Full Waveform Inversion

Hou, Sian - sianhou1987@outlook.com Jan/09/2017

Introduction

This is an explantion of Full Waveform Inversion program in Madagascar (https://github.com/ahay/src) to help us understand the details of seismic inversion workflow. The author of code is Pengliang Yang and the theory can be found on http://www.reproducibility.org/RSF/book/xjtu/primer/paper_html/. What's more, Karol Koziol published the LaTeX template on ShareLatex https://www.sharelatex.com/.

Main points:

1. Sub the source when recovery forward wavefield, see line 309 in main( ).
2. Calculate one time forward exploration for a better CG step, see line 370-413 in main( )

main( )

main( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 1: main()

```
1
2  /* Time domain full waveform inversion
3     Note: This serial FWI is merely designed to help the understanding of
4     beginners. Enquist absorbing boundary condition (A2) is applied!
5  */
6  /*
7     Copyright (C) 2014  Xi'an Jiaotong University, UT Austin (Pengliang Yang)
8
9     This program is free software; you can redistribute it and/or modify
10    it under the terms of the GNU General Public License as published by
11    the Free Software Foundation; either version 2 of the License, or
12    (at your option) any later version.
13
14    This program is distributed in the hope that it will be useful,
15    but WITHOUT ANY WARRANTY; without even the implied warranty of
16    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17    GNU General Public License for more details.
18
19    You should have received a copy of the GNU General Public License
20    along with this program; if not, write to the Free Software
21    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

```
22
23      Important references:
24      [1] Clayton, Robert, and Bjaqrn Engquist. "Absorbing boundary
25          conditions for acoustic and elastic wave equations." Bulletin
26          of the Seismological Society of America 67.6 (1977): 1529-1540.
27      [2] Tarantola, Albert. "Inversion of seismic reflection data in the
28          acoustic approximation." Geophysics 49.8 (1984): 1259-1266.
29      [3] Pica, A., J. P. Diet, and A. Tarantola. "Nonlinear inversion
30          of seismic reflection data in a laterally invariant medium."
31          Geophysics 55.3 (1990): 284-292.
32      [4] Dussaud, E., Symes, W. W., Williamson, P., Lemaistre, L.,
33          Singer, P., Denel, B., & Cherrett, A. (2008). Computational
34          strategies for reverse-time migration. In SEG Technical Program
35          Expanded Abstracts 2008 (pp. 2267-2271).
36      [5] Hager, William W., and Hongchao Zhang. "A survey of nonlinear
37          conjugate gradient methods." Pacific journal of Optimization
38          2.1 (2006): 35-58.
39  */
40
41  int main(int argc, char *argv[]) {
42
43      //! variables on host
44      bool verb, precon, csdgather;
45      int is, it, iter, niter, distx, distz, csd, rbell;
46      int nz, nx, nt, ns, ng;
47
48      //! parameters of acquisition geometery
49      int sxbeg, szbeg, gxbeg, gzbeg, jsx, jsz, jgx, jgz;
50      float dx, dz, fm, dt, dtx, dtz, tmp, amp, obj1, obj, beta, epsil, alpha;
51      float *dobs, *dcal, *derr, *wlt, *bndr, *trans, *objval;
52      int *sxz, *gxz;
53      float **vv, **illum, **lap, **vtmp, **sp0, **sp1, **sp2, **gp0, **gp1, **↩
            gp2, **g0, **g1, **cg, *alpha1, *alpha2, **ptr=NULL;
54
55      //! time
56      clock_t start, stop;
57
58      //! I/O files
59      sf_file vinit, shots, vupdates, grads, objs, illums;
60
61      //! initialize Madagascar
62      sf_init(argc,argv);
63
64      //! set up I/O files
65      vinit=sf_input ("in");
66      /* initial velocity model, unit=m/s */
67      shots=sf_input("shots");
```

