# SeisLab for Matlab

MATLAB Software for Seismic Data Analysis — A Tutorial

October 23, 2005[1]

---

# Contents

# List of Figures

# Chapter 1

# INTRODUCTION

## 1.1   General

This manual describes MATLAB functions/macros for input, output, and manipulation/analysis of seismic data and well log curves, as well as functions that manipulate tables, probability distributions, and geologic models. The seismic-related functions described here are not intended for seismic data processing but rather for the more experimental analysis of small data sets. They should facilitate and speed up testing of new ideas and concepts. Likewise, the well log functions are intended for simple log manipulation steps like those, for example, required for their use with seismic data.

Data sets representing seismic data, log data, and tables are represented by MATLAB structures. At first glance, the description of all the possible fields of these structures may make them look complicated. However, a user may never need to explicitly create one of these fields himself. These fields are all created by certain functions as part of their normal output. It was a design decision to make the data in these structures visible **and** easily accessible. A user who understands the concept of Matlab structures can access any piece of information encapsulated in these structures.

A truly object-oriented design of a seismic data set would make all these details invisible — accessible only by means of specific tools. A user could not "mess up" an object, but — by the same token — he would loose a great deal of flexibility. Since it is rapid testing of new ideas and quick development of tools not available anywhere else that are the main purpose of these functions, easy access to every item of a data set is highly desirable.

This manual assumes that the user is reasonably familiar with MATLAB and, in particular, with MATLAB structures and cell arrays, which were introduced in MATLAB 5 and are used extensively. Hence, the functions described here will not work with earlier versions of MATLAB. Furthermore,

I have used the functions only under Windows. It is not inconceivable that there may be problems — in particular with file I/O and/or graphics — under UNIX/Linux.

The manual is not meant to be an exhaustive description of all the features, parameters, keywords, etc. used in all the functions, but rather intended to provide an overview over the functionality available and examples of the use of specific functions. The MATLAB help facility can be used to find out what arguments a particular function accepts. Wherever practical, default settings of parameters have been chosen so that the functions can be useful with a minimal number of input arguments.

Most functions can be grouped into one of five different categories:[1]

- seismic-related functions,

- log-related functions,

- functions that deal with probability distributions.

- functions that deal with tables (this last data type is still somewhat in flux; consequently, this manual is not yet very specific).

- functions that manipulate subsurface models

These categories are discussed below.

## 1.2   Initialization

In order to function properly SeisLab needs certain parameters. These parameters are set by function `presets` which, it turn, calls two other functions, `systemDefaults` and `userDefaults`. The latter sets parameters that a user is likely to customize (such as the directories where data files, such as SEG-Y files or LAS files with well data, are located). Function `systemDefaults`, on the other hand, sets those parameters that do not depend on a user's environment. In any case, every parameter set in `systemDefaults` can be changed in `userDefaults`. Since these parameters are used in many functions, a session using SeisLab commands must be preceded by

    presets

More information about `presets` can be found in Section 4.1 on page 53 ff.

## 1.3   On-line help

Matlab's standard help tools, `help` and `lookfor` are, of course, available for SeisLab functions as well. In addition, the following functions are intended to locate quickly functions that perform specific tasks for a particular type of data structure.

---

[1]Only functions from the first two categories are included in the public-domain version. Hence, occasional references to functions from these other categories should be ignored in the public-domain version of SeisLab.

- `l_tools` List functions that deal with well logs.

- `s_tools` List functions that deal with seismic data.

Without argument each of these functions displays a list of all the functions available for the specific type of data set, together with a one-line explanation of their purpose. In order to restrict the output a search term can be added. Thus

```
>> s_tools plot
s_2d_spliced_synthetic  Plot synthetic spliced into seismic line
s_3d_header_plot        Make contour plot of one header as function of ...
s_3d_spliced_synthetic  Plot synthetic spliced into inline and cross-line ...
s_compare                Plot one seismic data set on top of another for ...
s_cplot                  Plot seismic data in form of color-coded pixels ...
s_header_plot           Plot header values of a seismic data set
s_ispectrum              Interactively pick windows on seismic plot and ...
s_plot                   "Quick-look" plot of seismic data (color if more ...
s_spectrum               Plot amplitude and/or phase spectra of one or ...
s_wedge_model           Compute/plot synthetic from wedge model and  ...
s_wplot                  Plot seismic data in wiggle-trace format
```

displays only functions that are related to plotting of seismic data sets; ellipses (...) indicate truncated lines.

The statement `p_tools sample` shows available sample distributions.

## 1.4   Input arguments of functions

The majority of SeisLab functions has required and optional input arguments. Required arguments precede optional arguments and are "positional"; they are in a specific position (e.g. first, second, etc.) in the list of arguments. The order of subsequent optional input arguments, if any, is arbitrary. They consist of a keyword followed by one or more values — all encapsulated in a cell array. Keywords are strings. The following SeisLab function call, which plots the seismic data set `seismic` in wiggle-trace format, illustrates this.

```
s_wplot(seismic,{'trough_fill',[0.6 0.6 0.6]},{'annotation','cdp'})
```

The first input argument, `seismic` is required. The other two input arguments — in curly brackets — are optional. In particular, {`'trough_fill'`,[0.6 0.6 0.6]} specifies that the trough of the seismic wiggles, which is normally not filled with any color, should be gray (the three-component vector [0.6 0.6 0.6] is the RGB representation of a darker shade of gray). The other argument, {`'annotation'`,`'cdp'`}, specifies that the traces should be annotated by CDP number. . The default annotation is trace number.

4

## 1.5 Test data sets

For testing and demonstration purposes it is frequently desirable to have quick access to test data. Hence there are functions that create a variety of test data sets. It is a common feature of these functions that they have no input arguments and only one output argument: the data set. They generally follow the a naming convention of the form x_data and x_datann where x stands for the letters l, m, p, s, t and nn is a one-digit or two-digit number. Examples are:

### 1.5.1 Seismic data

- s_data creates a seismic data set consisting of 12 traces, 1000 ms long, of filtered random noise as shown in Figure 1.1.
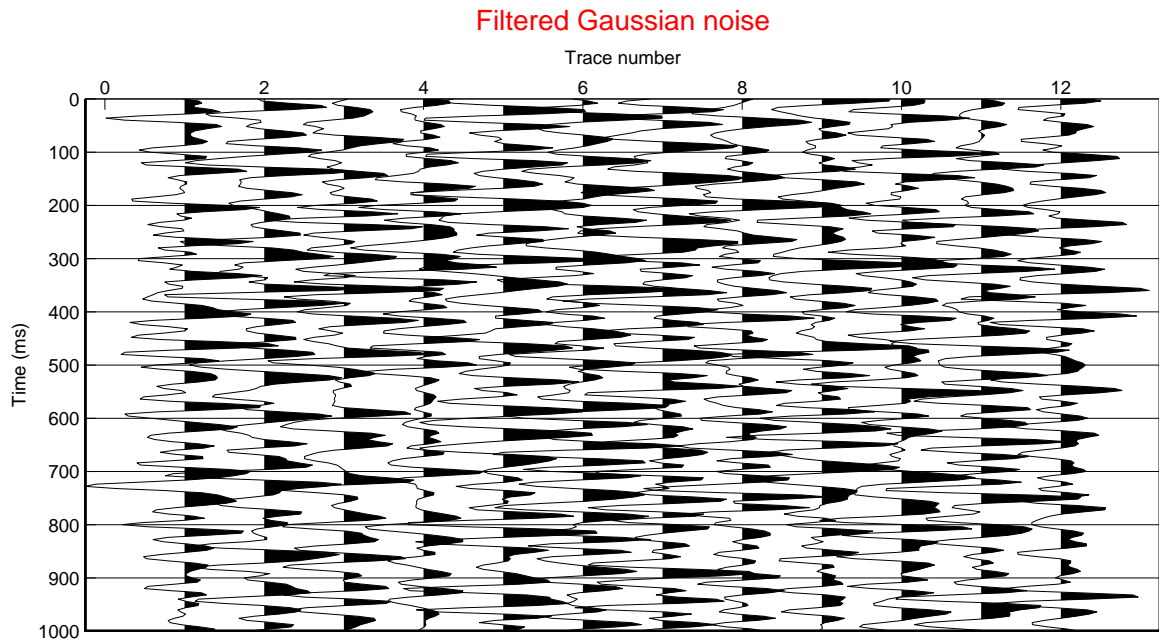


Figure 1.1: Filtered Gaussian noise; created by s_plot(s_data)

# Chapter 2

# SEISMIC DATA

## 2.1   A brief look at some functions for seismic data

The two statements

```
seismic = read_segy_file ;
s_wplot(seismic)
```

read an SEG-Y file and display the traces in a figure window in form of a wiggle-trace/variable area plot as shown in Figure 2.1. The function `read_segy_file` can take a number of arguments. One of them, of course, is the file name; but if it is not given, a file selection box allows interactive
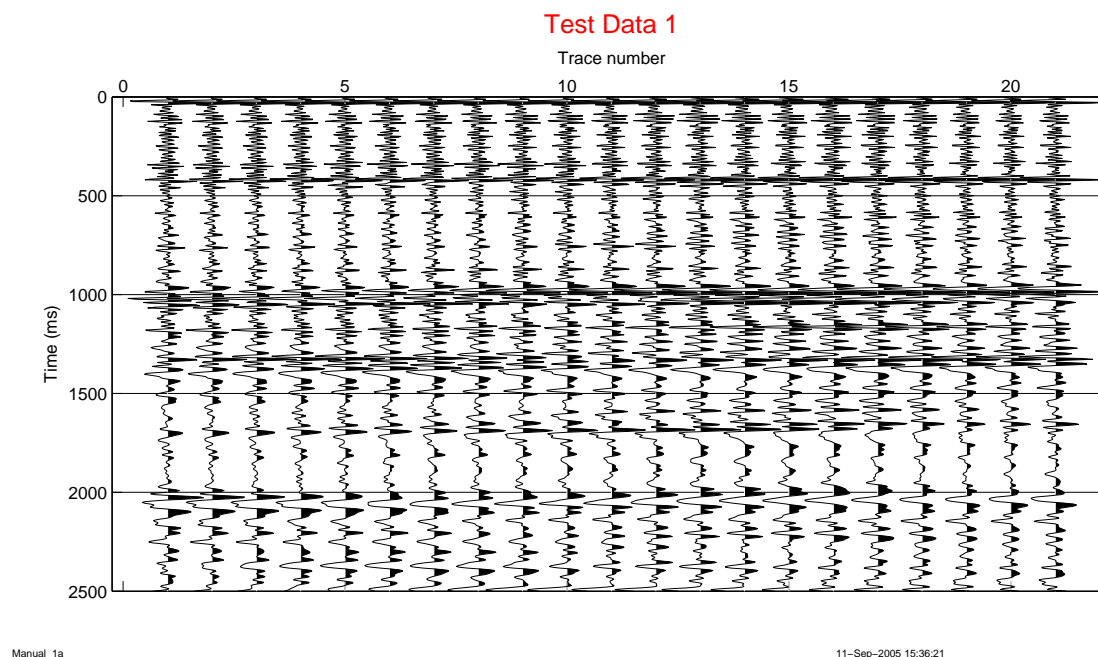


Figure 2.1: Wiggle-trace plot with default settings.

5

file selection.  Whenever a file is selected interactively the full path together with the name of
the file selected is printed to the screen. (The file selection function also remembers the directory
from which the file was copied and, on a subsequent request for an SEG-Y file, will open this
directory right away.)  If read_segy_file is part of a MATLAB script this file name can be
copied conveniently from the MATLAB window to the script so that the file will be read without
user intervention the next time the script is run.  File name and path name are also stored in
S4M.filename and S4M.pathname, i.e. in fields of the global structure S4M initialized in function
presets.  The file name without extension is also written to the field name of the seismic structure.

The data read from the SEG-Y file are stored in the structure seismic.  This structure is basically a
container which collects different pieces of data (seismic traces, headers, start time, sample interval,
etc.)  under one name.  The structure seismic is then input to the plot function s_wplot (most
seismic-related functions start with "s_", and the "w" in s_wplot stands for wiggle — s_cplot makes
seismic plots with amplitudes represented by color).

The function s_wplot has one required argument, the name of the seismic structure.  All other
arguments are optional.  Figure 2.1 shows the plot obtained with these defaults.  The traces are
numbered sequentially.  Because no title was specified explicitly the string in the field name of the
seismic structure (in this example Test Data 2) is used as a default title.

Using some of the optional arguments one can tailor the two statements to specific needs.  For
example,

```
seismic_data = read_segy_file('C:\Data\Test Data 2.sgy',{'times',500,1500}, ...
                              {'traces','cdp >= 1650 & cdp <=1660'});
s_wplot(seismic_data,{'direction','r2l'},{'annotation','cdp'})
```

will read from file C:\Data\test.sgy all traces with CDP numbers from 1650 to 1660 in the time
range from 500 to 1500 ms.  If the function read_segy_file is used with any parameters, the
filename must be the first one; but it can be an empty string ' ', and in this case the file can be
selected interactively.  The other two input parameters are cell arrays.  The first element of each
cell array is a keyword which tells the program how to interpret the subsequent elements.  The
keyword 'times' signals that the next two numbers are start and end time of the trace segment
to be retrieved.  The other keyword 'traces' indicates that the second element of the cell array,
'cdp >= 1650 & cdp <=1660', relates to the selection of a subset of the traces.  This subset
can be defined in various ways; here this is done via a logical condition for header values CDP. By
default, read_segy_file reads (and stores) all trace header values specified as essential in the
SEG-Y standard (this includes CDP), but then discards all those that turn out to be identically
zero.

Of course, read_segy_file can read any user-specified trace header.  Trace headers explicitly
requested are not discarded even if they turn out to be zero for every trace.

In this example, the output of read_segy_file is stored in the seismic structure seismic_data
which is then input to s_wplot.  Here s_wplot has two optional arguments — cell arrays whose
first elements are keywords.  The keyword direction indicates the plot direction.  The default is
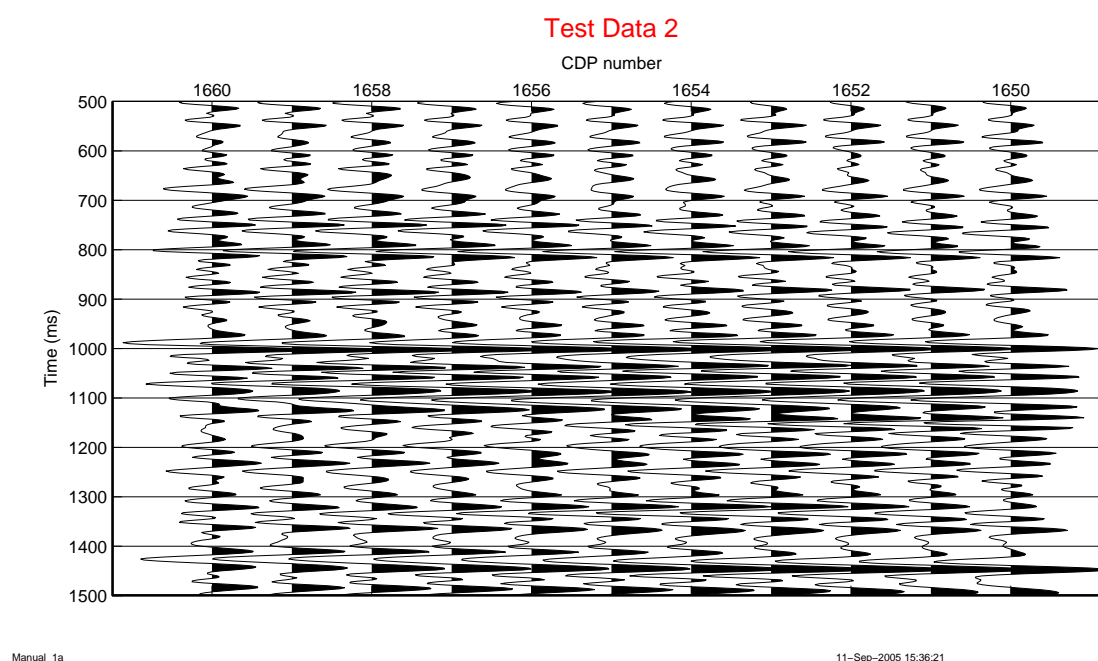
Figure 2.2: Wiggle-trace plot with traces labeled by CDP, increasing from right to left.

left-to-right, but here it is right-to-left. The other keyword, `annotation`, specifies which header to use to annotate traces. The plot obtained with these two commands is shown in Figure 2.2. Since no plot title has been defined the name of the input data set, `seismic_data`, is used.

