fdacrtmc: An acoustic RTM code

Max Enno Holicki

June 25, 2019

1 Introduction

FDACRTMC is a **F**inite **D**ifference **AC**oustic **R**everse **T**ime **M**igration code in the language **C** using OpenMP and operates on wavefields travelling in two spatial and one temporal dimensions. It works by cross-correlating forward modelled source wavefields with backward propagated receiver wavefields for the zero-lag imaging condition.

2 Options

What follows is a short description of the options offered by **fdacrtmc**. The mandatory inputs are:

```
1.  file_cp= #P-wave velocity file
2. file_den= #Density file
3. file_src= #Source wavefield
```

These are all filenames corresponding to the input model and source array. See their corresponding sections.

The following options define how the wavefield is modelled:

```
4. ischeme=1 #Finite difference scheme
             #1=acoustic, 2=visco-acoustic 3=elastic, 4=visco-elastic
5.  iorder=4 #Finite difference order
             #2=2nd, 4=4th, 6=6th
```

Option 4. defines the type of wave equation to model. Note that the wavefield decomposition schemes are only implemented for the acoustic case. By default acoustic wavefields are modelled. Option 5. defines the finite difference order. The higher the order the more exact the result in a homogeneous medium and the longer the modelling takes. Higher orders also smooth over interfaces, degrading results there. The default is to use a fourth order scheme, which seems to have an optimal tradeoff between accuracy and computation time.

The following options define the boundary type:

```
                #Boundary Types:
                #1: Free Surface
                #2: Perfectly Matched Layers (PML)
                #3: Rigid Surface
                #4: Tapered boundary
6.     top=1    #Top boundary type
7.  bottom=2    #Bottom boundary type,
8.    left=2    #Left boundary type
9.   right=2    #Right boundary type
10.   npml=     #Number of PML layers
11. ntaper=     #Taper Length
12.      R=1e-4 #Theoretical PML reflection coefficient
13.      m=2.0  #PML scaling function sigma order
```

Options 6–9. define the type of boundary at the side of the model. By default the right, bottom and left (clockwise) are absorbing boundaries, **P**erfectly **M**atched **L**ayers to be precise. The top boundary by default is a free surface. Alternative options for the model boundaries are to use rigid or tapered boundaries. The tapered boundary is an offset free-surface boundary where the wavefield is tapered to zero before it hits the boundary. Options 10. and 11. specify the number of PMLs at the boundary or the taper size respectively. Options 11. and 12. are related to the PML boundaries, see [**?**].

If you want to model wavefields the following output options are important:

```
14. file_rcv=   #Output receiver data base filename
15. rcv_write=0 #Write out receiver wavefield to disk",
16. file_snp=   #Output snapshot file
```

Option 14. defines the base filename of the output receiver data. The filename must end in ".su". Depending on the selected types of receivers the component type is appended to the filename before the ".su". Option 15. toggles if recorded wavefields should be written to disk. The option can also be set to zero to not record wavefields to disk but still use the recorded wavefields as migration input. Option 16. defines the base filename of the output snapshot gathers. It must also end in ".su". The snapshot type is also appended before the ".su".

There are two ways of specifying receiver coordinates and types. Either directly via the command line or more flexibly via SEG-Y headers. To specify reciever coordinates via the command line the following options are pertinent:

```
17    rcv_top=0 #Receivers along top    edge of model
18 rcv_bottom=0 #Receivers along bottom edge of model
19   rcv_left=0 #Receivers along left   edge of model
20  rcv_right=0 #Receivers along right  edge of model
21      rcv_p=0 #Record pressure
22     rcv_vx=0 #Record horizontal particle velocity
23     rcv_vz=0 #Record vertical   particle velocity
```

Options 17–20. specify which boundaries along which receivers should be placed. Note that pressure receivers are placed at the boundaries while particle velocity sensors are placed just inside the model as the code uses a staggered grid. Options 21–23. specify what wavefield components to record, respectively the pressure, and the horizontal and vertical particle velocities.

