An Automatic Horizon Picker, a Window Extractor and a Node Finder for the CREWES MATLAB Library.

Christopher B. Harrison

ABSTRACT

This paper covers three new programs created by Christopher Harrison for the CREWES MATLAB library. The first program is a first generation automatic horizon and event picker named PICKSLE. This program uses a windowed section on a single trace and cross correlates this area either across an entire section of data or a specified section as selected by a user. PICKSLE allows the developer or user to have control over various aspects of the cross correlation scanning process, from the percent at which the cross correlation will be checked and kept, to visualization of the programs scanning process. Depending on the parameters initially chosen by a developer or user, PICKSLE will either be slow and reasonably accurate, or fast and quite possibly inaccurate. This program is still under development, but is usable in its present configuration.

The last two smaller programs were specifically developed for use with PICKSLE, but as development progressed it became apparent that their potential could benefit other programs. The first of these smaller programs can be used in conjunction with other developments requiring a user to create their own windowed area on a single trace of data. The program is called WINEXTRACTOR and visualizes a single trace, an initial adjustable point of clicking, and two adjustable windowed areas. These attributes, along with several other predefined or user-selected arguments allow for user to adjust a precise window area on an input trace.

The final program covered in this paper is utilized by both the above programs in restricting either clicking or point movement on an input traces. The program's name is NODEFIND, and isolates points on an input trace as chosen by a user. The choices for isolation are peaks, troughs, and zero crossing points.

PICKSLE

PICKSLE was developed out of the desire for a picking tool that automatically and precisely chooses picks across an entire seismic section. PICKSLE is quick to auto-pick sections when a moderate level of accuracy is required. Higher levels of accuracy decrease the speed of operation. PICKSLE is best integrated into other programs that image data similar to and including seismic data. PLOTIMAGE was used as the testing base for PICKSLE with PICKLSE integrated into the newest release of PLOTIMAGE. PICKSLE is activated in following three ways.

picksout = **picksle**(*transfer,sesistruct*);

picksout = **picksle**(*transfer,sesistruct,masteraxis*);

picksout = **picksle**(*transfer,sesistruct,masteraxis,preferances*);

The ‘transfer’ argument is a function callback that executes upon the completion of the PICKSLE function. When PICKSLE is complete, the program automatically deletes all lines and points that were created by the PICKSLE routine and passes the argument ‘picksout’ to the ‘transfer’ routine as specified by the developer.

The argument ‘seisstruct’ is seismic or similar type of structured data is passed. As the arguments name suggests the information has to be set up in a MATLAB structure format. The structure filed names must appear as follows.

seisstruct.SEIS = [2D array of seismic data];

seisstruct.T = [Depth or time (optional)];

seisstruct.X = [Horizontal Distance (optional)];

PICKSLE has two optional arguments. The ‘masteraxis’ argument allows for the developer to specify which axis PICKSLE will be executed on. This argument is important if there are multiple axes associated with a master program calling PICKSLE. This ‘masteraxis’ argument will limit picking to the axis chosen.

The second optional argument in PICKSLE is the ‘preference’ structure. This structure contains information that allows for a wider range of controls of the PICKSLE program. A few of the arguments in the ‘preference’ structure are meant for strictly developer control. The preferences must be set up in a MATLAB structure format and require specific field names and types of inputs to work properly with PICKSLE. It should be noted that not all preferences need be included in the ‘preference’ structure when calling PICKSLE. The following describes each field name and the subsequent data needed for the ‘preference’ structure.

The ‘hmsg’ property is used to pass text information for the user from PICKSLE to the program it has been developed into. Information from PICKSLE will be sent to the handles ‘string’ property to inform a user of PICKSLE statues or if problems occur. This argument is optional and defaults to empty.

The ‘nodetolerance’ property determines how many picks are made on a given scan. The number of picks kept in a scan is determined by N picks / ‘nodetolerance’. The lower the number, the more picks are kept from the scan. The minimum number is 1, which keeps all picks as found by the PICKSLE scan. The value of this argument defaults to 5 and a developer could foresee ably allow a user to adjust this number as desired.

