A SEG-Y file I/O toolbox for Matlab

Henry C. Bland and Paul R. MacDonald

ABSTRACT

Programs that read and write seismic data are essential in exploration geophysics. Although many file formats exist for storing seismic data, the SEG-Y format is the most widely used for transferring data between geophysical applications. Unfortunately, this file format suffers from many shortcomings: Designed in a time when computers were four orders of magnitude slower than today, the format is unsophisticated, has limited extensibility, and stores its data in a way that is foreign to the vast majority of modern computers. The format's simplicity has been the key to its longevity. Programmers with little experience are able to produce code that reads and writes files resembling the SEG-Y standard. Unfortunately, the code usually falls short of implementing the full SEG-Y standard, and many SEG-Y files remain unreadable by this code. Even worse is that many programmers, for lack of time or ability, write software that creates non-conformant SEG-Y files. Many popular geophysical applications exhibit serious flaws in their ability to read and write SEG-Y files. Since our research group writes a lot of software that performs seismic I/O, we chose to write a new, easy-to-use seismic I/O toolbox that could read and write standard-conforming SEG-Y files.

The CREWES Seismic I/O toolbox provides a comprehensive set of functions for working with SEG-Y files. In addition to basic trace reading and writing, the toolbox offers features for sorting traces, trace selection based on trace header criteria, and file-wide indexing of gathers. Most importantly, the toolbox handles many of the tedious details of the format, such as floating-point-type conversion and byte-order swapping.

The authors hope that this toolbox will greatly assist geophysics-related research in Matlab. With the aid of this toolbox, Matlab can operate on data from other geophysical applications easily and reliably. This toolbox enhances Matlab and makes it an even more optimal environment for developing new geophysical algorithms and processing techniques.

OVERVIEW

The CREWES Seismic I/O library for Matlab is called the crSeisio Toolbox (pronounced C-R-seis-I-O). The toolbox has been designed to be easy to use. A large number of examples will be shown to illustrate correct usage of the toolbox. Installation of the toolbox is beyond the scope of the paper. Readers are encouraged to view the documentation that accompanies the toolbox in the CREWES Software Release. This accompanying documentation also contains a comprehensive reference for the toolbox.

Many of the examples that follow use a mixture of upper- and lower-case letters in function and variable names. Since Matlab is not a case sensitive language, users are

free to use whichever case they choose (or both, as the authors have done). We feel that mixed-case names are more easily understood, and aid in code readability.

TUTORIAL

Reading traces from a file

Let us begin with a short example – reading a single trace of data and plotting it on the screen.

```
1   sf=crSeisioFile('test.sgy','r');
2   trcData = readTrc(sf);
3   plot(trcData);
4   close(sf);
```

Line 1 opens the file called *test.sgy* for read access. The crSeisioFile function returns a handle[1] pointing to the newly opened file. Line 2 reads a single trace of data from the file. The seismic data is returned as the array `trcData`. Line 3 plots the data in a window.

We can expand this example to read and plot all the traces in a file.

```
1   sf = crSeisioFile('test.sgy','r');
2   allTrcs=zeros(0,0);
3   oneTrc = readTrc(sf);
4   while not(isempty(oneTrc))
5         allTrcs(:,end+1) = oneTrc;
6         oneTrc = readTrc(sf);
7   end
8   [nSamp nTraces]= size(allTrcs);
9   shiftArray = (ones(nSamp,1) * [1:nTraces]);
10  plot(allTrcs + shiftArray,[1:nsamp]);
```

By adding a *while* loop, we can load all the traces in a file into the 2-dimensional array `allTrcs`. The traces are stored in `allTrcs` as columns – this is the standard for all CREWES seismic toolboxes. The loop reads through all the traces in the file until the end-of-file is reached. When that happens, `readTrc` returns an empty matrix and the loop exits. In order to spread-out the traces (so that they don't all plot on top of each other) we've added a trace-shift to the 2-D ensemble of data. We add a constant of 1 to all samples in trace one, a constant of 2 to all samples in trace two, and so on.

