In this homework, you will write code for time-domain acoustic finite-differences modeling. This is one of the most basic programs employed in seismic imaging. I used in class for this program the name of "Hello World!" of seismic imaging. Everyone involved in seismic imaging must write this program once in their life. This is what you will do in this homework.

Your assignment is to modify an acoustic finite-differences modeling program and compute wavefields and data recorded on the surface. You will use the constant-density and the variable-density acoustic wave-equations.

This is an individual assignment and absolutely no collaboration on code is allowed.

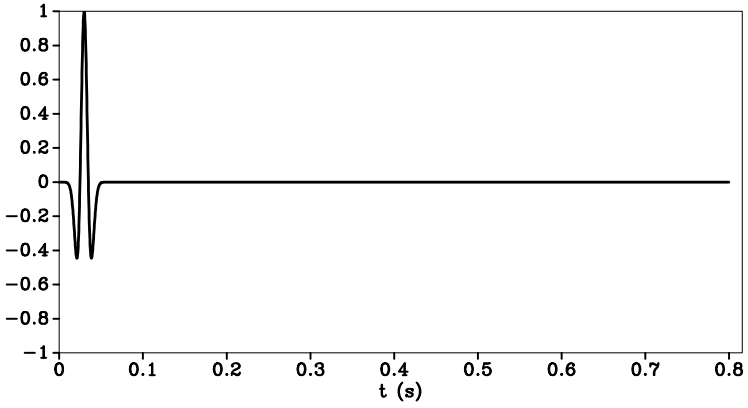

Figure 1: Source wavelet.

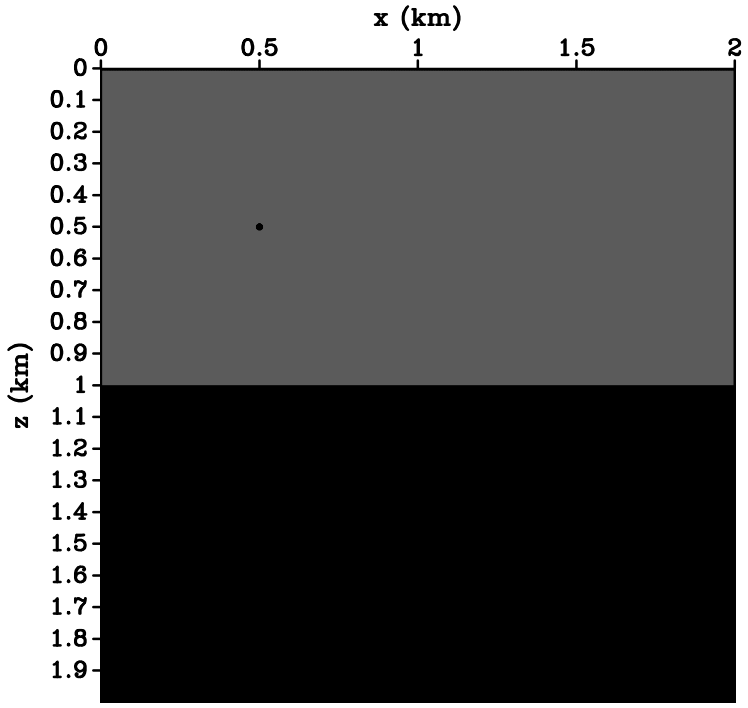

Figure 2: Velocity model.

EXERCISE

1. The program `AFDM.c` implements time-domain finite-differences modeling for the constant-density acoustic wave-equation. Your task is to add the density term to this program. Refer to the course slides for details about what needs to be added and where. Add comments in the code to indicate your modifications.

