressed length = 2966 B*
 * *I/O identity-check result = 1*

 ------

 * *Compressed message =*

x # # V É n G # # ø < # K # ù # È # Ä # 1 # S Ð #) #
÷ 2 î î ! 9 F þ Ç ¿ # O È « # # ¢ h # Ñ # « « ^ ½ W K # ó õ W é ï # Ó
*¿ ñ § # > # û 3 ñ · X P # b # ¡ \ F # Å W Ò W ! m . » # ± * m*
*# ³ a * Þ ê 2 # L ¥ # U A B # Ö # Ù (6 r K b I d Å # ¼ Z) Ê Å # ¦ ë*
© X : § I Ú ä ² T 6 ^ # ¦ < | À Î { r © S ü # ´ n Â # è Ë , ê x + ç #
 # Ý É # ö ¤ [þ E Y # 8 ¯ 1 ï Ö ^ £ ì Z # # ? 3 # " § # ² ø] Y Ä à
7 # î x j # # Ò Ô ù ý ® õ + Ë \ 9 Á é I - Ö d É K & ! ! Ñ y ¹ ¦ Æ ÿ ¸ Ò
æ æ t l ç È Ì µ î · î C e # N « L Ø Ò , É # Ù V z Å # Y w j : #
Ê) # m e Ç # Ó Û Ó Î ¿ o # g % # Þ # É K e ¡ ¬ # T d Ê « Ì Y H 8
« ! » Þ ª # X ¼ {
~ ø ¢ Ï l # ý Ä P Ì < 7 1 þ # ã Ë Ö ñ B V ® # # î T Ü p 5 ð » ¾ ô R
æ
v 1 2 á n Õ Ó × ã © O ® Ø j Ð E y Ò ñ # ù B Å Ú ¸ - { # b) µ N Ç P ¹
 # ¥ # > A Ù <
² â = U õ [e # ² / À ³ G 3 ² # K ¥ U ¬ Y] ¹ + ¡ ¥ # { # # K Ò # 3 F
× È á ÷ V £ + f x · £ # å b ¨ i g ^ 7 q # < ô 1 Ï # Ô x þ î é ä F
*# Í £ # A Ø ô À º (Û ë Ú # > # u # * D 2 " Õ # o ¨ k # ÷ F F ¯ ö M*
Á { S s ÷ K î ¬ ë Ã × j K 7 # W E < n Õ ³ ® § ¨ " # ù Â ý # 8 (9 g m #
@ ¢ ¶ (Ð ´ # V Q s ® Ñ û # | Ä J è ` # ñ # 2 á É J Ó T w ü A

Ç } s p ã # | á) Õ ³ Q # v ú ê I æ ³ Ô ê Í # b ¯ h Ù ¸ ã ! ø æ j ñ ú ê
i : s ý ë ï b . 1 ú #] [ ] ; # Ê T æ n â a Ì á ! $ í # ! s Æ L Ù î ¦ ù
á] ; # 8 ô ¢ # V ¥ > ä Ð ¯ # ç ` ° â ! » Q # ÿ ? 4 2 < ¯ ¥ F I ¢
N ¨ Ü µ j # d K # » ã ¿ ~ w v Ü y × * ¦ Ô Q è Û ú ` Ë < U Ö % u #
³ Ý # ò Ô 5 # ß ¨ ù b # # # Ø Ã é ^ # ý ¨ Í 6 õ ^ 9 á é ü ´ è # g ¾ a æ
c è ï F ô É * î ¤ § I » E < Ë) # 7 Ï ç # # e T Ü u ~ ñ Y ç 0 ³ i ± µ U
7 Þ # ë # p i B # B · µ ¦ Ý n R ð î ª E Â ë > B × S w ¥ @ X $ á # Ü £ Á
Ý , # r # # ® # · ³ ù è I ä í # # ¨ O (# d # \ ß Ì ç Ï F = ¶ Ã j
' Ë g É p # ² F Å # É p Ð t Ñ # Ñ b # ´ 6 ÷ K ð ¼ k Ò #] ñ ¯ # ~
t d U Ó É Z # - ë Î K ^ D 4 # Ú ü I Ì ¦ \ M ³ ¥ ï S p ¼ ó æ A # Y
(p ç ð ¥ Á ì × Y Û f . G # £ è X R ^ 2
Û w M = õ
¦ a å ë # ® ° / & v · Ä V ± # ½ x
Å _ O P r ù y # # Û Ä K k ö ù Û Û # _ ° ù ò ì X ° z â q ´ « ë g ½ ó
' _ ^ ü ý \ y g f ý b , 6 ´ # í # Ý 6 # # z + ô ð / ý ß [§ ± < # ï
' Ñ M ø ¿ ¾ + # z o r D ì Ù # Q Z « # # ° æ O M # á ¸ 8 ^ # w ã u · ®
Æ Ú Î º á « o ß # Æ " S # # ö) î ¨ ` c # ¥ Å K õ Q Þ Õ æ ¢ / > G û
x © | T Å ¬ ´ ø Ä Å £ ¹ ? á ì ¿ © Q #