SEG-Y Headers

The examples to this point have ignored the presence of file and trace headers that may have been in the SEG-Y file. The term *header* refers to a collection of information about the seismic recording that is not the set of time-sample seismic amplitudes. For example, the number of samples per trace is one item stored in the headers. Each piece of information stored in a header is called a *header word*. All

[1] To be strictly correct, crSeisioFile, returns a "crSeisioFile object". Those not familiar with Matlab's object oriented programming techniques can consider the variable as a "handle".

SEG-Y files contain three kinds of headers – the text header[2] (one per file), the file data header (one per file) and trace headers (several per file). Figure 1 shows the structure of a SEG-Y file. We can see that it begins with a file header, and is followed by a number of traces. The file header contains two parts – a text part and a data part. The text part has room for 40 lines of text, with each line having a length of 80 characters. The data part contains up to 120 header words, though only a handful are typically used. Each trace also has it's own header – a trace header. Trace headers contain a number of header words whose values typically change from trace to trace. The SEG-Y standard defines the attributes of all header words: their description, their data type (2-byte integer or 4-byte integer), and their byte position within the header. Many of the header words are considered optional, and are left containing a value of zero by most programs.

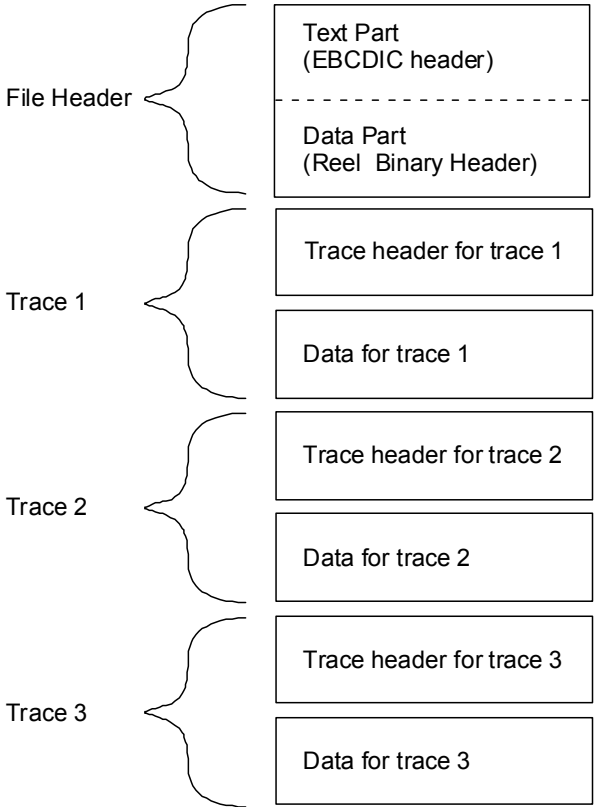

Figure 1. Structure of a SEG-Y file.

While some headers words can be left blank (full of zeros), certain header words must always be specified or the data file will be unreadable by most geophysical applications. If we only concern ourselves with reading the time-sample data from SEG-Y files, we can safely ignore the headers. If we want to know more about the file (such as the sample rate, shot-point of each trace, trace coordinates, etc) we must read both the trace data and the headers.

[2] The document that describes the SEG-Y standard uses the nomenclature "EBCDIC header" to describe the text part of the file header, and "reel binary header" to describe the data part of the file header. We'll use our nomenclature from this point on, as it easier to understand.

Reading text from the file header

Reading file headers and trace headers is made simple with the crSeisio library. Consider the following example that reads and displays the text stored in the file header.

```
1   sf = crSeisioFile('file.sgy','r');
2   fileHdr = crSeisioFileHdr;
3   readFileHdr(sf, fileHdr);
4   linesOfText = getText(fileHdr);
5   disp('SEG-Y Text header contents:');
6   disp(linesOfText);
7   close(sf);
```

Line 1 opens the file *file.sgy* for read access. Line 2 creates an empty file header object. One can consider `fileHdr` as a special type of variable that can hold an entire file header's contents. Although line 2 looks like it is a simple variable assignment, it is not – `crSeisioFileHdr` (on the right hand side of the equation) is a *object creation* function. The crSeisio toolbox currently contains four of these special functions: `crSeisioFile`, `crSeisioFileHdr`, `crSeisioTrcHdr` and `crSeisioHdrDefn`. Sometimes these functions are called with arguments (as in line 1), but the latter three are most frequently called without any arguments. It is easiest to think of line 2 as meaning "the variable `fileHdr` is going to hold a file header object."

