Introducing Madagascar

A Computational Platform for Geophysical Data Processing and Reproducible Numerical Experiments

Sergey Fomel\textsuperscript{1} Paul Sava\textsuperscript{2} Felix Herrmann\textsuperscript{3}

\textsuperscript{1}Bureau of Economic Geology
Jackson School of Geosciences
University of Texas at Austin

\textsuperscript{2}Colorado School of Mines

\textsuperscript{3}University of British Columbia

June 11, 2006
From RSF to Madagascar
1. New
Outline

1. New
2. Test Driven Development
Outline

1. New
2. Test Driven Development
3. Open Source
Outline

1. New
2. Test Driven Development
3. Open Source
4. Universal File Format
1. New
2. Test Driven Development
3. Open Source
4. Universal File Format
History

- Started in June 2003
- 6 main contributors
- Version 0.9 released June 2006
- Development version
  - 2,000 revisions
  - 300 main programs (140,000 lines of C)
  - 3,000 tests (40,000 lines of Python)
  - 30 papers (20,000 lines of \LaTeX)
- \url{http://rsf.sf.net/}

S. Fomel (UT Austin)
Heritage

- **SEPlib**
  - Rob Clayton, Jon Claerbout, Dave Hale, Stew Levin, Rick Ottolini, Joe Dellinger, Steve Cole, Dave Nichols, Martin Karrenbach, Biondo Biondi, Bob Clapp

- **SEP reproducible research system**
  - Martin Karrenbach, Matthias Schwab, Joel Schroeder, Sergey Fomel, Bob Clapp

- **Seismic Unix**

- **DDS**
Outline

1. New
2. Test Driven Development
3. Open Source
4. Universal File Format
Test Driven Development

- XP, agile software engineering, refactoring
  - Tests drive development
- Computational geophysics
  - Tests are numerical experiments
- Science
  - Reproducible experiments mean progress
### RSF Programs

Add a search feature later.

#### filt/imag

<table>
<thead>
<tr>
<th>filt/imag</th>
</tr>
</thead>
<tbody>
<tr>
<td>sfaddm2d</td>
</tr>
<tr>
<td>sfam2d</td>
</tr>
<tr>
<td>sfc2r</td>
</tr>
<tr>
<td>sfcgige</td>
</tr>
<tr>
<td>sfcgI2</td>
</tr>
<tr>
<td>sfsegcaen</td>
</tr>
<tr>
<td>sfoconstgmic2</td>
</tr>
<tr>
<td>sfcube2list</td>
</tr>
<tr>
<td>sfdsdistance</td>
</tr>
<tr>
<td>sfdsmp</td>
</tr>
<tr>
<td>sfdsr</td>
</tr>
<tr>
<td>sfdsst2</td>
</tr>
<tr>
<td>sfdsx1tal</td>
</tr>
<tr>
<td>sfdsxntalvti</td>
</tr>
<tr>
<td>sfdsxvt</td>
</tr>
<tr>
<td>sfdsxm</td>
</tr>
</tbody>
</table>

#### filt/main

<table>
<thead>
<tr>
<th>filt/main</th>
</tr>
</thead>
<tbody>
<tr>
<td>sfafad</td>
</tr>
<tr>
<td>sfattr</td>
</tr>
<tr>
<td>sfcct</td>
</tr>
<tr>
<td>sfecongrad</td>
</tr>
<tr>
<td>sfcedetest</td>
</tr>
<tr>
<td>sfeconmplx</td>
</tr>
<tr>
<td>sfecongrad</td>
</tr>
<tr>
<td>sfecon</td>
</tr>
<tr>
<td>sfecon</td>
</tr>
<tr>
<td>sfecon</td>
</tr>
<tr>
<td>sfecon</td>
</tr>
<tr>
<td>sfecon</td>
</tr>
<tr>
<td>sfecon</td>
</tr>
</tbody>
</table>

