Slab2 – Code Documentation

This document lists the procedure for accomplishing every step in the Slab2 modeling process using a MacOS. These steps have not been tested on Windows or Linux. Begin with the installation section. If creating a slab model from the data available in the Slab2 repository, follow the instructions for Creating an Input File and Creating a Slab Model. Optional steps for plotting results, calculating seismogenic width, and adding data to the current Slab2 dataset are also provided. To bypass installing and running the code in this repository, please visit the USGS Slab Model Cloud-based web application, found here.

Contents

Steps for Creating a Slab Model

Installation

1) If a version of Anaconda is not already installed, install the latest version of Anaconda. Instructions for installation can be found at https://www.anaconda.com/download/#macos. These instructions also work with miniconda (https://conda.io/miniconda.html latest version on MacOSX)
2) Clone slab2 from GitHub. Move the Slab2 repository to your preferred location on your local machine, referred to here as [Slab2_location].
3) Navigate to [Slab2_location]/slab2setup.
4) Now, we will create a virtual environment. In the terminal (using a bash Unix shell) enter: *bash slab2env.sh*
    a) This will take about 5-10 minutes to run depending on internet speed
    b) Once it has finished running, check to see if any error messages occurred during the run.
    c) If any error messages occurred, the environment just built must be removed, and pip and conda should be updated:
        i) Enter: *conda env remove -n slab2env # this removes the environment*
        ii) Enter: *y*
        iii) Enter: *conda update conda # updates conda*
        iv) Enter: *pip install --upgrade pip # updates pip*
        v) Enter: *bash slab2env.sh*
    d) If no error messages occurred, proceed
5) This created a virtual environment for running Slab2 code in Python 3.7 with all the required packages and libraries.
6) To activate the environment:
    a) In the terminal enter: *conda activate slab2env*
    Note: You must be in this environment in order to run any of the slab2 code.
    b) If you receive an error when typing *conda activate slab2env*, you will need to add the path to anaconda to your *.bash_profile* and initialize anaconda following the steps below:
        i) In your terminal, go to home and type vim *.bash_profile* and add *export PATH=/home/your_username/anaconda3/bin:$PATH*
        ii) In the terminal type source ~/*.bash_profile*
        iii) Open a new terminal and type *conda init bash*
        iv) Open a new terminal and now type *conda activate slab2env*
        v) If this still does not work, use *source activate slab2env*
7) To deactivate the environment:
    a) In the terminal enter: *conda deactivate*

Synthetic Test

We have created a synthetic data set (exp_04-18_input.csv) and have placed it in the [Slab2_location]/slab2code/Inp*ut* directory. A companion parameter file also exists at [Slab2_location]/slab2code/ library/parameterfiles/expinput.par. This can be used to test that the code is working properly before running further models. The figures below show the synthetic data and the model that the code should produce, following the steps in **Creating a Slab Model.**

The input data and model shown below were created with GMT tools (Project, Blockmean, Surface) using *makesynth.pl* in [Slab2_location]/slab2code/SYNTH. The input model (used as a reference surface for the Slab2 search) includes only the projected input data shown below, while the Slab2 code by default also uses an Average Active Source profile, and so differs slightly from the input model in its shallowest extent.

The output created via this process is included in [Slab2_location]/slab2code/Output/exp_slab2_06.14.18. The user should be able to reproduce these files by running the *slab2.py* code for *exp* following the steps in **Creating a Slab Model.** Once the new *exp* model is created, the user can compare the new *exp* output files with the *exp* output in Output/exp_slab2_06.14.18. For example, the GMT command grdinfo can be used on each file to compare the min and max values from the output grd files. Plots comparing the output can also be made. The user can use *plotmap.csh* and *plotxsec.csh* located in [Slab2_location]/slab2code/SYNTH for making plots using GMT. Be sure to move to the new *exp* output directory ([Slab2_location]/slab2code/Output/exp_slab2_MM.DD.YR) for plotting those results.

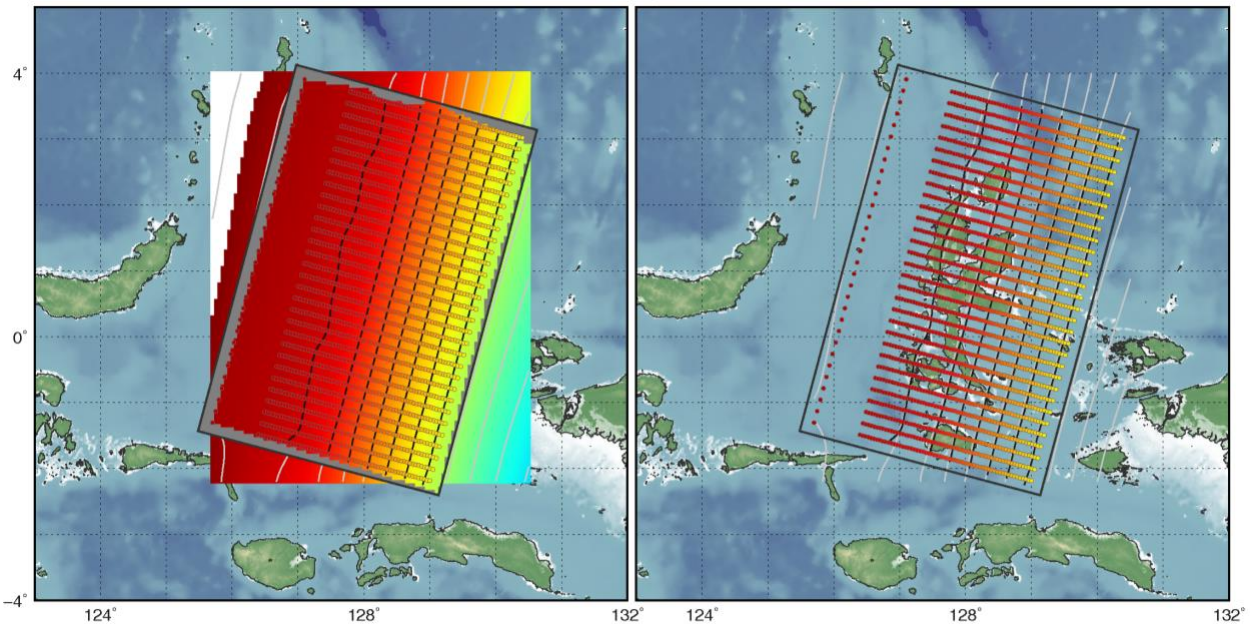

Figure 1: Left - synthetic data (circles colored by depth) and input surface (rectangular colored grid in the background), plotted versus the recovered Slab2 surface (clipped grid within black rectangle). Right - input data (circles colored by depth) plotted versus the contours of the input

data surface (gray) and the recovered Slab2 surface (black). Note the input data surface includes only data shown here, while the Slab2 surface also uses the global average active source profile (see main manuscript for details). Thus, the two surfaces differ in the shallowest trench region between the trench and first input data points shown here.