Line 3 reads the file header from the opened SEG-Y file and places it in the `fileHdr` variable. Line 4 extracts the text from `fileHdr`, giving us a 2-D character array containing the lines of text. This array is 40 rows by 80 columns and stores each line of text as a row. Simply displaying this array (line 6) prints the text header on the screen in human readable form. On line 7, the file is closed.

Wring text to the file header

Writing text into a file header is accomplished with setText:

```
1   sf = crSeisioFile('new.sgy','w');
2   fileHdr = crSeisioFileHdr;
3   for lineNumber=1:40
4         setText(fileHdr, 'it bears repeating', lineNumber);
5   end
6   writeFileHdr(sf, fileHdr)
7   close(sf);
```

This example fills the text header with forty lines of "it bears repeating". We generally strive to create more meaningful text headers than this. You may note that line 1 opens the SEG-Y file for write access (indicated by 'w') rather than read access (indicated by 'r'). To modify an existing SEG-Y file, one can specify a read/write mode of 'm'. The `writeFileHdr` function on line 6 writes-out the file header to the SEG-Y file. This can be done at any time – even after writing several traces to the file. It should be noted that the `crSeisioFileHdr` object (the file header) is not necessarily associated with a particular file. Within Matlab one can store and retrieve text in a `crSeisioFileHdr` object without performing any file operations (though

there is no reason to do this). The only time data is exchanged between the `crSeisioFileHdr` object and the `crSeisioFile` object is when `readFileHdr` or `writeFileHdr` is called.

Reading headers words from the file header

To read header word values from the data part of the file header, we call the function `get(fileHdr,`*`headerName`*`)` supplying the name of the desired header word as the second argument. This is illustrated in the following example:

```
1   sf = crSeisioFile('test.sgy','r');
2   fileHdr = crSeisioFileHdr;
3   readFileHdr(sf,fileHdr);
4   nSamp = get(fileHdr,'NSAMP');
5   disp(['The file header says, NSAMP=' num2str(nSamp)]);
6   close(sf);

    The file header says NSAMP=2001
```

On line 4 we get the value of a specific header word – in this example, NSAMP, the number of samples per trace. The names of the header words come from a standard list contained within the crSeisio library. Table 1 lists all these header word names. We can also obtain these names using the `getHdrNames` function:

```
1   fileHdr = crSeisioFileHdr;
2   names = getHdrNames(fileHdr)

    names =

    JOB_ID
    SEISLINE
    REAL_NO
    …
```

The previous example gets an array containing the header names that have been defined for the file header.

We can easily determine the values of all defined header-words stored within the data part of the file header as follows:

```
1   sf = crSeisioFile('file.sgy','r');
2   fileHdr = crSeisioFileHdr;
3   names = getHdrNames(fileHdr);
4   readFileHdr(sf, fileHdr);
5   close(sf);
6   values = get(fileHdr,names);
7   for i=1:size(names,1)
8      fprintf(1,'%-14s: %14s\n',names(i,:), num2str(values(i)));
9   end

    JOB_ID        :      98141
    SEISLINE      :         12
    REAL_NO       :          1
    …
```

Line 3 calls `getHdrNames` to obtain a vector of header names (`names`). In line 6 we pass an array of header names to `get(fileHdr…)` rather than just a single name. The result is an array of values. Each element in `values` has a corresponding element in `names` at the same index. Lines 7 through 9 simply print these two vectors side-by-side separated by a colon. Using this technique, we can get a complete dump of the header words defined in the file header.

Writing header words to the file header

When creating a SEG-Y file from scratch, a number of header words should be specified. In particular these include the number of samples per trace (`NSAMP`), the sample interval (`SAMPINT`), and the data format (`DATAFMT`). The following example shows how this can be done:

```
1   sf = crSeisioFile('new.sgy','w');
2   fileHdr = crSeisioFileHdr;
3   put(fileHdr,'NSAMP',2501);
4   put(fileHdr,'SAMPINT',2000);
5   put(fileHdr,'DATAFMT',1);
6   writeFileHdr(sf,fileHdr);
7   % add code here to write traces
8   close(sf);
```

This example is specifies that there are 2501 samples at a 2ms sample interval (5 seconds of data). The data sample format is set to 1, which corresponds with the default data format (4-byte IBM floating point).

