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.

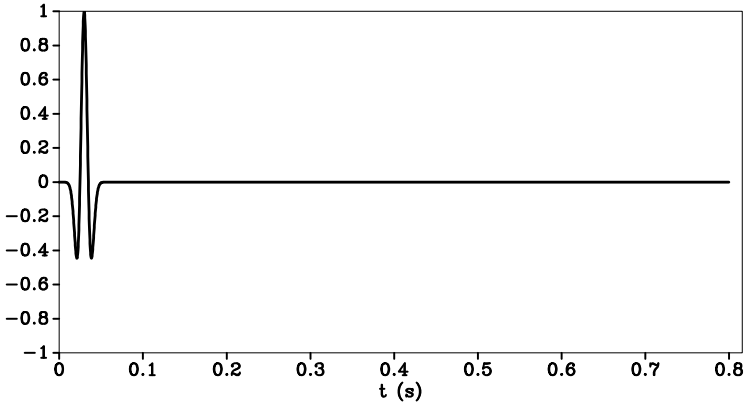

Figure 1: Source wavelet.

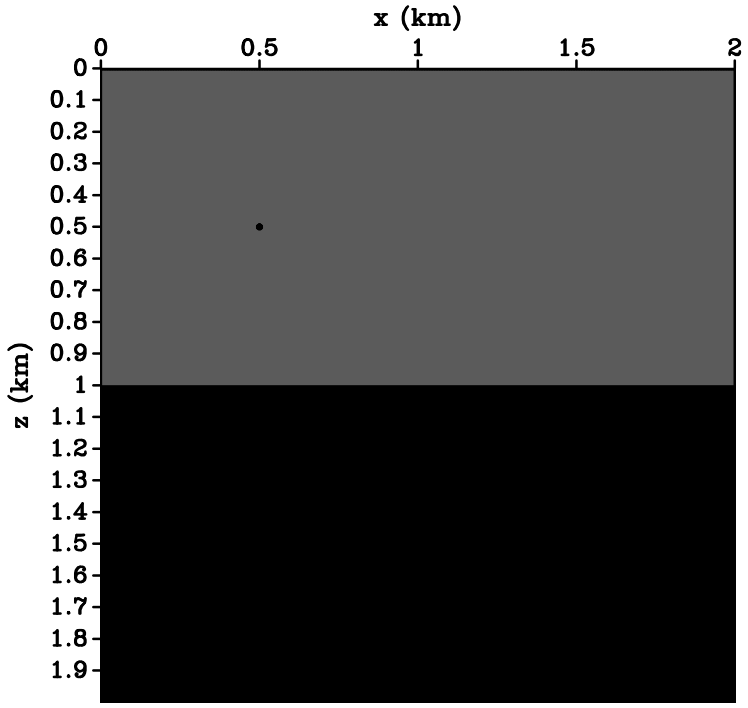

Figure 2: Velocity model.

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.

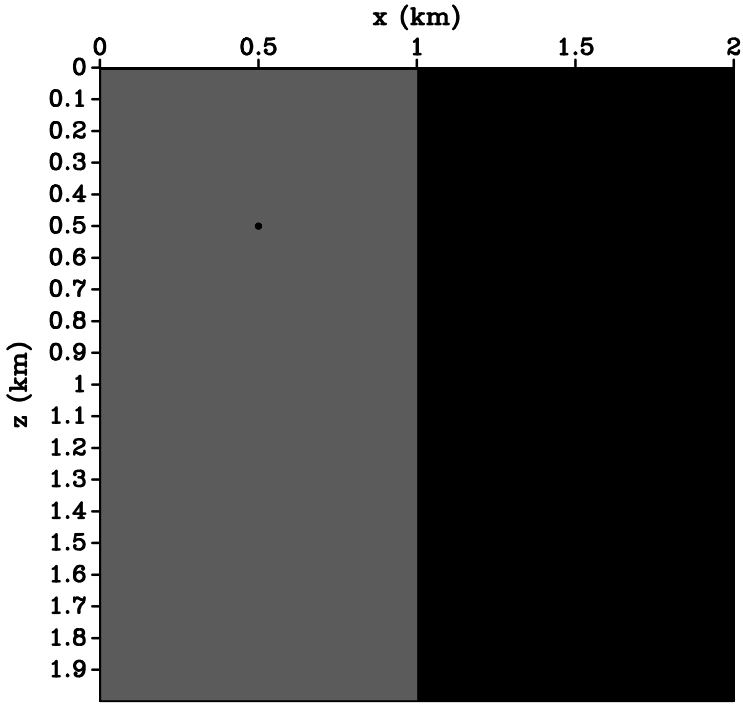

Figure 3: Density model.

