

---

# LTSPICE2MATLAB

## Table of Contents

Calling Convention: .....	1
Inputs: .....	1
Outputs: .....	2
Examples .....	3

`function raw_data = LTspice2Matlab( filename, varargin )`

LTSPICE2MATLAB -- Reads an LTspice IV (21/03/19 Modified to work with LTspice XVII) .RAW waveform file containing data from a Transient Analysis (.tran) or AC Analysis (.ac) simulation, and converts voltages and currents vs. time into Matlab variables. This function can read compressed binary, uncompressed binary, and ASCII file formats. It does not currently support files saved in the Fast Access Format. In the case of compressed binary, the data is automatically uncompressed using fast quadratic point insertion.

LTspice IV is an excellent Spice III simulator & schematic capture tool freely available for download at [www.linear.com/designtools/software](http://www.linear.com/designtools/software). It is optimized for simulation of switching regulators, but can simulate many other types of circuits as well and comes with a wide variety of component models. Note that the LTspice uses a lossy compression format (enabled by default) with user adjustable error bounds.

Use LTSPICE2MATLAB to import LTspice waveforms into Matlab for additional analysis or to compare with measured data.

This function has been tested with LTspice IV version 4.01p, and Matlab versions 6.1 and 7.5. Regression testing has been used to expose the function to a wide range of LTspice settings. Author: Paul Wagner 4/25/2009

## Calling Convention:

`RAW_DATA = LTSPICE2MATLAB( FILENAME );` %Returns all variables found in FILENAME

(or)

`RAW_DATA = LTSPICE2MATLAB( FILENAME, SELECTED_VARS );` %Returns only those variables covered by SELECTED\_VARS Set SELECTED\_VARS to [] to quickly determine the number and names of variables present in FILENAME without actually loading the variables.

(or)

`RAW_DATA = LTSPICE2MATLAB( FILENAME, SELECTED_VARS, N );` Returns variables listed in SELECTED\_VARS, with all waveforms downsampled by N. Set N > 1 to load very large data files using less memory, at the price of degraded waveform accuracy and possible aliasing.

## Inputs:

FILENAME is a string containing the name and path of the LTspiceIV .raw file to be converted.

SELECTED\_VARS (optional) is a vector of indexes indicating which variables to extract from the .raw file. For example, if a .raw file has 14 variables and SELECTED\_VARS is [1 8 9], then the output RAW\_DATA.VARIABLE\_MAT will be a 3 x NUM\_DATA\_PNTS matrix containing waveforms for variables 1, 8, and 9 only. Note that SELECTED\_VARS does not cover the time (or frequency) variable (index 0), which is returned separately in RAW\_DATA.TIME\_VECT (or RAW\_DATA.FREQ\_VECT). Extracting only a subset of variables is a way to use less memory when loading very large simulation files.

If this parameter is not specified, then all variables are returned by default. Setting `SELECTED_VARS` to 'all' will also cause all variables to be returned.

- To quickly determine the number and names of variables present in a .raw file, call `LTspice2Matlab` with `SELECTED_VARS` set to `[]`. In this case, all fields in `RAW_DATA` will be populated, except `.TIME_VECT` (or `.FREQ_VECT`) and `.VARIABLE_MAT`, which will both be empty (`[]`). Since only the header is read, the function call should execute very quickly, even for large files.

`N` (optional) must be a positive integer  $\geq 1$ . If `N` is specified, then `SELECTED_VARS` must also be specified. If `N` is unspecified, it defaults to 1, which does not change the sampling rate. If this value is 2 or larger, the returned voltage, current, and time data will be downsampled by keeping every `N`-th sample in the original data, starting with the first. Caution: No lowpass filtering is applied prior to downsampling, so aliasing may occur. Also, in many cases `LTspice` saves data with a non-constant sampling rate, in which case downsampling can result in substantial waveform distortion. This option should only be used if the waveform of interest is initially oversampled.

## Outputs:

`RAW_DATA` is a Matlab structure containing the following fields ...

`title`: String containing the title appearing in the .RAW file header.

`date`: String containing the date appearing in the .RAW file header.

`plotname`: String indicating simulation type ('Transient Analysis', 'AC Analysis')

`conversion_notes`: Description of modifications (if any) done to the data during conversion.