The ‘sampleadd’ property is responsible for the size of the windowed area used for scanning in PICKSLE. If the ‘sampleadd’ is n samples, the windowed area is 2n. The size of the windowed determines how fast or slow PICKSLE scans through a desired area. The larger the windowed area the longer the scan, and conversely, the smaller the windowed area the shorter the scan. This property defaults to 100 samples.

The 'showprogress' property controls the drawing of lines the size of the windowed area on the axis corresponding to the traces that PICKSLE is scanning. When the entire scanning operation is complete, the lines that were created in the scan area are then deleted from the axis. The purpose of the property is to allow users to visualize the progress of PICKSLE scan. The visualization will slow down the operations of PICKSLE considerably, so limiting this properties use or completely disabling of this property is suggested. The property is default to be 'on'.

The property 'percenttolerance' controls the percent at which the cross correlation will allow for minimal matching during the scanning process. The higher the percent, the higher the accuracy will be with the cross correlation and lower amount of actual matches in the scan. Alternately, a lower percent will allow for more matches being found during the scan. It should be noted that a high number of matches does not necessarily mean more accurate matches. The value is defaulted to 20% or 0.2. Note, the name 'percenttolerance' is deceptive because the value is not input as a percent, but a positive decimal value.

The 'lockto' property is actually applied to another program called NODEFIND, which is discussed in this paper. 'lockto' allows the user to specify whether the initial click point and subsequent scanning, automatically lock on to several different types of points an a trace. The options include, 'open', 'peak', 'trough', 'zerocross', 'zerocross+', and 'zerocross-'. The value is defaulted to 'open' where no restraints on either the initial click point or scanning are in place.

The 'peak' and 'trough' find the local peaks and troughs on an input trace. The term local means that peaks are found both on the positive and negative side of a potential trace. This also goes the same with local troughs that can be isolated on negative and positive sides of an input trace. The various 'zerocross' options need clarification. The fist 'zerocross' option will lock the initial pick point in PICKSLE to zero points and points on the trace that cross over from negative to positive, or positive to negative without any point in-between touching zero. Refer to NOFINED for more details on 'lockto' options.

The property 'askdetail' controls whether the window select figure as created by the routine WINEXTRACTOR is displayed or not when the user has clicked on a PICKSLE activated axis. A complete explanation of WINEXTRACTOR is included in this paper. This value defaults to 'on'.

Theory behind PICKSLE

Window Creation and Trace Scanning

Scanning is the term used for the main operation in PICKSLE. The scanning process searches for picks on a given input trace by using a windowed area and cross correlation. The scanning time depends on the size of the window that has been selected as well as the length of scanning area desired. The window itself is based on the size as either predetermined by the developer or as adjusted by the user in the window selection figure from the WINEXTRACTOR routine. The window is the first major component that

PICKSLE creates for the scan process. Figure 1 shows what two typical windowed areas would look like.