```
68      /* recorded shots from exact velocity model */
69      vupdates=sf_output("out");
70      /* updated velocity in iterations */
71      grads=sf_output("grads");
72      /* gradient in iterations */
73      illums=sf_output("illums");
74      /* source illumination in iterations */
75      objs=sf_output("objs");
76      /* values of objective function in iterations */
77
78      //! get parameters from velocity model and recorded shots
79      if (!sf_getbool("verb",&verb))      verb=true;
80      /* vebosity */
81      if (!sf_histint(vinit,"n1",&nz))    sf_error("no n1");
82      /* nz */
83      if (!sf_histint(vinit,"n2",&nx))    sf_error("no n2");
84      /* nx */
85      if (!sf_histfloat(vinit,"d1",&dz))  sf_error("no d1");
86      /* dz */
87      if (!sf_histfloat(vinit,"d2",&dx))  sf_error("no d2");
88      /* dx */
89      if (!sf_getbool("precon",&precon))  precon=false;
90      /* precondition or not */
91      if (!sf_getint("niter",&niter))     niter=100;
92      /* number of iterations */
93      if (!sf_getint("rbell",&rbell))     rbell=2;
94      /* radius of bell smooth */
95
96      if (!sf_histint(shots,"n1",&nt))    sf_error("no nt");
97      /* total modeling time steps */
98      if (!sf_histint(shots,"n2",&ng))    sf_error("no ng");
99      /* total receivers in each shot */
100     if (!sf_histint(shots,"n3",&ns))    sf_error("no ns");
101     /* number of shots */
102     if (!sf_histfloat(shots,"d1",&dt))  sf_error("no dt");
103     /* time sampling interval */
104     if (!sf_histfloat(shots,"amp",&amp))sf_error("no amp");
105     /* maximum amplitude of ricker */
106     if (!sf_histfloat(shots,"fm",&fm))  sf_error("no fm");
107     /* dominant freq of ricker */
108     if (!sf_histint(shots,"sxbeg",&sxbeg))  sf_error("no sxbeg");
109     /* x-begining index of sources, starting from 0 */
110     if (!sf_histint(shots,"szbeg",&szbeg))  sf_error("no szbeg");
111     /* x-begining index of sources, starting from 0 */
112     if (!sf_histint(shots,"gxbeg",&gxbeg))  sf_error("no gxbeg");
113     /* z-begining index of receivers, starting from 0 */
114     if (!sf_histint(shots,"gzbeg",&gzbeg))  sf_error("no gzbeg");
```

```
115     /* x-begining index of receivers, starting from 0 */
116     if (!sf_histint(shots,"jsx",&jsx))  sf_error("no jsx");
117     /* source x-axis  jump interval  */
118     if (!sf_histint(shots,"jsz",&jsz))  sf_error("no jsz");
119     /* source z-axis jump interval  */
120     if (!sf_histint(shots,"jgx",&jgx))  sf_error("no jgx");
121     /* receiver x-axis jump interval  */
122     if (!sf_histint(shots,"jgz",&jgz))  sf_error("no jgz");
123     /* receiver z-axis jump interval  */
124     if (!sf_histint(shots,"csdgather",&csd))
125     sf_error("csdgather or not required");
126     /* default, common shot-gather; if n, record at every point */
127
128     //! set up I/O parameters
129     sf_putint(vupdates,"n1",nz);
130     sf_putint(vupdates,"n2",nx);
131     sf_putint(vupdates,"n3",niter);
132     sf_putfloat(vupdates,"d1",dz);
133     sf_putfloat(vupdates,"d2",dx);
134     sf_putint(vupdates,"d3",1);
135     sf_putint(vupdates,"o3",1);
136     sf_putstring(vupdates,"label1","Depth");
137     sf_putstring(vupdates,"label2","Distance");
138     sf_putstring(vupdates,"label3","Iteration");
139     /* updated velocity in iterations */
140     sf_putint(grads,"n1",nz);
141     sf_putint(grads,"n2",nx);
142     sf_putint(grads,"n3",niter);
143     sf_putfloat(grads,"d1",dz);
144     sf_putfloat(grads,"d2",dx);
145     sf_putint(grads,"d3",1);
146     sf_putint(grads,"o3",1);
147     sf_putstring(grads,"label1","Depth");
148     sf_putstring(grads,"label2","Distance");
149     sf_putstring(grads,"label3","Iteration");
150     /* gradient in iterations */
151     sf_putint(illums,"n1",nz);
152     sf_putint(illums,"n2",nx);
153     sf_putint(illums,"n3",niter);
154     sf_putfloat(illums,"d1",dz);
155     sf_putfloat(illums,"d2",dx);
156     sf_putint(illums,"d3",1);
157     sf_putint(illums,"o3",1);
158     /* source illumination in iterations */
159     sf_putint(objs,"n1",niter);
160     sf_putint(objs,"n2",1);
161     sf_putfloat(objs,"d1",1);
```

