From modeling to full waveform inversion: A hands-on tour using Madagascar

Pengliang Yang
ISTerre - Univ. Grenoble Alpes
E-mail: ypl.2100@gmail.com

ABSTRACT

This tutorial is devoted to Madagascar school 2016 Zurich. In this tutorial, there are two aspects we would like to explore:

- Madagascar functionality, which is the tool. We may consider Madagascar to facilitate our research, from the numerical test to publication.
- Scientific aspects, which are the key things we care. Even though we are playing a game with simple exercise, we have to think about the scientific enhancement/improvement to polish the techniques used in modeling and inversion applications.

INTRODUCTION

As a brief introduction, I want to emphasize several points from my own understanding:

- Do you really need Madagascar? Yes because you will benefit a lot from it. Of course no if you are able to manage things in your own way, even better than Madagascar.
- Assuming we need Madagascar hereafter. In what aspects we can benefit from it?
- Is it a responsibility to contribute your papers, in particular your codes? Definitely no, especially when your research is sponsored by others while a permission public release is not accessible. You may want to be selfish: "I only want to use the codes from others instead of sharing mine with the community". No objection: Some people are doing things like that.
- Keep in mind Madagascar is not the goal, it is just a tool to share your research progress with others. If you are ready to be open, why not a contributor?

PRELIMINARY

In this section, I provide some fundamental gadgets to help the beginners run Madagascar with ease.

Reproduce numerical examples and papers using SConstruct

Processing workflow:

- `Fetch`(data_file,dir,ftp_server_info): download data_file from a specific directory dir of an FTP server
- `Flow`(targets,sources,commands): generate target[s] from source[s] using command[s]
- `Plot`(intermediate_plot[,source],plot_command): generate intermediate_plot in the working directory
- `Result`(plot,intermediate_plots,combination) generate a final plot in Fig folder of the working directory
- `End`(): collect default targets

Run scons for your computation

scons run an SConstruct to generate data

scons view view the results from an SConstruct

scons -c clean the local directory, delete all target files

pscons parallel execution of an SConstruct

Paper Sconstruct imports Python packages for processing TeX files:

```
from rsf.tex import *
End(name, lclass , options , use)
```

- name - name of the root tex file to build, paper.tex.
- lclass - name of the LaTeX class file to use.
- options - document options for LaTeX class file.
- use - names of LaTeX packages to import during compilation.

To generate your paper including numerical examples:

- `sftour scons lock`: lock the results from an SConstruct
- `scons read/paper.read`: look at the generated paper in pdf
- `scons -c` remove all intermediate files

Some of the most useful commands

Plotting the figures

sfgraph create line plots, or scatter plots

sfgrey create raster plots or 2D image plots

sfgrey3 create 3D image plots of panels (or slices) of a 3D cube

sfwiggle plot data with wiggly traces

Look up data attributes and data processing:

sfin check the layout of the data, number of points in each dimension, sampling intervals in each axis, labels, units, ...

sfattr check statistical properties: covariable, rms, mean, minimum and maximum etc

sfwindow window or select part of data

sfadd add two dataset with scaling factors

sfmath mathematical operations for the data: log, sin, tan, exp, ...

sfsmooth smoothing the data using triangular window (repeating many time to approximate Gaussian)

Image format: Vplot

- suffix '.vpl', vectorized image→scaling without loss of quality
- convert to be `pdf/eps/png/jpeg/tiff/mpeg/...`
- how: `vpconvert format=pdf fig.vpl`

FORWARD MODELING

The wave equation we consider in this course material

$$(\frac{1}{v^2}\partial_t^2 - \nabla^2)p = f \tag{1}$$

Omitting the source, extrapolate your wavefield:

$$p^{k+1} = 2p^k - p^{k-1} + \Delta t^2 v^2 \nabla^2 p^k \tag{2}$$

where

$$\nabla^2 p = \frac{p[ix][iz+1] - 2p[ix][iz] + p[ix][iz-1]}{\Delta z^2} + \frac{p[ix-1][iz] - 2p[ix][iz] + p[ix-1][iz]}{\Delta x^2} \tag{3}$$

The Clayton-Enquist absorbing boundary condition (ABC) (Clayton and Engquist, 1977)

$$\text{left boundary} : \frac{\partial^2 p}{\partial x \partial t} - \frac{1}{v}\frac{\partial^2 p}{\partial t^2} = \frac{v}{2}\frac{\partial^2 p}{\partial z^2} \tag{4}$$