The code fragments that created Figures 2.1 and 2.2 come from one and the same MATLAB script, `Manual_1`. One of the first statements in this script is the function `presets` (see page 53 ff.). which sets a number of global variables, among them a plot label for the lower left corner of plots and the date and the time the script was started. It is this date/time combination that is displayed in the lower right corner of the plots. Consequently, all plots created by the script in a particular run bear the same time stamp.

Another code fragment that illustrates the use of SeisLab functions is

```
>> filtered_seismic = s_filter(seismic_data,{'ormsby',5,10,20,30});
>> s_compare(seismic_data,filtered_seismic) ;
```

where the seismic data set `seismic_data` of the previous example is filtered with a trapezoidal filter with corner frequencies 5, 10, 20, 30 Hz and then compared with the unfiltered data. Unlike in the previous plot, where the default color is black, `s_compare` uses color by default; a gray-scale reproduction of Figure 2.3 does not do justice to this kind of comparison.

Presently there are some 130 utility-type functions to operate on seismic data sets.[1] The best way to find out what is available is to run function

---

[1]Only a subset of the available seismic functions is included in the public-domain version.

Test Data 2 vs. filtered Test Data 2
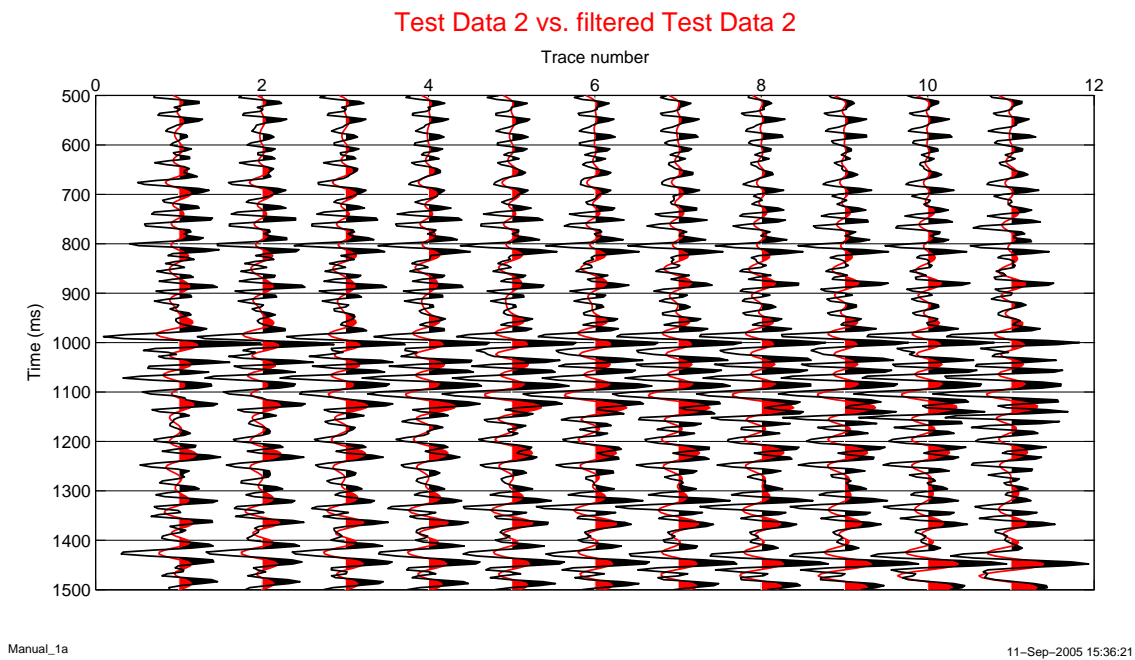


Figure 2.3: Comparison of unfiltered (black) and filtered (red) seismic traces.

        s_tools

which provides a one-line description of all functions which deal with seismic data sets. To make
the list more specific a keyword may be added. For example,

        s_tools seg

lists only those functions that deal with SEG-Y data files (the search is not case sensitive):

        read_segy_file     Read disk file in SEG-Y format
        show_segy_header   Output/Display EBCDIC header of SEG-Y file as ASCII
        write_segy_file    Write disk file in SEG-Y format

## 2.2 Headers

Trace headers (headers, for short) store trace-specific information such as offset, trace location, CDP number, in-line number, cross-line number, etc. and play a major role in seismic data processing. As mentioned above, when an SEG-Y file is read a number of headers are read by default; other headers may be read as requested by input arguments of `read_segy_file`. Additional headers will be added by certain functions: `s_align`, for example, which aligns (flattens) an event on a seismic section stores the shifts applied to each trace in a header value. This way the shifts can be undone if necessary.

Headers are fields in a seismic structure and are discussed again in the section on seismic structures. However, as long as established functions are used for their manipulation nothing needs to be known about the way they are stored in the seismic structure. Assume that `s3d` is a 3-d seismic data set with headers `cdp_x` and `cdp_y` representing CDP coordinates (while the only restriction on header names is that they must comply with the rules for MATLAB variables it appears to be advantageous to use the same names ProMAX uses). Then

   `s_header_plot(s3d,{'cdp_x','cdp_y'},{'colors','ro'})`

creates the base map shown in Figure 2.4 by plotting red circles (`'ro'`) at points defined by CDP_X and CDP_Y pairs.
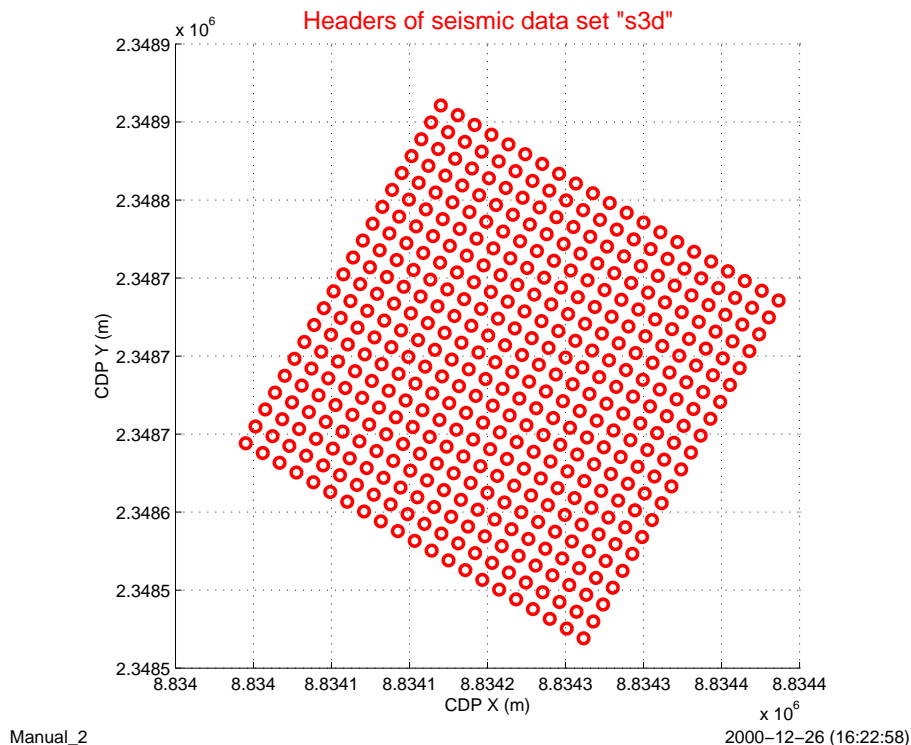


Figure 2.4: Plot of CDP locations.

The same base map can be created by means of the standard Matlab plot commands

```
>> figure
>> plot(s_gh(s3d,'cdp_x'),s_gh(s3d,'cdp_y')','ro','LineWidth',1.5)
>> xlabel([s_gd(s3d,'cdp_x'),'(',s_gu(s3d,'cdp_x'),')'])
>> ylabel([s_gd(s3d,'cdp_y'),'(',s_gu(s3d,'cdp_y'),')'])
>> title('Headers of seismic data set "s3d"','FontSize',14,'Color','r')
>> grid on
>> time_stamp
```

which is more tedious. It employs the functions

```
>> header_values = s_gh(seismic,header_mnemonic)
>> header_units = s_gu(seismic,header_mnemonic)
>> header_description = s_gd(seismic,header_mnemonic)
```

to retrieve, from seismic data set `seismic`, a row vector of header values and a text strings with units of measurement and a description, respectively, of the header with mnemonic `header_mnemonic` (`header_mnemonic` is a character string).

Obviously, the header values could be extracted directly from the matrix `seismic.headers` and units of measurement and header description from the cell array `seismic.header_info`. However, using the above functions insulates a user from the need to know in what particular row of `seismic.headers` and `seismic.header_info` the requested information is stored. Also, if the global variable `S4M.case_sensitive` is set to 0 (false) — see start-up function `presets` (page 53) — it does not matter if the header mnemonic specified is in lower case, upper case, capitalized, or consists of any mixture of lower-case and upper-case characters. Incidentally, function `s_gh` has a second, optional output argument; it is the appropriate row of cell matrix `seismic.header_info`, a three-element cell vector with the header mnemonic, the header's units of measurement, and the header description.

## 2.3   Description of seismic structures

The seismic-related functions assume that a seismic data set is represented by a structure which — in addition to the actual seismic traces — contains necessary ancillary information in form of required parameters, optional parameters, and headers. Parameters are pieces of information such as start time, sample interval, etc. which pertain to all traces of the seismic structure. Headers, on the other hand, are trace specific. They can vary from one trace to the next. Hence, each header has a value for each trace. Seismic data proper are stored in a matrix whose columns represent individual traces. In general, each row in this matrix represents a specific time. However, it is also possible that each row represents a frequency or a depth. Hence, the term "time" is used in a somewhat loose sense; it could also mean some other dimension. For this reason seismic structures have a field called `units` which defines the units of measurements. The default is `ms`.

The simplest seismic structure can be created by the MATLAB statement

```
>> seismic = s_convert(matrix,start_time,sample_interval)
```

where `matrix` denotes a matrix of seismic trace values, `start_time` is the time of the first sample and `sample_interval` the sample interval. The resulting structure `seismic` has the eight fields required of a seismic structure. They are described below, and the description uses the following variables

 `nsamp`    number of samples per trace
 `ntr`      number of traces per data set
 `nh`       number of headers in the data set

In this section the name `seismic` is used to refer to a seismic structure. Thus, `seismic.traces` denotes the field `traces` of a seismic structure.

- `type` type of data set. For seismic data it is set to the string `'seismic'`. This field is intended to allow interactive programs to identify quickly the type of data set represented by a particular structure; i.e. distinguish between seismic data sets, well logs, probability distributions, etc.

- `tag` an attribute that describes more clearly the kind of seismic data sets. Possible values are: `'wavelet'`, `'impedance'`, `'reflectivity'`, and the catch-all `'unspecified'`.

- `name` Name of the data set; by default, for data sets read from SEG-Y files, it is the file name.

- `traces` A matrix of numeric values with dimension `nsamp` by `ntr`. Each column of the structure field `traces` represents a seismic trace.

- `first` Time (or frequency, or depth) associated with the first row of `traces`.

- `last` Time (or frequency, or depth) associated with the last row of `traces`.

- `step` Sample interval; obviously (`last` - `first`)/`step` +1 = `nsamp`.

- `units` Units of measurements for time (or frequency, or depth); examples are 'ms', 'Hz', 'm', 'ft'.

In addition to the required fields seismic functions create and use a number of additional structure fields. Most frequently used are:

- `headers` A matrix of numeric header values with dimension `nh` by `ntr`. Each row of this matrix represents the value of a particular header for each trace. Examples of such headers are `cdp, offset`, etc.

- `header_info` A cell array of strings with dimension `nh` by 3. Each row of this cell array lists a header in terms of its mnemonic (first column), its units of measurement (second column), and its description. Many headers, such as `cdp`, have no units of measurement; they have 'n/a' in column 2.

- `history` A four-column cell array with a "processing history", i.e a list of the MATLAB functions used to create the seismic structure. Entries into this field are made automatically by most processes unless this option is turned off (see description of the function `presets`).

- `null` Null value or no-data value. This value, in general `NaN`, is set by a process if some of its output data in `traces` are not valid. This may happen if noise spikes have been removed, if data sets with differing start times or end times are concatenated, etc. While it is common to zero or to clip bad data (such as noise burst, data with NMO stretch, etc.) it is frequently more prudent to have a special value that identifies them as invalid. If the seismic structure has no `null` field or `if seismic.null == 0` then either all data are valid or invalid data have simply been zeroed. When data are written to an SEG-Y file (see `write_segy_file`) any `NaN`s are replaced by zeros.

- `header_null` Null value or no-data value. This value, in general `NaN`, is set by a process if some of its output data in `headers` are not valid. This may happen if data sets with differing headers are concatenated (e.g. synthetics spliced into real data), etc. When headers are written to an SEG-Y file any `NaN`s are be replaced by zeros.

- `time` One time value for each row of data matrix `seismic.trace` allowing non-uniformly spaced data. In this case `step` is set to zero

Furthermore there can be an arbitrary number of fields representing parameters

Below is an example of the simplest seismic structure:

```
wavelet
     type   :  'seismic'                 Type of structure
      tag   :  'wavelet'                 Tag; more specific description of dataset
     name   :  'Ormsby (zero-phase)'     Name
    first   :  -40                       Time of first sample
     step   :  4                         Sample interval in msec
     last   :  40                        Time of last sample in msec
    units   :  'ms'                      Units of measurement of time axis
   traces   :  [21x1 double]             One-column array with 21 entries
```

It represents an 80-ms wavelet with 4 ms sample interval and centered at time zero. While it may be advantageous to have more information attached to the structure (for example the CDP or in-line and cross-line number for which the wavelet was determined, or the processing history) this is not required. However, even if no header is explicitly specified there is an implied pseudo-header `trace_no` that can be used like any other header. It is a sequential number of each trace and is called pseudo-header since it is not attached to a specific trace; thus if one removes a trace from a data set the pseudo-header `trace_no` of all traces following the one that was removed will be decreased by one. "Real" headers of a trace, on the other hand, are not affected if one or more traces are added or removed from a data set.

The following example shows a more elaborate structure output by the function `read_segy_file` discussed below which reads an SEG-Y file.

```
        seismic
           type  :  'seismic'              Type of structure
            tag  :  'unspecified'          Tag; more specific description of dataset
           name  :  'Test Data 3'          Name
    line_number  :  1                      Line number
traces_per_record :  48                    Traces per record
          first  :  0                      Time of first sample
           step  :  2                      Sample interval in msec
           last  :  2000                   Time of last sample in msec
          units  :  'ms'                   Units of measurement for the time axis
    header_info  :  [6x3 char]             Descriptions of the header mnemonics
        headers  :  [6x480 double]         Header values
         traces  :  [1001x480 double]      Array (480 traces with 1001 samples, each)
        history  :  1x4 cell
```

The eight fields familiar from the first example indicate that the data set consists of 480 traces with 1001 samples each and a sample interval of 2 ms. Furthermore, there are the scalar fields `line_number`, `traces_per_record` and `units` which were take directly from the binary reel header of the SEG-Y file. Of generally more interest are the trace headers. This seismic data set has six trace headers. Information about these trace headers is stored in the field `header_info`. They resemble the way ProMAX lists headers. The six header mnemonics as well as the associated units of measurement and header descriptions are shown below.

| Header mnemonic | Units | Header description |
|---|---|---|
| 'ds_seqno' | 'n/a' | 'Trace sequence number within line' |
| 'ffid' | 'n/a' | 'Original Field record number' |
| 'o_trace_no' | 'n/a' | 'Trace sequence number within original field record' |
| 'cdp' | 'n/a' | 'CDP ensemble number' |
| 'seq_cdp' | 'n/a' | 'Trace sequence number within CDP ensemble' |
| 'iline_no' | 'n/a' | 'In-line number |

None of these headers is associated with units of measurement and so all entries in the second column are `'n/a'`. This would have been different if offsets or coordinates had been among the headers. The most convenient and informative way of looking at the headers of seismic data structure `seismic` is to execute the command

> *s_header(seismic).*

By default, `read_segy_file` initially reads 22 pre-set trace headers but then discards all those whose values are zero for every trace. The first five headers listed above represent the remaining non-zero preset trace headers. The sixth header (`iline_no`) is a user-requested header read from a user-defined byte location in the binary trace header of the SEG-Y file.

