SEISPP Library: Seismic Analysis Library in C++

Gary L. Pavlis
Department of Geological Sciences
Indiana University
pavlis@indiana.edu

Presentation for 2006 Antelope User Group, Tucson, Arizona June 11, 2006

Acknowledgment

Thanks to the Cecile and Ida Green Foundation at IGPP, Scripps Institute of Oceanography and Indiana University for supporting my sabbatical as a Green Scholar in the 2002-2003 academic year. This development would never have gotten off the ground without the time I had to focus on this during that year.

Why did I ever do this?

- Insanity, stupidity, tendency toward masochism – perhaps
- Real reason
  - Object-oriented programming rhetoric; wanted to figure out how much rhetoric is BS and how much is truth
  - Frustration at existing processing libraries
    - Antelope trace library problems
    - Inflexibility of integrated systems for reflection processing
  - Recognition of common concepts in divergent software I was developing

Packages in the C++ library

- Seismic data handling and processing library (libseispp)
- Geographical Curvilinear Grid Objects (libgclgrid)
- Simple, lightweight matrix class (dmatrix) (Note dmatrix is currently in libgclgrid)
- General 1D interpolators (part of libseispp but encapsulated in namespace INTERPOLATOR1D) – Generalized from code in SIA (Igor Mirozov)

Rest of talk will focus on

- SEISPP library and a bit of GCLgrid library
- How you might use these libraries
- Programming: If you don't write programs in some language, you're going to be lost and might want to take a break
- Applicability
  - C programs
  - I'm told it is easy to use C++ objects in python
  - There probably is a tcl/tk interface to C++, but I've not used it
  - Same for other packages like perl, matlab, etc.
  - FORTRAN: forget it

SEISPP Library Documentation

- Minimal man pages
- About 18 months ago stopped writing man pages (unless Dan forced me) in favor of ccdoc (C++ equivalent of Javadoc)
- On the web at:
  http://seismo.geology.indiana.edu/~pavlis/software.html
- Let's look at the copy on my laptop

Utility Data Objects

- Metadata
- DatascopeHandle
- AttributeMap
- AttributeProperties
- SeisppError

Seismic data objects

- Seismic trace data objects
  - TimeSeries
  - ThreeComponentSeismogram
  - ComplexTimeSeries
  - TimeSeriesEnsemble
  - ThreeComponentEnsemble
- Other seismological/geophysical data objects
  - VelocityModel1d
  - SeismicArray
  - Hypocenter
  - SlownessVector
  - RectangularSlownessGrid
  - StationChannelMap
  - TimeWindow
  - TopMute

Processing Objects

- Apply() method model
  - ResampleOperator
  - TimeInvariantFilter
- Creation is initialization model
  - Stack
  - MultichannelCorrelator
- Wrapper
  - XcorProcessingEngine (to be released shortly)

Graphics (not yet released)

- SeismicPlot
  - Simple plot derived from Seismic Unix
- SeismicPlot widget
  - Motif-based widget evolved from SeismicPlot
- SeismicPick
  - Generic pick object returned by plot objects
  - Simplifies interface

GCLgrid library

- See Fan et al. (2006). *Computers in Geosciences, 32*, pp. 371-381.
- 2d and 3d grid objects
- Grids that know where on earth they are located
- Being used for
  - Plane wave migration code (receiver function imaging)
  - Tomography model visualization
  - Pmelgrid clustering geometry
  - 3D travel time table calculator (planned)

Examples overview

- Trace-by-trace algorithm
  - Assumed table driven by db view (row by row processing)
  - Attributes needed and what is saved is variable
- Ensemble algorithm
  - There are TimeWindow based constructors and db view oriented constructors. Example is for TimeWindow
  - Example is for TimeSeriesEnsemble. Similar functionality exists for ThreeComponentEnsembles