Trace headers

Our discussion so far has been of the SEG-Y file header. In reality, the most interesting header words are stored in the trace headers. Well-behaved SEG-Y files store information such as the shot number, receiver number, receiver coordinates, and shot coordinates within the trace header. The following example shows how one might read various trace header values. It prints the field file ID (shot number) and offset for each trace in the file.

```
1   sf = crSeisioFile('file.sgy','r');
2   trcHdr = crSeisioTrcHdr;
3   trcData = readTrc(sf, trcHdr)
4   trcNo = 0;
5   while not(isempty(trcData))
6         ffid = get(trcHdr,'FFID');
7         offset = get(trcHdr,'OFFSET');
8         trcNo = trcNo + 1
8         fprintf(1,'TRC=%d FFID=%d OFFSET=%f',trcNo,ffid,offset);
9         trcData = readTrc(sf, trcHdr);
10  end
11  close(sf);
```

```
TRCNO=1  FFID=100 OFFSET=-240
TRCNO=2  FFID=100 OFFSET=-220
TRCNO=3  FFID=100 OFFSET=-200
…
```

We use the `get` function extract a value of a named header word from the trace header `trcHdr`.

Just as before, in the file header example, we can get the list of header words in the trace header using `getHdrNames`. We can then get a vector of values for all the header names, and print them as before:

```
1   sf = crSeisioFile('file.sgy','r');
2   trcHdr = crSeisioTrcHdr;
3   names = getHdrNames(trcHdr);
4   trcData = readTrc(sf, trcHdr);
5   values = get(trcHdr,names);
6   for i=1:size(names,1)
7      fprintf(1,'%30s: %12.2f\n',names(i,:), values(i));
8   end
9   close(sf);

    LINETRCNO        :   1
    FILTRCNO         :   1
    FFID             :  17
    CHAN             :   1
    …
```

The values can be placed in the trace header using `put(`*`trcHdr`*`,` *`headerName`*`)`. Upcoming examples will illustrate the use of this function.

Processing from one file to another

We will now try and write our first processing algorithm – a DC removal filter. This can be achieved by subtracting the mean value from each trace in a file. Rather than hard-code the input file and output file, we will use the Matlab function `uigetfile`. This function prompts the user with a file selection box. The selected file is then returned as a filename and path (directory name). We combine the filename and path so that `crSeisioFile` can locate the file from any directory on the system.

Unlike many of our previous examples, we don't need to access the contents of any of the headers. Since we're only changing the data, we can simply copy the file header and trace headers from the input file to the output file.

```
1   [filename, path]=uigetfile('*.sgy', 'Select input file');
2   inputFileName = [path filename];
3   [filename, path]=uiputfile('*.sgy', 'Save to file');
4   outputFileName = [path filename];
5   sfIn = crSeisioFile(inputFileName, 'r');
6   sfOut = crSeisioFile(outputFileName, 'w');
7   fileHdr = crSeisioFileHdr
8   readFileHdr(sfIn, fileHdr);
9   writeFileHdr(sfOut, fileHdr);
```

```
10   trcHdr = crSeisioTrcHdr;
11   trcData = readTrc(sfIn, trcHdr);
12   while not(isempty(trcData))
13         trcData = trcData - mean(trcData);
14         writeTrc(sfOut, trcData, trcHdr);
15         trcData = readTrc(sfIn, trcHdr);
16   end
17   close(sfIn);
18   close(sfOut);
```

Lines 1 to 4 determine the input file name and output file name. Line 5 opens a SEG-Y file for read access (note the `'r'` for *read* as the second argument). Line 6 opens a new SEG-Y file for *write* access (note the `'w'`). Lines 7, 8, and 9 create a file header, fill it with the contents of the old file, and copy it to the new file. Line 10 creates a trace header – as we read each trace, the contents of the trace header will be copied across to the new file without any modification. Line 12 checks for the end-of-file, and stops the loop when one is reached. Line 13 performs our mathematical operation – removal of the D.C. component. Line 14 writes the trace to the output SEG-Y file (indicated by the `sfOut`). Lines 17 and 18 close the two SEG-Y files thereby freeing up system resources and flushing the contents of the SEG-Y file to disk.

