Full Waveform Inversion

Hou, Sian - sianhou1987@outlook.com Jan/01/2017

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  int main(int argc, char *argv[]) {
2
3      //! variables on host
4      bool verb, precon, csdgather;
5      int is, it, iter, niter, distx, distz, csd, rbell;
6      int nz, nx, nt, ns, ng;
7
8      //! parameters of acquisition geometery
9      int sxbeg, szbeg, gxbeg, gzbeg, jsx, jsz, jgx, jgz;
10     float dx, dz, fm, dt, dtx, dtz, tmp, amp, obj1, obj, beta, epsil, alpha;
11     float *dobs, *dcal, *derr, *wlt, *bndr, *trans, *objval;
12     int *sxz, *gxz;
13     float **vv, **illum, **lap, **vtmp, **sp0, **sp1, **sp2, **gp0, **gp1, **↩
           gp2, **g0, **g1, **cg, *alpha1, *alpha2, **ptr=NULL;
14
15     //! time
16     clock_t start, stop;
17
18     //! I/O files
19     sf_file vinit, shots, vupdates, grads, objs, illums;
20
```

```
    //! initialize Madagascar
    sf_init(argc,argv);

    //! set up I/O files
    vinit=sf_input ("in");
    /* initial velocity model, unit=m/s */
    shots=sf_input("shots");
    /* recorded shots from exact velocity model */
    vupdates=sf_output("out");
    /* updated velocity in iterations */
    grads=sf_output("grads");
    /* gradient in iterations */
    illums=sf_output("illums");
    /* source illumination in iterations */
    objs=sf_output("objs");
    /* values of objective function in iterations */

    //! get parameters from velocity model and recorded shots
    if (!sf_getbool("verb",&verb))      verb=true;
    /* vebosity */
    if (!sf_histint(vinit,"n1",&nz))    sf_error("no n1");
    /* nz */
    if (!sf_histint(vinit,"n2",&nx))    sf_error("no n2");
    /* nx */
    if (!sf_histfloat(vinit,"d1",&dz))  sf_error("no d1");
    /* dz */
    if (!sf_histfloat(vinit,"d2",&dx))  sf_error("no d2");
    /* dx */
    if (!sf_getbool("precon",&precon))  precon=false;
    /* precondition or not */
    if (!sf_getint("niter",&niter))     niter=100;
    /* number of iterations */
    if (!sf_getint("rbell",&rbell))     rbell=2;
    /* radius of bell smooth */

    if (!sf_histint(shots,"n1",&nt))    sf_error("no nt");
    /* total modeling time steps */
    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 */
```

```c
    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");
    /* z-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 */

    //! set up I/O parameters
    sf_putint(vupdates,"n1",nz);
    sf_putint(vupdates,"n2",nx);
    sf_putint(vupdates,"n3",niter);
    sf_putfloat(vupdates,"d1",dz);
    sf_putfloat(vupdates,"d2",dx);
    sf_putint(vupdates,"d3",1);
    sf_putint(vupdates,"o3",1);
    sf_putstring(vupdates,"label1","Depth");
    sf_putstring(vupdates,"label2","Distance");
    sf_putstring(vupdates,"label3","Iteration");
    /* updated velocity in iterations */
    sf_putint(grads,"n1",nz);
    sf_putint(grads,"n2",nx);
    sf_putint(grads,"n3",niter);
    sf_putfloat(grads,"d1",dz);
    sf_putfloat(grads,"d2",dx);
    sf_putint(grads,"d3",1);
    sf_putint(grads,"o3",1);
    sf_putstring(grads,"label1","Depth");
    sf_putstring(grads,"label2","Distance");
    sf_putstring(grads,"label3","Iteration");
    /* gradient in iterations */
    sf_putint(illums,"n1",nz);
    sf_putint(illums,"n2",nx);
    sf_putint(illums,"n3",niter);
    sf_putfloat(illums,"d1",dz);
    sf_putfloat(illums,"d2",dx);
    sf_putint(illums,"d3",1);
    sf_putint(illums,"o3",1);
    /* source illumination in iterations */
    sf_putint(objs,"n1",niter);
    sf_putint(objs,"n2",1);
    sf_putfloat(objs,"d1",1);
    sf_putfloat(objs,"o1",1);
    /* values of objective function in iterations */
    dtx=dt/dx;
    dtz=dt/dz;
    csdgather=(csd>0)?true:false;

