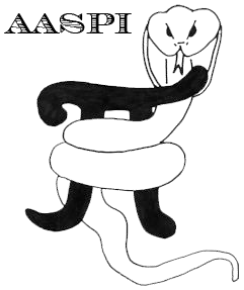


PRESTACK STRUCTURE-ORIENTED FILTERING – PROGRAM sof_prestack

Contents

Overview

Prestack seismic analysis provides information on rock properties, lithology, fluid content, and the orientation and intensity of anisotropy. Such analysis demands high-quality seismic data whereas noise may be present even after careful processing. Noise in the prestack gathers may not only contaminate the seismic stacked image, thereby lowering the quality of seismic interpretation, but it may also bias the seismic prestack inversion for rock properties, such as acoustic- and shear-impedance estimation. Noise on the migrated gathers also contaminates the semblance scans used in residual velocity analysis.
Common postmigration data conditioning includes running window median and Radon filters that are applied to the flattened common reflection point gathers. In program **sof_prestack**, we generalize the concept of structure-oriented filtering used in program **sof3d** to filter 4D and 5D supergathers (t,h,φ,x,y) (time, offset, azimuth, inline distance, crossline distance) along structure with edge preservation in the xy plane.

Computation Flow Chart

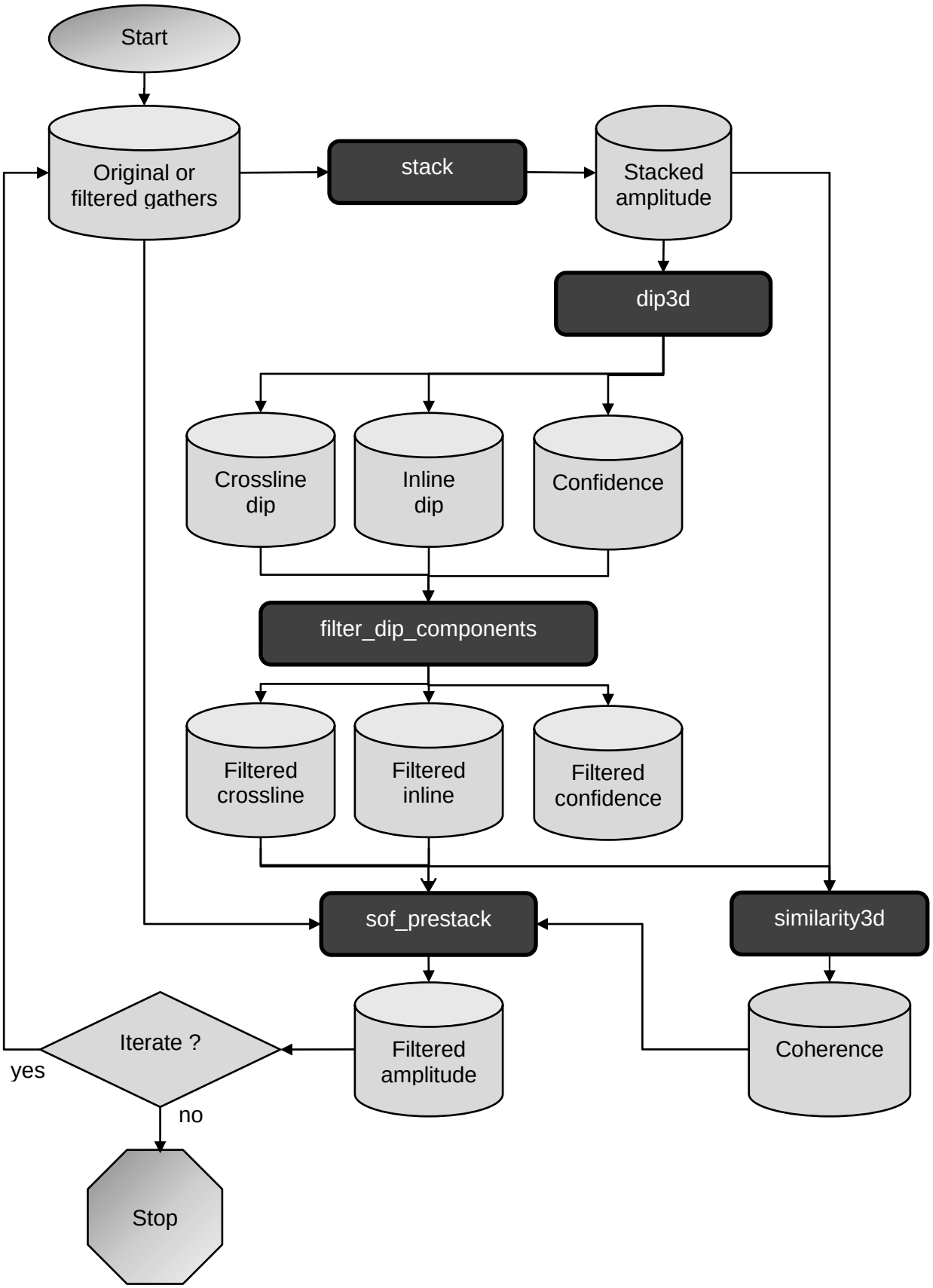
Program **sof_prestack** is a generalization of program **sof3d.** For this reason, the input parameters and workflow of the two algorithms are very similar, except for the intermediate need to stack the prestack data prior to computing a consistent volumetric dip, azimuth, and similarity.

The outputs include principal component- (also called Karhunen–Loève, or KL-), Lower-Upper-Middle (LUM), alpha-trimmed-mean-, or mean-filtered versions of the input seismic amplitude data.

Most data will have been furnished by a seismic processing company as prestack flattened seismic gathers with each trace forming a CRP containing offset values, azimuth values, and or tile number. For flatter reflectors common to resource plays, migration stretch in the farther offsets can be ameliorated through program **compensate_for_migration_stretch**.

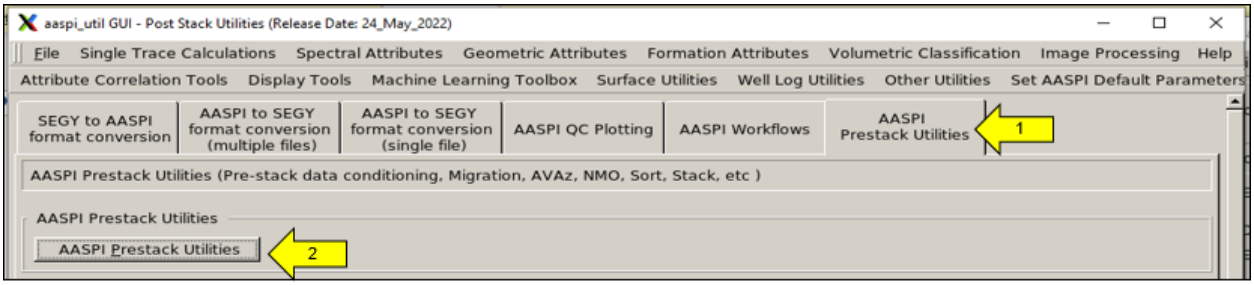
The structure-oriented filter is applied to each common-offset/common-azimuth or common tile gather using the dip, azimuth, and similarity volumes computed from the stacked data volume.