Creating a new SEG-Y file from scratch

When copying an existing file using the crSeisio toolbox, we don't need to concern ourselves with populating the file and trace headers – we just copy the existing headers and hope that the creator of the original SEG-Y file provided sufficient information in the headers. If we write our a brand new SEG-Y file, it is important to set a small number of headers to a reasonable value:

```
 1   sf=crSeisioFile('new.sgy','w');
 2   fh = crSeisioFileHdr;
 3   th = crSeisioTrcHdr;
 4   nSamp = 2001;                  % 2001 samples per trace
 5   sampleInterval = 2000;         % 2000us = 2ms sample interval
 5   put(fh,'NSAMP',nSamp);
 6   put(fh,'DATAFMT',1);           % 1 = 4-byte floating point(IBM)
 7   put(fh,'SAMPINT',sampInt);
 8   put(th,'NSAMP',nSamp);
 9   put(th,'SAMPINT',sampInt);
10   put(th,'TRC_ID',1);            % 1 = production data
11   writeFileHdr(sf, fh);
12   trcData = sin([1:nSamp]/100);
13   writeTrc(sf,th,trcData);
14   close(sf);
```

The example above creates a file with a single trace – containing a sine-wave signal. It should be readable by most geophysical applications. Unless `setDataFmt` is called, crSeisio toolbox creates files with data stored as 4-byte floating point numbers using the SEG-Y standard encoding (based on old IBM mainframes). This data format is indicated in the SEG-Y file's header by setting the DATAFMT header to 1.

Reading traces out of order

We can read the traces out of order, by specifying a trace number parameter to the `readTrc` and `writeTrc` functions. The first trace of a file is always trace number 1. The following example reverses the order of all traces in a file:

```
 1   sfIn = crSeisioFile('input.sgy','r');
 2   sfOut = crSeisioFile('output.sgy','w');
 3   trcHdr = crSeisioTrcHdr;
 3   numTrcsIn = getNumTrcs(sfIn);
 4   trcNumIn = numTrcsIn;
 5   for trcNumOut = numTrcsIn:-1:1
 6         trcData = readTrc(sfIn,trcHdr,trcNumIn);
 8         writeTrc(sfOut,trcData,trcHdr,trcNumOut);
 9         trcNumIn = trcNumIn - 1
10   end
11   close(sfIn);
12   close(sfOut);
```

Here, we make use of the `getNumTrcs` function to obtain the total number of traces in the file. We read the input file from back-to-front, and write to the output file from front-to-back.

Reading traces in sorted order

One of the more powerful features of the crSeisio toolbox is its ability to sort traces as they are read from a SEG-Y file. For example, if a SEG-Y file contains data shot out-of-order (which is typical in 3-D surveys), it is easy to re-order based on trace header word values:

```
1   sf = crSeisioFile('file.sgy','r');
2   sf = crSeisioSort(sf,'SRC_X');
3   sf = crSeisioSort(sf,'REC_X');
4   trcData = readTrc(sf);
```

The above example first sorts by source-X coordinate in ascending order. If any traces share the same source-X coordinate, they are further sorted by receiver-X coordinate. By default, the sorting is performed in ascending order. In order to sort in descending order, one needs add the extra argument 'descending'.

Any trace operations that follow the crSeisioSort call now operate on the traces in their sorted order. If a trace number is explicitly specified to the `readTrc` or `writeTrc` function, the trace number must now be taken in the context of the sort. For example, if we sort by descending receiver elevation, and read trace number 1, this will correspond to the trace with the highest elevation, regardless of that trace's position within the input file.