* Decompressed message =

§ Ex1 | Test "timedate", "diary" and "whos" functions. Plus, data type assignement have been verified (e.g. boolean and uint8).

§ Ex2 | Test "table", "cell" and "struct" data format.

§ Ex3 | Test object-oriented programming using class defined in "class_wave.m" file.

§ Ex4 | Test audio signal generation and storage using "sound" and "audiowrite" functions.

§ Ex5 | Test symbolic numbers/variables/functions, besides "eval" and "solve" functions.

§ Ex6 | Test custom functions for binary to decimal conversion (and vice versa).

§ Ex7 | Test custom functions for char-message to binary conversion (and vice versa).

§ Ex8 | Test Taylor series with "taylor" function and plotting of symbolic functions with "fplot".

§ Ex9 | Test scrpit for moving a ball within a chessboard depending on keyboard input.

§ Ex10 | Test capability of using double axis labeling on plots for both X (top/bottom) and Y (left/right) axis.

§ Ex11 | Test custom DVB-S encoding/decoding chain.

§ Ex12 | Test equation system solver with "equationsToMatrix" and "linsolve" functions.

§ Ex13 | Test LiveScript format.

§ Ex14 | Test eye diagram for periodic signal quality estimation with "eyediagram" function.

§ Ex15 | Test custom script for automatic renaming of files.

§ Ex16 | Test custom script for interpreting and converting read/write data files between MATLAB and GNU Radio.

§ Ex17 | Test Gold pseudo-random sequences with "comm.GoldSequence" function.

§ Ex18 | Test image manipulation with "rgb2gray", "histeq", "filter2" and other functions.

§ Ex19 | Test linear interpolation with "interp1" function.

§ Ex20 | Test multiplication between polynomials with "conv" function.

§ Ex21 | Test reading of Excel file (.xlsx) with "readtable" function.

§ Ex22 | Test custom script to simulate real-time software-defined radio (SDR) reception.

§ Ex23 | Test custom script to reconstruct signal through oversampling and low-pass filtering.

§ Ex24 | Test interpolation comparison between custom function and "spline" function.

§ Ex25 | Test TX/RX square-root-raised-cosine (SRRC) filtering with "comm.RaisedCosineTransmitFilter" and comm.RaisedCosineReceiveFilter" functions.

§ Ex26 | Test generation of GIF displaying antenna standing wave trend in time with "imwrite" function.

§ Ex27 | Test custom script for signal spectrum calculation plus estimation of overall power in both time and frequency domains with "fft" and "fftshift" functions.

§ Ex28 | Test custom script for TX-RX chain with QPSK modulation, SRRC filtering and AGC.

§ Ex29 | Test custom functions for converting from/to decimal, hexadecimal and binary formats.

§ Ex30 | Test custom functions for converting from/to volts peak-to-peak (Vpp) and decibel-milliwatts (dBm).

§ Ex31 | Test custom functions for plotting waveform linear (H+V), circular and elliptical polarizations.

§ Ex32 | Test 3rd-part functions to zip/unzip a message.

Published with MATLAB® R2022a