Harbin Madagascar Workshop 2015

Developing your own Madagascar Programs

Jeffrey Shragge

School of Physics and School of Earth and Environment
University of Western Australia

January 8th, 2015

Writing / adding your own codes

Isn't there a lot there already?!?

- There are $>$ 1480 programs already in Madagascar
- Includes both seismic and non-seismic tools
- Incorporates generic data manipulation tools

Yes! But not everything!

- Some tasks not easily doable with existing tools
- Some tools might not exist at all (i.e. your research!)
- Include some existing tools with your programs

Where to begin?

Should you build programs for all of your needs?

NO!

Examples of “Wheel” programs

- Matrix multiply
- Dataset concatenation
- FFTs,
- Bandpass filtering
- ...

Where to find existing tools

Make sure to check out ...

sfdoc -k .

Where to begin ...

- Focus your time / energy on doing your research!
- Do not waste time reinventing/reimplementing existing algorithms
- Look at existing Madagascar modules and programs
  - $RSFSRC/book/Recipes
  - $RSFSRC/user/
  - User / Developer mailing lists

Presentation Goals

What is the main goal of this tutorial?

After this presentation you should know how to put your own programs into Madagascar

How are we going to do it?

1. Finish coding a “Vector Addition” program
2. Compile and Install it in RSF
3. Test it with various SConstruct Flow() and Plot() rules

Agenda

This talk will focus on:

1. When should I start adding my own codes?
2. Madagascar's API
3. RSF program structure
4. Assignment 1: Vector Addition

Agenda

This talk will focus on:

1. When should I start adding my own codes?
2. Madagascar's API
3. RSF program structure
4. Assignment 1: Vector Addition

Where to draw the line with development

Program design goals

RSF programs are task-centric:

- Each program performs a single or a common set of tasks:
  - Spray (Forward operator)
  - Stack (Adjoint operator)
- Programs constructed to run in a **pipeline** with input from standard in and output to standard out:
  - < in.rsf sf_my_program > out.rsf
  - < in.rsf sftransp | sfmyprogram | sftransp > out.rsf
- Pass parameters from:
  - Command line or SConstruct file (in rule or in dictionary)

Where to draw the line with development

Rafael wants to apply his newest filter in the frequency domain: $\mathbf{L}(\omega)$. However, his RSF data is in the time domain $\mathbf{d}(t)$. How should Rafael design his new RSF program to obtain filtered data $\mathbf{d}_{filt}(t)$?

Use a solution that involves FFT pair $\mathbf{F}(t \rightarrow \omega)$ and $\mathbf{F}^{-1}(\omega \rightarrow t)$:

$$\mathbf{d}_{filt} = \mathbf{F}^{-1}\mathbf{L}\mathbf{F}\mathbf{d} \tag{1}$$

Let us explore three solutions:

1. Write new code that applies $\mathbf{F}$, then $\mathbf{L}$, and then $\mathbf{F}^{-1}$.
2. Write new code that applies $\mathbf{L}$, but calls an existing library for $\mathbf{F}$ and $\mathbf{F}^{-1}$.
3. Write an $\mathbf{L}$ filter program. Use Madagascar to apply $\mathbf{F}$ and $\mathbf{F}^{-1}$.

Thinking about program design

Three possible solutions

1. Rafael writes code that applies **F**, then **L**, and then $\mathbf{F}^{-1}$.
2. Rafael writes a new code that applies **L**, and calls existing libraries for **F** and $\mathbf{F}^{-1}$
3. Rafael writes an **L** filter program, and uses Madagascar to apply **F** and $\mathbf{F}^{-1}$

Pros and Cons

1. Not task-centric and Rafael wastes time researching / writing / debugging a FFT code.
2. Not task-centric but Rafael uses existing libraries to shorten development time.
3. Task-centric coding that can be used in a pipeline, and be applied to any frequency domain data set.

