```
'Roni Hogri 4.7.22
'Script to automatically analyze calcium signals
'''''''''''''''''''''''''''''''''''''''''''''''''


'Imported data should be non-normalized (raw data from imageJ as txt file) - maximum of 15 channels (cells) per file!
'f0 is defined locally from the last N (normally 30) subthreshold bins (1 s each).
'Filtered channels (DC and/or IIR) are marked by "*".
'If DCTC (DC time constant) or IIR frequency and/are set to zero, DC remove and/or IIR filtration are skipped, respectively.
'After creating the smr file with the dF/F values, the process of looking for Ca2+ events begins:
'A threshold is calculated for each channel, and the times of suprathreshold events are marked in a level channel (1 per waveform channel).
'Threshold = baseline mean + multiplier x SD of baseline. Search for suprathreshold events starts with treatment onset (end of baseline)


''''''''''''''''''''''''''''''''''''''''''''''''''''
'Outputs:
'1. Text file with normalized calcium signal intensities (dF/F values), after filtration (if set)
'2. Text file with values of all parameters measured + analysis info
'''''''''''''''''''''''''''''''''''''''''''''''''''''

'**********************************************************************


main();

proc main();


'''''''''''''''''User-defined variables:

var BaseStart := 0; 'when should the baseline period start (s)?
var BaseEnd% := 30; 'when should the baseline period end (s)?
var NumFBins% := 30; 'number of bins used for calculation of df/f
var HPFreq := 0.001; 'corner frequency to high pass with the IIR filter - higher value is stronger filter
var FilType% := 1; 'type of filter: 0 = butterworth, 1 = bessel
var DCTC := 0; 'time constant for DC remove process (s) - lower value is stronger filter. Set to 0 if you don't want to use it
var Mult := 2; 'multiplier of SD for threshold calculation
var MinDur% := 5; 'minimal duration of event (in whole s), i.e. all points during this period must be suprathreshold for an event to be detected
'(does NOT affect dF/F calculation!)

'suffix of file names for the report files created by program:
var FileEnd1$ := "drift_corrected.txt"; 'for file with all data points for all cells after drift correction (dF/F and filtration)
var FileEnd2$ := "SUMMARY.csv"; 'for file holding extracted results (peaks, areas, etc...)

'what to do when script finishes:
var ShowLast% := 5; 'which file to show at the end: 0 = smr, 1 = log, 2 = script, 3 = txt file; window must not be closed by Closewins%!
var CloseWins% := 1; 'set to 1 if you want to close all files except the script once program finishes



'**********************************************
'END OF USER-DEFINED VARIABLES
'**********************************************




''''''''''''' General variables and processes

var vs% := App(3); 'view handle for the script
var vh%; 'view handle for smr file
var vl% := LogHandle(); 'handle for log file (for printing output)
var t1%; 'view handle for first txt file (value per s after processing)
var OGFNE$, OGFN$; 'full name of original txt file (including path), with and without file extension
var NSMRN$; 'full name of new smr file with running F values
var NumWav%; 'number of valid waveform chans
var mt%; 'number of seconds (lines) in data file (equivalent to maxtime)
var FT$; 'for filter types
docase
case FilType% = 0 then FT$ := "Butterworth";
case FilType% = 1 then FT$ := "Bessel";
endcase


FrontView(vl%); EditSelectAll(); EditClear(); 'clear log

DefTxt(OGFN$, OGFNE$, NSMRN$, mt%, NumWav%, vs%); 'read txt file with values from imageJ, count rows and columns

var OGVals[mt%][NumWav%]; 'for storing the raw values per s from txt file created by imageJ
var BaseMean[NumWav%]; 'mean value of baseline period per chan