`num_variables`: Number of variables (does not include the "time" or "frequency" variable)

`variable_type_list`: A cell of strings indicating the variable type (i.e. voltage, current etc.)

`variable_name_list`: A cell of strings indicating the name of each variable.

`selected_vars`: A vector of indices referencing `VARIABLE_TYPE_LIST` cells, corresponding to each row in `VARIABLE_MAT`.

`num_data_pnts`: Number of data points for each variable.

`variable_mat`: Double precision matrix with `NUM_VARIABLES` rows and `NUM_DATA_PNTS` columns. This matrix contains node voltages (in Volts) and device currents (in Amps) for each variable and each time point listed in `TIME_VECT` (or `FREQ_VECT`). For AC Analysis simulations, `VARIABLE_MAT` will have complex values showing the real and imaginary components of the voltage or current at the corresponding frequency. To convert this to log magnitude and normalized phase representation used in `LTspice` plots, use the following formulas:  $\text{Log\_Magnitude\_dB} = 20 \cdot \log_{10}(\text{abs}(\text{variable\_mat}))$   $\text{Norm\_Phase\_degrees} = \text{angle}(\text{variable\_mat}) \cdot 180/\pi$

`time_vect`: [Field returned for Transient Analysis only] Double precision row vector of time values (in seconds) at each simulation point

(or)

`freq_vect`: [Field returned for AC Analysis only] Double precision row vector of frequency values (in Hz) at each simulation point

Currently this function is able to import results from Transient Analysis (.tran) and AC Analysis (.ac) simulations only.

# Examples

These examples assume you've run a .TRAN simulation in LTspice for a hypothetical file called BASIC\_CIRCUIT.ASC, and that an output file called BASIC\_CIRCUIT.RAW has been created. It also assumes your current Matlab directory is pointing to the directory where the .RAW file is located (or that you prepended the full path to the input parameter FILENAME).

To import BASIC\_CIRCUIT.RAW into Matlab and create a labeled plot of a single variable vs. time:

```
raw_data = LTspice2Matlab('BASIC_CIRCUIT.RAW');
variable_to_plot = 1; %This example plots the first variable in the
plot(raw_data.time_vect, raw_data.variable_mat(variable_to_plot,:), '
title(sprintf('Waveform %s', raw_data.variable_name_list{variable_to_
ylabel(raw_data.variable_type_list{variable_to_plot} );
xlabel('Time (sec)' );
```

To superimpose all variables in BASIC\_CIRCUIT.RAW on a single graph with a legend:

```
raw_data = LTspice2Matlab('BASIC_CIRCUIT.RAW');
plot(raw_data.time_vect, raw_data.variable_mat);
title(sprintf('File: %s', raw_data.title));
legend(raw_data.variable_name_list);
ylabel('Voltage (V) or Current (A)');
xlabel('Time (sec)');
```

To quickly determine the number and names of variables in BASIC\_CIRCUIT.RAW without loading the entire file:

```
raw_data = LTspice2Matlab('BASIC_CIRCUIT.RAW', []);
disp(sprintf(' \n\nThis file contains %.0f variables:\n', raw_data.num
disp(sprintf('NAME          TYPE\n-----'));
disp([char(raw_data.variable_name_list), char(zeros(raw_data.num_vari
```

