Least Square Migration

Hou, Sian - sianhou1987@outlook.com Jan/10/2017

Introduction

This is an explantion of Least Square Migration 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. You'd better to read the "Full Waveform Inversion in Madagascar.pdf" firstly.

main( )

main( ) in $(RSFROOT)/src/user/pyang/Mlsprtm2d.c.

Listing 1: main()

```
/* 2-D prestack least-squares RTM using wavefield reconstruction
   NB: Sponge ABC is applied!
*/
/*
   Copyright (C) 2014  Xi'an Jiaotong University (Pengliang Yang)

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

#include <rsf.h>
```

```
23
24 #include "prtm2d.h"
25
26 int main(int argc, char* argv[]){
27
28     bool verb;
29     int nb, nz, nx, nt, ns, ng, niter, csd, sxbeg, szbeg, jsx, jsz, gxbeg, ↩
           gzbeg, jgx, jgz;
30     float dz, dx, dt, fm, o1, o2, amp;
31     float **v0, *mod, *dat;
32
33     //! I/O files
34     sf_file shots, imag, imgrtm, velo;
35
36     //! initialize Madagascar
37     sf_init(argc,argv);
38
39     shots = sf_input ("in");
40     /* shot records, data */
41     velo = sf_input ("vel");
42     /* velocity */
43     imag = sf_output("out");
44     /* output LSRTM image, model */
45     imgrtm = sf_output("imgrtm");
46     /* output RTM image */
47
48     if (!sf_histint(velo,"n1",&nz)) sf_error("n1");
49     /* 1st dimension size */
50     if (!sf_histint(velo,"n2",&nx)) sf_error("n2");
51     /* 2nd dimension size */
52     if (!sf_histfloat(velo,"d1",&dz)) sf_error("d1");
53     /* d1 */
54     if (!sf_histfloat(velo,"d2",&dx)) sf_error("d2");
55     /* d2 */
56     if (!sf_histfloat(velo,"o1",&o1)) sf_error("o1");
57     /* o1 */
58     if (!sf_histfloat(velo,"o2",&o2)) sf_error("o2");
59     /* o2 */
60     if (!sf_getbool("verb",&verb)) verb=true;
61     /* verbosity */
62     if (!sf_getint("niter",&niter)) niter=10;
63     /* totol number of least-squares iteration*/
64     if (!sf_getint("nb",&nb)) nb=20;
65     /* number (thickness) of ABC on each side */
66     if (!sf_histint(shots,"n1",&nt)) sf_error("no nt");
67     /* total modeling time steps */
68     if (!sf_histint(shots,"n2",&ng)) sf_error("no ng");
```

```
    /* total receivers in each shot */
    if (!sf_histint(shots,"n3",&ns)) sf_error("no ns");
    /* number of shots */
    if (!sf_histfloat(shots,"d1",&dt)) sf_error("no dt");
    /* time sampling interval */
    if (!sf_histfloat(shots,"amp",&amp)) sf_error("no amp");
    /* maximum amplitude of ricker */
    if (!sf_histfloat(shots,"fm",&fm)) sf_error("no fm");
    /* dominant freq of ricker */
    if (!sf_histint(shots,"sxbeg",&sxbeg)) sf_error("no sxbeg");
    /* x-begining index of sources, starting from 0 */
    if (!sf_histint(shots,"szbeg",&szbeg)) sf_error("no szbeg");
    /* x-begining index of sources, starting from 0 */
    if (!sf_histint(shots,"gxbeg",&gxbeg)) sf_error("no gxbeg");
    /* x-begining index of receivers, starting from 0 */
    if (!sf_histint(shots,"gzbeg",&gzbeg)) sf_error("no gzbeg");
    /* x-begining index of receivers, starting from 0 */
    if (!sf_histint(shots,"jsx",&jsx)) sf_error("no jsx");
    /* source x-axis  jump interval */
    if (!sf_histint(shots,"jsz",&jsz)) sf_error("no jsz");
    /* source z-axis jump interval */
    if (!sf_histint(shots,"jgx",&jgx)) sf_error("no jgx");
    /* receiver x-axis jump interval */
    if (!sf_histint(shots,"jgz",&jgz)) sf_error("no jgz");
    /* receiver z-axis jump interval  */
    if (!sf_histint(shots,"csdgather",&csd))
        sf_error("csdgather or not required");
    /* default, common shot-gather; if n, record at every point*/

    sf_putint(imag,"n1",nz);
    sf_putint(imag,"n2",nx);
    sf_putint(imag,"n3",1);
    sf_putfloat(imag,"d1",dz);
    sf_putfloat(imag,"d2",dx);
    sf_putfloat(imag,"o1",o1);
    sf_putfloat(imag,"o2",o2);
    sf_putstring(imag,"label1","Depth");
    sf_putstring(imag,"label2","Distance");
    /* output LSRTM image, model */

    sf_putint(imgrtm,"n1",nz);
    sf_putint(imgrtm,"n2",nx);
    sf_putint(imgrtm,"n3",1);
    sf_putfloat(imgrtm,"d1",dz);
    sf_putfloat(imgrtm,"d2",dx);
    sf_putfloat(imgrtm,"o1",o1);
    sf_putfloat(imgrtm,"o2",o2);
```

```
116     sf_putstring(imgrtm,"label1","Depth");
117     sf_putstring(imgrtm,"label2","Distance");
118     /* output RTM image */
119 
120     v0=sf_floatalloc2(nz,nx);
121     mod=sf_floatalloc(nz*nx);
122     dat=sf_floatalloc(nt*ng*ns);
123     /*
124      * In rtm, vv is the velocity model [modl], which is input parameter;
125      * mod is the image/reflectivity [imag];
126      * dat is seismogram [data]!
