# SPECFEM3D_ANAT- a script driven package for ambient noise adjoint tomography

Kai Wang, University of Toronto

December, 2018

This is a simple guide for conducting ambient noise adjoint tomography using our ANAT package. We first introduce the workflow of the three-stage iterative inversion in section 1, and the structure of the package in section 2. Then, a step-by-step tutorial of running iterative inversion is given in section 3. Finally, we give the details for some scripts and programs in section 4 as references.

## 1 Workflow

The basic idea of ANAT is to iteratively minimize the traveltime misfit between empirical Green's functions (EGFs) from ambient noise and synthetic Green's functions (SGFs) from spectral-element simulations based on misfit gradient. According to the job submission schedule, we divide the inversion into three stages as shown in Fig. 1. In stage one, we conduct the forward and adjoint simulation in one PBS job (including misfit measurement) to obtain the event kernel of each virtual source (or master station). After all the simulations are finished, event kernels are summed, preconditioned, and smoothed in stage two to obtain the final misfit gradient. In stage three, we use line search to determine the optimal step length for model updating.

## 2 Package structure

ANAT package consists of some workflow-control bash scripts and programs in several folder.

### 2.1 Inputs

- **data** is a database of ambient noise symmetric cross correlation functions sorted by virtual sources. All the data are in SAC format and named as data/net1.evt1/net1.sta1. comp.sac, where net1 and evt1 are the network and name for virtual source 1, sta1 is for station 1, comp is the component. For example, data/CI.STC/CI.BAK.HXZ.sac is the EGF between the master station CI.STC and general station CI.BAK.

- **src_rec** stores all source and station information needed for the inversion. There are three files prepared for the `SPECFEM3D Cartesian`, namely CMTSOLUTION_evt1, FORCESOLUTION_evt1, and STATIONS_evt1. The format of these files should be the same as used in `SPECFEM3D Cartesian`. The other two files sources_set1.dat and sources_ls.dat are name lists of event (virtual sources), which are used for making directories for simulations by event name. We can divide the events into different sets, such as set1, set2, ..., etc.

- **specfem3d** contains three inputs for the software `SPECFEM3D Cartesian`, **DATA**, **OUTPUT_FILES**, and **bin**. These three inputs are essential for launching numerical simulations, and will be linked to all forward simulations directories.

## 2.2 Data processing and plot

- **ADJOINT_TOMOGRAPHY_TOOLS** is a bunch of seismic tools, among which we use **flexwin** for picking window and **measure_adj** for misfit measuring.
- **seismic_process** contains some C shell and Perl scripts for data preprocessing, such as converting ASCII seismogram to SAC files, bandpass filtering, etc.
- **plots** contains some example scripts for plotting misfit, model, kernel, waveform. The corresponding directories are **misfit_plot**, **model_plot**, **kernel_plot**, **seismo_plot**.

## 2.3 Outputs

- **output** stores all the misfits files for a specific model. For example, all misfits for model M01 are stored in ./output/misfits/M01.
- **optimize** is where postprocessing procedure applied. All event kernels are processed in the directories **sum_kernels_M01**, and model updating files are stored in **SD_M01**. All inputs and outputs in these two directories are listed in Fig. 2.
- **solver** is where numerical simulations are performed and stored. For event1 in the source list set1, the forward simulation root directory is solver/M01.set1/event1. For line search at step length of 0.01, the corresponding directory is solver/M01.slen0.01/ event1. Inside each forward simulation root directory, the three sub-directories of **specfem3d** are linked as the inputs for `SPECFEM3D Cartesian`. The script *change _simulation_type.pl* and directory **SEM** is generated for adjoint simulation.

## 2.4 Driven scripts

- *run_iteration.bash* is the main scripts for running forward and adjoint simulations. This scrips will prepare all inputs in the forward simulation directories located at **solver** and call the other two scripts *run_preprocessing.3band.bash* and *pbs_mesh_fwd _measure_adj.bash*.
- *run_preprocessing.3band.bash* is to make all inputs and directories for preprocessing in **seis_process_M01.set1**, and for misfit measurement in **measure_adj_M00.set1**.
- *run_postprocessing.bash* is to prepare all inputs and directories for postprocessing needed by the script *pbs_postprocessing.bash*.
- *run_line_search.bash* is the same as *run_iteration.bash* but for a linear search.