%%Code

```
function raw_data = LTspice2Matlab( filename, varargin )
    raw_data = []; %Initialize the output structure.
    if nargin==0,
        error('LTspice2Matlab takes 1, 2, or 3 input parameters.
Type "help LTspice2Matlab" for details' );
    elseif nargin==1,
        selected_vars = 'all';
        downsamp_N = 1;
    elseif nargin==2,
        selected_vars = varargin{1};
        if ischar(selected_vars), selected_vars =
lower(selected_vars); end
        downsamp_N = 1;
    elseif nargin==3,
        selected_vars = varargin{1};
        if ischar(selected_vars), selected_vars =
lower(selected_vars); end
        downsamp_N = varargin{2};
    else
```

```

        error( 'LTspice2Matlab takes only 1, 2, or 3 input parameters.
Type "help LTspice2Matlab" for details' );
    end

    if length(downsamp_N)~=1 | ~isnumeric(downsamp_N) |
isnan(downsamp_N) | mod(downsamp_N,1)~=0.0 | downsamp_N<=0,
        error( 'Optional parameter DOWNSAMP_N must be a positive
integer >= 1' );
    end

    filename = fliplr(deblank(fliplr(deblank(filename)))); %Remove
leading and trailing spaces from filename.
    %fid = fopen(filename, 'rb'); %ltspiceiv
    fid = fopen(filename, 'rb', 'n', 'UTF16LE');
    if length(fid)==1 & isnumeric(fid) & fid==-1,
        %try to append ".raw" to the file name ...
        %fid = fopen(sprintf( '%s.raw', filename ), 'rb'); %ltspice iv
        fid = fopen(sprintf( '%s.raw', filename ), 'rb', 'n', 'UTF16LE');
        if length(fid)==1 & isnumeric(fid) & fid==-1,
            error( sprintf( 'Could not open file "%s"', filename ) );
        end
    end

    [filename, the_permission, machineformat] = fopen(fid);
    %Load header tags & information
    variable_name_list = {}; variable_type_list = {}; %These
include voltages and currents only. Does not include the time
vector.
    variable_flag = 0;
    file_format = '';
    while 1,
        the_line = fgetl(fid);
        if length(the_line)==1 & isnumeric(the_line) &
double(the_line)==-1,
            try fclose( fid ); catch end
            error( sprintf( 'Format error in LTspice file "%s" ... End
of file unexpectedly encountered', filename ) );
        end

        the_line = char(the_line);

        if length(strfind( the_line, 'Binary:' ))~=0, file_format
= 'binary'; break; end
        if length(strfind( the_line, 'Values:' ))~=0, file_format
= 'ascii'; break; end

        if variable_flag==0, %Non-variable header section
            if length(the_line)==0, colon_index = [];
            else, colon_index = find( the_line == ':' ); end
            if length(colon_index)==0,
                try fclose( fid ); catch end

```

```

        error( sprintf( 'Format error in LTspice file "%s"',
filename ));
        end
        var_name = the_line(1:(colon_index(1)-1));
        var_value =
fliplr(deblank(fliplr(deblank(the_line((colon_index(1)+1):end)))));
        vn_keep_index = find( var_name~=' ' & var_name~='.' &
var_name~=char(9) & var_name~=char(10) & var_name~=char(13) );
        var_name = lower(var_name(vn_keep_index));
        if length(var_name)==0 | (var_name(1)>='0' &
var_name(1)<='9'),
            try fclose( fid ); catch end
            error( sprintf('Format error in LTspice file "%s" ...
Bad tag name found', filename ));
            end
            if strcmpi( var_name, 'variables' ) |
strcmpi( var_name, 'variable' ), variable_flag = 1; continue; end
            value_try = str2num(var_value);
            try
                if length(value_try)==0, raw_data =
setfield( raw_data, var_name, var_value );
                else raw_data = setfield( raw_data, var_name,
value_try ); end
            catch
                try fclose( fid ); catch end
                error( sprintf('Format error in LTspice file "%s" ...
Bad tag name found', filename ));
            end

            else %Variable header section
                leading_ch_index = find( (the_line(1:end-1)=='
' | the_line(1:end-1)==char(9)) & (the_line(2:end)~=' ' &
the_line(2:end)~=char(9)) );
                if length(leading_ch_index)~=3,
                    try fclose( fid ); catch end
                    error( sprintf('Format error in LTspice file "%s" ...
Wrong number of columns in the variable define section', filename ));
                end

                part1 =
fliplr(deblank(fliplr(deblank(the_line( (leading_ch_index(1)+1) :
leading_ch_index(2) )))));
                part2 =
fliplr(deblank(fliplr(deblank(the_line( (leading_ch_index(2)+1) :
leading_ch_index(3) )))));
                part3 =
fliplr(deblank(fliplr(deblank(the_line( (leading_ch_index(3)+1) :
end )))));

                if str2num(part1)~=length(variable_name_list),
                    try fclose( fid ); catch end
                    error( sprintf('Format error in LTspice file "%s" ...
Inconsistency found in the variable define section', filename ));
                end

