```
#include "tdmrflib.h"

/* ###########################
  Modified in 2004/07/22
  **outsyn **outdata -> outsyn[3][WSIZE] outdata[3][WSIZE]
  Modified in 2004/07/23 Bug fixed
    L809-813 Moment unit Nm
    Because observed wave unit is cm,
    m1,m2,m3,m4,m5 are devied by 10
  Modified in 2004/07/26
    L921-L931
    Add a part of Caluculation for V.R at each station.
  Modified in 2004/08/03
    STNM 50 -> 100
    Modified in 2004/08/04
    STNM 100->200
############################## */

#define     STNM 100
#define     DMY 256
#define     WSIZE 7200

extern void exit_prg() ;
extern void splitc() ;
extern int check_fileline() ;
extern void dcp_() ;
extern void dcp2_() ;

struct INARGV inargv ;

struct WAVEDATA {
  int wavelength ;
  int cmpn[STNM] ;
  double data[STNM][3][WSIZE] ;
}waved ;

struct GREENFUN {
  int wavelength ;
  double gfnc[STNM][8][WSIZE] ;
}grenf ;

struct MATRIXA {
  int a,b ;
  double **mtxA ; /* mtxA[a][b]  a>=b */
  double **mtxAT ; /* mtxAT[b][a] */
  double **mtxATA ; /* mtxATA[b][b] */
  double **mtxATAI ; /* mtxATAI[b][b] */
  double **mtxB ; /* mtxB[b][a]=ATAIAT */
  double *mtxW ; /* mtxW[a]=Weight[a][a] */
}strc ;


void usage(){
  static char *mes[] = {
    " PROGRAM NAME   VAR   by Y.Ito", /*1*/
    0
  } ;
  static int no_ln = 1 ;
  int i ;
  for( i=0 ; i< no_ln ; i++)
    fprintf( stderr, "%s\n", mes[i] ) ;
}
```

```
int get_waveform_data( int stnum, char *filename, struct WAVEDATA *strc ){
  FILE *fp ;
  int i, j,cpnum, dpnum  ;
  char dmy[DMY] ;

  if((fp=fopen( filename, "r" ))==NULL){
    printf( "FILE NOT FOUND %s\n", filename )  ;
    return(-1) ;
  }

  fgets( dmy, sizeof(dmy), fp) ; // line 1
  sscanf( dmy, "%d", &strc->cmpn[stnum] ) ;
  cpnum=strc->cmpn[stnum] ;
  fgets( dmy, sizeof(dmy), fp) ; // line 2
  for(i=0; i< cpnum ; i++){
    fgets( dmy, sizeof(dmy), fp) ; // componet line 1
    fgets( dmy, sizeof(dmy), fp) ; // componet line 2
    sscanf( dmy, "%d", &dpnum ) ;
    //fprintf( stderr, "%d\n",dpnum ) ;
    for(j=0 ; j<dpnum ; j++){
      fscanf( fp, "%lf", &strc->data[stnum][i][j] ) ;
    }
    fgets(dmy, sizeof(dmy),fp) ;
  }
  fclose(fp) ;
  strc->wavelength=dpnum ;
  return(0) ;
}

int get_greenfunction( int stnum, char *filename, struct GREENFUN *strc ){
  FILE *fp ;
  int i, j,cpnum, dpnum,fflag ;
  char dmy[DMY],dmy2[DMY] ;

  if((fp=fopen( filename, "r" ))==NULL){
    printf( "FILE NOT FOUND %s\n", filename )  ;
    return(-1) ;
  }

  fgets( dmy, sizeof(dmy), fp) ; // line 1
  sscanf( dmy, "%d", &cpnum ) ;
  //printf( "COMPNUM %d\n",cpnum) ;
  fgets( dmy2, sizeof(dmy2), fp) ; // line 2
  //printf( "%s",dmy2) ;
  if( strncmp( dmy2, "(6e12.5)",8)==0){
    //printf( "FORMAT TYPE 6e12.5 READ\n") ;
    for(i=0; i< cpnum ; i++){
      fgets( dmy, sizeof(dmy), fp) ; // componet line 1
      fgets( dmy, sizeof(dmy), fp) ; // componet line 2
      sscanf( dmy, "%d", &dpnum ) ;
      //printf( "DATA NUM %d\n",dpnum) ;
      //printf( "%d\n", dpnum ) ;
      //fprintf( stderr, "%d\n",dpnum ) ;
      for(j=0 ; j<dpnum ; j++){
        fscanf( fp, "%lf", &strc->gfnc[stnum][i][j] ) ;
        //if( j==0)
        //printf( "%1.5e\n",strc->gfnc[stnum][i][j]) ;
      }
      fgets(dmy, sizeof(dmy),fp) ;
    }
    //printf("Loop end\n");
    fclose(fp) ;
    //printf("file close\n");
    strc->wavelength=dpnum ;
  }
  else{
    //printf( "FORMAT TYPE ELSE\n") ;
    for(i=0; i< cpnum ; i++){
      fgets( dmy, sizeof(dmy), fp) ; // componet line 1
      fgets( dmy, sizeof(dmy), fp) ; // componet line 2
      sscanf( dmy, "%d", &dpnum ) ;
      //fprintf( stderr, "%d\n",dpnum ) ;
      for(j=0 ; j<dpnum ; j++){
        fscanf( fp, "%lf", &strc->gfnc[stnum][i][j] ) ;
      }
      fgets(dmy, sizeof(dmy),fp) ;
    }
    fclose(fp) ;
    strc->wavelength=dpnum ;
  }
  //printf("bfore return\n");
  return(0) ;
}

int gaussjordan_strc( strc )
     struct MATRIXA *strc ;
{