Alternatively or additionally receiver locations can be specified using SEG-Y trace headers:

```
24. file_loc= #Receiver location file.
```

Option 24. allows one to use a seismic unix file, SEG-Y file stripped of file headers, to specify receiver coordinates. Note that only the first field record is read to define the coordinates. The important header words here are the horizontal group coordinate (**gx**) and the group elevation (**gelev**). Note that these coordinates are scaled by there respective scale factors, **scalel** and **scalco**. Additionally the trace identification (**trid**) defines the receiver type. A value of one defines a pressure receiver while a value of five or six define horizontal and vertical particle velocity receivers respectively.

To specify snapshots the following options are important:

```
25.         tsnap1=0.1  #First snapshot time
26.         tsnap2=0.0  #Last snapshot time
27.         dtsnap=0.1  #Snapshot temporal sampling rate
28.         dxsnap=dx   #Horizontal snapshot sampling rate
29.         dzsnap=dz   #Vertical   snapshot sampling rate
30.         xsnap1=xmin #X-coordinate of upper left  snapshot point
31.         xsnap2=xmax #X-coordinate of lower right snapshot point
32.         zsnap1=zmin #Z-coordinate of upper left  image point
33.         zsnap2=zmax #Z-coordinate of lower right image point
34.    snapwithbnd=0    #Write snapshots with absorbing boundaries
35.    snap_type_p=0    #Pressure Snapshots, default 1 if none selected
36.   snap_type_vx=0    #Horizontal Particle Velocity Snapshots
37.   snap_type_vz=0    #Vertical Particle Velocity Snapshots
38.   snap_type_pu=0    #Up-Going     Pressure    Snapshots
39.   snap_type_pd=0    #Down-Going   Pressure    Snapshots
40.   snap_type_pl=0    #Left-Going   Pressure    Snapshots
41.   snap_type_pr=0    #Right-Going  Pressure    Snapshots
42.   snap_type_pn=0    #Normal-Going Pressure    Snapshots
43. snap_vxvztime=0     #Registration time for vx/vz
                        #The FD scheme is also staggered in time.
                        #Time at which vx/vz snapshots are written:
                        #- 0=previous vx/vz relative to txx/txz/tzz(p) at time t
                        #- 1=next     vx/vz relative to txx/txz/tzz(p) at time t
```

Option 25. specifies the first snapshot time, while Option 26. specifies the last snapshot time. If the last snapshot time occurs before the first snapshot time no snapshots are recorded. The default is not to record snapshots. Option 27. specifies the temporal snapshot interval. Snapshots are recorded at this rate from the first snapshot onwards. Options 28. and 29. specify the horizontal and vertical snapshot sampling rate, by default this corresponds to the model sampling

rate. Options 30–33. specify top-left and bottom-right corners of the spatial snapshot window. By default this covers the entire model. Option 34. is active if these coordinates are not specified, then the snapshot area also includes the PML and tapered boundaries. Options 35–38 specify what type of snapshot should be taken, pressure, and/or horizontal and/or vertical particle velocity. Options 38–43. specify if directionally decomposed pressure snapshots should be recorded. Options 38. and 39. corresponds to up- or down-going snapshots respectively while options 40. and 41. correspond to left- and right-going snapshots respectively. Option 42. corresponds to creating a snapshot in a user defined direction. Option 43. specifies if the particle velocity snapshots should be for the previous time instance or the next time instance with respect to the pressure as the employed modelling scheme is staggered in time.

3 Example: Plane-Wave Migration – Synform Model

Let us begin by migrating a simple model. We are going to do a plane wave migration, meaning that we are going to inject a horizontal plane-wave source wavefield at the surface, record the corresponding reflection response at the surface and use that to form an image of the subsurface. We will construct the image of the subsurface by forward modelling the source wavefield and back-injecting the recorded wavefield and zero-lag cross-correlating the two.

Note that this example will be done in bash. It is strongly recommended to install the seismic unix package[1]. It is assumed that you are familiar with the seismic unix, that you have 0.1 GiB of available storage and 0.16 GiB of RAM.

We begin:

```
# 0.1: Go to the Examples/Synform directory.
cd <fdacrtmc Install Path>/Examples/Annerveen

# 0.2: Display the files in the directory
ls
```

where <FDACRTMC Install Path> is the install path for **fdacrtmc**.

