Guide to programming using RSF

Paul Sava[1]

ABSTRACT

This guide demonstrates a simple time-domain finite-differences modeling code in RSF.

INTRODUCTION

This section presents time-domain finite-difference modeling [2] written with the RSF library. The program is demonstrated with the C, C++ and Fortran 90 interfaces.

The acoustic wave-equation

$$\Delta U - \frac{1}{v^2}\frac{\partial^2 U}{\partial t^2} = f(t) \tag{1}$$

can be written as

$$\left[\Delta U - f(t)\right] v^2 = \frac{\partial^2 U}{\partial t^2} \ . \tag{2}$$

Δ is the Laplacian symbol, $f(t)$ is the source wavelet, v is the velocity, and U is a scalar wavefield.

A discrete time-step involves the following computations:

$$U_{i+1} = \left[\Delta U - f(t)\right] v^2 \Delta t^2 + 2U_i - U_{i-1} \ , \tag{3}$$

where U_{i-1}, U_i and U_{i+1} represent the propagating wavefield at various time steps.

[1]**e-mail:** paul.sava@beg.utexas.edu
[2]"Hello world" of seismic imaging.

C PROGRAM

```c
/* time-domain acoustic FD modeling */
#include <rsf.h>
int main(int argc, char* argv[])
{
    /* Laplacian coefficients */
    float c0=-30./12.,c1=+16./12.,c2=- 1./12.;

    bool verb;             /* verbose flag */
    sf_file Fw=NULL,Fv=NULL,Fr=NULL,Fo=NULL; /* I/O files */
    sf_axis at,az,ax;      /* cube axes */
    int it,iz,ix;          /* index variables */
    int nt,nz,nx;
    float dt,dz,dx,idx,idz,dt2;

    float  *ww,**vv,**rr;      /* I/O arrays*/
    float **um,**uo,**up,**ud;/* tmp arrays */

    sf_init(argc,argv);
    if(! sf_getbool("verb",&verb)) verb=0; /* verbose flag */

    /* setup I/O files */
    Fw = sf_input ("in" );
    Fo = sf_output("out");
    Fv = sf_input ("vel");
    Fr = sf_input ("ref");

    /* Read/Write axes */
    at = sf_iaxa(Fw,1); nt = sf_n(at); dt = sf_d(at);
    az = sf_iaxa(Fv,1); nz = sf_n(az); dz = sf_d(az);
    ax = sf_iaxa(Fv,2); nx = sf_n(ax); dx = sf_d(ax);

    sf_oaxa(Fo,az,1);
    sf_oaxa(Fo,ax,2);
    sf_oaxa(Fo,at,3);

    dt2 =    dt*dt;
    idz = 1/(dz*dz);
    idx = 1/(dx*dx);

    /* read wavelet, velocity & reflectivity */
    ww=sf_floatalloc(nt);     sf_floatread(ww   ,nt   ,Fw);
    vv=sf_floatalloc2(nz,nx); sf_floatread(vv[0],nz*nx,Fv);
    rr=sf_floatalloc2(nz,nx); sf_floatread(rr[0],nz*nx,Fr);

    /* allocate temporary arrays */
    um=sf_floatalloc2(nz,nx);
    uo=sf_floatalloc2(nz,nx);
    up=sf_floatalloc2(nz,nx);
    ud=sf_floatalloc2(nz,nx);

    for (ix=0; ix<nx; ix++) {
        for (iz=0; iz<nz; iz++) {
            um[ix][iz]=0;
            uo[ix][iz]=0;
            up[ix][iz]=0;
            ud[ix][iz]=0;
        }
    }

    /* MAIN LOOP */
    if(verb) fprintf(stderr,"\n");
    for (it=0; it<nt; it++) {
        if(verb) fprintf(stderr,"\b\b\b\b\b%d",it);

        /* 4th order laplacian */
        for (iz=2; iz<nz-2; iz++) {
            for (ix=2; ix<nx-2; ix++) {
                ud[ix][iz] =
                    c0* uo[ix  ][iz  ] * (idx+idz) +
                    c1*(uo[ix-1][iz  ] + uo[ix+1][iz  ])*idx +
                    c2*(uo[ix-2][iz  ] + uo[ix+2][iz  ])*idx +
                    c1*(uo[ix  ][iz-1] + uo[ix  ][iz+1])*idz +
                    c2*(uo[ix  ][iz-2] + uo[ix  ][iz+2])*idz;
            }
        }

        /* inject wavelet */
        for (iz=0; iz<nz; iz++) {
            for (ix=0; ix<nx; ix++) {
                ud[ix][iz] -= ww[it] * rr[ix][iz];
            }
        }

        /* scale by velocity */
        for (iz=0; iz<nz; iz++) {
            for (ix=0; ix<nx; ix++) {
                ud[ix][iz] *= vv[ix][iz]*vv[ix][iz];
            }
        }
```

```

        /* time step */
        for (iz=0; iz<nz; iz++) {
            for (ix=0; ix<nx; ix++) {
                up[ix][iz] =
                    2*uo[ix][iz]
                    - um[ix][iz]
                    + ud[ix][iz] * dt2;

                um[ix][iz] = uo[ix][iz];
                uo[ix][iz] = up[ix][iz];
            }
        }

        /* write wavefield to output */
        sf_floatwrite(uo[0],nz*nx,Fo);
    }
    if(verb) fprintf(stderr,"\n");

    exit (0);
}
```