  int i,j,k,l,n,irow,icol,ll ;
  /* int indxc[5], indxr[5],ipiv[5] ; */
  int *indxc, *indxr, *ipiv ;
  double s,big,pivinv,dum,temp,sum ;
  /* double nv[5][5] ; */
  double **nv ;
  irow=icol=0 ;
  indxc=calloc( strc->b, sizeof( int )) ;
  indxr=calloc( strc->b, sizeof( int )) ;
  ipiv =calloc( strc->b, sizeof( int )) ;
  nv=calloc( strc->b,sizeof( double*)) ;
  for(i=0 ; i<strc->b ; i++)
    nv[i]=calloc(strc->b, sizeof(double)) ;
#if DEBUG==1
  fprintf(stderr,"matrix A\n");
  for(i=0 ; i<strc->a ; i++)
    for( j=0 ; j<strc->b ; j++){
      if( j==0 )fprintf( stderr, "[" ) ;
      fprintf( stderr," %4.3e",strc->mtxA[i][j] ) ;
      if( j==strc->b-1)fprintf( stderr," ]\n" ) ;
    }
#endif
  // make AT matrix
  for( i=0 ; i< strc->a ; i++){
    for( j=0; j<strc->b ; j++){
      strc->mtxAT[j][i]=strc->mtxA[i][j] ;
    }
  }

#if DEBUG==1
  fprintf(stderr,"matrix AT\n");
  for(i=0 ; i<strc->b ; i++)
    for( j=0 ; j<strc->a ; j++){
      if( j==0 )fprintf( stderr, "[" ) ;
      fprintf( stderr," %4.3e",strc->mtxAT[i][j] ) ;
      if( j==strc->a-1)fprintf( stderr," ]\n" ) ;
    }
#endif
```

```
  // make ATA matrix
  for(i=0 ; i<strc->b ; i++){
    for( j=0 ; j<strc->b ; j++){
      for( k=0,s=0.0 ; k<strc->a ; k++)
        s+=strc->mtxAT[i][k]*strc->mtxW[k]*strc->mtxA[k][j];
      strc->mtxATA[i][j]=s ;
    }
  }

#if DEBUG==1
  fprintf(stderr,"matrix ATA\n");
  for(i=0 ; i<strc->b ; i++)
    for( j=0 ; j<strc->b ; j++){
      if( j==0 )fprintf( stderr, "[" ) ;
      fprintf( stderr," %4.3e",strc->mtxATA[i][j] ) ;
      if( j==strc->b-1)fprintf( stderr," ]\n" ) ;
    }
#endif
  for(i=0; i<strc->b ; i++)
    for(j=0 ; j<strc->b ;j++)
      strc->mtxATAI[i][j]=strc->mtxATA[i][j] ;


  n=strc->b ;
  for(j=0; j<strc->b ; j++){
    ipiv[j]=0 ;
    indxc[j]=0 ;
    indxr[j]=0 ;
  }
  for(i=0 ; i<strc->b ; i++){
    big=0.0 ;
    for(j=0 ; j<strc->b ; j++)
      if(ipiv[j]!=1)
        for(k=0 ; k<strc->b ; k++){
          if(ipiv[k]==0){
            if( fabs( strc->mtxATAI[j][k]) >=big ){
              big=fabs( strc->mtxATAI[j][k]) ;
              irow=j ;
              icol=k ;
            }
          }
          else if( ipiv[k]>1){
            fprintf( stderr, "gaussj: Singular Matrix-1\n" ) ;
            exit_prg( 2, "calculate inverse matrix" ) ;
          }
        }
    ++(ipiv[icol]) ;
    if( irow != icol){
      for(l=0 ; l<strc->b ; l++)
        SWAP(strc->mtxATAI[irow][l], strc->mtxATAI[icol][l]);
    }
    indxr[i]=irow ;
    indxc[i]=icol ;
    if( strc->mtxATAI[icol][icol]==0.0){
      fprintf( stderr, "gaussj: Singular Matrix-2\n" ) ;
      exit_prg( 3, "calculate inverse matrix" ) ;
    }
    pivinv=1.0/strc->mtxATAI[icol][icol] ;
#if DEBUG==1
    fprintf( stderr, "pivinv= %e\n", pivinv ) ;
#endif
    strc->mtxATAI[icol][icol]=1.0 ;
    for(l=0 ; l<strc->b ; l++)
      strc->mtxATAI[icol][l]*= pivinv ;
    for(ll=0 ;ll<strc->b;ll++)
      if( ll != icol){
        dum=strc->mtxATAI[ll][icol] ;
        strc->mtxATAI[ll][icol]=0.0 ;
        for( l=0 ; l<strc->b ; l++)
          strc->mtxATAI[ll][l] -= strc->mtxATAI[icol][l]*dum ;
      }
  }

  for(l=strc->b-1 ; l>=0 ; l--){
    if( indxr[l] != indxc[l] )
      for(k=0 ; k<strc->b ; k++)
        SWAP( strc->mtxATAI[k][indxr[l]], strc->mtxATAI[k][indxc[l]] ) ;
  }
  /* make ATAI*AT */
  for( i=0 ; i<strc->b ; i++ )
    for( j=0 ; j< strc->a ; j++){
      for(k=0, sum=0.0 ; k< strc->b ; k++)
        sum+=strc->mtxATAI[i][k]*strc->mtxAT[k][j] ;
      strc->mtxB[i][j]=sum ;
    }

#if DEBUG==1
  fprintf( stderr, "Matrix ATAI*AT\n" ) ;
  for(i=0 ; i<strc->b ; i++)
    for( j=0 ; j<strc->a ; j++){
      if( j==0 )fprintf( stderr, "[" ) ;
      fprintf( stderr," %4.3e",strc->mtxB[i][j] ) ;
      if( j==strc->a-1)fprintf( stderr," ]\n" ) ;
    }
  fprintf( stderr,"Matrix ATAI(invert ATA)\n" ) ;
  for(i=0 ; i<strc->b ; i++)
    for( j=0 ; j<strc->b ; j++){