The flow chart below shows a typical workflow described in the documentation entitled “Prestack Workflows: Structure-oriented filtering” and shows how one should first stack the migrated gathers and then compute structural dip and coherence from the stacked data volume prior to prestack structure-oriented filtering. This documentation will only describe the details of program **sof_prestack**.

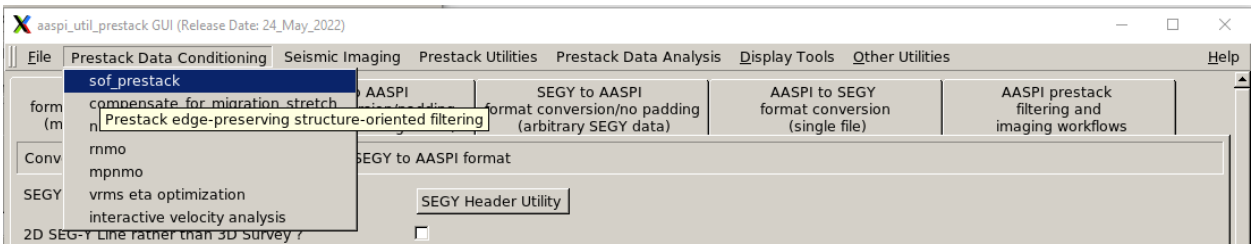

Start
Original or filtered gathers
stack
Stacked amplitude
dip3d
Crossline dip
Inline dip
Confidence
filter_dip_components
Filtered crossline
Filtered inline
Filtered confidence
sof_prestack
similarity3d
Iterate ?
yes
no
Filtered amplitude
Coherence
Stop

Launching the Graphical User Interface (GUI) - aaspi_util_prestack

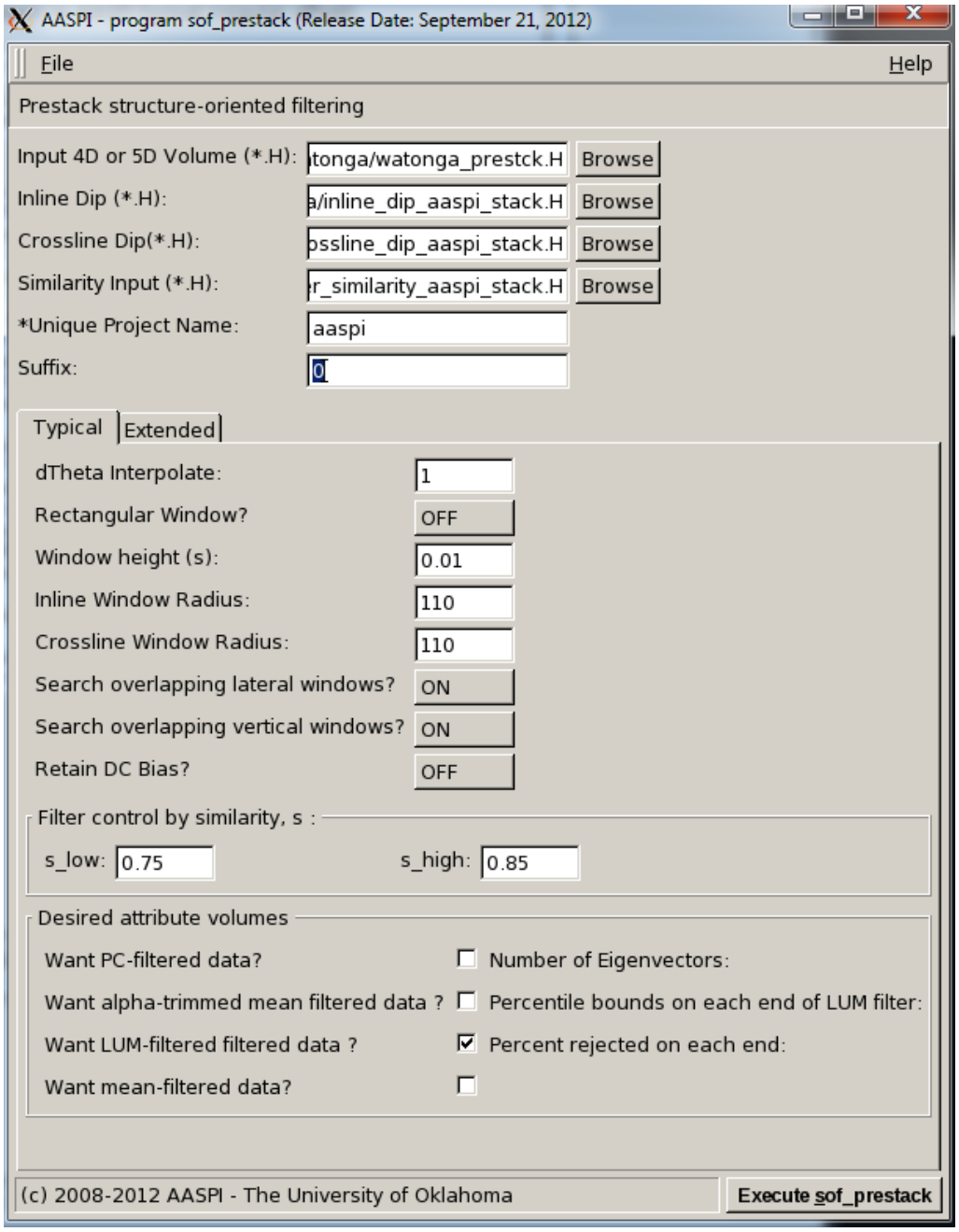
There are two ways to invoke **the aaspi_util_prestack** GUI: either by (in Linux only) typing it in on the command line, or by choosing it on the upper right-hand corner of the (poststack analysis) **aaspi_util GUI** (1) and clicking on the AASPI Prestack Utilities button (2):



Program **sof_prestack** is found under the *Prestack Data Conditioning* tab



In either manner, the following GUI appears. One of the first things we will wish to do is to stack our prestack time-migrated data volume:



As you might suspect, if we have *n* offsets/azimuths, the program runs *n* times longer than program **sof3d**. As in all AASPI codes, program progress is echoed to the *xterm* from which **aaspi_util_prestack** was launched. The end of the print-out looks like this (see next page):

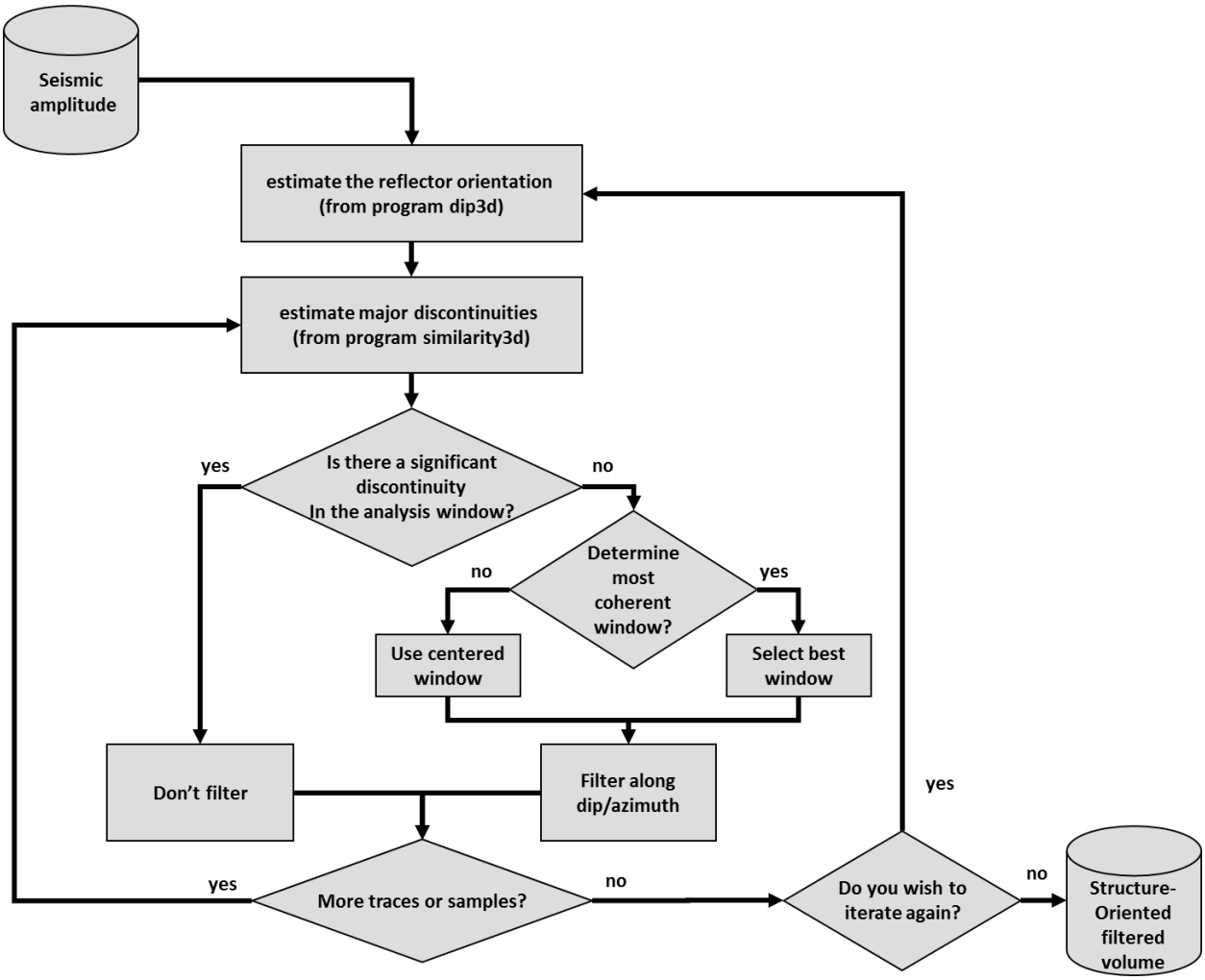
```
 process                                    task       time (hr)  time/trace (s)
  0:                      read and scale data           0.004           0.000
  0:             compute analytic filter bands          0.000           0.000
  0:                         send data via MPI           0.000           0.000
  0:                      receive data via MPI           0.000           0.000
  0:                      send results via MPI           0.000           0.000
  0:                   receive results via MPI           0.007           0.000
  0:                        scan discrete dips           0.000           0.000
  0:                           interpolate dip           0.000           0.000
  0:                      calculate cov matrix           0.000           0.000
  0:                    calculate eigenvectors           0.000           0.000
  0:                      project data onto v1           0.000           0.000
  0:                     write results to disk           0.020           0.000
  0:                                 total time           0.179           0.000
total data written to disk:     5082084 traces         500 samples
total data written to disk:    1574.233 Mbytes
transfer rate              :      21.648 Mbytes/s
          0 : memory deallocated on master
          0 : memory deallocated on master and slaves
          3 :normal completion. routine sof_prestack
          6 :normal completion. routine sof_prestack
          7 :normal completion. routine sof_prestack
          4 :normal completion. routine sof_prestack
         10 :normal completion. routine sof_prestack
          1 :normal completion. routine sof_prestack
          8 :normal completion. routine sof_prestack
         11 :normal completion. routine sof_prestack
         12 :normal completion. routine sof_prestack
          0 :normal completion. routine sof_prestack
          2 :normal completion. routine sof_prestack
Closing file: /home/oswaldo/projects/watonga/d_gathers_alphatrim_filt_aaspi_0.H
          5 :normal completion. routine sof_prestack
          9 :normal completion. routine sof_prestack
Closing file: /home/oswaldo/projects/watonga/d_gathers_lum_filt_aaspi_0.H
Closing file: /home/oswaldo/projects/watonga/d_gathers_mean_filt_aaspi_0.H
Closing file: /home/oswaldo/projects/watonga/d_gathers_pc_filt_aaspi_0.H
Closing file: /nfs/raid5/oswaldo/watonga/cropped_cmp_gathers_aaspi_0.H
Closing file: /nfs/raid5/oswaldo/watonga/crossline_dip_aaspi_stack.H
Closing file: /nfs/raid5/oswaldo/watonga/inline_dip_aaspi_stack.H
Closing file: /nfs/raid5/oswaldo/watonga/sobel_filter_similarity_aaspi_stack.H
sleep 10
+ sleep 10
```

If you type '*ls –ltr*' in the above *xterm*, you find the most recent files to be,