Selecting subsets of traces for input

Consider the case where we want to obtain a single receiver gather from a set of shot-gathers. If we know the X-coordinate of the receiver, we can build the receiver gather using the select function:

```
 1   sf=crSeisioFile('input.sgy','r');
 2   sf=select(sf, 'REC_X', '=', 260);
 3   recGather=[];
 4   trcData = readTrc(sf);
 5   while not(isempty(trcData))
 6         recGather(:,end+1) = trcData;
 7         trcData = readTrc(sf);
 8   end
```

The `select` function, used on line 2, supports a number of comparison operators: <, >, =, >=, <=. This allows one to make expressions like this:

CDP <= 120 Read all CDPs less or equal to 120
SHOT > 12 Read all SHOTSs greater than 12

Calling `select` multiple times with different selection expressions has a cumulative effect. Only those traces that satisfy all selection criteria are made available to `readTrc`. Processing of all *sort* and *select* operations occurs at the first call to `readTrc` or `writeTrc`. Since select operations are performed in the order of the `select` calls, it is best to place the most restrictive selection firsts to maximize efficiency. Consider the following example:

```
sf=select(sf, 'CDP' , '>', 12);
sf=select(sf, 'OFFSET', '>=', 20);
sf=select(sf, 'OFFSET', '<=', 200);
sf=sort(sf, 'CDP');
sf=sort(sf, 'REC_X');
```

In the example above, only traces with a CDP greater than 12, and an offset between 20 and 200 are selected. All traces that pass the selection criteria are sorted by CDP.

Customizing header word locations

The SEG-Y standard allows for some flexibility in the assignment of trace header words. There is room in each trace header for about 15 user-definable header words. It is also common to reassign certain header words in non-user-definable portions of the trace header. Though this makes the file non-compliant with the standard, it is common practice. Fortunately, the crSeisio toolbox was designed to handle these situations. Consider the case where we want to read the job-id from a non-standard location in the file data header (byte 31). At the same time, we want to read the X and Y-coordinates for CDPs from bytes 181-184 and 185-188 of the trace header. Here is how this is done:

```
 1   hdrDefn = crSeisioHdrDefn
 2   loadDefaults(hdrDefn);
 3   set(hdrDefn, 'JOB_ID', 'JOB ID', 'FILE', 'I4', 31);
 4   set(hdrDefn, 'CDP_X', 'CDP X coord UTM', 'TRC', 'I4', 181);
 5   set(hdrDefn, 'CDP_Y', 'CDP Y coord UTM', 'TRC', 'I4', 185);
 6   fileHdr = crSeisioFileHdr(hdrDefn);
 7   trcHdr = crSeisioTrcHdr(hdrDefn);
 8   sf = crSeisioFile('input.sgy','r');
 9   readFileHdr(sf, fileHdr);
10   jobId = get(fileHdr, 'JOB_ID');
```

```
11   disp(['job id = ' num2str(jobId)]);
12   trcData = readTrc(sf, trcHdr);
13   cdpX = get(trcHdr, 'CDP_X');
14   cdpY = get(trcHdr, 'CDP_Y');
15   disp(['CDP_X=' num2str(cdpX)]);
16   disp(['CDP_Y=' num2str(cdpY)]);
17   close(sf);


     job id = 9281
     CDP_X = 841621
     CDP_Y = 5610921
```

Until now, our examples have called `crSeisioFileHdr` and `crSeisioTrcHdr` without supplying any arguments. In the above example we specify our own header definition rather than use the default one. We create the definition on line 1 and populate it with the default header words on line 2. Line 3 re-defines the location of JOB_ID to byte 31 of the file header (the default is byte 1). Lines 4 and 5 define new trace header words that are not part of the default set of header words (table 1). Now, when we create a file header and trace header objects (lines 5 and 6), they will use our customized header definitions rather than the system default.

To allow programs to customize the header definitions through an external text file, rather than setting header definitions with Matlab code, we have provided the `readFromFile` function:

```
1   hdrDefn = crSeisioHdrDefn;
2   loadDefaults(hdrDefn);
3   readFromFile(hdrDefn, 'test.def');
```

This example creates an empty header definition on line 1 and populates it with header definitions from the system defaults on line 2. On line 3 we read additional (or replacement) header definitions from a file called *test.def*. Both the `loadDefaults` and `readFromFile` functions add definitions to a header definition object cumulatively.