## 2.5 PBS scripts

- *pbs_mesh_fwd_measure_adj.bash* is the PBS script to do meshing, forward simulations, misfit measuring, and adjoint simulations.
- *pbs_postprocessing.bash* is the PBS script to do postprocessing procedures include, kernel summation, precondition, and smoothing.

# 3 Iterative inversion procedures

There are four basic steps to do the iterative inversion using our ANAT package.

## 3.1 Preparing input files

Before running any iteration, the users should write their own scripts to make **data** and **src_rec** in the format described above. The corresponding example scripts are *generate_data .bash* and *mk_forcesolution.bash*.

The inputs files in **specfem3d** should be tested to make sure that they are ready for running any simulations. This means that the user should set up the meshing files at *DATA/mesh fem3D_files*, initial xyz model at *specfem3d/DATA/tomo_files* or gll model in a user defined folder, and *DATA/Par_file*. Set APPROXIMATE_HESS_KL = .true. and USE_RHO_SCAL ING = .true. in *Par_file*. For testing if the mesh and initial model is correctly set up, set SAVE_MESH_FILE = .true. in *Par_file* and plot the velocity model *.bin files in *OUTPUT_FILES/DATABASES_MPI*, and set it back to .false. after the test. The user should also check *output_mesher.txt* and *output_meshfem3D.txt* for the minimum period resolved, suggested time step, and other useful information regarding the meshing and solver parameters.

## 3.2 Running forward and adjoint simulations

The first step of the iterative inversion is to run the script *run_iteration.bash*. We should first set the model name (such as mod=M01), and step length (such as step=0.02) for reading updated model from previous iterations. Basically, this script can be divided into three parts, including:

1. Making forward simulation directory. The script will make all forward simulation directories by looping over event sets defined by a variable ipart according to the file sources_set$ipart.dat. Then, it will link the all input files in **specfem3d**, corresponding STATIONS and FORCESOLUTION files in **src_rec** to each forward directory. The script also have an option to change the model for the first iteration and other iterations.

2. Running *run_preprocessing.sh*. We adopt a multi-scale strategy in misfit measurement, thus there are different version of these scripts, such as *run_preprocessing. 1band.sh*, *run_preprocessing.2band.sh*, *run_preprocessing.3band.sh*. The author should choose how many and what frequency bands needed in current iteration depending on the data, then make changes to these pre-processing scripts. There will be two directories created, namely **seis_process_M01.set1** and **measure_adj_M01.set1**. The former is to do some pre-processing procedures on data and synthetics, and the latter is to measure the misfits between each data and synthetic pair and calculate the corresponding adjoint sources.

3. Submitting *pbs_mesh_fwd_measure_adj.sh*. As long as all forward and data processing directories are created correctly, this script is ready to be submit after setting the proper walltime. After all the forward and adjoint simulations are successfully finished, the event kernels and Hessians are generated at solver/M01.set1/event1/OUTPUT _FILES/DATABASES_MPI/*kernel.bin.

## 3.3 Post-processing and model update

Then, we use the following two steps to do post-processing on event kernels to obtain the total misfit gradient and update the model.

1. Run *run_postprocessing.sh*. We should first set the current model name (such as mod=M01), and step length (such as step=0.02) for reading the current model from previous iterations. This script is also divided into two parts, which is controlled by setting is_sumkern=.true. and is_update=.true. The resulting two directories are **sum_kernels_M01** and **SD_M01**, where "SD" refers to Steep Descent method. In the model update part, the user should set proper step lengths for line search in the text stage, and also the input model for updating. If it is the first iteration, the input model is the initial model, otherwise, it is the updated model from previous iteration.

2. Submit *pbs_postprocessing.sh*. This script has three parts: is_sumkern, is_smooth, and is_update. The three part can submit in one time by setting them to true, or submit one by one sequentially. The most time consuming part is the kernel smoothness, which might take a few hours. The summed, preconditioned, and smoothed gradient are stored OUTPUT_SUM_KERNELS, OUTPUT_PREC_KERNELS,

and OUTPUT_SMOOTH_KERNELS. And the updated models are stored at OUTPUT_MODEL _slen0.01, OUTPUT_MODEL_slen0.02, ..., etc.