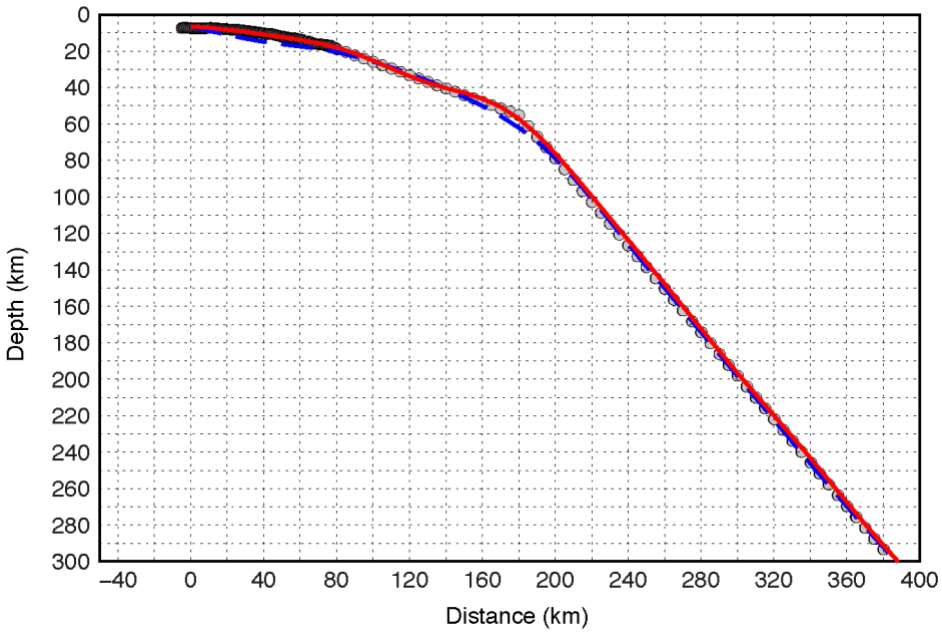

Figure 2: Cross section of the synthetic data (gray circles), the input model (blue dashed line) and the recovered Slab2 model (red line).

Creating an Input File

Input files are required for creating a Slab2 model. The input file can be created using any of the existing databases within the Slab2 repository (e.g., 0418database, 1219database, 1120database) or, if new seismic data is being added, then a new database should be created. If creating an input file from an existing database, then proceed to step 2 below. If adding new seismic data, then start at step 1. See section Adding to an Additional Catalog Query for adding in new catalog data.

1) To create a new database with new seismic data, create a directory located in [Slab2_location]. See part a below for how to name the directory. Copy over any existing data either from an existing database (e.g., 0418database, 1219database, 1120database) or from within MasterDB. Please note that hypocenter information from the GCMT and ComCat catalogs are required whereas other seismic data sets (e.g., active source, receiver functions, tomography) are optional.
    a) A good name for the new directory is MMYYdatabase, where MMYY is the month and year of the most recent query in the PDE. MM and YY can be identified by the name of the file (ALL_EQ_MMDDYY.csv) located in [Slab2_location]/MasterDB/gcmt_pde_assc.

b) Note: the Slab2 team aims to keep the database up to date, so there are existing MMYYdatabase folders in the slab2 repository.
    i) The 0418database folder contains the data that was used to create the published slab2 models from Hayes et al., 2018.
    ii) Any other MMYYdatabase folders contains data from the GCMT and ComCat catalogs up until MMYY.
    iii) Within MaterDB there is a directory called PublishedSlab2DB. This also contains the catalog data used in the published Slab2 models.

2) Navigate to [Slab2_location]/slab2code in the terminal and activate the environment: *conda activate slab2env*
3) For example, if creating an input file for Alaska (*-p alu*), with a database and earthquake catalog that was queried on Dec 2017 (*-d [Slab2_location]/1217database*) and saving the file using the appropriate naming convention (*-f alu_12-17_input.csv*):
    a) In the terminal enter: *python s2d.py -p alu -d [Slab2_location]/1217database -f alu_12-17_input.csv*
    b) This will take about 5 minutes to run for most regions.
    c) In the terminal enter: *mv alu_12-17_input.csv Input*
4) Note that the input file and output name must be in format [slab]_[MM]-[YY]_input.csv where [slab] is the three-letter slab code, [MM] is the month of the associated database, and [YY] is the year of the associated database.
5) For more help and descriptions, see documentation within *s2d.py* and *s2d_fnctns.py*. For example, entering *python s2d.py -h* in the terminal will give more details on the running options and required arguments.
6) To make input for all slab2 slab regions, in the terminal type *python makeallinputs.py -d FilePathToDatabase -f MM-YR -c No.OfCores*
    a) FilePathToDatabase is the database you want to make the input from, such as *0418database*
    b) MMYR is the two-digit month and two-digit year of the database
    c) Use the -c NoOfCores option to run the code in parallel, where NoOfCores is the number of core to run on.
    d) Afterwards move all the input files to the Input directory:
        i) Type: *mv *_input.csv Input/*

Creating a Slab Model

1) Navigate to [Slab2_location]/slab2code in the terminal and activate the slab2 environment: *conda activate slab2env*
2) If generating a complete model of the Aleutians slab using the input file with the most recent date and year (stored in [Slab2_location]/slab2code/Input), retaining all other predetermined parameters:
    a) In the terminal enter: *python slab2.py -p library/parameterfiles/aluinput.par*
3) Regions *mue, sul, phi,* and *ryu* are truncated by other slabs and require the depth grid from the other slab to be listed in the command line argument when running *slab2.py*. The slab model for the other slab must be made first. The *mue* region depends on the *car* region, *sul* and *phi* depend on *hal*, and *ryu* depends on *kur*. For example, for making a

model for the *mue* region, first make the model for the *car* region following the instructions above in 2a, and then use,

a) *python slab2.py -p library/parameterfiles/mueinput.par -u Output/car_slab2_09.19.19/car_slab2_dep_09.19.19.grd*

4) These models can also be run in parallel. To run the example in 2a using three cores:
    a) In the terminal enter: *python slab2.py -p library/parameterfiles/aluinput.par -c 3*
    b) This can take anywhere from 3 minutes to 3 hours to complete, depending on several variables including: how many cores are used, the grid resolution/size, the surface resolution/size, any other loads your system is under, if there is a vertical component to the slab, etc.
5) To change any parameters, navigate to [Slab2_location]/slab2code/library/parameterfiles, and edit the desired slab parameter file. It is important to maintain the same formatting.
    a) Parameters that one may want to change include:
        i) radius1 - long axis of search ellipsoid (km)
        ii) radius2 - short axis of search ellipsoid (km)
        iii) taper - depth over which to taper from no shift to full shift (km)
        iv) filt - width of smoothing filter (degrees)
        v) fracS - fraction of slab thickness to shift
        vi) grid - grid node resolution (degrees). Can range between 0.1 and 0.5.
        vii) minstk - maximum range of reference model strikes in search ellipsoid (degrees)
        viii) node - grid resolution of least squares interpolation (degrees)
    b) If changing any parameters, it would be useful to use the find command in the text editor that you are using in *slab2.py*, s*lab2_functions.py*, and *loops.py* for the string "if slab == '[slab]'" where [slab] is the three-letter string associated with this region. These statements currently override any parameters in the parameter file and are necessary to take in to consideration when modifying parameters. These are in place for:
        i) Sections within slabs that do not fit the general parameters for that region,
        ii) Regions that do not fit the general modeling approach at a specific step in the algorithm.
6) To plot general maps and cross sections, see **Creating Generalized Maps and Cross-sections**.
7) Once a general set of parameters is defined, see **Determining Shift Magnitude** to find the optimum shift magnitude parameter. This is found from comparing the width of the event distribution with the inferred slab thickness at each node.
8) After determining the parameters that make the post shifted nodes behave as desired, proceed to steps listed in **Running L-Curve for a Model** to determine the best width for the gaussian smoothing filter.
9) Several slabs can be generated concurrently, but only one can be generated in parallel at any given time.
10) The interpolation of the larger slabs (sum, alu, sam, kur) and medium sized slabs with vertical components (ker, izu) is very memory intensive, and will consume all RAM if several of them are run at the same time. It is fine to run these concurrently overnight (up

to 4 of them) but this is not recommended if the machine needs to be used for anything else while they are running (e.g. in the middle of the workday).