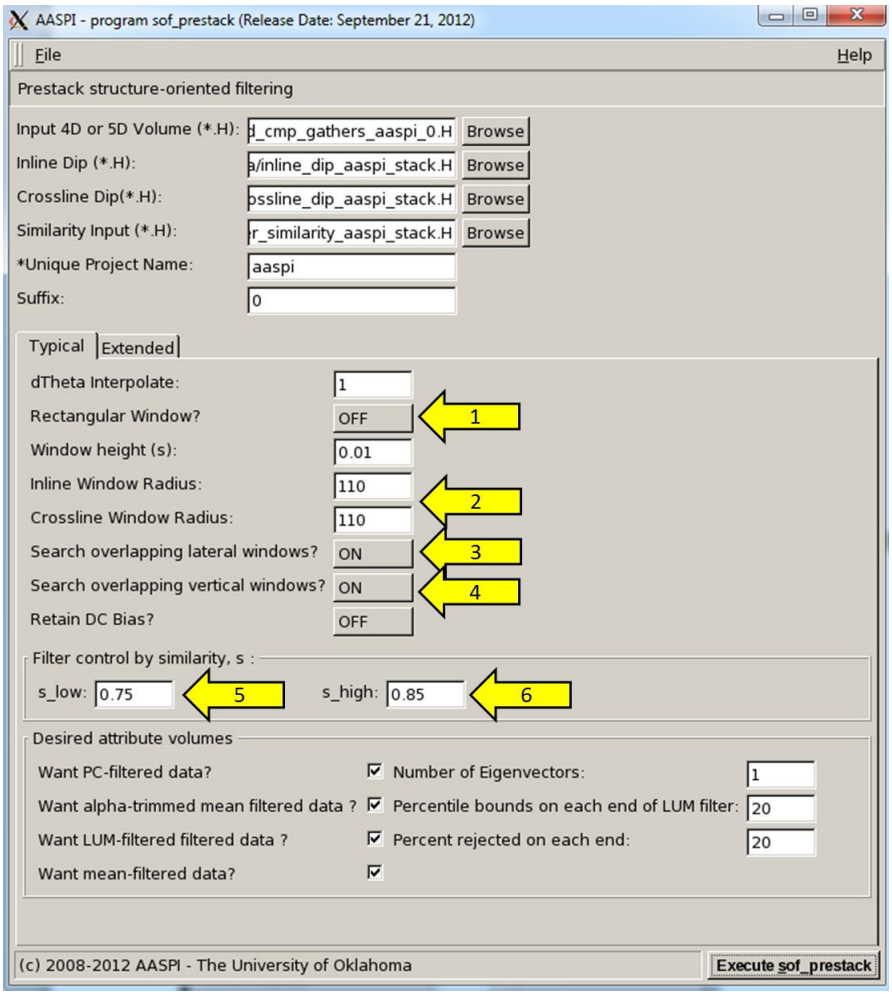
```
-rw-r--r-- 1 oswaldo  aaspi   37 Oct  2 08:49 live_processor_list
-rwxrwxrwx 1 oswaldo  aaspi 2.3K Oct  2 08:49 d_gathers_pc_filt_aaspi_0.H@@
-rwxrwxrwx 1 oswaldo  aaspi 5.9K Oct  2 08:49 d_gathers_pc_filt_aaspi_0.H
-rwxrwxrwx 1 oswaldo  aaspi 2.3K Oct  2 08:49 d_gathers_mean_filt_aaspi_0.H@@
-rwxrwxrwx 1 oswaldo  aaspi 5.9K Oct  2 08:49 d_gathers_mean_filt_aaspi_0.H
-rwxrwxrwx 1 oswaldo  aaspi 2.3K Oct  2 08:49 d_gathers_lum_filt_aaspi_0.H@@
-rwxrwxrwx 1 oswaldo  aaspi 5.9K Oct  2 08:49 d_gathers_lum_filt_aaspi_0.H
-rwxrwxrwx 1 oswaldo  aaspi 2.3K Oct  2 08:49 d_gathers_alphatrim_filt_aaspi_0.H@@
-rwxrwxrwx 1 oswaldo  aaspi 5.9K Oct  2 08:49 d_gathers_alphatrim_filt_aaspi_0.H
-rwxrwxrwx 1 oswaldo  aaspi  64K Oct  2 08:59 sof_prestack_aaspi_0.out
-rwxrwxrwx 1 oswaldo  aaspi  82K Oct  2 09:00 aaspi_sof_prestack_pf.out
-rwxrwxrwx 1 oswaldo  aaspi  341 Oct  2 09:04 aaspiviewer.parms
-rw-r--r-- 1 oswaldo  aaspi 2.6K Oct  2 09:04 temp_crop_09:04:56.H@@
-rw-r--r-- 1 oswaldo  aaspi 6.9K Oct  2 09:04 temp_crop_09:04:56.H
-rw-r--r-- 1 oswaldo  aaspi 2.8K Oct  2 09:05 temp_slice_09:04:56.H@@
-rw-r--r-- 1 oswaldo  aaspi 7.2K Oct  2 09:05 temp_slice_09:04:56.H
[oswaldo@tripolite watonga]$
```

which shows that the output of program **sof3d_prestack** is called *d_pc_filt_westcam_1.H*. You may wish to run TWO iterations of structure-oriented filtering. To do so, return to your **sof3d_prestack** GUI, and use the browser to find the file *d_pc_filt_westcam_1.H*. Let's use a suffix of 'pc_2' to indicate that the results are from 2 passes of structure-oriented filtering. Use the same dip calculation and Kuwahara window (though you can rerun program **dip3d** on the file *d_pc_filt_westcam_1.H*.

The main workflow for the structure oriented filter is described in Davogustto and Marfurt (2011):



Let's explain the advanced parameters in more detail:



The program default is to use a circular search window that for equal inline and crossline spacings will contain 5 traces. These parameters can be changed by (arrow 1) selecting *rectangular* window analysis (where 3x3=9 traces fall within the smallest window) and/or by (arrow 2) increasing the inline and crossline radii to define an elliptical or rectangular window of the desired size. Cost (computation time) increases but the strength of the filter increases with increasing window size. Rather than double the window size in both directions, a more effective workflow is to iterate smoothing by smaller windows as shown in the flow chart above. These smaller windows not only follow curving reflectors better but implicitly taper the filter towards the edges

Marfurt (2006) built on Luo et al.'s (2002) Kuwahara algorithm to implement a robust volumetric dip and azimuth calculation that avoided smearing of faults, fractures and other discontinuities using an overlapping window method. This technique along with the seismic data input can be used to implement volumetric filters based on mean, median, α-trimmed mean or principal component algorithms (see box below in this chapter for an overview of PC filtering). Rather than using a centered analysis window, the algorithm uses the most coherent window containing each analysis point, hence enhancing the lateral resolution near discontinuities and reducing both random and coherent noise (Marfurt, 2006).