The codes in every time step looks like

```
//dtz=dt/dz; dtx=dt/dx
 void step_forward(float **p0, float **p1, float **p2,
       float **vv, float dtz, float dtx, int nz, int nx)
/*< forward modeling step, Clayton-Enquist ABC incorporated >*/
{
    int ix,iz;
    float v1,v2,diff1,diff2;

    for (ix=0; ix < nx; ix++)
    for (iz=0; iz < nz; iz++)
    {
    v1=vv[ix][iz]*dtz; v1=v1*v1;
    v2=vv[ix][iz]*dtx; v2=v2*v2;
    diff1=diff2=-2.0*p1[ix][iz];
    diff1+=(iz-1>=0)?p1[ix][iz-1]:0.0;
    diff1+=(iz+1<nz)?p1[ix][iz+1]:0.0;
    diff2+=(ix-1>=0)?p1[ix-1][iz]:0.0;
    diff2+=(ix+1<nx)?p1[ix+1][iz]:0.0;
    diff1*=v1;
    diff2*=v2;
    p2[ix][iz]=2.0*p1[ix][iz]-p0[ix][iz]+diff1+diff2;
    }

/*
    // top boundary
    iz=0;
    for (ix=1; ix < nx-1; ix++) {
           v1=vv[ix][iz]*dtz;
           v2=vv[ix][iz]*dtx;
           diff1=   (p1[ix][iz+1]-p1[ix][iz])-
                    (p0[ix][iz+1]-p0[ix][iz]);
           diff2=     p1[ix-1][iz]-2.0*p1[ix][iz]+p1[ix+1][iz];
           diff1*=v1;
           diff2*=0.5*v2*v2;
           p2[ix][iz]=2.0*p1[ix][iz]-p0[ix][iz]+diff1+diff2;
    }
*/
    /* bottom boundary */
    iz=nz-1;
    for (ix=1; ix < nx-1; ix++) {
           v1=vv[ix][iz]*dtz;
           v2=vv[ix][iz]*dtx;
```

```
                diff1=-(p1[ix][iz]-p1[ix][iz-1])+
                        (p0[ix][iz]-p0[ix][iz-1]);
                diff2=p1[ix-1][iz]-2.0*p1[ix][iz]+p1[ix+1][iz];
                diff1*=v1;
                diff2*=0.5*v2*v2;
                p2[ix][iz]=2.0*p1[ix][iz]-p0[ix][iz]+diff1+diff2;
        }

        /* left boundary */
        ix=0;
        for (iz=1; iz <nz-1; iz++){
                v1=vv[ix][iz]*dtz;
                v2=vv[ix][iz]*dtx;
                diff1=p1[ix][iz-1]-2.0*p1[ix][iz]+p1[ix][iz+1];
                diff2=(p1[ix+1][iz]-p1[ix][iz])-
                        (p0[ix+1][iz]-p0[ix][iz]);
                diff1*=0.5*v1*v1;
                diff2*=v2;
                p2[ix][iz]=2.0*p1[ix][iz]-p0[ix][iz]+diff1+diff2;
        }

        /* right boundary */
        ix=nx-1;
        for (iz=1; iz <nz-1; iz++){
                v1=vv[ix][iz]*dtz;
                v2=vv[ix][iz]*dtx;
                diff1=p1[ix][iz-1]-2.0*p1[ix][iz]+p1[ix][iz+1];
                diff2=-(p1[ix][iz]-p1[ix-1][iz])+
                        (p0[ix][iz]-p0[ix-1][iz]);
                diff1*=0.5*v1*v1;
                diff2*=v2;
                p2[ix][iz]=2.0*p1[ix][iz]-p0[ix][iz]+diff1+diff2;
        }
}
```

Write your own code and run it as a test

You have to

1. create a user directory in /RSFSRC/user/dirname, where dirname is the directory name, for example, 'pyang'.
2. copy a SConstruct from other existing users, and use it as a template to create your own things. For example, take /RSFSRC/user/psava/SConstruct. Assume you are going to do C programming to generate a target executable `sfmodeling2d`. You need to empty all other code list while add a name in C code list:

```
import os, sys

try:
    import bldutil
    glob_build = True # scons command launched in RSFSRC
    srcroot = '../..'
    Import('env bindir libdir pkgdir')
    env = env.Clone()
except:
    glob_build = False # scons in the local directory
    srcroot = os.environ.get('RSFSRC', '../..')
    sys.path.append(os.path.join(srcroot,'framework'))
    import bldutil
    env = bldutil.Debug() # Debugging flags for compilers
    bindir = libdir = pkgdir = None
    SConscript(os.path.join(srcroot,'api/c/SConstruct'))

targets = bldutil.UserSconsTargets()

# C mains
targets.c = '''
modeling2d
'''

  targets.build_all(env, glob_build, srcroot,
                    bindir, libdir, pkgdir)
```

3. Use text editor (emacs, gedit, ...) to create a file Mmodeling2d.c (which will generate the target `sfmodeling2d`, `M` will be automatically replaced by `sf`). In Mmodeling2d.c, start your codes by including the RSF header file: `#include <rsf.h>`, which defines many useful interfaces/subroutines for the convenience of data I/O (including parameters and files)

   ```
   sf_input()/sf_output()
   sf_histint()/sf_histfloat()
   sf_getint()/sf_getfloat()
   ```

   and memory allocation for the variables

   ```
   sf_intalloc()/sf_floatalloc()
   sf_intalloc2()/sf_floatalloc2()
   ...
   ```

   which will be used frequently when coding with Madagascar.

4. specify your input and output files, and initialize Madagascar:

```
sf_file vinit, shots;    /* input and output file*/
sf_init(argc,argv);      /* initialize Madagascar */
vinit=sf_input ("in");   /* initial velocity model in m/s */
shots=sf_output("out");  /* output image */
```

   Here the input file `vinit` is the velocity model, while the output `shots` is a shot gather (or many shots) collected at many receivers for different sources.

5. read the parameters from the input file using the interfaces Madagascar prepared: `sf_hist*(),sf_get*()`

```
/* get parameters for forward modeling */
if (!sf_histint(vinit,"n1",&nz)) sf_error("no n1");
if (!sf_histint(vinit,"n2",&nx)) sf_error("no n2");
if (!sf_histfloat(vinit,"d1",&dz)) sf_error("no d1");
if (!sf_histfloat(vinit,"d2",&dx)) sf_error("no d2");

if (!sf_getfloat("fm",&fm)) fm=10;
/* dominant freq of ricker */
if (!sf_getfloat("dt",&dt)) sf_error("no dt");
/* time interval */
if (!sf_getint("nt",&nt))   sf_error("no nt");
/* total modeling time steps */
if (!sf_getint("ns",&ns))   sf_error("no ns");
/* total shots */
if (!sf_getint("ng",&ng))   sf_error("no ng");
/* total receivers in each shot */
if (!sf_getint("jsx",&jsx))   sf_error("no jsx");
/* source x-axis  jump interval  */
if (!sf_getint("jsz",&jsz))   jsz=0;
/* source z-axis jump interval  */
if (!sf_getint("jgx",&jgx))   jgx=1;
/* receiver x-axis jump interval */
if (!sf_getint("jgz",&jgz))   jgz=0;
/* receiver z-axis jump interval */
if (!sf_getint("sxbeg",&sxbeg))   sf_error("no sxbeg");
/* x-begining index of sources, starting from 0 */
if (!sf_getint("szbeg",&szbeg))   sf_error("no szbeg");
/* z-begining index of sources, starting from 0 */
if (!sf_getint("gxbeg",&gxbeg))   sf_error("no gxbeg");
/* x-begining index of receivers, starting from 0 */
if (!sf_getint("gzbeg",&gzbeg))   sf_error("no gzbeg");
/* z-begining index of receivers, starting from 0 */
if (!sf_getbool("csdgather",&csdgather)) csdgather=false;
/* default, common shot-gather; if n, record at every point*/
```