    //! allocate memory
    vv=sf_floatalloc2(nz, nx);
    /* updated velocity */
    vtmp=sf_floatalloc2(nz, nx);
    /* temporary velocity computed with epsil */
    sp0=sf_floatalloc2(nz, nx);
    /* source wavefield p0 */
    sp1=sf_floatalloc2(nz, nx);
    /* source wavefield p1 */
    sp2=sf_floatalloc2(nz, nx);
    /* source wavefield p2 */
    gp0=sf_floatalloc2(nz, nx);
    /* geophone/receiver wavefield p0 */
    gp1=sf_floatalloc2(nz, nx);
    /* geophone/receiver wavefield p1 */
    gp2=sf_floatalloc2(nz, nx);
    /* geophone/receiver wavefield p2 */
    g0=sf_floatalloc2(nz, nx);
    /* gradient at previous step */
    g1=sf_floatalloc2(nz, nx);
    /* gradient at curret step */
    cg=sf_floatalloc2(nz, nx);
    /* conjugate gradient */
    lap=sf_floatalloc2(nz, nx);
    /* laplace of the source wavefield */
    illum=sf_floatalloc2(nz, nx);
    /* illumination of the source wavefield */
    objval=(float*)malloc(niter*sizeof(float));
    /* objective/misfit function */
    wlt=(float*)malloc(nt*sizeof(float));
    /* ricker wavelet */
    sxz=(int*)malloc(ns*sizeof(int));
    /* source positions */
    gxz=(int*)malloc(ng*sizeof(int));
    /* geophone positions */
    bndr=(float*)malloc(nt*(2*nz+nx)*sizeof(float));
    /* boundaries for wavefield reconstruction */
    trans=(float*)malloc(ng*nt*sizeof(float));
    /* transposed one shot */
    dobs=(float*)malloc(ng*nt*sizeof(float));
    /* observed seismic data */
    dcal=(float*)malloc(ng*sizeof(float));
    /* calculated/synthetic seismic data */
    derr=(float*)malloc(ns*ng*nt*sizeof(float));
    /* residual/error between synthetic and observation */
    alpha1=(float*)malloc(ng*sizeof(float));
    /* numerator of alpha, length=ng */
    alpha2=(float*)malloc(ng*sizeof(float));
    /* denominator of alpha, length=ng */

    //! initialize varibles
    sf_floatread(vv[0], nz*nx, vinit);
    memset(sp0[0], 0, nz*nx*sizeof(float));
    memset(sp1[0], 0, nz*nx*sizeof(float));
    memset(sp2[0], 0, nz*nx*sizeof(float));
    memset(gp0[0], 0, nz*nx*sizeof(float));
    memset(gp1[0], 0, nz*nx*sizeof(float));
    memset(gp2[0], 0, nz*nx*sizeof(float));
    memset(g0[0], 0, nz*nx*sizeof(float));
    memset(g1[0], 0, nz*nx*sizeof(float));
    memset(cg[0], 0, nz*nx*sizeof(float));
    memset(lap[0], 0, nz*nx*sizeof(float));
    memset(vtmp[0], 0, nz*nx*sizeof(float));
    memset(illum[0], 0, nz*nx*sizeof(float));
    /* set up zero for each array */

    for(it=0;it<nt;it++){
        tmp=SF_PI*fm*(it*dt-1.0/fm);
        tmp*=tmp;
        wlt[it]=(1.0-2.0*tmp)*expf(-tmp);
    }
    /* calculate source wavelet */

    if (!(sxbeg>=0 && szbeg>=0 &&
          sxbeg+(ns-1)*jsx<nx && szbeg+(ns-1)*jsz<nz)) {
        sf_warning("sources exceeds the computing zone!\n");
        exit(1);
    }
    /* check source position */
    sg_init(sxz, szbeg, sxbeg, jsz, jsx, ns, nz); !  GOTO sg_init( )
    /* shot position initialize */

    distx=sxbeg-gxbeg;
    distz=szbeg-gzbeg;
    if (csdgather){
        if(!(gxbeg>=0 && gzbeg>=0 &&
              gxbeg+(ng-1)*jgx<nx && gzbeg+(ng-1)*jgz<nz &&
              (sxbeg+(ns-1)*jsx)+(ng-1)*jgx-distx <nx  &&
              (szbeg+(ns-1)*jsz)+(ng-1)*jgz-distz <nz)){
                  sf_warning("geophones exceeds the computing zone!\n");
                  exit(1);
        }
    } else{
        if(!(gxbeg>=0 && gzbeg>=0 &&
             gxbeg+(ng-1)*jgx<nx && gzbeg+(ng-1)*jgz<nz)){
                sf_warning("geophones exceeds the computing zone!\n");
                exit(1);
        }
    }
    /* check receivers position */
    sg_init(gxz, gzbeg, gxbeg, jgz, jgx, ng, nz); !  GOTO sg_init( )
    /*
     * receiver position initialize
     * this code is available when csdgather==false
     */