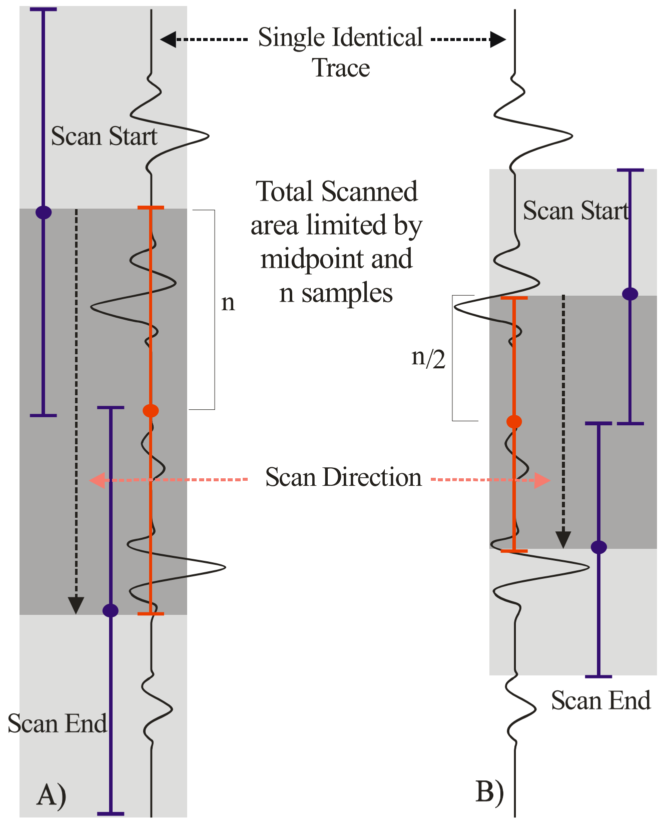

FIG. 1. This figure shows two examples of how PICKSLE creates and scans though a windowed section on a single trace. A) Shows a larger windowed area as determined by n samples specified by the ‘sampleadd’ property. B) shows a smaller windowed area as determined by n/2 samples. The time for scanning would be longer on A) then on B) due to the increased samples used in A). The dark grey area on each section shows where the initial window area is taken. The light gray area represents the entire scan area PICKSLE would undertake on the single trace.

As shown in Figure 1, the windowed areas also act as the limiting factor for PICKSLE’s total scanning. The top of the window limits the upper edge scanning while the bottom of the window limits the lower edge of scanning. The midpoint on the windowed area acts as the third component in limiting scanning by not allowing the scan to go further up than the top of the original windowed area, and not allowing the scan to go further down than he bottom of the original windowed area.

PICKSLE scanning starts at the top of the light grey area as seen on Figure 1 and works down in samples steps. For each sample step, the window area is cropped against constraints of trace length. The cropped window length is then used to crop out the corresponding area on the trace itself. The two cropped sections are then cross correlated using equation 1.

$$cc = \frac{\sum_i wvlt \bullet ctrc}{\left(\sum_i (wvlt^2) \bullet \sum_i (ctrc^2)\right)^{\frac{1}{2}}} \tag{1}$$

$$cc = crosscorrelation$$

$$wvlt = userwindowarea$$

$$ctrc = croppedtrace$$

For each sample step, PICKSLE calculates each cross correlation value and checks the number against the ‘percenttolerance’ property. If the cross correlation value at any particular point is greater then the ‘percenttolerance’ property, the value is checked against any previous stored cross correlation values. If the new cross correlation value is larger then a previous value stored, the details of where the windowed area are located and kept for referencing as a pick location. Figure 2 graphically represents the process of scanning and finding the highest cross correlation on a given trace.

