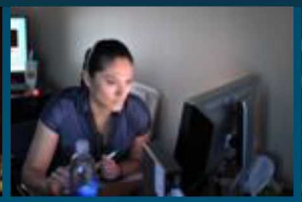
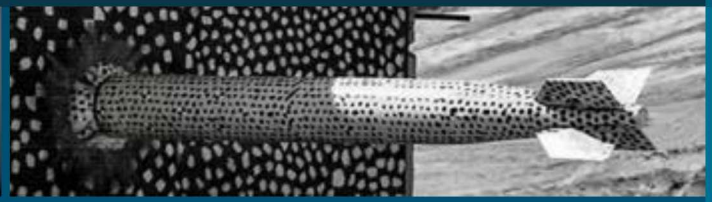
Sandia National Laboratories

SAND2019-2764PE

Overview and planned updates for the RSTT software





PRESENTED BY

Brian Young // Geophysics (8861) // 03 Apr 2019

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Introduction

Software package for rapidly computing Regional Seismic Travel Times (RSTT)

- Rapid enough to incorporate into real-time event location systems

Tri-lab (Sandia, Los Alamos, Lawrence Livermore National Laboratories)

Software originally developed by Sandy Ballard (recently retired)

- I am new Sandia POC

Originally developed in 2007

- Latest release v3.0.5 in July 2016
- Undergoing development and updates (discussed in later slides)

Core library written in C++

- Interfaces in C, FORTRAN, Java

NOTE: Originally called "Seismic Location Baseline Model."

- Some legacy code is still uses "SLBM" instead of "RSTT"

Design

Seismic Location Base Model Software
Design Document Version 3.0
http://www.sandia.gov/rstt/downloads/SLBM_Design.pdf

Tessellated grid (Vp, Vs)

- Grid with which travel times are computed
- Layers:
  - Water
  - Sediment1
  - Sediment2
  - Sediment3
  - Upper crust
  - Middle crust
    - Pn, Sn
    - Pg, Lg
  - Lower Crust
  - Mantle (gradient)

Model is stored in GeoTess format

- http://www.sandia.gov/geotess/

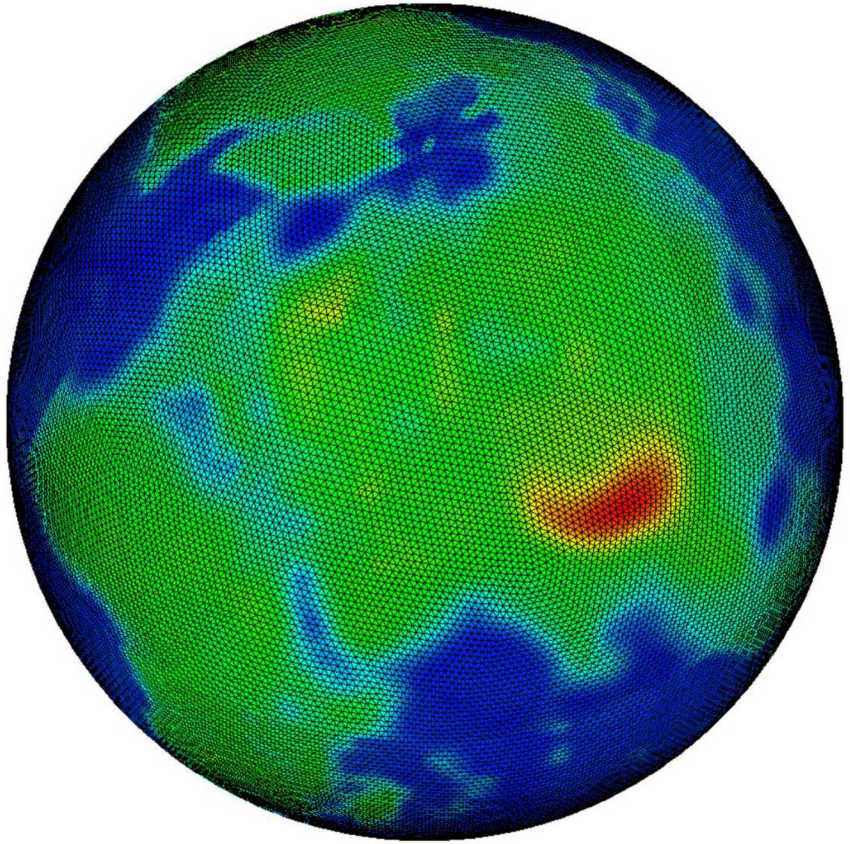
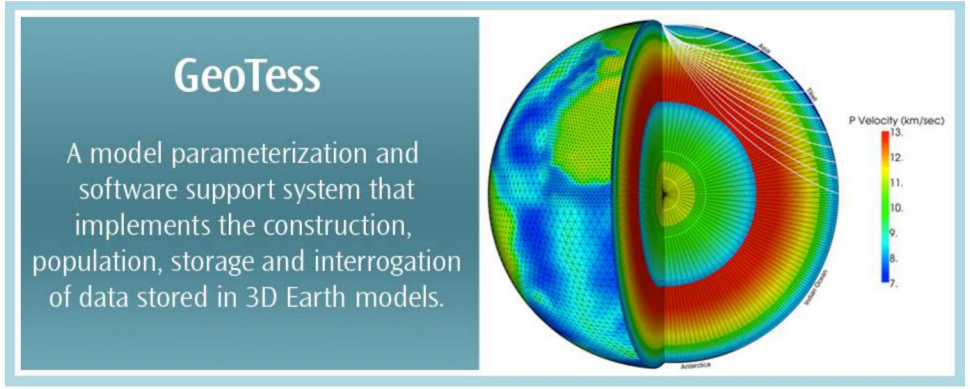