Agenda

This talk will focus on:

1. When should I start adding my own codes?
2. Madagascar's API
3. RSF program structure
4. Assignment 1: Vector Addition (25 mins)

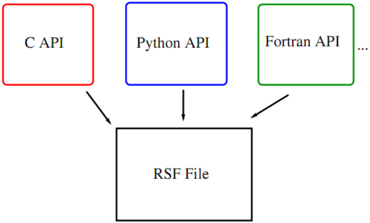
RSF framework

Application Programming Interface (API)

- A set of rules that software programs follow to communicate with each other
- Specifies routines, data structures and the protocols for communicating between the consumer program and the implementer program of the API

Madagascar has APIs for a number of computer languages:

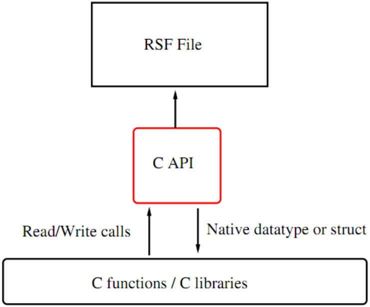
- C/C++
- Python
- f77, f90
- Matlab, Octave
- Java



Overview of the C API

Strength of Madagascar API (here C):

- **Interoperable**:
  - Common RSF file structure
  - Defines standard for data exchange
  - Enables pipelining with other programs
- Improves **development efficiency**
  - Access RSF C functions / libraries
  - **Encapsulate** many tasks (e.g. predefined data I/O subroutines)
- Enhances **usability**
  - Common program documentation style
  - Helps other people use your code
  - Helps you use other people's code



Agenda

This talk will focus on:

1. When should I start adding my own codes?
2. Madagascar's API
3. RSF program structure
4. Assignment 1: Vector Addition (25 mins)

RSF Clipit Example (F90)

Geophysical task: Clip 1D data set wherever values are greater than *clip*

```
!! STEP 1 – Documentation
program Clipit

!! STEP 2 – Import RSF API
  use rsf

   implicit none
  type ( file )   :: in, out
  integer   :: n1, n2, i1 , i2
  real   :: clip
  real , dimension(:), allocatable :: trace

!! STEP 3 – Init RSF command line parser
  call  sf_init ()

!! STEP 4 – Read command line variables
  call from_par("clip", clip )

!! STEP 5 – Declare input / output RSF files
  in = rsf_input () ; out = rsf_output ()

!! STEP 6 – Read input data headers
  call from_par(in,"n1",n1)

!! STEP 7 – Write output data headers
  call to_par(out,"n1",n1)

  n2 = filesize (in,1)
  allocate (trace (n1))

!! STEP 8 – Read in input data set
  call rsf_read(in,trace)
  do i2=1, n2 ! loop over traces

     !! STEP 9 – Do "geophysics"
     where (trace(:, i2) > clip ) trace (:, i2) =
          clip
     where (trace(:, i2) < –clip) trace (:, i2) =
          –clip

     !! STEP 10 – Write output data sets
     call rsf_write (out,trace)
  end do
end program Clipit
```

Generic RSF program

1. Documentation (comments)
2. Import RSF API
3. Initialize RSF command line parser
4. Read command line variables
5. Declare all input / output RSF files
6. Read input data headers
7. Create output data headers
8. Read input data sets
9. (Do geophysics)...
10. Write output data

```
!! STEP 1
! Clipit - Program to clip a traces
!! STEP 2
use rsf
!! STEP 3
call sf_init()
!! STEP 4
call from_par("clip",clip)
!! STEP 5
in = rsf_input();
out = rsf_output()
!! STEP 6
call from_par(in,"n1",n1)
!! STEP 7
call to_par(out,"n1",n1)
!! STEP 8
call rsf_read(in,trace) !! STEP 9
where (trace > clip) trace = clip
!! STEP 10
call rsf_write(out,trace)
```

Agenda

This talk will focus on:

1. When should I start adding my own codes?
2. Madagascar's API
3. RSF program structure
4. Assignment 1: Vector Addition