0.1 Welcome

Welcome to the 2008 Workshop on Fiber Optic Distributed Temperature Sensing (DTS) for Ecological Characterization. In this section we will look at some basic routines in Matlab for visualizing, processing, and understanding the information contained in our DTS data. This is the second hands on workshop and I want to thank Fred Day-Lewis of the USGS who taught this section last year. We will be building on Fred's initial approach. This section assumes a rudimentary knowledge of Matlab. If you think you need more background with Matlab, please take at look at some of the introductory tutorials I have posted at the 'hmfwiki.org' site below.

The programs, documents, and example data sets we will be discussing here are all available at the web site:

http://www.hmfwiki.org 'hmf' stands for 'Hydrologic Measurement Facility.'

Please extend and build on these rudimentary examples, and share your results with others. I can set up a 'open source,' source code control system to help with that, in the mean time just pass on anything you come up with to me and I will integrate it into the current toolbox. I can be contacted at:

Email: *nbt.osu@gmail.com*

Websites: *http://www.drchaos.net, http://oregonstate.edu/~tufillan/*

0.2 Outline

This tutorial will introduce you to using Matlab to perform a few basic activities with your DTS data such as:

- Loading your data into Matlab,
- Producing rich graphics to examine your temperature data,
- Field based re-calibration of your temperature data, and
- Editing and processing your temperature data to bring out the salient features.

The tools provided here will assist you with these activities, but are really only a meant to provide a helping hand. For a successful environmental monitoring project, it is hoped that you can build on what you learn here to visualize and process your data to address the specific questions of your study.

This tutorial does not cover a lot of very important additional topics such as physical location, or "registration," of your temperature data (mapping the 'internal fiber distance' to a set of coordinates, longitude and latitude), storing your data in a database (like mySQL), integration of the DTS data with other important databases (e.g. maps, fish counts, weather ...), i.e. 'data fusion,' or development of an application rich 'presentation layer' based on web services (e.g. the CUAHSI Hydrologic Information System (HIS)).

For this tutorial we will walk through scripts illustrating the tasks outlined above with typical data sets. Along the way we will describe the data structures and functions currently part of the Matlab 'dts toolbox' — which is still pretty rustic.

0.3 Getting Started

Download the package 'dts_tools.zip' from 'http://www.hmfwiki.org'. Unzip the package in a directory of your choice and notice that it creates a number of directories including:

```
dts_data
dts_docs
dts_toolbox
dts_workspace
```

Change directory to 'dts_workspace' and run the script:

```
start_up_dts_tools
```

which adds the above directories to your current Matlab path. Take a gander at the various dts directories to get an overview of what they contain, and then return to the dts_workspace directory.

By the way, I'd like to mention that it is very helpful to include 'underscores' for spaces in filenames, as in 'my_data_file.dat.' This practice helps to reduce some of the grief when moving files between different operating systems. Another good practice is to provide file names to data files that will sort correctly in directory listing, for instance, I often include a numeric date in the file name to help with this as in:

```
data_file_2008_06_15.dat
data_file_2008_06_16.dat
...
data_file_2008_06_25.dat
```

0.4 Example 1 — Loading, camera, action.

Here is the first example script, it introduces many of the functions we will be using.

```
% Example Script
% Read in Sensortran data set and examine data
% Nick Tufillaro June 2008

% Load data into matlab

disp('EXAMPLE SCRIPT: READ IN DATA, BUILD dts_experiment DATA STRUCTURE, AND DO VARIOUS PLOTS');
disp('Press ENTER to continue at each step');
disp('READ DATA AND CREATE dts_experiment DATA STRUCTURE');
disp(char(10));  % Creates a new line
pause;
notes = 'Data example from Chadi'
dts_exper = read_sensortran_csv('sensortran_data_example_1935TO1945.csv', notes)

% Plot mean and standard deviation

disp('PLOT SUMMARY STATISTICS (MEAN and STANDARD DEVIATION)');
disp(char(10));
pause;
[stdtemp, meantemp] = plot_stats(dts_exper)

% Plot color maps of temperature

disp('PLOT TEMPERATURE');
disp('ROTATE FIGURE (2)');
disp('SAVE IMAGE in FILE chadi_temp_plot');
disp(char(10));
```

```
pause;
minimum_temperature = 0; maximum_temperature = 25;
plot_temperature(dts_exper, minimum_temperature, maximum_temperature, 'distance (meters)', ...
    'time (julianday)', 'Temperature Plot', 'chadi_temp_plot')

% Make a movie of temperature snapshots

disp('MAKE MOVIE OF DATA SET');
disp('SAVE .avi FILE IN chadi_temp_plot');
disp(char(10));
pause;
pause_time = 0.1    % pause to slow down movie
start_position = 0  % start position along the fiber (in meters)
stop_position = 800 % stop position along the fiber (in meters)
animate_temperature(dts_exper,'chadi_movie',pause_time, start_position, stop_position, ...
    minimum_temperature, maximum_temperature)
```

To run this first example just type:

```
example_script_1
```

in the 'dts_workspace.' There is quite a lot going on here so let's take it step by step.

0.4.1 Loading the data

Each DTS manufacturer provides a different set of information in different formats. So the task of just getting the relevant data into Matlab can be a chore. The toolbox provides two data loading programs for both the Agilent and SensorTran instruments, others can be added during the workshop as needed. The import function for SensorTran is:

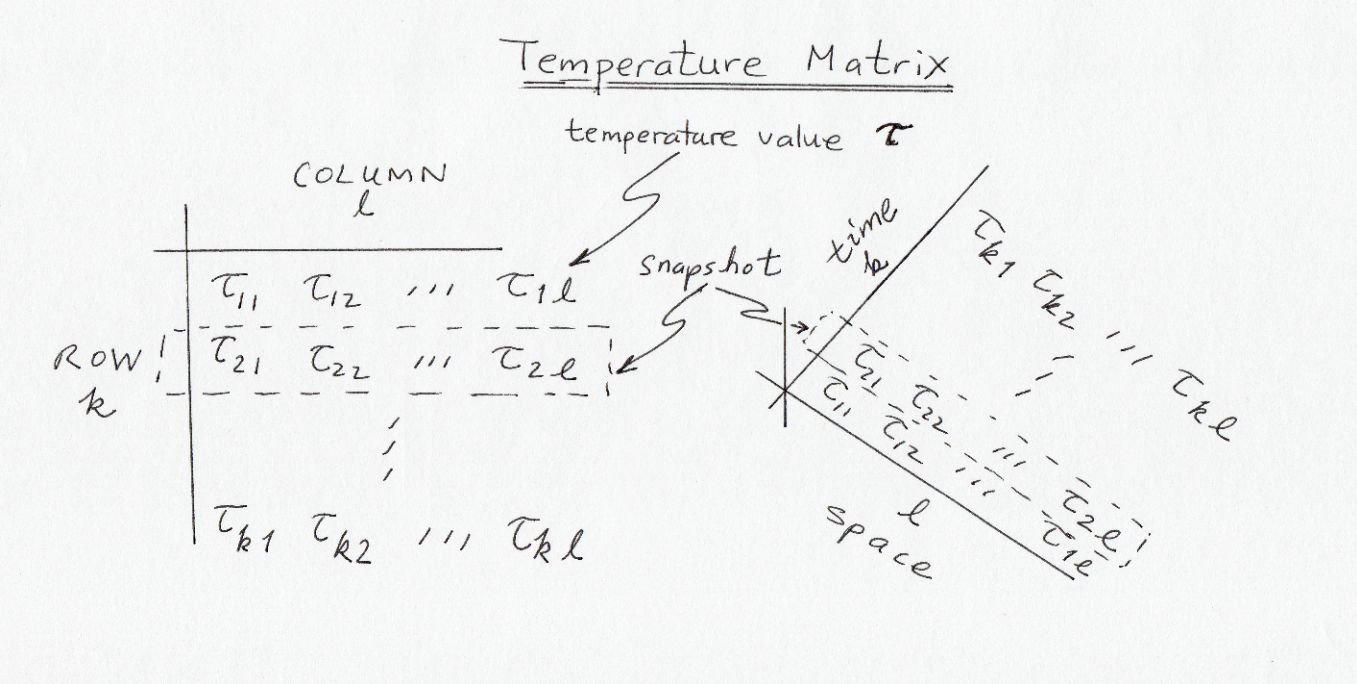
```
dts_exper = read_sensortran_csv('sensortran_data_example_1935TO1945.csv', 'data_from_Chadi')
```

It takes two arguments, both of which are Matlab 'string' variables. The easiest way to make a string variable it to put the single quotes around the text. In the example above the first argument is a file name for a SensorTran '.csv' file (comma separated), and the second argument is a place holder for any brief notes you want to write about the experiment. The 'read' program should be able to parse the SensorTran data file properly and grab the correct data — if not, you will need to take a look at the function in the directory 'dts_toolbox' and create a suitable variant.

The information in the manufacture's data file(s) contains lots of details about the instrument and calibration along with the temperature data. For our purpose we only need a small subset of this data to get started, namely: the temperature data, the date, time, and channel of each data set, and a brief description about the data. We will package all of this information and keep it together in a Matlab 'structure.' This structure will then get passed to functions, instead of the individual pieces of data.

First some terminology. We will call each individual temperature measurement associated with a specific time a '*snapshot*.' This is the minimal piece of data and it consists of an array of temperatures measurements along the fiber all done at the same time. We usually tell the instrument to collect a series of these 'snapshots,' at equally spaced time intervals. We call this series of snapshots a '*dts experiment*,' or record. A dts experiment is naturally organized as a matrix, which we call the '*temperature data*' or '*temperature matrix*.' We will adopt the convention that the columns (x-axis) of this matrix represent spatial position — length along the fiber, and the rows (y-axis) represent different times or snapshots. The reason for this convention is that in a plot like that shown in Figure 1, we can imagine the fiber as spread out along the x-axis, and as new temperature data comes at different times we just move the fiber up the y-axis. The 'temperature matrix' thus has a very natural geometrical representation. Notice that

'length' and 'time' information are *not* part of the temperature matrix, it will be part of the dts_experiment structure. In matrix notation, the temperature matrix is:



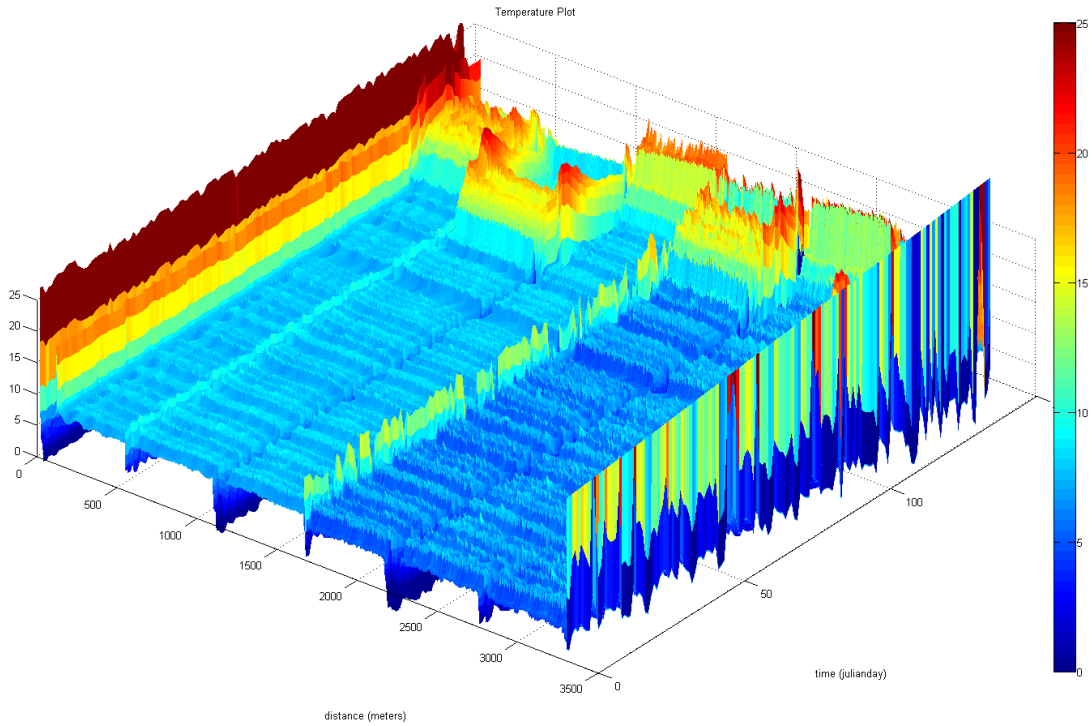


Figure 1: Surface plot of temperature matrix. The x-axis is position along the fiber in meters, and the y-axis shows snapshots at different times, and the z-axis is the value of the temperature matrix at a specific location and time. The color bar on the right shows temperature in degrees C.