6. specify the parameters for the output file using the interfaces Madagascar prepared: `sf_put*()`

```
/* put the labels, legends and parameters in output */
sf_putint(shots,"n1",nt);
sf_putint(shots,"n2",ng);
sf_putint(shots,"n3",ns);
sf_putfloat(shots,"d1",dt);
sf_putfloat(shots,"d2",jgx*dx);
sf_putfloat(shots,"o1",0);
sf_putstring(shots,"label1","Time");
sf_putstring(shots,"label2","Lateral");
sf_putstring(shots,"label3","Shot");
sf_putstring(shots,"unit1","sec");
sf_putstring(shots,"unit2","m");
sf_putfloat(shots,"amp",amp);
sf_putfloat(shots,"fm",fm);
sf_putint(shots,"ng",ng);
sf_putint(shots,"szbeg",szbeg);
sf_putint(shots,"sxbeg",sxbeg);
sf_putint(shots,"gzbeg",gzbeg);
sf_putint(shots,"gxbeg",gxbeg);
sf_putint(shots,"jsx",jsx);
sf_putint(shots,"jsz",jsz);
sf_putint(shots,"jgx",jgx);
sf_putint(shots,"jgz",jgz);
sf_putint(shots,"csdgather",csdgather?1:0);
```

7. allocate memory for the arries, format: `sf_floatalloc2(n1,n2)`

```
/* allocate the variables */
wlt=(float*)malloc(nt*sizeof(float));
bndr=(float*)malloc(nt*(2*nz+nx)*sizeof(float));
dobs=(float*)malloc(ng*nt*sizeof(float));
trans=(float*)malloc(ng*nt*sizeof(float));
vv=sf_floatalloc2(nz, nx);
p0=sf_floatalloc2(nz, nx);
p1=sf_floatalloc2(nz, nx);
p2=sf_floatalloc2(nz, nx);
sxz=(int*)malloc(ns*sizeof(int));
gxz=(int*)malloc(ng*sizeof(int));
```

8. do your own computation (forward simulation) as usual. The whole time stepping looks like

```
    memset(p0[0],0,nz*nx*sizeof(float));
    memset(p1[0],0,nz*nx*sizeof(float));
    memset(p2[0],0,nz*nx*sizeof(float));
    /* forward modeling */
```

```
  for(it=0; it<nt; it++)
  {
    add_source(p1, &wlt[it], &sxz[is], 1, nz, true);
    step_forward(p0, p1, p2, vv, dtz, dtx, nz, nx);
    ptr=p0; p0=p1; p1=p2; p2=ptr;
    record_seis(&dobs[it*ng], gxz, p0, ng, nz);
  }
```

9. free the variables

```
/* free the variables */
free(sxz);
free(gxz);
free(bndr);
free(dobs);
free(trans);
free(wlt);
free(*vv); free(vv);
free(*p0); free(p0);
free(*p1); free(p1);
free(*p2); free(p2);
```

10. Finally, you end up with a complete code /RSFSRC/user/pyang/Mmodeling2d.c. You can go into directory RSFSRC, compile and install the target executable `sfmodeling2d`:

```
cd $RSFSRC
scons install
```

If there exists any error in your code, you will get the reporting message in the terminal.

A 2-D Modeling experiment using SConstruct

1. Invoke RSF module to create a experiment environment.

```
  from rsf.proj import *

  End()
```

Almost every SConstruct for numerical test has to start with `from rsf.proj import *` and ends with `End()`.

2. In between, add several lines to create a very simple velocity model including 3 layers using `Flow(target,source,command)`. The command here is `sfmath`. One may type the command name '`sfmath`' to look up the manual in the terminal.

3. Performing 2-D forward simulation with 1 point source (a Ricker wavelet) by specifying the parameters for the command `sfmodeling2d`. Again, one may type the command name to look up the manual in the terminal if you cannot keep everything in mind.

4. Polish your resulting figures. Help yourself by self-documentation of the command, type the command name `sfgrey` or `man sfgrey` [1] in the terminal. For example, try color style: `color=g bartype=h`

The final SConstruct looks like

```
from rsf.proj import *

Flow('vel0',None,
         '''
         math output=1.6 n1=50 n2=200 d1=0.005 d2=0.005
         label1=x1 unit1=km label2=x2 unit2=km
         ''')

Flow('vel1',None,
         '''
         math output=1.8 n1=50 n2=200 d1=0.005 d2=0.005
         label1=x1 unit1=km label2=x2 unit2=km
         ''')
Flow('vel2',None,
         '''
         math output=2.0 n1=100 n2=200 d1=0.005 d2=0.005
         label1=x1 unit1=km label2=x2 unit2=km
         ''')
Flow('vel',['vel0','vel1','vel2'], 'cat axis=1 ${SOURCES[1:3]}')

Result('vel',
        '''
        grey title="velocity model: 3 layers"
        color=j scalebar=y bartype=v
        ''')

Flow('shots','vel',
         '''
         modeling2d nt=1100 dt=0.001 ns=10 ng=200
         sxbeg=5 szbeg=2 jsx=20 jsz=0
         gxbeg=0 gzbeg=3 jgx=1 jgz=0
         ''')

Result('shots',
         '''
        byte allpos=n gainpanel=all |
        grey3 flat=n frame1=300 frame2=100 frame3=5
        label1=Time unit1=s
        label2="Receiver no." label3="Shot no."
        title="Shot records" point1=0.8 point2=0.8
```

[1] Keep in mind the initial 'sf' has to be included when looking for the functionality of the command, although it can be omitted in SConstruct.

```
             ''')

Plot('shots','grey title=Shots',view=1)

#use sfwindow to select 5-th shot and display it using sfgrey


End()
```

We obtain the velocity model in Figure 1 and the shot gathers in Figure 2. To have a look at the movie by looping over each shot gather, run `scons`.