```
162     sf_putfloat(objs,"o1",1);
163     /* values of objective function in iterations */
164     dtx=dt/dx;
165     dtz=dt/dz;
166     csdgather=(csd>0)?true:false;
167 
168     //! allocate memory
169     vv=sf_floatalloc2(nz, nx);
170     /* updated velocity */
171     vtmp=sf_floatalloc2(nz, nx);
172     /* temporary velocity computed with epsil */
173     sp0=sf_floatalloc2(nz, nx);
174     /* source wavefield p0 */
175     sp1=sf_floatalloc2(nz, nx);
176     /* source wavefield p1 */
177     sp2=sf_floatalloc2(nz, nx);
178     /* source wavefield p2 */
179     gp0=sf_floatalloc2(nz, nx);
180     /* geophone/receiver wavefield p0 */
181     gp1=sf_floatalloc2(nz, nx);
182     /* geophone/receiver wavefield p1 */
183     gp2=sf_floatalloc2(nz, nx);
184     /* geophone/receiver wavefield p2 */
185     g0=sf_floatalloc2(nz, nx);
186     /* gradient at previous step */
187     g1=sf_floatalloc2(nz, nx);
188     /* gradient at curret step */
189     cg=sf_floatalloc2(nz, nx);
190     /* conjugate gradient */
191     lap=sf_floatalloc2(nz, nx);
192     /* laplace of the source wavefield */
193     illum=sf_floatalloc2(nz, nx);
194     /* illumination of the source wavefield */
195     objval=(float*)malloc(niter*sizeof(float));
196     /* objective/misfit function */
197     wlt=(float*)malloc(nt*sizeof(float));
198     /* ricker wavelet */
199     sxz=(int*)malloc(ns*sizeof(int));
200     /* source positions */
201     gxz=(int*)malloc(ng*sizeof(int));
202     /* geophone positions */
203     bndr=(float*)malloc(nt*(2*nz+nx)*sizeof(float));
204     /* boundaries for wavefield reconstruction */
205     trans=(float*)malloc(ng*nt*sizeof(float));
206     /* transposed one shot */
207     dobs=(float*)malloc(ng*nt*sizeof(float));
208     /* observed seismic data */
```

```
209     dcal=(float*)malloc(ng*sizeof(float));
210     /* calculated/synthetic seismic data */
211     derr=(float*)malloc(ns*ng*nt*sizeof(float));
212     /* residual/error between synthetic and observation */
213     alpha1=(float*)malloc(ng*sizeof(float));
214     /* numerator of alpha, length=ng */
215     alpha2=(float*)malloc(ng*sizeof(float));
216     /* denominator of alpha, length=ng */
217
218     //! initialize varibles
219     sf_floatread(vv[0], nz*nx, vinit);
220     memset(sp0[0], 0, nz*nx*sizeof(float));
221     memset(sp1[0], 0, nz*nx*sizeof(float));
222     memset(sp2[0], 0, nz*nx*sizeof(float));
223     memset(gp0[0], 0, nz*nx*sizeof(float));
224     memset(gp1[0], 0, nz*nx*sizeof(float));
225     memset(gp2[0], 0, nz*nx*sizeof(float));
226     memset(g0[0], 0, nz*nx*sizeof(float));
227     memset(g1[0], 0, nz*nx*sizeof(float));
228     memset(cg[0], 0, nz*nx*sizeof(float));
229     memset(lap[0], 0, nz*nx*sizeof(float));
230     memset(vtmp[0], 0, nz*nx*sizeof(float));
231     memset(illum[0], 0, nz*nx*sizeof(float));
232     /* set up zero for each array */
233
234     for(it=0;it<nt;it++){
235         tmp=SF_PI*fm*(it*dt-1.0/fm);
236         tmp*=tmp;
237         wlt[it]=(1.0-2.0*tmp)*expf(-tmp);
238     }
239     /* calculate source wavelet */
240
241     if (!(sxbeg>=0 && szbeg>=0 &&
242           sxbeg+(ns-1)*jsx<nx && szbeg+(ns-1)*jsz<nz)) {
243         sf_warning("sources exceeds the computing zone!\n");
244         exit(1);
245     }
246     /* check source position */
247     sg_init(sxz, szbeg, sxbeg, jsz, jsx, ns, nz); !  GOTO sg_init( )
248     /* shot position initialize */
249
250     distx=sxbeg-gxbeg;
251     distz=szbeg-gzbeg;
252     if (csdgather){
253         if(!(gxbeg>=0 && gzbeg>=0 &&
254              gxbeg+(ng-1)*jgx<nx && gzbeg+(ng-1)*jgz<nz &&
255              (sxbeg+(ns-1)*jsx)+(ng-1)*jgx-distx <nx  &&
```