- Declare input, output and auxiliary file tags: `Fw` for input wavelet, `Fv` for velocity, `Fr` for reflectivity, and `Fo` for output wavefield.

```
    sf_file Fw=NULL,Fv=NULL,Fr=NULL,Fo=NULL; /* I/O files */
```

- Declare RSF cube axes: `at` time axis, `ax` space axis, `az` depth axis.

```
    sf_axis at,az,ax;    /* cube axes */
```

- Declare multi-dimensional arrays for input, output and computations.

```

    float *ww,**vv,**rr;     /* I/O arrays*/
```

- Open files for input/output.

```
    Fw = sf_input ("in" );
    Fo = sf_output("out");
    Fv = sf_input ("vel");
    Fr = sf_input ("ref");
```

- Read axes from input files; write axes to output file.

```
    at = sf_iaxa(Fw,1); nt = sf_n(at); dt = sf_d(at);
    az = sf_iaxa(Fv,1); nz = sf_n(az); dz = sf_d(az);
    ax = sf_iaxa(Fv,2); nx = sf_n(ax); dx = sf_d(ax);

    sf_oaxa(Fo,az,1);
    sf_oaxa(Fo,ax,2);
    sf_oaxa(Fo,at,3);
```

- Allocate arrays and read wavelet, velocity and reflectivity.

```
    ww=sf_floatalloc(nt);      sf_floatread(ww   ,nt   ,Fw);
    vv=sf_floatalloc2(nz,nx); sf_floatread(vv[0],nz*nx,Fv);
    rr=sf_floatalloc2(nz,nx); sf_floatread(rr[0],nz*nx,Fr);
```

- Allocate temporary arrays.

```
    um=sf_floatalloc2(nz,nx);
    uo=sf_floatalloc2(nz,nx);
    up=sf_floatalloc2(nz,nx);
    ud=sf_floatalloc2(nz,nx);
```

- Loop over time.

```
    for (it=0; it<nt; it++) {
```

- Compute Laplacian: ΔU.

```
        for (iz=2; iz<nz-2; iz++) {
            for (ix=2; ix<nx-2; ix++) {
                ud[ix][iz] =
                    c0* uo[ix  ][iz  ] * (idx+idz) +
                    c1*(uo[ix-1][iz  ] + uo[ix+1][iz  ])*idx +
                    c2*(uo[ix-2][iz  ] + uo[ix+2][iz  ])*idx +
                    c1*(uo[ix  ][iz-1] + uo[ix  ][iz+1])*idz +
                    c2*(uo[ix  ][iz-2] + uo[ix  ][iz+2])*idz;
            }
        }
```

- Inject source wavelet: $[\Delta U - f(t)]$

```
        for (iz=0; iz<nz; iz++) {
            for (ix=0; ix<nx; ix++) {
                ud[ix][iz] -= ww[it] * rr[ix][iz];
            }
        }
```

