CAMBRIDGE ELECTRONIC DESIGN LIMITED

eNEWSLETTER

#14
September 2021

Contents

Welcome

Thank you for downloading our September newsletter. We hope you have had a wonderful summer and would like to welcome everyone back for the start of another academic year.

A minor fix for Spike2 was released in update 10.11a in August; update your copy here. We are also working hard at bringing you more new features in update 10.12 which should be released soon.

The new Digitimer D360R differential amplifier was released earlier this year, and we are happy to announce support for this device is coming soon to Signal and Spike2. The D360R is a multi-channel isolated research amplifier intended for electroencephalography (EEG), electromyography (EMG) or evoked potential (EP) studies. Combined with a powerful CED1401 data acquisition device and CED software, you will have one of the most versatile systems available.

We would also like to congratulate Dr Monica A. Perez for recently being awarded an NIH R35 Outstanding Investigator Award for her research into spinal cord injury (SCI). Dr Perez is an internationally recognized leader in SCI research at the Shirley Ryan AbilityLab in Chicago.

Training

Full recordings from this year's earlier online training sessions may be viewed on our website. We have more sessions planned for later in the year, for which details will be shared nearer the time.

We also offer remote training sessions by Skype/Zoom/Teams either one-to-one or with groups. Join us and learn how to make the best use of Spike2 and Signal to save hours of repetitive analysis. Our sessions are free to arrange and are suitable for both existing and prospective users of our data acquisition and analysis systems. If you would like to schedule a session, please get in touch: Marjorie@ced.co.uk.

If you see this button in our newsletters, it means a file or script relating to the section is available to download:

Latest versions of Spike2 and Signal

Spike2	Released	Signal	Released
Version 10.11a	08/2021	Version 7.06	04/2021
Version 9.14	06/2021	Version 6.06	04/2021
Version 8.22	06/2021	Version 5.12a	02/2018
Demo	08/2021	Demo	04/2021

Back to contents

Here to help

We know access to laboratories has been erratic for the past year, but with the current lockdown measures easing for some we hope that conditions will continue to improve. We are now able to offer site visits when necessary, however we will continue to support remotely in the first instance. CED will also continue to do all in our power to support you for increased home working. Should you require any help or wish to discuss your system, email Marjorie@ced.co.uk and we can arrange for a video call via Skype/Zoom/Teams.

We also have tutorial videos for both Spike2 and Signal available on our website for you to peruse at your leisure. There are new videos and updates in the pipeline. If you have a particular topic you think could benefit from a tutorial video, please let us know. All these videos are also available through our YouTube channels: Spike2 / Signal.

Try the CED Forums bulletin board for further software and hardware support. Ask your questions and receive valuable input from our super users.

Back to contents

Script Spotlight – Accessing channel data

Scripters often need to access values or arrays of data from channels, either to transform the data or perform a dedicated analysis function. For data files and memory views in Signal and result views in Spike2, this is achieved using the `View(vh%, c%).[]` form of the View() script command. This form gives access to the channel data as an array where vh% is the view handle of the file and c% is the channel of interest. See the in-software help topic Script language > Script language syntax > Result views as arrays (Spike2) or Data views as arrays (Signal) for more information and examples.

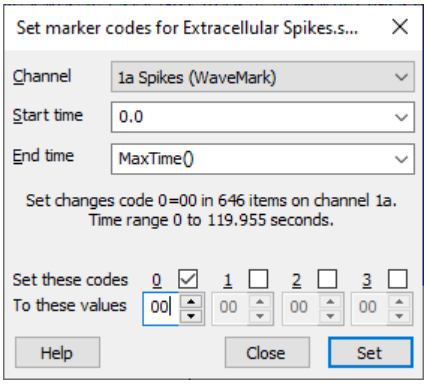
For Spike2 data files there is a dedicated command that will fill an array with data values from a waveform or RealWave channel, or with event times from all other channel types: `ChanData(chan%, arr%[], sTime, eTime)`. For this command chan% is the channel to read data from and arr[] is either a real or integer array to fill. Real arrays collect the waveform values in user units as set by the y-axis, so one would most often use a real array here. Integer arrays collect waveforms in ADC units and event times in the underlying time units (as returned by Binsize()). sTime and eTime set the start and end times in seconds to collect the data from.

