Seismic Data Input/Output

Jonathan M. Lees
University of North Carolina, Chapel Hill
Department of Geological Sciences
CB #3315, Mitchell Hall
Chapel Hill, NC 27599-3315
email: jonathan.lees@unc.edu
ph: (919) 962-1562

July 13, 2012

1 Seismic Data I/O

One of the big problems with seismic data is format and exchange. Unfortunately, seismologists spend an inordinate amount of time writing codes to reformat data so that it conforms with one or anotehr programs that are commonly used. Even though there are standard formats defined and in use today, many times these standards are not adhered to. In many circumstances the original definitions were too restrictive and investigators chose to extend the format in one way or another, making the standard "non-standard". A case in point is the SEGY standard and the PASSCAL-SEGY modification.

Another problem with exchange of seismic data is platform compatibility. To get a good binary format that is compatible on MAC, Windows and Linux systems is apparently difficult. This is further complicated by differences in CPU models (e.g. 64 bit versus 32 bit) and other compiler issues. I discovered some years ago that on some systems a "long int" is misnamed and is actually defined as a "short" This can cause havoc when reading in binary format data.

A few (somewhat) standard data formats can be read in directly in **REIS** . These are `SAC` and `SEGY` as defined by PASSCAL-distributed software. I have not written an **R** function for reading `SEED` format, but it is probably not too difficult. Maybe in the future.

I am currently developing a new package called `TELES` aimed at analysis of teleseismic data extracted from the IRIS DMC web site. The code has tau-p code for predicting global travel times. This work is still in progress. `TELES` currently works in LINUX and MAC environments and can be obtained by contacting me directly.

1.1 SAC format

SAC format data can now be read directly using native **R** binary codes. Earlier I/O functions in package `SACR` relied on C-code for the binary input, and this lead to some problems when transferring data across platforms.

The basic code for I/O on SAC data is:

```
j1 = JSAC.seis(f1, Iendian=1, HEADONLY=TRUE , BIGLONG=FALSE, PLOT=FALSE)
```

This is a short explanation of the arguments to `JSAC.seis`.

f1 vector of file names to be extracted and converted

Iendian Endian-ness of the data: 1,2,3: ”little”, ”big”, ”swap”

HEADONLY logical, TRUE= header information only

BIGLONG logical, TRUE=long=8 bytes

PLOT logical, whether to plot the data after reading in

Here *f1* is the path to one, or many, file names on the local system. When HEADONLY=TRUE only the SAC header is returned, and this can be used to set up the input of large digital signal files. The other arguments are important for making **REIS** platform independent. Argument *Iendian* is critical if the data were created on one platform transferred and read in on another. This argument refers to the “endian-ness” (byte order) of memory in the computer. In **R** one can find out the “endian-ness” of the system by accessing the variable

```
print(.Platform$endian)
```

```
[1] "little"
```

If data is created on the same system on which it is analyzed, and you stay consistent, there should be no problem. The problem of compatibility arises when data is shared across platforms. If you know what the endian-ness of the data is from the platform where the data was written in binary format and it is different than your system, use “swap”. Else, stay consistent. My desktop Linux machine and my laptop MAC are both “little-endian”. My older SUN computers were “big-endian”.

The *BIGLONG* argument was introduced because the SAC header has both long and short integer numbers. The issue stems from the fact that many systems (32 bit) do not recognize the LONG definition and internally convert to short, i.e. long is defined as 4 bytes. This can create a problem when transferring data created on a 64 bit machine to a 32 bit machine, and vice versa. So, if the format of the source machine is known - use that for the *BIGLONG* argument to indicate how to treat LONG ints.

1.2 SEGY format

SEGY formatted data follow the same convention that SAC data do, except that there is slightly different information in the header.

1.3 WIN format

There is a routine for reading `WIN` format from Japan, in a separate package called `WINR`. These codes were written in C, actually converted from the original FORTRAN code. They are not platform independent and they require re-compilation when converting from Windows to Linux types of systems. While they work well on my Linux system, I have had trouble getting them to work on different systems when the endian-ness is changing and the BIGLONG problems arise. You can try to use these, but I recommend simply converting WIN format to some native **R** format and reading the files in **REIS** .