- Scale by velocity: $[\Delta U - f(t)]\, v^2$

```
        for (iz=0; iz<nz; iz++) {
            for (ix=0; ix<nx; ix++) {
                ud[ix][iz] *= vv[ix][iz]*vv[ix][iz];
            }
        }
```

- Time step: $U_{i+1} = [\Delta U - f(t)]\, v^2 \Delta t^2 + 2U_i - U_{i-1}$

```
        for (iz=0; iz<nz; iz++) {
            for (ix=0; ix<nx; ix++) {
                up[ix][iz] =
                    2*uo[ix][iz]
                    - um[ix][iz]
                    + ud[ix][iz] * dt2;

                um[ix][iz] = uo[ix][iz];
                uo[ix][iz] = up[ix][iz];
            }
        }
```

C++ PROGRAM

```
// time-domain acoustic FD modeling
#include <valarray>
#include <iostream>
#include <rsf.hh>
#include <cub.hh>
#include <vai.hh>
using namespace std;

int main(int argc, char* argv[])
{
    // Laplacian coefficients
    float c0=-30./12.,c1=+16./12.,c2=- 1./12.;

    sf_init(argc,argv);// init RSF
    bool verb;         // vebose flag
    if(! sf_getbool("verb",&verb)) verb=0;

    // setup I/O files
    CUB Fw("in", "i"); Fw.headin(); //Fw.report();
    CUB Fv("vel","i"); Fv.headin(); //Fv.report();
    CUB Fr("ref","i"); Fr.headin(); //Fr.report();
    CUB Fo("out","o"); Fo.setup(3,Fv.esize());

    // Read/Write axes
    sf_axis at = Fw.getax(0); int nt = sf_n(at); float dt = sf_d(at);
    sf_axis az = Fv.getax(0); int nz = sf_n(az); float dz = sf_d(az);
    sf_axis ax = Fv.getax(1); int nx = sf_n(ax); float dx = sf_d(ax);

    Fo.putax(0,az);
    Fo.putax(1,ax);
    Fo.putax(2,at);
    Fo.headou();

    float dt2 =    dt*dt;
    float idz = 1/(dz*dz);
    float idx = 1/(dx*dx);

    // read wavelet, velocity and reflectivity
    valarray<float> ww( nt    ); ww=0; Fw >> ww;
    valarray<float> vv( nz*nx ); vv=0; Fv >> vv;
    valarray<float> rr( nz*nx ); rr=0; Fr >> rr;

    // allocate temporary arrays
    valarray<float> um(nz*nx); um=0;
    valarray<float> uo(nz*nx); uo=0;
    valarray<float> up(nz*nx); up=0;
    valarray<float> ud(nz*nx); ud=0;

    // init ValArray Index counter
    VAI k(nz,nx);

    // MAIN LOOP
    if(verb) cerr << endl;
    for (int it=0; it<nt; it++) {
        if(verb) cerr << "\b\b\b\b\b" << it;

        // 4th order laplacian
        for (int iz=2; iz<nz-2; iz++) {
            for (int ix=2; ix<nx-2; ix++) {
                ud[k(iz,ix)] =
                    c0* uo[ k(iz  ,ix  )] * (idx+idz) +
                    c1*(uo[ k(iz  ,ix-1)]+uo[ k(iz  ,ix+1)]) * idx +
                    c1*(uo[ k(iz-1,ix  )]+uo[ k(iz+1,ix  )]) * idz +
                    c2*(uo[ k(iz  ,ix-2)]+uo[ k(iz  ,ix+2)]) * idx +
                    c2*(uo[ k(iz-2,ix  )]+uo[ k(iz+2,ix  )]) * idz;
            }
        }

        // inject wavelet
        ud -= ww[it] * rr;

        // scale by velocity
        ud *= vv*vv;

        // time step
        up=(float)2 * uo - um + ud * dt2;
        um =   uo;
        uo =   up;

        // write wavefield to output output
        Fo << uo;
    }
    if(verb) cerr << endl;

    exit(0);
}
```

1. Declare input, output and auxiliary file cubes (of type `CUB`).

```
CUB Fw("in", "i"); Fw.headin(); //Fw.report();
CUB Fv("vel","i"); Fv.headin(); //Fv.report();
CUB Fr("ref","i"); Fr.headin(); //Fr.report();
CUB Fo("out","o"); Fo.setup(3,Fv.esize());
```