11) For more help and descriptions, see documentation within *slab2.py* and *slab2functions.py*. Entering: *python slab2.py -h* will give more details on running options and required arguments.

Optional Steps

Creating Generalized Maps and Cross-sections

1) Navigate to [Slab2_location]/slab2code/plotting
2) Find the python scripts named *map.py* for making generalized maps, and *xsec.py* for creating cross-sections
3) To make use of these python scripts, the *pygmt* environment must be activated.
    a. In the terminal, enter *conda activate pygmt*
4) In the pygmt environment, run *map.py* to make general plots of the slab depth, dip, thickness, strike, and uncertainty.
    a. There are 2 required command line arguments
        i. 3 letter slab code (e.g., alu)
        ii. Date of model output in the form MM.DD.YY (e.g., 06.14.22)
    b. --color_blind is an optional command line argument that will create the maps with a color-blind friendly color palette from Crameri et al. 2020.
    c. To make 2D maps of the Aleutian slab model made on 06.14.22 with a color-blind friendly palette, use the following command.
        i. *python map.py alu 06.14.22 --color_blind*
    d. The maps will be stored in [Slab2_location]/slab2code/Outputs/[slab]_slab2_[date]/[slab]_[date]_maps

5) In the pygmt environment, run *xsec.py* to make general cross section plots
    a. There are 4 required positional command line arguments
        i. 3 letter slab code (e.g., izu)
        ii. Date of model output in the form MM.DD.YY (e.g., 06.17.22)
        iii. The date corresponding to the database used in the form MM-YY (e.g. 12-19)
            1. Note: The input file MUST be placed in [Slab2_location]/slab2code/Input/[Database Date] and must be named as [slab]_[database-date]_input.csv

iv. The last required argument, N, can take a variety of forms
    1. Integer
        a. If making a single plot, this will correspond to the index of the trench file
        b. If making multiple plots, this will correspond to the total number of plots to make, evenly spaced
    2. ‘all’
        a. If specified, the indices of the trench file that are spaced 5 apart will be created
    3. List of ‘[lon,lat,az]’
        a. This will make a cross section at a specific longitude, latitude, and azimuth
        b. Lon may be specified as an angle from 0-360, or a negative value for degrees west
        c. Lat values must be negative for degrees South
        d. Note: this may only be used for single plots. This argument MUST NOT contain spaces
    4. Note: N will be used for the naming of the output file for single plots

b. Optional arguments include the following:
    i. The width of the cross section in km, with the line at the center of the width. If specified, this argument MUST come DIRECTLY after N
    ii. --multiple
        1. If specified and N is an integer, N will become the number of cross sections
        2. If specified and N is ‘all’, every fifth index in the trench file will have a cross section made
    iii. --on_map
        If specified, the input data that is shown on the cross-section plot will also be plotted on the overview map
    iv. --focal
        If specified, focal mechanisms will be plotted on an adjusted cross-sectional plot, where the axes are equalized
    v. --color_blind
        If specified, a color-blind friendly palette will be used for the plot

c. To create 100 equally spaced cross section plots of the alu slab model created on 06.14.22 with the 12-19 input database with widths of 50km, input data on the overview map, and focal mechanisms, use the following command:
    i. *python map.py alu 06.14.22 12-19 100 50 --on_map --multiple --focal*

d. The plots will be stored in [Slab2_location]/slab2code/Outputs/[slab]_slab2_[date]/[slab]_[date]_focal_xsecs
    i. Note: If focal mechanisms are specified, the plots will be stored in a separate directory from plots without focal mechanisms, noted by the inclusion of 'focal' in the directory name.

6) Notes
    a. *xsec.py* script depends on sea trench strike values to create cross sections. Because slabs hal, hin, and pam do not have sea trenches, the only way to create cross section plots for these slabs is with [lon/lat/az] values inputted for N.

    b. The *izu*, *ker*, *man*, and *sol* slabs have complex geometries, with parts of the slab curling under itself, referred in these models as the "tilted" region of the slab. When making cross sections of these slabs, *xsec.py* will remove ten data points from the end of the 'normal region'. The code then interpolates ten new data points and appends them to the end of the normal region. The tilted region is then appended and plotted. This is done in order to smooth the plot of the cross section.

    c. For more help and descriptions, see documentation within *map.py* and *xsec.py*. *python map.py -h* or *python xsec.py -h* produces more details on running options and required arguments.

Convert grd files to Shapefile and KML Formats

- If desired, one can convert the slab2 grd files to contours in Shakefile and KML formats that can be used in GoogleEarth and GIS software

1) Navtigate to Navigate to [Slab2_location]/slab2code/library/convert and activate the slab2 environment: *conda activate slab2env*
2) The bash script MakeShapefileKML.sh can be used for converting slab2 output grd files to contours in Shapefile and KML formats
3) To convert grd file for one slab2 region, in the terminal type
    a) bash MakeShapefileKML.sh Region Data Date ContourInterval
    b) Example: *bash MakeShapefileKML.sh alu dep 04.08.22 20*
        i) Region is the three-letter acronym of the slab2 region
        ii) Data is the three-letter acronym for the type of output data to convert
            (1) depth (dep)
            (2) dip (dip)
            (3) strike (str)
            (4) uncertainty (unc)
            (5) thickness (thk)
        iii) Date is the date the slab model was generated (e.g., 04.08.22). This date is included in the name of the file (alu_slab2_dep_04.08.22.grd)

iv) ContourInterval is the contour interval in km

4) To convert grd file for all 27 slab2 regions, in the terminal type
   a) *bash MakeShapefileKML.sh all Data Date ContourInterval*
   b) Example: *bash MakeShapefileKML.sh all dep 04.08.22 20*
      i) Data is the three-letter acronym for the type of output data to convert
         (1) depth (dep)
         (2) dip (dip)
         (3) strike (str)
         (4) uncertainty (unc)
         (5) thickness (thk)
      ii) Date is the date the slab model was generated (e.g., 04.08.22). This date is included in the name of the file (alu_slab2_dep_04.08.22.grd)
      iii) ContourInterval is the contour interval in km

Calculating Seismogenic Zone Width and Organizing Files

- *sztcalc.py* takes a depth grid and an unfiltered Slab2 input dataset and calculates the width of the seismogenic zone based on a distribution of depths of slab related events. The events are deemed slab related based on a series of filters, using spatial and Kagan's angle information.
- This tool can be used to calculate the sz width for a model within a discrete bounding box or in an area bounded by a mask or polygon.
- The filelist within *sztcalc.py* must first be updated by the user prior to running the code.
  a) In *sztcalc.py*, search for "filelist = ["
  b) Change the contents inside the brackets of this line to the list of new models to calculate the seismogenic zone width for. Follow syntax and naming convention in previously existing list.
- The required inputs include:
  a) Input file information *(-[flag]: description of input for [flag])*
     i) -s: The path to the depth grid
     ii) -r: The model name ([slab]_slab2[MM.DD.YY])
     iii) -i: The directory holding the input file
     iv) -t: The input file date (MM-YY)
     v) -f: The folder to save these files to
  b) Filter information:
     i) -l: Flag indicating origin or CMT lon, lat to compare with the slab (o or c)
     ii) -d: Flag indicating origin or CMT depths to compare with the slab (o or c)
     iii) -b: Flag indicating a distribution of event depths or slab depths (e or s)
     iv) -m: A depth filter to exclude points outside of in szt calculations
     v) -x: A depth extent to cut distributions off at
- Optional inputs include:
  a) -o: lonmin, lonmax, latmin, latmax
  b) -k: path to a clipping mask. Mask must have unlabled columns of lon lat, separated by white space.