Back to contents

Getting started with Spike Sorting – Clustering and PCA

Clustering

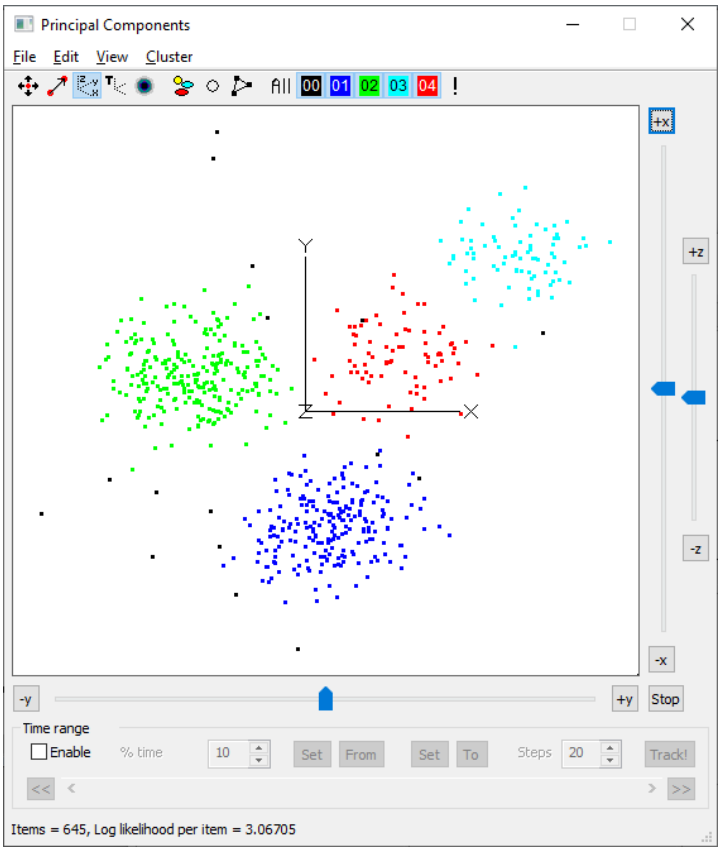
Clustering is the generic term for methods that group similar objects into classes. For Spike2, the objects are spikes that consist of 1, 2 or 4 waveform traces of typically 20 to 40 data points each. If you have n data values that define each object, you can cluster in n dimensions. However, if n is greater than 3 it is difficult to visualise the clusters. For spikes, n is typically 30 or more, so we need methods to extract 2 or 3 independent measurements from our data that maximise the differences we care about between the classes and minimise the differences due to noise in the data. Spike2 provides four methods for this: Principal Component Analysis (PCA), Feature Measurements, Template Correlation, and Template Errors. We suggest you start by trying PCA, as this avoids you needing to make subjective decisions about which features of your data are more important than others. It also tends to produce clusters that are suitable for use with automatic clustering methods.



How to use PCA to create clusters

For beginners, the example data Extracellular Spikes.smr can be used to experiment on. This is included with your Spike2 installation, usually found in C:\Users\User\Documents\Spike10\Data. First set all template marker codes to 00 by right clicking the data and opening the *Set Marker codes* dialog.

Next, navigate to the *Edit WaveMarks* dialog (right click the data) and delete any current templates. Drag the black triangles at the top of the data display area to select the region of each spike to process. Ideally you should exclude baseline regions from the spikes to reduce noise and improve the cluster separation. You can also optimise the data display with the *Home* key or  button. The clustering analyses operates on data in the time range set with the  button; ensure this is set from 0 to MaxTime() unless you wish to exclude data.



With the setup finished, use the *Analyse* menu > *Principal components* (shortcut Ctrl+A) to open the *PCA parameters* dialog. If your spikes have a single trace, everything except the *Normalise measurements* check box is disabled. With more than 1 trace (stereotrode and tetrode data), all dialog items are enabled. Full information on these options is accessed from the *Help* button, or by pressing F1. Click *OK* to open the clustering dialog.

The clustering dialog is linked to the *Edit WaveMark* dialog and will be closed if any changes are made to *Edit WaveMarks*. The *clustering* dialog behaves almost identically for all analysis methods; it contains a menu, a toolbar that selects what to display, the cluster window, sliders for rotation around the x, y and z axes, the *Time range* area, and a status line at the bottom.

All clustering methods generate a table of values with one row per WaveMark. Each table row holds the spike time, a class code and the x, y and z values derived from the spikes. All clustering operations operate on the data in this table, not on the original spikes in the data file. Therefore, we call the items in the cluster window *events* to distinguish them from the original spike data. If you click on an event in the cluster window, the *Edit WaveMark* dialog will jump to the WaveMark that generated the event. Furthermore, Cursor 0 will jump to the WaveMark in your time view if *Track Cursor 0* is on.

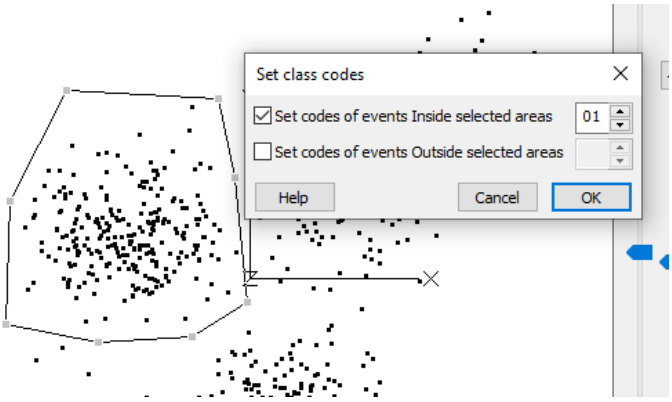
There are four options to assign codes to clusters; manually (with ellipses), K means, Normal mixtures, or Match to classes. K means and Normal mixtures are fully automated using their respective algorithms, whereas Match to classes is considered a semi-automatic method which may be used in conjunction with any of the other methods. Typically, if you have well defined clusters you should try K means or Normal mixtures first. If your data is chaotic then you may need to manually define your cluster centres, and then use another method to finish. Any changes made to the clustering dialog have no effect on the original data until you use the *File* menu > *Apply* command.

Manual

The clustering dialog contains tools that make it easy to visualise the clusters in two and three dimensions and tools that allow you to classify the clusters manually. However, if you are not yet confident with clustering, we suggest you first begin with one of the algorithm methods like Normal mixtures.

The density plot may be used to identify dense centres of clusters. Use this in conjunction with the *View* menu > *Density Colour Map* to alter the colour scale (the thermal pre-set is a common favourite). The X, Y and Z sliders may be altered to view the clusters in 3D. This may help identify clusters hidden in the background, however Spike2 usually orientates the cluster window to show the best separation when first opened. We recommend you experiment with the tools to familiarise yourself; further details on the cluster window commands are found in the in-software Help (F1).

