RSEIS: Seismic Time Series Analysis in **R**

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

1 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).

2 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
/home/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
/home/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.

3 I/O in REIS

The I/O routines in **REIS** read in binary data from disk and store the data in lists for processing. The data is generally organized for manipulation of earthquake seismic data, but there is no specific restriction.

The routimes for extracting data in SEGY and SAC format use the binary I/O native to **R** , specifically *readBin*. It is important for the user to know which conventions were used on the machine that created the binary files as well as the conventions used on the machine reading the binary files. Care must be taken to correctly associate the “endianness” and “size of LONG” integers on the data creating and the data reading systems. See below on how to accomodate the differences in these systems.

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 = '/home/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. Attention must be paid to the system where the binary data was created and the system that is trying to read the binary data. (If the data are in RSEIS native format this does not apply.) Without proper handling, the I/O codes for reading the data will not work and may cause the system to crash.

4.2 Iendian

Iendian is a flag indicating the endian-ness of the data and whether swapping needs to be performed.

The following definition and example are taken from: `http://cplus.about.com/od/glossar1/g/Endian.htm`

Definition: Endian-ness refers to the layout of (usually) bits in CPUs and affects the layout of multi byte numbers, particularly integers. There are two schemes that really matter- big endian and little endian. PCs for example are little endian (which means little end first). Motorola processors used in Macs (until the switch to Intel processors) were big endian.

For example, the decimal int 56789652 is 0x03628a94 in hexadecimal. On a big endian PC the 4 bytes in memory at address 0x18000 start with the leftmost hex digit. Each 8 bit memory location holds one hexadecimal number in the range 0x00 to 0xxFF.

```
 Big Endian
 18000 18001 18002 18003
  0x03 0x62 0x8a 0x94

 Little Endian
 18000 18001 18002 18003
  0x94 0x8a 0x62 0x03
```

From the *readBin* documentation in **R** :

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

The byte-order ("endian-ness") is different for different operating systems. You can determine the endianness of a system by accessing the **R** `.Platform` list of system parameters:

```
> .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.

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 (8 bytes), even though most 32 bit machines actually use long=short (4 bytes). To determine what convention a particular operating system is using one can query the `.Machine` list in **R** :

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

```
[1] 4
```

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

IMPORTANT NOTE: If you get your data from someone else, or you download the binary files from elsewhere (e.g. PASSCAL), 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 database is simply a list of vectors giving information on where to find a particular file on the computer, the times, station and component in each file. The data can be accessed using this database sequentially, or different combinations of files can be sorted and ordered according to the needs of the user.

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

With *Mine.seis* you give it 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).

See the documentation for *makeDB* and *Mine.seis* to get the full instructions and examples of ow to use.

5.1 Example 1

As an example here is the code used to extract data from my Karymsky data set. The database is in list "k97DB". I want to extract 24 hours of data, one hour at a time. I want to see only one station ("kar1") and only components 1 (infrasound) and 4 (vertical). The loop cycles through the traces and gives me a chance to quit if I hit the "QUIT" button in swig.

```
k97DB = makeDB("/home/beer/lees", "R*", kind=1)

 for(i in 1:24)
     {
     at1 = 232+(i-1)/24
     at2 = at1+1/24

     GH = Mine.seis(at1, at2, k97DB, "kar1", c("4", "1") )
     w = swig(GH)
     if(identical(w$but, "QUIT"))break
     }
```

5.2 Example 2

The we use data from the SIGNET experiment that spans 2 years of deployment. There are hundreds of files downloaded from the IRIS DMC. In this case I felt that *makeDB* was not an effective way of storing the data and searching. Instead I wrote a specialized database for more efficient retrieval of the relavant files.

In this example the dataset is very large. there are 18 stations, each recording at different sample rates. The record lengths have been fixed at one hour per file.

The data is stored in a top directory with each station in its own subdirectory. This organization is not convenient for searching for time slices. To get around this problem we created a new database of the file structure, this time based on julian day. In this case, given a time slice, we will only have to search through a much smaller subset of the file names to extract the information we need.

The "database" is simply a list of the files organized first by day and then by station, where the base file names contain the start time of each trace. Given a time slice, i.e. an event time with a pre and post-event span, extract the names of files that fall within that time, read in the traces, cut them down to match the desired times. Then return a swig-compatable structure.

```
##### LISTday is a list consisting of each day's seismic data


###  program to mine the data

signet.mine<-function(LISTday, STA=NULL, COMP=NULL, yr=2009, day=288, hr=0, min=0, sec=0,
                      bef=10*60, aft=50*60, d1="/home/lees/SIGNET-GALAPAGOS")
  {

    NDAYS = names(LISTday)

    R1  = recdate(jd = day, hr = hr, mi = min, sec = sec-bef, yr = yr)
    R2  = recdate(jd = day, hr = hr, mi = min, sec = sec+aft, yr = yr)

####   count up number of hours needed
    sdif = ceiling(((aft-bef)/3600))+1

####  collect all hours of relevance
    zipper = vector()

    for(i in 1:sdif)
      {
        R3 =   recdate(R1$jd, (R1$hr+(i-1)), mi = 0, sec = 0, yr = R1$yr)

        slapit1 = paste(sep=".",formatC(R3$yr, format="d", flag="0", width=4),
          formatC(R3$jd, format="d", flag="0", width=3),
          formatC(R3$hr, format="d", flag="0", width=2))
        slapit2 = paste(sep=".",formatC(R3$yr, format="d", flag="0", width=4),
          formatC(R3$jd, format="d", flag="0", width=3))
        gday  = grep(slapit2, NDAYS )
        g = grep(slapit1, LISTday[[gday]]  )
        zip = LISTday[[gday]][g]
####  select station and or component
        if(!is.null(STA)) zip = grep(STA, zip, value = TRUE )
        if(!is.null(COMP)) zip = grep(COMP, zip, value = TRUE )

        zipper = c(zipper, paste(d1, zip, sep="/"))
      }
##########   read in the binary data
    U1 = JSAC.seis(zipper, Iendian = 1 , BIGLONG=FALSE , PLOT = FALSE)
####   glue the traces that are the same station and component
    RR = GLUE.GET.seis(U1)
#######  convert this list into a more detailed list used by swig
    GH = prepSEIS(RR)

###   chop window down to size

    G1 = getGHtime(GH, wi=which.min(GH$info$off)  )
    t1 = secdifL( G1, R1)
    t2 = secdifL(G1, R2)
    secdifL(R1, R2)
    HH = CHOP.SEISN(GH, WIN = c(t1, t2)  )

    invisible(HH)
  }
```

To extract data we use the above program along with time schedule.

```
###############      schedule
Yr = 2010

 schedule=seq(from=280, to= 288, by=1/24)

for(i in 100:length(schedule))
{

at1 = schedule[i]
rat1 = recdate(jd=at1, hr=0, mi=0, sec=0, yr=Yr)

HH = signet.mine(LISTday, STA=NULL, COMP=NULL, yr=2010, day=rat1$jd,
                  hr=rat1$hr, min=rat1$mi, sec=rat1$sec,
                  bef=30*60, aft=50*60, d1="/home/lees/SIGNET-GALAPAGOS")

SOUT =  swig(HH , sel=which(HH$COMPS=="V") , filters=thefilts  )
}
```