      if( j==0 )fprintf( stderr, "[" ) ;
      fprintf( stderr," %4.3e",strc->mtxATAI[i][j] ) ;
      if( j==strc->b-1)fprintf( stderr," ]\n" ) ;
    }

  for(i=0 ; i< strc->b  ; i++)
    for(j=0 ; j< strc->b ; j++){
      for(k=0, sum=0.0 ; k < strc->b ; k++ )
        sum+=strc->mtxATAI[i][k]*strc->mtxATA[k][j] ;
      nv[i][j]=sum ;
    }
  fprintf(stderr,"matrix N=ATAI*ATA\n");
  for(i=0 ; i<strc->b ; i++)
    for( j=0 ; j<strc->b ; j++){
      if( j==0 )fprintf( stderr, "[" ) ;
      fprintf( stderr," %4.3lf",nv[i][j] ) ;
      if( j==strc->b-1)fprintf( stderr," ]\n" ) ;
    }
#endif
  free(indxc) ;
  free(indxr) ;
  free(ipiv) ;
  free( nv ) ;
  return(0) ;

}

int mtrx_memory_alloc2( a, b, strc )
```

```
     int a, b ;
     struct MATRIXA *strc ;
{
  int i;
  strc->a = a ;
  strc->b = b ;

  strc->mtxA   = calloc( a, sizeof( double*)) ;
  strc->mtxW   = calloc( a, sizeof( double )) ;
  for(i=0 ; i<a ; i++){
    strc->mtxA[i] = calloc( b, sizeof( double)) ;
    strc->mtxW[i] = 1.0 ;
  }
  strc->mtxATA = calloc( b, sizeof( double*)) ;
  strc->mtxATAI= calloc( b, sizeof( double*)) ;
  strc->mtxAT  = calloc( b, sizeof( double*)) ;
  strc->mtxB  = calloc( b, sizeof( double*)) ;
  for(i=0 ; i<b ; i++){
    strc->mtxAT[i]  = calloc( a, sizeof( double)) ;
    strc->mtxATA[i] = calloc( b, sizeof( double)) ;
    strc->mtxATAI[i]= calloc( b, sizeof( double)) ;
    strc->mtxB[i]   = calloc( a, sizeof(double)) ;
  }
  return(0) ;
}

int mtrx_memory_free2( strc )
     struct MATRIXA *strc ;
{

  int i,a,b;

  a=strc->a ;
  b=strc->b ;
  free( strc->mtxW ) ;
  for(i=0 ; i<a ; i++)
    free( strc->mtxA[i]);

  free( strc->mtxA );

  for(i=0 ; i<b ; i++){
    free(strc->mtxAT[i]) ;
    free(strc->mtxATA[i]) ;
    free(strc->mtxATAI[i]) ;
    free(strc->mtxB[i]) ;
  }

  free( strc->mtxATA ) ;
  free( strc->mtxATAI ) ;
  free( strc->mtxAT );
  free( strc->mtxB );

  return(0) ;
}

int main( int argc, char *argv[] ){

  /***********************/
  FILE *fp,*fp2,*fpout ;
  int i,j,k,ret,n ;
  char dmy[200], infname[200], *buffer ;
  /****** USER *********/
  int stnum ,allnum, nm, cnt1,cnt2,cnt3,np,z,tsp,smoothline ;
  int maxdnum, dz, itrnum, digsys,nn ;
  char obsfname[STNM][DMY], gfcfname[DMY][STNM][DMY] ;
  char evlab[7][DMY], deps[DMY][DMY] ;
  double delta[STNM], azimuth[STNM] ;
  double **smooth ;
  double *obs, *cals, *mtn, outdata[3][WSIZE], outsyn[3][WSIZE] ;
  double mindist, cormax,dx,var,vr,pw,smoval ;
  double maxamp,maxvr ;
  int zcor[STNM], obsleng[STNM] ;
  int gzcor[STNM], gfcleng[STNM] ;
  int digit[STNM], **zmat, bzcor[STNM] ;
  int deplabn, dtime, maxtime ;
  int gfnum, depitr,tst ;
  /**********************/
  float m1, m2, m3, m4, m5,m6,mm[3],mmmax,mmmin ;
  float sm1,sm2,sm3,sm4,sm5,sm6 ;
  float mrr,mtt,mff,mrt,mrf,mtf ;
  float fstr[2], fdip[2], frak[2], fmo[2], fdmo[2] ;
  double mw ;
  /****************/

  static char *param[]={
    "-h", /* 0:help */
    NULL, /* 1 */
  } ;
  static int no_param=1 ;
  /* DEFAULT */

  /* SET PARAMETER */
  for( i=1 ; i<argc ; i++ ){
    if( argv[i][0]!='-'){
      strcpy( infname, argv[i] ) ;
    }
    else{
      buffer=calloc( 100, sizeof(char)) ;
      for( j=0 ; j<no_param ; j++ ){
        if( strncmp( param[j],argv[i],2 )==0){
          k=0 ;
          while(argv[i][k+2]!='\0'){
            buffer[k]=argv[i][k+2] ;
            k++ ;
          }
          buffer[k]='\0' ;
          break ;
        }
      }

      switch( j ){
      case 0:
        usage() ; exit_prg( 0, "main()" ) ; exit(1) ;
        /*********** SWITCH OPTION *************/

        /*****************************************/
      default:
        usage() ;
        exit_prg( 2, "main()[main .c]" ) ;
      }
      free(buffer) ;
    }
  }
  /* check input argument */
  /****** MAIN PART *********/
  /*** get inversion param ****/
  if( (fp=fopen( infname, "r" ))==NULL){
    printf( "%s NOt FOUND\n", infname ) ;