In the figure below (a) represents a 13-trace circular analysis window centered about the analysis point indicated by the red solid dot. Each of the traces represented by the green dots in (b) form the center of their own 13-trace analysis windows. Each of these overlapping analysis windows also contains the trace represented by the red dot. The original Kuwahara et al. (1979) algorithm estimated the mean and standard deviation of the data in each window. The window having the smallest standard deviation was declared to best represent the signal; the mean of this window was then assigned to be the filtered data at the output (typically uncentered) analysis point. Marfurt (2006) applied this same approach to 3D seismic data using a simple extension. Rather than using the standard deviation, he computed the dip-steered coherence in 3D overlapping windows. After selecting the window with the highest coherence, he then computed either the mean, alpha-trimmed mean, or principal-component filtered estimate of the signal and assigned the result to the filtered volume at the analysis point. Use of such (arrow 3) laterally shifted windows helps avoid smoothing across faults. Use of (arrow 4) vertically shifted analysis windows helps avoid smearing across angular unconformities.

Theory: Review of linear and nonlinear filters

Let's assume we have *J* voxels that fall within a 2D or 3D analysis window. There are several linear and nonlinear filters that can be applied.

The mean filter

The mean filter is the simplest, where the mean μ of *J* samples d_j is defined as:

$$\mu = \frac{1}{J}\sum_{j=1}^{J} d_j \,. \tag{1}$$

The mean filter is a smoothing filter, and may not only smooth across faults but smooth in erroneous spikes into the output.

The median filter

The first step of the median filter is to sort the data vector, **d**, into a new vector **u** where $u_k \le u_{k+1}$:

$$\mathbf{u} = \text{sort}\{d_1, d_2, \ldots, d_j, \ldots, d_{J-1}, d_J\}. \tag{2}$$

Then the median, *m*, is defined as:

$$m = u_{(J+1)/2} \,. \tag{3}$$

The median filter is an edge-preserving filter and will preserve changes in dips across faults. It also rejects erroneous spikes in the input data.

The α-trimmed mean filter

The α-trimmed filter is an extension of the median filter. First, the algorithm sorts the data in ascending order as in equation 2. Then one defines a fraction (usually defined as a percentage) of the data that falls within the range,

$$0 \le \alpha \le \frac{1}{2} \,. \tag{4}$$

The filter rejects αJ "outliers" on each end of the data vector and computes the mean of the values of u_j with indices $1+\alpha J \le j \le J-\alpha)(J-1)$:

$$u_{\alpha-trim} = \frac{1}{J - 2\alpha(J-1)} \sum_{j=1+\alpha(J-1)}^{J-\alpha(J-1)} u_j \,. \tag{5}$$

The alpha-trimmed mean filter thus rejects outliers and smooths the remaining values. As such it may still smooth changes in dip across faults.

The Lower-Upper-Median (LUM) filter

The LUM filter is the default filter in **filter_dip_components** and acts in the following manner:

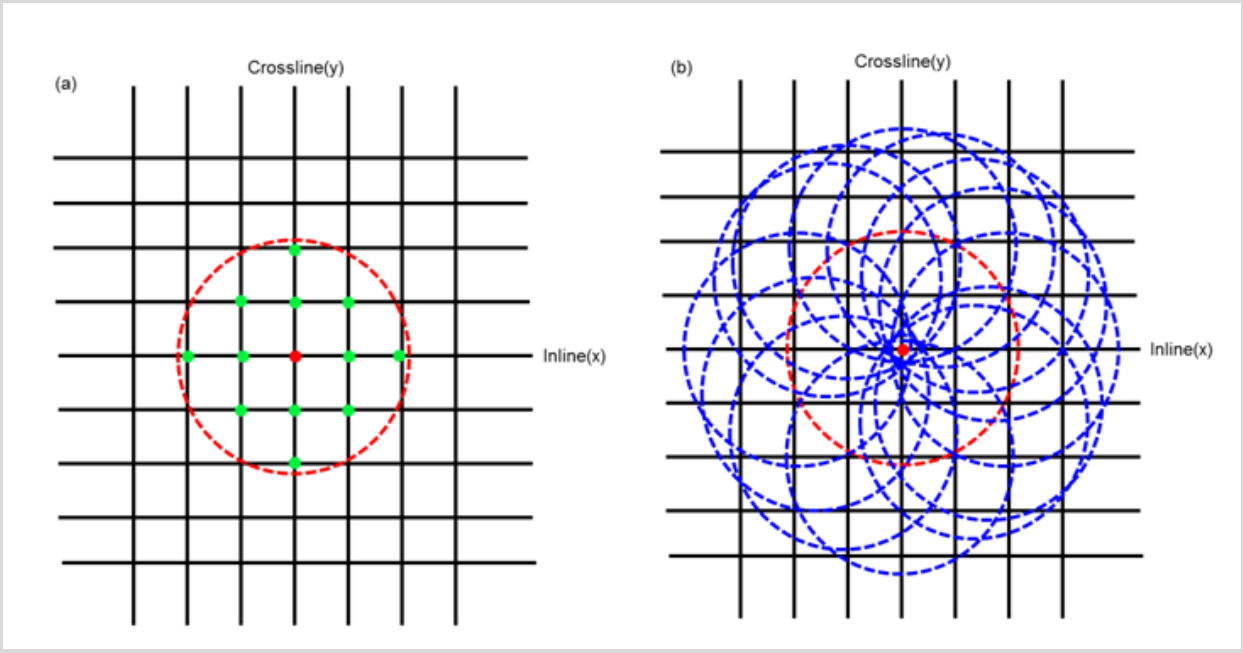
$$u_{LUM} = \text{median}\left(u_{1+\alpha(J-1)}, u^*, u_{J-\alpha(J-1)}\right) = \begin{cases} u_{1+\alpha(J-1)} & u^* < u_{1+\alpha(J-1)} \\ u_{J-\alpha(J-1)} & u^* > u_{J-\alpha(J-1)} \\ u^* & otherwise \end{cases} \quad 0 \le \alpha \le 0.5 \tag{6}$$

Like the alpha-trimmed mean filter, the LUM filter rejects high and low amplitude "outliers". Instead of taking the mean of the remaining samples, it compares the dip value at the center of the analysis window u* to the upper and lower percentiles. If u* falls beyond these percentiles, it clips the value to the upper or lower percentile; otherwise, it leaves the value alone. In this manner, the LUM filter preserves detailed variation, but rejects erroneous values.