var BaseSD[NumWav%]; 'SD value of baseline period per chan
var Thresh[NumWav%]; 'threshold value per chan
var EventChan%[NumWav%]; 'identifier for marker channels (threshold crossing)
var EventNum[NumWav%]; 'number of suprathreshold events per chan
var MaxBins% := (mt% - BaseEnd%) / MinDur%; 'maximal possible number of suprathreshold events per channel
var RiseTime[NumWav%][MaxBins%]; 'times of suprathreshold events - start
var FallTime[NumWav%][MaxBins%]; 'times of suprathreshold events - end
var EventDur[NumWav%][mt% - BaseEnd%]; 'event durations
var Peak[NumWav%][MaxBins%]; 'peak amplitudes of suprathreshold events
var AOT[NumWav%][MaxBins%]; 'area over threshold per event
var dFF0[mt%][NumWav%]; 'for storing the calculated dF/F values throughout the session

ImpText(OGFNE$, OGVals[][], mt%); 'import raw values from txt file

CalF(NumFBins%, mt%, NumWav%, Mult, OGVals[][], dFF0[][]); 'calculate the df/f with a running window

WriteNewFs(mt%, NumWav%, NSMRN$, dFF0[][], vh%); 'write df/f values onto new smr file

if DCTC > 0 or HPFreq > 0 then 'if filter(s) set by user
    ApplyFilters(NumWav%, HPFreq, FilType%, DCTC); 'filter out slow drifts in all wave chans
endif

ThreshCalc(BaseMean[], BaseSD[], Thresh[], NumWav%, Mult, BaseStart, BaseEnd%); 'Calculate baseline mean, baseline SD, and threshold per chan

ThreshMark(mt%, Thresh[], NumWav%, MinDur%, EventChan%[], BaseEnd%); 'Create event chan, mark suprathreshold events in each equivalent wavchan

EventParams(RiseTime[][], FallTime[][], EventDur[][], Peak[][], AOT[][], NumWav%, EventChan%[], mt%, MinDur%, Thresh[], BaseEnd%);
'For each event, find peak amplitude and area over threshold

'print results to 2 separate files:
PrintResults_dFF(mt%, vh%, NumWav%, FileEnd1$, t1%, CloseWins%, OGFN$); '1. txt file with the corrected values per chan per s, no titles
PrintResults_Summary(vh%, HPFreq, Mult, NumWav%, MaxBins%, EventDur[][], Peak[][], AOT[][], Thresh[], MinDur%, FileEnd2$, DCTC, CloseWins%,
OGFN$, FT$); '2. csv file with peak, area, threshold, info


if CloseWins% = 1 then
    FileClose(-1, 0);
endif

docase 'which window to show when program finishes running
case ShowLast% = 0 then FrontView(vh%); 'show smr data file
case ShowLast% = 1 then FrontView(vl%); 'show log
case ShowLast% = 2 then FrontView(vs%); 'show script
case ShowLast% = 3 then FrontView(t1%); 'show text file (all values after filtration)
endcase


end 'end of main procedure
```

```
'Start of sub-procedures
'''''''''''''''''''''''''''''''''''''''''''''''''


''''''''''''''''''''''read txt file with values from imageJ, count rows and columns
proc DefTxt(&OGFN$, &OGFNE$, &NSMRN$, &mt%, &NumWav%, vs%);


var vot%; 'view of txt file with original pixel values
vot% := FileOpen("",8,8,"Please select a .txt file to import raw data from");

if vot% < 0 then 'if no file selected
    FrontView(vs%); 'return to script and abort
    halt
endif

OGFN$ := FileName$(1) + FileName$(2) + FileName$(3) + FileName$(4); 'full path of original file, without file extention

NSMRN$ := OGFN$ + ".smr"; 'name of new smr data file with dF/F values
OGFNE$ := OGFN$ + ".txt"; 'original file, with file extention

var col[100]; 'original array for holding column (channel) values from first row (time bin)

var c%;
ReadSetup("","");
c% := Read(col[]); 'fill array with values of first row, return number of cells/columns/channels

if c% > 15 then 'Avoid exceeding Spike2 channel number limitation
    Message("Too many channels (columns) in txt file!! Maximum of 15 channels allowed!");
    halt
endif

resize col[c%]; 'resize the columns array according to the actual number of channels

var i%; 'loop index

for i%:=0 to c%-1 do
    if col[i%] > 0 then
        NumWav% += 1; 'Spike2 channel number is column number + 1