### 3.4 Line search for optimal step length

After the updated models are generated, we use them as trial models to do a line search in order to obtain the optimal step length. The script responsible for this is *run_line_search.bash*, which is changed from the script *run_iteration.bash*. The script will make all forward simulation directories by looping over step lengths, and each of them has a selected number of events from the file sources_ls.dat. The data processing script is still run_preprocessing.sh, but the PBS script is *pbs_mesh_fwd_measure.sh* without adjoint simulation. The option is_ls is turned to true, which will delete the forward output directories because we only need the misfit measurements.

As long as all the forward simulations for different trial models are finished, the corresponding traveltime misfits are collected in the directory **outputs/M01**. Then, we use the script *plt_line_search.3band.bash* to plot the total misfit curve over step length.

## 4 Data processing scripts

### 4.1 *run_preprocessing.sh*

This long script can be divided into two parts: pre_processing and measure_adj. In each part, the script will write all commands into an extra job$i.sh for paralleling.

In pre_processing, data and synthetics are re-sampled to 1Hz, cut to preferred length and band-passed filtered before window selection in the next step as suggested by FLEXWIN manual.Filtered data are normalized by the maximum of its absolute amplitude and then multiply the maximum of aboulute amplitude of synthetics.

The pre-processed data and synthetics are saved as:
data/CI.ADO/CI.BAK.BXZ.sac.T010_T020
solver/m00/CI.ADO/CI.BAK.BXZ.fwd.semd.sac.T010_T020
In measure_adj, we adapt several Perl scripts from [**?**] to do misfit measurement and make adjoint sources.
*run_measure_adj.pl*– main Perl script for running the program MEAS_ADJ.
*combine_3_adj_src.pl*– combine adjoint sources from three different frequency band

### 4.2 *process_data.pl*

I modyfied "process_data.pl" by adding -v option to get EGFs from time domain derivation of cross-correlation functions, -n option to normalize the amplitude and -A option to make the ammplitude of EGFs comparable to the their corresponding SGFs from SEM.