```

```

        variable_name_list{end+1} = part2;
        variable_type_list{end+1} = part3;
    end
end

%Check raw_data structure for required fields
expected_tags =
{'title', 'date', 'plotname', 'flags', 'novariables', 'nopoints'
};
expected_tags_full = {'Title', 'Date', 'Plotname', 'Flags', 'No.
Variables', 'No. Points' };
for q=1:length(expected_tags),
    if ~isfield( raw_data, lower(expected_tags{q}) ),
        try fclose( fid ); catch end
        error( sprintf('Format error in LTspice file "%s" ... tag
"%s" not found', filename, expected_tags_full{q} ));
    end
end

raw_data.conversion_notes = '';
raw_data.num_data_pnts = raw_data.nopoints; raw_data =
rmfield( raw_data, 'nopoints' );
raw_data.num_variables = raw_data.novariables-1; raw_data =
rmfield( raw_data, 'novariables' );
%"raw_data.num_variables" does not include the time vector (index
0 in the .raw file)
if isfield( raw_data, 'command' ), raw_data =
rmfield( raw_data, 'command' ); end
if isfield( raw_data, 'backannotation' ), raw_data =
rmfield( raw_data, 'backannotation' ); end
if isfield( raw_data, 'offset' ),
    general_offset = raw_data.offset; %(sec)
    raw_data = rmfield( raw_data, 'offset' );
else
    general_offset = 0.0;
end

raw_data.variable_name_list = {variable_name_list{2:end}};
%cut off the time variable.
raw_data.variable_type_list = {variable_type_list{2:end}};

simulation_type = '';
if length(strfind( lower(raw_data.plotname), 'transient
analysis' ))~=0, simulation_type = '.tran'; %SUPPORTED
elseif length(strfind( lower(raw_data.plotname), 'ac analysis'
))~=0, simulation_type = '.ac'; %SUPPORTED
elseif length(strfind( lower(raw_data.plotname), 'dc transfer
characteristic' ))~=0, simulation_type = '.dc'; %This is a DC
sweep (Not supported)
elseif length(strfind( lower(raw_data.plotname), 'operating point'
))~=0, simulation_type = '.op'; %This is a DC
operating point (Not supported)
end

```

```

        if length(simulation_type)==0 |
~(strcmpi(simulation_type, '.tran') |
strcmpi(simulation_type, '.ac')),
            try fclose( fid ); catch end
            error( 'Currently LTspice2Matlab is only able to import
results from Transient Analysis (.tran) and AC Analysis (.ac)
simulations.' );
        end
        if length(strfind( lower(raw_data.flags), 'fastaccess' ))~=0,
            try fclose( fid ); catch end
            error( 'LTspice2Matlab cannot convert files saved in the "Fast
Access" format.' );
        end
        if strcmpi(simulation_type, '.tran') &
length(strfind( lower(raw_data.flags), 'real' ))==0,
            try fclose( fid ); catch end
            error( 'Expected to find "real" flag for a Transient Analysis
(.tran) simulation. Unsure how to convert the data' );
        end
        if strcmpi(simulation_type, '.tran') &
length(strfind( lower(raw_data.flags), 'forward' ))==0,
            try fclose( fid ); catch end
            error( 'Expected to find "forward" flag for a Transient
Analysis (.tran) simulation. Unsure how to convert the data' );
        end

        if strcmpi(simulation_type, '.ac') &
length(strfind( lower(raw_data.flags), 'complex' ))==0,
            try fclose( fid ); catch end
            error( 'Expected to find "complex" flag for an AC Analysis
(.ac) simulation. Unsure how to convert the data' );
        end
        if strcmpi(simulation_type, '.ac') &
length(strfind( lower(raw_data.flags), 'forward' ))==0,
            try fclose( fid ); catch end
            error( 'Expected to find "forward" flag for an AC Analysis
(.ac) simulation. Unsure how to convert the data' );
        end
        if isfield( raw_data, 'flags' ), raw_data =
rmfield( raw_data, 'flags' ); end

        if ischar(selected_vars),
            if strcmpi(selected_vars, 'all')
| strcmpi(selected_vars, 'everything') |
strcmpi(selected_vars, 'complete') | strcmpi(selected_vars, 'all
variables') | ...
                strcmpi(selected_vars, 'all
vars') | strcmpi(selected_vars, 'every thing') |
strcmpi(selected_vars, 'every'),
                selected_vars = 1:raw_data.num_variables; %Return all
variables
            else
                try fclose( fid ); catch end
    
```

```

        error( 'Bad value for optional input parameter
SELECTED_VARS' );
    end
end

if size(selected_vars,1)==0 | size(selected_vars,2)==0,
    raw_data.selected_vars = [];
    raw_data.variable_mat = [];
    raw_data.time_vect = [];
    try fclose( fid ); catch end
    return;
end
if size(selected_vars,1)>1 & size(selected_vars,2)>1,
    try fclose( fid ); catch end
    error( 'SELECTED_VARS must be a row or column vector, not a
matrix' );
end
if length(find(selected_vars==0))~=0,
    try fclose( fid ); catch end
    error( 'The time vector (index 0) is returned separately. \n
Values in input parameter SELECTED_VARS must be positive integers >=
1 and <= NUM_VARIABLES' );
end
non_integer_index = find(isnan(selected_vars) |
~isnumeric(selected_vars) | mod( selected_vars, 1 )~=0.0);
if length(non_integer_index)~=0,
    try fclose( fid ); catch end
    error( 'Values in input parameter SELECTED_VARS must be
positive integers >= 1 and <= NUM_VARIABLES' );
end
missing_index = find( ~ismember( selected_vars,
1:raw_data.num_variables ) );
if length(missing_index)~=0,
    try fclose( fid ); catch end
    error( 'Error in input parameter SELECTED_VARS ... Out of
range value(s) found' );
end

selected_vars = unique(selected_vars); %remove duplicates and
sort in ascending order.
raw_data.selected_vars = selected_vars;

NumPnts = raw_data.num_data_pnts;
NumPnts_DS = floor(NumPnts/downsamp_N);
raw_data.num_data_pnts = NumPnts_DS; %Updated # of points
NumVars = raw_data.num_variables+1;

%READ IN THE ACTUAL WAVEFORM DATA
if strcmpi(file_format, 'binary'),
    binary_start = ftell(fid); %start of binary data section.

    if strcmpi( simulation_type, '.tran' ),