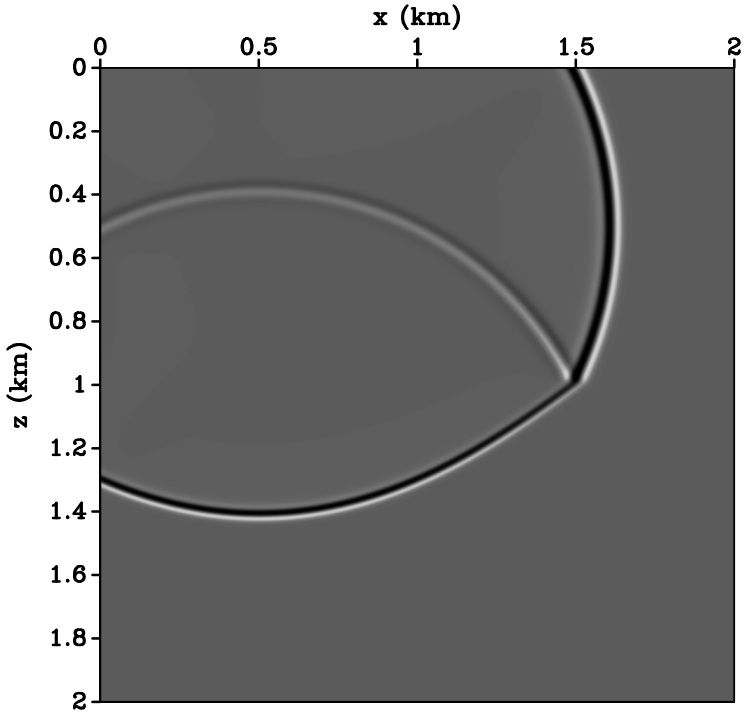

Figure 4: Wavefield.

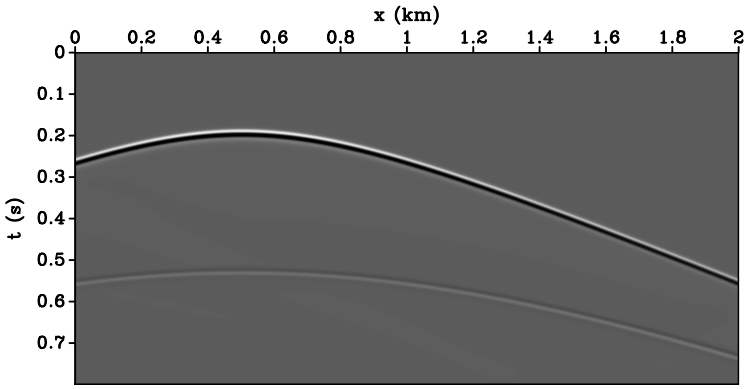

Figure 5: Data.

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.

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.