Header definitions can be written to a text file using the `writeToFile` function, as in the following example:

```
1   hdrDefn = crSeisioHdrDefn;
2   loadDefaults(hdrDefn);
3   readFromFile(hdrDefn, 'test.def');
4   writeToFile(hdrDefn, 'defaults_plus_test.def');
```

The header definition files contain five columns of text separated by at least one space or tab. Here is an example:

```
     JOB_ID_HDR  F I4   1  Job identification number
     SEISLINE    F I4   5  Line number
     REAL_NO     F I4   9  Reel number
     FFID        T I4   9  Original field file number
     NUMSMP      T I2 114  Number of samples per trace
```

This list of header words is just a small sample of the entire list of defaults. The first column is the header name to be used when calling `get(hdrDefn,hdrName)` and `put(hdrDefn,hdrName,…)`. The second column contains the header type – either "F" for file header or "T" for trace header. The third column contains the format of the header. The most common format is 4-byte integer (I4), but header words can also be 2-byte integers (I2) or 4-byte floating-point numbers (F4IBM or F4IEEE). The last column contains the description of the header word.

Reading a whole file at a time

Previous examples have always read traces one at a time. Though simple, this method is far from efficient. The following example shows how we can read a whole file of trace data and trace headers at a time:

```
1   sf = crSeisioFile('input.sgy','r');
2   sort(sf,'CDP','descending');
3   [trcData, trcHdrs] = readAllTrcs(sf);
4   close(sf);
```

Here `trcData` is a 2-dimensional matrix, with each trace in a different column. The `trcHdrs` variable contains vector of trace header objects, with one element per trace. Although this example sorts the traces before reading them, this is not a requirement.

The array of trace headers can be used to find the header words associated with the all the traces. The next example illustrates how this is done.

```
 1   sf = crSeisioFile('test.sgy','r');
 2   [trcData, trcHdrs] = readAllTrcs(sf);
 3   close(sf);
 4   [nSamp nTraces]= size(trcData);
 5   shiftArray = (ones(nSamp,1) * [1:nTraces]);
 6   plot(trcData + shiftArray,[1:nTraces]);
 7   values = get(trcHdrs,'CDP');
 8   set(gca,'xtickmode','manual');
 9   set(gca,'xtick',[1:nTraces]);
10   set(gca,'xticklabel',values);
11   set(gca,'ydir','reverse');
```

The previous example illustrates how we can obtain a vector of trace header values given a vector of trace headers (line 7). Each element in `values` contains a CDP value corresponding to each trace (column) in `trcData`. Using Matlab's built-in `plot` function, we can plot the traces, and apply the CDP as tick labels under each trace. Line 7 sets the tickmode to 'manual', allowing us to specify our own tick locations along the X-axis. Line 8 creates tick marks along the zero-line of each trace. Line 9 applies the CDP values as the tick labels. Line 10 reverses the direction of the Y-axis, making time-zero at the top. The resulting plot is shown in figure 2.