There are some mild restrictions on header names (they must satisfy all requirements placed on MATLAB variables). Furthermore, it is highly recommended that the following header mnemonics be used where applicable since they are expected to be present in certain functions. They correspond to those used in ProMAX and, hence, ProMAX users should not find them difficult to remember.

| Header mnemonics | Header descriptions |
|---|---|
| cdp | CDP ensemble number |
| offset | Source-receiver distance |
| iline_no | In-line number |
| xline_no | Cross-line number |
| cdp_x | X-coordinate of CDP |
| cdp_y | Y-coordinate of CDP |
| sou_x | X coordinate of source |
| sou_y | Y coordinate of source |
| sou_elev | Surface elevation at source |
| rec_x | X coordinate of receiver |
| rec_y | Y coordinate of receiver |
| rec_elev | Receiver elevation |
| source | Energy source point number |
| sou_depth | Source depth below surface |
| rec_h2od | Water depth at receiver |
| sou_h2od | Water depth at source |
| ffid | Field file ID number |

Table 2.1: Partial list of headers read from SEG-Y files; the complete list can be obtained with the `help read_segy_file` command.

The last field in the above structure is the `history field`. This is an optional field generally created by all functions that create seismic structures (e.g. `read_segy_file`). Other functions append information to this field, if it exists.

## 2.4   Operator overloading

Operator overloading refers to a facility in Matlab where operators such as "+", "-" or built-in functions such as `abs, sqrt,`  are given a special meaning in situations where they had none before. An example is multiplication by a number of a Matlab structure such as a seismic data set. Thus `3*seismic` (here and in the following `seismic` is a seismic data set) would normally result in an error message. However, in SeisLab the multiplication operator has been overloaded to make this a meaningful statement for seismic data sets. In fact, `3*seismic` means that the samples of the seismic traces, i.e. the elements of the matrix `seismic.traces`, are multiplied by 3. In general, the operator is applied to one field of the seismic data set — the field `traces`. This is a convenience feature meant to simplify interactive operations. Since it involves function calls it is somewhat slower than direct operations on the field `traces`. Overloaded operators are:

### 2.4.1   Unary plus (+) and minus (-)

+ seismic   legal, but the same as seismic
– seismic   means seismic.traces → – seismic.traces

Thus

```
wavelet=s_create_wavelet('step',1);
s_compare(abs(wavelet),-wavelet)
```

is legal and results in the plot shown in Figure 2.5. The black wavelet is the absolute value of the original one (the abs operator is introduced below), the red wavelet had the sign flipped.
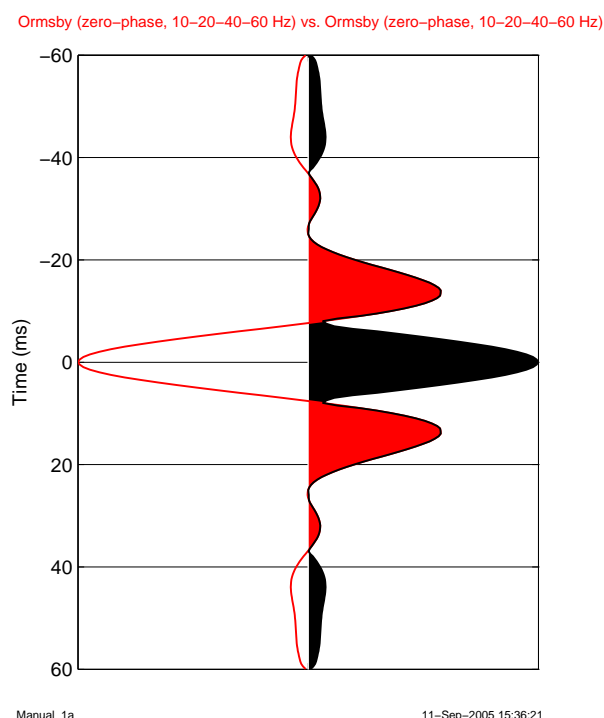


Figure 2.5: Illustration of the effect of the abs operator (black) and of unary minus (red).

## 2.4.2   Addition and subtraction

seismic ± a   means seismic.traces → seismic.traces ± a
a ± seismic   means seismic.traces → a ± seismic.traces

The variable a can be a constant or a matrix with as many rows as there are samples and as many columns as there are traces in the seismic data. But it can also be a row vector with as many elements as there are traces or a column vector with as many elements as there are seismic samples. In the former case a is element-by-element added to each row (time-slice), in the latter case it is added element-by-element to each trace.

### 2.4.3   Multiplication

`seismic * a`   means `seismic.traces` → `seismic.traces * a`
`seismic.traces` → `a * seismic.traces`, i.e. the same result as `seismic * a`

The variable `a` should be a scalar (1×1 matrix).

### 2.4.4   Element-by-elements multiplication

`seismic .* a`   means `seismic.traces` → `seismic.traces .* a`
`a .* seismic`   means `seismic.traces` → `a .* seismic.traces`, i.e. the same result as `seismic .* a`

The variable `a` can be a constant or a matrix with as many rows as there are samples and as many columns as there are traces in the seismic data. But it can also be a row vector with as many elements as there are traces or a column vector with as many elements as there are seismic samples. In the former case each row (time-slice) is element-by-element multiplied by `a`, in the latter each trace is element-by-element multiplied by `a`.

### 2.4.5   Division

`seismic/a`   means `seismic.traces` → `seismic.traces/a`


The variable `a` should be a scalar (1×1 matrix).

### 2.4.6   Element-by-element division

`seismic ./ a`   means `seismic.traces` → `seismic.traces ./ a`


The variable `a` can be a constant or a matrix with as many rows as there are samples and as many columns as there are traces in the seismic data. But it can also be a row vector with as many elements as there are traces or a column vector with as many elements as there are seismic samples. In the former case each row (time-slice) is element-by-element divided by `a`, in the latter each trace is element-by-element divided by `a`.

Thus, for example,

```
seismic_scaled=seismic./max(seismic.traces);
```

normalizes traces so that the maximum of each trace is 1.

`a ./ seismic`   means `seismic.traces` → `a ./ seismic.traces`


The variable `a` can be a constant or a matrix with as many rows as there are samples and as many columns as there are traces in the seismic data. But it can also be a row vector with as

many elements as there are traces or a column vector with as many elements as there are seismic samples. In the former case `a` is element-by-element divided by each row (time-slice), in the latter case `a` is element-by-element divided by each trace.

### 2.4.7   Element-by-element exponentiation

`seismicˆa`   means `seismic.traces` → `seismic.traces.ˆa`

### 2.4.8   Absolute value

`abs(seismic)`   means `seismic.traces` → `abs(seismic.traces)`

An example of the use of the `abs` operation is shown in Figure 2.5, above.

### 2.4.9   Sign

`sign(seismic)`   means `seismic.traces` → `sign(seismic.traces)`

This means that positive samples are replaced by 1 and negative samples by -1.

## 2.5   Description of selected functions for seismic data analysis

At the time of this writing there were about 130 seismic-related functions.[2] Only a few of them are discussed here. (a full list can be obtained by means of the command `s_tools`). They are listed in alphabetical order. In general, only examples that characterize their use are provided. The standard Matlab `help` command lists all parameters of a function.

**read_segy_file**

**Purpose:** This function reads a disk file written in SEG-Y format and outputs a seismic structure. Input parameters allow selection of the filename, the floating point data format, special trace headers, and a selection of the traced read as well as the time range. The simplest form is

```
seismic = read_segy_file
```

In this case — when no file name is specified — a file selection window pops up allowing interactive file selection. The seismic data read are stored in the structure `seismic`. The full path and filename of the file selected is printed out so that it can be pasted into a command of the form

```
seismic = read_segy_file(file_name)
```

---

[2]Only a subset of the functions are included in the public-domain release.

which does not invoke interactive file selection (unless `filename` is an invalid file name). The file name without the extension is also saved in field `name` of the seismic structure.

A more complicated function call is

```
seismic  =  read_segy_file('', ...
        {'headers',{'ILINE_NO',181,4,'n/a','In-line number'}, ...
        {'offset',37,4,'m','Source-receiver distance'}}, ...
        {'traces','cdp == 100 & offset > 100' })
```

In this case the file is interactively selected (the filename is empty). Furthermore, headers `iline_no` and `offset` are read from the 4 bytes beginning at byte locations 181 and 37, respectively, and stored in the trace header. Finally, not all traces but only those with `cdp` number 100 and an `offset` > 100 are read.

Another example is

```
seismic  =  read_segy_file(filename,{'traces','mod(cdp,2) == 0'})
```

in which only traces with even CDP numbers are read.

It is also possible to restrict the range of times read. If a conversion from IBM floating point format to IEEE format is required, restricting the number of traces read and the times may considerably reduce the time required to read the data from a file. Function `read_segy_file` supports only IBM floating point format and IEEE big-endian format.. By default, the binary header bytes 25-26 are used to determine the format. A user may, however, override this format selection.

## s_align

**Purpose:** This function flattens an event by aligning all traces to a reference trace.

## s_append

**Purpose:** This function appends one seismic data structures to another to form one single seismic data structure. The simplest form is `seisout = s_append(seisin1,seisin2);` where `seisout` contains the traces of `seisin1` followed by the traces of `seisin2`. Times of the first and the last sample of `seisout` are defined by:
`seisout.first  =  min(seisin1.first,seisin2.first);`
`seisout.last  =  max(seisin1.first,seisin2.first).`

Likewise the headers of `seisout` are the union of the headers of `seisin1` and `seisin2`. Times in one input data set that are not present in the other are set to the trace null value in the output data set. If the trace null value is `NaN` the field `seisout.null` is set to `NaN`. Similarly, headers absent in one data set but present in the other are set to the header null value which may differ from the trace null value. If the header null value is `NaN` the field `seisout.header_null` is set to `NaN`.

Other ways of handling the combination of the two data sets can be specified by keyword-controlled input arguments.

### s_attribute

**Purpose:** For each trace of the input data set this function computes attributes of the seismic traces and stores them in header(s). The attributes computed are: maximum of trace; maximum of absolute values of trace; minimum of minimum of absolute values of trace; mean of trace; mean of absolute values of trace; median of trace; median of absolute values of trace; RMS amplitude of trace. The simplest form is

```
s_attribute(seismic)
```

which computes and prints to the screen a summary of the attributes. If an output dataset is provided the attributes are added to the trace headers or replace already existing headers with the same name.

```
seismic = s_attribute(seismic)
```

which adds all above listed attributes to the header

It ia also possible to specify that only a subset of the attributes be computed.

```
seismic=s_header(s_attribute(seismic,'add_ne','max','rms'));
```

In this example `s_attribute` computes the maximum amplitude and the RMS amplitude of each trace, stores this information in headers named `max` and `rms`, respectively, and outputs the new dataset.

### s_check

**Purpose:** This functions checks a seismic data structure for formal errors such as inconsistencies, missing required fields, etc. It takes only one argument, the data set to be checked. An example is

```
s_check(seismic)
```

which checks for errors of the structure `seismic`; if indeed errors are found, it will print messages explaining them. Otherwise, the message `No formal errors found in ''seismic''` will be printed.

It is expected that every seismic-related function listed in this manual will pass this consistency test; however, users may modify structures outside of these functions and, if these changes are more severe, checking if they violate any formal requirements may be appropriate.

**s_compare**

**Purpose:** This function plots two seismic data sets, one on top of the other to allow comparison. An example is shown in Figure 2.5. The function has numerous parameters to control all aspects of the plotting of the two functions (which do not need to have the same start or end times or sample interval).

**s_convert**

**Purpose:** This function converts a matrix into a minimal seismic data structure. An example is

```
seismic=s_convert(randn(201,20),0,4)
```

which converts a 200 by 20 matrix of random, normally distributed noise into a 20-trace seismic data set with start time 0 and 4 ms sample interval. The result is the following seismic structure

```
seismic =
     type:  'seismic'
      tag:  'unspecified'
     name:  ''
    first:  0
     step:  4
     last:  800
   traces:  [200x20 double]
    units:  'ms'
  history:  {'29-Aug-2001 21:15:41 ' [2036] 'S_CONVERT' []}
```

Since no content has been specified for the `history` field it is set by default. The same is true for the fields `units`, `tag`, and `name`.

**s_convolve**

**Purpose:** `seisout = s_convolve(seisin1,seisin2)` convolves the two seismic data sets and outputs the result. There are optional input arguments that allow specification of various aspects of the operation. The two seismic data sets must satisfy the following restrictions: either both data sets have the same number of traces (in this case corresponding traces of the two data sets are convolved) or at least one of the data sets must consist of one trace only (in this case it is convolved with all the traces of the other data set. As a result the number of traces in `seisout` is equal to the number of traces of the larger input data set (in terms of traces). Consequently, the headers of this larger data set are passed on to the output data set. If both data sets have the same number of traces the headers of the first data sets are copied to the output data set. This behavior can be changed by use of the keyword `header`

- {'header',numerical_parameter} Select input data set to supply the headers for `seisout`. Possible values are 1 and 2. Default is 1.

If one of the data sets consists of one trace and the other has more than one trace the headers of the larger data sets are copied to the output data set.

### s_correlate

**Purpose:** `seisout = s_correlate(seisin1,seisin2)` performs a crosscorrelation of the traces of the two seismic data sets and outputs the result. Arguments `args` are optional and allow specification of various aspects of the operation. The default is that each trace of the second data set is correlated with each trace of the first data set. The output data set has two headers (default: 'seis1' and 'seis2') which, for each output trace indicate which input trace of the first and of the second data set was used for its calculation.

Alternatively, each trace of the second data set can be correlated with the corresponding trace of the first data set. In this latter case the two seismic data sets must have the same number of traces. As a result the number of traces in `seisout` is equal to the number of traces of the input data sets (in terms of traces). The headers of the first input data set are passed on to the output data set.

### s_cplot

**Purpose:** This function plots seismic data in form of color-coded pixels (intended for larger data sets). The simplest form is

```
s_cplot(seismic)
```

which plots the seismic structure seismic using default settings. To allow more general use the function does not abort if `seismic` is not a seismic structure, but rather a matrix. In this case the matrix columns are plotted as seismic traces and the vertical axis is annotated as "Samples".

There are numerous parameters — ranging from plot direction to color scheme — that can be set. An example is

```
s_cplot(seismic,{'limits',-8000,8000},{'direction','r2l'}), ...
          {'annotation','cdp'},{'colormap','hot'})
```

where the seismic-sample values in the interval from -8000 to 8000 are represented by colors. The plot direction is from left to right and the horizontal axis is annotated by CDP (the header `CDP` must, of course, be present in the seismic data set `seismic`. Furthermore, the color map `hot` predefined by MATLAB, has been chosen.