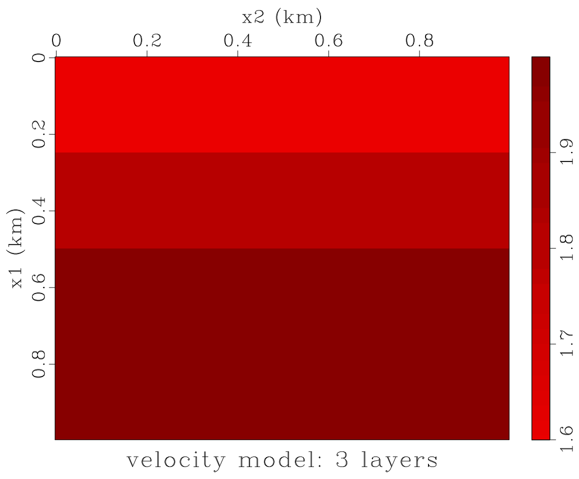

Figure 1: A 3-layer velocity model modeling2d/ vel

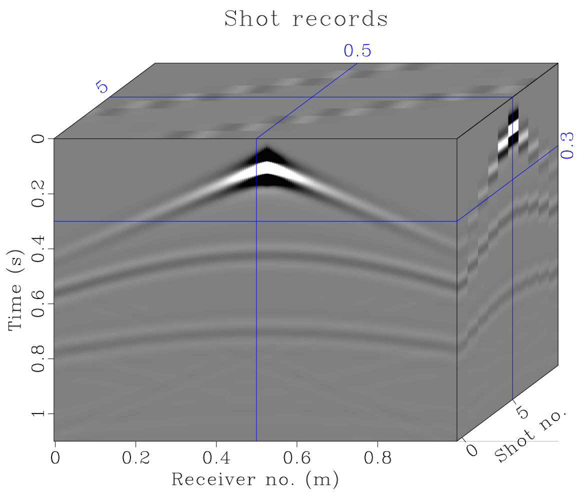

Figure 2: Shot gather modeling2d/ shots

Further exercises

How to:

1. higher accuracy in space $\rightarrow$ higher order FD stencil $\rightarrow$ Fourier pseudospectral method (Carcione, 2010), see the code /RSFSRC/user/pyang/Mps2d.c?
2. implement sponge/Gaussian taper boundary condition (Cerjan et al., 1985), PML (Komatitsch and Martin, 2007)? (Yang, 2014)
3. increase temporal discretization accuracy: leap-frog $\rightarrow$ Runge-Kutta scheme?
4. locate your source and receiver position at arbitrary position: Kaiser windowed sinc interpolation (Hicks, 2002)

FULL WAVEFORM INVERSION

FWI is a nonlinear iterative minimization process by matching the waveform between the synthetic data and the observed seismograms (Tarantola, 1984; Virieux and Operto, 2009). In least-squares sense, the misfit functional of FWI reads

$$C(m) = \frac{1}{2}\|R_r p - d\|^2, \tag{5}$$

where m is the model parameter (i.e. the velocity) in model space; R_r is a restriction operator mapping the wavefield onto the receiver locations; $d := d(x_r, t)$ is the observed seismogram at receiver location x_r while $p := p(x, t)$ is the synthetic wavefield whose adjoint wavefield $\bar{p}$ is given by

$$(\frac{1}{v^2}\partial_t^2 - \nabla)\bar{p} = -\frac{\partial C}{\partial p} = -R_r^{\dagger}(R_r p - d) \tag{6}$$

which indicates that the adjoint wave equation is exactly the same as the forward wave equation except that the adjoint source is data residual backprojected into the wavefield. In each iteration the model has to be updated following a Newton descent direction Δm^k

$$m^{k+1} = m^k + \gamma_k \Delta m^k, \tag{7}$$

with a stepsize γ_k. Away from the sources ($f = 0$), the gradient can be computed by

$$\nabla C = -\frac{2}{v^3}\int_T \mathrm{d}t \bar{p}\partial_t^2 p = -\frac{2}{v}\int_T \mathrm{d}t \bar{p}\nabla^2 p \tag{8}$$

Reconstruct your source/incident wavefield backwards in time

As one can see from above, the computation of FWI gradient requires simultaneously accessing the source/incident wavefield and the adjoint wavefield at each time step. To achieve this goal, we may store the absorbing boundary at each time step when doing forward simulation, and then reconstruct the incident wavefield backwards in time using the stored boundary. According to equation (2), the reconstruction is easy

$$p^{k-1} = 2p^k - p^{k+1} + \Delta t^2 v^2 \nabla^2 p^k \tag{9}$$

Therefore, we may employ the same subroutine by switching the role of wavefield p^{k+1} and p^{k-1}. The elements in the boundary does not follow the above equation but we can store it in forward simulation and re-inject them for backward reconstruction.