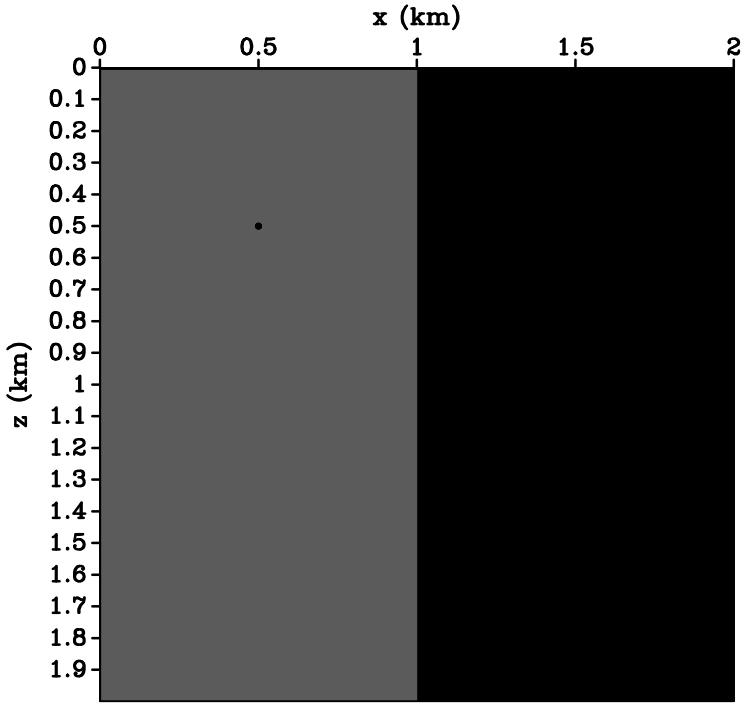

Figure 3: Density model.

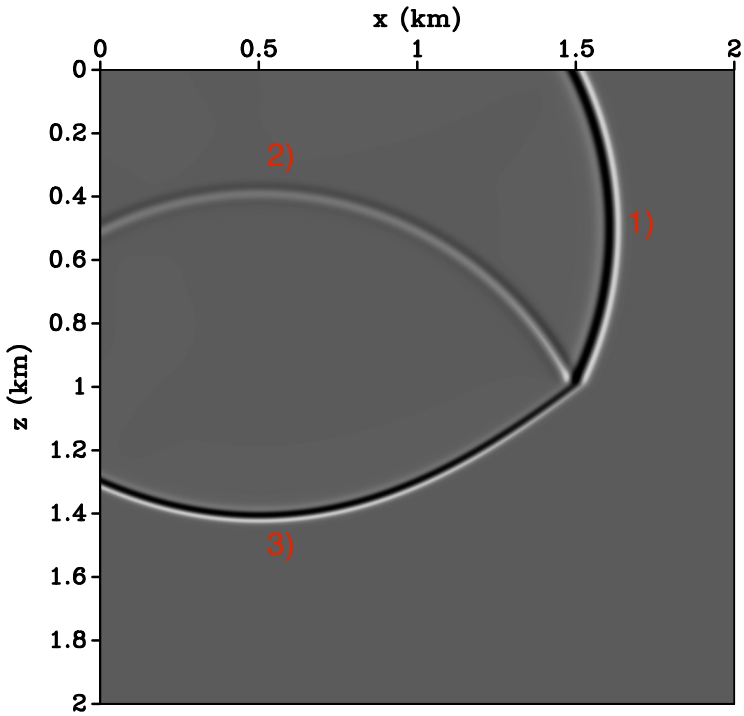

Figure 4: Wavefield with constant density.

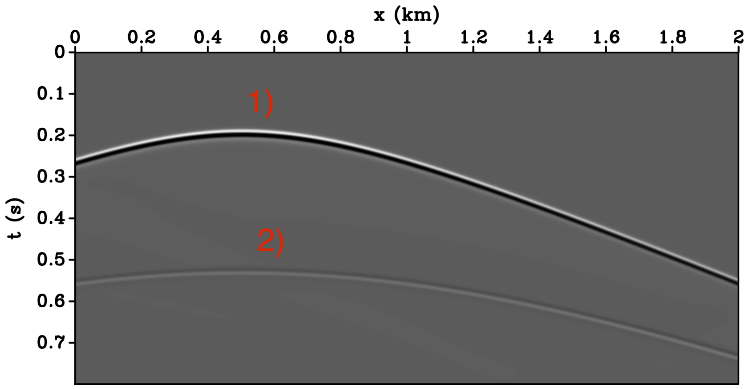

Figure 5: Data with constant density.