To manually assign codes to a cluster, ellipses are used to group events. The  button will add an ellipse to the dialog which can be dragged and resized as needed (alternatively right click > Place user ellipse to select small/medium/large). If the ellipse of 4 points is not flexible enough, you may draw your own shape with the  button. Hold the *Alt* key and click to place the first point, release *Alt* and continue adding points to complete your shape. This shape may also be dragged and resized as needed. Once your ellipse/user-defined shape covers the events you wish to group, right click and select *Set codes...* to open a new dialog. Assign the new codes for events inside or outside the shape and click *OK*. Your events are now coded and coloured appropriately; go to *File* menu > *Apply changes* to create your spike templates and assign to your WaveMarks.



K means

The *K Means* algorithm is a well-known and fast clustering method that is effective when the number of clusters is already known, the clusters are spherical and are of a similar size. Therefore, if the clusters are not spherical (or cannot be made spherical by scaling) or are of very different sizes, K Means will not give useful results.

The *K Means from existing* command uses existing cluster centres (set manually) as the seeds for the K Means algorithm. The K Means command runs the algorithm 10 times (or until you stop it) with random seeds and chooses the result that has the best ratio of cluster separation to cluster size. You can find details of the algorithm in the software *Help*.

Normal mixtures

The *Normal Mixtures* algorithm assumes that the probability density of data points around each cluster centre follows a multivariate normal distribution. This is a more general approach than the K Means algorithm, as it does not require spherical clusters and can often give results that seem more intuitively correct. However, it is a more complex algorithm than K Means and takes noticeably longer to run.

The *Normal Mixtures from existing* command uses the existing clusters (set manually) as the seeds for the Normal Mixtures algorithm. The Normal Mixtures command runs the algorithm 10 times (or until you stop it) with random seeds and chooses the result that has the maximum likelihood of fitting the data.

Match to classes

This method uses the current cluster statistics to assign visible events to the nearest visible cluster and mark outliers as code 00. The typical use of this command is to clean up manual cluster assignments in situations where the K Means from Existing or Normal Mixtures from Existing commands would make too drastic a change.

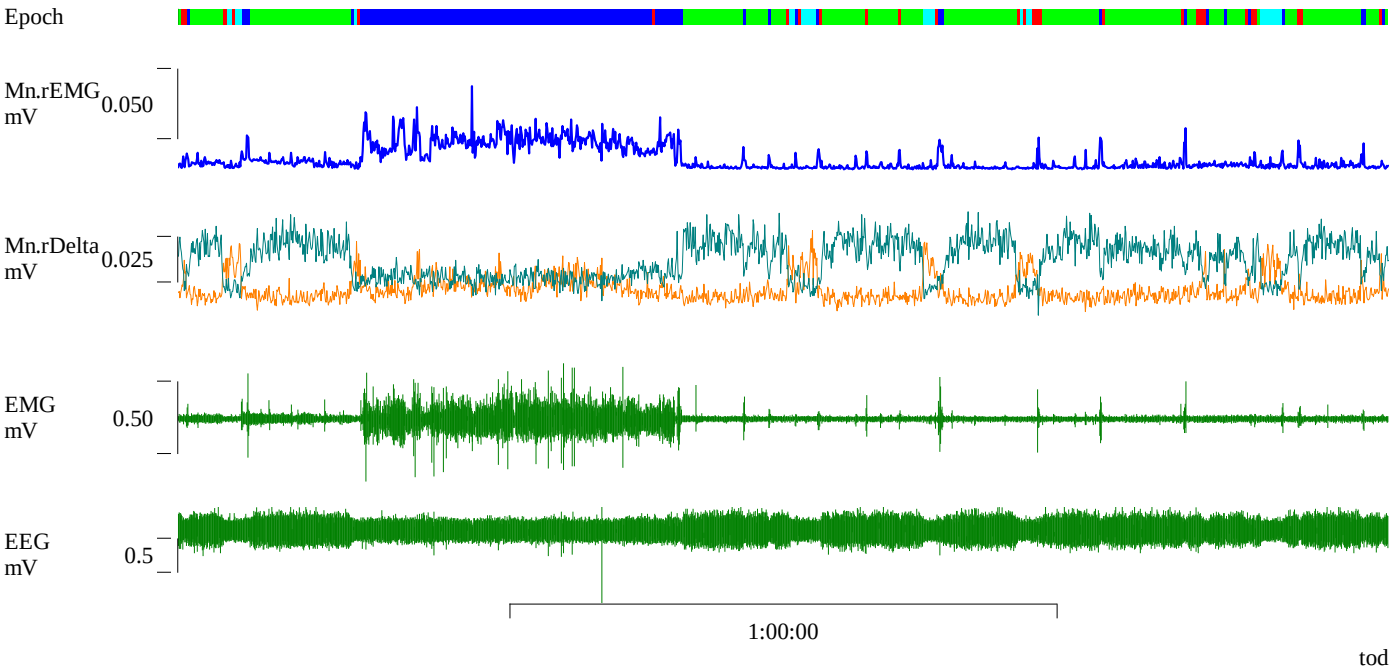
The basic idea is that each visible event is assigned to the cluster that it is most likely to belong to. The shape of each cluster is considered, so a cluster need not be spherical or aligned with the X, Y or Z axes. Probability is measured in terms of the Mahalanobis distance, which is the multi-dimensional equivalent of standard deviation.

Once you have finished, go to *File* menu > *Apply changes*. When you apply the results of the cluster analysis, the *Edit WaveMark* dialog will re-evaluate its templates to match the new classifications.