2. Declare, read and write RSF cube axes: `at` time axis, `ax` space axis, `az` depth axis.

```
sf_axis at = Fw.getax(0); int nt = sf_n(at); float dt = sf_d(at);
sf_axis az = Fv.getax(0); int nz = sf_n(az); float dz = sf_d(az);
sf_axis ax = Fv.getax(1); int nx = sf_n(ax); float dx = sf_d(ax);

Fo.putax(0,az);
Fo.putax(1,ax);
Fo.putax(2,at);
Fo.headou();
```

3. Declare multi-dimensional `valarrays` for input, output and read data.

```
valarray<float> ww( nt    ); ww=0; Fw >> ww;
valarray<float> vv( nz*nx ); vv=0; Fv >> vv;
valarray<float> rr( nz*nx ); rr=0; Fr >> rr;
```

4. Declare multi-dimensional `valarrays` for temporary storage.

```
valarray<float> um(nz*nx); um=0;
valarray<float> uo(nz*nx); uo=0;
valarray<float> up(nz*nx); up=0;
valarray<float> ud(nz*nx); ud=0;
```

5. Initialize multidimensional `valarray` index counter (of type `VAI`).

```
VAI k(nz,nx);
```

6. Loop over time.

```
for (int it=0; it<nt; it++) {
```

7. Compute Laplacian: ΔU.

```
    for (int iz=2; iz<nz-2; iz++) {
        for (int ix=2; ix<nx-2; ix++) {
            ud[k(iz,ix)] =
                c0* uo[ k(iz  ,ix  )] * (idx+idz) +
                c1*(uo[ k(iz  ,ix-1)]+uo[ k(iz  ,ix+1)]) * idx +
                c1*(uo[ k(iz-1,ix  )]+uo[ k(iz+1,ix  )]) * idz +
                c2*(uo[ k(iz  ,ix-2)]+uo[ k(iz  ,ix+2)]) * idx +
                c2*(uo[ k(iz-2,ix  )]+uo[ k(iz+2,ix  )]) * idz;
        }
    }
```

8. Inject source wavelet: $[\Delta U - f(t)]$

```
    ud -= ww[it] * rr;
```

9. Scale by velocity: $[\Delta U - f(t)]\, v^2$

```
    ud *= vv*vv;
```

10. Time step: $U_{i+1} = [\Delta U - f(t)]\, v^2 \Delta t^2 + 2U_i - U_{i-1}$

```
    up=(float)2 * uo - um + ud * dt2;
    um =   uo;
    uo =   up;
```

FORTRAN 90 PROGRAM

```
! time-domain acoustic FD modeling
program AFDMf90
  use rsf

  implicit none

  ! Laplacian coefficients
  real :: c0=-30./12.,c1=+16./12.,c2=- 1./12.

  logical     :: verb          ! verbose flag
  type(file) :: Fw,Fv,Fr,Fo    ! I/O files
  type(axa)   :: at,az,ax      ! cube axes
  integer     :: it,iz,ix      ! index variables
  real        :: idx,idz,dt2

  real, allocatable :: vv(:,:),rr(:,:),ww(:)            ! I/O arrays
  real, allocatable :: um(:,:),uo(:,:),up(:,:),ud(:,:)  ! tmp arrays

  call sf_init() ! init RSF
  call from_par("verb",verb,.false.)

  ! setup I/O files
  Fw=rsf_input ("in")
  Fv=rsf_input ("vel")
  Fr=rsf_input ("ref")
  Fo=rsf_output("out")

  ! Read/Write axes
  call iaxa(Fw,at,1); call iaxa(Fv,az,1); call iaxa(Fv,ax,2)
  call oaxa(Fo,az,1); call oaxa(Fo,ax,2); call oaxa(Fo,at,3)

  dt2 =    at%d*at%d
  idz = 1/(az%d*az%d)
  idx = 1/(ax%d*ax%d)

  ! read wavelet, velocity & reflectivity
  allocate(ww(at%n     )); ww=0.; call rsf_read(Fw,ww)
  allocate(vv(az%n,ax%n)); vv=0.; call rsf_read(Fv,vv)
  allocate(rr(az%n,ax%n)); rr=0.; call rsf_read(Fr,rr)

  ! allocate temporary arrays
  allocate(um(az%n,ax%n)); um=0.
  allocate(uo(az%n,ax%n)); uo=0.
  allocate(up(az%n,ax%n)); up=0.
  allocate(ud(az%n,ax%n)); ud=0.

  ! MAIN LOOP
  do it=1,at%n
     if(verb) write (0,*) it

     ! 4th order laplacian
     do iz=2,az%n-2
        do ix=2,ax%n-2
           ud(iz,ix) = &
                c0* uo(iz,  ix  ) * (idx + idz)        + &
                c1*(uo(iz  ,ix-1) + uo(iz  ,ix+1))*idx + &
                c2*(uo(iz  ,ix-2) + uo(iz  ,ix+2))*idx + &
                c1*(uo(iz-1,ix  ) + uo(iz+1,ix  ))*idz + &
                c2*(uo(iz-2,ix  ) + uo(iz+2,ix  ))*idz
        end do
     end do

     ! inject wavelet
     ud = ud - ww(it) * rr

     ! scale by velocity
     ud= ud *vv*vv

     ! time step
     up = 2*uo - um + ud * dt2
     um =   uo
     uo =   up

     ! write wavefield to output
     call rsf_write(Fo,uo)
  end do

  call exit(0)
end program AFDMf90
```