```
256                 (szbeg+(ns-1)*jsz)+(ng-1)*jgz-distz <nz)){
257                     sf_warning("geophones exceeds the computing zone!\n");
258                     exit(1);
259             }
260         } else{
261             if(!(gxbeg>=0 && gzbeg>=0 &&
262                  gxbeg+(ng-1)*jgx<nx && gzbeg+(ng-1)*jgz<nz)){
263                 sf_warning("geophones exceeds the computing zone!\n");
264                 exit(1);
265             }
266         }
267         /* check receivers position */
268         sg_init(gxz, gzbeg, gxbeg, jgz, jgx, ng, nz); !  GOTO sg_init( )
269         /*
270          * receiver position initialize
271          * this code is available when csdgather==false
272          */
273 
274         memset(bndr, 0, nt*(2*nz+nx)*sizeof(float));
275         memset(dobs, 0, ng*nt*sizeof(float));
276         memset(dcal, 0, ng*sizeof(float));
277         memset(derr, 0, ns*ng*nt*sizeof(float));
278         memset(alpha1, 0, ng*sizeof(float));
279         memset(alpha2, 0, ng*sizeof(float));
280         memset(dobs, 0, ng*nt*sizeof(float));
281         memset(objval, 0, niter*sizeof(float));
282         /* set up zero for each array */
283 
284         for(iter=0; iter<niter; iter++){
285             if(verb){
286                 start=clock();/* record starting time */
287                 sf_warning("iter=%d",iter);
288             }
289             sf_seek(shots, 0L, SEEK_SET);
290             memcpy(g0[0], g1[0], nz*nx*sizeof(float));
291             memset(g1[0], 0, nz*nx*sizeof(float));
292             memset(illum[0], 0, nz*nx*sizeof(float));
293             for(is=0;is<ns;is++){
294                 sf_floatread(trans, ng*nt, shots);
295                 /* read shot gather */
296                 matrix_transpose(trans, dobs, nt, ng); !  GOTO matrix_transpose( )
297                 /* transpose the matrix to get dobs */
298                 if(csdgather){
299                     gxbeg=sxbeg+is*jsx-distx;
300                     sg_init(gxz, gzbeg, gxbeg, jgz, jgx, ng, nz); !  GOTO sg_init( )
301                 }
302                 /* receiver position initialize */
```

```
303
304             memset(sp0[0], 0, nz*nx*sizeof(float));
305             memset(sp1[0], 0, nz*nx*sizeof(float));
306             for(it=0; it<nt; it++){
307                 add_source(sp1, &wlt[it], &sxz[is], 1, nz, true);
308                 !  GOTO add_source( )
309                 /* add source */
310
311                 step_forward(sp0, sp1, sp2, vv, dtz, dtx, nz, nx);
312                 !  GOTO step_forward( )
313                 /* forward exploration */
314
315                 ptr=sp0; sp0=sp1; sp1=sp2; sp2=ptr;
316                 /* update wavefield */
317
318                 rw_bndr(&bndr[it*(2*nz+nx)], sp0, nz, nx, true);
319                 !  GOTO rw_bndr( )
320                 /* save boundary value for saving memory */
321
322                 record_seis(dcal, gxz, sp0, ng, nz);
323                 !  GOTO record_seis( )
324                 /* save seismic record at recevier position */
325
326                 cal_residuals(dcal,&dobs[it*ng],&derr[is*ng*nt+it*ng],ng);
327                 !  GOTO cal_residuals( )
328                 /* calculate record residual at recevier position */
329             }
330             /* forward exploration complete */
331
332             ptr=sp0; sp0=sp1; sp1=ptr;
333             memset(gp0[0], 0, nz*nx*sizeof(float));
334             memset(gp1[0], 0, nz*nx*sizeof(float));
335             for(it=nt-1; it>-1; it--){
336                 rw_bndr(&bndr[it*(2*nz+nx)], sp1, nz, nx, false);
337                 !  GOTO rw_bndr( )
338                 /* read boundary value for saving memory */
339
340                 step_backward(illum,lap,sp0,sp1,sp2,vv,dtz,dtx,nz,nx);
341                 !  GOTO step_backward( )
342                 /*
343                  * this step is to recovery forward wavefield
344                  * via backward exploration and boundary condition
345                  * illum is the source compensate
346                  * lap is the laplace operator * velocity^2
347                  */
348
349                 add_source(sp1, &wlt[it], &sxz[is], 1, nz, false);
```