    endif
next

var line$; 'for reading lines from txt file
ReadSetup("","");
while Read(line$) >= 0 do    'keep going until line is empty
    mt% += 1; 'count number of lines (time bins) in text file
wend
mt% += 1; 'add 1 to line count since the reading of the first line for column counting skipped a line

FileClose(); 'close txt file so that next read starts from top


end




'''''''''''''''''''''import raw values from txt file
proc ImpText(OGFNE$, &OGVals[][], mt%);


FileOpen(OGFNE$, 8); 'open original text file and read it
ReadSetup("\n", "\t");

var ii%; 'loop index

for ii%:=0 to mt%-1 do
    Read(OGVals[ii%][]); 'in each line (time bin), store column (channel) values in subarray
next


end
```

```
''''''''''''''''''''calculate the df/f with a running window
proc CalF(NumFBins%, mt%, NumWav%, Mult, OGVals[][], &dFF0[][]);

var FsArray[NumFBins%][NumWav%]; 'holds current F values in running window
var locmean, locSD, locthresh; 'local values that change as the window advances

var i%, ii%, jj%; 'loop indices

for i%:=0 to NumWav%-1 do 'for each cell
    for ii%:=0 to NumFBins%-1 do 'for each time bin
        FsArray[ii%][i%] := OGVals[ii%][i%]; 'the first window contains all the F values of the first N bins
    next
next


for i%:=0 to NumWav%-1 do 'for each cell

    ArrSum(FsArray[][i%], locmean, locSD); 'calculate the "local" mean and SD of F values in the current window

    for ii%:=0 to NumFBins%-1 do 'calculate the dF/F of the first window
        dFF0[ii%][i%] := (OGVals[ii%][i%] - locmean) / locmean;
    next
    locthresh := locmean + locSD * Mult; 'original threshold based on baseline period

    for ii%:=NumFBins% to mt%-1 do 'for all remaining time bins
        if OGVals[ii%][i%] < locthresh then
        'if the current F value is lower than the threshold, add the new F value to the window and move the window by 1 bin
            for jj%:=0 to NumFBins%-2 do
                FsArray[jj%][i%] := FsArray[jj%+1][i%];
            next
            FsArray[NumFBins%-1][i%] := OGVals[ii%][i%];
        endif
        ArrSum(FsArray[][i%], locmean, locSD); 'calculate the mean and SD of F values in the current window
        dFF0[ii%][i%] := (OGVals[ii%][i%] - locmean) / locmean; 'calculate dF/F value for this time point in this cell
        locthresh := locmean + locSD * Mult; 'updated threshold for inclusion of next point in the running window
    next

next


end



'''''''''''''''''''''''''import data from txt file into smr file
proc WriteNewFs(mt%, NumWav%, NSMRN$, dFF0[][], &vh%);

vh% := FileNew(7, 0, 1000, 1, mt%); 'create and name the new smr file
FileSaveAs(NSMRN$, -1, 0);

var ChanName$; 'for naming the created channels
var mch%; 'memory channel

var i%, ii%; 'loop indices

for i%:=0 to NumWav%-1 do 'for each imported channel
    mch% := MemChan(1, 0, 1, 0); 'create temporary memory channel (waveform), with offset of 0 V
    ChanOffset(mch%, 0);
    for ii%:=0 to mt%-1 do 'for each time bin in this channel
        MemSetItem(mch%, 0, ii%, dFF0[ii%][i%]); 'Add waveform point to mem channel at time ii%
    next

    ChanSave(mch%, i%+1); ChanShow(i%+1); 'save memory channel; show and name permanent channel
    ChanName$ := "Chan " + Str$(i%+1); ChanTitle$(i%+1, ChanName$);
```

```
        ChanDelete(mch%); 'delete memory channel
next


Draw(0, MaxTime()); 'optimise axes
Optimise();


end



'''''''''''''filter out slow drifts in all wave chans
proc ApplyFilters(NumWav%, HPFreq, FilType%, DCTC);

var ChanName$; 'filtration is indicated in channel name

var ch1%; 'memory buffer identifier
var i%; 'loop index

for i%:=1 to NumWav% do 'for all waveform channels
    ChanName$ := ChanTitle$(i%); 'name of original channel

    if HPFreq > 0 then 'apply IIR filter?