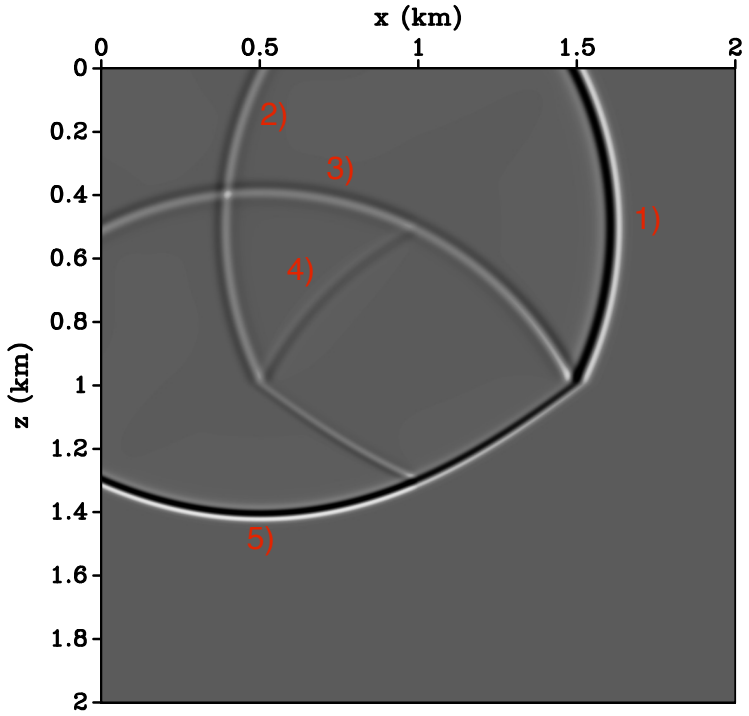

Figure 6: Wavefield with variable density.

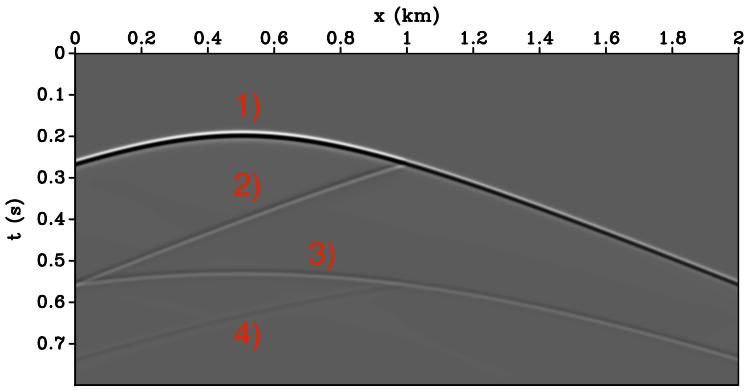

Figure 7: Data with variable density.

2. Run `scons view` after your code is modified. All figures are rebuilt with your new code and displayed on screen.

3. Run `scons lock` once you are satisfied with your results. All figures are copied to the storage directory.

4. Add comments to this document indicating the changes to the simulated data and wavefields. Are your results expected? Describe the data and wavefield figures indicating what the various events represent.
   For the constant-density acoustic wave-equations, we do not need to make any changes to the codes.
   For the variable-density acoustic wave equations, please check code changes indicated by the gray background in the `AFDM.c` attached in the end of the report.
   In Figures 4 and 5 for the constant-density case, the events marked by 1) and 2) in the wave-field figure (Figure 4) are corresponding the events marked by 1) and 2) in the data figure (Figure 5). The event marked by 3) in Figure 4 cannot be received and therefore there is no corresponding event in the recorded data shown in Figure 5. Since the density is constant, there is only one horizontal interface at z =1km (velocity model in Figure 2) that generates reflection events as marked by 2) in Figures 4 and 5.
   In Figures 6 and 7 for the variable-density case, the events marked by 1), 2), 3) and 4) in the wave-field figure (Figure 6) are corresponding the events marked by 1), 2), 3) and 4) in the data figure (Figure 7). The event marked by 5) in Figure 6 cannot be received and therefore there is no corresponding event in the recorded data shown in Figure 7. Since the density has a vertical interface at x =1km, and the velocity has a horizontal interface at z =1km. The vertical interface generates events marked by 2) in the Figures 6 and 7. The horizontal interface generated the events marked by 3) in the Figures 6 and 7. The events marked by 4) in Figures 6 and 7 are the scattered waves at the intersect point of the vertical and horizontal interface.