1. Redo forward modeling using the same model while storing the boundary at each time step

```
for(it=0; it<nt; it++){
  add_source(p1, &wlt[it], &sxz[is], 1, nz, true);
  step_forward(p0, p1, p2, vv, dtz, dtx, nz, nx);
  ptr=p0; p0=p1; p1=p2; p2=ptr;
  rw_bndr(&bndr[it*(2*nz+nx)], p0, nz, nx, true);
  record_seis(&dobs[it*ng], gxz, p0, ng, nz);

  if(it==kt){
    sf_floatwrite(p0[0],nz*nx, check1);
  }
}
```

2. reverse propagate the incident wavefield backwards from final snapshot and stored boundary: at each time step, inject the corresponding boundary

```
ptr=p0; p0=p1; p1=ptr;
for(it=nt-1; it>-1; it--){
  rw_bndr(&bndr[it*(2*nz+nx)], p1, nz, nx, false);

  if(it==kt){
    sf_floatwrite(p1[0],nz*nx, check2);
  }
  step_backward(p0, p1, p2, vv, dtz, dtx, nz, nx);
  add_source(p1, &wlt[it], &sxz[is], 1, nz, false);
  ptr=p0; p0=p1; p1=p2; p2=ptr;
}
```

3. check whether you perfectly reconstruct your incident wavefield at any time step `kt`, as shown in Figure 3.

The final SConstruct looks like

```
from rsf.proj import *

Flow('vel0',None,
        '''
        math output=1.6 n1=50 n2=200 d1=0.005 d2=0.005
        label1=x1 unit1=km label2=x2 unit2=km
        ''')

Flow('vel1',None,
```

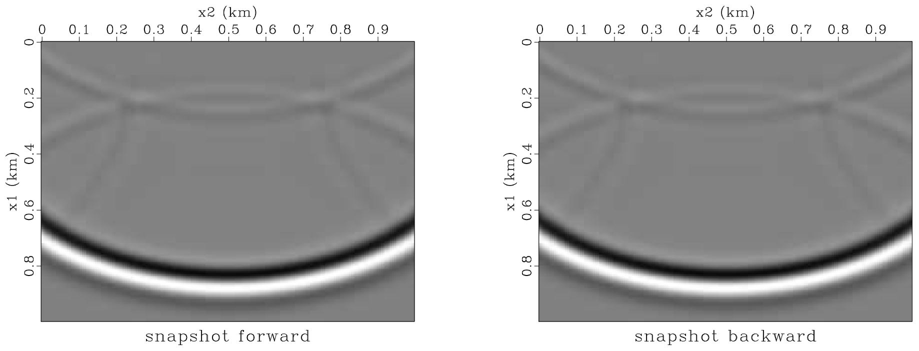

Figure 3: The backward reconstructed wavefield is the same as the incident wavefield fbrec2d/ check

```
        '''
        math output=1.8 n1=50 n2=200 d1=0.005 d2=0.005
        label1=x1 unit1=km label2=x2 unit2=km
        ''')
Flow('vel2',None,
        '''
        math output=2.0 n1=100 n2=200 d1=0.005 d2=0.005
        label1=x1 unit1=km label2=x2 unit2=km
        ''')
Flow('vel',['vel0','vel1','vel2'], 'cat axis=1 ${SOURCES[1:3]}')


Flow('shot check1 check2','vel',
        '''
        sffbrec2d check1=${TARGETS[1]} check2=${TARGETS[2]}
        csdgather=n fm=15 dt=0.001 ns=1 ng=200 nt=1100 ng=200 kt=550
        sxbeg=100 szbeg=2 jsx=37 jsz=0
        gxbeg=0 gzbeg=1 jgx=1 jgz=0
        ''')

Result('shot','grey gainpanel=all title=shot')


Flow('diff','check1 check2','sfadd ${SOURCES[1]} scale=1,-1')


Plot('check1','grey title="snapshot forward" scalebar=y')
Plot('check2','grey title="snapshot backward" scalebar=y')
Plot('diff','grey title="difference" scalebar=y')

Result('check','check1 check2 diff','SideBySideIso')

```

```
End()
```

Do a synthetic FWI test using Marmousi model

It is convenient to perform adjoint simulation when reconstructing the incident wavefield backwards. The computation of FWI gradient can be done on the fly by adding several lines:

```
memset(sp0[0], 0, nz*nx*sizeof(float));
memset(sp1[0], 0, nz*nx*sizeof(float));
for(it=0; it<nt; it++)
{
  add_source(sp1, &wlt[it], &sxz[is], 1, nz, true);
  step_forward(sp0, sp1, sp2, vv, dtz, dtx, nz, nx);
  ptr=sp0; sp0=sp1; sp1=sp2; sp2=ptr;
  rw_bndr(&bndr[it*(2*nz+nx)], sp0, nz, nx, true);

  record_seis(dcal, gxz, sp0, ng, nz);
  cal_residuals(dcal, &dobs[it*ng], &derr[is*ng*nt+it*ng], ng);
}

ptr=sp0; sp0=sp1; sp1=ptr;
memset(gp0[0], 0, nz*nx*sizeof(float));
memset(gp1[0], 0, nz*nx*sizeof(float));
for(it=nt-1; it>-1; it--)
{
  rw_bndr(&bndr[it*(2*nz+nx)], sp1, nz, nx, false);
  step_backward(illum, lap, sp0, sp1, sp2, vv, dtz, dtx, nz, nx);
  add_source(sp1, &wlt[it], &sxz[is], 1, nz, false);

  add_source(gp1, &derr[is*ng*nt+it*ng], gxz, ng, nz, true);
  step_forward(gp0, gp1, gp2, vv, dtz, dtx, nz, nx);

  cal_gradient(g1, lap, gp1, nz, nx);
  ptr=sp0; sp0=sp1; sp1=sp2; sp2=ptr;
  ptr=gp0; gp0=gp1; gp1=gp2; gp2=ptr;
}
```

Of course, to apply the steepest descent method for minimization of the misfit function for waveform inversion, a step length has to be determined at each iteration. You may use the estimation proposed by Pica et al. (1990,in the appendix) in your FWI code as `RSFSRC/user/pyang/Mfwi2d.c`.

The workflow for synthetic FWI test based on Marmousi model follows the several steps:

1. Obtain the Marmousi velocity model, which can be downloaded by `Fetch('marmvel.hh','marm')` in SConstruct, as shown in the top panel of Figure 4.

2. We may start to generate the observed seismograms/shots using resampled Marmousi, as shown in Figure 5.

3. By smoothing the true model using triangular window many times, an initial model plotted in the bottom panel of Figure 4 is generated to do the FWI test.