Color plot and wiggle trace plot can be overlaid. For example,

```
s_cplot(seismic,{'title','Wiggle trace overlay over color plot'})
s_wplot(seismic,{'title',''},{'figure','old'})
```

creates the plot shown in Figure 2.6. Of course, the two data sets can be different — say interval velocity and seismic (the "c" in `s_cplot` stands for color, the "w" in `s_wplot` denotes wiggle;

there is also a function `s_plot` that plots many-trace data sets in color and those with fewer traces (usually fewer than 102) as wiggle traces. Function `s_cplot` has four additional menu buttons. In
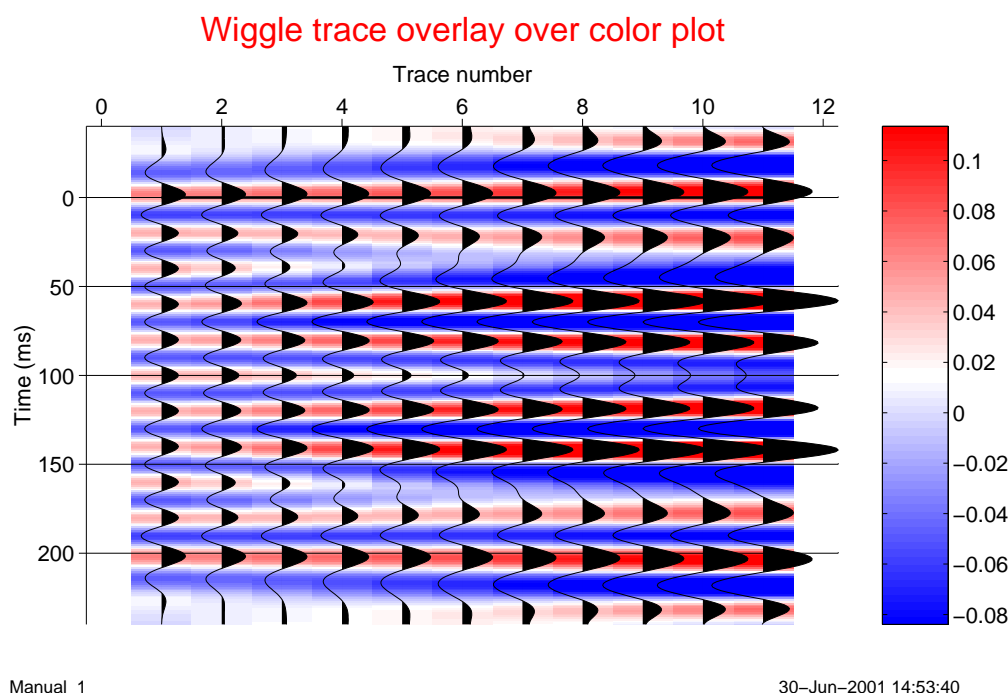


Figure 2.6: Wiggle trace plot on top of color plot of the same seismic data.

Matlab versions 7 and above these buttons have red labels. Due to a bug in Matlab the button labels are black for Matlab versions 6.5.2 and earlier.

**Save plot** Attached to this button is a drop-down menu with three choices to save the figure. The top two save the figure as an EMF file specifically for use in PowerPoint displays (EMS files can be edited in PowerPoint). The directory to which these files are saved can be specified in field `pp_directory` of global structure `S4M` (see Section 4.1 on pages 53 ff). The third choice save the files as encapsulated Postscript for use in LaTeX documents. The directory to which these file are saved is specified in field `eps_directory` of global structure `S4M`.

**Add scrollbars** When pressed for the first time this button creates scroll bars along the right-hand side and along the bottom of the figure. The size of the scroll window needs to be selected by the user. While the scroll bars are present the normal zoom function is unavailable. Clicking the button again will remove the scroll bars. The scroll bars can be turned on and off repeatedly.

**Tracking is off** Cursor tracking refers to a feature where the current position of the cursor as well as the seismic amplitude at the cursor location is displayed in the lower left corner of the figure. This button toggles cursor tracking on and off. While cursor tracking is on the normal zoom function is unavailable.

The above three menu buttons are also available for wiggle plots (`s_wplot`).

**Modify display** Attached to this button is a drop-down menu with a number of fairly self-explanatory choices. The option "Image limits ..." is meant to allow a user to specify what

seismic amplitudes represent the end colors of the color bar. This last option is only available for color plots.

### s_create_qfilter

**Purpose:** This function creates constant-Q absorption filters. For a range of $t$ and $Q$ values these filters have an amplitude spectrum

$$A(f) = \exp\left(-\frac{\pi f t}{Q}\right)$$

and a phase spectrum that ensures that they are causal.

### s_create_wavelet

**Purpose:** This function computes a Ricker wavelet or a wavelet with trapezoidal amplitude spectrum. In the latter case the phase options are minimum-phase, zero-phase,and maximum-phase. An example is

```
wav=s_create_wavelet({'type','min-phase'},{'frequencies',10,10,40,60},  ...
                     {'step',2})
```

which creates a minimum-phase wavelet with 2 ms sample interval.

### s_filter

**Purpose:** This function filters a seismic data set using an Ormsby band-pass filter (trapezoidal amplitude spectrum).

### s_header

**Purpose:** This function displays or manipulates trace headers of a seismic structure. It can be used to add, replace or delete trace headers (mnemonics, descriptions, and values). The simplest use is

```
s_header(seismic);
```

which prints to screen the name of every trace header together with its minimum value, maximum value, minimum and maximum trace-to-trace increment, units of measurement and description. The general usage of the function is:

```
seismic = s_header(seismic,action,mnem,values,units,description)
```

The second argument, `action`, specifies the action taken by the function. Possible values of `action` are:

- `add` Add header with mnemonic`mnem` to seismic data set. . Error if it already exists.

- `add_ne` Add header with mnemonic`mnem` to seismic data set. Replaces it if it already exists.

- `replace` Replaces header with mnemonic`mnem` in seismic data set. Error if it already exists.

- `delete` Delete header with mnemonic(s) `mnem` in seismic data set. Error if it header does (headers do) not exist.

- `delete_ne` Delete header(s) with mnemonic(s) `mnem` in seismic data set. No error if it one or more header does (headers do) not exist.

- `keep` Keep header(s) with mnemonic(s) `mnem` in seismic data set. Delete all others. Error if it one or more headers do not exist.

- `keep_ne` Keep header(s) with mnemonic(s) `mnem` in seismic data set. Delete all others. No error if it one or more header does (headers do) not exist.

- `rename` Rename header mnemonic, keep everything else the same

- `list` Print short list: for specified header mnemonic(s) it lists minimum and maximum value, smallest and greatest trace-to-trace change, units of measurement, and header description. This is the default that is being used if the seismic dataset is the only input argument.

## s_header_math

**Purpose:** This function manipulates trace headers of a seismic structure. It can be used to add or replace trace headers by arithmetic manipulation of existing headers. As usual the pseudo-header "trace_no" is implied. An example of its use is:

```
seismic = s_header_math(seismic,'add','offset=sqrt((sou_x-rec_x)^2  + ...
          (sou_y-rec_y)^2)','m','Source-receiver offset')
```

which computes a new header, `offset`, from the source and receiver coordinates. These coordinates must, of course be headers of data set seismic.

## s_history

**Purpose:** This function manipulates the processing history as stored in the field history. The simplest use is `s_history(seismic)` which prints to screen the contents of the history field of dataset `seismic`.

## s_header_plot

**Purpose:** This function plots one or more header values or cross-plots header values. An example is Figure 2.4 on page 9.

**s_header_sort**

**Purpose:** This function sorts seismic header value(s) and outputs an index vector (sorting can be performed in increasing or decreasing order). This index vector can be used to sort seismic traces by header values.

Assume the traces of dataset `wavelets` are wavelets estimated for a number of traces and/or time intervals and that the correlation coefficient is stored in header `cc_coefficient`. The following code segment will find and plot the five wavelets with the highest correlation coefficient.

```
index=s_header_sort(wavelets,{'headers','cc_coefficient'}, ...
        {'sortdir','decreasing'});
bestwavelets=s_select(wavelets,{'traces',index(1:5)});
s_wplot(bestwavelets)
```

**s_ispectrum**

**Purpose:** Interactive spectrum computation. This function allows a user to pick one or more rectangular windows on a seismic display after pressing the menu button labeled "Pick windows". Windows are picked by pressing the left mouse button at one corner of the window and releasing it at the opposite corner. A spectrum window will immediately display the average amplitude spectrum in the picked seismic window. The process can be repeated. Each window on the seismic display has a different border color and the same color is used to represent its spectrum. A window can be deleted by pressing the right mouse button anywhere on its border. When a window in the seismic figure is deleted the corresponding spectrum curve on the spectrum plot is deleted as well. To end window picking click on the same menu button again (its label has changed to "Done picking windows").

Upon exiting the function a legend is written to the spectrum window which includes traces and time range used for each spectral curve.

The spectrum display is protected from being deleted as long as the associated seismic window exists or as long as the "Done picking windows" button has not been pressed.

The simplest use of this function is

```
s_ispectrum(seismic)
```

If the data set has more than 101 traces (default setting of `S4M.ntr_wiggle2color`; see the description of `presets` on page 53 ff.) the seismic traces are displayed in form of a color plot, otherwise they are displayed in wiggle format.

```
b s_ispectrum(seismic)
```

An example is shown in Figures 2.7 and 2.8.

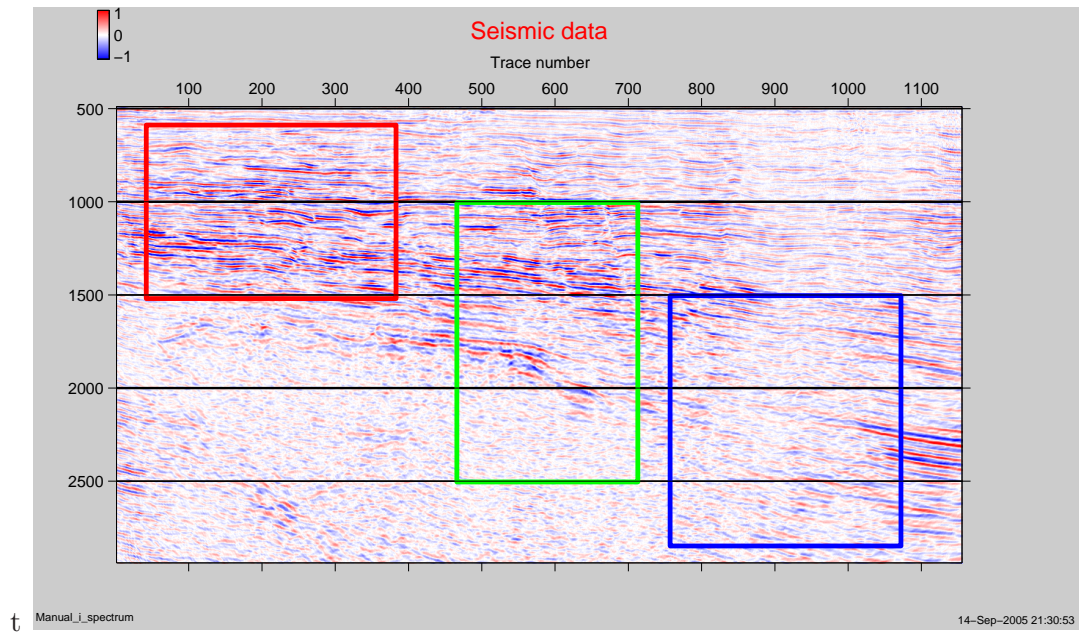The function has a number of keyword-controlled parameter options. For example

Figure 2.7: Seismic display created by `s_ispectrum` with three windows; the associated spectra are shown in the next figure.
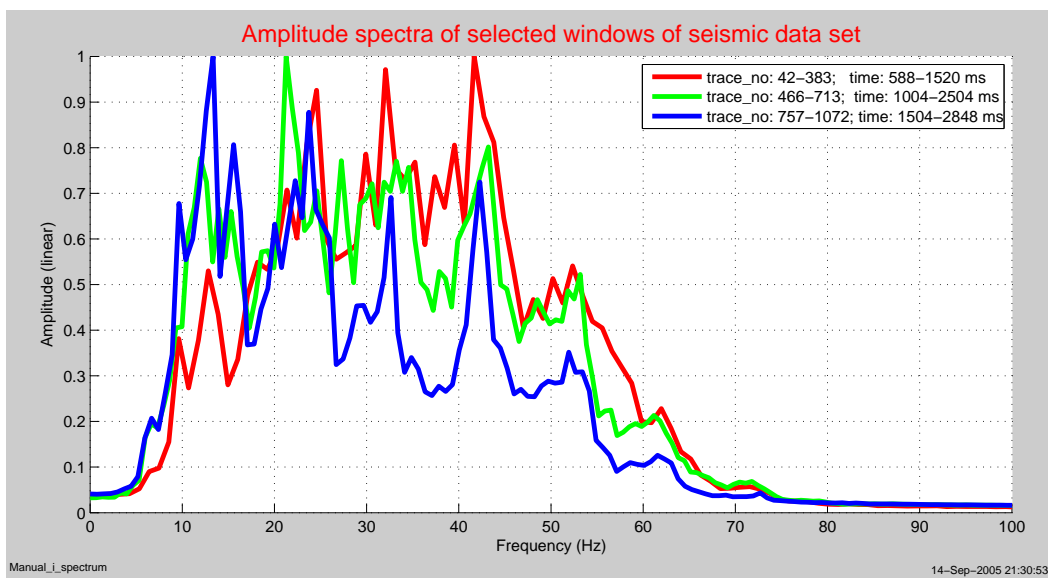


Figure 2.8: Spectra of the seismic data in the three windows shown on the seismic display above (created by `s_ispectrum`).

```
s_ispectrum(seismic,{'plottype','wiggle'},{'annotation','cdp'}, ...
{'frequencies',0,80})
```

specifies that the seismic plot should be wiggle-trace even if there are more traces than specified in `S4M.ntr_wiggle2color`, trace annotation should be in CDP number, and the frequency axis of the spectrum plot should range from 0 to 80 Hz. Other parameters can be found by using the `help s_ispectrum` command

### s_principal_components

**Purpose:** This function computes principal components of the input data set. Let $s_j(t)$ denote $J$ seismic traces. Then the first principal component is the function $s(t)$ which minimizes

$$\sum_{j=1}^{J}[s_j(t) - a_j s(j)]^2$$

with appropriately chosen scale factors $a_j$ (the function $s(t)$ is usually normalized to unit energy). The first principal component $s(t)$ can be regarded as the single seismic trace that "best represents" the $J$ different seismic traces $s_j(t)$. It is easy to show that the first principal component is nothing but a weighted stack of all the seismic traces (the weights may turn out be positive or negative).

The function `s_principal_components` either creates a new data set consisting of one or more of the principal components of the input data set or it creates a data set where each of the input traces is represented by the one or more of the principal components. An example, with all the default parameters, is

```
pcs=s_principal_components(seismic);
```

The output data set `pcs` has as many traces as the input data set but each trace is a scaled version of the first principal component of the traces of data set seismic, i.e. the $j$-th trace of the output data set is $a_j s(j)$ where $a_j$ is the scale factor in the equation above and $s(t)$ is the first principal component. The statement

```
pcs=s_principal_components(seismic,{'components',1:5});
```

creates a data set `pcs` where each trace is an approximation of the corresponding trace of the input dataset by means of the first 5 principal components (this assumes that the data set `seismic` has at least 5 traces). Obviously, there are as many output traces as there are input traces.

Assume the seismic data set `seismic` consists of 7 traces.

```
seis1_4=s_principal_components(seismic,{'components',1:4});
seis5_7=s_principal_components(seismic,{'components',5:7});
error=max(seismic.traces-(seis1_4.traces+seis5_7.traces))
```

Then the traces of `seis1_4` (the seismic data represented by the first 4 principal components) added to the traces of `seis5_7` (the seismic data represented by the last 3 principal components) produce the traces of the original input data. In theory the 7-component vector `error` should be zero. In practice, due to rounding errors, its elements are generally non-zero but very small.

On the other hand, if the principal components are requested,

```
pcs=s_principal_components(seismic,{'output','pc});
```

produces a seismic data set with only one trace, the first principal component $s(t)$ as defined above — regardless of the number of input traces.

```
pcs=s_principal_components(seismic,{'output','pc},{'components',2:5});
```

creates an output data set whose 4 traces are the second to fifth principal component.


## s_phase_rotation

**Purpose:** Rotate phase of each trace of the input data set by a user-specified amount. A simple example is

```
wavelet90 = s_phase_rotation(wavelet,90)
```

The output data set consists of all the traces of the input data rotated by 90 degrees (for a cosine a phase rotation by 90 degrees means conversion to -sine signal).