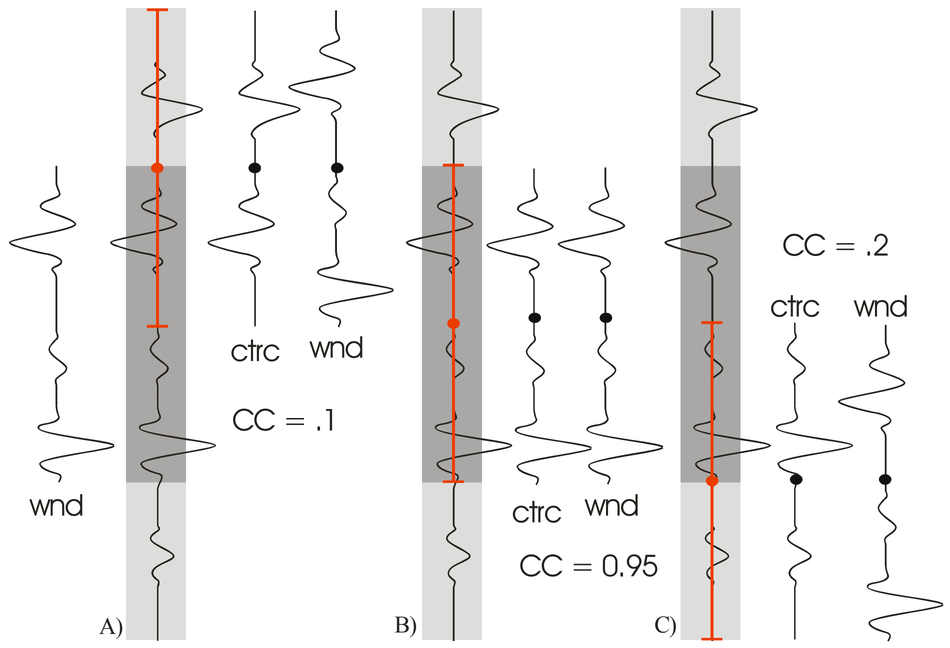

FIG. 2. This figure represents the scanning process that PICKSLE undertakes on a single trace. The initial windowed area that is used as the basis for scanning can be seen on the far left as the ‘wnd’ or window as taken from the dark grey area on A). A) Represents the first cross correlation check with a resulting cross correlation (cc) of 0.1. PICKSLE would keep this initial value and midpoint if the cropped trace cross correlation of 0.1 was greater then or equal to the ‘percenttolerance’ property. If the cross correlation is larger then the ‘percenttolerance’, the value would then be checked against previous stored cross correlations. B) Represents a point half way through the trace scan which results in a cross correlation of .95. This cross correlation and midpoint would replace any value with a smaller cross correlation that PICKSLE may have found during its scanning process. C) shows where the last cross correlation would take place on this scan. The cross correlation of 0.2 at this point would not be kept due to a higher cross correlation already found at point B).

Section Scanning

PICKSLE allows two different ways to choose picks across any number of traces in a section. The first type of section scanning occurs when the users clicks once on the axis where PICKSLE has been activated. A single click picks a horizon across the entire section. To prevent accidental execution of this command, a dialog window prompts for confirmation before proceeding with this (possibly) lengthy operation. The second process to choose picks across a section allows for scanning to occur only within a user defined area. In this case a section would be chosen by clicking on an initial location in a PICKSLE activated axis, holding down and dragging the line that appears to the end location the user desires to be scanned. Figure 3 graphically represents the areas PICKSLE would undertake with both operations.

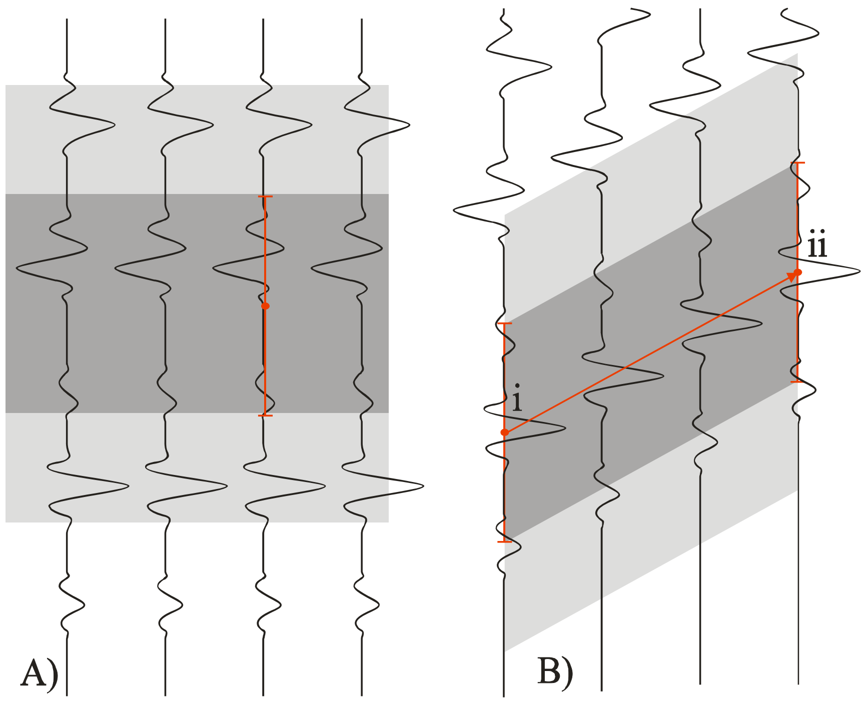

FIG. 3. This figure shows the two ways that PICKSLE undertakes pick scanning. A) Shows the single click PICKSLE activation. It can be seen here by the shaded areas that PICKSLE will scan through the entire section horizontally. Scanning in this case will be carried out from right to left. B) Shows the area that PICKSLE will scan through when the user clicks and holds at (i) while dragging and releasing at (ii). Scanning in this case will be carried from the initial click point to the end click point.