1.4 UW format

There are many routines in **REIS** for handling UW format seismic data. UW format comes from the University of Washington and is used for earthquake event data. In that case many traces are stored for each event, arrival time information is stored in a pickfile, as well as polarities. Event location and focal mechanism solutions are also gathered and saved in the RSEIS list. See package **RFOC** for instructions on how to plot and manipulate focal mechanisms.

1.5 IRIS DMC data

1.5.1 Teleseismic data

1.6 REIS format

One way to store data is in native **REIS** format. In this case one might read in the data in one of the previous formats and follow with a save to a binary **R** file on the local system. Then consequent I/O is simply a load command in **R** . I use this method when I have isolated a specific section of data that I am working on and need to read it for different purposes on different platforms, or share it with others.

As an example, suppose I have isolated a set of date/times that have events of interest. The event times, or windows, are stored in a list of $day, hr, s1, s2$ where $s1$ and $s2$ are starting and ending seconds for the event.

A database (DB, see **??**) has been created earlier that describes the location of the SEGY files and their content. I use RSEIS program *Mine.seis* to extract the selected time window from the full data set. Here is snippet of code:

```
for(i in 1:length(chugs$day))
{

  print(i)

  at1 = chugs$day[i]+chugs$hr[i]/24 + chugs$s1[i]/(24*3600)

  if(chugs$s2[i]>3600) {
    at2  = chugs$day[i]+(chugs$hr[i]+1)/24 + (chugs$s2[i]-3600)/(24*3600)

  }
  else
    {
      at2  = chugs$day[i]+chugs$hr[i]/24 + chugs$s2[i]/(24*3600)
    }

  CH = Mine.seis(at1, at2, DB, usta, ucomp)

  fnsave =  paste(sep=".", Zdate(CH$info, sel=1, t=0), "RCHUGseis")
  print(paste(sep=" ", "Working on",fnsave))
  save(file=fnsave, CH)
  ##  sbut = swig(CH, sel=which(CH$STNS=="CAL") )

}
```

The Mine.seis call extracts the data from the database and the data is saved in the file *fnsave* with the **REIS** list named "CH".

In the future this data can be recalled in **REIS** by loading. Here that operation is put in a loop that breaks when the `QUIT` button is clicked in **swig**

```
for(i in 1:length(LCHUG ))
{

  load(LCHUG[i])
  sbut = swig(CH, sel=which(CH$STNS=="CAL" & CH$COMPS %in% c(VNE, IJK[c(1,2)] ) ) )
  if(sbut$but=="QUIT") { break }

}
```

Data stored in this format can be shared with others using **REIS** (or other **R**) software. The advantage is that the data will work on any platform (Linux, MAC or Windows) seamlessly.

1.7 ASCII format

Data may be stored in simple ASCII format and read in to **R** . To use **swig** , however, a proper list should be created. In this section I will present an example illustraing how to create the appropriate list for input into swig.

Suppose I have a data set consisting of seismic, infrasound and gravity recordings stored in 3 different files on disc. First the data is loaded into R via any means available (scan, read.table, load, etc...).

Here, create two time series using ricker wavelets and combine them together for analysis in swig:

```
freq1=1/50
dt1=1/100
nw1= 300/dt1
g1 = genrick(freq1, dt1, nw1)
date1  = recdate(45, 11, 11, 4, yr=2011)
sig1  = prep1wig(wig = g1, dt = dt1, sta = "STA1", comp = "CMP",
    units = "BLAH", starttime =date1 )
freq2=1/300
dt2=1/100
nw2= 100/dt2
g2 = genrick(freq2, dt2, nw2)
date2  = recdate(45, 11, 11, 4+100, yr=2011)
sig2  = prep1wig(wig = g2, dt = dt2, sta = "STA2", comp = "CMP",
    units = "BLAH", starttime =date2 )
```

Combine the wiggles into one list, and prepare for swig:

```
SIG = list(sig1=sig1[[1]], sig2=sig2[[1]])
EH=prepSEIS(SIG)
```