Inside the directory you will find three files:

```
1. Run.scr                  #The script to run the migration.
2. Synform_cp.su            #The Annerveen velocity model.
3. SrcWav_Ricker_40Hz_2s.su #A 40 Hz peak-frequency Ricker wavelet.
```

We can have a look at the velocity model using **suximage**, from the seismic unix package, and at the source wavelet using **suxwigb**.

```
# 0.3: Display Velocity Model
suximage <Synform_cp.su title=Synform_Velocity_Model &

# 0.4: Display Ricker Wavelet
suxwigb <SrcWav_Ricker_40Hz_2s.su title=20Hz_Ricker_Wavelet &
```

We now wish to migrate this model using reverse time migration. To do this we need to generate some data. We will do this by using **fdacrtmc** to forward model some synthetic data using a horizontal plane-wave source along the top of the model. We will record the wavefield along the top of the model as well. To do this we need to generate an array of sources. The model size is 601-by-2401 elements which corresponds to an actual size of 1.5-by-6 km with an element spacing of 2.5 m in both spatial directions. We want to place sources every 2.5 meters along the top of the model leaving a 1 km gap to the vertical boundaries on either side of the model. This means that we need a horizontal plane-wave source arry consisting of 1601 sources.

We need to prep our input source array for this. We begin by duplicating the Ricker wavelet 161 times and concatenating the results:

```
# 1.1: Generate Base Source Array
for i in {1..1601};do SrcWav_Ricker_40Hz_1s.su >> SrcArr.su; done
```

We now have 1601 identical sources inside **SrcArr.su**. We want to equally space them out along the top of the model with 2.5 m spacing starting at an offset of -2 km to the top-middle of the model, which corresponds to the origin. To do this we need to modify the source horizontal-location header word (**sx**) of the source array such that each source is at a different location.

```
# 1.2: Set header words.
sushw <SrcArr.su key=tracl,tracr,fldr,tracf,scalco,sx a=1,1,1,1,-10,-20000\
        b=1,1,0,1,0,25 >tmp.su; mv -f tmp.su SrcArr.su;
```

[1] https://github.com/JohnWStockwellJr/SeisUnix/wiki (Accessed:June 25, 2019)

We now have a horizontal plane-wave source array consisting of 161 unique sources. It was important in this case to also modify the **scal**ing for **el**evation (**scalel**) and increase by a factor of 10 such that 2.5 m can be represented as the interger 25 dm.

Now that the source array is ready we can begin to model our recorded wavefields along the top of the model. For this example we will be recording pressure data. We will now model the data using **fdacrtmc**.

```
# 1.3: Model Data
fdelrtmc \                          #The RTM engine
        file_cp=Synform_cp.su\      #The input acoustic velocity model
        file_src=SrcArr.su\         #The input source array
        file_rcv=RcvArr.su\         #The output receiver data base filename
        top=2\                      #Top boundary is absorbing, others are by default
        npml=50\                    #Number of absorbing layers
        rcv_top=1\                  #Specifies to record along the top of the model
        rcv_p=1\                    #Specifies to record pressure data
        rcv_write=1 \               #Write out recorded data
        mig_mode=0\                 #Specifies to only model data, not migrate data
        verbose=2;                  #Verbosity level, >0 so that we see something
```

This will take some time. Take a break, get a drink. You can check how far the modelling is at any time. Read on when done, or if interested.

Now that modelling the data is complete we are nearly ready to migrate the data. There is just one small problem. Our recorded data has the direct wave. We need to remove it. There are various ways to do this, the simplest is to model it and subtract it from the recorded data.

To do this we need to create a new model that is the same as the Synform model at the top but otherwise homogeneous such that we can model the direct wave. Luckily the Synform model is homogeneous along the top. The acoustic velocity at the top is 1,900 m s^{-1}. We can make homogeneous copies of the Synform model as follows:

```
# 2.1: Create Direct-Wave Models
suwind <Annerveen_cp.su itmax=50 | sugain dt=1 scale=0.0\
        | sugain dt=1 bias=1900 >Direct_cp.su;
```

Note that we have reduced the models vertically in size to reduce computation time.