5) Navigate to [Slab2_location]/slab2code and activate the slab2 environment: *conda activate slab2env*
6) To move a list of slab models contained in [Slab2_location]/Output to a new file structure in directory [newdir] and calculate the seismogenic zone width using input files stored in [Slab2_location]/Input:
    a) In the terminal enter: *python sztcalc.py -s [path to depth grid] -l [origin or cmt lat,lon] -d [origin or cmt depths] -b [slab or event depth histogram] -m [maxdepdiff] -x [distcutoff] -i Input -f [newdir] -t [MM-YY] -r [model]*
        i) *[path to depth grid]* is the path to the slab model
        ii) [origin or cmt lat,lon] can be "c" or "o"
            (1) "c" indicates to use the CMT lon, lat in spatial filters
            (2) "o" indicates to use the origin lon, lat in spatial filters
        iii) [origin or cmt depths] can be "c" or "o"
            (1) "c" indicates to use the CMT depth in spatial filters
            (2) "o" indicates to use the origin depth in spatial filters
        iv) [slab or event depth histogram] can be "e" or "s"
            (1) "e" indicates to make a histogram of event depths within the Kagan's angle and spatial filters. This will be a histogram of origin or CMT depths depending on the -d flag
            (2) "s" indicates to make a histogram of the slab depths queried at event lon, lat locations within the Kagan's angle and spatial filters. The slab depth will be queried at the CMT or origin lon, lat coordinates depending on the -l flag.
        v) [maxdepdiff] is the width of the spatial filter around the slab model in km.
        vi) [distcutoff] is the depth in kilometers that the histogram is cut off at
        vii) *-i Input* indicates that the input files being used to generate the depth histograms are stored in the Input directory
        viii) [newdir] is the folder where the new file structure and seismogenic zone thickness calculations will be saved
        ix) [MM-YY] is the date of the input files to use. Input file naming convention is [slab]_[MM-YY]_input.csv.
        x) [model] is in the form [slab]_slab2_[MM.DD.YY]
7) For example. to calculate the seismogenic zone width of alu_slab2_01.08.18 using event origins (*-l o*) and centroid MT depths (*-d c -b e*) with a near slab filter of 20 km depth (*-m 20*) and a distribution cutoff of 65 km depth (*-x 65*) using input files with date December 2017 (*-i Input -t 12-17*) and save them to ~/Desktop/alutest:
    c) In the terminal enter: *python sztcalc.py -s Output -f ~/Desktop/alutest -i Input -t 12-17 -l o -d c -m 20 -x 65 -b e*
    d) A plot showing a histogram of slab related event depths and a best fitting double normal distribution will be plotted for the model in *~/Desktop/alutest*.
    e) The file containing all events within specified Kagan's Angle and spatial filters will be saved in the same folder.
    f) A table listing the upper and lower bounds of the seismogenic zone and the number of events to constrain those calculations will be written to *~/Desktop/alutest/alu_slab2_szt_01.08.18.csv.*

- To do the same as above, but bounded by a mask stored in *~/Desktop/alutest/alu_west_mask.txt:*
    a) In the terminal enter: *python sztcalc.py -s Output -f ~/Desktop/alutest -i Input -t 12-17 -l o -d c -m 20 -x 65 -b e -k ~/Desktop/alutest/alu_west_mask.txt*
    b) The histogram and kagan files that were saved here previously will be deleted if they were made for the same model. New tables indicating the final calculations, however, will be saved with different file names.
- To do the same as above, but with a bounding box instead of a mask with lonmin=160, lonmax=195, latmin=45, latmax=70:
    a) In the terminal enter: *python sztcalc.py -s Output -f ~/Desktop/alutest -i Input -t 12-17 -l o -d c -m 20 -x 65 -b e -b 160 195 45 70*
    b) The histogram and kagan files that were saved here previously will be deleted if they were made for the same model. New tables indicating the final calculations, however, will be saved with different file names.
- To organize the slab model output by output file type, rather than by slab model *sztcalc.py* can also be used.

8) Move all original model output folders to a common folder (if they are not already in a common folder). It is easiest to keep them in the original [Slab2_location]/slab2code/Output folder that they are written to from slab2.py.
9) Make a new directory [newdir] to save the new file structure and seismogenic zone thickness calculations to
10) To move a list of slab models contained in [Slab2_location]/slab2code/Output to the new file structure in directory [newdir] and calculate the seismogenic zone width using input files stored in [Slab2_location]/slab2code/Input:
    a) In the terminal enter: *python sztcalc.py -s Output -l [origin or cmt lat,lon] -d [origin or cmt depths] -b [slab or event depth histogram] -m [maxdepdiff] -x [distcutoff] -i Input -f [newdir] -t [MM-YY]*
        i) *-s Output* indicates that all of the models are in the [Slab2_location]/slab2code/Output directory.
        ii) [origin or cmt lat,lon] can be “c” or “o”
            (1) “c” indicates to use the CMT lon,lat in spatial filters
            (2) “o” indicates to use the origin lon,lat in spatial filters
        iii) And [origin or cmt depths] can be “c” or “o”
            (1) “c” indicates to use the CMT depth in spatial filters
            (2) “o” indicates to use the origin depth in spatial filters
        iv) And [slab or event depth histogram] can be “e” or “s”
            (1) “e” indicates to make a histogram of event depths within the Kagan’s angle and spatial filters. This will be a histogram of origin or CMT depths depending on the -d flag
            (2) “s” indicates to make a histogram of the slab depths queried at event lon,lat locations within the Kagan’s angle and spatial filters. The slab depth will be queried at the CMT or origin lon, lat coordinates depending on the -l flag.
        v) [maxdepdiff] is the width of the spatial filter around the slab model in km.
        vi) [distcutoff] is the depth in kilometers that the histogram is cut off at

vii) And *-i Input* indicates that the input files being used to generate the depth histograms are stored in the Input directory
viii) [newdir] is the folder where the new file structure and seismogenic zone thickness calculations will be saved
ix) [MM-YY] is the date of the input files to use. Input file naming convention is [slab]_[MM-YY]_input.csv.

b) If there are 27 slab models, the script should take less than a minute to complete.
c) A new file structure will be saved in [newdir] with folders:
i) clippingmasks: contains the polygons bounding each slab grid
ii) filtereddata: contains the list of data points that were used in constraining the slab model
iii) Grids: contains the grid formats of the slab geometry, thickness, and uncertainty. (including: depth, strike, dip, thickness, and uncertainty)
iv) inputdata: the original unfiltered input files for *slab2.py*
v) nodes: contains the lists of all original grid node locations that were used to constrain the final slab surfaces. Important information regarding filter dimensions and pre/post shifted locations is listed for each point.
vi) parameters: contains metadadata files listing the parameters that were used in creating this model.
vii) Supplement: contains ascii files representing the vertical component of the slab (for slab regions where it exists) and the associated geometry, thickness, and uncertainty information.
viii) Surfacetext: contains the ascii file versions of the geometry grids.
ix) Szt: the seismogenic thickness histograms for each new slab model will be saved in this directory, along with a general table and distribution plot.
x) Contours: contains the contour text files for vertical slab sections (where they exist).
xi) figures: contains simple postscript files showing the geometry grids, filtered and unfiltered datasets, and post shifted node locations (if the figures were generated prior to running this).
xii) crossections: contains cross sections showing the slab model and associated filtered and unfiltered datasets (if they were generated prior to running this).