Now they are ready for plotting:

```
swig(EH, SHOWONLY = TRUE)
```

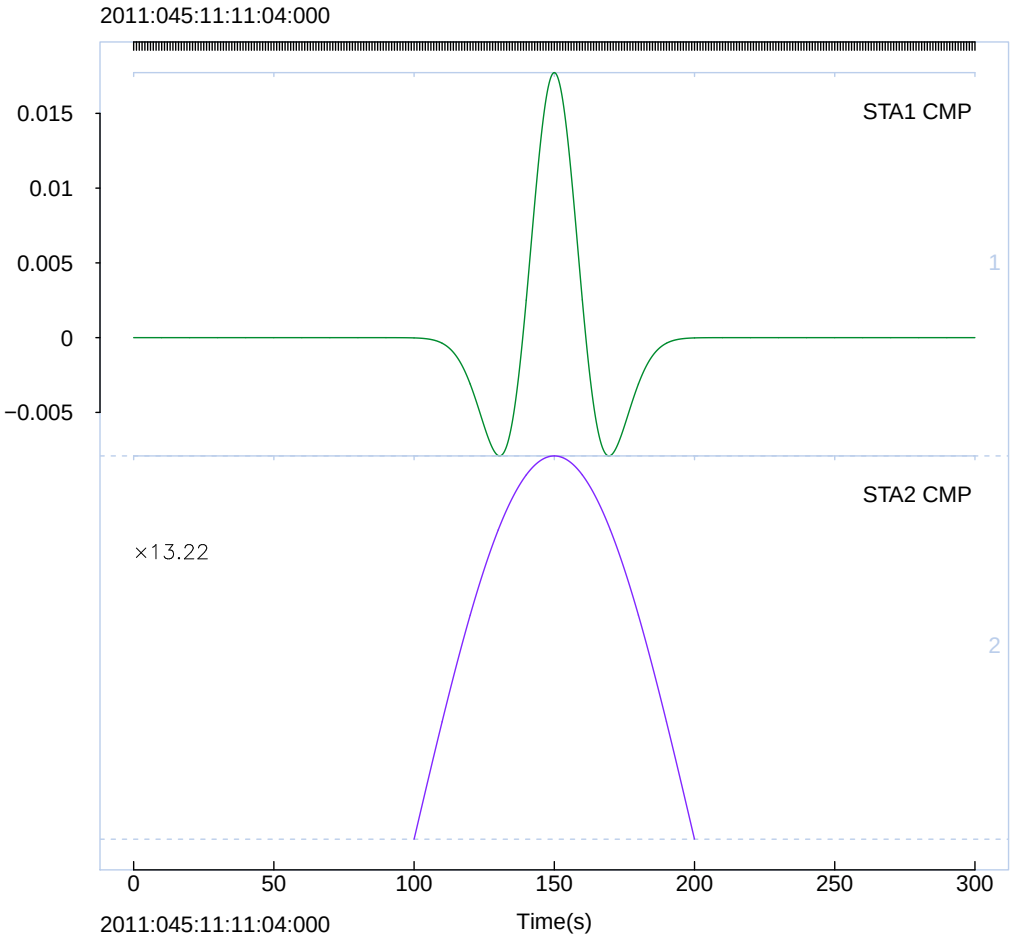

Figure 1: PLot showing swig window with signals 1 and 2.

2 MakeDB

In this document I will illustrate how to create a simple flat database for use in **REIS** . The data base is constructed from files, usually SEGY or SAC, but they could be native **R** files already processed so that conversion is not necessary (better still).

3 File Structure

The basic structure for this code is based on the output of a program written by PASSCAL called ref2segy (or ref2sac). After extracting data from disks in the field the "ref" files are dumped into a directory on the hard drive of a computer. The program ref2segy extracts the data from messy reftek format, and converts them to SEGY format. A log is created and other output useful for getting information about the field operations. For now we do not need to pay attention.