Picks finalization

If a developer or a user has enabled the 'showprogress' property, a windowed section on each traced scanned will appear as a vertical green line while PICKSLE progresses through the scanning process. These green horizontal lines will be plotted on each trace scanned and in order of the scanning. The single midpoint of the picks that are found by suitable cross correlations on each trace are also plotted during this process. When scanning is completed, PICKSLE deletes the lines and midpoints from the axis.

After PICKSLE has cleared the axis of any lines created during its scanning operation, the picks that have been stored for each trace are then reduced. Picks are reduced in by PICKSLE dividing the number of picks found by the ‘nodetolerance’ number. If the ‘nodetolerance’ property is greater then one, a least squares fit is applied to the picks to straighten out the remaining selections.

The picks are returned to the program that called PICKSLE via the ‘transfer’ callback controlled by a developer. The picks are sent in an array format [y x] for each pick found.

Future developments for PICKSLE could possibly include choice of node straightening technique, or none at all depending on preference. It should also be noted that this is an early development of this program and PICKSLE may require more third party usage to work out the bugs.

WINEXTERACTOR

This is the first of the two smaller programs that came about with the development of PICKSLE. WINEXTRACTOR allows users to input a single trace and visually adjust their own windowed area that will be extracted from that trace. In the case of its use in PICKSLE, WINEXTRACTORS allows users to adjust windowed area that would be extracted and used to scan. WINEXTRACTOR can be executed through the following means.

*dataou*t = **winextractor**(*masteraxis,trc,midpt,smples*);

*dataou*t = **winextractor**(*masteraxis,trc,midpt,smples,lockto*);

The “masteraxis” argument is the handle of the main axis in the figure that is calling WINEXTEACTOR. This handle is needed for alignment purposes of the WINEXTERACTOR figure. The new figure is centered overtop of the ‘masteraxis’ and constrained by the monitor dimensions.

The argument “trc” is a single trace array that a user desires WINEXTRACTOR to acquire a window from. The trace can either be a single column of data with only the trace information, or a double column with both the trace information and the depth information. For displaying purposes the trace array is normalized on the WINEXTRACTOR axis.

The third argument ‘midpt’ or midpoint is the central point from where the initial window will be visualized on the WINEXTRACTOR axes. The location of this point will also depend on if the developer or user has also specified the “lockto” argument discussed below.

The forth argument ‘smples’ is the number of samples away from the midpoint that WINEXTRACTOR will generate a window. This argument is a single positive integer that will create a top and bottom of a window equidistant n samples from the midpoint. The top and bottom of the window are constrained by the limitations of the input trace.

The last argument ‘lockto’ is not required but can be quite useful when used properly. This argument defines which nodes on the input trace WINEXTRACTOR will allow the midpoint to lock on to. There are several choices that a user of developer can input to have WINEXTRACTOR lock a midpoint to. These choices are “peak”, “trough”, “zerocross”, “zerocross+”, and “zerocross-“. The ‘lockto’ argument will default to ‘open’ or simply allow users to select any points on an input trace in the WINEXTRACTOR figure. For a more detailed description of this operation, see NODEFIND in this paper.

WINEXTRACTOR figure

Figure 4 shows the WINEXTRACTOR figure as created with proper input arguments.