Now that we have the direct-wave models we can model the direct wave in the same way as we modelled the full wavefield:

```
# 2.2: Model Direct-Wave Data
fdelrtmc\
        file_cp=Direct_cp.su\ #Direct-wave velocity model
        file_src=SrcArr.su\   #Same source array
        file_rcv=DirArr.su\   #The output direct-wave record base filename
        top=2 npml=50 rcv_top=1 rcv_p=1 mig_mode=0 verbose=2;
```

Now that we have modelled the direct-wave data we can subtract it from recorded data of the full wavefield to remove it:

```
# 2.3: Remove Direct Wave
sudiff RcvArr.su DirArr.su >RcvArr_ND.su; #Direct-Wave-Free Data
```

The data is now nearly ready for migration. We just need to taper the last 50 ms of each trace such that the modelling does not become unstable during back-injection of the recorded wavefield.

```
ntr=$(surange <RcvArr_ND.su key=tracl\              #We need to get the number
    | sed -n '1p' | awk '{print $1}');              #of traces in the file
sutaper <RcvArr_ND.su ntr=${ntr} tend=50 >tmp.su;   #This tapers the data
mv -f tmp.su RcvArr_ND.su;                          #at the end of the trace
```

Note that we need the number of traces for **sutaper** to work.

To save on space let us delete the unnecessary data:

```
# 2.5: Clean-up
rm -f Direct_cp.su Direct_ro.su RcvArr.su DirArr.su;
```

Now its time to migrate. Reverse time migration is a memory intensive task as it needs to store one of the modelled wavefields in its entirety. This migration, which is quite small, needs about 1.3 GiB of Random Access Memory (RAM).

Bigger models needs significantly more. As such it makes sense to talk about compression. The compression option compresses uses the zfp package[2] to compress the stored wavefield. With the default precision of 3 decimals, which is sufficiently accurate for the generally employed fourth order finite difference scheme, the memory consumption drops down to 0.16 GiB. The penalty that compression incurs is that the code takes about 1.5 times longer to run. If you are RAM constrained consider turning it on for the following runs.

We migrate the data by calling **fdacrtmc** using an actual migration mode:

```
# 3.1: Migrate!
fdelrtmc file_cp=Synform_cp.su file_src=SrcArr.su\
        file_rcv=RcvArr_ND.su\ #The input receiver data
        file_mig=Synform.su\   #Base Filename of Migrated Output
        top=2 npml=50\
        migdt=0.0025\          #This upsamples the cross-correlation
        migdx=5 migdz=5\       #This upsamples the final migration image
        mig_mode=1\            #The zero-lag imaging condition
        compress=0 \           #Turning this on reduces memory consumption
        verbose=2;
```

This is going to take some more time. Take another break, eat a sandwich, etc. This will take twice as long as modelling the original data. Continue reading when this is done.

Once the migration is done can have a look at the final migrated image:

```
# 3.2: Display Migrated Image
suximage <Synform_mig.su title=Migrated_Image
```

Note that "Synform_img.su" contains the individual migration images.

The migration image looks acceptable, there are some low vertical-wavenumber artefacts though. The traditional fix for this is to apply a Laplacian filter to the image. A better alternative is to directionally decompose the wavefield before imaging and to then only image wavefield components travelling in opposite directions. This can be robustly done using the Hilbert transform migration mode, see [**?**]:

```
# 3.3: Migrate Using The Hilbert Transform Imaging Condition
fdelrtmc file_cp=Synform_cp.su file_src=SrcArr.su file_rcv=RcvArr_ND.su\
        file_mig=Hilbert.su\ #Base Filename of Migrated Output
        top=2 npml=50 migdt=0.0025 migdx=5 migdz=5\
        mig_mode=5\          #Hilbert Transform Imaging
        compress=0 verbose=2;
```

This will take even longer, not much but a bit.

After the migration is done we can compare the migration results:

```
# 3.4: Display Migrated Image
suximage <Hilbert_mig.su title=Hilbert_Image
```

The image now looks like a real migration image.

Congratulations! You have successfully migrated the Annerveen model. Now go out and explore the **fdacrtmc** options or apply the code to your own data.

4 Modelling Engine

At the heart of this code is the same modelling engine as used by **FDELMODC**[3] by Jan Willem Thorbecke.

[2] https://computation.llnl.gov/projects/floating-point-compression (Accessed on:June 25, 2019)
[3] https://janth.home.xs4all.nl/ (Accessed on:June 25, 2019)