11) To calculate the seismogenic zone width of a list of models inside of [Slab2_location]/slab2code/Output (*-s Output*) using event origins (*-l o*) and centroid MT depths (*-d c -b e*) with a near slab filter of 20 km depth (*-m 20*) and a distribution cutoff of 65 km depth (*-x 65*) using input files with date December 2017 (*-i Input -t 12-17*) and save them to [newdir] created in step 2 (*-f [newdir]*):
a) In the terminal enter: *python sztcalc.py -s Output -f [newdir] -i Input -t 12-17 -l o -d c -m 20 -x 65 -b e*
b) A plot showing a histogram of slab related event depths and a best fitting double normal distribution will be plotted for each slab model in [newdir]/szt.
c) The file for each slab containing all events within specified Kagan's Angle and spatial filters will be saved in the folder [newdir]/szt.
d) A plot of all of the summed double normal distributions will be saved as [newdir]/szt/allpdf.png.

e) A table listing the upper and lower bounds of the seismogenic zone and the number of events to constrain those calculations will be written to [newdir]/szt/table.csv.

- This script requires that all models have an original unfiltered input file from the same date, so multiple runs might be necessary to get around this if all input files were not made from the same database version.
- For more help and descriptions, see documentation within *sztcalc.py* and *slab2functions.py*. Entering: *python sztcalc.py -h* will give more details on running options and required arguments.

Setup Bathymetry and Sediment Thickness Datasets

- If you would like to include new bathymetry and/or sediment thickness datasets, then proceed with the following steps and then jump to the section **Making a Bathymetry File**.

1) Navigate to [Slab2_location]/misc/bathymetry.
2) Activate environment: *conda activate slab2env*
3) Convert coarse sediment thickness data to a CSV file:
    a. gmt grd2xyz sedmap.grd | awk '{if(NR==1) print "lon,lat,thickness\n"$1","$2","$3; else print $1","$2","$3}' >sedmap.csv
4) Unzip and convert dense sediment thickness data to a CSV file:
    a. Gunzip sedthick.xyz.gz, then
    b. awk '{if(NR==1) print "lon,lat,thickness\n"$1","$2","$3; else print $1","$2","$3}' sedthick.xyz > sedthick2-2.csv
5) Download GEBCO bathymetry data (https://www.gebco.net/). We used the one min gird (https://www.gebco.net/data_and_products/gridded_bathymetry_data/gebco_one_minute_grid/). The GEBCO_2019 global grid is now available in a NetCDF file format. If you would like to use that, please convert from NetCDF to CSV. Else, use the included 5minx5min grid (gebco5x5.grd) and skip (a) below.
    a. Downsample resulting grid to 5x5 minute data.
    b. Convert the grd filed into a CSV file (gebco5x5.csv) using,
        gmt grd2xyz gebco5x5.grd | awk '{if(NR==1) print "lon,lat,elev\n"$1","$2","$3; else print $1","$2","$3}' >gebco5x5.csv
6) Run *mergest.py* by typing *python mergetest.py* in the terminal. This will create the file totalBA.csv, which combines bathymetry and sediment thickness datasets.
7) Follow the steps in **Making a Bathymetry File** to update the existing bathymetry files or add a file if one does not exist for a specific subduction zone (check [Slab2_location]/MasterDB/bathymetry).

Making a Bathymetry File

- Before making a new bathymetry file, follow the steps in **Set Up Bathymetry and Sediment Thickness Datasets** to first generate the file totalBA.csv.

1) Navigate to [Slab2_location]/misc/bathymetry and activate the slab2 environment: *conda activate slab2env*
2) Open *newbathymetry.py* in text editor of choice
3) Look for line with "edit next two lines to create bathymetry files"
4) In the lines between the commented strings, edit the list of slabs to create bathymetry files for and name path to trench file that will be used to constrain the search.
    a) The trench file must have columns: lon, lat, az, bound, slab
        i) If the trench file is not already made, it would be good to first generate a file in this format using the steps/tools listed in **Adding or Modifying a Trench File**
    b) The trench file can have any number of other columns, and the order does not matter
    c) Every slab in the slablist in *newbathymetry.py* must be listed somewhere in the slab column in the trench file
5) To create a new bathymetry file for every slab in the slab list, in the terminal enter: *python newbathymetry.py*
6) Depending on how many bathymetry files are being made, how densely spaced the trench file is, and how long the trench is for each slab in the slablist the script will finish in ~1-20 minutes
7) The output files will be written to [Slab2_location]/misc/bathymetry/[slab]_BA_bath.csv
    a) Where [slab] is the three-letter slab code associated with each region in the slablist.
8) Move the new files to the bathymetry folder in the master database ([Slab2_location]/masterDB/bathymetry)
9) Create a new input file for this slab following steps in **Creating an input file**.

Adding or Modifying a Trench File

- When adding a new slab to the database, if the subduction in this region initiates at an oceanic trench, a set of points delineating the trench segment must be acquired then processed to have the format, spacing, and details required by *slab2.py*.
- For consistency, all trench segments should be made with the tools outlined in this section.
- The first three steps discuss setting up trench sections. If a set of points delineating the trench already exists, proceed to step 4.

1. Check the USGS trench file (found in [Slab_location]/slab2code/library/forplotting/trenches_usgs_2017.csv) to see if the slab region has a segment (lon, lat coordinates that represent the trench location). Citations used for specifying each trench location is found in slab2/misc/trenches/citations.txt.
2. If the USGS file does not have a segment for this slab, search the literature for a segment. A good place to start this search is with Bird, 2003. To do this, go to http://peterbird.name/publications/2003_PB2002/2003_PB2002.htm to see if a plate boundary exists in this database over the margin associated with this region. If one exists, go to http://peterbird.name/oldFTP/PB2002/PB2002_boundaries.dig.txt and identify the segment(s) that delineate the trench for this slab model.