127      */
128 
129     //! initialize velocity, model and data
130     sf_floatread(v0[0], nz*nx, velo);
131     memset(mod, 0, nz*nx*sizeof(float));
132     sf_floatread(dat, nt*ng*ns, shots);
133     prtm2d_init(verb, csd, dz, dx, dt, amp, fm, nz, nx, nb, nt, ns, ng,
134     sxbeg, szbeg, jsx, jsz, gxbeg, gzbeg, jgx, jgz, v0, mod, dat);
135      !  GOTO prtm2d_init( )
136 
137     prtm2d_lop(true, false, nz*nx, nt*ng*ns, mod, dat);
138      !  GOTO prtm2d_lop( )
139     /* original RTM is simply applying adjoint of prtm2d_lop once!*/
140     sf_floatwrite(mod, nz*nx, imgrtm);
141     /* output RTM image */
142 
143     //! least squares migration
144     sf_solver(prtm2d_lop, sf_cgstep, nz*nx, nt*ng*ns, mod, dat, niter, "verb",↩
            verb, "end");
145     sf_floatwrite(mod, nz*nx, imag);
146     /* output inverted image */
147 
148     sf_cgstep_close();
149     prtm2d_close();
150     free(*v0); free(v0);
151     free(mod);
152     free(dat);
153 
154     exit(0);
155 }
```

prtm2d.c

prtm2d.c in `$(RSFROOT)/src/user/pyang/prtm2d.c`.

Listing 2: prtm2d.c

```
/* 2-D prestack LSRTM linear operator using wavefield reconstruction method
   Note: Sponge ABC is applied!
*/
/*
   Copyright (C) 2014  Xi'an Jiaotong University, UT Austin (Pengliang Yang)

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/
#include <rsf.h>

#ifdef _OPENMP
#include <omp.h>
#endif

#include "prtm2d.h"

static bool csdgather, verb;
static int nzpad, nxpad, nb, nz, nx, nt, ns, ng, sxbeg, szbeg, jsx, jsz, gxbeg
    , gzbeg, jgx, jgz, distx, distz;
static int *sxz, *gxz;
static float c0, c11, c21, c12, c22;
static float *wlt, *bndr,*rwbndr, *mod, *dat;
static float **sp0, **sp1, **gp0, **gp1, **vv, **ptr=NULL;


void boundary_rw(float **p, float *spo, bool read)
 !  GO BACK
/* read/write using effective boundary saving strategy:
if read=true, read the boundary out; else save/write the boundary */
{
    int ix,iz;

    if (read){
        #ifdef _OPENMP
```

```
46              #pragma omp parallel for            \
47              private(ix,iz)              \
48              shared(p,spo,nx,nz,nb)
49          #endif
50          for(ix=0; ix<nx; ix++){
51              for(iz=0; iz<2; iz++){
52                  p[ix+nb][iz-2+nb]=spo[iz+4*ix];
53                  p[ix+nb][iz+nz+nb]=spo[iz+2+4*ix];
54              }
55          }
56          #ifdef _OPENMP
57              #pragma omp parallel for            \
58              private(ix,iz)              \
59              shared(p,spo,nx,nz,nb)
60          #endif
61          for(iz=0; iz<nz; iz++){
62              for(ix=0; ix<2; ix++){
63                  p[ix-2+nb][iz+nb]=spo[4*nx+iz+nz*ix];
64                  p[ix+nx+nb][iz+nb]=spo[4*nx+iz+nz*(ix+2)];
65              }
66          }
67      } else {
68          #ifdef _OPENMP
69              #pragma omp parallel for            \
70              private(ix,iz)              \
71              shared(p,spo,nx,nz,nb)
72          #endif
73          for(ix=0; ix<nx; ix++){
74              for(iz=0; iz<2; iz++){
75                  spo[iz+4*ix]=p[ix+nb][iz-2+nb];
76                  spo[iz+2+4*ix]=p[ix+nb][iz+nz+nb];
77              }
78          }
79          #ifdef _OPENMP
80              #pragma omp parallel for            \