```



---

```

        % For Transient Analysis simulations, the time data is
        stored in double precision floating point binary format,
        % and everything else is stored in single precision
        format.

        %Extract the binary data in the fewest possible number of
        contiguous blocks
        if length(selected_vars)>1,
            g_border = find( [2, diff(selected_vars), 2]~=1 );
            block_list = {};
            for k=1:length(g_border)-1, block_list{k} =
g_border(k):(g_border(k+1)-1); end
        else
            block_list = {1:length(selected_vars)};
        end
        raw_data.variable_mat = zeros(length(selected_vars),
NumPnts_DS); %Initialize.
        for k=1:length(block_list),
            target_var_index = selected_vars(block_list{k});
            fseek(fid, binary_start +
(target_var_index(1)+1)*4, 'bof');
            TVIL = length(target_var_index);
            bytes_skip = (NumVars+1-TVIL)*4 +
(downsamp_N-1)*(NumVars+1)*4;
            precision_str = sprintf('%0f*float',TVIL);
            raw_data.variable_mat(block_list{k},:) =
reshape( fread(fid, NumPnts_DS*TVIL, precision_str, bytes_skip,
machineformat), TVIL, NumPnts_DS );
        end
        fseek(fid, binary_start, 'bof'); %rewind to start, then
        extract the time vector.
        raw_data.time_vect = fread( fid, NumPnts_DS, 'double',
(NumVars-1)*4 + (downsamp_N-1)*(NumVars+1)*4, machineformat ).';
        if downsamp_N==1, raw_data.conversion_notes = 'Converted
from Binary format';
        else raw_data.conversion_notes = sprintf( 'Converted from
Binary format. Downsampled from %0f to %0f points', NumPnts,
NumPnts_DS ); end

        elseif strcmpi( simulation_type, '.ac' ),
            % For AC Analysis simulations, the frequency data is
            stored in double precision floating point binary format (8 bytes),
            % and the variables are stored as complex double precision
            arrays (8 bytes real followed by 8 bytes imag)

            %Extract the binary data in the fewest possible number of
            contiguous blocks
            if length(selected_vars)>1,
                g_border = find( [2, diff(selected_vars), 2]~=1 );
                block_list = {};
                for k=1:length(g_border)-1, block_list{k} =
g_border(k):(g_border(k+1)-1); end
            else
                block_list = {1:length(selected_vars)};
            end