Back to contents

Scripts: Spike2

Our script writers have recently updated our Rat Sleep Auto.s2s script. This script automatically assigns sleep scores (Wake, NREM and REM sleep) based on hippocampal EEG and nuchal EMG recordings from a rat. The script presents the sleep score in a 'State' channel (TextMarks in the *state* draw mode) with each state differentiated by colour. The script implements the method of Costa-Miserachs et al (2003).

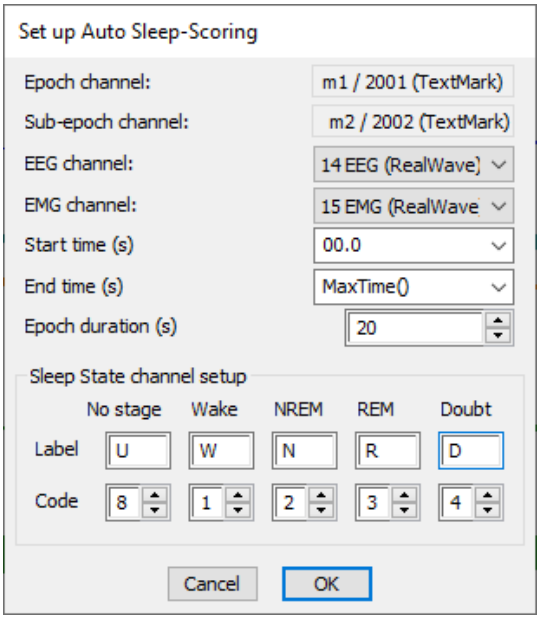


Full instructions are included with the script which can be accessed through the *Help* button on the script toolbar, however, we recommend you first read the original paper before proceeding. The starting point is one channel of hippocampal EEG and neck muscle EMG per rat. Staging accuracy will be low/impossible unless the recording is long

enough to provide clear episodes of all three sleep states. The script generates Delta and Theta wave channels by applying digital FIR filters to copies of the original EEG channel. The filter characteristics are:

Delta: 0.55 - 4Hz (-3dB points 0.38 Hz and 4.17 Hz. Transition gap 0.5 Hz to -65 dB).

Theta: 6 - 10Hz (-3dB points 5.54 Hz and 10.17 Hz. Transition gap 0.5 Hz to -65 dB).



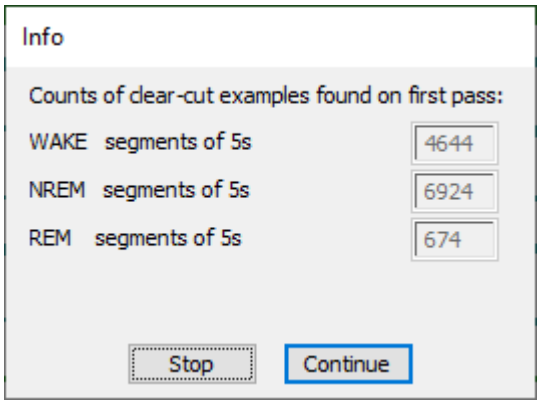
EEG, Theta, Delta, and EMG channels are first rectified; the mean value of each is plotted in a new channel in bins 1/4 of the epoch duration specified by the user (the default value being 20 seconds equalling 5 second bins). A virtual Theta/Delta ratio channel (T:D) is also created based on the rectified Theta and Delta channels. The mean and standard deviation of these channels are used together with the below pre-set values to generate criteria to identify the 5 second segments most clearly assigned to WAKE (W), NREM (N) and REM (R) as described in Fig 2 step 1 of Costa-Miserachs et al (2003). For example, bins with T:D > 0.9 and EMG > mean EMG were scored as W.

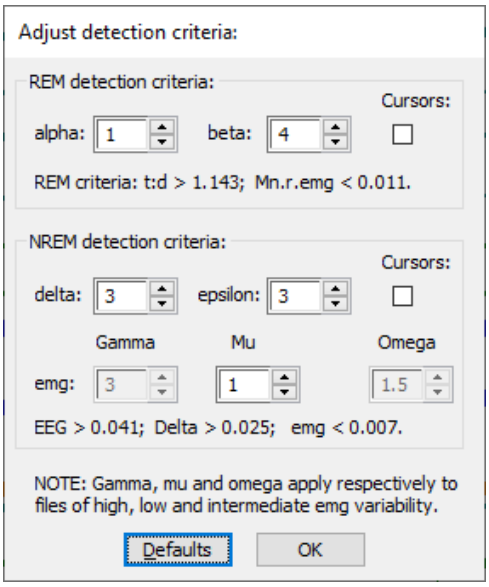
These pre-set values are listed at the beginning of the script code:

```
var tdthr:=0.9;            'threshold T:D used to distinguish definite WAKE and
                           NREM (step 1 Fig 2 and Fig 3 part 7)

var REMthrTD:=1.1;         'threshold T:D used for detecting definite REM (part 3
                           of Step 1 in Fig 2)

var artcoef:=1.5;          'multiple of SD of all EEG used to detect artifacts
                           during WAKE (Fig 3 part 1)

var sdemgmulNREM:=0.1;     'factor used to define definite NREM step 1 part 2

var sdemgmulREM:=0.3;      'factor used to define definite REM step 1 part 3 d

var sdemgWmul:=1.5;        'factor used in defining possible REM with high EMG as
                           DOUBT in part 3 of Fig 3

var NREMemgvarthr1:=1.6;   'EMG mean/SD NREM ratios used to distinguish between
var NREMemgvarthr2:=2.1;   files with high, medium and low EMG variability
```

These may be altered if you wish. However, this should only be attempted after a careful reading of the original article to understand the consequences of any changes. Random alteration is likely to produce nonsensical results.