    exit_prg(-1, "main()" ) ;
  }

  fgets(dmy,sizeof(dmy),fp) ;
  //sscanf(dmy, "%d %lf\n",&stnum, &sdep ) ;
  sscanf(dmy, "%d %d %d %d %s %s %s %s %s %s %s",&stnum, &deplabn, &dtime, &maxtime,
         evlab[0],evlab[1],evlab[2],evlab[3],evlab[4],evlab[5],evlab[6]) ;
  //printf( "%d %d %d %d %s %s %s %s %s %s %s\n",stnum,deplabn, dtime, maxtime,
  //      evlab[0],evlab[1],evlab[2],evlab[3],evlab[4],evlab[5],evlab[6] ) ;
  ///////////////////////////////////////////
  for(i=0, mindist=100000.0  ; i<stnum ; i++){
    fgets(dmy,sizeof(dmy), fp ) ;
    sscanf( dmy, "%s %lf %lf %d %d", obsfname[i], &delta[i], &azimuth[i], &zcor[i], &ob
sleng[i]) ;
    if( delta[i] < mindist )
      mindist=delta[i] ;
    azimuth[i]/=PD ;
  }

  for(j=0 ; j<deplabn ; j++){
    for(i=0 ; i<stnum ; i++){
      fgets(dmy, sizeof(dmy), fp) ;
      sscanf( dmy, "%s %d %d %s\n", gfcfname[j][i], &gzcor[i], &gfcleng[i],deps[j]) ;
    }
  }

  //sscanf( dmy, "%d %d %lf", &maxdnum, &tsp, &smoval ) ;
  //sscanf( dmy, "%d", &dz) ; // allowable time shift width
  //fclose(fp) ;
  // For multiple source
  maxdnum=1 ;
  tsp=10  ;
  smoval=5e-13 ;
  dz=0 ;
  //////////////////////////
  /*** get waveform data ***/
  for(i=0; i<stnum ;  i++){
    //printf( "FILENAME %s\n", obsfname[i] ) ;
    ret=get_waveform_data( i, obsfname[i], &waved ) ;
    if( ret!=0 )
      exit_prg( -1, "GET WAVEFORM DATA" ) ;
  }

  //for(i=0; i<waved.wavelength ;i++  ){
  //  printf( "%d %1.5e %1.5e %1.5e\n",i,
  //   waved.data[2][0][i], waved.data[2][1][i],waved.data[2][2][i] ) ;
  //
  //}
  /*** Memory Allocate ****/
  for(i=0, allnum=0 ; i<stnum; i++){
    //printf( "St. %d  %d x %d\n", i,obsleng[i], waved.cmpn[i]) ;
    allnum+=obsleng[i]*waved.cmpn[i] ;

  }
//printf( "ALLNUM %d\n",allnum ) ;

  if( maxdnum > 2 ){
    nm=5*maxdnum ;
    smoothline=5*maxdnum ;
  }
  else{
    nm=5 ;
    smoothline=0 ;
    maxdnum = 1 ;
  }
  mtrx_memory_alloc2( allnum+smoothline, nm, &strc) ;

  obs=calloc( allnum+smoothline, sizeof(  double)) ;
  cals=calloc( allnum+smoothline, sizeof( double)) ;
  mtn=calloc( nm, sizeof( double )) ;
//printf( "Memalloc end\n" ) ;
  //////////////////////////////


  //  fpout=fopen( "grid_tdmrf_inv.out","w") ;
  fpout=fopen( "grid_tdmrf_inv.out","a") ;
  /*** get Green Function ***/
  gfnum=0 ;
  for(gfnum=0 ; gfnum < deplabn ; gfnum++){
    for(i=0; i<stnum ;  i++){
      //printf( "%s\n", gfcfname[gfnum][i]) ;
      ret=get_greenfunction( i, gfcfname[gfnum][i], &grenf ) ;
      if( ret!=0 )
        exit_prg( -1, "GET GREEN FUNCTION" ) ;
      for( j=0; j<grenf.wavelength ; j++){
        // After Dreger’s code
        grenf.gfnc[i][5][j]*=-1.0 ; // DREGER: Note the vertical GF’s are
        grenf.gfnc[i][6][j]*=-1.0 ; // DREGER: flipped in earqt1.f and TW’s
        grenf.gfnc[i][7][j]*=-1.0 ; // DREGER: Blackbox.f DVH conv. z + down
      }

    }

    //for(i=0; i<grenf.wavelength ;i++  ){
    //  printf( "%d %1.5e %1.5e %1.5e %1.5e %1.5e %1.5e %1.5e %1.5e\n",
    //    i,grenf.gfnc[1][0][i], grenf.gfnc[1][1][i],grenf.gfnc[1][2][i], grenf.gfnc[1]
[3][i],
    //    grenf.gfnc[1][4][i], grenf.gfnc[1][5][i],grenf.gfnc[1][6][i], grenf.gfnc[1][7
][i]) ;
    // }

    /*** Memory Allocate ****/
    //for(i=0, allnum=0 ; i<stnum; i++){
    //  printf( "St. %d  %d x %d\n", i,obsleng[i], waved.cmpn[i]) ;
    //  allnum+=obsleng[i]*waved.cmpn[i] ;
    //
    /// }
    //printf( "ALLNUM %d\n",allnum ) ;
    //
    //if( maxdnum > 2 ){
    //  nm=5*maxdnum ;
    //  smoothline=5*maxdnum ;
    //}
    //else{
    //  nm=5 ;
    //  smoothline=0 ;
    //  maxdnum = 1 ;
    //}
    //mtrx_memory_alloc2( allnum+smoothline, nm, &strc) ;

    //obs=calloc( allnum+smoothline, sizeof(  double)) ;
    //cals=calloc( allnum+smoothline, sizeof( double)) ;
    //mtn=calloc( nm, sizeof( double )) ;
    //printf( "Memalloc end\n" ) ;
    /*** MAKE MATRIX ****/