Theory: Kuwahara windows

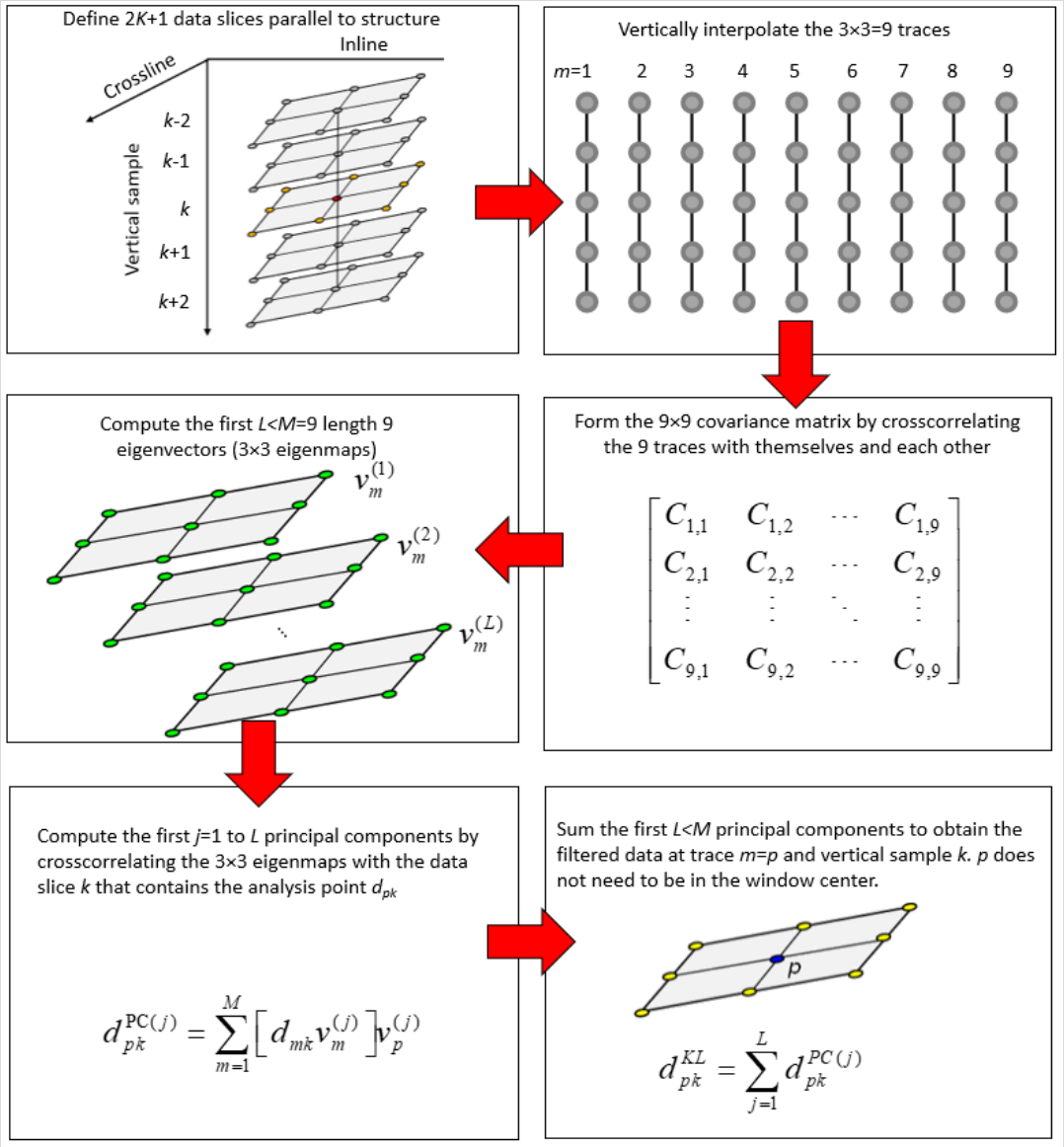
Programs **dip3d, sof3d**, **sof_prestack,** and **kuwahara3d** all use a modification of overlapping window parameter estimates introduced by Kuwahara et al. (1976) in medical imaging. The original idea is simple. If an analysis window contains five traces, then there is a total of five windows (a centered window and four adjacent, offset windows) that contain the analysis point. In Kuwahara et al.'s (1976) original work and Luo et al.'s (2002) edge-preserving smoothing algorithm, one calculates the mean and standard deviation of each window. That window which has the smallest standard deviation is hypothesized to be less noise contaminated. The mean of this window is then used as the output for the analysis point. Marfurt (2006) modified this approach for volumetric dip calculations where he used 3D rather than 2D overlapping windows. In program **sof3d** the "best" window is the one with the highest measure of similarity (e.g., semblance, Sobel filter, or energy ratio). In program **dip3d** using the GST estimate of dip the best window is the one with highest measure of planarity. In program **kuwahara3d** the best window is the one exhibiting the smallest coefficient of variation.

The lower left figure represents a 13-trace circular analysis window centered about the analysis point indicated by the red solid dot. Each of the traces represented by the green dots in the lower left figure form the center of their own 13-trace blue circular analysis windows shown in the lower right figure. For program **sof3d and sof_prestack**, we have precomputed and saved the similarity of each of these 13 blue analysis windows using program **similarity3d**. We therefore read these precomputed data in, select the one with the highest similarity, and apply a mean, alpha-trimmed mean, Lower-Upper-Middle, or principal-component (Karhunen-Loève) filtered estimate of the signal and assign the result to the filtered volume at the (perhaps non-centered) analysis point. Use of such laterally shifted windows helps avoid smoothing across faults. Use of vertically shifted analysis windows helps avoid smearing across angular unconformities.