5. `cd awe`, run `scons handout.read` to build your answer. A PDF file is constructed using your newly created figures and modifications to the text. The modified code is automatically added to the document.

WRAP-UP

After you are satisfied that your document looks ok, print it from the PDF viewer and bring it to class.

```
/* 2D acoustic time-domain FD modeling */
/*
  Copyright (C) 2007 Colorado School of Mines

  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 "fdutil.h"
#include "omputil.h"

/* check: dt<= 0.2 * min(dx,dz)/vmin */

#define NOP 2 /* derivative operator half-size */

#define C0 -2.500000 /*    c0=-30./12.; */
#define CA +1.333333 /*    ca=+16./12.; */
#define CB -0.083333 /*    cb=- 1./12.; */

#define C1  0.6666666666666666666 /*  2/3  */
#define C2 -0.0833333333333333333 /* -1/12 */

/* centered FD derivative stencils */
#define DX(a,ix,iz,s) (C2*(a[ix+2][iz  ] - a[ix-2][iz  ]) +  \
                       C1*(a[ix+1][iz  ] - a[ix-1][iz  ])  )*s
#define DZ(a,ix,iz,s) (C2*(a[ix  ][iz+2] - a[ix  ][iz-2]) +  \
                       C1*(a[ix  ][iz+1] - a[ix  ][iz-1])  )*s

int main(int argc, char* argv[])
{
    bool verb,fsrf,snap,expl,dabc;
    int  jsnap,ntsnap,jdata;

    /* OMP parameters */
#ifdef _OPENMP
    int ompnth;
#endif

    /* I/O files */
    sf_file Fwav=NULL; /* wavelet   */
    sf_file Fsou=NULL; /* sources   */
    sf_file Frec=NULL; /* receivers */
    sf_file Fvel=NULL; /* velocity  */
    sf_file Fden=NULL; /* density   */
    sf_file Fdat=NULL; /* data      */
```

```
61      sf_file Fwfl=NULL; /* wavefield */
62
63      /* cube axes */
64      sf_axis at,az,ax;
65      sf_axis as,ar;
66
67      int     nt,nz,nx,ns,nr,nb;
68      int     it,iz,ix;
69      float   dt,dz,dx,idz,idx;
70
71      /* FDM structure */
72      fdm2d    fdm=NULL;
73      abcone2d abc=NULL;
74      sponge   spo=NULL;
75
76      /* I/O arrays */
77      float  *ww=NULL;           /* wavelet   */
78      pt2d   *ss=NULL;           /* sources   */
79      pt2d   *rr=NULL;           /* receivers */
80      float  *dd=NULL;           /* data      */
81
82      float **tt=NULL;
83      float **ro=NULL;           /* density */
84      float **dr=NULL;           /* reciprocal density */
85      float **vp=NULL;           /* velocity */
86      float **vt=NULL;           /* temporary vp*vp * dt*dt */
87
88      float **um,**uo,**up,**ua,**ut; /* wavefield: um = U @ t-1; uo = U @ t; up = U @ t+1 */
89
90      /* linear interpolation weights/indices */
91      lint2d cs,cr;
92
93      /* FD operator size */
94      float co,cax,cbx,caz,cbz;
95
96      /* wavefield cut params */
97      sf_axis   acz=NULL,acx=NULL;
98      int       nqz,nqx;
99      float     oqz,oqx;
100     float     dqz,dqx;
101     float     **uc=NULL;
102
103     /*------------------------------------------------------------*/
104     /* init RSF */
105     sf_init(argc,argv);
106
107     /*------------------------------------------------------------*/