    for( n=0 ; n< maxdnum ; n++ ){
      cnt1=cnt2=cnt3=0 ;
      for( i=0 ; i<stnum ; i++ ){
        np=obsleng[i] ;
        z=gzcor[i] ;
        cnt1=cnt2=cnt3 ;
        if( waved.cmpn[i] == 3){
          cnt2+=np ;
          cnt3+=2*np ;
        }
        else{
          cnt2+=np ;
          cnt3+=np ;
        }
        cormax = delta[i]/mindist ;
        //distw[i]=cormax ;


        for(j=z,k=0 ; j<z+np; j++,k++){
          if( k< tsp*n ){
            strc.mtxA[cnt1][0+5*n]=0 ;
            strc.mtxA[cnt1][1+5*n]=0 ;
            strc.mtxA[cnt1][2+5*n]=0 ;
            strc.mtxA[cnt1][3+5*n]=0 ;
            strc.mtxA[cnt1][4+5*n]=0 ;

            strc.mtxA[cnt2][0+5*n]=0 ;
            strc.mtxA[cnt2][1+5*n]=0 ;
            strc.mtxA[cnt2][2+5*n]=0 ;
            strc.mtxA[cnt2][3+5*n]=0 ;
            strc.mtxA[cnt2][4+5*n]=0 ;

          if( waved.cmpn[i] == 3 ){
            strc.mtxA[cnt3][0+5*n]=0 ;
            strc.mtxA[cnt3][1+5*n]=0 ;
            strc.mtxA[cnt3][2+5*n]=0 ;
            strc.mtxA[cnt3][3+5*n]=0 ;
            strc.mtxA[cnt3][4+5*n]=0 ;
          }
          }
          else{
            // Transverse
            strc.mtxA[cnt1][0+5*n] = 0.5*sin(2.0*azimuth[i])* grenf.gfnc[i][0][j-tsp*n]
 ;    // T Mxx
            strc.mtxA[cnt1][1+5*n] = (-0.5)*sin(2.0*azimuth[i])* grenf.gfnc[i][0][j-tsp
*n];  // T Myy
            strc.mtxA[cnt1][2+5*n] = (-1.0)*cos(2.0*azimuth[i])* grenf.gfnc[i][0][j-tsp
*n];  // T Mxy
            strc.mtxA[cnt1][3+5*n] = (-1.0)*sin(azimuth[i])*grenf.gfnc[i][1][j-tsp*n] ;
      // T Mxz
            strc.mtxA[cnt1][4+5*n] =        cos(azimuth[i])*grenf.gfnc[i][1][j-tsp*n] ;
      // T Myz

            // Radial
            strc.mtxA[cnt2][0+5*n] = 0.5*(grenf.gfnc[i][4][j-tsp*n]
                                          - cos(2.0*azimuth[i])*grenf.gfnc[i][2][j-tsp*
n]) ; // R Mxx
            strc.mtxA[cnt2][1+5*n] = 0.5*(grenf.gfnc[i][4][j-tsp*n]
                                          + cos(2.0*azimuth[i])*grenf.gfnc[i][2][j-tsp*
n]) ; // R Myy
            strc.mtxA[cnt2][2+5*n] = (-1.0)*sin(2.0*azimuth[i])* grenf.gfnc[i][2][j-tsp
*n];  // R Mxy
            strc.mtxA[cnt2][3+5*n] =        cos(azimuth[i])*grenf.gfnc[i][3][j-tsp*n] ;
      // R Mxz
            strc.mtxA[cnt2][4+5*n] =        sin(azimuth[i])*grenf.gfnc[i][3][j-tsp*n] ;
      // R Myz

            if( waved.cmpn[i] == 3 ){
              // Zcomp
              strc.mtxA[cnt3][0+5*n] = 0.5*(grenf.gfnc[i][7][j-tsp*n]
                                            - cos(2.0*azimuth[i])*grenf.gfnc[i][5][j-ts
p*n]) ; // Z Mxx
              strc.mtxA[cnt3][1+5*n] = 0.5*(grenf.gfnc[i][7][j-tsp*n]
                                            + cos(2.0*azimuth[i])*grenf.gfnc[i][5][j-ts
p*n]) ; // Z Myy
              strc.mtxA[cnt3][2+5*n] = (-1.0)*sin(2.0*azimuth[i])* grenf.gfnc[i][5][j-t
sp*n];  // Z Mxy
              strc.mtxA[cnt3][3+5*n] =        cos(azimuth[i])*grenf.gfnc[i][6][j-tsp*n]
 ;      // Z Mxz
              strc.mtxA[cnt3][4+5*n] =        sin(azimuth[i])*grenf.gfnc[i][6][j-tsp*n]
 ;      // Z Myz
            }
          }
          //weight for distance
          strc.mtxW[cnt1]=cormax ;
          strc.mtxW[cnt2]=cormax ;
          if( waved.cmpn[i] == 3 )
            strc.mtxW[cnt3]=cormax ;

          //if( cormax != 1.0 )
          //printf( "Cormax > 1 \n" ) ;

          cnt1++;
          cnt2++;
          cnt3++;

        }
      }
    }

    //printf( "MAKE MATRIX Asize %d x %d\n",nm, cnt3 ) ;

    /* */

    /* make smooth matrix **/
    //if( maxdnum > 2){
    //  smooth=calloc( smoothline, sizeof( double* )) ;
    //  for( i=0 ; i < smoothline ; i++){
    //    smooth[i]=calloc( nm, sizeof( double )) ;
    //    for(j=0 ; j< nm ;  j++ ){
    //  smooth[i][j]=0.0 ;
    //   }
    //  }
    //
    //  cnt1=0 ;
    //  for( j=0; j< 5 ;j++){
    //    for(k=0 ; k< maxdnum ; k++){
    //  if( k==0 ){