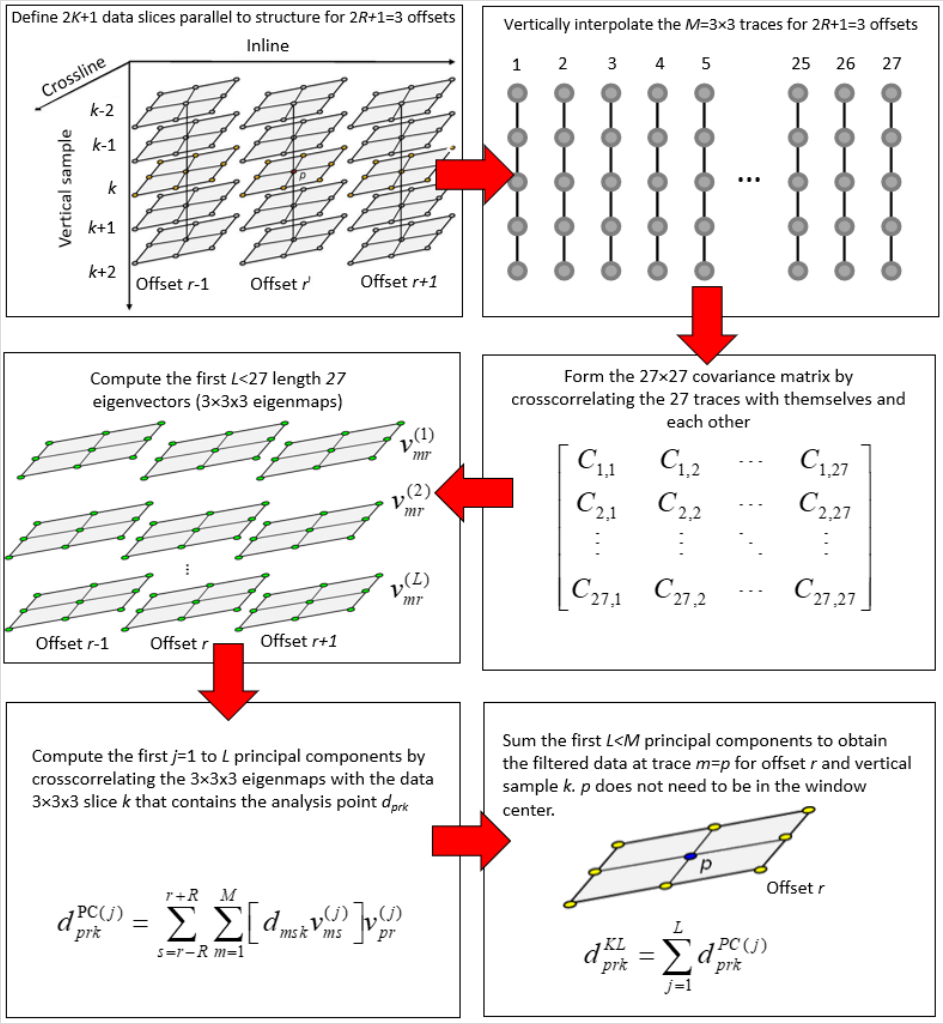
3D analysis windows and covariance matrices for an offset- or azimuth-limited stack

Program **sof3d** can be applied to offset- or azimuth-limited stacks:



Cartoon showing structure-oriented filtering applied to an offset- or azimuth-limited stacked data volume along structural dip using a centered analysis window about the red analysis point. In this example there are 3 crosslines by 3 lines resulting a length M=9 "sample vector" for each interpolated dipping horizon slice at time k. These sample vectors are cross-correlated and averaged from k=-K to k=+K (K=2) time samples resulting in a 9×9 covariance matrix. The first L length 9 eigenvectors represent 3×3 "maps" that best represent the lateral variation of amplitude within the analysis window. These eigenmaps are cross-correlated with the sample vector at time k to compute a suite of L principal components. One or more of these components are then summed to form the filtered data at the analysis point at trace m=p and vertical sample k.

3D analysis windows and covariance matrices for 3 adjacent offset volumes

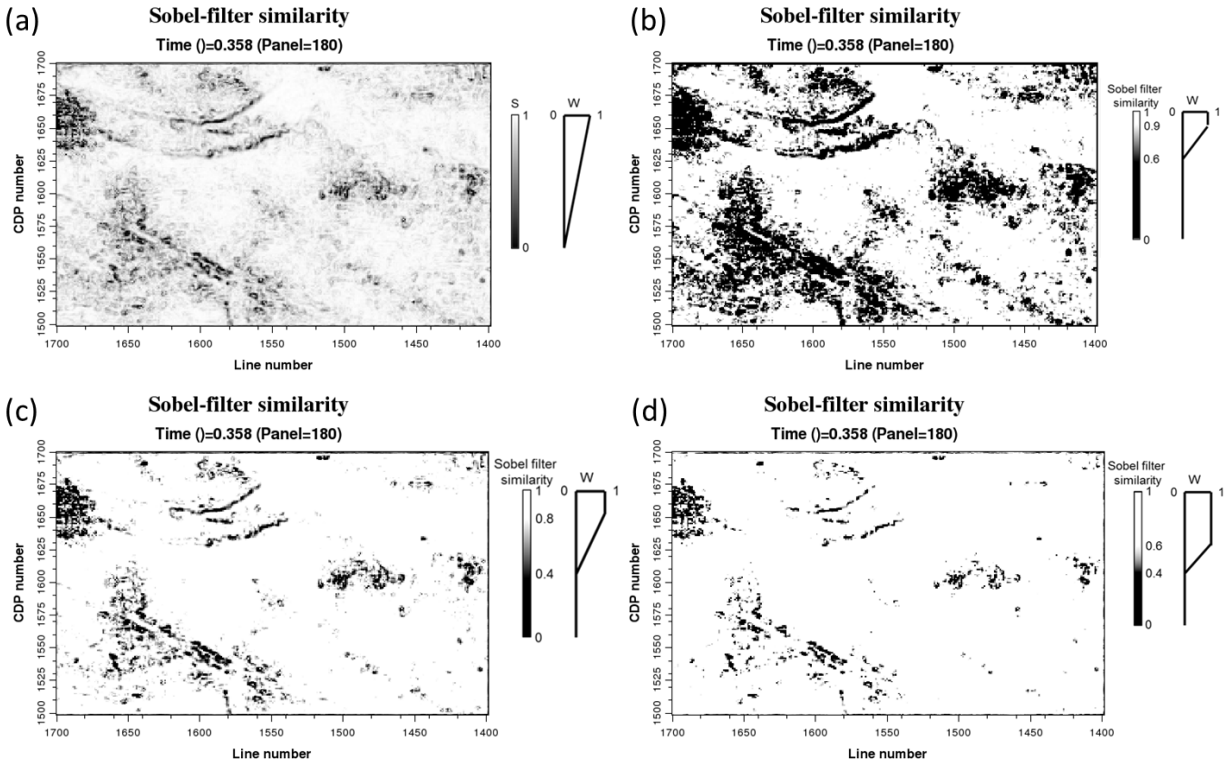


Cartoon showing structure-oriented filtering applied to a 3×3 window of data across 3 adjacent offsets *r*-1, *r*, and *r*+1. The orientation of the analysis window is determined by the structure of the stacked data volume and is the same for all offsets *r*. The window about offset is centered about offset r and ranges from *r-R* to *r+R*. In this example there are 3 crosslines by 3 lines and 3 offsets, resulting a length *27* "sample vector" for each interpolated dipping horizon slice at time *k*. These sample vectors are cross-correlated and averaged from *k*=-*K* to *k*=+*K* (*K*=2) time samples resulting in a 27×27 covariance matrix. The first *L* length 27 eigenvectors represent 3×3×3 "maps" that best represent the lateral variation of amplitude within the analysis window. These eigenmaps are cross-correlated with the sample vector at time *k* to compute a suite of *L* principal components. The *L* components are then summed to form the filtered data at the analysis point at trace *m=p,* offset *r,* and vertical sample *k*.

