Developing your own programs in Madagascar

Jeffrey Shragge (with thanks to Jeff Godwin, CSM)

Centre for Petroleum Geoscience and CO2 Sequestration
University of Western Australia

July 22, 2011

Writing / adding your own codes

Isn't there a lot there already?!?

- There are $>$ 850 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!

重新发明轮子

Examples of “Wheel” programs

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

Standing on the shoulders of giants ...

Make sure to check out ...

sfdoc -k .

Where to begin ...

- Focus your time / energy on doing YOUR new research!
- Do not waste time reinventing things
- Look at existing Madagascar programs for help
  - $RSFROOT/RSFSRC/book/Recipes
  - $RSFROOT/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 architecture goals

RSF programs are task-centric:

- Each program performs one task or a common task set:
  - 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 | sfwindow | sf_my_program | sfwindow > out.H
- Pass parameters from:
  - Command line or SConstruct file (in rule or in dictionary)

Where to draw the line with development

Zhang wants to apply the newest XYZ filter in the frequency domain: $\mathbf{L}(\omega)$. However, his RSF data is in the time domain $\mathbf{d}(t)$. How should Zhang 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 3 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. Zhang writes code that applies **F**, then **L**, and then $\mathbf{F}^{-1}$.
2. Zhang writes a new code that applies **L**, and calls existing libraries for **F** and $\mathbf{F}^{-1}$
3. Zhang writes an **L** filter program, and uses Madagascar to apply **F** and $\mathbf{F}^{-1}$

Pros and Cons

1. Not task-centric and Zhang wastes time researching / writing / debugging a FFT code.
2. Not task-centric but Zhang 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

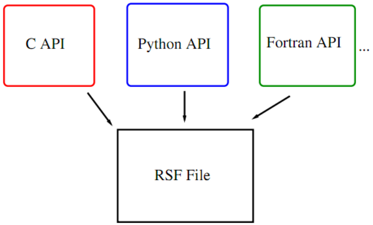
This talk will focus on:

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 or interface that software programs follow to communicate with each other
- Specifies routines, data structures and the protocols used for communicate between the consumer program and the implementer program of the API



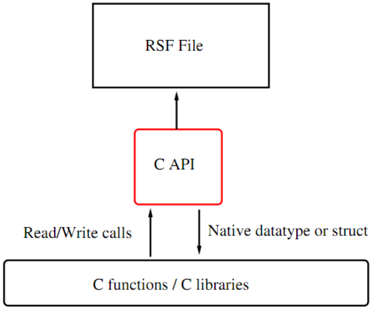
Madagascar has a number of APIs

- C/C++
- Python
- Fortran 77
- Fortran 90
- Matlab
- Java
- Octave

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 where greater than user defined value.

```
!!  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 –  Initialize  RSF command line
       parser
  call    sf_init  ()

  !!  STEP 4 – Read command line variables
  call  from_par(”clip ”, clip )
  –
  !!  STEP 5 – Declare all 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))

  do i2=1, n2                         !  loop over traces
      !!  STEP 8 – Read input data sets
      call   rsf_read(in ,trace)   !!   STEP 7

      !!  STEP 9 – Do “geophysics”
     where (trace >   clip )  trace  =   clip
     where (trace < –clip) trace  = –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
```

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

Part 1: Building your program

1. Take your copy of SCHOOL_CODE.tgz and do:
    - cp SCHOOL_CODE.tgz $RSFROOT/RSFSRC/user/; cd $RSFROOT/RSFSRC/user/;
    - tar -xcvf SCHOOL_CODE.tgz;
2. You have a copy of an **almost finished** "vector addition" code in C
    - $RSFROOT/RSFSRC/user/school/Mvectoradd_C.c

   Your assignment is to put the "geophysics" into the vector addition code. Open the file in a text editor and complete the C=A+B assignment.
    - Hint: Vector index in F90 is A(); Vector index in C is A[].
3. After completing this task build the code in the local directory by:
    - Type: **scons sfvectoradd_C**
4. Install the C files (not yet F90?) into $RSFROOT/bin/ by
    - Type: **cd $RSFROOT/RSFSRC/ ; scons install**

Part 2: Testing your program

5. Take your copy of SCHOOL_TEST.tgz and do:
    - cp SCHOOL_TEST.tgz /path/to/work/dir/; cd /path/to/work/dir/;
    - tar -xcvf SCHOOL_TEST.tgz

   You have an incomplete **SConstruct** file in ./school_test/. We have to add lines into this file in order to test our **sfvectoradd_C** programs.

7. Create two random vectors **A** and **B** (of the same length) to add. Create these with a Madagascar program in the provided **Flow()** rule.
    - Hint: there is more than 1 answer: **sfdoc -k .**
    - Build the Flow() rule for A and B by: **scons A.rsf; scons B.rsf**

8. You must obtain **C** from **A** and **B**. Do this by: **scons C_C.rsf**

9. Do you know that you got the correct answer? Let's test our program against a (correct) Madagascar one: **sfmath**. Look at the **sfmath** self-doc page to find out how to complete the provided **Flow()** rule.
    - Hint: there is more than 1 answer: **sfdoc -k .**
    - Build example by: **scons Ctest_C.rsf**