To each temperature matrix we would also like to associate additional data such as time and date stamps, length (both the internal length across the fiber, as well as real world position), etc. To keep all this additional information organized we will define and create a dts_experiment data structure in matlab.

0.4.2 Data structure: 'dts_experiment'

The 'dts_experiment' is an evolving definition. Looking in the 'read_sensortran_csv.m' file we see something like:

```
% fill dts_experiment structure
dts_experiment.notes      = experimentnotes;     % string variable with user notes
dts_experiment.datestring = datestring;          % Matlab date string array for each snapshot
dts_experiment.juliandate = juliandate;          % Numeric array contain Julian day, time
dts_experiment.k          = k;                   % Integer array of time clicks
dts_experiment.channel    = channel;             % Integer array of measurement channel
dts_experiment.x          = x;                   % Numeric array of distance along fiber in meters
dts_experiment.l          = l;                   % Integer array of distance ticks
dts_experiment.tempdata   = tempdata;            % Numeric array of temperature matrix
dts_experiment.file       = fname;               % String array, filename used to import data
dts_experiment.mean       = mean_tempsnapshot';  % Mean temperature at each snapshot location
dts_experiment.std        = std_tempsnapshot';   % Standard deviation at each location
```

If you have not used a Matlab structure before, it is pretty simple. To add add or get data from the structure Matlab uses the lower 'dot' notation. The first part of the structure is the dts_experiment.notes (pronounced 'dts experiment DOT notes') which is populated by a string array. Next are some arrays, both string and numeric variables, associated with date and time information. Next is a 'channel' array. Each DTS instrument can have more than one 'channel,' that is places where a fiber can connect. We want to keep track of which snapshot comes from which channel, and this array has that information. Next are some arrays with the length information. The length information is both an integer count 'l' (el), as well as an actual distance in meters, 'x'. The distance is computed from $x = l * xscalefactor$. The $xscalefactor$ is read from each manufactures data or configuration file. For example, it is called $ScaleResolution$ in the Agilent instrument files. And finally, the actual temperature matrix, and some arrays with summary statistics about the data.

This is the core data of a dts experiment. Other information can be added to this structure by just adding new elements to the structure, such as 'registration' information, the longitude and latitude of where the fiber exist on the earth.

In the example script above, if we want to extract the temperature matrix, we would type:

```
temperature_data_matrix = dts_exper.tempdata;
```

or to extract the length and time associated with the temperature matrix, we type:

```
length = dts_exper.x;
time   = dts_exper.juliandate
```

As as simple exercise we can try to plot the variables extracted from the dts_experiment structure. Try:

```
figure;
plot(length);
figure;
plot(time);
figure;
plot(temperature_data_matrix);
figure;
plot(length, dts_exper.mean);
```

and describe the results.

This is probably a good time to mention the 'help' and 'doc' commands in Matlab. To find out more about Matlab structures type:

```
doc struct
```

or just Google 'Matlab structures tutorial' and see what pops up.

Perhaps even more useful, you can take a quick look at what a structure holds by just typing its name:

```
>> dts_exper

dts_exper =

         notes: 'Data example from Chadi'
    datestring: [146x20 char]
     julianday: [146x1 double]
             k: [146x1 double]
       channel: [146x1 char]
             x: [1x3285 double]
             l: [1x3285 double]
      tempdata: [146x3285 double]
          file: 'sensortran_data_example_1935TO1945.csv'
          mean: [1x3285 double]
           std: [1x3285 double]

>>
```

The information above tells us that there are 146 snapshots (time stamps) and the cable is 3285 steps long. To see the actual time or (fiber) distance, we can look at the variables dts_exper.datestring and dts_exper.x.

0.4.3 Dates and Time

Matlab has its own conventions for storing dates and times. To learn about these type:

```
doc datestr
```

Matlab time

You will see that Matlab has three primary representations of dates and times:

```
Date String:           '24-Oct-2003 12:45:07'
Date Vector:           [2003  10  24  12  45  07] % yyyy mm dd hh mm ss
Serial Date Number:    7.3188e+005
```

The Date String — a string variable, a Date Vector, and a Serial Date time which is a numerically ordered time index and can be used like a Juliandate, namely it can be used to sort and order the data in a proper temporal ordering. In our dts_experiment data structure each snapshot is associated with a date string and Juliandate. Matlab has functions for converting the date string to many different string or numeric formats (oddly, it looks like Matlab does not have one for Julian time or Modified Julian Time). To see some of the date strings in our example data set and their corresponding Juliandates type:

```
>> dts_exper.datestring(1:4,:)

ans =

21-Mar-2008 19:35:01
21-Mar-2008 19:35:04
21-Mar-2008 19:35:10
21-Mar-2008 19:35:17

>> dts_exper.juliandate(1:4)

ans =

   80.8160
   80.8160
   80.8161
   80.8162

>>
```

Julian date

For our purpose a Julian date is used to order, date, and time-stamp the temperature snapshots. Look up in 'wiki' the meanings of Julian day, Julian date, and modified Julian day if you are not familiar with this dating system.

The Julian date is defined by:

$$JD = JDN + \frac{hour - 12}{24} + \frac{minute}{(60 * 24)} + \frac{second}{(24 * 60 * 60)} \tag{1}$$

where JD is the Julian date, and JDN is the Julian Day Number (defined as the integer number of days that have elapsed since the initial epoch defined as noon Universal Time (UT) Monday, January 1, 4713 BC in the proleptic Julian calendar).