- Declare input, output and auxiliary file tags.

```
type(file) :: Fw,Fv,Fr,Fo   ! I/O files
```

- Declare RSF cube axes: `at` time axis, `ax` space axis, `az` depth axis.

```
type(axa)  :: at,az,ax      ! cube axes
```

- Declare multi-dimensional arrays for input, output and computations.

```
real, allocatable :: vv(:,:),rr(:,:),ww(:)             ! I/O arrays
real, allocatable :: um(:,:),uo(:,:),up(:,:),ud(:,:)   ! tmp arrays
```

- Open files for input/output.

```
Fw=rsf_input ("in")
Fv=rsf_input ("vel")
Fr=rsf_input ("ref")
Fo=rsf_output("out")
```

- Read axes from input files; write axes to output file.

```
call iaxa(Fw,at,1); call iaxa(Fv,az,1); call iaxa(Fv,ax,2)
call oaxa(Fo,az,1); call oaxa(Fo,ax,2); call oaxa(Fo,at,3)
```

- Allocate arrays and read wavelet, velocity and reflectivity.

```
allocate(ww(at%n      )); ww=0.; call rsf_read(Fw,ww)
allocate(vv(az%n,ax%n)); vv=0.; call rsf_read(Fv,vv)
allocate(rr(az%n,ax%n)); rr=0.; call rsf_read(Fr,rr)
```

- Allocate temporary arrays.

```
allocate(um(az%n,ax%n)); um=0.
allocate(uo(az%n,ax%n)); uo=0.
allocate(up(az%n,ax%n)); up=0.
allocate(ud(az%n,ax%n)); ud=0.
```

- Loop over time.

```
do it=1,at%n
```

- Compute Laplacian: ΔU.

```
   do iz=2,az%n-2
      do ix=2,ax%n-2
         ud(iz,ix) = &
              c0* uo(iz,  ix  ) * (idx + idz)        + &
              c1*(uo(iz  ,ix-1) + uo(iz  ,ix+1))*idx + &
              c2*(uo(iz  ,ix-2) + uo(iz  ,ix+2))*idx + &
              c1*(uo(iz-1,ix  ) + uo(iz+1,ix  ))*idz + &
              c2*(uo(iz-2,ix  ) + uo(iz+2,ix  ))*idz
      end do
   end do
```

- Inject source wavelet: $[\Delta U - f(t)]$

```
   ud = ud - ww(it) * rr
```

- Scale by velocity: $[\Delta U - f(t)]\, v^2$

```
   ud= ud *vv*vv
```

- Time step: $U_{i+1} = [\Delta U - f(t)]\, v^2 \Delta t^2 + 2U_i - U_{i-1}$

```
   up = 2*uo - um + ud * dt2
   um =   uo
   uo =   up
```