```
            !  GOTO add_source( )
            /* sub source to elminate source in backward scattering */

            add_source(gp1, &derr[is*ng*nt+it*ng], gxz, ng, nz, true);
            !  GOTO add_source( )
            /* stack residual as backward scattering source */

            step_forward(gp0, gp1, gp2, vv, dtz, dtx, nz, nx);
            !  GOTO step_forward( )
            /* backward scattering residual wavefield */

            cal_gradient(g1, lap, gp1, nz, nx);
            !  GOTO cal_gradient( )
            /* calculate gradient via correlation */

            ptr=sp0; sp0=sp1; sp1=sp2; sp2=ptr;
            ptr=gp0; gp0=gp1; gp1=gp2; gp2=ptr;
            /* update wavefield */
        }
    }
    /* simulating complete */

    obj=cal_objective(derr, ng*nt*ns);
    !  GOTO cal_objective( )
    /* obj = norm2(derr) */

    scale_gradient(g1, vv, illum, nz, nx, precon);
    !  GOTO scale_gradient( )
    /*
     * g1 = 2.0*g1/velocity^2
     * IF precon == true DO source compensate
     */
    sf_floatwrite(illum[0], nz*nx, illums);
    /* output illum */
    bell_smoothz(g1, illum, rbell, nz, nx);
    !  GOTO bell_smoothz( )
    bell_smoothx(illum, g1, rbell, nz, nx);
    !  GOTO bell_smoothx( )
    /* smooth g1 while use illum as temp store */
    sf_floatwrite(g1[0], nz*nx, grads);
    /* output gradient */
    /* calculating gradient complete */

    if (iter>0)
        beta=cal_beta(g0, g1, cg, nz, nx);
    else
        beta=0.0;
```

```
397         !  GOTO cal_beta( )
398         /* calculate beta */
399         cal_conjgrad(g1, cg, beta, nz, nx);
400         !  GOTO cal_conjgrad( )
401         /* calculate cg direction */
402         epsil=cal_epsilon(vv, cg, nz, nx);
403         !  GOTO cal_epsilon( )
404         /* calculate cg step size */
405         /* calculating CG direction complete */
406
407         sf_seek(shots, 0L, SEEK_SET);
408         memset(alpha1, 0, ng*sizeof(float));
409         memset(alpha2, 0, ng*sizeof(float));
410         cal_vtmp(vtmp, vv, cg, epsil, nz, nx);
411         !  GOTO cal_vtmp( )
412         /* update the velocity */
413         for(is=0;is<ns;is++){
414             sf_floatread(trans, ng*nt, shots);
415             /* read shot gather */
416             matrix_transpose(trans, dobs, nt, ng);
417             !  GOTO matrix_transpose( )
418             /* transpose the matrix to get dobs */
419             if(csdgather){
420                 gxbeg=sxbeg+is*jsx-distx;
421                 sg_init(gxz, gzbeg, gxbeg, jgz, jgx, ng, nz);
422                 !  GOTO sg_init( )
423             }
424             /* receiver position initialize */
425             memset(sp0[0], 0, nz*nx*sizeof(float));
426             memset(sp1[0], 0, nz*nx*sizeof(float));
427             for(it=0; it<nt; it++){
428                 add_source(sp1, &wlt[it], &sxz[is], 1, nz, true);
429                 !  GOTO add_source( )
430                 /* add source */
431
432                 step_forward(sp0, sp1, sp2, vv, dtz, dtx, nz, nx);
433                 !  GOTO step_forward( )
434                 /* forward exploration */
435
436                 ptr=sp0; sp0=sp1; sp1=sp2; sp2=ptr;
437                 /* update wavefield */
438
439                 record_seis(dcal, gxz, sp0, ng, nz);
440                 !  GOTO record_seis( )
441                 /* save seismic record at recevier position */
442
443                 sum_alpha12(alpha1, alpha2, dcal, &dobs[it*ng], &derr[is*ng*nt↩
```