4. Using the observed seismograms/shots from true model, we start the inversion with the rough initial model.

5. You may appreciate your the inverted velocity during the iterations in Figure 6, as well as the variations of the misfit function in Figure 7.

A complete SConstruct for the above workflow appears in the following:

```
from rsf.proj import *

# marmvel.hh contains Marmousi model which can
# be downloaded from the server using Fetch.
Fetch('marmvel.hh','marm')

Flow('vel','marmvel.hh',
        '''
        dd form=native | window j1=8 j2=8 | sfsmooth rect1=3 rect2=3|
        put label1=Depth  unit1=m label2=Lateral unit2=m
        ''')
Plot('vel',
        '''
        grey color=j mean=y title="Marmousi model"
        scalebar=y bartype=v barlabel="V"
        barunit="m/s" screenratio=0.45 color=j labelsz=10 titlesz=12
        ''')

Flow('shots','vel',
        '''
        sfmodeling2d csdgather=n fm=4 amp=1 dt=0.0015 ns=7 ng=288 nt=2800
        sxbeg=4 szbeg=2 jsx=45 jsz=0 gxbeg=0 gzbeg=3 jgx=1 jgz=0
        ''')
Plot('shots','grey color=g title=shot label2= unit2=',view=1)


Plot('shot1','shots',
     'window n3=1 f3=0| grey title=shot1 label2=Lateral unit2=m')
Plot('shot3','shots',
     'window n3=1 f3=2| grey title=shot3 label2=Lateral unit2=m')
Plot('shot5','shots',
     'window n3=1 f3=4| grey title=shot5 label2=Lateral unit2=m')
Plot('shot7','shots',
     'window n3=1 f3=6| grey title=shot7 label2=Lateral unit2=m')
Result('shotsnap','shot1 shot3 shot5 shot7',
        'SideBySideAniso',vppen='txscale=2.')

# smoothed velocity model
Flow('smvel','vel','smooth repeat=6  rect1=8 rect2=10')
Plot('smvel',
     '''
     grey title="Initial model" wantitle=y allpos=y color=j
     pclip=100 scalebar=y bartype=v barlabel="V" barunit="m/s"
         screenratio=0.45 color=j labelsz=10 titlesz=12
     ''' )

Result('marm','vel smvel','TwoRows')

# use the over-smoothed model as initial model for FWI
Flow('vsnaps grads objs illums','smvel shots',
        '''
        sffwi2d shots=${SOURCES[1]}
        grads=${TARGETS[1]} objs=${TARGETS[2]}
        illums=${TARGETS[3]} niter=10 precon=y rbell=1
        ''')
Result('vsnaps',
        '''
        grey title="Updated velocity" allpos=y color=j pclip=100
        scalebar=y bartype=v barlabel="V" barunit="m/s"
        ''')
Plot('vsnap1','vsnaps',
        '''
        window n3=1|grey title="Updated velocity, iter=1"
        allpos=y color=j pclip=100 labelsz=10 titlesz=12
        scalebar=y bartype=v barlabel="V" barunit="m/s"
        ''')
Plot('vsnap2','vsnaps',
        '''
        window n3=1 f3=1|grey title="Updated velocity, iter=2"
        allpos=y color=j pclip=100 labelsz=10 titlesz=12
        scalebar=y bartype=v barlabel="V" barunit="m/s"
        ''')
Plot('vsnap4','vsnaps',
        '''
        window n3=1 f3=3|grey title="Updated velocity, iter=4"
        allpos=y color=j pclip=100 labelsz=10 titlesz=12
        scalebar=y bartype=v barlabel="V" barunit="m/s"
        ''')

Plot('vsnap6','vsnaps',
        '''
        window n3=1 f3=5|grey title="Updated velocity, iter=6"
         allpos=y color=j pclip=100 labelsz=10 titlesz=12
        scalebar=y bartype=v barlabel="V" barunit="m/s"
        ''')
Plot('vsnap8','vsnaps',
        '''
        window n3=1 f3=7|grey title="Updated velocity, iter=8"
        allpos=y color=j pclip=100 labelsz=10 titlesz=12
        scalebar=y bartype=v barlabel="V" barunit="m/s"
        ''')
Plot('vsnap10','vsnaps',
        '''
        window n3=1 f3=9|grey title="Updated velocity, iter=10"
        allpos=y color=j pclip=100 labelsz=10 titlesz=12
        scalebar=y bartype=v barlabel="V" barunit="m/s"
        ''')

Result('vsnap','vsnap1 vsnap2 vsnap4 vsnap6 vsnap8 vsnap10',
       'TwoRows')

Result('objs',
        '''
        sfput n2=1 label1=Iteration unit1= unit2= label2= |
        graph title="Misfit function" dash=0 plotfat=5
        grid=y yreverse=n
        ''')

End()
```