Figure 1 – Orthographic projection of the Earth showing a portion of the SLBM tessellation grid. Colors are contours of the Moho depth.



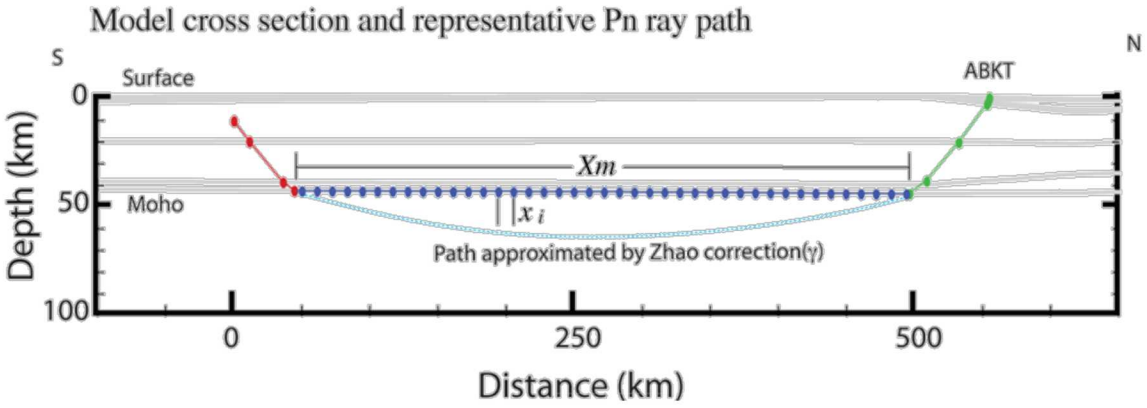
Design

Pn/Sn travel time (crustal events)

- Downwards crustal leg
- Travel across Moho
- Upwards crustal leg
- Mantle gradient correction

Pg/Lg

- Smallest of two travel times
- Headwave method
  - Source --> top of middle crust
  - Along middle crust
  - Middle crust --> receiver
- TauP method (typically local distances)
  - Assume 1D velocity equal to that beneath receiver
  - Compute via TauP



Documentation

SLBM 3.0

Regional Seismic Travel Time

Main Page | Namespaces | Classes | Files

Class List | Class Hierarchy | Class Members

slbm > SlbmInterface

Public Member Functions | Static Public Member Functions | Protected Member Functions | Protected Attributes | Static Protected Attributes | Private Attributes | List of all members

slbm::SlbmInterface Class Reference

The primary interface to the SLBM library, providing access to all supported functionality. More...

```
#include <SlbmInterface.h>
```

Inherited by **slbm::SlbmInterfaceToJNI**.