The script provides a warning message if the count of segments of any sleep stage is zero after the first pass through the data. You will have the option to abandon the analysis or to continue, understanding the quality of the results will be questionable if you proceed. Mean and SD values of the relevant data channels during the segments of data scored as W, N and R are further used to create less stringent criteria (Fig 3 of Costa-Miserachs et al (2003)) for assigning scores to bins that were not identified on this first pass. The threshold levels used may be adjusted by changing the values of the coefficients (alpha, beta, delta epsilon, mu and omega) in a user dialog. There is an option to mark the threshold levels with horizontal cursors. When this second phase finishes all segments will have been scored. Near-miss REM bins are scored as DOUBT (D) and segments that did not meet any of the main criteria are scored as W by default.





In the final stages (steps 4 and 5 of Costa-Miserachs et al (2003)), the segments are combined in groups of 4 to create epochs. The score assigned to them is, in most cases, the state of most of the segments that comprise it. Where there is no majority the epoch as scored as DOUBT. In some cases, these DOUBT epochs may be re-assigned based on the score of the previous or following epoch.

The script allows you to generate multiple scores with different coefficients, and on each pass generates a report holding information to help you to refine the settings. You can also compare multiple scores and generate an agreement matrix. The sleep score can be edited manually, for example, to replace DOUBT epochs that could not be scored automatically with your own assessment. You can also generate power spectra for individual sleep states and generate tables of sleep statistics with the data subdivided into relevant time periods, for example hour by hour.

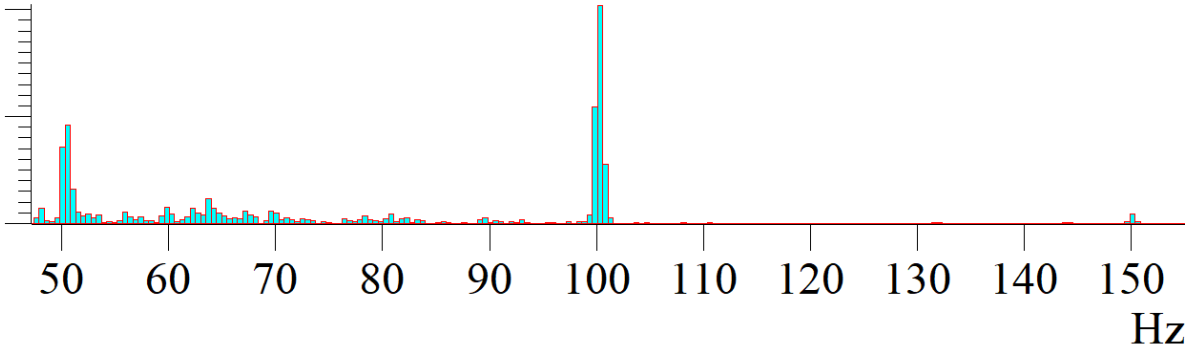
Back to contents

Signal

How do I remove mains hum from a recording?

Sometimes we are unable to fully shield our recordings from electrical interference. A typical source of interference is mains power, and ideally this interference should be removed at the source by identifying and eliminating earth loops, using Faraday cages, and improving shielding. However, in many cases there remains a signal that is linked to the mains frequency (50 or 60 Hz, depending on where you are in the world). It is possible to use digital filtering techniques to eliminate the 50 or 60 Hz bands. However, mains-related interference is never pure 50/60 Hz cycles; there are variable components at higher harmonics of the mains frequency. A single notch filter inevitably distorts the signal and multiple filters are needed to remove the higher harmonics in the data.

You can easily identify mains hum by performing a power spectrum of your data. Go to the *Analysis* menu > *New memory view* > *Power spectrum* to identify the components:

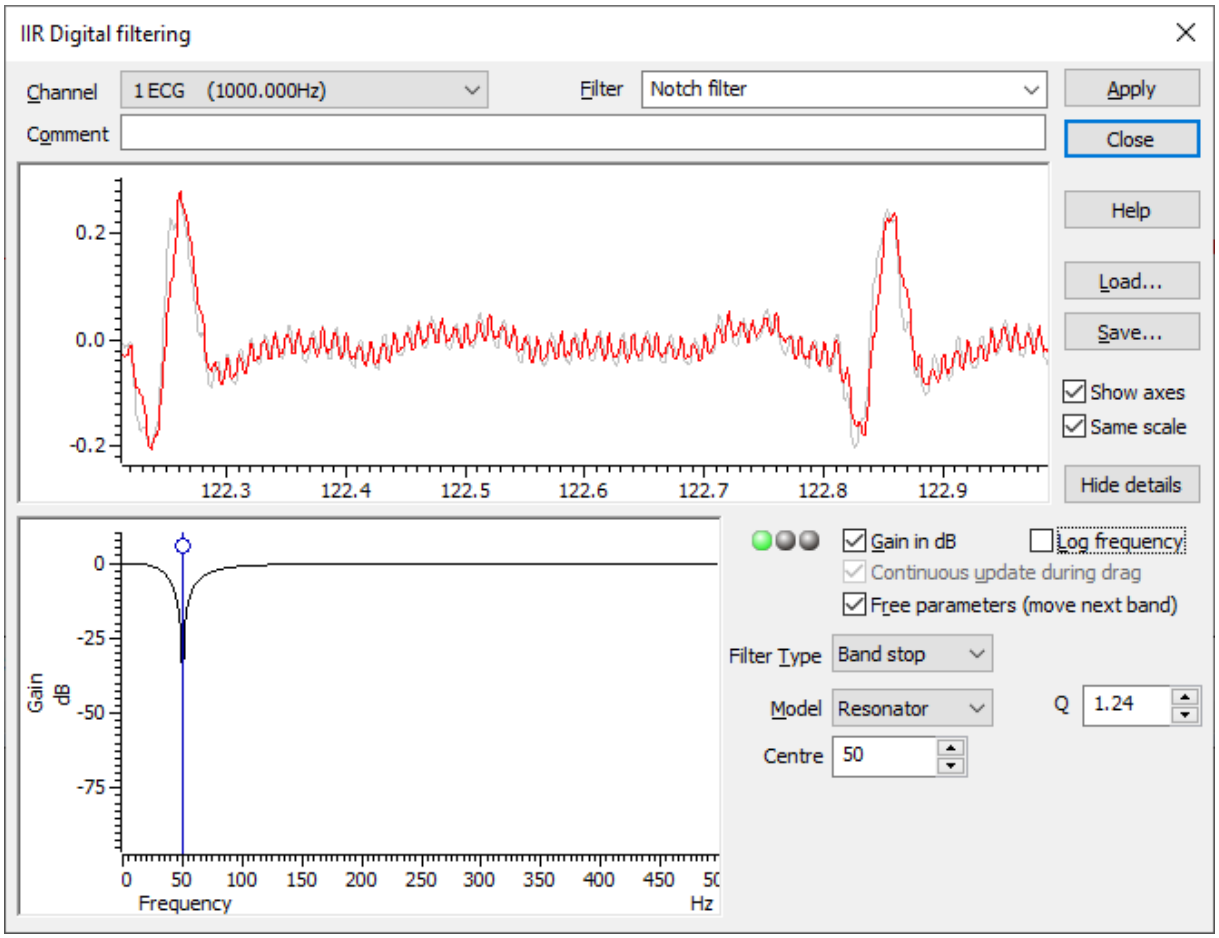


Here we have run a 2048 FFT with a Hamming window on some ECG data, from which you can see the mains interference around 50 and 100Hz, with a smaller component at 150Hz.

Nevertheless, Signal provides both Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) type filters to improve your recordings, however slight. These filters remove unwanted frequency components from waveforms and can also be used to differentiate a signal. To access the digital filtering tools, go to the *Analysis* menu > *Digital filtering* and select either FIR or IIR (alternatively right click your data and select a filter from the menu). This will open a new dialog for you to set up your desired filter.

There are general guidelines to select which filter depending on your application. As mentioned above, a notch filter may be used to eliminate single frequency signals. A notch filter is also defined as a band stop resonator, a type of IIR filter in Signal. Within the IIR dialog, use the *Filter* drop down menu to select the example notch filter and click *Show details* to see the filter details. Resonators are defined by a centre frequency and a Q factor. The Q is the centre frequency divided by the width of the resonator at the -3 dB point. The Q is adjusted by dragging the small circle or with the Q field to the right of the displayed frequency response.

When designing your filter, keep in mind that the notch should not be too narrow as mains hum does not manifest as pure frequency but has a slight spread of frequencies around its centre value as seen in the power spectrum above. For notch filters, the higher the Q, the narrower the notch. To really tackle mains hum with digital filters you will likely need to set notches at the first few odd harmonics of the mains frequency (50/100/150Hz or 60/120/180Hz). The example below has a very low Q (1.24) to make the filter response visible.



The filter dialog uses a traffic light system; they show green if the filter appears to be OK, amber if the filter may be unstable, or red if the filter calculation failed, the filter is unstable, or if one of the input parameters is illegal. There will usually be an explanatory message in the lower right-hand corner of the dialog explaining the problem. In the amber state, the filter may still be usable; you can look at the frequency response and the filtered data to see if the result is acceptable.

You can have a band stop (notch) or a band pass resonator filter. A band pass resonator is the inverse of a notch and are sometimes used as alternatives to other narrow bandpass filters. The higher the Q set for a resonator, the longer it will take for the output to stabilise at the start of the filter output.

Once you are happy with your filter design, click *Apply* and choose which frames to apply the filter to. The data for the relevant channels will be overwritten, so make sure you are comfortable with the filter before proceeding. Check the frequency response and the filtered data to see if the result is acceptable. If in doubt, create a copy of your data before applying any digital filters.