        ch1% := MemChan(0, i%); 'new memory channel with same properties as original chan
        IIRCreate(-1, 1, FilType%, 2, HPFreq); 'IIR High pass 0.001 Hz, second order
        IIRApply(-1, ch1%, i%, 0, MaxTime(), HPFreq); 'Apply created filter - IIR butterworth order of 2, time constant 0.001 s
        ChanDelete(i%); 'delete original channel
        ChanSave(ch1%, i%); 'save memory channel instead of original channel
        ChanShow(i%); 'show new channel
        ChanDelete(ch1%); 'delete memory channel
    endif


    if DCTC > 0 then
        ChanProcessAdd(i%, 2, DCTC); 'Use the 'DC remove' process (additional filtration)
    endif

    ChanName$ += "*"; ChanTitle$(i%, ChanName$); 'add * to channel name to indicate filtration
    Optimise(); 'optimize y axis to filtered signal

next


end



''''''''''''''Calculate baseline mean, baseline SD, and threshold per channel
proc ThreshCalc(&BaseMean[], &BaseSD[], &Thresh[], NumWav%, Mult, BaseStart, BaseEnd%);

var i%; 'loop index

for i%:=0 to NumWav%-1 do 'for each cell
    BaseMean[i%] := ChanMeasure(i%+1, 2, BaseStart, BaseEnd%); 'get the mean and SD of baseline calcium activity
    BaseSD[i%] := ChanMeasure(i%+1, 12, BaseStart, BaseEnd%);
    Thresh[i%] := BaseMean[i%] + BaseSD[i%] * Mult; 'calculate threshold for calcium event detection
next


end


''''''''''''''Create event chan, mark suprathreshold events detected in each source wavechan
proc ThreshMark(mt%, Thresh[], NumWav%, MinDur%, &EventChan%[], BaseEnd%);

var mc%, cc%; 'memory channel index, counter of suprathreshold events per channel
var i%,ii%; 'loop indices

for i%:=0 to NumWav%-1 do 'for all cells

```

```
    cc%:=0; 'reset suprathreshold counter
    mc%:=MemChan(4, 0); 'create temporary chan for storing of suprathreshold event periods (level)

    'Scan all data points after baseline, see where thershold is crossed, then see if the minimal duration rule is fulfilled;
    'Mark in level type channel with high level indicating event duration:

    for ii%:=BaseEnd% to mt%-1 do 'go over file, from end of baseline until the end, look for continuous suprathreshold events
        if ChanMeasure(i%+1, 2, ii%-0.25, ii%+0.25) > Thresh[i%] then 'suprathreshold point?
            cc% += 1; 'add to counter

            if cc% = MinDur% then 'duration rule fulfilled?
                MemSetItem(mc%, 0, ii%-MinDur%+1); 'mark level transition from low to high
            endif

        else 'value at this timepoint is below threshold
            if cc% >= MinDur% then 'were previous timepoints within a detected event?