AFDM.C

```
/* 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  0.66666666666666666666 /*  2/3  */
#define C2 -0.08333333333333333333 /* -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      */
    sf_file Fwfl=NULL; /* wavefield */

    /* cube axes */
    sf_axis at,az,ax;
    sf_axis as,ar;

    int     nt,nz,nx,ns,nr,nb;
    int     it,iz,ix;
    float   dt,dz,dx,idz,idx;

    /* FDM structure */
    fdm2d    fdm=NULL;
    abcone2d abc=NULL;
    sponge   spo=NULL;

    /* I/O arrays */
    float  *ww=NULL;           /* wavelet   */
    pt2d   *ss=NULL;           /* sources   */
    pt2d   *rr=NULL;           /* receivers */
    float  *dd=NULL;           /* data      */

    float **tt=NULL;
    float **ro=NULL;           /* density */
    float **vp=NULL;           /* velocity */
    float **vt=NULL;           /* temporary vp*vp * dt*dt */

    float **um,**uo,**up,**ua,**ut; /* wavefield: um = U @ t-1; uo = U @ t; up = U @ t+1 */

    /* linear interpolation weights/indices */
    lint2d cs,cr;

    /* FD operator size */
    float co,cax,cbx,caz,cbz;

    /* wavefield cut params */
    sf_axis    acz=NULL,acx=NULL;
    int        nqz,nqx;
    float      oqz,oqx;
    float      dqz,dqx;
    float      **uc=NULL;

    /*------------------------------------------------------------*/
    /* init RSF */
    sf_init(argc,argv);

    /*------------------------------------------------------------*/
    /* OMP parameters */
#ifdef _OPENMP
    ompnth=omp_init();
#endif
```

```
/*------------------------------------------------------------*/

if(! sf_getbool("verb",&verb)) verb=false; /* verbosity flag */
if(! sf_getbool("snap",&snap)) snap=false; /* wavefield snapshots flag */
if(! sf_getbool("free",&fsrf)) fsrf=false; /* free surface flag */
if(! sf_getbool("expl",&expl)) expl=false; /* "exploding reflector" */
if(! sf_getbool("dabc",&dabc)) dabc=false; /* absorbing BC */
/*------------------------------------------------------------*/

/*------------------------------------------------------------*/
/* 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);

    if(!sf_getfloat("oqz",&oqz)) oqz=sf_o(az);
    if(!sf_getfloat("oqx",&oqx)) oqx=sf_o(ax);

    dqz=sf_d(az);
    dqx=sf_d(ax);

    acz = sf_maxa(nqz,oqz,dqz);
    acx = sf_maxa(nqx,oqx,dqx);
    /* check if the imaging window fits in the wavefield domain */

    uc=sf_floatalloc2(sf_n(acz),sf_n(acx));

    ntsnap=0;
    for(it=0; it<nt; it++) {
        if(it%jsnap==0) ntsnap++;
    }
    sf_setn(at, ntsnap);
    sf_setd(at,dt*jsnap);
    if(verb) sf_raxa(at);

    sf_oaxa(Fwfl,acz,1);
    sf_oaxa(Fwfl,acx,2);
    sf_oaxa(Fwfl,at,3);
}

if(expl) ww = sf_floatalloc(1);
else     ww = sf_floatalloc(ns);
dd = sf_floatalloc(nr);

/*------------------------------------------------------------*/
/* setup source/receiver coordinates */
ss = (pt2d*) sf_alloc(ns,sizeof(*ss));
rr = (pt2d*) sf_alloc(nr,sizeof(*rr));

pt2dread1(Fsou,ss,ns,2); /* read (x,z) coordinates */
pt2dread1(Frec,rr,nr,2); /* read (x,z) coordinates */

cs = lint2d_make(ns,ss,fdm);
cr = lint2d_make(nr,rr,fdm);
```

```
    /*------------------------------------------------------------*/
    /* setup FD coefficients */
    idz = 1/dz;
    idx = 1/dx;

    co = C0 * (idx*idx+idz*idz);
    cax= CA *  idx*idx;
    cbx= CB *  idx*idx;
    caz= CA *  idz*idz;
    cbz= CB *  idz*idz;

    /*------------------------------------------------------------*/
    tt = sf_floatalloc2(nz,nx);

    ro  =sf_floatalloc2(fdm->nzpad,fdm->nxpad);
    vp  =sf_floatalloc2(fdm->nzpad,fdm->nxpad);
    vt  =sf_floatalloc2(fdm->nzpad,fdm->nxpad);

    /* input density */
    if (NULL != Fden) {
        sf_floatread(tt[0],nz*nx,Fden);
    } else {
        for (ix=0; ix< nz*nx; ix++) tt[0][ix] = 1.0f;
    }
    expand(tt,ro ,fdm);

    free(*ro); free(ro);

    /* input velocity */
    sf_floatread(tt[0],nz*nx,Fvel );     expand(tt,vp,fdm);
    /* precompute vp^2 * dt^2 */
    for     (ix=0; ix<fdm->nxpad; ix++) {
        for(iz=0; iz<fdm->nzpad; iz++) {
            vt[ix][iz] = vp[ix][iz] * vp[ix][iz] * dt*dt;
        }
    }
    if(fsrf) { /* free surface */
        for     (ix=0; ix<fdm->nxpad; ix++) {
            for(iz=0; iz<fdm->nb; iz++) {
                vt[ix][iz]=0;
            }
        }
    }

    free(*tt); free(tt);
    /*------------------------------------------------------------*/

    /*------------------------------------------------------------*/
    /* allocate wavefield arrays */
    um=sf_floatalloc2(fdm->nzpad,fdm->nxpad);
    uo=sf_floatalloc2(fdm->nzpad,fdm->nxpad);
    up=sf_floatalloc2(fdm->nzpad,fdm->nxpad);
    ua=sf_floatalloc2(fdm->nzpad,fdm->nxpad);

    for     (ix=0; ix<fdm->nxpad; ix++) {
        for(iz=0; iz<fdm->nzpad; iz++) {
            um[ix][iz]=0;
            uo[ix][iz]=0;
            up[ix][iz]=0;
            ua[ix][iz]=0;
        }
    }

    /*------------------------------------------------------------*/
    if(dabc) {
        /* one-way abc setup */
        abc = abcone2d_make(NOP,dt,vp,fsrf,fdm);
        /* sponge abc setup */
        spo = sponge_make(fdm->nb);
    }

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

#ifdef _OPENMP
#pragma omp parallel for                                \
    schedule(dynamic,fdm->ompchunk)                     \
    private(ix,iz)                                      \
    shared(fdm,ua,uo,co,cax,caz,cbx,cbz,idx,idz)
#endif
        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]);
            }
        }

        /* 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);
          sf_floatwrite(uc[0],sf_n(acz)*sf_n(acx),Fwfl);
        }
        if(         it%jdata==0)
          sf_floatwrite(dd,nr,Fdat);
  }
  if(verb) fprintf(stderr,"\n");

   /*------------------------------------------------------------*/
   /* deallocate arrays */
   free(*um); free(um);
   free(*up); free(up);
   free(*uo); free(uo);
   free(*ua); free(ua);
   if(snap) {
           free(*uc); free(uc);
   }

   free(*vp);  free(vp);
   free(*vt);  free(vt);

   free(ww);
   free(ss);
   free(rr);
   free(dd);
   /*------------------------------------------------------------*/

   exit (0);
}
```