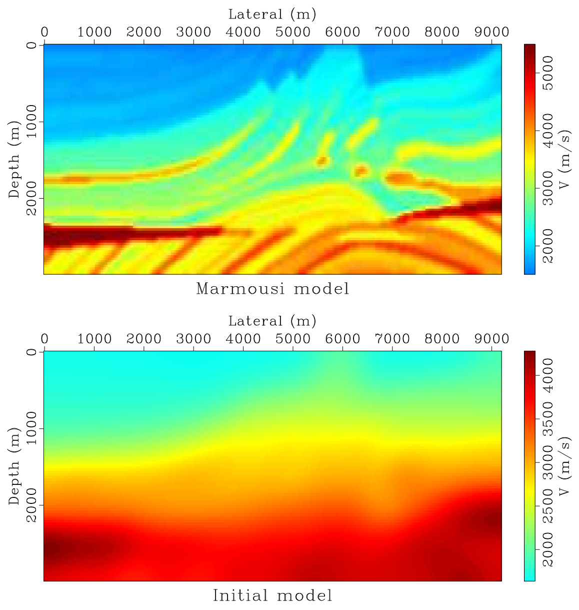

Figure 4: Top: True Marmousi model; bottom: Initial model for FWI marmtest/ marm

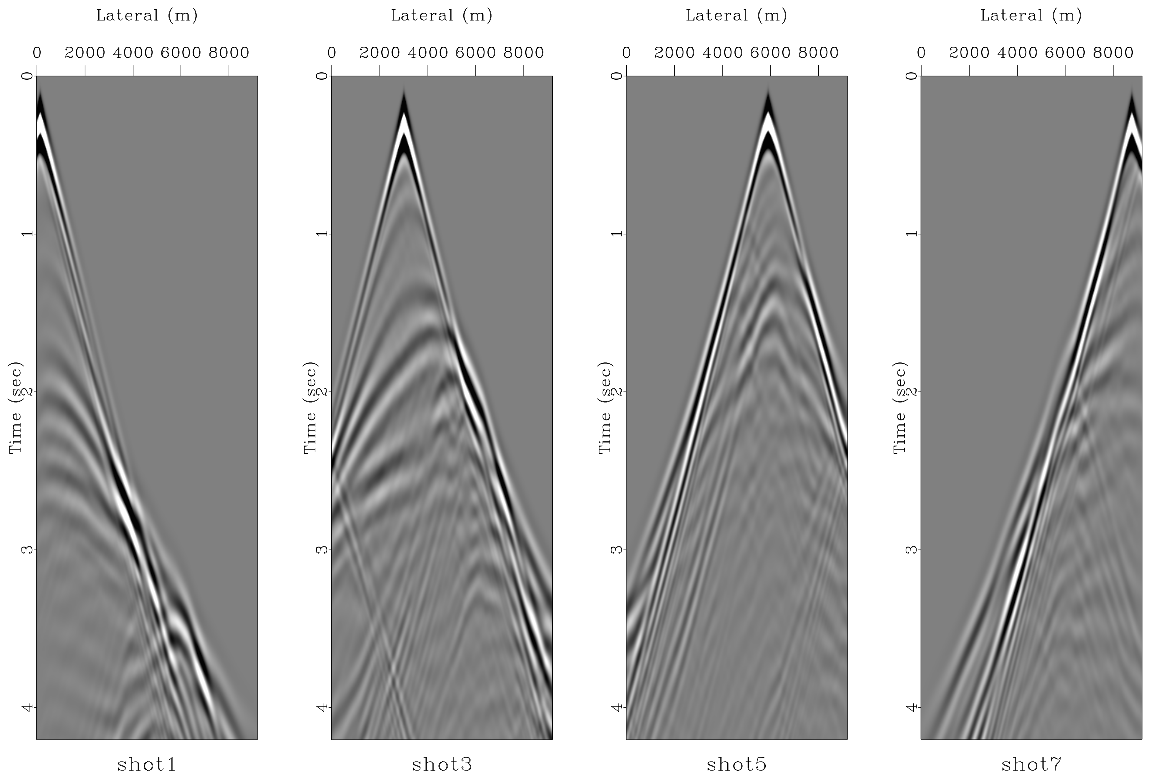

Figure 5: Shots from true Marmousi model marmtest/ shotsnap

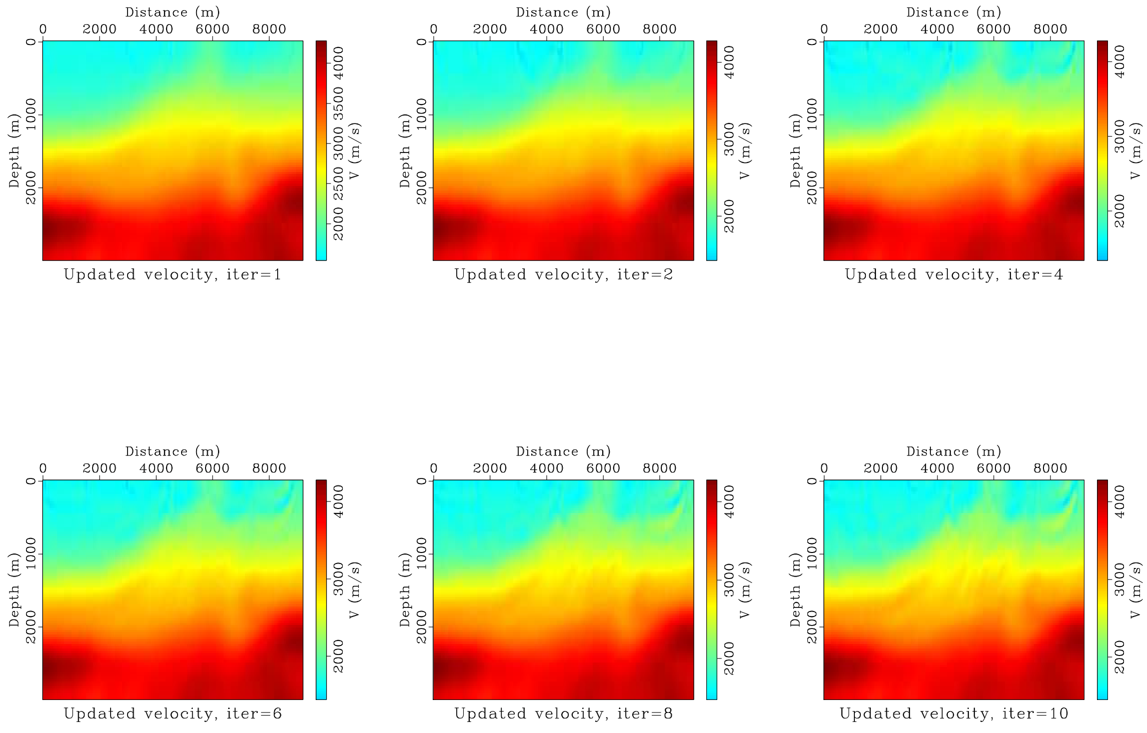

Figure 6: Inverted velocity during iterations marmtest/ vsnap

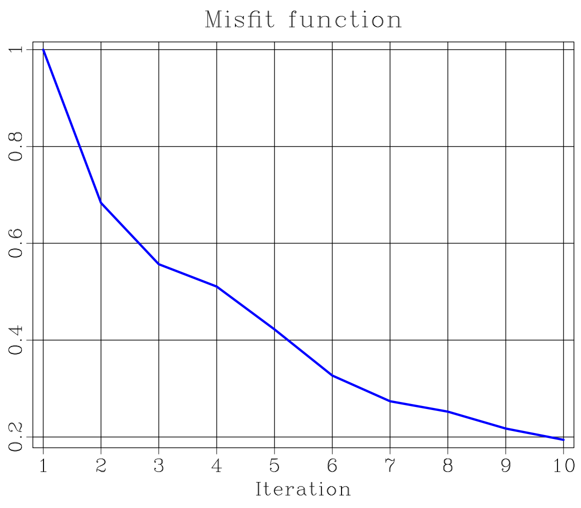

Figure 7: The misfit function decreases during iterations marmtest/ objs

Your exercise

- It works so slow! How to speeed up? $\rightarrow$ CUDA (Yang et al., 2015,Mgpufwi.cu) + MPI high performance computing (Jeff)?
- How to improve the poor resulting velocity model in Figure 6? A better initial model by less smoothing? Increase the number of iterations? Estimate a good step length to satisfy the Wolf condition? Estimate Hessian $\rightarrow$ Truncated newton (Métivier et al., 2014)+ Source encoding+ Good preconditioning?
- Derive the adjoint equation for first order wave equation system

$$\begin{cases} \partial_t p = \kappa \nabla \cdot \mathbf{v} + f_p \\ \rho \partial_t \mathbf{v} = \nabla p \end{cases} \tag{10}$$

  where $\kappa = \rho v^2$.
- Write a forward simulation code based on sponge boundary condition using the above system
- Derive your gradient expressions for velocity and density
- code your multiparameter FWI for inverting v and ρ

FURTHER THINKING

- The storage complexity using regular grid FD and staggered grid FD stencil?
- How to reduce the storage requirement for a 3-D volume? CFL $\rightarrow$ Nyquist: decimation+ interpolation (Yang et al., 2016c,d)

- How to derive the adjoint state equation? $\rightarrow$ Lagrange multiplier+cost function (Plessix, 2006)? What is the adjoint state equation for 1st order acoustic wave equation, viscoacoustic system, viscoelastic system (Yang et al., 2016a)?
- Is it possible to do reverse propagation in attenuating medium? How to handle instability? Binomial checkpointing$\rightarrow$ CARFS (checkpointing-assisted reverse-forward simulation) (Yang et al., 2016b)?
- What if FWI using other norms/misfit function C? Only change the adjoint source $\frac{\partial C}{\partial p}$?

CONCLUSION

1. No answer sheet for your exercises!
2. Too many open questions in FWI: good initial model? Misfit function immune to cycle-skipping issue? Better preconditioning? Inverting attenuating?
3. FWI is a research field waiting for your addition!

REFERENCES

Carcione, J. M. (2010). A generalization of the fourier pseudospectral method. *Geophysics*, 75(6):A53–A56.