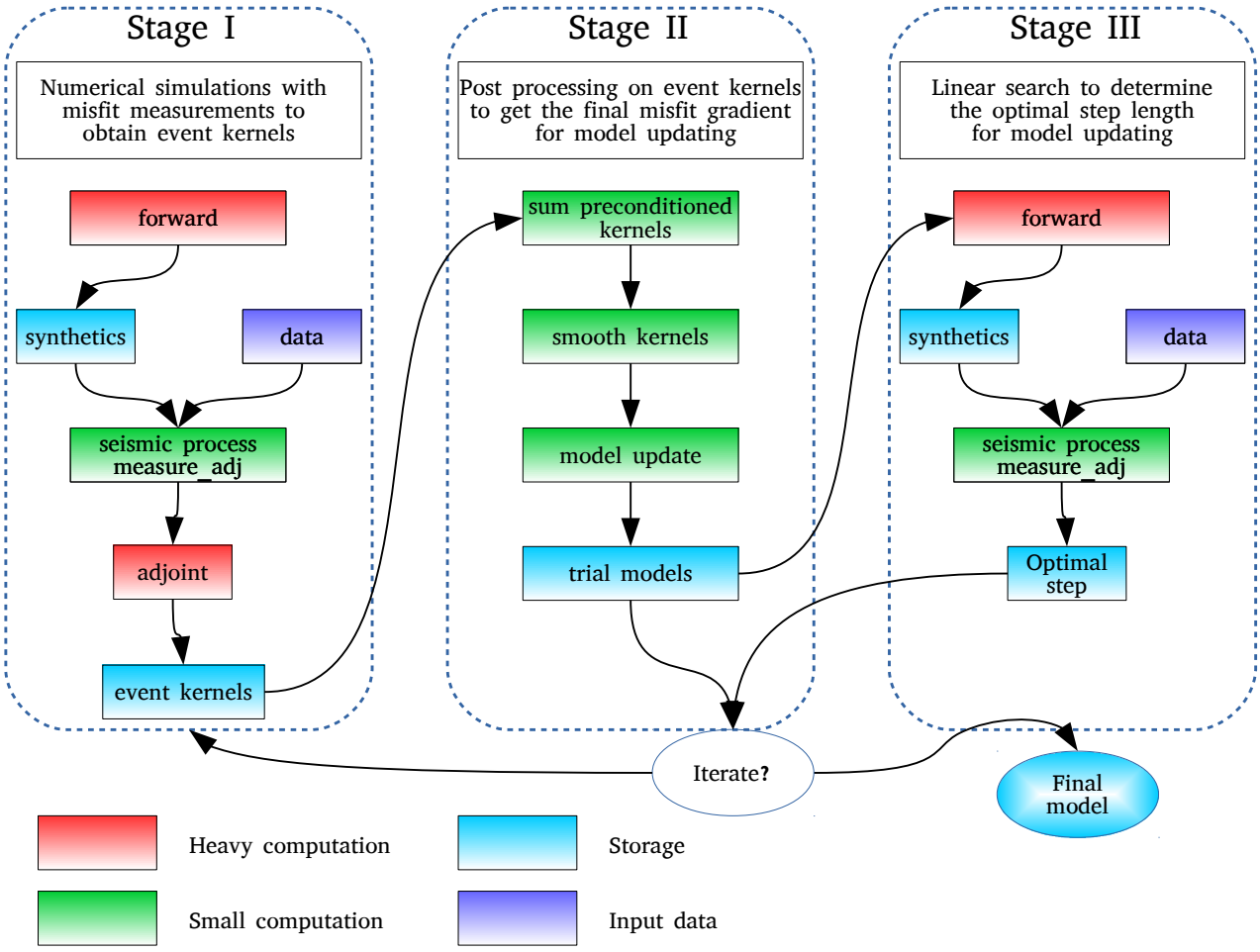


Figure 1: Workflow of ANAT package.