It is possible to specify more than one phase. Assume `wavelet` is a one-trace data set. Then

```
wavelets = s_phase_rotation(wavelet,[0:15:90])
```

creates a seven-trace data sets whose first trace is equal to the input trace. In general, the $n$-th trace is shifted by $15(n-1)$ degrees with respect to the input trace. All headers of the input data set are preserved. In addition, the phase is added to the headers of `wavelets`.

If the input data set has more than one trace there are two options for the output data set. It can be a structure array where each element is a seismic structure and contains the input data shifted by one of the angles specified. An example is

```
wavelets_array=s_phase_rotation(wavelets,[15:15:90],{'output','array'})
s_wplot(wavelets_array(3))
```

where the output is a 6-element structure array whose third element (the input data shifted by 45 degrees) is displayed in form of a wiggle plot. If no output form is specified or if `{'output','standard'}` the output data set is a normal seismic structure. The number of traces is equal to the product of input traces and number of phase angles specified. For each trace the phase angle is stored in a header whose default mnemonic is `'phase'`. The sequence is as follows: input traces

rotated by the first phase angle, input traces rotated by the second phase angle, ... . The following code fragment shows how one can extract all rotations of a particular trace (in this example trace 5):

```
%       Create a header with trace numbers
wavelets = s_header(wavelets,'add','trace_number', ...
          1:size(wavelets.traces,2),'n/a','Trace number')
%       Apply phase rotation
rotated_wavelets=s_phase_rotation(wavelets,[15:15:90],{'output','array'})
%       Select all traces with trace number 3
wavelet5 = s_select(rotated_wavelets,{'traces','trace_number == 5'}))
```

Then, using `s_header_sort`, one can resort the traces in any desired way.

### s_reflcoeff

**Purpose:** This function computes reflection coefficients from a seismic data structure representing impedance.

### s_resample

**Purpose:** This function samples a seismic data set to a new sample interval which can be greater or smaller than the sample interval of the input data.

The `wavelet` option prepends and appends a zero to the traces of the input data set prior to interpolation and then removes it again before returning the interpolation result.

### s_rm_trace_nulls

**Purpose:** This function removes common null values (NaN's) at the beginning/end of traces and replace other null values with zeros. Such null values are usually introduced if datasets with different start time and/or end time are concateneted (see `s_append`) or if traces of a dataset are shifted by different amounts (`s_shift`)..

### s_select

**Purpose:** This function outputs a subset of a seismic dataset by specifying a time range and/or a trace range. The most frequently used keywords are `times` and `traces`. The former is used to select a time range. For example,

```
s_select(seismic,{'times',1000,2000})
```

selects all samples of the seismic data set `seismic` within the time range from 1000 to 2000 ms.

```
s_select(seismic,{'times',seismic.first:2*seismic.step:seismic.last})
```

outputs every other sample of the seismic data set `seismic`. If the start time selected is less than the start time (and/or the end time selected is greater than the end time) of the seismic input data then null values are output for those times for which no input data are available. The null value can be chosen with the keyword . The default is `{'null',0}`.

Traces cane be selected by trace number, individual header values, or by means of a logical expression. For example, the first example above can be expanded to only read the first 10 traces:

```
s_select(seismic,{'times',1000,2000},{'traces',1:10})
```

An equivalent formulation is

```
s_select(seismic,{'times',1000,2000},{'traces','trace_no',1:10})
```

In this case the "pseudo-header" `trace_no`, which represents trace numbers, is used to specify the traces to output. Of course, any header of the data set `seismic` can be used to specify traces. The command

```
s_select(seismic,{'traces','cdp',1000:1010})
```

selects traces by CDP-number. The same selection can be achieved by means of a logical expression:

```
s_select(seismic,{'traces','cdp >= 1000 & cdp <= 1010'})
```

Likewise, the two commands

```
s_select(seismic,{'traces','cdp',1000:inf})
```

and

```
s_select(seismic,{'traces','cdp >= 1000'})
```

produce the same output. The former command shows that requests for traces that are not in the input data are ignored (this differs from the way a time range is selected).

In general, a logical expression for trace selection provides more flexibility in that multiple headers can be used. The command

```
s_select(seismic,{'traces','iline_no>1000 & iline_no<=1100 & xline_no==2000'})
```

outputs all in-lines with in-line numbers from 1001 to 1100 for cross-line 2000. The logical expression may contain MATLAB functions such as `fix, round, ceil, mod`. Thus

```
s_select(seismic,{'traces','cdp >= 1000 & mod(cdp,2) == 0'})
```

outputs all traces with even CDP number not less than 1000.

### s_shift

**Purpose:** This function applies time shifts to individual traces of a seismic data set.

### s_spectrum

**Purpose:** This function plots amplitude and/or phase spectra of one or more seismic data sets.

### s_stack

**Purpose:** This function stacks seismic traces. It a header mnemonic is specified, traces with the same value of that header are stacked; otherwise all traces of the input data set are stacked. Thus

```
stack = s_stack(seismic)
```

stacks all traces in `seismic` into one single trace while the more elaborate example

```
[stack,multiplicity] = s_stack(seismic,{'header','CDP'});
```

stacks all traces of `seismic` with the same CDP number. The number of output traces is thus equal to the number of different CDP's in `seismic`. The optional second output argument `multiplicity` is a seismic structure identical to `stack`, except that a sample of the field `traces` represents the number of samples that were stacked to compute the corresponding sample of `seismic`. This is only relevant if not all CDs have the same number of traces or if at least some traces have null values.

### s_tools

**Purpose:** This function writes one-line description for seismic function (in alphabetic order). The simplest call is

```
s_tools
```

which displays this description for all functions. The output can be restricted by adding a search string. For example,

```
s_tools create
```

will show all functions that create seismic data sets;

```
s_tools seg
```

will show all functions that deal with SEG-Y data sets. The search is not case-sensitive.

### s_wiener_filter

**Purpose:** This function computes one or more Wiener filters to convert one seismic data set into another.

### s_wplot

**Purpose:** This function plots a seismic data set in wiggle trace format. The simplest form is

> `s_wplot(seismic)`

which plots the seismic structure `seismic`. The vertical axis is annotated in time, the horizontal axis in trace number. To allow more general use the function does not abort if `seismic` is not a seismic structure, but rather a matrix. In this case the matrix columns are plotted as seismic traces and the vertical axis is annotated as "Samples".

A large number of keyword-activated options allows control of many aspects of the plot. More commonly used keywords are:

- {`'annotation'`,`string_parameter`} Specifies a header mnemonic for the annotation of the horizontal axis. Default is the trace number `'trace_no'`.

- {`'deflection'`,`numerical_parameter`} Amount of trace deflection. Default is 1.5.

- {`'direction'`,`string_parameter`} Possible values are `'l2r'` and `'r2l'`. Default is `'l2r'`.

- {`'interpol'`,`string_parameter`} Interpolation to create smooth plot. Options are `'cubic'`, `'v5cubic'`, and `'linear'`. Default is `'v5cubic'`.

- {`'orient'`,`string_parameter`} Orientation of plot. Options are `'landscape'` and `'portrait'`. Default is `'portrait'` for ten or fewer traces and `landscape` for more than ten traces.

- {`'peak_fill'`,`string_parameter`} Color of peak fill. Any MATLAB color is allowed. Default is `'k'` (black). An empty string_parameter means no fill.

- {`'scale'`,`string_parameter`} Scaling of the data prior to plotting. Options are `'yes'` (scale individual traces) and `'no'` (do not scale individual traces; this preserves relative amplitudes). Default is `'yes'`.

- {`'trough_fill'`,`string_parameter`} Color of trough fill. Any MATLAB color is allowed. Default is the empty string implying no fill.

- {`'wiggle'`,`string_parameter`} Color of wiggle. Any MATLAB color is allowed. Default is `'k'` (black). An empty string_parameter means no wiggle.
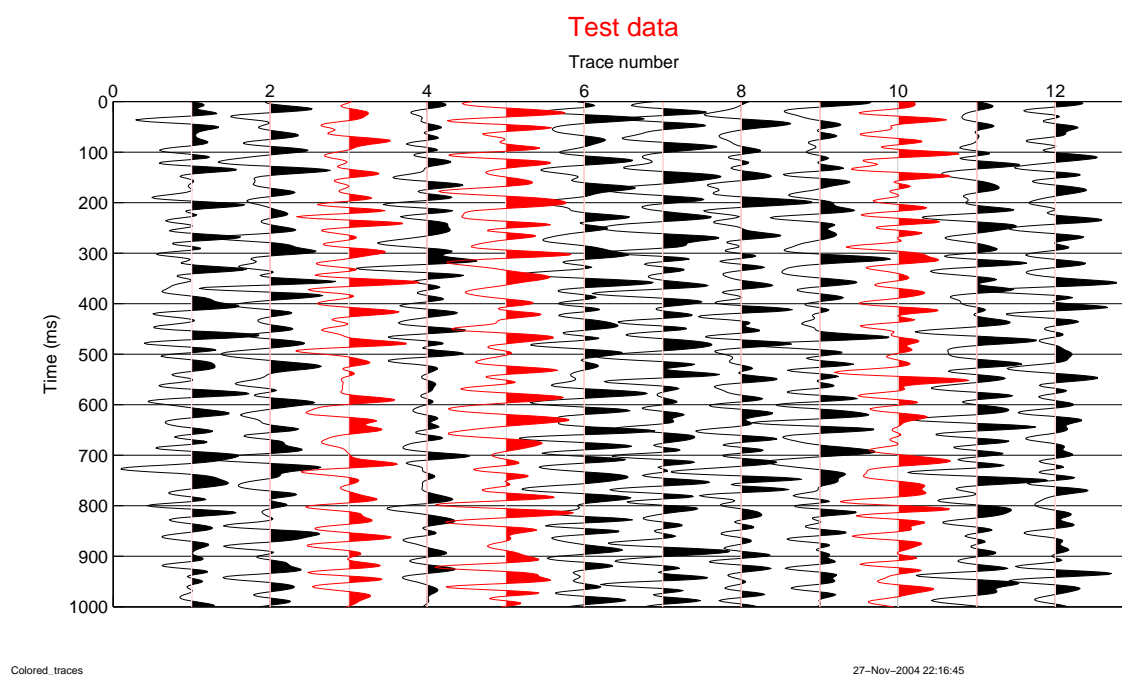
Figure 2.9: Plot of seismic traces in different colors.

Examples of seismic plots are, for example, in Figures 2.1 and 2.2. Seismic traces can be plotted in different colors. This is illustrated in Figure 2.9

The code that generated Figure 2.9 is shown below. It creates two copies, `seismic1` and `seismic2`, of the original seismic data. Traces 3, 5, and 10 of `seismic1` are set to null values (NaN) and thus will not be plotted. In `seismic2` all traces except 3, 5, and 10 are set to null values. Then the two data sets are plotted with `seismic2` plotted in the same figure window as `seismic1` (argument {'figure','old'}).

```
seismic=s_data;
ntr=size(seismic.traces,2);
bool=ones(1,ntr);
bool([3,5,10])=0;
seismic1=seismic;
seismic2=seismic;
seismic1.traces(:,find(~bool))=NaN;
seismic2.traces(:,find(bool))=NaN;
s_wplot(seismic1,{'deflection',0.9},{'orient','landscape'});
s_wplot(seismic2,{'deflection',0.9},{'figure','old'}, ...
                {'peak_fill','r'},{'wiggle','r'})
```

## show_segy_header

**Purpose:** This function reads a disk file written in SEG-Y format and outputs the EBCDIC header (converted to ASCII) to a file or prints it to the screen.

## write_segy_file

**Purpose:** This function writes a seismic structure to a disk file in SEG-Y format.

# Chapter 3

# WELL LOGS

## 3.1 A brief look at some functions for well log curves

The Log ASCII Standard (LAS) developed by the Canadian Well Logging Society represents the most popular ASCII file format for the exchange of well log data. In complete analogy to the seismic case discussed earlier there is a function

```
show_las_header
```

In principle, this function can have an input argument, the name of an LAS file and an output variable to receive the header. If no file name is given or if the filename provided is invalid a file selection box pops up and allows interactive file selection. The file selection function remembers the directory from which the file was read and, on a subsequent request for a LAS file will go to this directory right away. If no output variable is specified the header is displayed on the screen.

The two statements

```
wlog = read_las_file;
l_plot(wlog)
```

read an LAS file and display all curves in a figure window (batch mode). In the interactive mode (see description of `presets` on pages 53 ff.) a listbox with the curve mnemonics is displayed to allow interactive selection of the curves that should be plotted.

The function `read_las_file` can take two arguments. The first is the name of the LAS file to be read, the second is a print switch which allows one to keep track of the reading process. If a file name is not provided or if the file name is invalid a file selection window opens to allow interactive file selection. The well curves and ancillary data from the LAS file are stored in the log structure `wlog` which is then input to the function `l_plot` (most well-log-related functions start with "l_"). Figure 3.1 is an example of such a plot. Since no title was provided the plot title is taken from `wlog.name`, by default the name of the LAS file. The function `l_plot` has a number of options which can be found in the standard way by typing
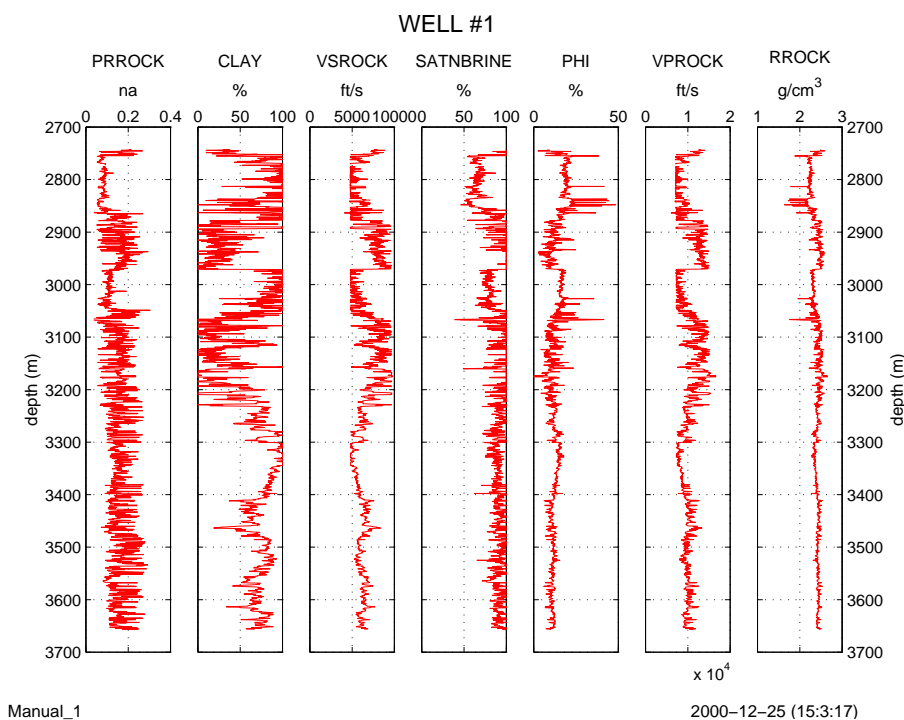
Figure 3.1: Plot of all traces of log structure logout.

        help l_plot

Presently there are some 70 utility-type functions that deal with log structures.[1] One way to find
out what is available is to run

        l_tools

which prints one-line descriptions of all functions that deal with log structures. To make the list
more specific a keyword may be added. For example

        l_tools las

lists not only those functions that deal with LAS files (the search is not case sensitive) but also
other functions that have the character group "las" as part of their description (the description of
l_elastic_impedance  does not fit onto one line of this manual and, hence, as been broken into
two) .

        l_elastic_impedance   Compute ''elastic impedance'' for user-defined
                              angles of incidence
        read_las_file         Read disk file in LAS 2.0 format
        show_las_header       Output/display header of LAS 2.0 file
        write_las_file        Write disk file in LAS 2.0 format

---

[1]Only a subset of the available log-related functions is included in the public-domain version.