For our purpose, we will define a dts 'workshop date' as the number of days plus the time of day — measured in fractions of a second of a day — since the first day of January 2008. The following table could be helpful:

		DTS Workshop June 2008	
Date	Time	DTS Workshop Date	Julian Date
1 June	00:00:00	153.00	JD 2454618.50
2 June	00:00:00	154.00	JD 2454619.50
2 June	12:00:00	154.50	JD 2454620.00
2 June	18:00:00	154.75	JD 2454620.25
3 June	00:00:00	155.00	JD 2454620.50
4 June	00:00:00	156.00	JD 2454621.50
5 June	00:00:00	157.00	JD 2454622.50
6 June	00:00:00	158.00	JD 2454623.50

0.4.4 Summary statistics

After importing the data into matlab, the next step is to take a quick look at it with summary statistics. Figure 2 shows the output from the function:

```
[stdtemp, meantemp] = plot_stats(dts_exper)
```

The input is just the dts_experimental data structure, and the output is the standard deviation and mean temperature. The mean is just jargon for the average value of the a signal. The standard deviation is a measure of the noise in a signal. The standard deviation and mean are computed by averaging over time, that is they represent values of the temperature at a fixed location while averaging over time. The mean is useful to identify any regions that are significantly different in temperature. These fiber sections could be sections that are coiled in an ice bath (the sections near zero in Figure 2(a)), sections that are 'out-of-stream,' in air (typically at the ends of the fiber).

The 'plot_stats()' also produces three sets of graphics showing mean and standard deviation taken over both space and time. The last graph, the third graphic produced by this function, is usually the one of most interest and is shown in Fig. 2.

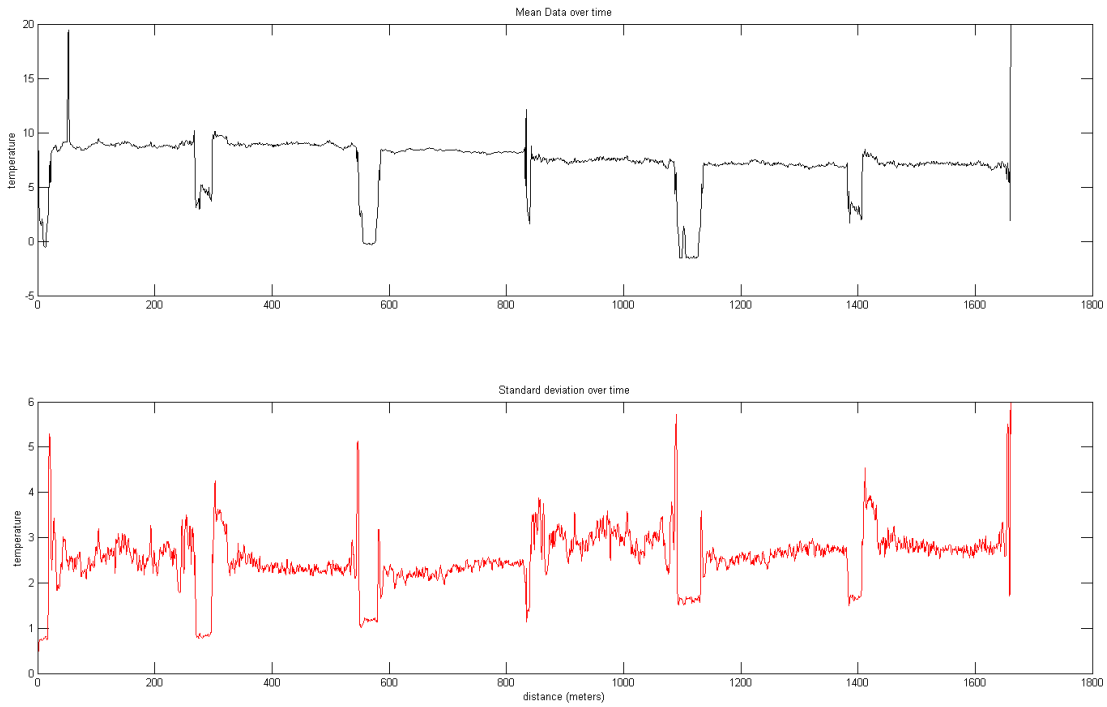


Figure 2: Plot of mean (top) and standard deviation (bottom) of snapshots. Sections of fiber where the mean value is near 0 degrees are probably in an ice-bath, and sections with an extended elevated temperature are probably out of the medium (stream, soil ...), in air. It looks like there is a 'sharp bend' in the fiber at the middle, so this fiber probably 'loops back' to retrace its path.

0.4.5 Temperature Plots

Now on to the picture show. The function plot_temperature():

```
minimum_temperature = 0; maximum_temperature = 25;
plot_temperature(dts_exper, minimum_temperature, maximum_temperature, 'distance (meters)', ...
```

```
'time (julianday)', 'Temperature Plot', 'chadi_temp_plot')
```

produces the colorful pictures shown in Figures 1 and 3. The input is the dts_experiment structure again, along with a number of mandatory strings for labeling the axis variables. The range of the temperature axis (z-direction) is set with the 2nd and 3rd numeric arguments to the plot_temperature() function. All arguments are mandatory in this prototype version of the function. A nice exercise would be to add a method for having arguments being optional.