81              private(ix,iz)              \
82              shared(p,spo,nx,nz,nb)
83          #endif
84          for(iz=0; iz<nz; iz++){
85              for(ix=0; ix<2; ix++){
86                  spo[4*nx+iz+nz*ix]=p[ix-2+nb][iz+nb];
87                  spo[4*nx+iz+nz*(ix+2)]=p[ix+nx+nb][iz+nb];
88              }
89          }
90      }
91  }
92
```

```
93  void step_forward(float **u0, float **u1, float **vv, bool adj)
94   !  GO BACK
95  /*< forward step for wave propagation >*/
96  {
97      int i1, i2;
98
99      if(adj){
100         #ifdef _OPENMP
101             #pragma omp parallel for default(none)          \
102             private(i2,i1)                  \
103             shared(nzpad,nxpad,u1,vv,u0,c0,c11,c12,c21,c22)
104         #endif
105         for (i2=2; i2<nxpad-2; i2++) {
106             for (i1=2; i1<nzpad-2; i1++) {
107                 u0[i2][i1]=2.*u1[i2][i1]-u0[i2][i1]+
108                     c0*vv[i2][i1]*u1[i2][i1]+
109                     c11*(vv[i2][i1-1]*u1[i2][i1-1]+vv[i2][i1+1]*u1[i2][i1+1])+
110                     c12*(vv[i2][i1-2]*u1[i2][i1-2]+vv[i2][i1+2]*u1[i2][i1+2])+
111                     c21*(vv[i2-1][i1]*u1[i2-1][i1]+vv[i2+1][i1]*u1[i2+1][i1])+
112                     c22*(vv[i2-2][i1]*u1[i2-2][i1]+vv[i2+2][i1]*u1[i2+2][i1]);
113             }
114         }
115     }else{
116         #ifdef _OPENMP
117             #pragma omp parallel for default(none)          \
118             private(i2,i1)                  \
119             shared(nzpad,nxpad,u1,vv,u0,c0,c11,c12,c21,c22)
120         #endif
121         for (i2=2; i2<nxpad-2; i2++) {
122             for (i1=2; i1<nzpad-2; i1++) {
123                 u0[i2][i1]=2.*u1[i2][i1]-u0[i2][i1]+
124                     vv[i2][i1]*(c0*u1[i2][i1]+
125                     c11*(u1[i2][i1-1]+u1[i2][i1+1])+
126                     c12*(u1[i2][i1-2]+u1[i2][i1+2])+
127                     c21*(u1[i2-1][i1]+u1[i2+1][i1])+
128                     c22*(u1[i2-2][i1]+u1[i2+2][i1]));
129             }
130         }
131     }
132 }
133
134 void apply_sponge(float **p0)
135 /*< apply sponge (Gaussian taper) absorbing boundary condition
136 L=Gaussian taper ABC; L=L*, L is self-adjoint operator. >*/
137 {
138     int ix,iz,ib,ibx,ibz;
139     float w;
```

```
140
141     for(ib=0; ib<nb; ib++) {
142         w = bndr[ib];
143         ibz = nzpad-ib-1;
144         for(ix=0; ix<nxpad; ix++) {
145             p0[ix][ib ] *= w; /*    top sponge */
146             p0[ix][ibz] *= w; /* bottom sponge */
147         }
148
149         ibx = nxpad-ib-1;
150         for(iz=0; iz<nzpad; iz++) {
151             p0[ib ][iz] *= w; /*   left sponge */
152             p0[ibx][iz] *= w; /*  right sponge */
153         }
154     }
155 }
156
157 void sg_init(int *sxz, int szbeg, int sxbeg, int jsz, int jsx, int ns)
158 /*< shot/geophone position initialize
159     sxz/gxz; szbeg/gzbeg;
160     sxbeg/gxbeg;
161     jsz/jgz; jsx/jgx; ns/ng; >*/
162 {
163     int is, sz, sx;
164
165     for(is=0; is<ns; is++){
166         sz=szbeg+is*jsz;
167         sx=sxbeg+is*jsx;
168         sxz[is]=sz+nz*sx;
169     }
170 }
171
172 void add_source(int *sxz, float **p, int ns, float *source, bool add)
173 /*< add seismic sources in grid >*/
174 {
175     int is, sx, sz;
176
177     if(add){