```
                    +it*ng], ng);
444                 !  GOTO sum_alpha12( )
445                 /* calculate alpha12 */
446             }
447         }
448
449         alpha=cal_alpha(alpha1, alpha2, epsil, ng);
450         !  GOTO cal_alpha( )
451         /* calculate alpha */
452
453         update_vel(vv, cg, alpha, nz, nx);
454         !  GOTO update_vel( )
455         /* update velocity */
456         sf_floatwrite(vv[0], nz*nx, vupdates);
457         /* output velcotiy */
458         /* updating velocity complete */
459
460         if(iter==0) {
461             obj1=obj;
462             objval[iter]=1.0;
463         } else{
464             objval[iter]=obj/obj1;
465         }
466         /* calcuate obj */
467
468         if(verb) {
469             sf_warning("obj=%f  beta=%f  epsil=%f  alpha=%f", obj, beta, epsil↩
                    , alpha);
470             /* output important information at each FWI iteration */
471             stop=clock();
472             /* record ending time */
473             sf_warning("iteration %d finished: %f (s)",iter+1, ((float)(stop-↩
                    start))/CLOCKS_PER_SEC);
474         }
475     }
476     sf_floatwrite(objval, niter, objs);
477     /* output obj */
478
479     free(*vv); free(vv);
480     free(*vtmp); free(vtmp);
481     free(*sp0); free(sp0);
482     free(*sp1); free(sp1);
483     free(*sp2); free(sp2);
484     free(*gp0); free(gp0);
485     free(*gp1); free(gp1);
486     free(*gp2); free(gp2);
487     free(*g0); free(g0);
```

```
488     free(*g1); free(g1);
489     free(*cg); free(cg);
490     free(*lap); free(lap);
491     free(*illum); free(illum);
492     free(objval);
493     free(wlt);
494     free(sxz);
495     free(gxz);
496     free(bndr);
497     free(trans);
498     free(dobs);
499     free(dcal);
500     free(derr);
501     free(alpha1);
502     free(alpha2);
503 
504     exit(0);
505 }
```

sg_init( )

sg_init( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 2: sg_init( )

```
1     void sg_init(int *sxz, int szbeg, int sxbeg,
2                  int jsz, int jsx, int ns, int nz)
3     /*< shot/geophone position initialize >*/
4     {
5         int is, sz, sx;
6         for(is=0; is<ns; is++) {
7             sz=szbeg+is*jsz;
8             sx=sxbeg+is*jsx;
9             sxz[is]=sz+nz*sx;
10        }
11    }
12    //!  RETURN main( )
```

matrix_transpose( )

matrix_transpose( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 3: matrix_transpose( )

```
void matrix_transpose(float *matrix, float *trans, int n1, int n2)
/*< matrix transpose: matrix tansposed to be trans >*/
{
    int i1, i2;

    for(i2=0; i2<n2; i2++)
        for(i1=0; i1<n1; i1++)
            trans[i2+n2*i1]=matrix[i1+n1*i2];
}
//!  RETURN main( )
```

add_source( )

add_source( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 4: add_source( )

```
void add_source(float **p, float *source, int *sxz, int ns, int nz, bool add)
/*< add/subtract seismic sources >*/
{
    int is, sx, sz;
    if(add){
        for(is=0;is<ns; is++){
            sx=sxz[is]/nz;
            sz=sxz[is]%nz;
            p[sx][sz]+=source[is];
        }
    }else{
        for(is=0;is<ns; is++){
            sx=sxz[is]/nz;
            sz=sxz[is]%nz;
            p[sx][sz]-=source[is];
        }
    }
}
//!  RETURN main( )
```

step_forward( )

step_forward( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 5: step_forward( )

```
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;
        }
    }
    for (ix=1; ix < nx-1; ix++) {
        /* top boundary */
        /*
        iz=0;
        diff1=  (p1[ix][iz+1]-p1[ix][iz])-
        (p0[ix][iz+1]-p0[ix][iz]);
        diff2=  c21*(p1[ix-1][iz]+p1[ix+1][iz]) +
        c22*(p1[ix-2][iz]+p1[ix+2][iz]) +
        c20*p1[ix][iz];
        diff1*=sqrtf(vv[ix][iz])/dz;
        diff2*=vv[ix][iz]/(2.0*dx*dx);
        p2[ix][iz]=2*p1[ix][iz]-p0[ix][iz]+diff1+diff2;
        */
        /* bottom boundary */
        iz=nz-1;
        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;
    }

    for (iz=1; iz <nz-1; iz++){
```

```
        /* left boundary */
        ix=0;
        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;
        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;
    }
}
//!  RETURN main( )
```

rw_bndr( )

rw_bndr( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 6: rw_bndr( )

```
void rw_bndr(float *bndr, float **p, int nz, int nx, bool write)
    /*< if write==true, write/save boundaries out of variables;
    else  read boundaries into variables (for 2nd order FD) >*/
{
    int i;
    if(write){
        for(i=0; i<nz; i++){
            bndr[i]=p[0][i];
            bndr[i+nz]=p[nx-1][i];
        }
        for(i=0; i<nx; i++)
            bndr[i+2*nz]=p[i][nz-1];
    }else{
        for(i=0; i<nz; i++){
            p[0][i]=bndr[i];
            p[nx-1][i]=bndr[i+nz];
        }
```