Public Member Functions

SlbmInterface ()
Default constructor. Instantiates an **SlbmInterface** object based on an ellipsoidal earth. More...

SlbmInterface (const double &earthRadius)
Parameterized constructor. Instantiates an **SlbmInterface** object that is only partly based on an ellipsoidal earth. More...

virtual **~SlbmInterface** ()
Destructor. More...

string **getVersion** ()
Retrieve the SLBM Version number. More...

void **loadVelocityModel** (const string &modelPath)
Load the velocity model into memory from the specified file or directory. This method automatically determines the format of the model. More...

void **saveVelocityModel** (const string &modelFileName, const int &format=4)
Save the velocity model currently in memory to the specified file. More...

int **getBufferSize** () const
Returns the size of a DataBuffer object required to store this SLBMInterface objects model data. More...

void **setInterpolatorType** (const string &interpolatorType)
Specify the interpolation type to use, either 'linear' or 'natural neighbor'. More...

...plications that loop over many calls to **createGreatCircle()** will see a performance improvement if **clear()** is not called within the loop. However, for problems ...e number of sources and or receivers, if memory becomes an issue, applications could call **clear()** within the loop to save memory.

...::SlbmInterface::clearActiveNodes ( ) inline

...::SlbmInterface::clearGreatCircles ( ) protected

...rrent greatCircle object and sets ttHminus, ttHplus, ttZplus and ttHZplus equal to NA_VALUE.

...::SlbmInterface::clearNodeHitCount ( ) inline

...node hit count by setting the hit count of every node to zero.

```
...::SlbmInterface::createGreatCircle ( const string &   phase,
                                        const double &   sourceLat,
                                        const double &   sourceLon,
                                        const double &   sourceDepth,
                                        const double &   receiverLat,
                                        const double &   receiverLon,
                                        const double &   receiverDepth
                                      )                                  inline
```

Instantiate a new **GreatCircle** object between two locations.

Parameters

phase	the phase that this **GreatCircle** is to support. Recognized phases are Pn, Sn, Pg and Lg.
sourceLat	the geographic latitude of the source in radians.
sourceLon	the longitude of source in radians.
sourceDepth	the depth of the source in km.
receiverLat	the geographic latitude of the receiver in radians.
receiverLon	the longitude of the receiver in radians.

Installation

Download

- http://www.sandia.gov/rstt/

`make`

```
$ make
=======================================
Building SLBM_Root on Linux
gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

=======================================
Building GeoTessCPP on Linux with -m64
gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


mkdir lib
gcc -DLinux -m64 -O3 -fPIC -Iinclude -o src/ArrayReuse.o -c src/ArrayReuse.cc
gcc -DLinux -m64 -O3 -fPIC -Iinclude -o src/CPPUtils.o -c src/CPPUtils.cc
gcc -DLinux -m64 -O3 -fPIC -Iinclude -o src/CpuTimer.o -c src/CpuTimer.cc
gcc -DLinux -m64 -O3 -fPIC -Iinclude -o src/DataType.o -c src/DataType.cc
gcc -DLinux -m64 -O3 -fPIC -Iinclude -o src/EnumType.o -c src/EnumType.cc
gcc -DLinux -m64 -O3 -fPIC -Iinclude -o src/GeoTessDataArray.o -c src/GeoTessDataArray.cc
gcc -DLinux -m64 -O3 -fPIC -Iinclude -o src/GeoTessData.o -c src/GeoTessData.cc
```