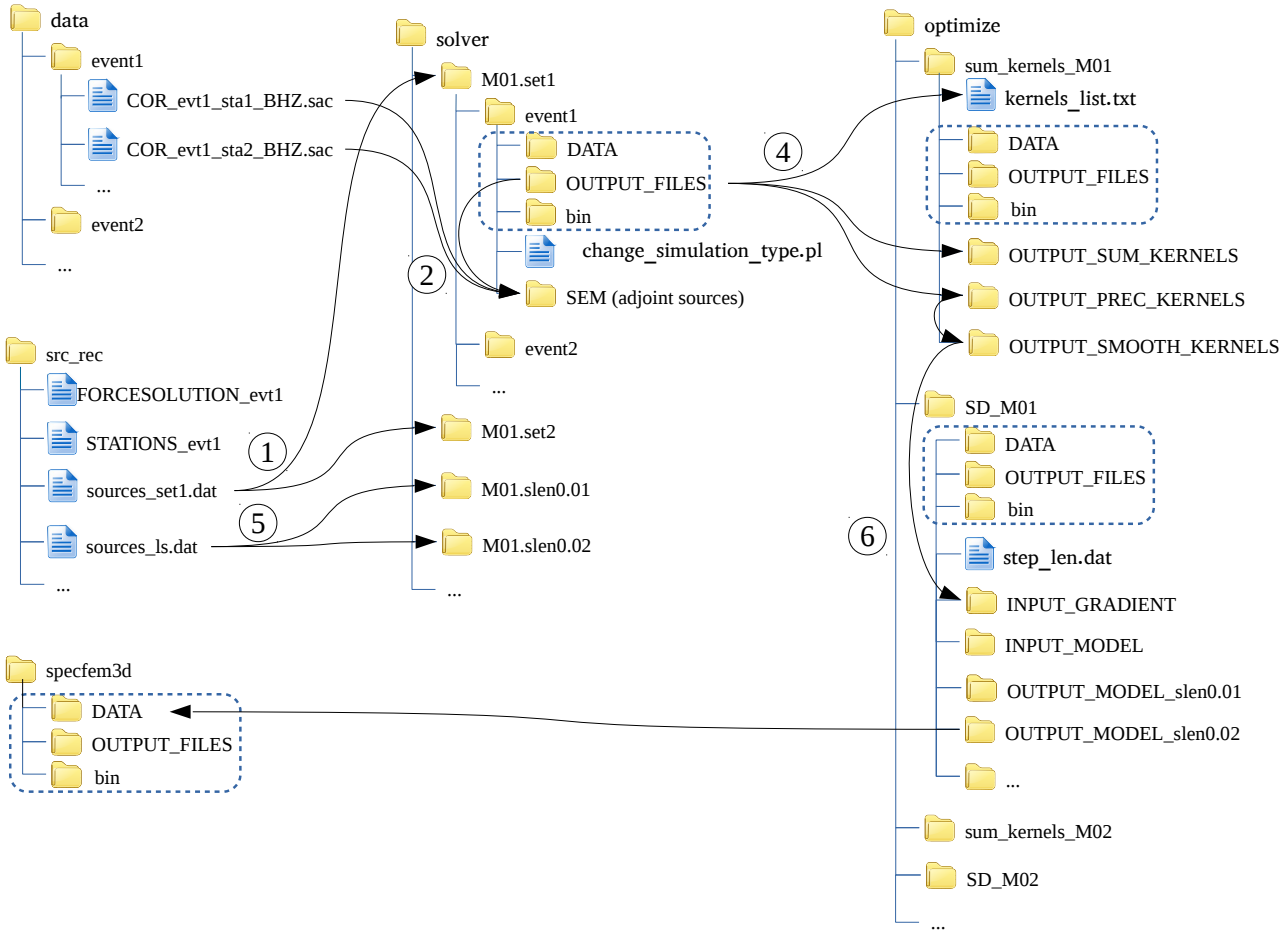


Figure 2: Structure of ANAT package.

```
#!/bin/bash


mod=M01.set1
srfile=sources_set1.dat

R0=2.5 # minimum group velocity  ### need to change this
R1=4.5 # naximum group velocity  ### need to change this
#======
is_preproc=true
is_meas_adj=true

is_bd=false # if banana-doughnought or multi-taper kernel
i_plot=false # if plot misfit measurements
#------
preprocdir=seis_process_$mod
measdir=measure_adj_$mod
kerndir=sum_kernels
#=========================================
if $is_preproc;then
i=1
cat src_rec/$srfile |
while read evtfile; do # loop over all virtual evts
  eid=`echo $evtfile |awk '{printf"%s.%s",$2,$1}'`
  fwd_dir=solver/$mod/$eid
  ####
  echo "seis_process commands" >$preprocdir/job$i.sh
  ####
  let i=i+1
done # end loop over evt
fi
#=========================================
if $is_meas_adj;then
i=1
cat src_rec/$srfile |
while read evtfile; do # loop over all virtual evts
  eid=`echo $evtfile |awk '{printf"%s.%s",$2,$1}'`
  fwd_dir=solver/$mod/$eid
  ###
  echo "meas_adj commands"  >$measdir/job$i.sh
  ###
  let i=i+1
done # end of loop over evts
fi
#=================================================
```

Figure 3: Command line of *run_preprocessing.sh*.

```
cat /dev/null >CI.ADO/process.log
perl process_syn.pl -m CI.ADO/CMTSOLUTION -a CI.ADO/STATIONS -s 100.0 -l -20/235 -t 5/10 -x T005_
T010 ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd >>CI.ADO/process.log
synmin=`saclst depmin f ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd.sac.T005_T010
|awk '{print $2}'`
synmax=`saclst depmax f ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd.sac.T005_T010
|awk '{print $2}'`
norm=`echo $synmin $synmax |awk '{if($1*$1>$2*$2) {print sqrt($1*$1);} else {print sqrt($2*$2)}}'
`
perl process_data.pl -m CI.ADO/CMTSOLUTION -s 100.0 -l -20/235 -t 5/10 -v -n -A $norm -x T005_T01
0 CI.ADO/DATA_NORM/CI.ALP.HXZ.sac >>CI.ADO/process.log
perl process_syn.pl -m CI.ADO/CMTSOLUTION -a CI.ADO/STATIONS -s 100.0 -l -20/235 -t 10/20 -x T010
_T020 ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd >>CI.ADO/process.log
synmin=`saclst depmin f ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd.sac.T010_T020
|awk '{print $2}'`
synmax=`saclst depmax f ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd.sac.T010_T020
|awk '{print $2}'`
norm=`echo $synmin $synmax |awk '{if($1*$1>$2*$2) {print sqrt($1*$1);} else {print sqrt($2*$2)}}'
`
perl process_data.pl -m CI.ADO/CMTSOLUTION -s 100.0 -l -20/235 -t 10/20 -v -n -A $norm -x T010_T0
20 CI.ADO/DATA_NORM/CI.ALP.HXZ.sac >>CI.ADO/process.log
perl process_syn.pl -m CI.ADO/CMTSOLUTION -a CI.ADO/STATIONS -s 100.0 -l -20/235 -t 20/50 -x T020
_T050 ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd >>CI.ADO/process.log
synmin=`saclst depmin f ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd.sac.T020_T050
|awk '{print $2}'`
synmax=`saclst depmax f ../solver/M01.set1/CI.ADO/OUTPUT_FILES/CI.ALP.HXZ.fwd.semd.sac.T020_T050
|awk '{print $2}'`
norm=`echo $synmin $synmax |awk '{if($1*$1>$2*$2) {print sqrt($1*$1);} else {print sqrt($2*$2)}}'
`
perl process_data.pl -m CI.ADO/CMTSOLUTION -s 100.0 -l -20/235 -t 20/50 -v -n -A $norm -x T020_T0
50 CI.ADO/DATA_NORM/CI.ALP.HXZ.sac >>CI.ADO/process.log
```