3. As a last resort, if a trench cannot be identified, digitize the trench based on topography using an appropriately colored and labeled map.
    a. If the segment comes from a source not listed in [Slab2_location]/misc/trenches/citations.txt, add this citation to that list. Include the slab code name and the trench segment source.
4. Convert the list of lon, lat points to a CSV file, with a third column listing the three-letter slab code in every row, and a fourth column listing the plates on either side of the trench in every row.
    a. For example, the first two rows in the alu trench segment might be:

        lon,lat,slab,bound
        167.1909,54.0207,alu,PA/NA
        167.4184,53.9208,alu,PA/NA
5. Label the columns of the file lon, lat, slab, bound. Other columns may exist but will be discarded throughout the initial sorting process.
6. Make sure that the last point in the file is the point that you want to start sorting from according to the right-hand rule. In other words:
    a. Picture yourself walking along the length of the trench
    b. If the slab would be subducting to your right for the whole walk, which point on the trench would you start from? Call this [lon1,lat1].
    c. Now make sure that the row containing [lon1,lat1] is listed as the last row of the file.
    d. For example, the last point in the south America trench should be the most southern point, for the Aleutians, the most eastern point, for Kermadec, the most northern point, for Papua New Guinea, the most western point.
    e. Don't worry about sorting the rest of the trench data. This will be done using the tools listed in the next few steps.
    f. Many trenches can be made at once. Just list the segments one after another in the same CSV file with the row organization in step (4). No special marker between each trench segment is necessary.
    g. For example:

        Lon,lat,slab,bound
        -137.9525,59.0277,alu,PA/NA
        -137.819,58.9251,alu,PA/NA
        -26.071,-60.391,sco,SA/SC
        -25.8205,-60.2865,sco,SA/SC
    h. Make sure that the last row for each slab follows the last row rule listed in step (6).
7. Save the file to a logical place, and don't forget to document the source(s) in [Slab2_location]/misc/trenches/citations.txt
8. Navigate to [Slab2_location]/misc/trenches
9. Use *sortpoints.py* to sort the list of points according to the right-hand rule, and calculate the azimuth between consecutive points:
    a. In the terminal enter: *conda activate slab2env*
    b. Enter: *python sortpoints.py -n [newtrenchlist] -t [trenchfile]*
        i. [newtrenchlist] is the file you just made listing the new trench points to sort and calculate azimuths for

ii. [trenchfile] is the output file to save the sorted trench segment to

10. After the trench segment(s) have been sorted and azimuths have been calculated for each point, use *trenchsmooth.py* to smooth over any outliers that may have slipped through the sorting process.
    a. In the terminal enter: *python trenchsmooth.py -n [newtrenchlist] -t [trenchfile] -s s*
        i. [newtrenchlist] is the file listing the sorted trench points with azimuth values to smooth
        ii. [trenchfile] is the output file to save the smoothed trench segment to.
        iii. *-s s* indicates that we are smoothing the trench segment, rather than filling in (filling in explained in next step).
11. After the trench segment(s) are smoothed with *trenchsmooth.py*, use *trenchsmooth.py* to make sure that the distance between each consecutive trench point is no more than 20 km. This is necessary for regions with very linear trenches that can be plotted with very few vertices. Often the list of points describing these straight trench segments are very far apart, which is bad for the purpose that these points serve in *slab2.py*.
12. *trenchsmooth.py* linearly interpolates between these far away points until a sufficient point density is attained.
    a. In the terminal enter: *python trenchsmooth.py -n [newtrenchlist] -t [trenchfile] -s f*
        i. [newtrenchlist] is the file listing the sorted trench points with azimuth values to smooth
        ii. [trenchfile] is the output file to save the new smoothed trench segment to.
        iii. *-s f* indicates that we are filling in the trench segment, rather than smoothing the azimuth values.
    b. The file listed as [trenchfile] is in the proper format to be added to the global trench dataset in the slab2code library. Identify the currently used trench file in the library located in [Slab2_location]/slab2code/library/misc/trenches_usgs_2017_depths.csv. If no trench information for this slab exists, add the new slab trench data to end of this file. If just updating pre-existing slab trench information, replace the new data with the old data for the specific slab trench segment.
13. To determine bathymetric depth along each trench segment,
    a. go to [Slab2_location]/slab2/misc/trenches and activate slab2 conda environment
    b. If a new trench file has been created, replace the currently used trench file (trenches_usgs_2017.csv) in this directory and, if needed, update the input file name/path on line 6 in maketrenchdepth.py
    c. If using a new bathymetry file, line 8 in maketrenchdepth.py should also be updated
    d. To extract depths from the bathymetry file along each trench location in the trench file, run *python maketrenchdepth.py* in your terminal
    e. This will generate a file called trenches_usgs_2017_depths.csv. To use this file in the generation of slab models, move the file to [Slab2_location]/slab2code/library/misc

Adding a Dataset

1) Reorganize any new dataset into a CSV file format with labeled columns. Document where the dataset came from and any modification details in masterdb.xlsx. Some things to note:
    a) Available list of column names is: time, lat, lon, depth, mag, mrr, mtt, mpp, mrt, mrp, mtp, type, mlon, mlat, mdep
    b) All columns with a name not listed in (1a) will be discarded
    c) The order of the columns is inconsequential
    d) The file at *minimum* must have labeled columns of lon, lat, depth
2) Add the new file to the appropriate folder in MasterDB. The database is organized by data type. The name of the file is important:
    a) File naming convention is [slab]_[data type]_[source ID].csv
    b) [slab] is the three-letter slab code (e.g. alu, sam, cam, etc.)
    c) [data type] is the two-letter abbreviation indicating the kind of slab related dataset. Available list of event types is: EQ (earthquake hypocenters), ER (relocated earthquake hypocenters), AS (active source), RF (receive function), BA (bathymetry), CP (control points), TO (tomography), SR (seismic reflection), ST (active source seismic tomography)
    d) Abbreviations of SR and ST are merged into the same event type as AS in the modeling algorithm.
    e) [source ID] can be any name referring to the source of the data.
3) If a common folder containing all datasets listed in the masterdb spreadsheet exists and is up-to-date, add this file to that folder, and proceed to step (3) in **Creating an Input File**. If this folder does not exist, proceed to step (1) in **Creating an Input File.**

Adding in a Tomography Dataset

- Regional or global teleseismic body wave tomography data can be used as input for creating slab models, specifically when there is a lack of earthquakes, or other seismic data, that are typically used for delineating slab geometry.
- The procedure involves searching the tomography model for the fast velocity anomaly associated with the center of the slab. Refer to Portner and Hayes, 2018 and the supporting information of Hayes et al., 2018 for further information on the methodology.
- Due to the large uncertainties associated with tomography modeling and with the approach used here for generating an irregularly spaced set of data points from the tomography data, tomography data points are given large error bars. Thus, tomography data points are given a high uncertainty in the Slab2 workflow and have low influence in the determination of slab geometry when other seismic datasets exist.
- Currently, tomography models are used for the *cas*, *hel*, *jap*, *kur*, *man*, *ryu*, *sam*, and *sum* regions. A user may wish to replace tomography models for these regions or to add tomography models for other regions.

1. Navigate to [Slab2_location]/ TomoSlab_v1_2018 and activate the slab2 environment: *conda activate slab2env*
2. Instructions for obtaining data points from a tomography model are provided in the README. You do not have to install any packages to run the tomography code. The code will run in *slab2env*.
3. The current code is set up for creating data points from the SAM4_P_2017 tomography model (Portner et al., 2017). To generate data points from another tomography model,
    a. Place your tomography model into [Slab2_location]/ TomoSlab_v1_2018/Tomo_files
        i. The column order should be latitude, longitude, depth, velocity (as %dV). Do not include a header.
        ii. Further information on the format for the input tomography data can be found in the supporting information of Portner and Hayes, 2018.
    b. Place the slab guide for the slab region into [Slab2_location]/ TomoSlab_v1_2018/Slab_files. Refer to the section in this documentation on **Making or Using a Slab Guide** for details.
    c. Open *tomo_slab.py* in your preferred editor
        i. Follow instructions in the README for updating the file paths to your tomography model and slab guide, and for updating input parameters.
    d. Once the input flags have been updated, in the terminal enter *python tomo_slab.py* to run the code on your tomography model.
        i. The data points will be saved in [Slab2_location]/ TomoSlab_v1_2018/Output
    e. Copy the output tomography data points to [Slab2_location]/MasterDB/tomography