Done
**Synopsis**

```bash
sfeikonal < vel.rsf > time.rsf shotfile=shots.rsf vel=vel order=2 br1=d1 br2=d2 br3=d3 plane1=n plane2=n plane3=n b1=plane[2]? n1: (int) (br1/d1+0.5) b2=plane[]? n2: (int) (br2/d2+0.5) b3=plane[0]? n3: (int) (br3/d3+0.5) zshot=0. yshot=0+0.5*(n2-1)*d2 xshot=0+0.5*(n3-1)*d3 shotfile=
```

**Parameters**

- `int b1=plane[2]? n1: (int) (br1/d1+0.5)`
- `int b2=plane[1]? n2: (int) (br2/d2+0.5)`
- `int b3=plane[0]? n3: (int) (br3/d3+0.5)`
- `float br1=d1`
- `float br2=d2`
- `float br3=d3`
- `int order=2 [1,2]`
- `bool plane1=n [y/n]`
- `bool plane2=n [y/n]`
- `bool plane3=n [y/n]`

- **Constant-velocity box around the source (in samples)**
- **Constant-velocity box around the source (in physical dimensions)**
- **Accuracy order**

- **Plane-wave source**
bool plane3 = n [y/n]  
string shotfile=  
float xshot = 0 + 0.5*(n3-1)*d3  
float yshot = 0 + 0.5*(n2-1)*d2  
float zshot = 0.

used in

GEO391
- hw3/sigbee2
- hw3/sigbee

GTI
- fdmod/zyggyFDM
- multi/tree
- multi/basic
- dmecc/mai
- wavened/sr
- wavened/imp

SEP
- fmleko/linarch
- fmsec/cvel
- fmsec/marm
```python
from refproj import *
import math

Fetch('salt_slow_desp.HH', 'segsgne')
Flow('salt', 'salt_slow_desp.HH', 'dd_form=native')
Flow('seg', 'salt', 'window j1=2 j2=2 j3=2 | eikonal vel=0 zshot=1500')

def traverse(n2, n3, d2, d3):
    return put n2=d2 n3=1 d2=mg | window j2=d2' %
    (n2*n3, math.hypot(d2, d3)/(n2+1), n2+1)

Flow('st', 'salt', 'traverse(676, 676, 20, 20)')
Flow('sgt', 'seg', 'traverse(338, 338, 10, 10)')

Plot('xy', 'salt',
    window n1=1 f1=75 |
    grey pclip=100 gpow=1 bias=0.00034
    color=j scalebar=n title="SEG/SEAG salt model" screenratio=1 screenht=0.6571
    ***
)
Plot('xy', 'seg',
    window n1=1 f1=75 |
    contour nc=100 plotcol=7 wantaxis=n wanttitle=n
    screenratio=1 screenht=0.6571
    ***
)
Plot('cxy', 'sky', 'opty', 'Overlay')

Plot('sxz', 'st',
    grey pclip=100 gpow=1 bias=0.00034
    color=j scalebar=n wanttitle=n screenratio=0.21966 screenht=3
    ***
)
Plot('ezx', 'sgt',
    contour nc=100 plotcol=7 wantaxis=n wanttitle=n
    screenratio=0.21966 screenht=3
    ***
)
Plot('cxz', 'sxz', 'opty', 'Overlay')

Result('salt', 'exz', 'overty', 'OverlayIno')

End()
```
according to the first-order difference operator (1). As a result, the computational error of this method goes to zero with the decrease in the grid size in a linear fashion. The proof of validity of the method (omitted here) is also analogous to that of Dijkstra's algorithm (Sethian, 1996a; Sethian, 1996b). As in most of the shortest-path implementations, the computational cost of extracting the minimum point at each step of the algorithm is greatly reduced [from $O(N)$ to $O(\log N)$ operations] by maintaining a priority-queue structure (heap) for the NarrowBand points (Cormen et al., 1990).

Figure 1 shows an example application of the fast marching eikonal solver on the three-dimensional SEG/EAGE salt model. The computation is stable despite the large velocity contrasts in the model. The current implementation takes about 10 seconds for computing a 100x100x100 grid on one node of SGI Origin 200. Alkhalifah and Fomel (1997) discuss the differences between Cartesian and polar coordinate implementations.