Figure 4: Command line of job1.sh in seis_process_M01.set1

```
cd CI.ADO
cat /dev/null >run.log
cp MEASUREMENT.WINDOWS.T005_T010 MEASUREMENT.WINDOWS
./run_measure_adj.pl M01.set1 -20/235 0 0 0 -2.0/0.01/24000 7 HX 5/10 0/0/0/1 -2.5/2.5/-1.0/1.0/0
.80 1/1.0/0.5 1/0.02/2.5/2.0/2.5/3.5/1.5 >>run.log
mv ADJOINT_SOURCES ADJOINT_SOURCES_T005_T010
mv window_chi ../../output/misfits/M01.set1_T005_T010_CI.ADO_window_chi
cp MEASUREMENT.WINDOWS.T010_T020 MEASUREMENT.WINDOWS
./run_measure_adj.pl M01.set1 -20/235 0 0 0 -2.0/0.01/24000 7 HX 10/20 0/0/0/1 -3.5/3.5/-1.0/1.0/
0.80 1/1.0/0.5 1/0.02/2.5/2.0/2.5/3.5/1.5 >>run.log
mv ADJOINT_SOURCES ADJOINT_SOURCES_T010_T020
mv window_chi ../../output/misfits/M01.set1_T010_T020_CI.ADO_window_chi
cp MEASUREMENT.WINDOWS.T020_T050 MEASUREMENT.WINDOWS
./run_measure_adj.pl M01.set1 -20/235 0 0 0 -2.0/0.01/24000 7 HX 20/50 0/0/0/1 -4.5/4.5/-1.0/1.0/
0.80 1/1.0/0.5 1/0.02/2.5/2.0/2.5/3.5/1.5 >>run.log
mv ADJOINT_SOURCES ADJOINT_SOURCES_T020_T050
mv window_chi ../../output/misfits/M01.set1_T020_T050_CI.ADO_window_chi
./combine_3_adj_src.pl HX ADJOINT_SOURCES_T005_T010 ADJOINT_SOURCES_T010_T020 ADJOINT_SOURCES_T02
0_T050 ADJOINT_SOURCES iker07 iker07 iker07 >>run.log
cd ../../
cdir=`pwd`
cd solver/M01.set1/CI.ADO
rm -rf SEM
mkdir -p SEM

cd SEM
for meas_adj in `ls $cdir/measure_adj_M01.set1/CI.ADO/ADJOINT_SOURCES/*.adj`;do
    adj=`echo $meas_adj |awk -F$cdir/measure_adj_M01.set1/CI.ADO/ADJOINT_SOURCES/ '{print $2}'`
    stnm=`echo $adj |awk -F. '{print $1}'`
    net=`echo $adj |awk -F. '{print $2}'`
    ch=`echo $adj |awk -F. '{print $3}'`
    adj_new=$net.$stnm.$ch.adj
    mv $meas_adj $adj_new
    cat $cdir/measure_adj_M01.set1/CI.ADO/ADJOINT_SOURCES/STATIONS_ADJOINT |sed -n '2,$p' >../DAT
A/STATIONS_ADJOINT
done
cd .. # done of make adj
cd $cdir
```

Figure 5: Command line of job1.sh in measure_adj_M01.set1.