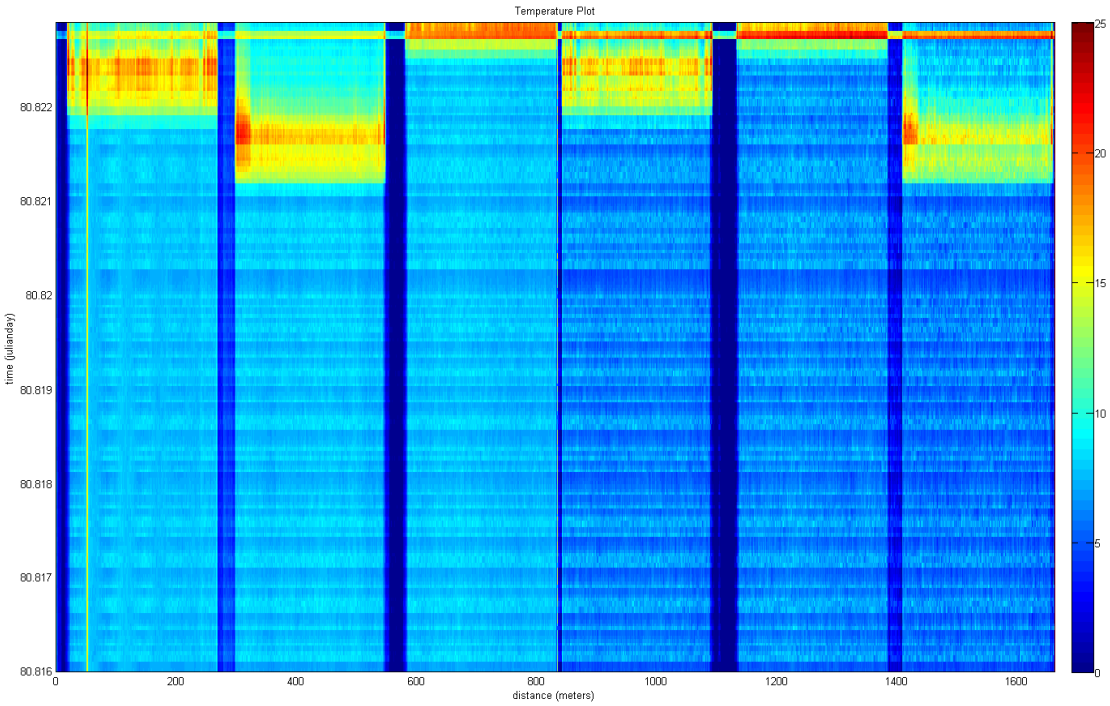


Figure 3: Color map of temperature data. Temperature is indicated in degrees C by the color bar on the right. The horizontal axis spans space (length, meters), and the vertical axis spans time (Juliandates).

0.4.6 Temperature Movie

Last but not least — Action! The last function in the example_script_1 is animate_temperature():

```
pause_time = 0.1
start_position = 0
stop_position = 800
animate_temperature(dts_exper,'chadi_movie',pause_time, start_position, stop_position, ...
    minimum_temperature, maximum_temperature)
```

makes a movie in '.avi' format from the temperature data. Again it has a number of mandatory arguments including string variables for labels and titles, numeric variables for the start and stop position, as well as numeric variables for the temperature range to be plotted. The pause_time arguments can be used for slowing the movie down. The second argument, 'chadi_movie' in this example, is the file name where the movie is placed, so in this case the movie is written to the file 'chadi_movie.avi.'

Matlab constructs the movie by plotting each frame at a time, and then collecting all the frames together to form a motion picture. This can take a while. The window for the temperature plots should not be disturbed during the process.

The various graphic presentations are all built from the same data, but are useful for identifying different features in the data such as possible ground water inputs, or the location of calibration baths.

0.4.7 Many names

Notice that we have been fast and loose with coordinate names — and how the use of a 'structure' enables this behavior. Sometimes we use physical coordinates, actual length in meters or real time in minutes and seconds, to grab hold of, or name a set of data. Other times we just use integer indexes — 'l' (el) for space, and 'k' for time, which are just integer counts. If you are just doing data preparation work, then using the integer indexes is more convenient. If you are trying the relate the data to other data sets, or observations on the ground, then real coordinates are more useful. It seems the the use of multiple coordinate names for different tasks is an inevitable feature of environmental data sets — at least when humans are in the loop. The reference being used can be gleaned from context. Caveat emptor.

0.5 Example 2 — Loop-back; Agilent read

The second example will look at data from the Willamette River, and shows how to automatically identify the 'loop back' position along a fiber. Some fibers are built with two strands in the same housing, and at the far end of the fiber the light from the outgoing fiber can be connected to incoming fiber. This 'loop back' configuration is useful for calibration, and for getting a more accurate measurement, since we now have two temperature measurement at every point along the fiber.

We can recognize a 'loop back' configuration by noticing that the dts temperature snapshots are symmetric about the middle. We can look at individual snapshots, or the mean value of the snapshots as shown in Figure 4. As a first guess, we can estimate the loop back point as being half way down the fiber. The function

```
[loopback_l, delay] = find_loopback_position(dts_willamette);
```

used in the script below refines this guess by using a 'cross-correlation' function to estimate the offset needed from the middle of the fiber to 'line up' the out-going and 'in-coming' temperature snapshots. In this case, it turns out that our guess of halfway down the fiber is off by 3 steps (or, in this case, 3 * 1.5 meter = 5.5 meters). A section of the unaligned and aligned snapshots are shown in Figure 5.

This script also illustrates loading data from an Agilent DTS system with the function call

```
dts_willamette = read_agilent_files(willamette, 'willamette data from mike');
```

Unlike the SensorTran instrument, the Agilent instrument places each snapshot (or 'trace') stored in a in a separate file, using the naming convention agilent_data_file_name.tra. A good practice then is to place all the snapshot files from a single dts experiment into a single directory. In this case called the directory used is:

```
willamette  =  '..\dts_data\agilent_data_examples\willamette_data';
```

and is defined in the 'start_up_dts_tools' script in the 'dts_workspace' directory. The read_agilent_files() function then goes to this directory holding the Willamette river temperature data and collates all the individual snapshot (or trace) files into one temperature matrix and dts_experiment structure for Matlab.

The full script for this loop back example is:

```
echo on;
% Example Script 2: Loop back data

% Load Willamette Data file with Agilent Instrument
```

```
% The string variable willamette is set in the start up routine
% willamette = '..\dts_data\agilent_data_examples\willamette_data'

dts_willamette = read_agilent_files(willamette, 'willamette data from mike');

% Plot mean and standard deviation

[stdtemp, meantemp] = plot_stats(dts_willamette);

% Plot temperature

minimum_temperature = -2; maximum_temperature = 35;
plot_temperature(dts_willamette, minimum_temperature, maximum_temperature, 'distance (meters)', ...
    'time (julianday)', 'Temperature Plot', 'Willamette River Temperature Data')

% Find loop back position along fiber using cross-correlation

[loopback_l, delay] = find_loopback_position(dts_willamette);

% Add loop back information to 'dts_experiment' structure

dts_willamette.loopback = [loopback_l, delay];

disp('ENTER RETURN TO PLAY MOVIE');
pause;

% Play recomputed temperature snapshot movie of a fiber with loop back
% This system call assumes you have a AVI player installed in your
% operating system

system(['..\dts_data\agilent_data_examples\',...
    'example_of_processed_data_from_willamette_using_cygwin_and_matlab\',...
                                    'temperature_movie_of_willamette_dts_data.avi']);

echo off;
```

This script ends by playing a canned '.avi' movie of Willamette temperature data showing out-going and in-coming temperature snapshots on the same graph. The data covers about two weeks and the diurnal oscillations are easy to follow in the movie.