    memset(bndr, 0, nt*(2*nz+nx)*sizeof(float));
    memset(dobs, 0, ng*nt*sizeof(float));
    memset(dcal, 0, ng*sizeof(float));
    memset(derr, 0, ns*ng*nt*sizeof(float));
    memset(alpha1, 0, ng*sizeof(float));
    memset(alpha2, 0, ng*sizeof(float));
    memset(dobs, 0, ng*nt*sizeof(float));
    memset(objval, 0, niter*sizeof(float));
    /* set up zero for each array */

    for(iter=0; iter<niter; iter++){
        if(verb){
            start=clock();/* record starting time */
            sf_warning("iter=%d",iter);
        }
        sf_seek(shots, 0L, SEEK_SET);
        memcpy(g0[0], g1[0], nz*nx*sizeof(float));
        memset(g1[0], 0, nz*nx*sizeof(float));
        memset(illum[0], 0, nz*nx*sizeof(float));
        for(is=0;is<ns;is++){
            sf_floatread(trans, ng*nt, shots);
            /* read shot gather */
            matrix_transpose(trans, dobs, nt, ng); !  GOTO matrix_transpose( )
            /* transpose the matrix to get dobs */
            if(csdgather){
                gxbeg=sxbeg+is*jsx-distx;
                sg_init(gxz, gzbeg, gxbeg, jgz, jgx, ng, nz); !  GOTO sg_init( )
            }
            /* receiver position initialize */

            memset(sp0[0], 0, nz*nx*sizeof(float));
            memset(sp1[0], 0, nz*nx*sizeof(float));
            for(it=0; it<nt; it++){
                add_source(sp1, &wlt[it], &sxz[is], 1, nz, true);
                !  GOTO add_source( )
                /* add source */

                step_forward(sp0, sp1, sp2, vv, dtz, dtx, nz, nx);
                !  GOTO step_forward( )
                /* forward exploration */

                ptr=sp0; sp0=sp1; sp1=sp2; sp2=ptr;
                /* update wavefield */

                rw_bndr(&bndr[it*(2*nz+nx)], sp0, nz, nx, true);
                !  GOTO rw_bndr( )
                /* save boundary value for saving memory */

                record_seis(dcal, gxz, sp0, ng, nz);
                !  GOTO record_seis( )
                /* save seismic record at recevier position */

                cal_residuals(dcal,&dobs[it*ng],&derr[is*ng*nt+it*ng],ng);
                !  GOTO cal_residuals( )
                /* calculate record residual at recevier position */
            }
            /* forward exploration complete */

            ptr=sp0; sp0=sp1; sp1=ptr;
            memset(gp0[0], 0, nz*nx*sizeof(float));
            memset(gp1[0], 0, nz*nx*sizeof(float));
            for(it=nt-1; it>-1; it--){
                rw_bndr(&bndr[it*(2*nz+nx)], sp1, nz, nx, false);
                !  GOTO rw_bndr( )
                /* read boundary value for saving memory */

                step_backward(illum,lap,sp0,sp1,sp2,vv,dtz,dtx,nz,nx);
                !  GOTO step_backward( )
                /*
```

```
             * this step is to recovery forward wavefield
             * via backward exploration and boundary condition
             * illum is the source compensate
             * lap is the laplace operator * velocity^2
             */

            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;
        !  GOTO cal_beta( )
        /* calculate beta */
        cal_conjgrad(g1, cg, beta, nz, nx);
        !  GOTO cal_conjgrad( )
        /* calculate cg direction */
        epsil=cal_epsilon(vv, cg, nz, nx);
        !  GOTO cal_epsilon( )
        /* calculate cg step size */
        /* calculating CG direction complete */