Querying the Preliminary Determination of Epicenters (PDE) Bulletin

1) Navigate to [Slab2_location]/catalogquery and activate the slab2 environment: *conda activate slab2env*
2) Since the catalog takes an outrageous amount of time to query in its entirety, the querying script, *newcomcatquery.py*, prompts the user for an interval between 1 and 11 with the assumption that the user will run ten simultaneous queries and merge the files once all are finished. This drops the amount of time from > 2 weeks to ~15 hours. There is an option to just update the existing catalog as well, which is described in **Adding to and Existing Catalog Query.**
3) To query the whole catalog at the same time, open 11 terminal windows and repeat steps (1) and (2) in each. For a later step, it is important that these are new blank terminal windows.
4) In each terminal window, enter: *libcomcatquery.py -d [qdir] -i [intervalno]*
    a) [intervalno] is an integer between 1 and 11
    b) [qdir] is a folder name within this directory that all query files will be saved to
5) The entire query is designed to run overnight, and should take around 15 hours

6) After all queries are completed, save the printed terminal output for each window into a new directory. These files will be searched using *getfailedevents.py* for events that failed using the libcomcat API (due to random timeouts).
7) Once all terminal content is saved, run *getfailedevents.py*:
    a) In the terminal window enter: *python getfailedqueries.py -f [fdir] -q [qdir]*
        i) [fdir] is the directory where the terminal outputs were saved
        ii) [qdir] is the directory containing the output files of the original query.
    b) A new query of all failed searches will be saved in [qdir], which is used in the next step
8) In order to combine all of these queries and condense the amount of columns returned, use *concatcomcat.py* to concatenate all of the output files in [qdir]
    a) In the terminal window enter: *python concatcomcat.py*
    b) Enter *1*
    c) Enter *[qdir]*
    d) Wait ~ 5 minutes
    e) The concatenated and condensed file will be written to [qdir].csv
9) Once a comprehensive query has been attained, it should be associated with the most current version of GCMT for use in Slab2. See **Associating the PDE and GCMT Data** for next steps.

Adding to an Existing Catalog Query

1) Navigate to [Slab2_location]/Slab2_Code/catalogquery and activate the slab2 environment: *conda activate slab2env*
2) In the terminal window, enter: *python libcomcatquery.py -d [qdir] -i 100 -p [previousquery]*
    a) [qdir] is a folder name within this directory where all query files will be saved.
    b) [previousquery] is the file (e.g., pde_0618.csv) that we are adding more recent events to with this query. The script will take the date of the most recent event of this file then search from that date/time to present over the globe.
3) Wait for the query to finish. This will take longer for longer time spans.
4) Add this file to the existing query [previousquery].
    a) First, we need to convert the file to the condensed slab2 format. Use concatcomcat for this:
        i) In the terminal enter: *python concatcomcat.py*
        ii) Enter: *1*
        iii) Enter: *[qdir]*
    b) A file in the correct format will be written to [qdir].csv
    c) Now add [qdir].csv to [previousquery]:
        i) In the terminal enter: *python concatcomcat.py*
        ii) Enter: *2*
        iii) Enter: *[previousquery]*
        iv) Enter: *[qdir].csv*
        v) Enter: *pde_MMYY.csv*

        Where MMYY is the month and year of the most recent event in the new query.

5) Every time the catalog is updated, it should be associated with the most current version of GCMT for use in Slab2. See **Associating the PDE and GCMT Data** for the next steps.

Adding Data from the Global Centroid Moment Tensor (GCMT) Catalog

1) Navigate to [Slab2_location]/catalogquery and activate the slab2 environment: *conda activate slab2env*
2) Identify most recent event in the latest gcmt catalog on file. This will be listed in the catalogquery folder as gcmt_MMYY.csv.
3) Go to http://www.ldeo.columbia.edu/~gcmt/projects/CMT/catalog/NEW_MONTHLY/ and see if any months have been added since MMYY. Please note that we are only using the monthly CMTs as opposed to the quick CMTs.
4) Copy and paste each new MMYY to individual files. Put all of these files into a single folder.
5) Use *makecolumns.py* to merge all of these files into a single column file.
    a) In the terminal enter: *python makecolumns.py*
    b) Follow prompts
    c) Two files will be written:
        i) Converted file listing all events in new folder in appropriate format.
        ii) New updated gcmt file in the appropriate format.
6) Every time that this file is updated, it should be associated with the most current version of the PDE for use in Slab2. See **Associating the PDE and GCMT Data** for next steps.

Associating the PDE and GCMT Data

1) Navigate to [Slab2_location]/catalogquery and activate the slab2 environment: *conda activate slab2env*
2) Identify the most recent gcmt catalog query (named gcmt_MMYY.csv) and PDE catalog query (named pde_MMYY.csv).
3) Make sure gcmt catalog is in CSV format with the column: [year month day hour minute second lat lon depth gmlat gmlon gmdep mrr mtt mpp mrt mrp mtp smo expo mag fss fth fclvd]
4) Make sure PDE file is in .csv format with columns: [id_no, time, lat, lon, depth, mag, mag type, moment_lon, moment_lat, moment_depth, mrr, mtt, mpp, mrt, mrp, mtp, type]
5) It is okay if there are duplicates in either file, they will be removed within the association code.
6) If it does not exist, make a directory called “matchfiles”
7) In the terminal enter: *python pdegcmtdupassc.py -g [gcmtfile] -c [ccatfile] -f [assofile]*
    a) [gcmtfile] is the gcmt file to be associated (*gmt_MMYY.csv)*
    b) [ccatfile] is the comcat/pde query file to be associated (*pde_MMYY.csv)*
    c) [assofile] is the filename to write the associated output to
8) This could take up to ~1 hour to run
9) Extra information files will be written to [Slab2_location]/catalogquery/matchfiles.

10) Move the new associated file to [Slab2_location]/MasterDB/gcmt_pde_assc
11) Rename the file to ALL_EQ_MMDDYY.csv where MMDDYY indicates the date of the most recent event in the catalog or the date of the PDE query.
12) Remove the other existing file in this folder, as this file is a more updated version.
13) Document this change in the first line of the file: [Slab2_location]/MasterDB/masterdb.xlsx
14) Make a new folder as described in step (1) of **Creating an Input File,** or if no other changes to that folder need to be made, replace the existing ALL_EQ_MMDDYY.csv with this newfile, and rename the folder to reflect the update in month and year.
15) To make new Slab2 input files with the updated catalog query, see **Creating an Input File.**

Removing Data Points

- If an event or a cluster of events is clearly not associated with the slab but is not filtered by the broader filter specifications associated with the slab model, it may need to be added to a remove file for the slab model.
- Removing a point is the last resort option to filtering an event after the best filter parameters are identified.

1) Make a file listing the points to be removed in CSV format.
    a) The file must at minimum have the columns: lon, lat, depth, etype
    b) The file can have any number of other columns, and the order does not matter
2) Document why the points are being removed, name the file [slab]_[desc].csv, and save the file in [Slab2_location]/slab2code/library/points_to_remove/current_files/
    a) Where [slab] is the three-letter slab code associated with this region
    b) And [desc] is a short descriptive word or phrase that hints as to why the points in the file are being removed from that slab.
3) Re-run the model using *slab2.py*. The points in the added remove file will then be filtered from the input dataset at the onset of the modeling algorithm.