0.6 Example 3 — Just an average dts

It is useful to look at processing a temperature matrix using a running average over either space or time. When this operation is over time, Fred called it 'stacking'. Hence the origin of the name of the function 'stack_data()':

```
[dts_airshed_stacked_10] = stack_data(dts_airshed, 10);
```

in the script for example 3 below. Since position is represent by the columns in our temperature matrix, if we add up 'N' rows, or 'stack' them, and replace each row by its average value for the 'N/2' surrounding rows, then effectively we have a moving average filter over time. This filter will 'smooth' out the data

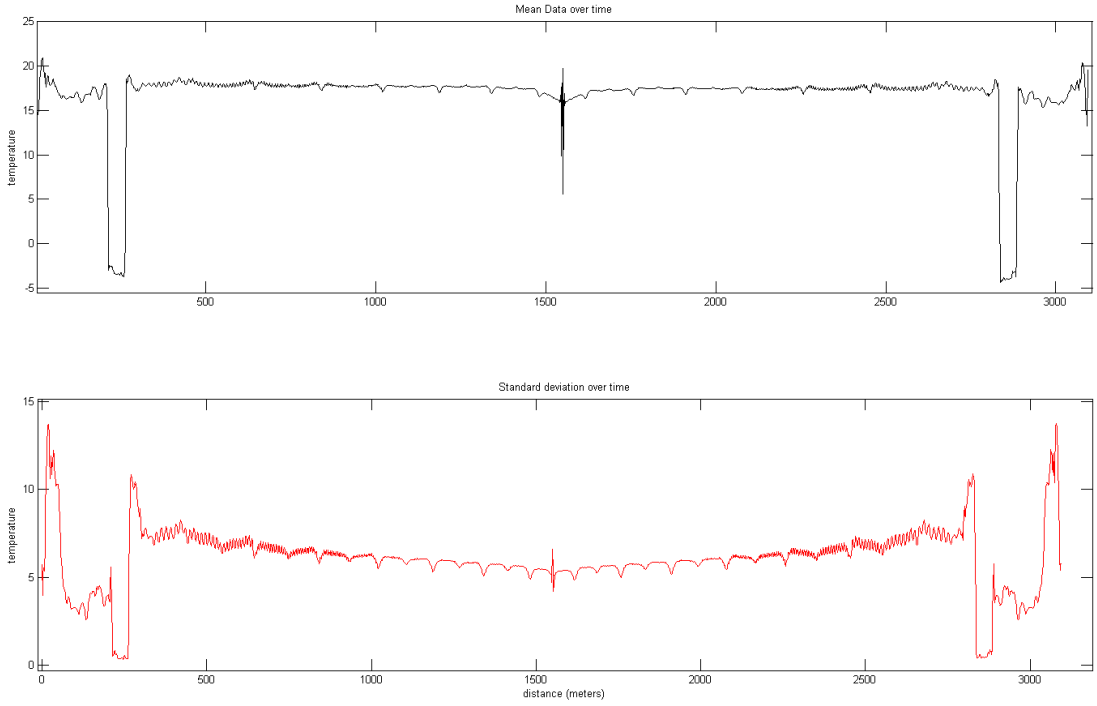


Figure 4: Mean value of snapshots for Willamette river data. Note the symmetry about the middle indicating fiber is arranged in a 'loop back' configuration.

and allow to get a better view of trends, at the expense of some details in time. Figure 6 shows the raw data on the left, and the 'stacked' data on the right for a window length of size 10, that is the value on the 'k' row is formed by the average value of computed from 5 rows above and 5 rows below. Since we are using 'future values,' this operation is a noncausal filter, strictly speaking. Many different filters are possible to handle lots of tasks beyond smoothing (for instance, change detection filters to identify ground water springs in streams). Here, this simple soothing filter allows us to reduce the noise on individual temperature snapshots and also it helps untangle the overall temperature trend, as shown again in Figure 6.

The script for example 3 is:

```
echo on;
% Example script 3 --- averaging data (stacking)

% Load data from H. J. Andrews Airshed study

dts_airshed = read_agilent_files(['..\dts_data\agilent_data_examples', ...
              '\h_j_andrews_airshed_data'], 'h. j. andrews airshed study');

% Extract time, length, and temperature data

k =    dts_airshed.k;
l =    dts_airshed.l;
temp = dts_airshed.tempdata;

% Plot a few temperature traces, zoom in on section from 400:500
```