    //    smooth[cnt1][j]=-2 ;
    //    smooth[cnt1][5+j]=1 ;
    //  }
    //  else if( k==maxdnum - 1){
    //    smooth[cnt1][(k-1)*5+j]=1 ;
    //    smooth[cnt1][k*5+j]=-2 ;
    //  }
    //  else{
    //    smooth[cnt1][(k-1)*5+j]=1 ;
    //    smooth[cnt1][k*5+j]=-2 ;
    //    smooth[cnt1][(k+1)*5+j]=1 ;
    //  }
    //  ++cnt1 ;
    //   }
    //  }
    //
    //    for( i=0 ; i< smoothline ; i++){
    //   for(j=0 ; j < nm ; j++){
    //  strc.mtxA[i+allnum][j]=smooth[i][j] ;
    //   }
    //  }

    //  /* make smooth part */
    //  for( i= 0 ; i< smoothline ; i++){
    //   obs[i+allnum]=0.0 ;
    //    strc.mtxW[i+allnum]=smoval ;
    //  }
    // }


    //if( dz > 0 ){
    //  digsys=2*dz+1 ;
    //  printf( "Allowable +- dz %d\n", dz ) ;
    //  printf( "Allowable size %d\n", digsys ) ;
    //  itrnum=(int)pow((double)digsys, (double)stnum) ;
    //  printf( "Iteration %d\n", itrnum ) ;
    //  zmat=calloc( itrnum, sizeof(int*)) ;
    //
    //  digit[0]=-1 ;
    //  for(i=0 ; i<itrnum ; i++){
    //    zmat[i]=calloc( stnum, sizeof(int)) ;
    //    for(j=0 ; j<stnum ; j++){
    //zmat[i][j]=0 ;
    //   }
    //   digit[0]++ ;
    //   k=0 ;
    //   while(digit[k]==digsys){
    //  if( digit[k]== digsys ){
    //    digit[k]=0 ;
    //    digit[k+1]++ ;
    //  }
    //  k++ ;
    //   }
    //   for(j=0 ; j< stnum; j++){
    //  zmat[i][j]=digit[j]-dz ;
    //   }
    // }
    //}
    //else{
    itrnum=1  ;
    //}
    //for(i=0 ; i< itrnum ; i++){
    //for(j=0 ; j< stnum ; j++){
    //    printf( "%d ", zmat[i][j] ) ;
    //  }
    //  printf( "\n" ) ;
    //}
    // make observation data
    //if( itrnum > 1 ){
    //  for(nn=0,maxvr=0.0 ; nn < itrnum ; nn++){
    //    printf("%d (%d) ", nn,itrnum) ;
    //    cnt1=cnt2=cnt3=0 ;
    //    for(i=0; i < stnum ; i++){
    //
    //  z=zcor[i]+zmat[nn][i] ;
    //
    //  np=obsleng[i];
    //  //printf( "OBSLENG %d\n", obsleng[i] ) ;
    //  cnt1=cnt2 = cnt3;
    //  if( waved.cmpn[i]== 3){
    //    cnt2 += np;
    //    cnt3 += 2*np;
    //  }
    //  else{
    //    cnt2+=np ;
    //    cnt3+=np ;
    //  }
    //  printf("%d %d ",z,np ) ;
    //  for( j=z ; j < z+np ; j++){
    //    obs[cnt1]        = waved.data[i][0][j] ;
    //    obs[cnt2]        = waved.data[i][1][j] ;
    //    if(waved.cmpn[i]==3)
    //      obs[cnt3]        = waved.data[i][2][j] ;
    //
    //    cnt1++;
    //    cnt2++;
    //    cnt3++;
    //  }
    //   }
    //   /*** INVERSE MATRIX ***/
    //    gaussjordan_strc( &strc ) ;
    //    /*** RESULT OUT ***/
    //    for(i=0 ; i<nm ; i++){
    //  for(j=0, mtn[i]=0 ; j<allnum+smoothline ; j++)
    //    mtn[i]+=strc.mtxB[i][j]*obs[j]*strc.mtxW[j] ;
    //   }
    //   /*** Make synthetic wave ****/
    //   for(i=0,var=0.0,pw=0.0 ; i<allnum ; i++){
    //  for(j=0,cals[i]=0.0 ; j<nm ; j++){
    //    cals[i]+=strc.mtxA[i][j]*mtn[j] ;
    //  }
    //  dx=cals[i]-obs[i] ;
    //  dx*=dx ;
    //  var+=dx ;
    //  pw+=obs[i]*obs[i] ;
    //   }
    //   vr = (1-var/pw)*100.0 ;
    //   printf( " Var.Red %1.2f\n" ,vr) ;
    //   if( maxvr < vr ){
    //  maxvr=vr ;
    //
    //  for(j=0 ; j< stnum ; j++){
    //    bzcor[j]=zcor[j]+zmat[nn][j] ;
    //  }
    //
    //   }
    // }
    //
    // printf( "Max Var.Red = %1.2f\n", vr ) ;
    // for(i=0 ; i<stnum ; i++){
```

```
    //   printf( "station %d Zcor %d\n", i, bzcor[i]) ;
    // }
    //}
    //else{
    for(j=0 ; j< stnum ; j++){
      bzcor[j]=zcor[j] ;
    }
    //}
    /*** INVERSE MATRIX ***/
    gaussjordan_strc( &strc ) ;
    //////////////////////////


    // Re_Calculate Best Zcor
    for(tst=0 ; tst<=maxtime ;tst+=dtime){
      cnt1=cnt2=cnt3=0 ;
      for(i=0; i < stnum ; i++){
        z=bzcor[i]+tst;
        //printf( "%d\n", z ) ;
        np=obsleng[i];
        //printf( "OBSLENG %d\n", obsleng[i] ) ;
        cnt1=cnt2 = cnt3;
        if( waved.cmpn[i]==3 ){