In a LAS file each log curve is associated with at least three pieces of information: a mnemonic, units of measurement, and a description. Examples of mnemonics are "DT", "DTCO", or "BHC" for sonic interval transit time, "RHOB" for bulk density, etc.; examples of units of measurement are "us/ft" meaning $\mu$s/ft, or "g/cm3", meaning g/cm$^3$. Curve mnemonics are frequently chosen to provide some idea of how the curve has been measured/computed and hence may vary from one logging company or log analyst to the next. There is also a tremendous variation in the way units are denoted. When writing a LAS file one commercial program, for example, denotes feet by "F" which lead another commercial program, while converting non-metric units to MKS, to output depth in Kelvin, as it obviously interpreted "F" as Fahrenheit.

The function `read_las_file` leaves header mnemonics as they are in the LAS file (the only exception is "DEPT" which is converted to "DEPTH"). However, units of measurements are converted to a standard form (for example, LAS file writers have come up with at least 6 different ways to say "g/cm$^3$") — at least for those curves that are most relevant for someone dealing with seismic data. Of course, misinterpretations are possible; however, "F" would be interpreted as "feet" and converted to "ft" (and not assumed to represent Fahrenheit). This conversion is performed in function `unit_substitution` which contains the list of what is converted into what.

To simplify their use many functions assume standard mnemonics for curves. These standard mnemonics are defined in function `systemDefaults` in a global structure called `CURVES`. They are shown in Tables 1-3 on pages 57 ff.

Thus "acoustic impedance", for example, has the mnemonic `aImp`. This list is likely to grow.

Several of the curves (those identifying lithology) are designated as logical. This means that their values are either 1 (true) or 0 (false). Obviously, curve mnemonics can be changed globally or locally at any time. The following code fragment illustrates this.

```
well_log = read_las_file;idlmread_las_file
well_log = l_rename(well_log,{'RHOB','rho'},{'DTCO','DTp'});   1a
well_log = l_seismic_acoustic(well_log);
```

It reads an LAS file with sonic and density curves, and changes their mnemonics of from "DTCO" to "DTp" and from "RHOB" to "rho", respectively. In the last statement the function, `l_seismic_acoustic`, assumes that density and sonic curves use these standard mnemonics, computes compressional velocity curve "Vp" and acoustic impedance "aImp" and adds them to the data set `well_log`.

One of the parameters set in the initialization function `presets` (see page 53 ff.) is `S4M.case_sensitive`. If this parameter is set to "false" (0), which is the default, then 1a could have been written as

```
well_log = l_rename(well_log,{'rhob','rho'},{'dtco','dtp'});   1b
```

In fact any combination of upper-case letters and lower-case letters is permissible.

However, a user is not wedded to these standard mnemonics. First of all, they can be changed in function `systemDefaults` or by means of function `l_redefine`. Assume it were necessary to preserve the original mnemonics in the example above. Then one could write

```
    well_log = read_las_file;
    l_redefine({'rho','RHOB'},{'DTp','DTCO'});
    well_log = l_seismic_acoustic(well_log);
```

where `l_redefine` changes the default mnemonics `rho` and `DTp` to `RHOB` and `DT`, respectively (note that `l_redefine` has no output argument; it changes fields of the global structure `CURVES`)

But the standard mnemonics can also be changed on a case-by-case basis. One could write

```
    well_log = read_las_file;
    well_log = l_seismic_acoustic(well_log,{'rho','RHOB'},{'DTp','DTCO'});
```

The two additional arguments tell `l_seismic_acoustic` that the density curve has mnemonic `RHOB` (instead of the standard mnemonic `rho`) and the sonic log has mnemonic `DTCO` (instead of the standard mnemonic `DTp`).

The two curves computed in `l_seismic_acoustic` in all the above cases would still have default mnemonics `Vp` and `aImp`. But this could be changed as well. With either

```
    well_log = read_las_file;
    l_redefine({'rho','RHOB'},{'DTp','DTCO'},{'Vp','Vel'},{'aImp','IMP'});
    well_log = l_seismic_acoustic(well_log);
```

or

```
    well_log = read_las_file;
    well_log = l_seismic_acoustic(well_log,{'rho','RHOB'},{'DTp','DTCO'}, ...
                 {'aImp','IMP'},{'Vp','Vel'});
```

the mnemonics of acoustic impedance and compressional velocity will be `IMP` and `Vel`, respectively. In this last case,which avoids the call to function `l_redefine`, the standard mnemonics are not changed. Consequently, a subsequently called function that needs, for example, the acoustic impedance must be told explicitly what its mnemonic is. Thus the true benefit of standard mnemonics accrues when a number of functions are to be executed in sequence; there is no need to tell each of them what curve mnemonics to use. Renaming of curve mnemonics or redefining of standard curve mnemonics occurs only once — preferably right after reading the LAS file. Thereafter, the code need not be changed if another log is to be processed in the same way.


## 3.2   Description of log structures

The log-related functions assume that a log is represented by a structure which — in addition to the actual log curves represented by a matrix — contains necessary ancillary information in form of the mnemonics associated with each curve, the units of measurement, and a more understandable description. Furthermore, there can be parameters such as Kelley bushing elevation, water depth, etc. Basically, a log structure has nine required fields and any number of optional fields. This description uses the variable `ncurves` to denote the number of curves in the well log.

- `type` General identifier of the type of structure. For a well log it is the string `'well_log'`.

- `tag` This field is used to identify the type of table — if appropriate. The default tag is `'unspecified'`.

- `name` Name associated with the data set. This could be the well name. When read from a las file it is the file name without `.las` extension. By default it is used as a plot title.

- `first` Start of log, (first depth value).

- `last` End of log, (last depth value).

- `step` Depth increment (0 if unequal).

- `null` No-data (null) value; generally NaN. This field is only present if there are null values. In LAS files null values are frequently represented by the number -999.25.

- `curves` A matrix of log curves with `ncurves` columns; the first column must be DEPTH and this description always refers to depth with the understanding that it could be something equivalent such as TIME.

- `curve_info` Cell array of dimension `ncurves`×3. The first column contains the curve mnemonics, the second column the units of measurement, and the last column a description of each curve. There must be one row for each curve.

## 3.3 Description of functions for well log analysis

### l_check

**Purpose:** This functions checks a log data structure for formal errors such as inconsistencies, missing required fields, etc. It takes only one argument, the data set to be checked. An example is

> `l_check(wlog)`

which checks for errors of the structure `wlog`; if indeed errors are found, it will print messages explaining them. Otherwise, the message `No formal errors found in ''wlog''` will be printed.

It is expected that every log-related function listed in this manual will pass this consistency test; however, users may modify structures outside of these functions and, if these changes are more severe, checking if they violate any formal requirements may be appropriate.

### l_checkshot

**Purpose:** This function applies check shot corrections to sonic log; computes depth-time relationship

## l_compare

**Purpose:** Compare two or more log curves. The function plots one or more log curves from one or more wells into one figure window. A simple example is

```
l_compare({log1,'Vp'},{log1,'Vs'})
```

which plots compressional velocity and shear velocity of log structure `log1`. Of course the curves need not come from the same log structure.

```
l_compare({log1,'Vp'},{log2,'Vp_pred'})
```

compares a measured velocity curve of `log1` with a predicted velocity curve from `log2`. If depth units or units of measurement of the curves differ they are automatically converted to those of the first log. There are a number of parameters that can be set. The following function call

```
l_compare({log1,'DT',{'color','r'},{'linewidth',2},{'legend','Sonic log'}},...
{log2,'DTCO',{'color','g'},{'linewidth',2},{'legend','Sonic log'}})
```

plots the first curve in red with a line width of 2 points and the second curve in green with the same line width.

## l_convert

**Purpose:** This function converts a matrix of curve values, curve names, curve units of measurement, curve description, etc. into a well log structure. An example is

```
wlog=l_convert(columns,'depth','ft','Depth'; ...
          'DTp','us/ft','Sonic'; ...
          'VpVs','n/a','Vp-Vs ratio'; ...
          'rho','g/cm3','Density'; ...
          'epsilon','n/a','epsilon'; ...
          'delta','n/a','delta');
```

which converts the six-column matrix `columns` into a log structure with six curves with the mnemonics `depth, DTp, VpVs, rho, epsilon`, and `delta`. The first column of matrix `columns`, representing the depth, must be strictly monotonic.

## l_crossplot

**Purpose:** This function makes cross-plots of log curves. A simple example, is
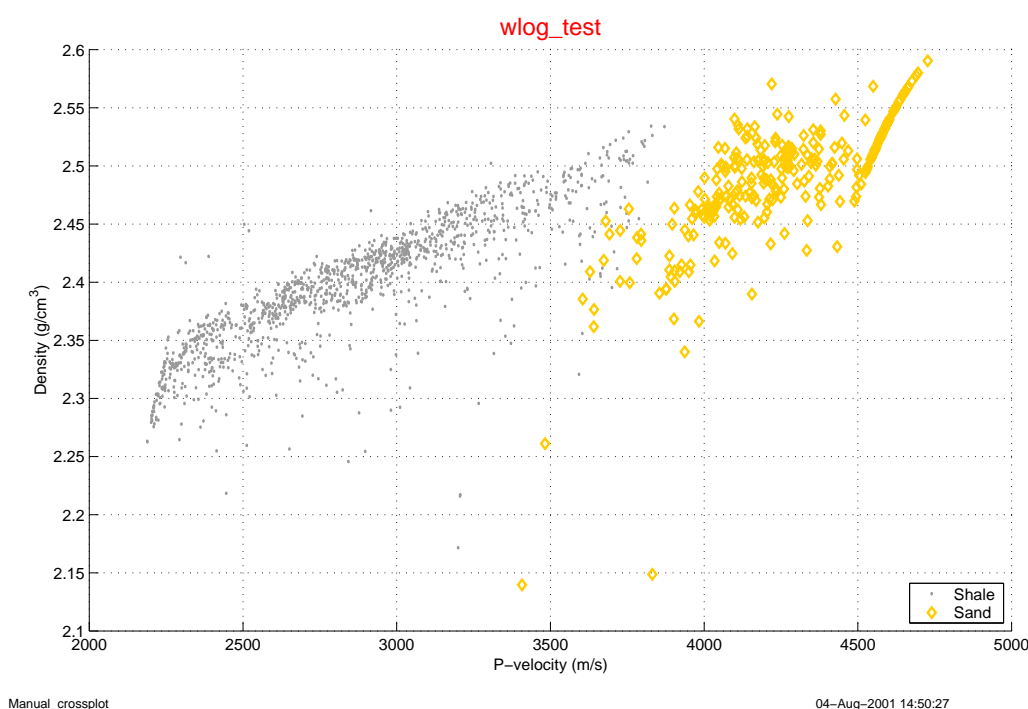
```
l_crossplot(wlog,'Vp','rho')
```

Figure 3.2: Cross-plot of velocity and density for two lithologies: sand (yellow diamonds) and shale (gray dots); created by two calls to `l_crossplot`

which creates a plot in which the horizontal axis is velocity and the vertical axis the density (assuming curves `Vp` and `rho` are present in the log structure `wlog`). A plot like this could also be generated with modest additional effort with standard MATLAB tools. The function is more useful for more elaborate plots; an example is shown in Figure 3.2; it has been generated by

```
l_crossplot(wlog_test,'Vp','rho',{'depths',3000,3500}, ...
        {'rows','shale & rho > 2.1 & Vp < 5000'}, ...
        {'color',[0.6, 0.6, 0.6]},{'marker','.'})
l_crossplot(wlog_test,'Vp','rho',{'depths',3000,3500}, ...
        {'rows','sand & rho > 2.1 & Vp < 5000'}, ...
        {'color',[1 0.8 0]},{'marker','d'},{'figure','old'})
legend('Shale','Sand',4)
```

Here, log values are restricted to the depth range from 3000 to 3500 m, Furthermore, only depths are considered for which the density is greater than 2.1 g/cm$^3$ and the velocity is less than 5000 m/s. The first call to `l_crossplot` displays the velocity-density relationship for shales, the second the one for sands.

### l_curve_math

**Purpose:** This function performs arithmetic using curves to append new curves or replace existing ones. A curve is created through arithmetic operations on existing curves. A simple example is the

computation of the acoustic impedance from a sonic curve and a density curve. The resulting log structure `logout` has all the curves of `login` and, in addition, an impedance curve.

```
logout = l_curve_math(login,'add','aImp=(1.0e6/DTp)*rho',...
         'ft/sec x g/cm3','Acoustic impedance')
```

The first argument is the input log, the second argument defines the operation to perform (`add`, `add_ne`, or `replace`) and the third is an expression in MATLAB syntax. A new curve with mnemonic `aImp` is created by dividing $10^6$ by the sonic log (creates velocity) and multiplying the result by the density log. The last two arguments are the units and the description of the new curve. In this example it is assumed that the original log has curves with mnemonics `DTp` and `rho` representing a sonic and a density curve and that their units of measurement are $\mu$sec/ft and g/cm$^3$, respectively. The difference between `add` and `add_ne` is that with the former the function aborts with an error message if the mnemonic is already in use while it will overwrite it (`_ne` means "no error") in the latter case. On the other hand, `replace` will abort with an error message if the curve to be replaced does not exist.

Another example is

```
logout = l_curve_math(login,'replace','depth=depth-login.ekb',
'm','Depth below sea level')
```

which changes the depth column (first curve) from measured depth to depth below sea level (ground level) by removing the Kelly bushing elevation from the depth (assuming a parameter `login.ekb` exists and depth and Kelly bushing elevation are measured in meter). If the first curve (depth) is changed the fields `first`, `step`, and `last` of `logout` will reflect this change.

More sophisticated results can be achieved by repeated use of `l_curve_math`. The following two lines of code create a curve of shale velocity with `NaN's` wherever there is no shale.

```
wlog = l_curve_math(wlog,'add','Vp_shale=Vp',l_gu(wlog,'Vp'), ...
              'Compressional velocity in shale')
wlog = l_curve_math(wlog,'replace','Vp_shale(find(shale == 0)) = NaN')
```

The first line simply adds a copy of the velocity curve; the second function then places `NaN's` in this curve wherever the shale-curve is zero (the shale-curve is a logical curve which is 1 (true) when shale is present and 0 (false) otherwise. Since the second function call replaces an existing curve the last two arguments (units of measurements and description, respectively) can be omitted.

See also `s_select`.

### l_interpolate

**Purpose:** This function interpolates null values of all curves specified by an optional list of mnemonics. The function assumes that null values are represented by NaN's.

## l_lithocurves

**Purpose:** Create "logical" curves to identify lithology. The curve values are 1 (true) if the lithology is present at a depth value and 0 (false) if it is not. It assumes that the input log has at least the curves `Vclay` and computes the following additional curves if they do not exist (by default, the function aborts with an error message if one of the lithologies to be created already exists; this behavior can be changed via the keyword `action`)

```
sand   sh_sand   shale
```

based on the (default) condition

```
   sand  =  vclay   <   0.25
sh_sand  =  vclay  >=   0.25   &  vclay <= 0.35
  shale  =  vclay   >   0.35
```

The cut-offs used above can be changed via keyword `'clay_cutoffs'`.

If the input log has, in addition, the curve `Sbrine` the following additional logical curves are computed (if they do not exist)

```
hc_sand   w_sand
```

Thus `sand` is split up into wet sand and hydrocarbon sand based on the (default) condition

```
hc_sand  = sand   &   sbrine <= 0.60
 w_sand  = sand   &   sbrine > 0.60
```

The water saturation cut-off can be changed via keyword `'sw_cutoff'`. The above conditions assume that the units of `Vclay` and `Sbrine` are fractions; they are appropriately modified if one or both are in percent. Hence, if the default cut-offs are used and at least the `Vclay` curve is present then all it takes is

```
wlog=l_lithocurves(wlog)
```

It should be noted that the same end can be achieved, with somewhat more typing, via

```
wlog=l_curve_math(wlog,'add','sand=vclay < 0.25','logical','Sand')
wlog=l_curve_math(wlog,'add','sh_sand=vclay >= 0.25 & vclay < 0.35', ...
           'logical','Shaley sand')
wlog=l_curve_math(wlog,'add','shale=vclay >= 0.35','logical','Shale');
```

Splitting sand into wet and hydrocarbon sand would require two more calls to `l_curve_math`.