178     /* add sources*/
179         #ifdef _OPENMP
180             #pragma omp parallel for default(none)      \
181             private(is,sx,sz)               \
182             shared(p,source,sxz,nb,ns,nz)
183         #endif
184         for(is=0;is<ns; is++){
185             sx=sxz[is]/nz;
186             sz=sxz[is]%nz;
```

```
            p[sx+nb][sz+nb]+=source[is];
        }
    }else{
    /* subtract sources */
        #ifdef _OPENMP
            #pragma omp parallel for default(none)      \
            private(is,sx,sz)               \
            shared(p,source,sxz,nb,ns,nz)
        #endif
        for(is=0;is<ns; is++){
            sx=sxz[is]/nz;
            sz=sxz[is]%nz;
            p[sx+nb][sz+nb]-=source[is];
        }
    }
}

void expand2d(float** b, float** a)
/*< expand domain of 'a' to 'b': source(a)-->destination(b) >*/
{
int iz,ix;

#ifdef _OPENMP
#pragma omp parallel for default(none)      \
private(ix,iz)              \
shared(b,a,nb,nz,nx)
#endif
for     (ix=0;ix<nx;ix++) {
for (iz=0;iz<nz;iz++) {
b[nb+ix][nb+iz] = a[ix][iz];
}
}

for     (ix=0; ix<nxpad; ix++) {
for (iz=0; iz<nb;    iz++) {
b[ix][      iz  ] = b[ix][nb  ];
b[ix][nzpad-iz-1] = b[ix][nzpad-nb-1];
}
}

for     (ix=0; ix<nb;    ix++) {
for (iz=0; iz<nzpad; iz++) {
b[ix     ][iz] = b[nb       ][iz];
b[nxpad-ix-1 ][iz] = b[nxpad-nb-1   ][iz];
}
}
}
```

```
234
235
236 void window2d(float **a, float **b)
237 /*< window 'b' to 'a': source(b)-->destination(a) >*/
238 {
239 int iz,ix;
240
241 #ifdef _OPENMP
242 #pragma omp parallel for default(none)      \
243 private(ix,iz)              \
244 shared(b,a,nb,nz,nx)
245 #endif
246 for     (ix=0;ix<nx;ix++) {
247 for (iz=0;iz<nz;iz++) {
248 a[ix][iz]=b[nb+ix][nb+iz] ;
249 }
250 }
251 }
252
253
254 void prtm2d_init(bool verb_, bool csdgather_,
255                  float dz_, float dx_, float dt_,
256                  float amp, float fm,
257                  int nz_, int nx_, int nb_, int nt_, int ns_, int ng_,
258                  int sxbeg_, int szbeg_, int jsx_, int jsz_,
259                  int gxbeg_, int gzbeg_, int jgx_, int jgz_,
260                  float **v0, float *mod_, float *dat_)
261  !  GOTO main( )
262 /*< allocate variables and initialize parameters >*/
263 {
264     #ifdef _OPENMP
265         omp_init();
266     #endif
267     /* initialize OpenMP support */
268
269     float t;
270     int i1, i2, it,ib;
271     t = 1.0/(dz_*dz_);
272     c11 = 4.0*t/3.0;
273     c12= -t/12.0;
274     t = 1.0/(dx_*dx_);
275     c21 = 4.0*t/3.0;
276     c22= -t/12.0;
277     c0=-2.0*(c11+c12+c21+c22);
278     /* finite difference */
279
280     verb=verb_;
```

```
281     csdgather=csdgather_;
282     ns=ns_;
283     ng=ng_;
284     nb=nb_;
285     nz=nz_;
286     nx=nx_;
287     nt=nt_;
288     sxbeg=sxbeg_;
289     szbeg=szbeg_;
290     jsx=jsx_;
291     jsz=jsz_;
292     gxbeg=gxbeg_;
293     gzbeg=gzbeg_;
294     jgx=jgx_;
295     jgz=jgz_;
296
297     nzpad=nz+2*nb;
298     nxpad=nx+2*nb;
299
300     //! allocate temporary arrays
301     bndr=sf_floatalloc(nb);
302     sp0=sf_floatalloc2(nzpad,nxpad);
303     sp1=sf_floatalloc2(nzpad,nxpad);
304     gp0=sf_floatalloc2(nzpad,nxpad);
305     gp1=sf_floatalloc2(nzpad,nxpad);
306     vv=sf_floatalloc2(nzpad,nxpad);