108     /* OMP parameters */
109 #ifdef _OPENMP
110     ompnth=omp_init();
111 #endif
112     /*------------------------------------------------------------*/
113
114     if(! sf_getbool("verb",&verb)) verb=false; /* verbosity flag */
115     if(! sf_getbool("snap",&snap)) snap=false; /* wavefield snapshots flag */
116     if(! sf_getbool("free",&fsrf)) fsrf=false; /* free surface flag */
117     if(! sf_getbool("expl",&expl)) expl=false; /* "exploding reflector" */
118     if(! sf_getbool("dabc",&dabc)) dabc=false; /* absorbing BC */
119     /*------------------------------------------------------------*/
120
121     /*------------------------------------------------------------*/
122     /* I/O files */
```

```
    Fwav = sf_input ("in" ); /* wavelet   */
    Fvel = sf_input ("vel"); /* velocity  */
    Fsou = sf_input ("sou"); /* sources   */
    Frec = sf_input ("rec"); /* receivers */
    Fwfl = sf_output("wfl"); /* wavefield */
    Fdat = sf_output("out"); /* data      */

    if (NULL != sf_getstring("den")) {
            Fden = sf_input ("den"); /* density   */
    } else {
            Fden = NULL;
    }

    /*------------------------------------------------------------*/
    /* axes */
    at = sf_iaxa(Fwav,2); sf_setlabel(at,"t"); if(verb) sf_raxa(at); /* time */

    ax = sf_iaxa(Fvel,2); sf_setlabel(ax,"x"); if(verb) sf_raxa(ax); /* space */
    az = sf_iaxa(Fvel,1); sf_setlabel(az,"z"); if(verb) sf_raxa(az); /* depth */

    as = sf_iaxa(Fsou,2); sf_setlabel(as,"s"); if(verb) sf_raxa(as); /* sources */
    ar = sf_iaxa(Frec,2); sf_setlabel(ar,"r"); if(verb) sf_raxa(ar); /* receivers */

    nt = sf_n(at); dt = sf_d(at);
    nz = sf_n(az); dz = sf_d(az);
    nx = sf_n(ax); dx = sf_d(ax);

    ns = sf_n(as);
    nr = sf_n(ar);
    /*------------------------------------------------------------*/

    /*------------------------------------------------------------*/
    /* other execution parameters */
    if(! sf_getint("jdata",&jdata)) jdata=1;
    if(snap) {  /* save wavefield every *jsnap* time steps */
        if(! sf_getint("jsnap",&jsnap)) jsnap=nt;
    }
    /*------------------------------------------------------------*/

    /*------------------------------------------------------------*/
    /* expand domain for FD operators and ABC */
    if( !sf_getint("nb",&nb) || nb<NOP) nb=NOP;

    fdm=fdutil_init(verb,fsrf,az,ax,nb,1);

    sf_setn(az,fdm->nzpad); sf_seto(az,fdm->ozpad);
    sf_setn(ax,fdm->nxpad); sf_seto(ax,fdm->oxpad);
    /*------------------------------------------------------------*/

    /*------------------------------------------------------------*/
    /* setup output data header */
    sf_oaxa(Fdat,ar,1);

    sf_setn(at,nt/jdata);
    sf_setd(at,dt*jdata);
    sf_oaxa(Fdat,at,2);

    /* setup output wavefield header */
    if(snap) {
            if(!sf_getint  ("nqz",&nqz)) nqz=sf_n(az);
            if(!sf_getint  ("nqx",&nqx)) nqx=sf_n(ax);