### l_litho_plot

**Purpose:** Plot log curves with colors and/or markers representing lithology. This requires that 'logical' curves representing lithology exist in the log structure. Such curves could, for example, be created by means of function `l_lithocurves`.

The simplest example is

```
l_litho_plot(wlog)
```

which uses all the lithology curves in the log structure `wlog` to plot all the non-lithology curves. In general, it is preferable to restrict the number of curves and the number of lithologies.

```
l_litho_plot(wlog,{'curves','Vp','rho','aImp'}, ...
             {'lithos','shale','sh_sand','w_sand','g_sand'})
```

which only plots the P-velocity, density, and acoustic impedance with shale, shaley sand, wet sand, and gas sand indicated by specific colors and markers. For a number of frequently used lithologies these colors/markers are predefined (see `help l_litho_plot` for specifics). Shale, for example is represented by a gray dot (a small symbol because shale is generally quite dominant). It is, of course possible to change all this. For example, the above example could be expanded to

```
l_litho_plot(wlog,{'curves','Vp','rho','aImp'}, ...
             {'lithos','shale','sh_sand','w_sand','g_sand'}, ...
             {'shale','k','.'},{'sh_sand',[0.6, 0.6, 0.6],'.'})
```

which redefines shales to be represented by black dots and shaley sands by gray dots. This example also illustrates that colors represented by characters (`'r'` for red, `'k'` for black, etc.) as well as the RGB definition of colors can be used. An example of such a lithology plot is shown in Figure 3.3. It was created by

```
aux=l_litho_plot(wlog2,{'curves','Vp','rho'},{'depths',2000,2500}, ...
             {'lithos','shale','w_sand','hc_sand'})
set(aux.ahandles(1),'XDir','reverse')
```

Only log values that represent on of the three lithologies shale, wet sand, or hydrocarbon sand are displayed. Log samples representing, say, shaley sand, limestone, or coal are not plotted. The function `l_litho_plot` has an optional output argument, the structure `aux`; it has a field `ahandles` that is an array with the handles to all subplot axes. This is used here to reverse the x-axis of the first subplot to make it conform to standard log-display practice.

### l_redefine

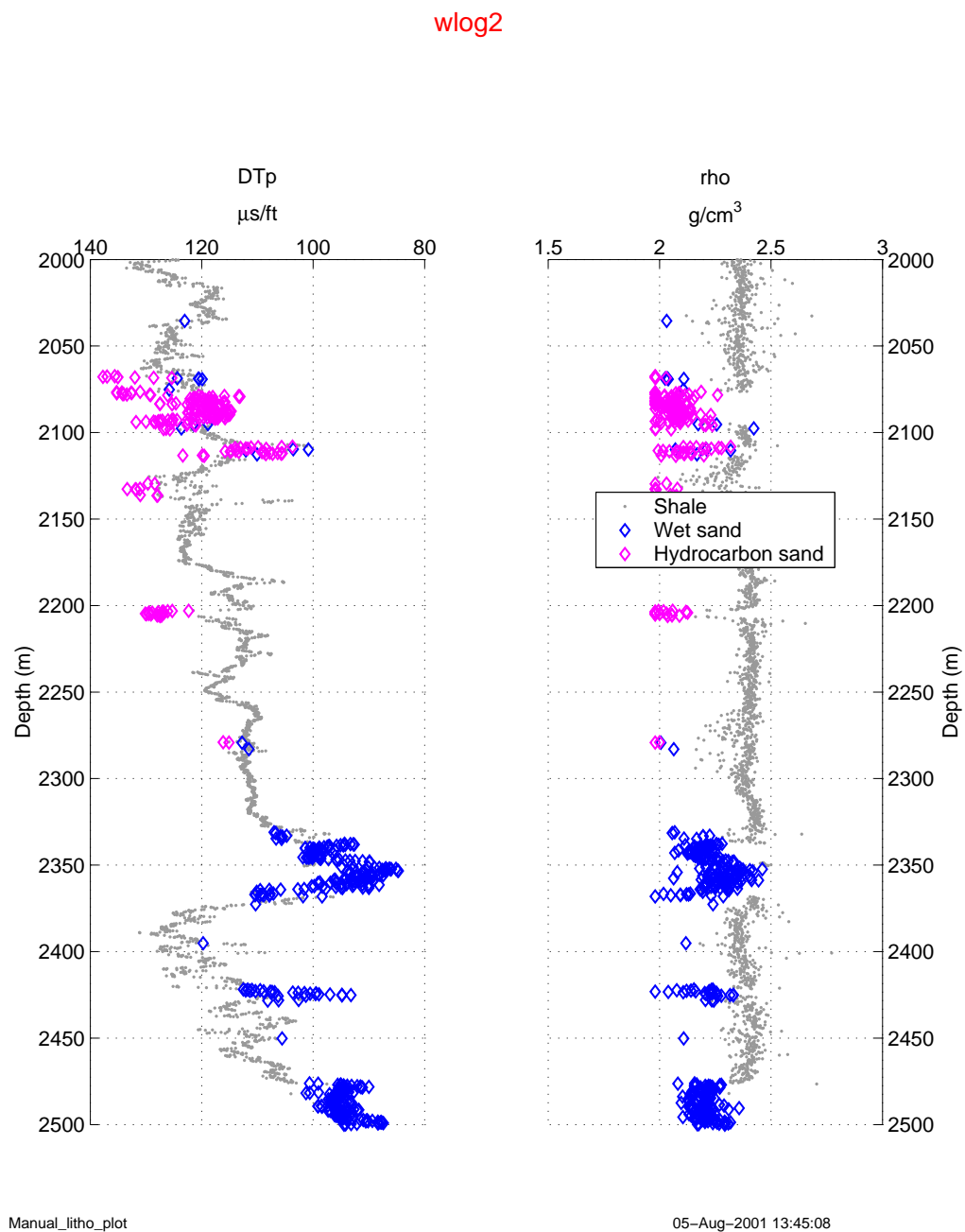**Purpose:** Change one or more default (standard) curve mnemonics. The simplest form is

wlog2



Figure 3.3: P-velocity and density with lithology (shale, wet sand, hydrocarbon sand indicated by different colors and markers; created by `l_litho_plot`

```
l_redefine({'rho','rhob'})
```

which changes the default curve mnemonic for the bulk density from `rho` to `rhob`. This is an alternative to changing the curve mnemonics in a log structure to be equal to the default mnemonics (see `l_rename`). An arbitrary number of default mnemonics can be changed with one `l_redefine`.

```
l_redefine({'rhob','rho'},{'DT','DTp'},{'VCL','Vclay'})
```

changes default curve mnemonics `rho, DTp, Vclay` to `rhob, DT, Vcl`, respectively.

This function changes the values of fields in global structure `CURVES` (see the description of `presets` on pages 53 ff.)

## l_regression

**Purpose:** Compute attribute relationships between two or more log curves. A simple example of its use is

```
wlog = l_regression(wlog,'vs=x1*vp+1000*x2')
```

which computes parameters `x1` and `x2` so that the shear velocity `vs` is expressed "as well as possible" in terms of a linear relationship with the compressional velocity `vp`. `Vs` (assuming that curve mnemonics are not case-sensitive) and `Vp` must be curve mnemonics for shear velocity and compressional velocity, respectively. The default meaning of the expression "as well as possible" is (L1 norm)

$$|v_s - x_1 v_p + 1000 x_2| = \min.$$

It is also possible to specify the L2 norm.

$$|v_s - x_1 v_p + 1000 x_2|^2 = \min).$$

by means of the keyword `norm`

```
wlog = l_regression(wlog,'vs=x1*vp+1000*x2',{'norm','L2'})
```

`l_regression` uses the MATLAB functions `fminunc` (for unconstrained minimization) and `fmincon` (for constrained optimization). These functions require a starting value for each variable, and if none are provided as arguments (as in the example above) then `l_regression` sets the starting values of all parameters equal to 1.

In the relationship above the parameter $x_2$ is multiplied by 1000. In theory, this is not necessary. In practice, any parameter estimation program works best if the unknowns are balanced (for linear systems this is equivalent to balancing matrix columns). The factor 1000 has been chosen to be in the order of magnitude of compressional and shear velocities measured in m/s. Hence the default starting value is not orders of magnitude off.

In general, it is good practice to use constraints to limit the search performed since regression parameters could easily range from a compaction factor of the order of $10^{-4}$ m$^{-1}$ to several thousand

ft/s as in the example above. It is even helpful to tell `l_regression` if one or more parameters must be non-negative. Bounds on the parameter values are particularly important if parameters are exponents or part of exponential functions.

In the example above all depth levels in the log structure are used for which there are valid compressional AND shear velocities. It is possible to restrict these values to a range of depth and/or use logical constraints. For example

```
wlog = l_regression(wlog,'vs=x1*vp+1000*x2',{'depths',2000,3000}, ...
                {'rows','sand'})
```

restricts the samples to sands (for which the values of logical curve `sand` are equal to 1) in the depth range from 2000 to 3000 m (assuming the depth units are m). Of course, a logical curve with mnemonic `sand` must exist in the log structure `wlog` (see function `l_litho_curves`). The keywords `rows` and `depths` are those used in `l_select` to extract specific rows from a log structure. The above statement could also have been written as

```
wlog = l_regression(wlog,'vs=x1*vp+1000*x2', ...
                {'rows','sand & depth >= 2000 & depth <= 3000'})
```

The expression relating shear velocity and compressional velocity used above is linear. It does not have to be. Any valid MATLAB statement is allowed provided the only variables are curve mnemonics and up to 9 parameters (`x1, x2, ...., x9`) and there is only one variable (mnemonic of an existing curve) to the left of the equal sign. The parameters can be constrained by specifying lower and upper bounds. For example

```
wlog = l_regression(wlog,'vs=x1*vp+1000*x2',{'lbounds',0,-inf})
```

requires that `x1` is non-negative since it has a lower bound 0; `x2` is not constrained since its lower bound is $-\infty$. The keyword `ubounds` can be used to set upper bounds. If bounds are given they must be given for all parameters, but `inf` and `-inf` are allowed. If bounds are known they should be given to prevent the algorithm from going astray.

As mentioned before fitting can be done via either the L1-norm or the L2-norm. However, there is a third option. It is based on the L1-norm but treats positive deviations different from negative deviations. With, say,
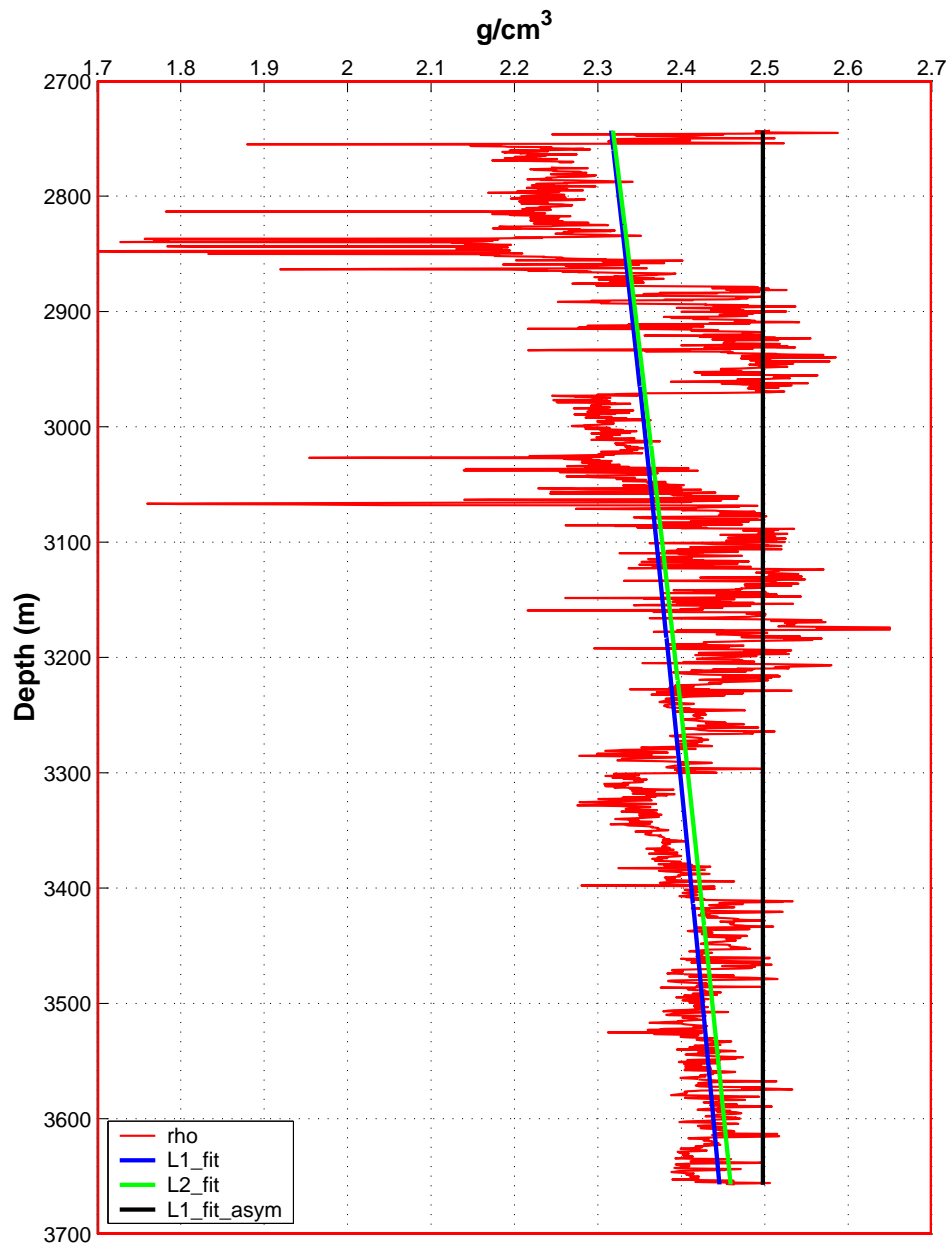
$$g(x_1, x_2) = v_s - x_1 v_p - 1000 x_2;$$

it can be expressed as

$$f_1 H(g(x_1, x_2)) * g(x_1, x_2) - f_2 H(-g(x_1, x_2))g(x_1, x_2) = \min. \tag{3.1}$$

where $H(\cdot)$ is the Heaviside function

$$H(x) = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x < 0 \end{cases}$$

Figure 3.4: Density log with superimposed trend curves

and $f_1$ and $f_2$ are positive factors. Obviously, expression (3.1) is equivalent to $f_1|g(x_1, x_2)|$ if $f_1 = f_2$. However, if $f_1 < f_2$, then positive deviations receive lower weight than negative ones. The regression will tend to better match the large values. If $f_1 > f_2$, the result is the opposite. Figure 3.4, which is the result of the following MATLAB statements illustrates this.

```
expression = 'rho=x1*(depth/1000)^x2';
```

```
[log_curves,aux1] = l_regression(log_curves,expression, ...
          {'norm','L1'},{'lbounds',0,0},{'mnem','L1_fit'}, ...
          {'description','L1 fit to density'});
[log_curves,aux2] = l_regression(log_curves,expression, ...
          {'norm','L2'},{'lbounds',0,0},{'mnem','L2_fit'}, ...
          {'description','L2 fit to density'});
[log_curves,aux3] = l_regression(log_curves,expression, ...
          {'norm','L1',0.1,1},{'lbounds',0,0},{'mnem','L1_fit_asym'}, ...
          {'description','L1 asymm.  fit to density'});
l_compare({log_curves,'rho'},{log_curves,'L1_fit',{'linewidth',2}}, ...
          {log_curves,'L2_fit',{'linewidth',2}}, ...
          {log_curves,'L1_fit_asym',{'linewidth',2}}, ...
          {'parameters',{'lloc',3}})
```

L1-norm and L2-norm fits are close together with the latter slightly higher, and the asymmetric fit is clearly separated from the two.