```
18          for(i=0; i<nx; i++)
19              p[i][nz-1]=bndr[i+2*nz];
20          }
21  }
22  //!  RETURN main( )
```

record_seis( )

record_seis( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 7: record_seis( )

```
1  void record_seis(float *seis_it, int *gxz, float **p, int ng, int nz)
2  /*< record seismogram at time it into a vector length of ng >*/
3  {
4      int ig, gx, gz;
5      for(ig=0;ig<ng; ig++){
6          gx=gxz[ig]/nz;
7          gz=gxz[ig]%nz;
8          seis_it[ig]=p[gx][gz];
9      }
10 }
11 //!  RETURN main( )
```

cal_residuals( )

cal_residuals( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 8: cal_residuals( )

```
1  void cal_residuals(float *dcal, float *dobs, float *dres, int ng)
2  /*< calculate residual >*/
3  {
4      int ig;
5      for(ig=0; ig<ng; ig++){
6          dres[ig]=dcal[ig]-dobs[ig];
7      }
8  }
9  //!  RETURN main( )
```

step_backward( )

step_backward( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 9: step_backward( )

```
void step_backward(float **illum, float **lap, float **p0, float **p1, float **p2, float **vv, float dtz, float dtx, int nz, int nx)
/*< step backward >*/
{
    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;
            lap[ix][iz]=diff1+diff2;
            diff1*=v1;
            diff2*=v2;
            p2[ix][iz]=2.0*p1[ix][iz]-p0[ix][iz]+diff1+diff2;
            illum[ix][iz]+=p1[ix][iz]*p1[ix][iz];
        }
    }
}
//!  RETURN main( )
```

cal_gradient( )

cal_gradient( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 10: cal_gradient( )

```
void cal_gradient(float **grad, float **lap, float **gp, int nz, int nx)
/*< calculate gradient >*/
{
    int ix, iz;
        for(ix=0; ix<nx; ix++){
            for(iz=0; iz<nz; iz++){
                grad[ix][iz]+=lap[ix][iz]*gp[ix][iz];
            }
```

```
        }
}
//!  RETURN main( )
```

cal_objective( )

cal_objective( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 11: cal_objective( )

```
float cal_objective(float *dres, int ng)
/*< calculate the value of objective function >*/
{
    int i;
    float a, obj=0;

    for(i=0; i<ng; i++){
        a=dres[i];
        obj+=a*a;
    }
    return obj;
}
//!  RETURN main( )
```

scale_gradient( )

scale_gradient( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 12: scale_gradient( )

```
void scale_gradient(float **grad, float **vv, float **illum, int nz, int nx, ↩
    bool precon)
/*< scale gradient >*/
{
    int ix, iz;
    float a;
    for(ix=1; ix<nx-1; ix++){
        for(iz=1; iz<nz-1; iz++){
            a=vv[ix][iz];
            if (precon)
                a*=sqrtf(illum[ix][iz]+SF_EPS);
                /*precondition with residual wavefield illumination*/
            grad[ix][iz]*=2.0/a;
```

```
        }
    }
    for(ix=0; ix<nx; ix++){
        grad[ix][0]=grad[ix][1];
        grad[ix][nz-1]=grad[ix][nz-2];
    }

    for(iz=0; iz<nz; iz++){
        grad[0][iz]=grad[1][iz];
        grad[nx-1][iz]=grad[nx-2][iz];
    }
}
//!  RETURN main( )
```

bell_smoothz( )

bell_smoothz( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 13: bell_smoothz( )

```
void bell_smoothz(float **g, float **smg, int rbell, int nz, int nx)
/*< gaussian bell smoothing for z-axis >*/
{
    int ix, iz, i;
    float s;

    for(ix=0; ix<nx; ix++){
        for(iz=0; iz<nz; iz++){
            s=0.0;
            for(i=-rbell; i<=rbell; i++)
            if(iz+i>=0 && iz+i<nz)
            s+=expf(-(2.0*i*i)/rbell)*g[ix][iz+i];
            smg[ix][iz]=s;
        }
    }
}
//!  RETURN main( )
```

bell_smoothx( )

bell_smoothx( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 14: bell_smoothx( )

```
void bell_smoothx(float **g, float **smg, int rbell, int nz, int nx)
/*< gaussian bell smoothing for x-axis >*/
{
    int ix, iz, i;
    float s;

    for(ix=0; ix<nx; ix++) {
        for(iz=0; iz<nz; iz++){
            s=0.0;
            for(i=-rbell; i<=rbell; i++)
                if(ix+i>=0 && ix+i<nx)
                    s+=expf(-(2.0*i*i)/rbell)*g[ix+i][iz];
            smg[ix][iz]=s;
        }
    }
}
//!  RETURN main( )
```

cal_beta( )

cal_beta( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 15: cal_beta( )