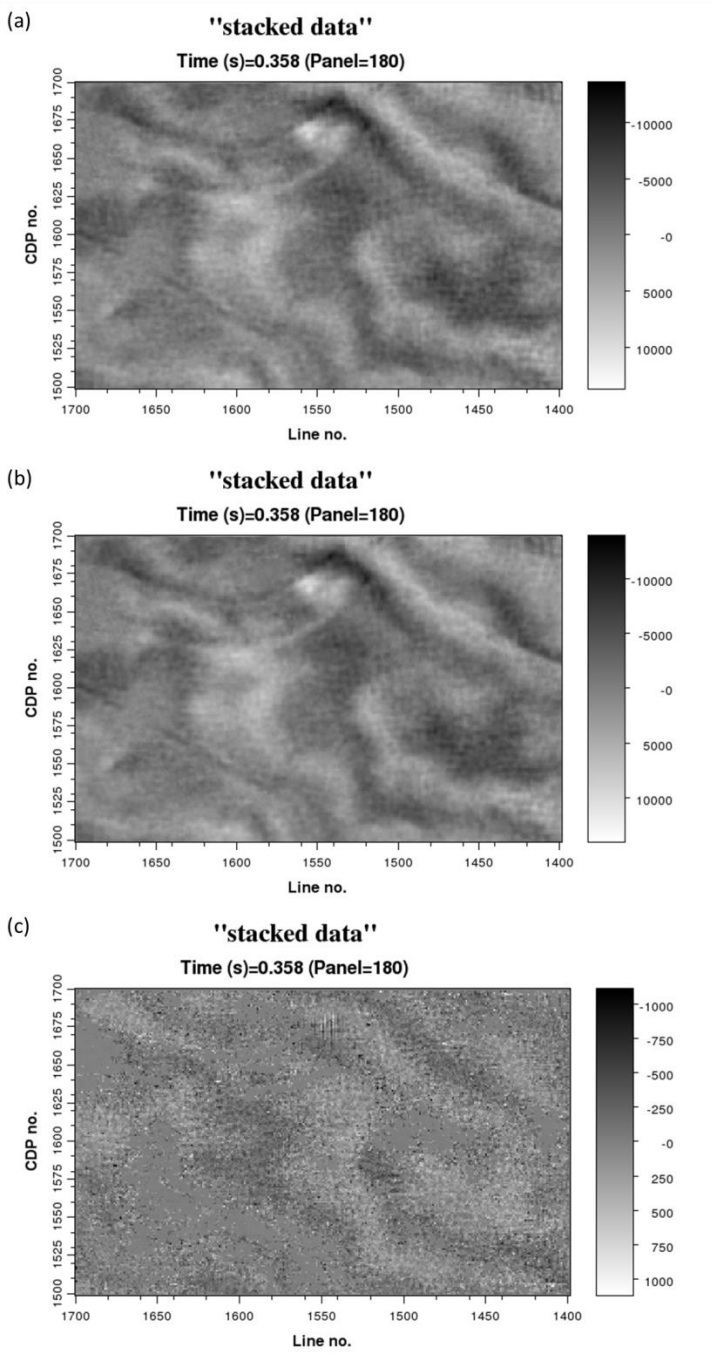
For simplicity, the flow chart shown above indicates a simple *Don't filter* vs. *Filter along dip/azimuth* branch. In the current implementation of program **sof3d_prestack** we've implemented components of the Fehmer's and Hoecker (2003) workflow. If the value of the similarity attribute at the analysis point falls *below* the threshold indicated by arrow 5, $s<s_{low}$, no filtering takes place and the filtered data are assigned weights of *w=0.0*. If the value of the similarity attribute at the analysis point falls *above* the threshold indicated by arrow 5, $s>s_{high}$,

the filtered data are assigned weights of *w=1.0* such that the filtered data replaces the original data on output. If the value of the similarity attribute at the analysis point falls *between* the two values indicated by arrows 5 and 6, the weights of the filtered data are $w=(s-s_{low})/(s_{high}-s_{low})$, and a linearly weighted average of the filtered and unfiltered data $d_{out}=w*d_{filt}+(1-w)*d_{orig}$, takes place.

The image (a) below shows the color bar applied to the similarity values, *s*, and weights, *w=s*. This is our normal display of similarity, *s*. By modifying the threshold values for S we increase or decrease the smoothing weights thereby changing the aggressiveness of the filter. In (b) we adjust the colorbar to enhance some geological features present in the data. Our filter would thus unfortunately *preserve* these features. (c) and (d) show a less optimal filtering parameter selection. Given this parameters, both noise and signal are going to be removed by the filter.



Let's see what structure-oriented filtering has done to our seismic amplitude data (see next page).



There is a subtle difference between the original data (a) and the filtered data (b). When we take a look a t the difference between (a) and (b) we note that has been removed is mostly incoherent noise and some linear patterns.

References

Davogustto, O., and K. J. Marfurt, 2011, Footprint suppression applied to legacy seismic data volumes: GCSSEPM 31st Annual Bob. F. Perkins Research Conference on Seismic attributes – New views on seismic imaging: Their use in exploration and production, 1-34.

Ha, T., and K. J. Marfurt, 2017, The value of constrained conjugate-gradient least-squares migration in seismic inversion: Application to a fractured-basement play, Texas Panhandle: Interpretation, **5**, SN13-SN23.

Kuwahara, M., K. Hachimura, S. Eiho, and Kinoshita, 1976, Digital processing of biomedical images: Plenum Press, 187-203.

Mutlu, O., and K. J. Marfurt, 2015, Improving seismic resolution of prestack time-migrated data: Interpretation, **3**, T245-T255.

Zhang, B., T, Lin, S. Guo, O. E. Davogustto, and K. J. Marfurt, 2016, Noise suppression of time-migrated gathers using prestack structure-oriented filtering: Interpretation, **4**, SG19-SG-29.