```

```
185             if(!sf_getfloat("oqz",&oqz)) oqz=sf_o(az);
186             if(!sf_getfloat("oqx",&oqx)) oqx=sf_o(ax);
187
188             dqz=sf_d(az);
189             dqx=sf_d(ax);
190
191             acz = sf_maxa(nqz,oqz,dqz);
192             acx = sf_maxa(nqx,oqx,dqx);
193         /* check if the imaging window fits in the wavefield domain */
194
195             uc=sf_floatalloc2(sf_n(acz),sf_n(acx));
196
197             ntsnap=0;
198       for(it=0; it<nt; it++) {
199               if(it%jsnap==0) ntsnap++;
200             }
201             sf_setn(at,  ntsnap);
202             sf_setd(at,dt*jsnap);
203             if(verb) sf_raxa(at);
204
205             sf_oaxa(Fwfl,acz,1);
206             sf_oaxa(Fwfl,acx,2);
207             sf_oaxa(Fwfl,at, 3);
208     }
209
210     if(expl) ww = sf_floatalloc( 1);
211     else     ww = sf_floatalloc(ns);
212     dd = sf_floatalloc(nr);
213
214     /*------------------------------------------------------------*/
215     /* setup source/receiver coordinates */
216     ss = (pt2d*) sf_alloc(ns,sizeof(*ss));
217     rr = (pt2d*) sf_alloc(nr,sizeof(*rr));
218
219     pt2dread1(Fsou,ss,ns,2); /* read (x,z) coordinates */
220     pt2dread1(Frec,rr,nr,2); /* read (x,z) coordinates */
221
222     cs = lint2d_make(ns,ss,fdm);
223     cr = lint2d_make(nr,rr,fdm);
224
225     /*------------------------------------------------------------*/
226     /* setup FD coefficients */
227     idz = 1/dz;
228     idx = 1/dx;
229
230     co = C0 * (idx*idx+idz*idz);
231     cax= CA *  idx*idx;
232     cbx= CB *  idx*idx;
233     caz= CA *  idz*idz;
234     cbz= CB *  idz*idz;
235
236     /*------------------------------------------------------------*/
237     tt = sf_floatalloc2(nz,nx);
238
239     ro  =sf_floatalloc2(fdm->nzpad,fdm->nxpad);
240     dr  =sf_floatalloc2(fdm->nzpad,fdm->nxpad);
241     vp  =sf_floatalloc2(fdm->nzpad,fdm->nxpad);
242     vt  =sf_floatalloc2(fdm->nzpad,fdm->nxpad);
243
244     /* input density */
245     if (NULL != Fden) {
246             sf_floatread(tt[0],nz*nx,Fden);
```

```
247     } else {
248             for (ix=0; ix< nz*nx; ix++) tt[0][ix] = 1.0f;
249     }
250     expand(tt,ro ,fdm);
251
252     //free(*ro); free(ro);
253
254     /* input velocity */
255     sf_floatread(tt[0],nz*nx,Fvel );    expand(tt,vp,fdm);
256     /* precompute vp^2 * dt^2 */
257     for     (ix=0; ix<fdm->nxpad; ix++) {
258             for(iz=0; iz<fdm->nzpad; iz++) {
259               vt[ix][iz] = vp[ix][iz] * vp[ix][iz] * dt*dt;
260             }
261     }
262     if(fsrf) { /* free surface */
263             for     (ix=0; ix<fdm->nxpad; ix++) {
264               for(iz=0; iz<fdm->nb; iz++) {
265                       vt[ix][iz]=0;
266               }
267             }
268     }
269
270     free(*tt); free(tt);
271     /*------------------------------------------------------------*/
272
273     /*------------------------------------------------------------*/
274     /* allocate wavefield arrays */
275     um=sf_floatalloc2(fdm->nzpad,fdm->nxpad);
276     uo=sf_floatalloc2(fdm->nzpad,fdm->nxpad);
277     up=sf_floatalloc2(fdm->nzpad,fdm->nxpad);
278     ua=sf_floatalloc2(fdm->nzpad,fdm->nxpad);
279
280     for     (ix=0; ix<fdm->nxpad; ix++) {
281             for(iz=0; iz<fdm->nzpad; iz++) {
282               um[ix][iz]=0;
283               uo[ix][iz]=0;
284               up[ix][iz]=0;
285               ua[ix][iz]=0;
286             }
287     }
288
289     /*------------------------------------------------------------*/