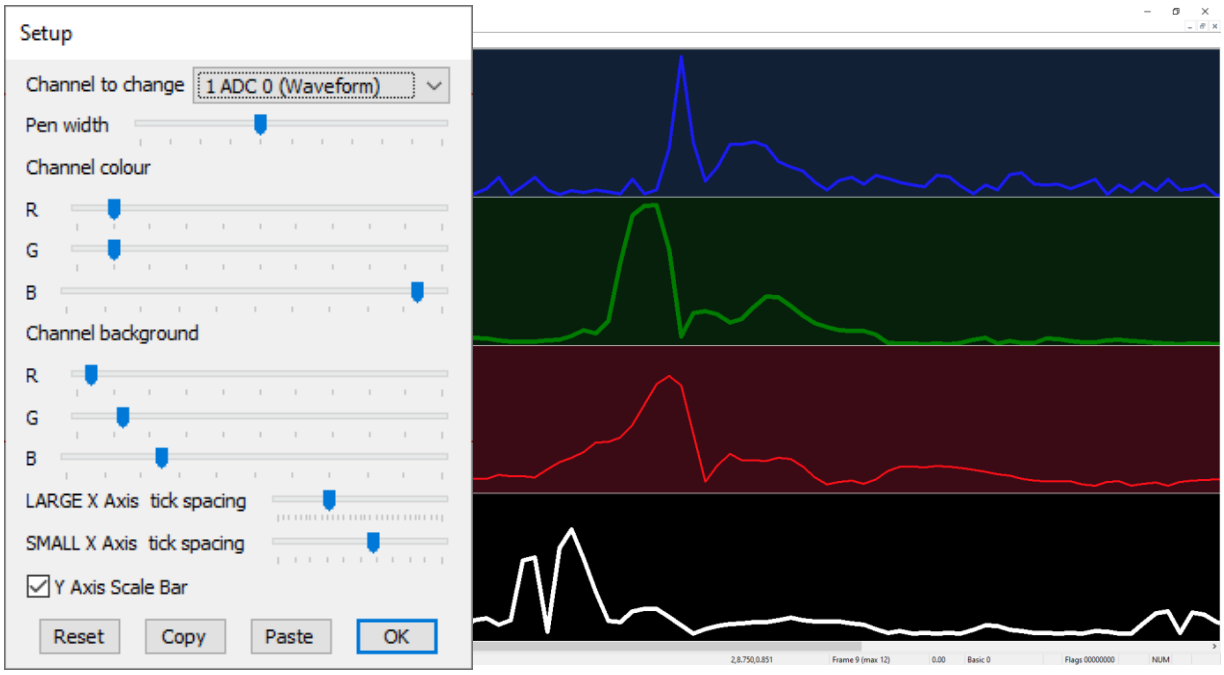
Back to contents

Scripts: Signal

There are many options within Signal to alter the format and presentation of your data. For example, the colour palette is used to alter view, channel, and text colours. The font selection is used to select the font style and size. The *View* menu > *Customise display* dialog controls the channels to display in a data or XY view, the display of x and y axes, channel numbers, grids and of the horizontal scroll bar, plus controls for horizontal Y axis labels. These are only a few examples, with many more available. Views altered dramatically can always be set back to the default settings quickly with the *View* menu > *Standard display* command.

All formatting options may also be accessed through the scripting language. We have put together a small example script that groups a few channel options together to showcase how it may be done. The attached script, ChannelAlter.sgs, combines the pen width, channel colour, channel background colour, X axis tick spacing, and Y axis scale bar options into one dialog.

The dialog features a channel selector and pen width slider to change the width of the data. The dialog also presents the colour options as sliders for Red (R) Green (G) and Blue (B) to scale as desired.



Dialogs such as this can be tailored to you, using the options you use most frequency. We welcome suggestions for how it may be expanded upon, and please feel free to also send us any other ideas for scripts you desire to Marjorie@ced.co.uk.

Back to contents

Scripters Corner – Obtaining the correct view examples

In our first scripters corner topic we introduced view handles and discussed the importance of ensuring your script is operating from the correct view. This time we provide more detailed examples of locating a view now that we have covered more of the scripting language.

Current view, front view and focus

The script language always has a current view. This is the window that any script command that applies to a window will use. The current view can be set explicitly by the View(vh%) command (where vh% is the handle of a view) and is also set implicitly by commands that create a new view, such as FileOpen() and by FrontView(vh%).

There is also the concept of the front view, which is the Time, Result, XY, Grid or text-based window that is active (at the front); the active view determines the contents of the menu bar at the top of the Spike2 application window. The front view is available with the FrontView() script command. When used with no arguments it returns the handle of the active Time, Result, XY, Grid or text-based window. You can use it with a view handle argument to set the active Time, Result, XY, Grid or text-based window and make it the current view. If you use it with a different window type, for example a Spike2 spike shape or Multi-media window, it moves this window to the top (so it is visible), makes it the current view, but does not change the active view (so the menu bar contents do not change).

Finally, there is the view handle that has the keyboard focus. This is the window that has the first chance to grab typed characters. The FocusHandle() script command returns this window. Unlike FrontView() (which only returns Time, Result, XY, Grid or text-based windows), this can return a script-controllable window of any type.

Example 1

The first example just accepts the current active view:

```
var vh% := FrontView(); 'get the active view handle
```

If you want a view of a particular type (for example a Time view) you will need to check the view type:

```
var vh%:=FrontView();
if ViewKind(vh%)<>0 then
    Message("Current view not a time view|"
            "Close all other views and bring correct view to the front.\n"
            "Halting script");
    Halt;
endif
```

The ViewKind() script command returns the current view type as an integer (0 for time views). Using this we can check the view handle and inform the user if the type is incorrect, forcing the user to run the script again after they have selected the correct view. Note the use of | in the message to set a message box title. We could go further and return the name of the view instead:

```
var vh%:=FrontView();
var text$:=WindowTitle$();
var err%:=Query("Is this the correct view?\n"+text$);
If err%=0 then
    Message("Halting script|Bring the correct view to the front");
    Halt;
endif
```

The Query() script command is often useful for questions with only two branches (usually yes or no). In its default form, pressing Yes or Enter on your keyboard returns 1, No returns 0. All the examples above end abruptly with Halt, forcing the user to correct and rerun the script. FrontView() is often beneficial for scripts written for your own use, as it is quick and you will know what is required. For scripts written for use by your colleagues it may be better to go further and create a dialog allowing the user to select from a list.