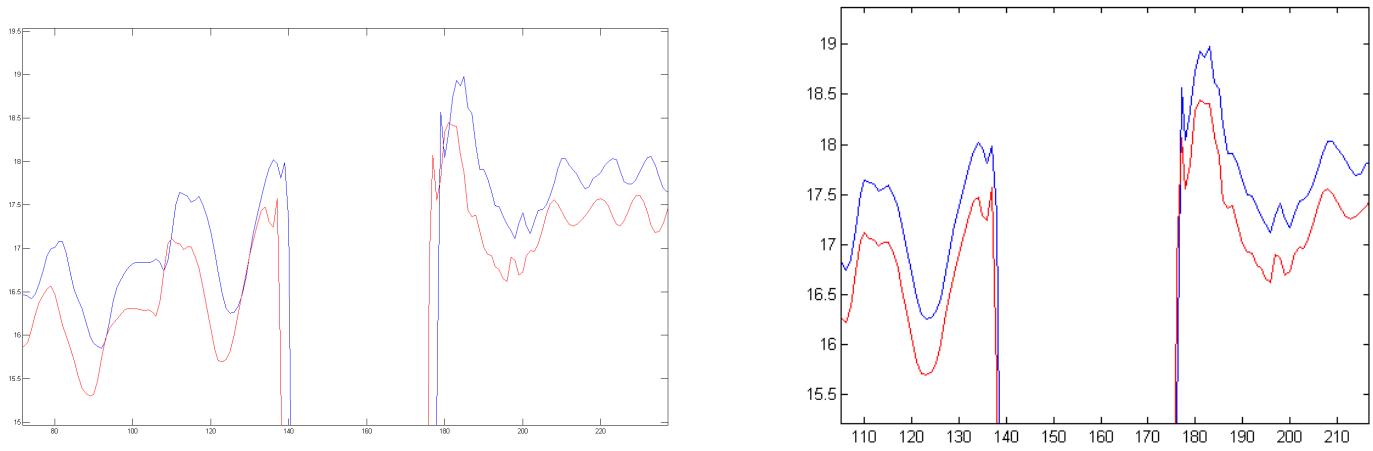


Figure 5: Left: Section of up (blue) and back (red) temperature snapshot with loop-back exactly in the middle. Right: With loop-back offset of 3 from the exact middle as found with the 'find_loopback_position()' function.

```
% axis([xmin xmax ymin ymax])

rows    =   100:104;
lmin = 400; lmax = 500;
columns =   lmin:lmax;
tempmin = 4.4; tempmax = 5.4;
figure; plot(l(columns), temp(rows,columns));
axis([lmin lmax tempmin tempmax])
title('Section of Airshed data, Raw');
xlabel('Length (integer count, l)'); ylabel('Temperature (deg C)');
legend('Five consecutive temperature snapshots', 'Location', 'NorthEast');

% Use 'stack' to (time) average every 10 snapshots

[dts_airshed_stacked_10] = stack_data(dts_airshed, 10);
figure;
plot(dts_airshed_stacked_10.l(columns), ...
          dts_airshed_stacked_10.tempdata(rows,columns));
axis([lmin lmax tempmin tempmax])
title('Section of Airshed Data with Averaging (Stack function)');
xlabel('Length (integer count, l)'); ylabel('Temperature (deg C)');
legend('Five consecutive temperature snapshots', 'Location', 'NorthEast');

echo off;
```

0.7 Example 4 — Resetting the temperature

Our last example script illustrates just some of the easy function fitting routines that are available in Matlab. Here we plot data from the Walla Walla river and notice that there are very low temperature regions (Figure 7) near the start and also at about 1000 meters (in integer count $l = 25$ or $l = 700$). These

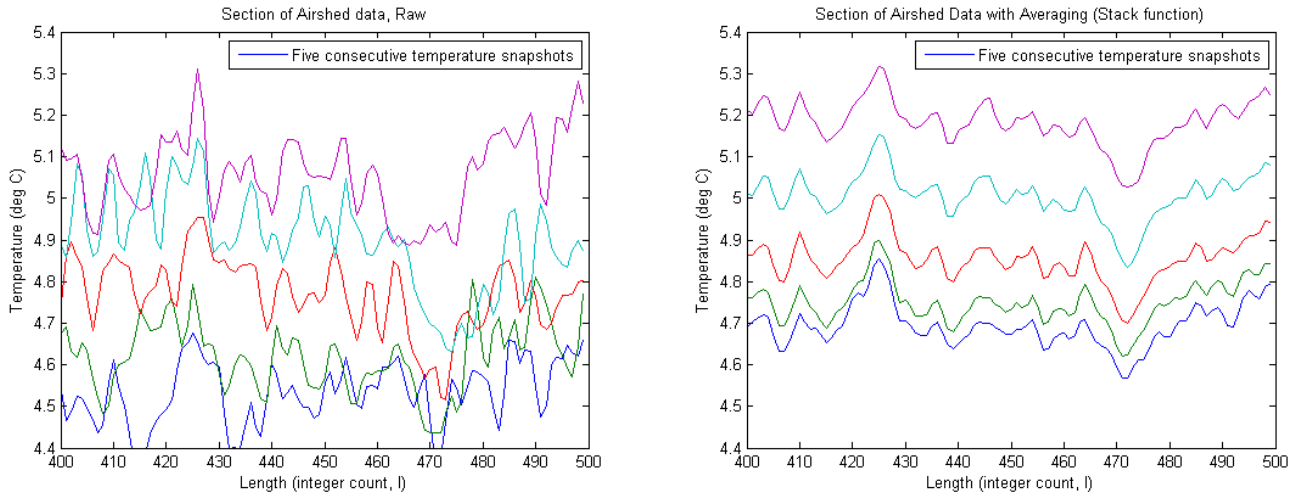


Figure 6: Left: Raw temperature snapshots. Right: Temperature snapshots with averaging ('stack_data()' function)

low temperature regions are from 'ice baths' which can be used as an easy way to check the temperature calibration, and reset it if necessary, for the fiber after it is placed in the field.

The script below takes a close look at the change in temperature at the first ice bath ($l = 25$), and then uses the built in fitting function in Matlab's Figure Window to estimate the temperature and its linear change. You can get to this function by the menus using the Menus Tools — Basic Fitting. The results from a linear fit are shown in Figure 8, and indicate that the fiber reads a temperature in the ice bath of about 3.9 degrees C. If the ice-bath is well packed and 'slushy,' then this this information can be used to reset the measurement temperature. A simple script for example 4 is:

```
% Example Script 4 --- Temperature at ice-baths

% Load in data

dts_walla_walla = read_agilent_files( ...
   '..\dts_data\agilent_data_examples\walla_walla_data', ...
                            'walla walla river data from mike')

% Inspect data with plots to look for ice-bath poitions

[stdtemp, meantemp] = plot_stats(dts_walla_walla);
mintemp = -5; maxtemp = 35;
plot_temperature(dts_walla_walla, mintemp, maxtemp, ...
      'Position', 'Time', 'Walla Walla River Temperature Data', 'walla_walla')

% Just Plot trace at ice bath l = 25

tempdata25  =  dts_walla_walla.tempdata(7:end,25);
figure; plot(tempdata25, '.');

% Will analyze trace with matlabs basic fitting functions in the figure
% window
```