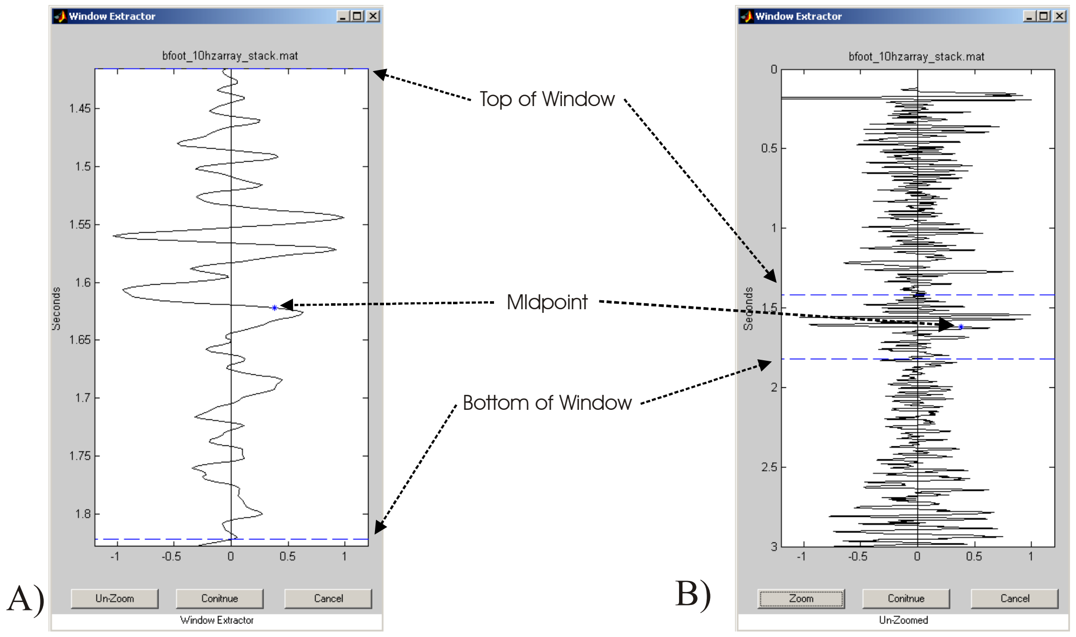

FIG. 4. This shows the figure created when WINEXTRACTOR is initiated. A) shows the axis of the figure as zoomed to the window. The top of the window, bottom of the window and midpoint are in blue. B) shows the WINEXTRACTOR axis when un-zoomed revealing the entire input trace. The window area can be adjusted by clicking and holding on the midpoint, or clicking and holding on either the top or bottom window lines.

The initial WINEXTRACTOR axis is zoomed to fit the window that has been input. This initial zoomed in axis was created to accommodate PICKSLE by allowing users to make small and quick adjustments to a windowed area. Zooming in and out on the axis is allowed however, to give the user more control over the window creation and extraction. Both the top and the bottom of the window are adjustable and stay equidistant away from the midpoint. The adjustment of the window lines is carried out by moving the window lines after clicking and holding the left mouse button on them. The windowed area will not extend beyond the limits of the trace.

The midpoint can also be moved by holding down on the point with the left mouse button. When the mouse moves, the midpoint will follow the outline of the trace. If the ‘lockto’ argument has been specified, the midpoint will only move on to the points as desired by the user. The window lines also move with respect to the movement of the midpoint and will not extend beyond the limits of the input trace. The midpoint motion is also restricted to the limits of the input trace.

The output from WINEXTRACTOR is simply two values, the midpoint and the number of samples between the midpoint and the largest window line.

NODEFIND

The last of the three programs described in this paper is NODEFINE. Unlike the other two programs, this program runs independently of the others. This program allows developers or users to find desired types of points on a single array of data. NODEFIND is meant to be used in conjunction with other programs but can be used on its own if arguments are set up properly. The following shows the various ways NODEFIND can be activated.

dataout = **nodefind**(*trc,midpt,lockto*);

dataout = **nodefind**(*trc,midpt,lockto,nhood*);

The argument “trc” is a single trace that the developer or user wishes NODEFIND to interrogate for its desired points. NODEFIND does not adjust the input trace in any way, so one must ensure that the original trace is evenly sampled before using the array as an argument.

The second argument “midpt” or midpoint refers to the initial point from which the user desires NOEFIND to begin its searching process. The last argument ‘nhood’ is the number of samples away from the midpoint the user wants NODEFIND to check for the desired ‘lockto’ points. Both these argument were developed into NODEFIND due to its original use with WINEXTRACTOR. In a future release of NODEFIND, a simple two-argument activation without midpoints or samples will be considered.

lockto

The ‘lockto’ argument is the major reason why the development of NODEFIND was undertaken. This argument defines which nodes on an input trace a user desires NODEFIND to search for. At present, there are several choices that a user or developer can input to have NODEFIND isolate. These choices are “peak”, “trough”, “zerocross”, “zerocross+”, and “zerocross-“. Each of these choices is represented in Figure 5.