As an example, the files for my 2009 santiaguito experiment in Guatemala are stored on my computer as:

```
wegener% ls
/Users/lees/Site/Santiaguito/SG09
#########################
segyDB    R365.02/                2009:019:15:39.9026.log  2009:007:17:11.run
filesDB   R366.02/                2009:019:15:39.run       2009:007:17:11.SMI.log
R001.02/  R006.02/                2009:007:17:12.CAL.log   2009:007:17:11.KAM.log
R002.02/  2009:019:15:40.9024.err 2009:007:17:12.run       2009:007:17:10.KAM.log
R003.02/  2009:019:15:40.9024.log 2009:007:17:12.DOM.log   2009:007:17:10.run
R004.02/  2009:019:15:40.run      2009:007:17:11.DOM.err   2009:007:17:10.CAL.err
R005.02/  2009:019:15:39.9026.err 2009:007:17:11.DOM.log   2009:007:17:10.CAL.log
```

The actual waveform files are in the directories starting with "R00" etc. The Julian day is on each folder name. This data was recorded in late December, 2008 and into January 2009. so the high julian days are at the beginning of the experiment and the low day numbers at the end. (The spanning of the new year actually presents some date problems that need to be overcome.)

For example, a listing of one of the subdirectories is:

```
wegener% ls
/Users/lees/Site/Santiaguito/SG09/R002.02
```

```
########################
09.002.00.47.50.CAS.I
09.002.00.47.50.CAS.J
09.002.00.47.50.CAS.K
09.002.01.47.50.CAS.I
```

Note that the files have already been altered, in that information from the headers has been placed in the file names. This is not critical, but it is important to get information about the station name and component into the file headers.

4 makeDB

The program *makeDB* will read in the data once its is told where to look, what to look for and what format to use.

The call uses a path and pattern to read in the data, file by file and store the header (and other) information for quick access. The path variable is a pointer to the base location of the data to be extracted. The pattern argument is used to direct the the program to read in some information and ignore extraneous folders or files. In this example, all the data is stored in directories starting with "R0" so the pattern is simple. We use a wild card to get all the folders for this experiment.

```
path = '/Users/lees/Site/Santiaguito/SG09'
pattern = "R0*"
XDB  =  makeDB(path, pattern, kind =1)
```

The other parameters are critical and care must be taken to make sure they are executed correctly. The default parameters are:

- kind = 1
- Iendian = 1,
- BIGLONG = FALSE

4.1 kind

The *kind* argument signals **REIS** that the data is a specific format. The standards now are

- 1=SEGY
- 2=SAC
- 0=native RSEIS

The program reads in each file, extracts station name, component, sample rate and timing information from the files and saves these in a list. The SEGY and SAC formats are read in using native **R** binary read commands. If the data is already in **REIS** format, then the processing uses **R** command *load* to read the data in and extract from the list already available.

The two other arguments relate to the format that the data is stored in and depend on the computer system they are read on.

4.2 Iendian

Iendian is a flag indicating the endian-ness of the data and whether swapping needs to be performed. The byte-order ("endian-ness") is different for different operating systems. You can determine the endianness of a system by accessing the **R** .Platform variable,

```
> .Platform$endian
```

```
[1] "little"
```

If the data was written on a little endian machine, then the little option should be provided. Likewise id big endian was used to create the data, and the machine reading it is also big-endian, then use big. If the machine writing the data and the machine reading the data use different conventions, then "swap" should be invoked.

```
endian: The endian-ness ('"big"' or '"little"' of the target system
          for the file.  Using '"swap"' will force swapping
          endian-ness.
```

4.3 BIGLONG

The BIGLONG variable is set so that data written with long=8 on a machine with long=4 can be accomodated. This problem arises mainly with SAC format data as the header for SAC data

calls for a long, even though most 32 bit machines actually use long=short. To determine what a machine is using one can query the .Machine variable in **R** :

```
> .Machine$sizeof.long
```

```
[1] 8
```

If the size is 8 use TRUE, if 4 use FALSE.

If you get your data from someone else, or you download the binary files, you need to determine how to set these parameters. Having the wrong arguments may lead to **R** crashing, or even crashing the whole system.

5 Extracting Data

The purpose of makeDB is to allow quick and easy access to the data files and to make it easy to extract time slices from the large set of files.

The **REIS** program I use for small data sets like the one illustrated above is called *Mine.seis*.

With *Mine.seis* you give it a the database and it finds the files that need to be accessed, extracts the waveforms and glues the files together to get a single trace for each station/component. This list is suitable for plotting and processing in swig. (swig=seismic wiggle).