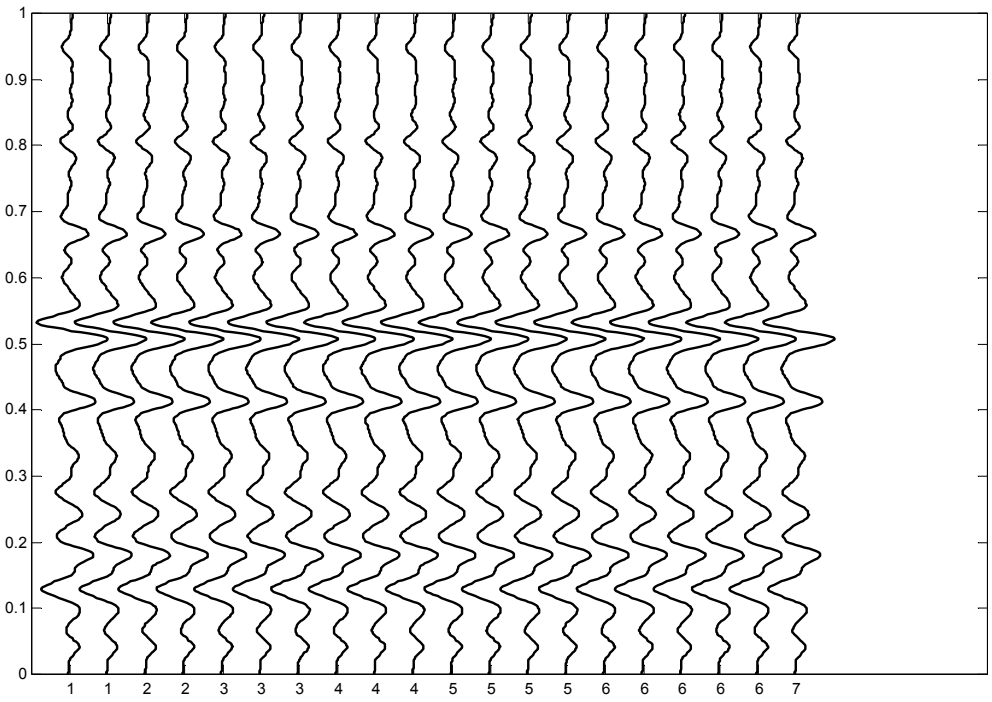

Figure 2. Synthetic seismic data plot in Matlab with CDP numbers shown as tick labels along the X-axis.

As seen in the final example, the combination of Matlab and the crSeisio toolbox makes it very simple to accomplish a great deal of work in only a few lines of code.

THE CREWES SEISMIC TOOLBOXES

The crSeisio is only one of the CREWES Toolboxes. These toolboxes contain hundreds of functions that perform common seismic operations; including much more sophisticated functions for seismic data plotting. Readers are urged to investigate these other toolboxes to make the most of their Matlab experience.

CONCLUSION

The CREWES Seismic I/O toolbox is a comprehensive collection of functions that makes it easy to read and write SEG-Y files. It provides both high-level functions (such as data sorting) as well as low-level functions (such as custom header definitions).

The crSeisio toolbox is based on a low-level SEG-Y library written in the C language. The Matlab interface to this library has been completed, and a Java interface is in the final stages of development. It is our hope to provide this library to users of other languages, such as C and Fortran, in the near future.

Table 1. Common default header word definitions. File headers are indicated by header type *F*. Trace headers are indicated by header type *T*.

Header Name	Description	Header Type	Header Format	Byte Pos.
JOB_ID	Job identification number	F	I4	1
SEISLINE	Line number	F	I4	5
REEL_NO	Reel number	F	I4	9
NTRACES	Number of traces per record	F	I2	13
NAUXTRACES	Number of auxiliary traces per record	F	I2	15
SAMPINT	Sample Interval (us)	F	I2	17
NSAMP	Number of samples per trace	F	I2	21
DATAFMT	Data sample format	F	I2	25
CDP_FOLD	CDP fold	F	I2	27
XYUNITS	Measurement system	F	I2	55
LINTRCNO	Trace sequence number within line	T	I4	1
FILTRCNO	Trace sequence number within file	T	I4	5
FFID	Original field record number	T	I4	9
CHAN	Trace number within original field record	T	I4	13
SRC_NO	Energy source point number	T	I4	17
TRC_ID	Trace identification code	T	I2	29
SHOTFOLD	Number of horizontally stacked traces	T	I2	33
DATAUSE	Data use	T	I2	35
OFFSET	Distance from source point to receiver group	T	I4	37
REC_ELEV	Receiver group elevation	T	I4	41
SRC_ELEV	Surface elevation at source	T	I4	45
XYSCALER	Scalar to be applied to all coordinates	T	I2	71
SRC_X	Source coordinate - X	T	I4	73
SRC_Y	Source coordinate - Y	T	I4	77
REC_X	Receiver coordinate - X	T	I4	81
REC_Y	Receiver coordinate - Y	T	I4	85
XYDOMAIN	Coordinate units	T	I2	89
NSAMP	Number of samples in trace	T	I2	115
SAMPINT	Sample interval (us)	T	I2	117
YEAR	Year data recorded	T	I2	157
DAY	Day data recorded	T	I2	159
HOUR	Hour data recorded (24hr clock)	T	I2	161
MIN	Minute data recorded	T	I2	163
SEC	Second data recorded	T	I2	165