                MemSetItem(mc%, 0, ii%); 'mark level transition from high to low
            endif

            cc%:=0; 'reset suprathreshold counter
        endif
    next

    EventChan%[i%] := ChanSave(mc%, i%+NumWav%+1); 'save to permanent channel and store identifier
    ChanShow(EventChan%[i%]); ChanTitle$(EventChan%[i%], "Events " + Str$(i%+1)); 'name event chan to identify the waveform chan it refers to
    ChanDelete(mc%); 'delete memory chan

next

end



''''''''''''''''''''For each calcium event, find peak amplitude and area over threshold
proc EventParams(&RiseTime[][], &FallTime[][], &EventDur[][], &Peak[][], &AOT[][], NumWav%, EventChan%[], mt%, MinDur%, Thresh[], BaseEnd%);

var et[NumWav%][mt% - BaseEnd%]; 'temporary array for eventtimes

var i%,ii%; 'loop indices
var c%,j%,k%; 'counters

for i%:=0 to NumWav%-1 do 'for all channels
    c% := Count(EventChan%[i%], BaseEnd%, MaxTime()); 'count rising/falling level events per chan

    if c% mod 2 <> 0 then 'mising falling event (over thershold until the end of recording)
        c% += 1;
    endif

    j% := 0; 'reset rise/fall counters
    k% := 0;

    if c% > 0 then 'are there any suprathreshold events in this channel?
        ChanData(EventChan%[i%], et[i%][], BaseEnd%, MaxTime()); 'find rising/falling times

        for ii%:=0 to c%-1 do 'go over all rising/falling, group to rising and falling

            if (ii% mod 2) = 0 then 'rising events

                RiseTime[i%][j%] := et[i%][ii%];
                j%+=1; 'add to rising events counter

            else 'falling events

                FallTime[i%][k%] := NextTime(EventChan%[i%], et[i%][ii%-1]);

                if FallTime[i%][k%] < 0 then 'if event doesn't go down until end of recording (returns negative value)
                    FallTime[i%][k%] := MaxTime(); 'consider end of recording as end of
```

```
event
                endif

                k%+=1; 'add to falling events counter
            endif

        next

        for ii%:=0 to c%/2-1 do 'measure parameters between each rise and fall
            EventDur[i%][ii%] := FallTime[i%][ii%] - RiseTime[i%][ii%]; 'event duration
            Peak[i%][ii%] := ChanMeasure(i%+1, 8, RiseTime[i%][ii%], FallTime[i%][ii%]); 'peak amplitude of event
            AOT[i%][ii%] := ChanMeasure(i%+1, 4, RiseTime[i%][ii%], FallTime[i%][ii%]); 'event area (over threshold)
            AOT[i%][ii%] := AOT[i%][ii%] - (Thresh[i%] * EventDur[i%][ii%]);
        next

    endif

next


end


''''''''''''''''''''print txt file with the drift-corrected values per chan per s, same format as original txt files
proc PrintResults_dFF(mt%, vh%, NumWav%, FileEnd1$, &t1%, CloseWins%, OGFN$);

FrontView(vh%); 'go to data file

var ValperSec[NumWav%][mt%]; 'matrix for storing [filtered] dF/F values per cell per sec

var i%,ii%; 'loop indices

for i%:=0 to NumWav%-1 do 'for each cell
    for ii%:=0 to mt%-1 do 'for each time point
        ValperSec[i%][ii%]:=ChanMeasure(i%+1, 2, ii%, ii%+0.99); 'corrected value
    next
next

var vps$; 'for filling rows w text
for ii%:=0 to mt%-1 do 'fill rows one by one
    for i%:=0 to NumWav%-1 do 'go column by column
        if i% = NumWav%-1 then 'if end of row, go to next row; otherwise move to next column
            vps$ += (Str$(ValperSec[i%][ii%]) + "\n");
        else
            vps$ += (Str$(ValperSec[i%][ii%]) + "\t");
        endif
    next
next

FileNew(1, 1); 'new txt file and view handle