          cnt2 += np;
          cnt3 += 2*np;
        }
        else{
          cnt2 +=np ;
          cnt3 +=np ;
        }
        //printf( "%d %d ",z,np ) ;
        for( j=z ; j < z+np ; j++){
          obs[cnt1]        = waved.data[i][0][j] ;
          obs[cnt2]        = waved.data[i][1][j] ;
          if( waved.cmpn[i]==3){
            obs[cnt3]        = waved.data[i][2][j] ;
          }
          cnt1++;
          cnt2++;
          cnt3++;
        }
      }
      /*** RESULT OUT ***/
      for(i=0 ; i<nm ; i++){
        for(j=0, mtn[i]=0 ; j<allnum+smoothline ; j++)
          mtn[i]+=strc.mtxB[i][j]*obs[j]*strc.mtxW[j] ;
      }
      /*** Make synthetic wave ****/
      for(i=0,var=0.0,pw=0.0 ; i<allnum ; i++){
        for(j=0,cals[i]=0.0 ; j<nm ; j++){
          cals[i]+=strc.mtxA[i][j]*mtn[j] ;
        }
        dx=cals[i]-obs[i] ;
        dx*=dx ;
        var+=dx ;
        pw+=obs[i]*obs[i] ;
      }
      vr = (1-var/pw)*100.0 ;
      /* END MT INVERSION */

      /* Result Out Put */
      sm1=sm2=sm3=sm4=sm5=0.0 ;
      /*****************************/
      //fp=fopen( "tdmrf_inv.out", "w" ) ;
      //fprintf( fp, "Moment tensor %d %d %1.1e\n", maxdnum, tsp, smoval ) ;
      //fprintf( fp, "Num  Mxx Mxy Mxz Myy Myz Mzz Mo CLVD strike dip rake\n" ) ;

      for(n=0 ; n<maxdnum ; n++){
        m1= -(float)mtn[2+5*n]; //  Basis tensor 1 Mxy
        m2= (float)mtn[1+5*n]; //  Basis tensor 2
        m3= -(float)mtn[4+5*n]; //  Basis tensor 3 Myz
        m4= -(float)mtn[3+5*n]; // Basis tensor 4 Mxz
        m5= (float)mtn[0+5*n]+(float)mtn[1+5*n]; // Basis tensor 5
        m6= 0 ;
        sm1+=m1 ;sm2+=m2 ; sm3+=m3 ;sm4+=m4; sm5+=m5 ;
        dcp_( &m1,&m2,&m3,&m4,&m5,&m6,&fstr[0],&fdip[0],&frak[0],&fmo[0],&fdmo[0],&mm[0
],&mm[1],&mm[2] ) ;
        dcp2_( &m1,&m2,&m3,&m4,&m5,&m6,&fstr[1],&fdip[1],&frak[1],&fmo[1],&fdmo[1]) ;
        mw=(log10((double)fmo[0]*1.0e13)-9.1)/1.5;
        for(i=0,mmmax=0, mmmin=1.0e+30 ; i<3 ; i++){
          if( mmmax < fabs(mm[i]))
            mmmax=fabs(mm[i]) ;
          if( mmmin > fabs(mm[i]))
            mmmin=fabs(mm[i]) ;
        }
        //printf( "Moment tensor %d\n", n ) ;
        //printf( "  Mxx = %f\n", -mtn[0+5*n] ) ;
        //printf( "  Mxy = %f\n", -mtn[2+5*n] ) ;
        //printf( "  Mxz = %f\n", -mtn[3+5*n] ) ;
        //printf( "  Myy = %f\n", -mtn[1+5*n] ) ;
        //printf( "  Myz = %f\n", -mtn[4+5*n] ) ;
        //printf( "  Mzz = %f\n", mtn[0+5*n]+mtn[1+5*n] ) ;
        //printf( "  CLVD= %f\n", 200.0*mmmin/mmmax ) ;
        //printf( "  Mw %1.1f\n", mw ) ;
        //printf( "  Moment %1.3e\n",(double)fmo[0]*1.0e13 ) ;
        //printf( "str1 %3.1f dip1 %2.1f rak1 %4.1f\n",
        //              fstr[0],fdip[0],frak[0]) ;
        //printf( "str2 %3.1f dip2 %2.1f rak2 %4.1f\n",
        //      fstr[1],fdip[1],frak[1]) ;

        //fprintf( fp, "%d     %1.5e %1.5e %1.5e %1.5e %1.5e %1.5e %1.4e %1.1f %1.1f %1
.1f %1.1f\n",
        //             n+1, -mtn[0+5*n]*1.0e13, -mtn[2+5*n]*1.0e13, -mtn[3+5*n]*1.0e13,
 -mtn[1+5*n]*1.0e13,
        //       -mtn[4+5*n]*1.0e13, (mtn[0+5*n]+mtn[1+5*n])*1.0e13, fmo[0]*1.0e13,
        //       200.0*mmmin/mmmax, fstr[0],fdip[0],frak[0]) ;

      }

      sm6=0.0 ;
      dcp_( &sm1,&sm2,&sm3,&sm4,&sm5,&sm6,&fstr[0],&fdip[0],&frak[0],&fmo[0],&fdmo[0],&
mm[0],&mm[1],&mm[2] ) ;
      dcp2_( &sm1,&sm2,&sm3,&sm4,&sm5,&sm6,&fstr[1],&fdip[1],&frak[1],&fmo[1],&fdmo[1])
 ;
      mw=(log10((double)fmo[0]*1.0e13)-9.1)/1.5;
      for(i=0,mmmax=0, mmmin=1.0e+30 ; i<3 ; i++){
        if( mmmax < fabs(mm[i]))
          mmmax=fabs(mm[i]) ;
        if( mmmin > fabs(mm[i]))
          mmmin=fabs(mm[i]) ;
      }

      //printf( "TOTAL\n" );
      //printf( "  CLVD= %f\n", 200.0*mmmin/mmmax ) ;
      //printf( "Var.Red = %1.2f\n", vr ) ;
      //printf( "str1 %3.1f dip1 %2.1f rak1 %4.1f\n",
      //              fstr[0],fdip[0],frak[0]) ;