307     wlt=sf_floatalloc(nt);
308     sxz=sf_intalloc(ns);
309     gxz=sf_intalloc(ng);
310     rwbndr=sf_floatalloc(nt*4*(nx+nz));
311
312     //! initialized sponge ABC coefficients
313     for(ib=0;ib<nb;ib++){
314         t=0.015*(nb-1-ib);
315         bndr[ib]=expf(-t*t);
316     }
317     mod=mod_;
318     dat=dat_;
319     //! initialize model
320     for (i2=0; i2<nx; i2++){
321         for (i1=0; i1<nz; i1++){
322             t=v0[i2][i1]*dt_;
323             vv[i2+nb][i1+nb] = t*t;
324         }
325     }
326     for (i2=0; i2<nxpad; i2++){
327         for (i1=0; i1<nb; i1++){
```

```
328             vv[i2][   i1  ] =vv[i2][   nb  ];
329             vv[i2][nzpad-i1-1] =vv[i2][nzpad-nb-1];
330         }
331     }
332     for (i2=0; i2<nb; i2++){
333         for (i1=0; i1<nzpad; i1++){
334             vv[   i2  ][i1] =vv[   nb  ][i1];
335             vv[nxpad-i2-1][i1] =vv[nxpad-nb-1][i1];
336         }
337     }
338     //! initialize source
339     for(it=0; it<nt; it++){
340         t=SF_PI*fm*(it*dt_-1.0/fm);t=t*t;
341         wlt[it]=amp*(1.0-2.0*t)*expf(-t);
342     }
343 
344     //! configuration of sources and geophones
345     if (!(sxbeg>=0 && szbeg>=0 &&
346           sxbeg+(ns-1)*jsx<nx && szbeg+(ns-1)*jsz<nz)) {
347             sf_warning("sources exceeds the computing zone!");
348             exit(1);
349     }
350     sg_init(sxz, szbeg, sxbeg, jsz, jsx, ns);
351     distx=sxbeg-gxbeg;
352     distz=szbeg-gzbeg;
353     if (!(gxbeg>=0 && gzbeg>=0 &&
354           gxbeg+(ng-1)*jgx<nx && gzbeg+(ng-1)*jgz<nz)) {
355             sf_warning("geophones exceeds the computing zone!");
356             exit(1);
357     }
358     if (csdgather &&
359         !((sxbeg+(ns-1)*jsx)+(ng-1)*jgx-distx <nx && (szbeg+(ns-1)*jsz)+(ng-1)↩
              *jgz-distz <nz)){
360             sf_warning("geophones exceeds the computing zone!");
361             exit(1);
362     }
363     sg_init(gxz, gzbeg, gxbeg, jgz, jgx, ng);
364 }
365 
366 void prtm2d_close()
367 /*< free allocated variables >*/
368 {
369 free(bndr);
370 free(*sp0); free(sp0);
371 free(*sp1); free(sp1);
372 free(*gp0); free(gp0);
373 free(*gp1); free(gp1);
```

```
374 free(*vv); free(vv);
375 free(wlt);
376 free(sxz);
377 free(gxz);
378 }
379
380 void prtm2d_lop(bool adj, bool add, int nm, int nd, float *mod, float *dat)
381  !  GOTO main( )
382 /*< prtm2d linear operator >*/
383 {
384     int i1,i2,it,is,ig, gx, gz;
385     if(nm!=nx*nz) sf_error("model size mismatch: %d!=%d",nm, nx*nz);
386     if(nd!=nt*ng*ns) sf_error("data size mismatch: %d!=%d",nd,nt*ng*ns);
387     sf_adjnull(adj, add, nm, nd, mod, dat);
388      !  GOTO sf_adjnull( )
389     /* set mod = 0.0f */
390
391     for(is=0; is<ns; is++) {
392         //! initialize is-th source wavefield Ps[]
393         memset(sp0[0], 0, nzpad*nxpad*sizeof(float));
394         memset(sp1[0], 0, nzpad*nxpad*sizeof(float));
395         memset(gp0[0], 0, nzpad*nxpad*sizeof(float));
396         memset(gp1[0], 0, nzpad*nxpad*sizeof(float));
397
398         if(csdgather){
399             gxbeg=sxbeg+is*jsx-distx;
400             sg_init(gxz, gzbeg, gxbeg, jgz, jgx, ng);
401         }
402
403         if(adj){
404             //! migration: mm=Lt dd
405             for(it=0; it<nt; it++){