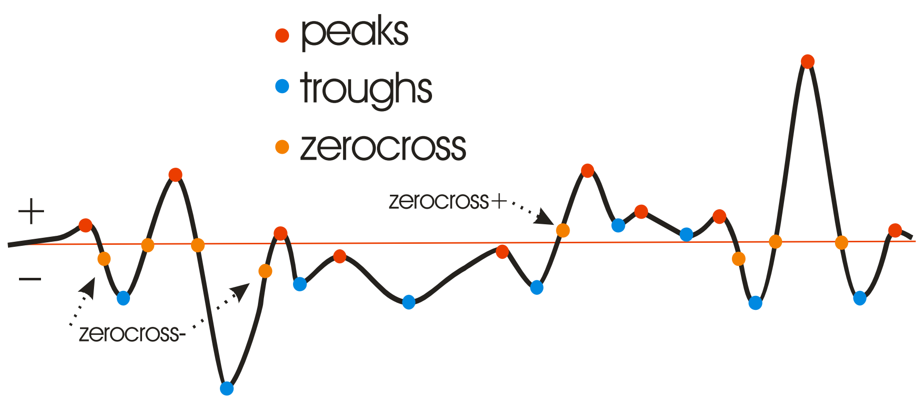

FIG. 5. The three types of choices that NODEFIND allows for isolating on an array of data. The red dots show where NODEFIND would find peaks. It can be seen that the peaks would appear on both the positive and the negative side of the trace. The blue dots show where NODEFIND would find troughs. As with the peaks, NODEFIND will also find troughs on both the negative and positive sides of the array. The zerocross selections can be seen in orange. NODEFIND will accept zeros on the traces as well as points that cross over from positive to negative or negative to positive without touching zero, as zeros.

NODEFIND breaks down the input trace into the desired node choices by using the MATLAB routine DIFF. This DIFF routine does an approximate first order derivative on the inputted trace by taking the array X with n elements and creating a new array Y=[X(2:n,:) – X(1:n-1:)]. Figure 6 shows graphically how peaks and troughs are found using DIFF.

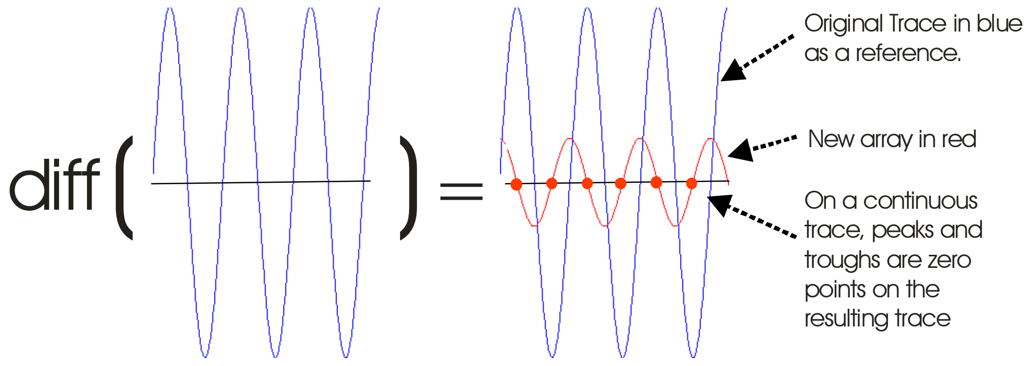
diff
Original Trace in blue as a reference.
New array in red
On a continuous trace, peaks and troughs are zero points on the resulting trace

FIG. 6. The MATLAB DIFF function is applied to a single continuous trace. The resulting array used by NODEFIND to acquire peak and trough information. The in red shows that peaks and troughs on the original trace show up as zeros (red dots) on the resulting array. The result in this case is slightly deceptive due to DIFF originally being applied to a continuous array resulting in a another continuous array. From the figure we can see however, that if the trace was not continuous, peaks and troughs on the original array would still correspond with polarity changes on the resulting array. Peaks go from positive to negative on the resulting array, while troughs go from negative to positive.