```
float cal_beta(float **g0, float **g1, float **cg, int nz, int nx)
/*< calculate beta >*/
{
    int ix, iz;
    float a,b,c;

    a=b=c=0;
    for(ix=0; ix<nx; ix++){
        for(iz=0; iz<nz; iz++){
            a += g1[ix][iz]*(g1[ix][iz]-g0[ix][iz]);// numerator of HS
            b += cg[ix][iz]*(g1[ix][iz]-g0[ix][iz]);// denominator of HS,DY
            c += g1[ix][iz]*g1[ix][iz];             // numerator of DY
        }
    }

    float beta_HS=(fabsf(b)>0)?(a/b):0.0;
    float beta_DY=(fabsf(b)>0)?(c/b):0.0;
    return SF_MAX(0.0, SF_MIN(beta_HS, beta_DY));
}
//!  RETURN main( )
```

cal_conjgrad( )

cal_conjgrad( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 16: cal_conjgrad( )

```
void cal_conjgrad(float **g1, float **cg, float beta, int nz, int nx)
/*< calculate conjugate gradient >*/
{
    int ix, iz;

    for(ix=0; ix<nx; ix++){
        for(iz=0; iz<nz; iz++){
            cg[ix][iz]=-g1[ix][iz]+beta*cg[ix][iz];
        }
    }
}
//!  RETURN main( )
```

cal_epsilon( )

cal_epsilon( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 17: cal_epsilon( )

```
float cal_epsilon(float **vv, float **cg, int nz, int nx)
/*< calculate epsilcon >*/
{
    int ix, iz;
    float vvmax, cgmax;
    vvmax=cgmax=0.0;

    for(ix=0; ix<nx; ix++){
        for(iz=0; iz<nz; iz++){
            vvmax=SF_MAX(vvmax, fabsf(vv[ix][iz]));
            cgmax=SF_MAX(cgmax, fabsf(cg[ix][iz]));
        }
    }

    return 0.01*vvmax/(cgmax+SF_EPS);
}
//!  RETURN main( )
```

cal_vtmp( )

cal_vtmp( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 18: cal_vtmp( )

```
void cal_vtmp(float **vtmp, float **vv, float **cg, float epsil, int nz, int nx)
/*< calculate temporary velcity >*/
{
    int ix, iz;

    for(ix=0; ix<nx; ix++){
        for(iz=0; iz<nz; iz++){
            vtmp[ix][iz]=vv[ix][iz]+epsil*cg[ix][iz];
        }
    }
}
//!  RETURN main( )
```

sum_alpha12( )

sum_alpha12( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 19: sum_alpha12( )

```
void sum_alpha12(float *alpha1, float *alpha2, float *dcaltmp, float *dobs, float *derr, int ng)
/*< calculate numerator and denominator of alpha >*/
{
    int ig;
    float a, b, c;
    for(ig=0; ig<ng; ig++){
        c=derr[ig];
        a=dobs[ig]+c;
        /*
         * since f(mk)-dobs[id]=derr[id],
         * thus f(mk)=b+c;
         */
        b=dcaltmp[ig]-a;
        /* f(mk+epsil*cg)-f(mk) */
        alpha1[ig]-=b*c;
        alpha2[ig]+=b*b;
    }
}
//!  RETURN main( )
```

cal_alpha( )

cal_alpha( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 20: cal_alpha( )

```
float cal_alpha(float *alpha1, float *alpha2, float epsil, int ng)
/*< calculate alpha >*/
{
    int ig;
    float a,b;

    a=b=0;
    for(ig=0; ig<ng; ig++){
        a+=alpha1[ig];
        b+=alpha2[ig];
    }

    return (a*epsil/(b+SF_EPS));
}
//!  RETURN main( )
```

update_vel( )

update_vel( ) in $(RSFROOT)/src/user/pyang/Mfwi2d.c.

Listing 21: update_vel( )

```
void update_vel(float **vv, float **cg, float alpha, int nz, int nx)
/*< update velcity >*/
{
    int ix, iz;

    for(ix=0; ix<nx; ix++){
        for(iz=0; iz<nz; iz++){
            vv[ix][iz]+=alpha*cg[ix][iz];
        }
    }
}
//!  RETURN main( )
```