406                 add_source(&sxz[is], sp1, 1, &wlt[it], true);
407                 step_forward(sp0, sp1, vv, false);
408                  !  GOTO step_forward( )
409                 apply_sponge(sp0);
410                 apply_sponge(sp1);
411                 ptr=sp0; sp0=sp1; sp1=ptr;
412                 boundary_rw(sp0, &rwbndr[it*4*(nx+nz)], false);
413                  !  GOTO boundary_rw( )
414             }
415
416             for (it=nt-1; it >-1; it--) {
417             /* reverse time order, Img[]+=Ps[]* Pg[]; */
418                 if(verb) sf_warning("%d;",it);
419
420                 //! reconstruct source wavefield Ps[]
```

```
421                 boundary_rw(sp0, &rwbndr[it*4*(nx+nz)], true);
422                 !  GOTO boundary_rw( )
423                 ptr=sp0; sp0=sp1; sp1=ptr;
424                 step_forward(sp0, sp1, vv, false);
425                 add_source(&sxz[is], sp1, 1, &wlt[it], false);
426 
427                 //! backpropagate receiver wavefield
428                 for(ig=0;ig<ng; ig++){
429                     gx=gxz[ig]/nz;
430                     gz=gxz[ig]%nz;
431                     gp1[gx+nb][gz+nb]+=dat[it+ig*nt+is*nt*ng];
432                 }
433                 step_forward(gp0, gp1, vv, false);
434                 apply_sponge(gp0);
435                 apply_sponge(gp1);
436                 ptr=gp0; gp0=gp1; gp1=ptr;
437 
438                 //! rtm imaging condition
439                 for(i2=0; i2<nx; i2++)
440                     for(i1=0; i1<nz; i1++)
441                         mod[i1+nz*i2]+=sp0[i2+nb][i1+nb]*gp1[i2+nb][i1+nb];
442             }
443     } else {
444         //! Born modeling/demigration: dd=L mm
445         for(it=0; it<nt; it++){
446         /* forward time order, Pg[]+=Ps[]* Img[]; */
447         if(verb) sf_warning("%d;",it);
448 
449         for(i2=0; i2<nx; i2++)
450             for(i1=0; i1<nz; i1++)
451                 gp1[i2+nb][i1+nb]+=sp0[i2+nb][i1+nb]*mod[i1+nz*i2];
452                 /* Born source */
453                 ptr=gp0; gp0=gp1; gp1=ptr;
454                 apply_sponge(gp0);
455                 apply_sponge(gp1);
456                 step_forward(gp0, gp1, vv, true);
457 
458                 for(ig=0;ig<ng; ig++){
459                     gx=gxz[ig]/nz;
460                     gz=gxz[ig]%nz;
461                     dat[it+ig*nt+is*nt*ng]+=gp1[gx+nb][gz+nb];
462                 }
463 
464                 add_source(&sxz[is], sp1, 1, &wlt[it], true);
465                 step_forward(sp0, sp1, vv, false);
466                 apply_sponge(sp0);
467                 apply_sponge(sp1);
```

```
                ptr=sp0; sp0=sp1; sp1=ptr;
            }
        }
    }
}

void sf_adjnull(bool adj /* adjoint flag */,
                bool add /* addition flag */,
                int nx   /* size of x */,
                int ny   /* size of y */,
                float* x,
                float* y)
 !  GO BACK, $(RSFROOT)/src/build/api/c/adjnull.c
/*< Zeros out the output (unless add is true).
Useful first step for any linear operator. >*/
{
    int i;

    if(add) return;

    if(adj) {
        for (i = 0; i < nx; i++) {
            x[i] = 0.;
        }
    } else {
        for (i = 0; i < ny; i++) {
            y[i] = 0.;
        }
    }
}
```