```

---

---

```

        raw_data.variable_mat = zeros(length(selected_vars),
NumPnts_DS); %Initialize.
        if prod(size(raw_data.variable_mat))~=0,
raw_data.variable_mat(1,1) = 0.0 + j*0.0; end %Allocate memory for
complex double.
        for k=1:length(block_list),
            target_var_index = selected_vars(block_list{k});
            fseek(fid, binary_start +
target_var_index(1)*16, 'bof');
            TVIL = length(target_var_index);
            bytes_skip = (NumVars-TVIL)*16 +
(downsamp_N-1)*NumVars*16;
            precision_str = sprintf('%.0f*double',TVIL*2);
            temp_buff = reshape(fread(fid, NumPnts_DS*TVIL*2,
precision_str, bytes_skip, machineformat), TVIL*2, NumPnts_DS );
            raw_data.variable_mat(block_list{k},:) =
temp_buff(1:2:end-1,:) + j*temp_buff(2:2:end,:);
            clear temp_buff;
        end
        fseek(fid, binary_start, 'bof'); %rewind to start, then
extract the time vector.
        raw_data.freq_vect = fread( fid, NumPnts_DS, 'double',
(NumVars-1)*16 + 8 + (downsamp_N-1)*NumVars*16, machineformat ).';

    else
        try fclose( fid ); catch end
        error( sprintf('Simulation type (%s) not currently
supported', simulation_type ));
    end

    elseif strcmpi(file_format, 'ascii' ),

        if strcmpi( simulation_type, '.tran' ),
            %Format: point number, time value, var1, var2, var3 ...
varN
            raw_data.variable_mat = fscanf( fid, '%g',
[raw_data.num_variables+2, raw_data.num_data_pnts] ); %matrix is
filled in column order.
            if (size(raw_data.variable_mat,1)~=raw_data.num_variables
+2) | (size(raw_data.variable_mat,2)~=raw_data.num_data_pnts),
                error( sprintf('Format error in ASCII Transient
Analysis LTspice file "%s" ... Incorrect number of data values read',
filename ));
            end
            raw_data.time_vect =
raw_data.variable_mat(2,1:downsamp_N:end);
            raw_data.variable_mat =
raw_data.variable_mat(2+selected_vars,1:downsamp_N:end);

            elseif strcmpi( simulation_type, '.ac' ),
                %Format: point number, freq value, 0, var1 real, var1
imag, var2 real, var2 imag, var3 real, var3 imag ... varN real, varN
imag

```

---

```

        all_data = fread( fid, inf, 'uchar' );
        all_data( find( all_data == ',' ) ) = sprintf( '\t' );
        %Replace commas with tab characters
        raw_data.variable_mat = sscanf( char(all_data), '%g',
[3+2*raw_data.num_variables, raw_data.num_data_pts] );
        clear all_data;
        %raw_data.variable_mat = fscanf( fid, '%g',
[3+2*raw_data.num_variables, raw_data.num_data_pts] );    %matrix is
filled in column order.

        if
(size(raw_data.variable_mat,1)~=(3+2*raw_data.num_variables)) |
(size(raw_data.variable_mat,2)~=raw_data.num_data_pts),
            error( sprintf('Format error in ASCII AC Analysis
LTspice file "%s" ... Incorrect number of data values read',
filename ));
        end
        raw_data.freq_vect =
raw_data.variable_mat(2,1:downsamp_N:end);
        raw_data.variable_mat =
raw_data.variable_mat(3+selected_vars*2-1,1:downsamp_N:end) +
j*raw_data.variable_mat(3+selected_vars*2,1:downsamp_N:end);

        else
            try fclose( fid ); catch end
            error( sprintf('Simulation type (%s) not currently
supported', simulation_type ));
        end

        if downsamp_N==1, raw_data.conversion_notes = 'Converted from
ASCII format';
        else raw_data.conversion_notes = sprintf( 'Converted from
ASCII format.  Downsampled from %.0f to %.0f points', NumPnts,
NumPnts_DS ); end

        else
            try fclose( fid ); catch end
            error( sprintf('Format error in LTspice file "%s" ... Data
type ID tag not found', filename ));
        end
        try fclose( fid ); catch end

        %Deal with potential compression in Transient Analysis simulations
        if strcmpi( simulation_type, '.tran' ) &
(min(diff(raw_data.time_vect)) < 0.0),    %Check to see if the time
vector is monotonically increasing.

            if downsamp_N~=1,    %If we have already downsampled then we
can't uncompress.
                raw_data.time_vect = abs(raw_data.time_vect);

            else