Cerjan, C., Kosloff, D., Kosloff, R., and Reshef, M. (1985). A nonreflecting boundary condition for discrete acoustic and elastic wave equations. *Geophysics*, 50(4):2117–2131.

Clayton, R. and Engquist, B. (1977). Absorbing boundary conditions for acoustic and elastic wave equations. *Bulletin of the Seismological Society of America*, 67:1529–1540.

Hicks, G. J. (2002). Arbitrary source and receiver positioning in finite-difference schemes using Kaiser windowed sinc functions. *Geophysics*, 67:156–166.

Komatitsch, D. and Martin, R. (2007). An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation. *Geophysics*, 72(5):SM155–SM167.

Métivier, L., Bretaudeau, F., Brossier, R., Operto, S., and Virieux, J. (2014). Full waveform inversion and the truncated Newton method: quantitative imaging of complex subsurface structures. *Geophysical Prospecting*, 62:1353–1375.

Pica, A., Diet, J. P., and Tarantola, A. (1990). Nonlinear inversion of seismic reflection data in laterally invariant medium. *Geophysics*, 55(3):284–292.

Plessix, R. E. (2006). A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophysical Journal International*, 167(2):495–503.

Tarantola, A. (1984). Inversion of seismic reflection data in the acoustic approximation. *Geophysics*, 49(8):1259–1266.

Virieux, J. and Operto, S. (2009). An overview of full waveform inversion in exploration geophysics. *Geophysics*, 74(6):WCC1–WCC26.

Yang, P. (2014). A numerical tour of wave propagation. Technical report, Xi'an Jiaotong University.

Yang, P., Brossier, R., Métivier, L., and Virieux, J. (2016a). A systematic formulation of 3D multiparameter full waveform inversion in viscoelastic medium. *submitted to Geophysical Journal International*.

Yang, P., Brossier, R., Métivier, L., and Virieux, J. (2016b). Wavefield reconstruction in attenuating media: A checkpointing-assisted reverse-forward simulation method. *submitted to Geophysics*.

Yang, P., Brossier, R., and Virieux, J. (2016c). Downsampling plus interpolation for wavefield reconstruction by reverse propagation. In *78th EAGE Conference & Exhibition Expanded Abstracts*, number SBT5 08.

Yang, P., Brossier, R., and Virieux, J. (2016d). Wavefield reconstruction from significantly decimated boundaries. *Geophysics, Accepted*.

Yang, P., Gao, J., and Wang, B. (2015). A graphics processing unit implementation of time-domain full-waveform inversion. *Geophysics*, 80(3):F31–F39.