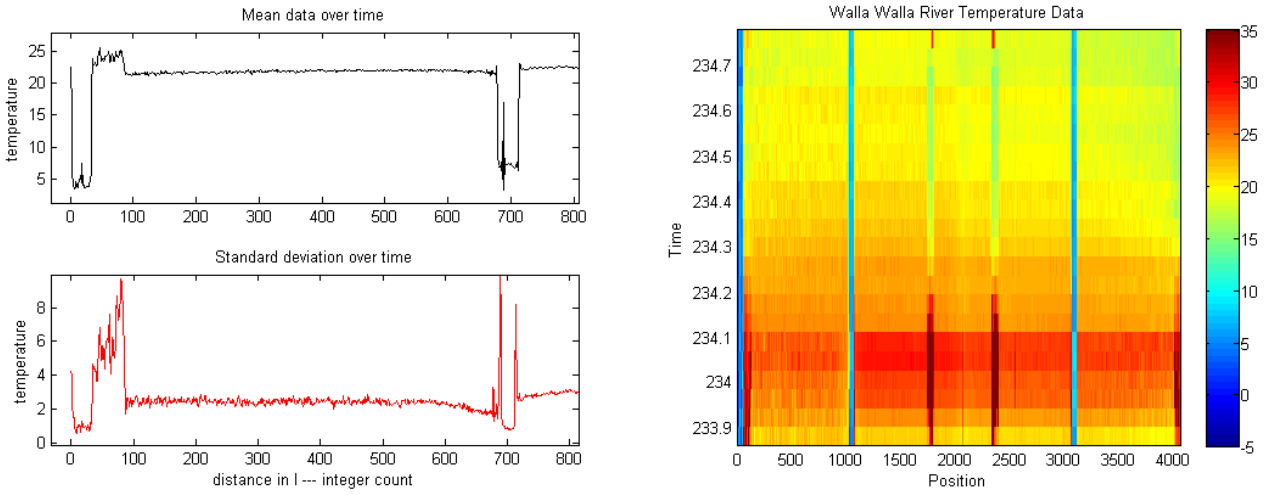


Figure 7: Identifying the location of the ice-baths in the Walla Walla river data set.

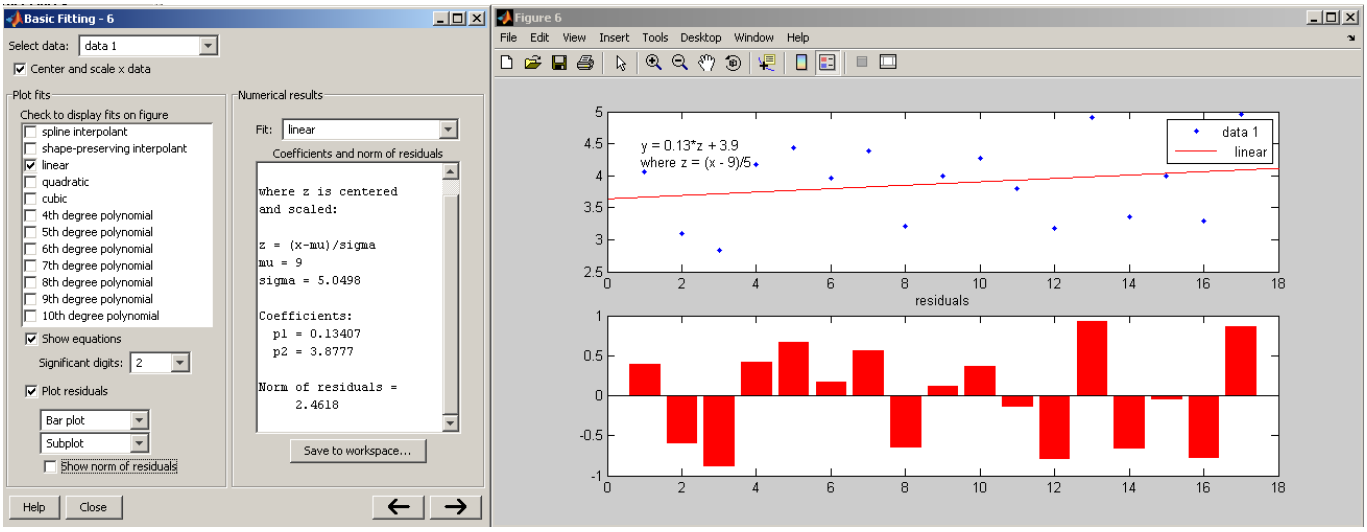


Figure 8: Analyzing the temperature in the ice-bath over time with Matlab's figure window fitting function tool (Tools $->$ Basic Fitting).

0.8 Much more to be done

Well, I hope these basic examples give you a good start on your dts data processing. There are quite a number of directions to go in now and things to do. Here some some suggestions for further improvements in no specific order:

- Improved functions for graphics,
- Add other manufactures importing functions: Sensernet, Sensa, Lios, ... ,
- Automated processes for detection of fiber defects, field based recalibration, change detector for spring inputs,
- Statistical analysis for detection of surface/subsurface water exchange,
- Tools for computing flow rates based on mass balance, heat exchange estimates,
- Automated loading to GCE Data Toolbox (LTER),
- Eventually a nice GUI to handle most of these functions,
- Editing commands, cut, join, sort,
- etc ... .

0.9 One more thing ...

At the end of a session, it is a good practice to clean up the 'dts_workspace,' leaving only 'start_up_dts_tools' file remaining.

Oh, one more thing, ... you can type 'help_dts' for a quick reminder of the commands.

```
% help_dts
% ------------------------------------------------------------------------------------------
% FUNCTIONS
% ------------------------------------------------------------------------------------------
% animate_temperature(dts_experiment, 'imagefileout', pausetime, xmin, xmax, tempmin, tempmax)
% str = cell2str(c)
% [loopback_l, delay] = find_loopback_position(dts_experiment)
% [pout,p] = grep(varargin)
% [stdtemp,meantemp] = plot_stats(dts_experiment)
% plot_temperature(dts_experiment, mintemp, maxtemp, 'xaxlabel', 'yaxlabel', ...
%                                                          'titlelabel', 'imagefileout')
% dts_experiment = read_agilent_files('agilent_data_dir', 'experiment notes')
% dts_experiment = read_sensortran_csv('fname', 'experiment notes')
% [dts_exper_time_avg] = stack_data(dts_experiment, time_window_integer_size)
% ------------------------------------------------------------------------------------------
%
% DATA STRUCTURE: dts_experiment.[notes ... std]
%               structure dts_experiment with the following fields:
%                 .notes       - first input string
%                 .datestring  - matlab date string
%                 .juliandate  - julian date (noon on jan 1 = 0.5)
%                 .k           - time index from 1 to length(juliandate)
%                 .channel     - channel used for dts collection
%                 .x           - external length in meters along fiber
%                 .l           - space integer index from 1 to length(x),
%                 .tempdata    - temperature data
%                 .name        - file name for input data
%                 .mean        - mean value over time at fixed location
%                 .std         - standard deviation over time at fixed loc
%
%   Current Version: Nick Tufillaro June 2008,
%   Eamil: nbt.osu@gmail.com
%   Web:   http://www.drchaos.net
%
% ------------------------------------------------------------------------------------------
```