Set paths

- `SLBM_ROOT=/path/to/rstt`
- `LD_LIBRARY_PATH+=$SLBM_ROOT/lib`
- `PATH+=$SLBM_ROOT/bin`

See `SLBM_Installation_Guide.pdf` in `doc`

Interfaces: C

```
153     // create a GreatCircle object at the requested location.
154     // Note that source and receiver latitudes and longitudes are converted
155     // from degrees to radians.
156     check_err(slbm_shell_createGreatCircle(phase,
157             &srclat, &srclon, &srcDepthKm,
158             &rcvlat, &rcvlon, &rcvDepthKm));
159
160     check_err(slbm_shell_getDistance(&distance));
161     distance *= RAD_TO_DEG;
162
163     printf("Distance       = %9.4f deg\n",distance);
164     // retrieve the travel time, in seconds
165     check_err(slbm_shell_getTravelTime(&tt));
166
167     printf("Travel time    = %9.4f sec\n",tt );
168
169     // retrieve the travel time uncertainty, in seconds
170     check_err(slbm_shell_getTTUncertainty(&tt_uncertainty));
171
172     printf("TT uncertainty = %9.4f sec\n",tt_uncertainty);
173
174     // retrieve slowness
175     check_err(slbm_shell_getSlowness(&slow));
176     slow *= DEG_TO_RAD; // convert from sec/radian to sec/degrees
177
178     printf("Slowness       = %9.4f sec/deg\n",slow);
179
180     // retrieve the travel time uncertainty, in seconds
181     check_err(slbm_shell_getSHUncertainty(&sh_uncertainty));
182     sh_uncertainty *= DEG_TO_RAD; // convert from sec/radian to sec/degrees
183
184     printf("SH uncertainty = %9.4f sec/deg\n",sh_uncertainty);
185
186     // retrieve dtt_dlat
187     check_err(slbm_shell_get_dtt_dlat(&dtt_dlat));
188     dtt_dlat *= DEG_TO_RAD; // convert from sec/radian to sec/degrees
189
190     printf("dtt/dlat       = %9.4f sec/deg\n",dtt_dlat);
191
192     // retrieve dtt_dlon
193     check_err(slbm_shell_get_dtt_dlon(&dtt_dlon));
194     dtt_dlon *= DEG_TO_RAD; // convert from sec/radian to sec/degrees
195
196     printf("dtt/dlon       = %9.4f sec/deg\n",dtt_dlon);
197
```

Interfaces: C++

```
152         // create a GreatCircle object at the requested location.
153         // Note that source and receiver latitudes and longitudes are converted
154         // from degrees to radians.
155         slbm.createGreatCircle(phase,
156             srclatDegrees*DEG_TO_RAD, srclonDegrees*DEG_TO_RAD, srcDepthKm,
157             rcvlatDegrees*DEG_TO_RAD, rcvlonDegrees*DEG_TO_RAD, rcvDepthKm);
158
159         cout << slbm.toString(99);
160
161         slbm.getDistance(distance);
162         distance *= RAD_TO_DEG;
163
164         cout << "Distance       = " << setw(9) << distance << " deg" << endl;
165         // retrieve the travel time, in seconds
166         slbm.getTravelTime(tt);
167
168         cout << "Travel time    = " << setw(9) << tt << " sec" << endl;
169
170         // retrieve the travel time uncertainty, in seconds
171         slbm.getTravelTimeUncertainty(tt_uncertainty);
172
173         cout << "TT uncertainty = " << setw(9) << tt_uncertainty << " sec" << endl;
174
175         // retrieve slowness
176         slbm.getSlowness(slow);
177         slow *= DEG_TO_RAD; // convert from sec/radian to sec/degrees
178
179         cout << "Slowness       = " << setw(9) << slow << " sec/deg" << endl;
180
181         // retrieve the travel time uncertainty, in seconds
182         slbm.getSlownessUncertainty(sh_uncertainty);
183         sh_uncertainty *= DEG_TO_RAD; // convert from sec/radian to sec/degrees
184
185         cout << "SH uncertainty = " << setw(9) << sh_uncertainty << " sec/deg" << endl;
186
187         // retrieve dtt_dlat
188         slbm.get_dtt_dlat(dtt_dlat);
189         dtt_dlat *= DEG_TO_RAD; // convert from sec/radian to sec/degrees
190
191         cout << "dtt/dlat       = " << setw(9) << dtt_dlat << " sec/deg" << endl;
192
193         // retrieve dtt_dlon
194         slbm.get_dtt_dlon(dtt_dlon);
195         dtt_dlon *= DEG_TO_RAD; // convert from sec/radian to sec/degrees
196
```