```

```

        %The binary file contains 2nd order compression ...
use 2nd-order interpolation to add data points in the vicinity of
negative time points
        t_vect = raw_data.time_vect;    %We will add in the offset
later.
        neg_pnt_index = find( t_vect < 0.0 &
[0,ones(1,length(t_vect)-1)] );
        t_vect = abs(t_vect);
        x1 = t_vect(neg_pnt_index-1);  x2 = t_vect(neg_pnt_index);
        x3 = t_vect(neg_pnt_index+1);
        x_new = [(2*x1 + x2)/3; (x1 + 2*x2)/3; (2*x2 + x3)/3;
(x2 + 2*x3)/3];    %New sample points
        t_vect_big = NaN*zeros(6,length(t_vect));
        t_vect_big(1,:) = t_vect;
        t_vect_big(4,neg_pnt_index) = t_vect(neg_pnt_index);
        t_vect_big(1,neg_pnt_index) = NaN;
        t_vect_big([2 3 5 6],neg_pnt_index) = x_new;
        full_index = find(~isnan(t_vect_big));
        time_vect_new = t_vect_big(full_index).';    %This is the
new time vector with the inserted points.
        t_vect_big([1,4],:) = NaN;
        nan_vect = isnan(t_vect_big(full_index));
        new_index = find( ~nan_vect );    %Index into time_vect_new
indicating the new points only.
        old_index = find( nan_vect );
        clear t_vect t_vect_big full_index nan_vect;
        x1sqr = repmat( x1.^2, [4,1] );  x2sqr = repmat( x2.^2,
[4,1] );  x3sqr = repmat( x3.^2, [4,1] );
        x1 = repmat( x1, [4,1] );  x2 = repmat( x2, [4,1] );  x3 =
repmat( x3, [4,1] );
        denom = (x1sqr-x2sqr).*(x2-x3) - (x2sqr-x3sqr).*(x1-x2);
        r1 = (x_new.^2 - x1sqr)./denom;
        r2 = (x_new - x1)./denom;
        p1 = (x2-x3).*r1 - (x2sqr-x3sqr).*r2;
        p3 = (x1-x2).*r1 - (x1sqr-x2sqr).*r2;
        p2 = -p1 - p3;
        p1 = p1 + 1;
        clear x_new x1sqr x2sqr x3sqr x1 x2 x3 denom r1 r2;
        raw_data.variable_mat(:,end+1:length(time_vect_new)) =
0.0;    %Init the memory
        for k=1:size(raw_data.variable_mat,1),
            y_vect =
raw_data.variable_mat(k,1:length(raw_data.time_vect));
            raw_data.variable_mat(k,old_index) = y_vect;
            y_new = repmat(y_vect(neg_pnt_index-1),
[4,1]).*p1 + repmat(y_vect(neg_pnt_index),[4,1]).*p2 +
repmat(y_vect(neg_pnt_index+1),[4,1]).*p3;
            raw_data.variable_mat(k,new_index) = y_new(:).';
        end
        raw_data.time_vect = time_vect_new;

clear time_vect_new y_vect y_new new_index old_index neg_pnt_index p1 p2 p3;

```

```
        raw_data.conversion_notes = sprintf( 'Converted from
Binary format with 2nd Order compression.  Upsampled waveforms from
%.0f to %.0f points', ...
        raw_data.num_data_pnts, length(raw_data.time_vect) );
        raw_data.num_data_pnts = length(raw_data.time_vect);
    end

end

    if isfield( raw_data, 'time_vect' ),        raw_data.time_vect =
raw_data.time_vect + general_offset;
    elseif isfield( raw_data, 'freq_vect' ),    raw_data.freq_vect =
raw_data.freq_vect + general_offset;
    end
```

*Published with MATLAB® R2018b*