**Figure 1.** Constant-traveltime contours of the first-arrival traveltime, computed in the SEG/EAGE salt model. A point source is positioned inside the salt body. The top plot is a diagonal slice; the bottom plot, a depth slice.

The difference equation (1) is a finite-difference approximation to the continuous eikonal equation

$$
\left( \frac{\partial t}{\partial x} \right)^2 + \left( \frac{\partial t}{\partial y} \right)^2 + \left( \frac{\partial t}{\partial z} \right)^2 = \phi^2(x, y, z),
$$

where $x$, $y$, and $z$ represent the spatial Cartesian coordinates. In the next two sections, I show how the updating procedure can be derived without referring to the eikonal equation, but with the direct use of Fermat's principle.
/* Fast marching eikonal solver (3-D). */

Copyright (C) 2004 University of Texas at Austin

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

#include "math.h"
#include "crsf.h"

#include "fastmarch.h"

int main (int argc, char* argv[])
{
    int b1, b2, b3, n1, n2, n3, i, nehot, ndir, is, order, nl23, *p;
    float br1, br2, br3, o1, o2, o3, d1, d2, d3, slow;
    float **x, *t, *v;
    char *ofile;
    bool isvel, plane[3];
    sf_file vel, time, shots;
    sf_init (argc, argv);
    vel = sf_input("in");
    time = sf_output("out");
    if (SF_FLOAT != sf_typeof(vel))
        sf_error("Need float input");
    if(!sf_histint(vel,"ni","ni")) sf_error("No ni- input");
}

Done
Abandoning the habit of secrecy in favor of process transparency and peer review was the crucial step by which alchemy became chemistry. In the same way, it is beginning to appear that open-source development may signal the long-awaited maturation of software development as a discipline.

*Eric S. Raymond, The art of UNIX programming, 2004*
The purpose of reproducible research is to facilitate someone going a step further by changing something. The first step that someone will want to make is to be sure that your work is reproducible before they change and improve upon it.

*Jon F. Claerbout, Reproducible research*
An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

**Jon B. Buckheit and David L. Donoho, WaveLab and reproducible research, 1995**
Within the world of science, computation is now rightly seen as a third vertex of a triangle complementing experiment and theory. However, as it is now often practiced, one can make a good case that computing is the last refuge of the scientific scoundrel... Where else in science can one get away with publishing observations that are claimed to prove a theory or illustrate the success of a technique without having to give a careful description of the methods used, in sufficient detail that others can attempt to repeat the experiment?

Randall J. LeVeque, Wave propagation software, computational science, and reproducible research, 2006
Reproducible Research Tools

- SEP
  - GNU make rules
  - Perl scripts
  - Shell scripts
  - \LaTeX\ macros

- Madagascar
  - SCons (Python replacement for make)
Outline

1. New
2. Test Driven Development
3. Open Source
4. Universal File Format
Open Source

- Open source means freedom to the user
  - Freedom to use
  - Freedom to study and modify
  - Freedom to redistribute
  - Freedom to improve
- Open source means collaboration and peer review
- Madagascar package is 100% open-source
  - GPL license
  - Hosted by Sourceforge
  - Developed with Subversion
Outline

1. New
2. Test Driven Development
3. Open Source
4. Universal File Format
Universal File Format

- Borrowed from classic SEPlib
- Binary data and test headers
- $N$-dimensional hypercubes
- Can represent seismic data (regular, irregular, 3-D, 9-D, etc.) as well as any other kind of data
To design a perfect anti-Unix, make all file formats binary and opaque, and require heavyweight tools to read and edit them.

... If you feel an urge to design a complex binary file format, or a complex binary application protocol, it is generally wise to lie down until the feeling passes.

Eric S. Raymond, *The art of UNIX programming*, 2004
Madagascar is a software package for geophysical data processing and reproducible numerical experiments.

New, test-driven, open-source, using a universal file format.
http://rsf.sf.net

School and workshop

Reproducible Research in Computational Geophysics
August 30-31, 2006
Vancouver BC, Canada