Example 2

A more complicated example that accepts a time view (if only one), or prompts the user to select one from all the time views. We have written this as a function so you could use this from multiple places in a script:

```
'Returns a time view handle, 0 if none set
Func SelTimeView%()
var vh%, Titles$[1], i%, ok%;
var select% := 1;               'Selected view index in list
var List%[2];                   'Space for size and 1 handle
var n% := ViewList(List%[], 1); 'Get count and first handle
if n% <= 0 then Message("No time views"); return 0 endif
if n% > 1 then                  'if multiple time views...
    resize List%[n%+1];         'Ensure space for all views...
    resize Titles$[n%+1];       '...and for all view titles
    ViewList(List%[], 1);       'Get all the time view handles.
    Titles$[0]:="Select view";  'Element 0 has dummy text
    for i%:= 1 to n% do         'fill array with window titles
        Titles$[i%] := View(List%[i%]).WindowTitle$();
    next

    DlgCreate("Choose a view");  'Create a user dialog
    DlgList(1, "View", Titles$[]); 'choose from titles
    ok% := DlgShow(select%);     'Display the dialog
    if (ok% = 0) or (select% = 0) then
        Message("User cancelled selection, no view");
        return 0;
    endif
```

```
endif
vh% := List%[select%];          'get the view handle
View(vh%);                      'set as current view
return vh%;                     'return it
end;

var th% := SelTimeView%();      'Select a view or return 0
```

Some notes on how this works:

- `List%[2]` is an array to hold view handles. The first call to `ViewList(list%[], 1);` gets a list of views of type 1 (Time views). `List%[0]` is set to the number of returned handles (which can only be 1 or 0 as there is no room for more), any remaining space is set to handles. The function returns the number of matching views (not limited by the size of `List%[]`).
- If 0 views are found, we are done and we exit, returning the value 0.
- If only one time view is open we need not go through the entire dialog selection, so use an `if...endif` branch to skip the dialog. The handle is at index `select%` (1) in `List%[select%]`.
- If there are multiple possible views, resize arrays List% and Titles$ to have enough space for all of them. Call ViewList() again, and this time get all the view handles.
- The array filled with ViewList() only holds view handles, therefore it is not useful to use this list in our dialog as it likely means nothing to the user. We create a second list of window titles with the for...next loop. Element 0 is given a dummy so the element order of the Titles$ array matches the order of view handles array (i.e. List%[1] view handle goes with the window title in Title$[1]).
- Lastly, we create a dialog asking the user to select a window by title. The select% variable in DlgShow() returns the position selected from the Title$[] list. If the user presses cancel or tries to select position 0 (our dummy text), the script sends a message to the user and halts the script. If they press OK having selected a window title, select% is updated and used to define the vh% variable and set it as the current view.
- As it stands, the script uses View(vh%) to set the current view. If you wanted to bring this to the front you could replace this with FrontView(vh%).

Back to contents

Recent Questions – *Is it possible to duplicate my recording window to show different channels on two monitors?*

Spike2 and Signal both support multiple monitor setups, allowing the application window to be spread across two or more screens. The number of monitors is only limited by the capabilities of Windows and the driving graphics hardware.

To spread the application manually, press the Windows key and left arrow to lock the application to the left-hand screen. Click and drag the right-hand edge to the right to fill both screens. When you exit Spike2, a configuration is updated with the new window size and the application will be restored to this size the next time it is run. Alternatively, you can use the scripting language to spread the application window across the entire desktop:

```
View(App()).Window(0,0,100,100,0); 'Application uses all desktop, including space reserved by system
View(App()).Window(0,0,100,100,1000); 'Application uses all desktop, less space reserved by system
```

Go to the *Script* menu > *Evaluate* dialog (Ctrl+L) and enter either of the above script commands, then press *Execute* to quickly expand the application window. If one of your screens is a different resolution you may find some of the application window is off screen on your smaller monitor. To solve this, you can either amend your screen resolution

so it is the same size (in Windows 10, right click desktop > display settings), or resize the application window so all edges are visible in the smaller screen.

Finally, create a new data file for recording and go to the *Window* menu > *Duplicate* your window. This command creates a duplicate window with all the attributes (list of displayed channels, event display modes, colours, cursors, and size) of the original window. Time view windows in Spike2 or file and memory view windows in Signal may be duplicated. Duplicating a window allows you to have different views of the same data file with different scales and drawing modes and different sets of channels. Use the *View* menu > *Show/Hide channels* dialog (Spike2) or *Customise display* (Signal) to achieve this.

Once you have created a new window, it is independent of the original. However, the data channels within it are the same data channels as in the original window, so any changes made to the data in one window will cause all duplicated windows to update. Cursors in one window are not linked to the parent allowing for multiple active cursor searches. Position the windows on the individual monitors to finish your set up. The individual monitors could also be used to display different information entirely. For example, one monitor could be used to display prompts or images as well as feedback signals to a subject.

When you have finished sampling and saved your data file, you can close all windows associated with a data document in Spike2 by holding down the Ctrl key and going to the *File* menu. When a time view is the current window, the *Close* command becomes *Close and Link*. In Signal select *File > Close All.* These commands will remember the position and state of all windows associated with the document.

Back to contents

Contact Us

If you have any comments about the newsletter format and content, please get in touch: Marjorie@ced.co.uk.

To adjust your subscription preferences, please visit our website: www.ced.co.uk/upgrades/subscribeenews.

Back to contents

Contact us:

In the UK:
Technical Centre, 139 Cambridge Road,
Milton, Cambridge, CB24 6AZ, UK
Telephone: (01223) 420186

Email: info@ced.co.uk
International Tel: [44] 1223 420186
USA and Canada Toll Free: 1 800 345 7794
Website: www.ced.co.uk