      //printf( "str2 %3.1f dip2 %2.1f rak2 %4.1f\n",
      //              fstr[1],fdip[1],frak[1]) ;
      //printf( "Mw %1.1f\n", mw ) ;
      //printf( "Moment %1.3e\n",fmo[0]*(float)1.0e13 ) ;
      //fprintf( fp, "TOTAL %1.5e %1.5e %1.5e %1.5e %1.5e %1.5e %1.4e %1.1f %1.1f %1.1f
 %1.1f\n",
      //     (-sm5+sm2)*1.0e13,sm1*1.0e13,sm4*1.0e13,-sm2*1.0e13,sm3*1.0e13,sm5*1.0e13,
fmo[0]*1.0e13,
      //     200.0*mmmin/mmmax, fstr[0],fdip[0],frak[0]) ;
      //fprintf( fp, "VR %1.2f\n",vr ) ;
      //fprintf( fp, "Best Zcor " ) ;

      //for(i=0 ;i<stnum ; i++){
      //  fprintf( fp, "%d ", bzcor[i] ) ;
      //}
      //fprintf( fp, "\n" ) ;
      //mrr=sm5 ;
      //mtt=(-sm5+sm2) ;
      //mff=-sm2 ;
      //mrt=sm4 ;
      //mrf=-sm3 ;
      //mtf=-sm1 ;
      //// Mxx Mxy Mxz Myy Myz Mzz
      fprintf( fpout, "%s %s %s %s %s %d %s %s %s ",
               evlab[0],evlab[1],evlab[2],evlab[3],evlab[4],z,evlab[5],evlab[6],deps[gf
num]) ;
      fprintf( fpout, "%1.1f %1.2f %1.1f %1.1f %1.1f ", mw, vr, fstr[0],fdip[0],frak[0]
) ;
      //// Mxx Mxy Mxz Myy Myz Mzz
      fprintf( fpout, "%1.5e %1.5e %1.5e %1.5e %1.5e %1.5e %1.4e %1.2lf\n",
               (-sm5+sm2)*1.0e13,sm1*1.0e13,sm4*1.0e13,-sm2*1.0e13,sm3*1.0e13,sm5*1.0e1
3,fmo[0]*1.0e13,
               200.0*mmmin/mmmax ) ;

      ///fprintf( fpout, "%1.2f %1.2f %1.2f %1.2f %1.2f %1.2f 1.0e+13 %1.2lf\n",
       ///       mrr,mtt,mff,mrt,mrf,mtf, 200.0*mmmin/mmmax ) ;

    }
    // No use following part in the SPA system
    //fprintf( fp, "L.V.R ") ;
    ///* out put data for GMT */
    //for(i=0,cnt1=0 ; i<stnum ; i++){
    //np=obsleng[i] ;
    // //printf( "NP %d\n",np) ;
    //for(j=0,maxamp=0.0 ; j<np ; j++){
    //  // Trans
    //  outdata[0][j]=obs[j+cnt1] ;
    //  outsyn[0][j]=cals[j+cnt1] ;
    //  // Radial
    //  outdata[1][j]=obs[j+np+cnt1] ;
    //  outsyn[1][j]=cals[j+np+cnt1] ;
    //  if( waved.cmpn[i]==3){
    //// Z
    //outdata[2][j]=obs[j+np+np+cnt1] ;
    //outsyn[2][j]=cals[j+np+np+cnt1] ;
    //  }
    //
    //  // Trans
    //  if( maxamp < fabs(outdata[0][j]) )
    //maxamp= fabs( outdata[0][j]) ;
    //  if( maxamp < fabs(outsyn[0][j]) )
    //maxamp= fabs( outsyn[0][j] )  ;
    // // Radial
    //  if( maxamp < fabs(outdata[1][j]))
    //maxamp=fabs(outdata[1][j]) ;
    //  if( maxamp < fabs(outsyn[1][j]))
    //maxamp=fabs(outsyn[1][j]) ;
    //  if( waved.cmpn[i]==3 ){
    //// Z
    //if( maxamp < fabs(outdata[2][j]))
    //  maxamp=fabs(outdata[2][j]) ;
    //if( maxamp < fabs(outsyn[2][j]))
    //  maxamp=fabs(outsyn[2][j]) ;
    //  }
    //
    //}
    //// Calculation Local Var.Red.
    ////for(j=0,var=0.0,pw=0.0 ; j<waved.cmpn[i]; j++){
    //for(k=0 ;k<np ; k++){
    //dx=outdata[j][k]-outsyn[j][k] ;
    //dx*=dx ;
    //var+=dx ;
    //pw+=outdata[j][k]*outdata[j][k] ;
    //  }
    //}
    //vr=(1-var/pw)*100 ;
    //fprintf( fp, "%1.1lf ", vr ) ;
    /////////////////////////
    //cnt1+=waved.cmpn[i]*np ;


    //buffer=calloc(100,sizeof(char)) ;
    //sprintf(buffer, "%s_gmt.out", obsfname[i] ) ;
    //printf( "OUTFNAME %s\n", buffer ) ;
    //printf( "MAXAMP %lf\n",maxamp) ;

    //fp2=fopen( buffer, "w" ) ;
    //fprintf( fp2, "%1.4e\n",maxamp ) ;

    //for(j=0 ; j<np ; j++){
    //  if( waved.cmpn[i]==3){
    //fprintf( fp2, "%1.4f %1.4f %1.4f %1.4f %1.4f %1.4f\n",
    //   -outdata[0][j]/maxamp, -outsyn[0][j]/maxamp,
    //   -outdata[1][j]/maxamp, -outsyn[1][j]/maxamp,
    //   -outdata[2][j]/maxamp, -outsyn[2][j]/maxamp ) ;
    //  }
    //  else{
    //fprintf( fp2, "%1.4f %1.4f %1.4f %1.4f\n",
    //   -outdata[0][j]/maxamp, -outsyn[0][j]/maxamp,
    //   -outdata[1][j]/maxamp, -outsyn[1][j]/maxamp ) ;
    //  }
    //}

    //fclose(fp2) ;
    //}

    // close mtinv.out
    //fprintf( fp,"\n" ) ;
    //fclose(fp) ;
  }
  fclose(fpout) ;
  /***** Free memory *******/
  return(0) ;
}
```