Interfaces: FORTRAN

```
! create a GreatCircle object at the requested location.
! Note that source and receiver lat, lon are converted from degrees to radians
call slbm_create_great_circle( iphase,  &
    slat*DEG_TO_RAD, slon*DEG_TO_RAD, sdepth, &
    rlat*DEG_TO_RAD, rlon*DEG_TO_RAD, rdepth, &
    err )

if (err .ne. 0) then
    ! an error occurred creating the great circle.
    ! Respond by setting all regression values for this record
    ! to -999 and do not retrieve values for tt, slowness, etc.
    ! Presumably the same error happened when the regression data were generated.
    distance = error_value
    tt = error_value
    tt_uncertainty = error_value
    slow = error_value
else
    ! make calls to retrieve distance, tt, tt_uncertainty and slowness.
    ! if any of the function calls return a non-zero error code
    ! set all the values to -999 and move on.
    call slbm_get_distance(distance, err)
    if (err .ne. 0) then
        distance = error_value
        tt = error_value
        tt_uncertainty = error_value
        slow = error_value
    else
        distance = distance * RAD_TO_DEG
        call slbm_get_travel_time(tt, err)
        if (err .ne. 0) then
            distance = error_value
            tt = error_value
            tt_uncertainty = error_value
            slow = error_value
        else
            call slbm_get_tt_uncertainty(tt_uncertainty, err)
            if (err .ne. 0) then
                distance = error_value
                tt = error_value
                tt_uncertainty = error_value
                slow = error_value
            else
                call slbm_get_slowness(slow, err)
                if (err .ne. 0) then
                    distance = error_value
```

Interfaces: Java

```
267                 // create a GreatCircle object at the requested location.
268                 // Note that source and receiver latitudes and longitudes are converted
269                 // from degrees to radians.
270                 slbm.createGreatCircle(phase,
271                         toRadians(srcLat),
272                         toRadians(srcLon),
273                         srcDepth,
274                         toRadians(rcvLat),
275                         toRadians(rcvLon),
276                         rcvDepth);
277
278                 distance = Math.toDegrees(slbm.getDistance());
279
280                 // retrieve the travel time, in seconds.
281                 tt = slbm.getTravelTime();
282
283                 tt_uncertainty = slbm.getTravelTimeUncertainty();
284
285                 slow = Math.toRadians(slbm.getSlowness());
286
287             }
288             catch (SLBMException ex)
289             {
290                 // slbm threw an exception.
291
292                 // Print the reason to screen
293                 // cout << ex.emessage << endl;
294
295                 // set computed travel time to invalid value and keep going.
296                 // Presumably the previously computed value was invalid also.
297                 distance = tt = tt_uncertainty = slow = -999.;
298             }
299
300             // increment number of records read from file.
301             ++nrecords;
302
303             // if computed travel time does not agree with previously
304             // computed value, print information to the screen
305             if (abs(distance-distance0) > tolerance
306                     || abs(tt-tt0) > tolerance
307                     || abs(tt_uncertainty-tt_uncertainty0) > tolerance
308                     || abs(slow-slow0) > 10*tolerance)
309             {
310                 ++ndifferent;
311
```

Upcoming software modifications

Various bug fixes, codebase updates

Path-specific travel-time uncertainty

Variable model resolution