t1%:=FrontView();

Print(vps$); 'print all columns and rows into txt file

var NFN$; 'name and save new file
NFN$:= OGFN$ + "_" + FileEnd1$;
FileSaveAs(NFN$, 1);

if CloseWins% = 1 then
    FileClose(0, 0);
endif


end



''''''''''''''''create csv file with peak, area, threshold, and summary for all cells in this recording
proc PrintResults_Summary(vh%, HPFreq, Mult, NumWav%, MaxBins%, EventDur[][], Peak[][], AOT[][], Thresh[], MinDur%, FileEnd2$, DCTC, CloseWins%,
OGFN$, FT$);
```

```


var ef%; 'indicator for if there are no suprathreshold events ("empty file")
var RealMaxBins%; 'number of most detected events in single channel - for determining number of rows

FrontView(vh%); 'go to data file

var C$, E$, P$, A$, T$, I$; 'for storing strings related to each parameter

C$:="Channel (cell) No.:\n";
E$:="Event durations (s):\n\n";
P$:="Peaks (absolute values):\n\n";
A$:="Areas (over threshold):\n\n";
T$:="Threshold per channel:\n";

if DCTC > 0 then 'if DC remove process was used, indicate this in summary

    I$:="Analysis info:\nIIR drift correction frequency = "+Str$(HPFreq)+" ("+FT$+"), followed by DC remove ("+Str$(DCTC)+
    " s)\nThreshold = mean + " +Str$(Mult)+" x SD of baseline\nMinimal event duration = "+Str$(MinDur%)+" s";

else
    I$:="Analysis info:\nIIR drift correction frequency = "+Str$(HPFreq)+" ("+FT$+")\nThreshold = mean + " +Str$(Mult)+
    " x SD of baseline\nMinimal event duration = "+Str$(MinDur%)+" s";

endif


var i%,ii%,c%; 'loop indices, counter

for i%:=0 to NumWav%-1 do 'go over all channels
    C$ += Str$(i%+1) + ",";
    c% := 0; 'initialize event counter
    for ii%:=0 to MaxBins%-1 do 'find maximum number of events across all channels
        if Peak[i%][ii%] > 0 then
            c% += 1; 'add event to count
        endif
    next
    if c% > RealMaxBins% then 'new highest event count
        RealMaxBins% := c%;
    endif


next

docase
case RealMaxBins% = 0 then ef% := 1 'no calucium events detected
case RealMaxBins% < 3 then RealMaxBins% := 3; 'minimum of 3 lines space
endcase


for ii%:=0 to RealMaxBins%-1 do 'go over all peak values per row
    for i%:=0 to NumWav%-1 do 'go over all channels
        if i%=NumWav%-1 then 'if last column, move to next row
            if EventDur[i%][ii%]>0 then 'event detected? if not, skip to next
                E$+=Str$(EventDur[i%][ii%])+"\n";
                P$+=Str$(Peak[i%][ii%])+"\n";
                A$+=Str$(AOT[i%][ii%])+"\n";
            else
                E$+="\n";
                P$+="\n";
                A$+="\n";
            endif


        else
            if EventDur[i%][ii%]>0 then 'event detected? if not, skip to next column
                E$+=Str$(EventDur[i%][ii%])+",";
                P$+=Str$(Peak[i%][ii%])+",";
                A$+=Str$(AOT[i%][ii%])+",";
            else
                E$+=",";
                P$+=",";
                A$+=",";
            endif
        endif

    next
next

for i%:=0 to NumWav%-1 do 'threshold value per channel
    T$+=Str$(Thresh[i%])+",";
next

FileNew(1, 1); 'new text file to print into (to be saved as CSV)

if ef% = 1 then
    Print("*******No suprathreshold events detected in file!*******\n\n\n", T$, "\n"*3, I$);
else
    Print(C$, "\n"*3, E$, "\n"*3, P$, "\n"*3, A$, "\n"*3, T$, "\n"*3, I$);
endif


var NFN$;
NFN$ := OGFN$ + "_" + FileEnd2$; 'save and close csv file
FileSaveAs(NFN$,1); FileClose(0, 0);


end
```