290     if(dabc) {
291         /* one-way abc setup */
292             abc = abcone2d_make(NOP,dt,vp,fsrf,fdm);
293         /* sponge abc setup */
294             spo = sponge_make(fdm->nb);
295     }
296
297     /*------------------------------------------------------------*/
298     /*
299      *  MAIN LOOP
300      */
301     /*------------------------------------------------------------*/
302     if(verb) fprintf(stderr,"\n");
303     for (it=0; it<nt; it++) {
304           if(verb) fprintf(stderr,"\b\b\b\b\b%d",it);
305
306 #ifdef _OPENMP
307 #pragma omp parallel for                                 \
308     schedule(dynamic,fdm->ompchunk)                     \
```

```
    private(ix,iz)                                          \
    shared(fdm,ua,uo,ro,co,cax,caz,cbx,cbz,idx,idz)
#endif
  float cox = 0.25f*idx*idx; //coefficient for the density term
  float coz = 0.25f*idz*idz; //coefficient for the density term
        for    (ix=NOP; ix<fdm->nxpad-NOP; ix++) {
            for(iz=NOP; iz<fdm->nzpad-NOP; iz++) {

                // 4th order Laplacian operator
            ua[ix][iz] =
                    co * uo[ix  ][iz  ] +
                    cax*(uo[ix-1][iz  ] + uo[ix+1][iz  ]) +
                    cbx*(uo[ix-2][iz  ] + uo[ix+2][iz  ]) +
                    caz*(uo[ix  ][iz-1] + uo[ix  ][iz+1]) +
                    cbz*(uo[ix  ][iz-2] + uo[ix  ][iz+2]) -
       (cox*(ro[ix+1][iz  ]-ro[ix-1][iz  ])*(uo[ix+1][iz  ]-uo[ix-1][iz  ])+
        coz*(ro[ix  ][iz+1]-ro[ix  ][iz-1])*(uo[ix  ][iz+1]-uo[ix  ][iz-1]))/ro[ix][iz];
            }
        }

        /* inject acceleration source */
        if(expl) {
            sf_floatread(ww, 1,Fwav);
            lint2d_inject1(ua,ww[0],cs);
        } else {
            sf_floatread(ww,ns,Fwav);
            lint2d_inject(ua,ww,cs);
        }

        /* step forward in time */
#ifdef _OPENMP
#pragma omp parallel for            \
    schedule(dynamic,fdm->ompchunk) \
    private(ix,iz)                  \
    shared(fdm,ua,uo,um,up,vt)
#endif
        for    (ix=0; ix<fdm->nxpad; ix++) {
            for(iz=0; iz<fdm->nzpad; iz++) {
                up[ix][iz] = 2*uo[ix][iz]
                    -         um[ix][iz]
                    +         ua[ix][iz] * vt[ix][iz];
            }
        }
        /* circulate wavefield arrays */
        ut=um;
        um=uo;
        uo=up;
        up=ut;

        if(dabc) {
          /* one-way abc apply */
          abcone2d_apply(uo,um,NOP,abc,fdm);
          sponge2d_apply(um,spo,fdm);
          sponge2d_apply(uo,spo,fdm);
          sponge2d_apply(up,spo,fdm);
        }

        /* extract data */
        lint2d_extract(uo,dd,cr);

        if(snap && it%jsnap==0) {
          cut2d(uo,uc,fdm,acz,acx);
```

```
371             sf_floatwrite(uc[0],sf_n(acz)*sf_n(acx),Fwfl);
372           }
373           if(        it%jdata==0)
374             sf_floatwrite(dd,nr,Fdat);
375     }
376     if(verb) fprintf(stderr,"\n");
377
378       /*------------------------------------------------------------*/
379       /* deallocate arrays */
380       free(*um); free(um);
381       free(*up); free(up);
382       free(*uo); free(uo);
383       free(*ua); free(ua);
384       if(snap) {
385               free(*uc); free(uc);
386       }
387
388       free(*vp);  free(vp);
389       free(*vt);  free(vt);
390
391       free(ww);
392       free(ss);
393       free(rr);
394       free(dd);
395       /*------------------------------------------------------------*/
396
397       exit (0);
398   }
```