### l_rename

**Purpose:** Change one or more curve mnemonics. The simplest form is

```
log = l_rename(log,{'rhob','rho'})
```

which changes the mnemonic `RHOB` of a density curve (assuming curve mnemonics have been defined as not case-sensitive; see function `presets`) to `rho`, the standard density mnemonic used in SeisLab. The above statement is equivalent to

```
log = l_curve(log,'rename','rhob','rho')
```

In both statements units of measurement and curve description are not changed. The main difference between `l_rename` and `l_curve` with the `'rename'` option is that the former allows renaming of more than one curve. Thus,

```
log_new = l_rename(log,{'rhob','rho'},{'dtco','DTp'},{'VCL','Vclay'})
```

renames the three curves with mnemonics `RHOB, DTCO`, and `VCL` to the standard SeisLab mnemonics. Curve mnemonics are renamed in the order they are listed. Hence,

```
log_new = l_rename(log,{'rho','rho1'},{'rhob','rho'})
```

first changes `rho` to `rho1` and then changes `rhob` to `rho`.

The function aborts with an error message if a new curve mnemonic is already in use.

A related function is `l_redefine` which changes/redefines one or more default (standard) curve mnemonics. This is an alternative to changing the curve mnemonics in a log to equal the default mnemonics.

### l_resample

**Purpose:** This function re-samples the curves of a log to a new, uniform sample interval. This sample interval can be smaller or larger than the original sample interval of the input data.

### l_select

**Purpose:** Select a subset of log curve(s) and/or depth values from a log structure.

A simple example is

```
new_log = s_select(wlog,{'curves','DTp','rho','shale'})
```

which copies three curves (compressional sonic, density, and a logical curve which is true when a sample represents shale and false if not) to a new log structure, `new_log`. The function terminates abnormally with an error message if any one of the three curves is not present in `wlog`. Of course, one can also chose a depth range. The following function call is identical to the one above, except that the three curves in `new_log` are restricted to depths ranging from 3000 to 4000 (the units are the those used for the depth).

```
new_log=s_select(wlog,{'curves','DTp','rho','shale'},{'depths',3000,4000})
```

There is a third way in which data from a log structure can be selected; the following line of code shows an example.

```
new_log = s_select(wlog,{'rows','shale == 1'})
```

In this case `new_log` includes all curves of `wlog` but only for those depths for which the shale marker is equal to 1 (i.e. the log curves only for shale).

### l_plot

**Purpose:** This function plots log curves. Using all the default settings

```
l_plot(wlog)
```

plots all the curves in the log structure `wlog`, each with its own axes. For fewer than 5 curves the default figure orientation is "portrait", otherwise it is "landscape". The string in field `name` is plotted as title. An example of the output is shown in Figure 3.1. For log structures with a large number of curves is it may be more practical to restrict the number of curves plotted. An example is

```
l_plot(slog,{'curves','dtp','dts'},{'depths',2000,3000})
```

which not only restricts the number of curves plotted to compressional and shear velocity but also the range of depths (from 2000 to 3000 in whatever depth units the log structure uses). Other keywords that can be used are `'figure'` (to specify if a new figure should be created (default) or if an existing figure should be used), `'orient'` to specify figure orientation if the default described above is not appropriate, and `'color'` to assign a curve color (default is red); all curves are plotted in the same color.

If one of the curve mnemonics requested via keyword `'curves'` does not exist in the log structure a warning message is issued and the corresponding subplot window is empty.

### l_plot1

**Purpose:** Like `l_plot` this function plots log curves. However, all log curves are plotted in one and the same window. If the units of measurement of all plotted curves are the same they are used to annotate the horizontal axis. If this is not the case, the horizontal axis ia annotated from 0 to 1, and all curves are scaled and shifted in such a way that the smallest value is 0 and the largest value is 1. The true minimum and maximum values are plotted as part of the legend next to the curve mnemonic. `l_plot1` understands all the keywords used by `l_plot` (with the exception that the plural `'colors'` is used instead of the singular in `l_plot`). Hence the example above reads

```
l_plot1(slog,{'curves','dtp','dts'},{'depths',2000,3000})
```

The number of curves plotted cannot exceed the number of colors available. Since there are 7 colors preset, a maximum of 7 curves can be plotted without increasing the number of colors. If there are more curves to plot than there are colors available, curves for which there are no colors left will not be plotted and an alert message will be printed.

Additional keywords, not available in `l_plot`, are `'linewidth'` and `'lloc'`; they set the line width of the curves and the location of the legend.

### l_tools

**Purpose:** This function writes one-line description for log function (in alphabetic order). The simplest call is `l_tools` which displays this description for all log functions. The output can be restricted by adding a search string. For example, `l_tools create` will show all functions that create log data sets; `l_tools las` will show all functions that deal with LAS files. The search is not case-sensitive.

### l_trim

**Purpose:** This function removes leading and trailing rows from log curves if they contain null values. Null values bracketed by non-null values are retained. The function assumes that null values are represented by NaN's. If this is not the case they are replaced by NaN's in the output structure. With the minimum number of input arguments

```
    wlog=l_trim(wlog);
```

the function `l_trim` removes leading and/or trailing null values that are common to ALL curves with the exception of the depth (first column of the curve matrix) which must not have null values at all. This is equivalent to

```
    wlog=l_trim(wlog,'all');
```

This is a very benign operation as it does not remove any valid data. Of a more drastic nature is this function with the option `'any'`

```
    wlog=l_trim(wlog,'any');
```

which removes leading and/or trailing rows of the matrix of curve values if ANY of the curves contains a null value. Of course, if one of the curves is very short (say a particular log had been measured only over a reservoir interval) all other curves are shortened to this interval as well. To avoid problems with such short curves it is possible to restrict the number of curves for which the condition is evaluated. For example,

```
    wlog=l_trim(wlog,'any',{'DTp','rho'});
```

removes leading and/or trailing rows of the matrix of curve values if the sonic and/or the density log have null values. All other curves are disregarded.

As mentioned above, gaps, i. e. null values within log curves (null values preceded and followed by valid curve values), are likely to be retained. The function `l_interpolate` can be used to interpolate across such gaps. The function `l_curve` can be used to find out if any of the log curves have gaps.

### read_las_file

**Purpose:** This function reads a disk file written in LAS format and outputs a log structure. The function tries to be lenient and accept files even if they do not quite follow the LAS standard.

### show_las_header

**Purpose:** This function reads a disk file written in LAS format and outputs the header to a file or prints it to the screen.

### write_las_file

**Purpose:** This function writes a log structure to disk in LAS format.

# Chapter 4

# GENERAL TOPICS

## 4.1 Initialization Function

**presets**

**Purpose:** This function creates four global structures, `CURVES`, `CURVE_TYPES`, `TABLES`, and `S4M` which are used by many SeisLab functions. The last one, `S4M`, is of particular importance. Some fields of this structure are meant to be customized by a user. These fields are set in function `userDefaults`. Others fields of `S4M` that are less likely to need modification and all fields of `CURVES`, `CURVE_TYPES`, and `TABLES` are defined in `systemDefaults`. Both functions are called by `presets` (any field set in `systemDefaults` can be overridden in `userDefaults` — or anywhere else for this matter). Some of the key fields of `S4M` with their default settings are (`logical(1)` means "true" and `logical(0)` means "false"):

- `seismic_path` — path to the directory with seismic data. SEG-Y files are assumed to have the file extension "sgy" or "segy". If a file name is not specified when `read_segy_file` or `write_segy_file` is called a file selection window opens. The directory in which it starts is set by this variable. This may save a number of mouse clicks. Analogous fields exist for log data (usually extension "las"), mat files (extension "mat"), and table files (extension "tbl").

- `default_path` — This is a global variable with a path for files with file extensions other than `''sgy''`, `''segy''`, `''las''`, `''tbl''`, or `''mat''`.

Those defined in `systemDefaults` are:

- `script`
  Default is `''`; but if the initialization function `presets` is called from a script this field stores the name of that script.

- `alert = logical(1)`
  This parameter specifies if, in certain circumstances, messages should be printed to alert the user to certain situations or results.

- `case_sensitive = logical(0)`
  This parameter specifies whether or not seismic header mnemonics and curve mnemonics of well logs are case-sensitive; i.e. it establishes if `'CDP'` is the same trace header as `'cdp'` or if `'RHOB'` is the same curve mnemonic as `'rhob'` or `'Rhob'` (with the default setting they are).

- `compiled = logical(0)`
  This parameter specifies if one is running a compiled version of SeisLab (compiled versions of Matlab functions may be somewhat restricted in their functionality; this limitation went away with Matlab versions 7.x).

- `experience=1`
  Experience level of a user. Three values are supported: Novice: -1; User: 0; Expert: 1. Mostly used in compiled versions of SeisLab.

- `font_name = 'Arial'`
  Name of default font for plots.

- `fp_format = 'ieee'`
  Floating point format used when writing data to a SEG-Y file. Default is IEEE with big-endian byte ordering which is now part of the SEG-Y standard.

- `history = logical(1)`
  The default setting of this field is 1 (true). This means that seismic data sets have a field `history`. Each seismic function adds one line to the history field before it outputs the seismic data. This way every seismic data set has a kind of processing history attached. Seismic functions add information to the `history` field of any data set they process (unless they are too deep down in the calling sequence; this is to avoid cluttering the history field).

- `history_level = 1`
  Specifies how deep in the calling sequence a function must be so that it does not write to the history field even if `S4M.history == 1`.

- `interactive=logical(0)`
  If interactive is off (`logical(0)`) a running script or function will not stop with an interactive message to request a user action but will perform the default action. An example is `l_plot(wlog)` which plots the curves of well log `wlog`. If `interactive=logical(1)` it will pop up a listbox requesting selection of the curves to plot;
  but if `interactive=logical(0)` it will plot all the curves without asking the user.

- `mymatlab='C:\MyMatab'`
  Name of the folder with a user's Matlab files

- `name='SeisLab'`
  Name of the package; used in the title pane of figures

- `ntr_wiggle2color=101`
  Number of traces for which automatic seismic plotting (e.g. `s_plot`, `s_ispectrum`) switches

from wiggle trace to color (`s_wplot` always plots wiggles and `s_cplot` always makes color plots.

- `plot_label`
  Label for lower left corner of plots; default is `S4M.script`.

- `eps_directory=fullfile('C:\Documents and Settings', ...`
  `getenv('USERNAME'),'My Documents','My Pictures')`
  Directory used to store encapsulated PostScript files. These files are intended for use in LaTeXdocuments. They are created by clicking the "Save plot" figure-menu button and selecting the "EPS" option.

- `pp_directory=fullfile('C:\Documents and Settings', ...`
  `getenv('USERNAME'),'My Documents','My Pictures')`
  Directory used to store PowerPoint files. These files use the EMF (Enhanced Meta File) format which convert readily to Microsoft Office drawing objects and can then be edited. They are created by clicking the "Save plot" figure-menu button and selecting one of the two "EMF" options.

- `start_time`
  Set to date and time when `presets` was run.
  Executing `presets` at the beginning of a script is a good idea because it restores the default values of all global parameters and updates the time information in `S4M.start_time`. Several plot programs put this time in the lower right corner of the plot. This way all plots generated with the same script during the same run bear the same "time stamp".

- `matlab_version = 7.1`
  Version of Matlab. Some functions require different code depending on the version of Matlab used.

- `landscape`
  Default figure position and size of landscape plots on the screen.

- `portrait`
  Default figure position and size of portrait plots on the screen.

- `seismic_path` — path to the directory with seismic data. If a file name is not specified when `read_segy_file` or `write_segy_file` is called a file selector window opens. The directory in which it starts is set by this variable. This may save a number of mouse clicks. Analogous fields exist for log data (extension "log"), mat files (extension "mat"), and table files (extension "tbl").

- `log_path` — path to the directory with well data. If a file name is not specified when `read_las_file` or `write_las_file` is called a file selector window opens. The directory in which it starts is set by this variable.

- `default_path` — This is a global variable with a path for files with file extensions other than `''sgy''`, `''las''`, `''tbl''`, or `''mat''`.

The file `presets` calls functions `systemDefaults` and `userDefaults` (if it exists). The latter is meant to be customized by the user. Function `presets` should be called prior to using any of the SeisLab functions.

Hence the first lines of a script might look like this:

```
clear all
presets
```

Others global variables are more general and are set in function `systemDefaults` which is called by `presets`.

- `CURVES` — defines default curve mnemonics for log structures (see Tables 4.2, 4.1, and 4.3)

- `CURVE_TYPES` — defines types of curves (e.g. impedance) of interest for geophysical work.. This is a five-column cell array.

  1. type of curve (curve name)
  2. possible units of measurements separated by a vertical bar ("|"). Units of measurements are used for a tentative determination of the curve type (there are obviously different curves that have the same mnemonics (e.g. clay volume and water saturation or P-velocity and S-velocity).
  3. Mnemonics for curve types; these mnemonics are largely identical with the corresponding curve mnemonics (see global variable `CURVES`)
  4. Curve description; mostly identical with the curve name
  5. Indicator if a curve mnemonic is related to the one in the following row. It is 0 if it is related and 1 if it is not (this is meant to allow grouping of the curve types).

- `TABLES` — defines default mnemonics for the columns of tables.

```
coal       Logical for coal
gas_sand   Logical for gas sand
hc_sand    Logical for hydrocarbon sand
lime       Logical for limestone
oil_sand   Logical for oil sand
salt       Logical for salt
sand       Logical for sand
sh_sand    Logical for shaley sand
shale      Logical for shale
wet_sand   Logical for wet sand
```

Table 4.1: Default mnemonics for lithologies

```
aImp     Acoustic impedance
aRefl    Acoustic reflectivity
BS       Bit size
cal      Caliper
depth    Depth
drho     Density correction
DTp      Sonic log (Pressure)
DTs      Shear log
GR       Gamma ray
MD       Measured depth
OWT      One-way time
Phie     Effective porosity
Phit     Total porosity
PR       Poisson's ratio
rho      Density
Sbrine   Brine saturation
Sgas     Gas saturation
Shc      Hydrocarbon saturation
Soil     Oil saturation
TVD      True vertical depth
TVDbSD   True vertical depth below seismic datum
TWT      Two-way time
Vclay    Clay volume
Vp       Compressional velocity
Vs       Shear velocity
```

Table 4.2: Default mnemonics for log curves

```
EP     Excess pressure
EPG    Excess pressure gradient
FP     Fracture pressure
FPG    Fracture pressure gradient
OBP    Overburden pressure
OBPG   Overburden pressure gradient
PP     Pore pressure
PPG    Pore pressure gradient
```

Table 4.3: Default mnemonics for pressures

## 4.2   Input Arguments via a Global Structure

The standard argument list of a function can have positional input parameters and parameters specified via keywords. An example is

```
s_wplot(seismic,{'annotation','cdp'},{'deflection',1})   2
```

which plots the seismic data set `seismic` in wiggle-trace form with trace deflection 1 and trace annotation `CDP` — assuming `seismic` has a header `CDP`. The first input argument, `seismic`, is a positional parameter; it must be the first parameter in the argument list. The other input arguments are keyword-specified, optional parameters. These keyword-specified parameters can also be provided to a function via the global structure `PARAMETERS4FUNCTION`. Thus statement `2` is equivalent to

```
global PARAMETERS4FUNCTION
PARAMETERS4FUNCTION.s_wplot.default.annotation='cdp';
PARAMETERS4FUNCTION.s_wplot.default.deflection=1;
s_wplot(seismic)
```

It is important to note that the field `default` of `PARAMETERS4FUNCTION.s_wplot` is deleted once it has been read in the subsequent call to function `s_wplot`. Hence it cannot accidentally be used again. However, a new field, `actual`, is created. `PARAMETERS4FUNCTION.s_wplot.actual` is a structure that holds all the keyword-controlled parameters of `s_wplot` used by the last call to `s_wplot`. Hence, one can re-plot `seismic` with the same parameters by using the following two statements, assuming that `PARAMETERS4FUNCTION` has already been defined as `global`.

```
PARAMETERS4FUNCTION.s_wplot.default=PARAMETERS4FUNCTION.s_wplot.actual;
s_wplot(seismic)
```

For interactive use this approach is a bit cumbersome but it turned out to be quite convenient for use in functions controlled by a graphic user interface.

# Index