As Figure 6 shows, NODEFIND isolates peaks and troughs resulting from the DIFF operation by searching for negative to positive responses in the case of troughs, or positive to negative responses in the case of peaks. To find these points NODFINE isolates only positive responses when searching for peaks, and negative responses when searching for troughs. On the isolated array, peaks and troughs will appear as nonconcurrent samples when checked against the original trace. It should be noted that whenever DIFF is applied to a trace one sample is lost at the end of the resulting array, due to the new array being Y=[X(2:n,:) – X(1:n-1:)].

The 'zerocross' options initially isolates all the points at which the input trace goes to zero. The positive-to-negative or negative-to-positive "zeros" are found by NODEFIND through the DIFF routine use as used in searching for peaks and troughs. In this case, to find the 'zerocross' points, the original trace is converted to a checking trace by making all positive values on the original trace equal to +1 and all the negative values on that trace equal to -1. Applying DIFF to this new checking traces results in the discovery of new "zeros" on the original trace. Figure 7 graphically shows the process for finding "zeros".

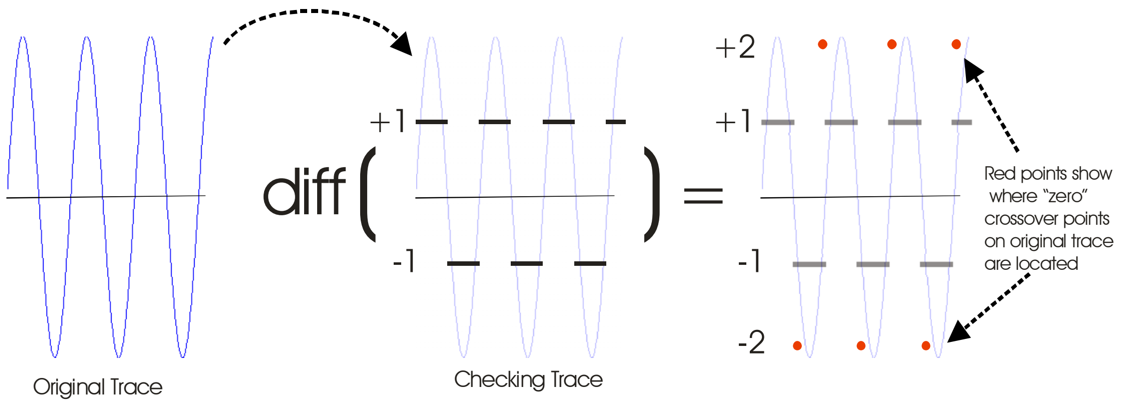
Original Trace
diff
+1
-1
Checking Trace
=
+2
+1
-1
-2
Red points show where "zero" crossover points on original trace are located

FIG. 7. This figure shows the process that NODEFIND goes through to acquire potential "zerocross" points on a trace when no actual zero points are available. The original trace is first changed so that all positive values are set to +1 and all negative values to -1. The DIFF routine is applied to the new checking array, which results in single points of +2 or –2. These single points correspond to crossovers from negative to positive or positive to negative on the original trace, essentially the new "zero" points.

The NODEFINE option 'zerocross' finds all zeros and both positive and negative cross over locations. The two other 'zerocross' options offer the user more control over exactly which of these types of "zero" crossings are desired. These options are 'zerocross+' and 'zerocross-'limit NODEFINE to only searching for and releasing from a trace "zero" crossing points either only positive in the 'zerocross+' case, or negative in the 'zerocross-'case.

Future developments for this program would include NODEFIND searching for inflection points.

SUMMARY

The three programs in this paper were all developed out of the original desire to have an automatic line picker available to the CREWES MATLAB library. While all three programs are in their early developmental stage, each program is fully functional, and, if needed, changeable. It is encouraged that developers find uses for both WINEXTRACTOR and NODEFIND beyond their present use in PICKSLE. PICKSLE has been integrated into the newest release of PLOTIMAGE and it is encouraged that PICKSLE be tested vigorously to find any problems that may need to be resolved.