Modifying Polygons

- Polygons are used for defining the extent of each slab model. The bounds of each polygon are determined by the spatial extent of the available data (see Figure S2 in the supporting information of Hayes et al., 2018).
- If new data are incorporated that are outside the bounds of the current polygon for a given subduction region, the user may want to modify the polygon for that region. If creating a model for a new slab region, a polygon must also be defined for that new region.

1. Navigate to [Slab2_location]/slab2code/library/misc and open *slab_polygons.txt*.
    a. This file contains the lon, lat coordinates for defining the polygon for each slab region.
2. To update a polygon for a particular slab region, the user must make changes to *slab_polygons.txt*.
    a. If the user would like to plot the polygons with GMT, use file *slab_polygons2.txt*.

b. To use changes made to *slab_polygons2.txt* in the slab2 code, in the terminal type: *bash ConvertPolygonsFile.sh*. This bash script will convert *slab_polygons2.txt* to *slab_polygons.txt* (i.e., *slab_polygons.txt* will be updated from *slab_polygons2.txt).*
c. Alternatively, a CSV polygon file for a particular slab region can be created (use headers with columns lon, lat). Edit *listopoly.py* to add the CSV file name and three-letter slab code on lines 8 and 10, respectively. In the terminal type *python listopoly.py*. The new polygon coordinates from the CSV file will be printed to the terminal window in the format of *slab_polygons.txt*. Replace the printed lines with those in *slab_polygons.txt* for the slab region to be updated. Make sure to delete the last comma (i.e., keep the format in *slab_polygons.txt* consistent).

3. In *slab_polygons.txt*, be sure to keep the format the same (slab name, lon, lat, lon, lat,). Also, the first lon, lat pair and the last lon, lat pair for each slab region should be the same.

Making or Using a Slab Guide

- Slab guides are a reference model used as an a priori approximation of the slab geometry for determining grid node location and the orientation of ellipses that are then used for data filtering (refer to the supporting information of Hayes et al., 2018 for more information).
- The slab guide, or reference model, utilized is Slab1.0. If a subduction region does not have a Slab1.0 model, then a hand contoured surface is used as the slab guide.
- A user may want to create a slab guide if modelling a subduction zone that is not included in the current Slab2 database. Likewise, a user may want to modify a pre-existing slab guide if they feel that the current slab guide is no longer appropriate.

1. The Slab1.0 reference grids are stored in [Slab2_location]/slab2code/library/slab1grid
    a. The user should not modify these files.
2. If the user would like to add or modify a slab guide, navigate to [Slab2_location]/slab2code/library/slabguides.
    a. New slab grids should be stored here as a grd file with the name [slab]_DD.MM.YR.grd, where [slab] is the three-letter name for slab model region and DD.MM.YR is the day, month, and year.
        i. The format should be lon, lat, negative depth.
    b. If modifying a pre-existing slab guide, the user should first make a copy of the guide and move it to slabguides/Originals.
    c. If modifying a slab guide that has an existing Slab1.0 grid in [Slab2_location]/slab2code/library/slab1grid, move the Slab1.0 grid into [Slab2_location]/slab2code/library/slab1grid/notused before running a slab model.

Determining Shift Magnitude

- The shift magnitude is used to account for the difference between interslab and intraslab data, starting at the base of the seismogenic zone. In other words, the shift takes nodes located within the slab and moves them to the slab's inferred surface.

- This calculation uses age of the seafloor/oceanic lithosphere from Mueller et al., 2008 at the trench and the thermal boundary layer approximation (Turcotte & Schubert, 2002) for determining the total lithosphere thickness based on the seafloor age. Please refer to the supporting information of Hayes et al., 2018 for more details.
- A percentage of this thickness, *fracshif*t, is then used as the total shift magnitude. This value is set within the parameter file for each slab region.
    - To change this value, navigate to [Slab2_location]/slab2code/library/parameterfiles
    - Open the desired slab region parameter file and change the parameter for *fracS.*
- A depth range over which to taper from no shift, at the base of the seismogenic zone, to a full shift is also specified in the parameter file for each slab region.
    - To change this value, navigate to [Slab2_location]/slab2code/library/parameterfiles
    - Open the desired slab region parameter file and change the parameter for *taper* (give value in km).
- Other changes to the determination of the shift magnitude could involve using a new seafloor age file (i.e., replace file interp_age.3.2g.nc located in [Slab2_location]/slab2code/library/misc), using different values for the variables in the thermal boundary layer approximation (i.e., modify values in the function slabShift_noGMT in slab2functions.py), using a different calculation for lithosphere thickness (i.e., replace the equation in slabShift_noGMT),or by using a user specified lithosphere thickness (see the next bullet).
- To enable user-specified lithospheric thickness values for a given slab region, create a file named [slab]_thickness.csv, where [slab] is the three-letter slab code.
    - Save the file in [Slab2_location]/slab2code/library/slabthickness in a CSV file format.
    - The file should have columns lon, lat, thickness (km). Do not include a header.
    - lon, lat should be the location at the trench. The user can take the locations directly from [Slab2_location]/slab2code/library/misc/trenches_usgs_2017_depths.csv. Else, refer to the section on **Adding or Modifying a Trench File** for adding in new trench data.
    - The code will automatically look for the slab thickness file in aforementioned directory. If no file exists, the code will use the default approach.

Running L-Curve for a Model

1) After identifying the optimum parameters for a given model, the optimum smoothness must be determined. Navigate to [Slab2_location]/slab2code and activate the slab2 environment: *conda activate slab2env*
2) To generate an L-curve for a 12.22.17 suluwesi model stored in the Output directory of slab2code:
    a) In the terminal enter: *python lcurvewrap.py -p library/parameterfiles/sulinput.par -f Output/sul_slab2_12.22.17*
3) The code should be run overnight for larger models (sam, sum, kur, alu), and takes between ~10 minutes and ~12 hours to complete.

4) An L-curve plot will be saved in the folder listed after -f in the input, along with a list of filters and the associated misfits.
5) Visually interpret the x value of the curve apex in the left subplot of [slab]_slab2_lcv_[date].png
6) Find the x value from previous step in the filt column of [slab]_slab2_lcv_[date].csv, then identify the actualfilter column for that row and use it as the smoothing filter for this slab.
7) Run the model with the suggested smoothing parameter, (change the filt value in the parameter file) and visually assess whether this value makes sense for this slab.
    a) The filters are often too high, deteriorating the geometric features of slabs with steep dips at shallow extents.
    b) If the model seems too smooth, continue making models, decreasing the suggested filter width for each run, until a suitable smoothing balance is achieved.
8) A histogram of difference values between each data/node and the surface for each smoothing value is saved in the model output directory in a directory with the name diffhists
9) The two plots in the Lcurve figure that are not the L curve are the first and second derivatives of the Lcurve (derived as the slope of the lcurve, then the slope of the slope of the lcurve).
    a) These were used as an attempt to identify the inflection point of the curve, which doesn't work. A different approach is needed to quantitatively identify the L-curve apex.

References

Crameri, F., Shephard, G.E. & Heron, P.J. The misuse of colour in science communication. *Nat Commun* **11**, 5444 (2020). https://doi.org/10.1038/s41467-020-19160-7

Hayes, G.P., Moore, G.L., Portner, D.E., Hearne, M., Flamme, H., Furtney, M. and Smoczyk, G.M., 2018. Slab2, a comprehensive subduction zone geometry model. *Science*, *362*(6410), pp.58-61.

Portner, D.E. and Hayes, G.P., 2018. Incorporating teleseismic tomography data into models of upper mantle slab geometry. *Geophysical Journal International*, *215*(1), pp.325-332.