        sf_seek(shots, 0L, SEEK_SET);
        memset(alpha1, 0, ng*sizeof(float));
        memset(alpha2, 0, ng*sizeof(float));
        cal_vtmp(vtmp, vv, cg, epsil, nz, nx);
        !  GOTO cal_vtmp( )
        /* update the velocity */
        for(is=0;is<ns;is++){
            sf_floatread(trans, ng*nt, shots);
            /* read shot gather */
            matrix_transpose(trans, dobs, nt, ng);
            !  GOTO matrix_transpose( )
            /* transpose the matrix to get dobs */
            if(csdgather){
                gxbeg=sxbeg+is*jsx-distx;
                sg_init(gxz, gzbeg, gxbeg, jgz, jgx, ng, nz);
                !  GOTO sg_init( )
            }
            /* receiver position initialize */
            memset(sp0[0], 0, nz*nx*sizeof(float));
            memset(sp1[0], 0, nz*nx*sizeof(float));
            for(it=0; it<nt; it++){
                add_source(sp1, &wlt[it], &sxz[is], 1, nz, true);
                !  GOTO add_source( )
                /* add source */

                step_forward(sp0, sp1, sp2, vv, dtz, dtx, nz, nx);
                !  GOTO step_forward( )
                /* forward exploration */

                ptr=sp0; sp0=sp1; sp1=sp2; sp2=ptr;
                /* update wavefield */

                record_seis(dcal, gxz, sp0, ng, nz);
                !  GOTO record_seis( )
                /* save seismic record at recevier position */

                sum_alpha12(alpha1, alpha2, dcal, &dobs[it*ng], &derr[is*ng*nt+it*ng], ng);
                !  GOTO sum_alpha12( )
                /* calculate alpha12 */
            }
        }

        alpha=cal_alpha(alpha1, alpha2, epsil, ng);
        !  GOTO cal_alpha( )
        /* calculate alpha */

        update_vel(vv, cg, alpha, nz, nx);
        !  GOTO update_vel( )
        /* update velocity */
        sf_floatwrite(vv[0], nz*nx, vupdates);
        /* output velcotiy */
        /* updating velocity complete */

        if(iter==0) {
            obj1=obj;
            objval[iter]=1.0;
        } else{
            objval[iter]=obj/obj1;
        }
        /* calcuate obj */

        if(verb) {
            sf_warning("obj=%f  beta=%f  epsil=%f  alpha=%f", obj, beta, epsil, alpha);
            /* output important information at each FWI iteration */
            stop=clock();
            /* record ending time */
            sf_warning("iteration %d finished: %f (s)",iter+1, ((float)(stop-start))/CLOCKS_PER_SEC);
        }
    }
    sf_floatwrite(objval, niter, objs);
    /* output obj */

    free(*vv); free(vv);
    free(*vtmp); free(vtmp);
    free(*sp0); free(sp0);
    free(*sp1); free(sp1);
    free(*sp2); free(sp2);
    free(*gp0); free(gp0);
    free(*gp1); free(gp1);
    free(*gp2); free(gp2);
    free(*g0); free(g0);
    free(*g1); free(g1);
    free(*cg); free(cg);
    free(*lap); free(lap);
    free(*illum); free(illum);
    free(objval);
    free(wlt);
    free(sxz);
    free(gxz);
    free(bndr);
    free(trans);
    free(dobs);
    free(dcal);
    free(derr);
    free(alpha1);
    free(alpha2);

    exit(0);
}
```

sg_init( )

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

Listing 2: sg_init( )

```
    void sg_init(int *sxz, int szbeg, int sxbeg,
                 int jsz, int jsx, int ns, int nz)
    /*< shot/geophone position initialize >*/
    {
        int is, sz, sx;
        for(is=0; is<ns; is++) {
            sz=szbeg+is*jsz;
            sx=sxbeg+is*jsx;
            sxz[is]=sz+nz*sx;
        }
    }
    //!  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( )

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

record_seis( )

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

Listing 7: record_seis( )

```
void record_seis(float *seis_it, int *gxz, float **p, int ng, int nz)
/*< record seismogram at time it into a vector length of ng >*/
{
    int ig, gx, gz;
    for(ig=0;ig<ng; ig++){
        gx=gxz[ig]/nz;
        gz=gxz[ig]%nz;
        seis_it[ig]=p[gx][gz];
    }
}
//!  RETURN main( )
```

cal_residuals( )

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

Listing 8: cal_residuals( )

```
void cal_residuals(float *dcal, float *dobs, float *dres, int ng)
/*< calculate residual >*/
{
    int ig;
    for(ig=0; ig<ng; ig++){
        dres[ig]=dcal[ig]-dobs[ig];
    }
}
```

```
//!  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 >*/
{
```

```c
    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( )

```c
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( )

```c
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( )
```