Trace-by-trace processing example: dbresample

```
AttributeMap am("css3.0");              // External to internal namespace mapping
MetadataList md_to_input=pfget_mdlist(pf, "input_list");     //List of attributes to be loaded from db
MetadataList md_to_output=pfget_mdlist(pf,"output_list");
 if(dbopen(const_cast<char *>(dbname.c_str()),"r",&db))
                    die(0,"dbopen failed on database %s",dbname.c_str());
 DatascopeHandle dbhi(db,pf,tag);                    //OOP handle to a Datascope database
 DatascopeHandle dbho(dboname,false);
ResamplingDefinitions rsampdef(pf);                  //General recipe for resampling data of different sample rates
dbhi.rewind();
for(int i=0;i<dbhi.number_tuples();++i,++dbhi) {    // Loop over rows of db view.  Note overloaded operator ++
        TimeSeries *tin;              //Pointer to input trace object
        TimeSeries traceout;
         string table("wfdisc");
         tin = new TimeSeries(dynamic_cast<DatabaseHandle&>(dbhi),md_to_input,am);  //db constructor
         traceout = ResampleTimeSeries(*tin,rsampdef,dtout,trim);        //Resampling operator procedure
         chan=traceout.get_string("chan");
         chan[0]=chan_code[0];
         traceout.put("chan",chan);    //Redefine channel code in Metadata with overloaded "put" method
         dfile_name = traceout.get_string("dfile");  //Change dir and dfile in preparation  for saving data
         dfile_name = dfile_name + string(".resampled");
         traceout.put("dfile",dfile_name);
         dbsave(traceout,dbho.db,table,md_to_output,am);   //Save data
         delete tin;    // C++ memory management
}
```

Ensemble processing: new xcor program

```
void XcorProcessingEngine::load_data(Hypocenter & h){
   try {
      current_data_window=TimeWindow(h.time+raw_data_twin.start,
            h.time+raw_data_twin.end);  //need rough time interval for next call
      UpdateGeometry(current_data_window);  //Private method.  Updates station geometry
      auto_ptr<TimeSeriesEnsemble> tse=auto_ptr<TimeSeriesEnsemble>(array_get_data
            (stations,h,analysis_setting.phase_for_analysis,analysis_setting.component_name,
                  raw_data_twin,analysis_setting.tpad,waveform_db_handle,ensemble_mdl,
                  trace_mdl,am));  //Event-based procedure carves out generous time window
      StationTime predarr=ArrayPredictedArrivals(stations,h,analysis_setting.phase_for_analysis);
      //Procedural function to produce a regular gather (uniform dt) with t=0 set by predarr
      regular_gather=auto_ptr<TimeSeriesEnsemble>(AssembleRegularGather(*tse,predarr,
               analysis_setting.phase_for_analysis,regular_gather_twin,target_dt,rdef,true));
      waveform_ensemble=*regular_gather;
      //analysis_setting.filter_param is a string as used by Antelope trfilter
      FilterEnsemble(waveform_ensemble,analysis_setting.filter_param);
      xcorpeak_cutoff=xcorpeak_cutoff_default;
      coherence_cutoff=coherence_cutoff_default;
      stack_weight_cutoff=stack_weight_cutoff_default;
   }
   catch (...) {throw;}
}
```

Some general issues in using C++ with Antelope

- C main program using libseispp
  - Probably won't work
  - C is a subset of C++, so compile main as C++ and avoid the problem (call it main.C, main.cpp, or main.cc instead of main.c)
  - Put plain C procedures called by main in a .c file use: extern "C" {} to declare prototypes
- C++ code calling plain C or FORTRAN code
  - Always requires: extern "C" {}
  - Advise using: ios::sync_with_stdio();
  - Don't forget: using namespace std; (Why this isn't default is beyond me. Something as simple as "string" won't work without it.)

Some SEISPP specific tips

- To use C++ objects is easy. Making one that works right is the hard part. Here are few examples:

```
y=x.s[i];   // set y to the ith sample of x
i=x.sample_number(time);  y=x.s[i];  // time-based indexing
ThreeComponentSeismogram a(blah,blah);    a.rotate_to_standard();
```

- To get a raw pointer to TimeSeries data use:

```
TimeSeries x(blah blah);
double *xptr=&(x.s[0]);
```

  Somewhat evil, but the STL standard requires this to be valid for efficiency. This makes lots of existing procedures easy to utilize.

- MetadataList and AttributeMap abstraction for database constructors
- wfprocess table for db output