Contents


```
function findEMG(filename)
```

```
% Authors: Nick Jackson (njackson@uoregon.edu) & Ian Greenhouse
%  (img@uoregon.edu)
%
% This function identifies various EMG/TMS events in prerecorded EMG data.
%
% The user defines appropriate data parameters either at the command line
% or in the first section of the code. Analysis parameters are defined in
% the second section, and should be adjusted according to the data.
%
% Event metrics will be added to the trials table depending on the type of
% data collected. The following shows the type of event detection and the
% associated metrics that are outputed:
%
% If EMG (electromyography) burst detection is on:
%
% ch#_EMGburst_onset = beginning of EMG burst
% ch#_EMGburst_offset = end of EMG burst
% ch#_EMG_RT = reaction time relative to photodiode event
% ch#_EMGburst_area = area under the EMG burst curve
%
% If MEP (motor evoked potential) detection is on:
%
% artloc = TMS artefact location
```

```
% ch#_MEP_time = MEP onset
% ch#_MEP_latency = time from TMS artefact to MEP time
% ch#_MEP_duration = length of MEP
% ch#_RMS_preMEP = root mean square tolerance, determines if pre-MEP noise
% may contaminate resting MEP measurements
% ch#_MEP_amplitude = peak-to-peak amplitude of MEP
% ch#_MEP_area = area under the MEP curve
%
% If CSP (cortical silent period) detection is on:
%
% ch#_CSP_onset = beginning of CSP
% ch#_CSP_offset = end of CSP
%
% output:
% Once the analysis code has run, a UI will open and prompt the user to enter
% a file name. The default file name is the original file name appended with
% "preprocessed". The saved file will contain the following:
%     parameters: struct
%         analysis parameters
%     subject : struct
%         generated from recordEMG
%     trials : trials
%         updated trials table with addition columns that contain calculated
%         metrics itemized above
```

Potentially Could be set to .015

We know TMS happens at 1, but it is not consistent when MEP happens

Inconsistent

define data parameters

```
use_command_line = 1;%toggle bool to suit parameter input preferences
if ~use_command_line
    parameters.EMG = 1; % Detect EMG bursts: 0 = no, 1 = yes
    parameters.EMG_burst_channels = [1 2];
    parameters.MEP = 1; % Detect MEPs: 0 = no, 1 = yes
    parameters.artchan_index = 3;
    parameters.MEP_channels = [1];
    parameters.CSP = 0; % Detect CSP: 0 = no, 1 = yes
    parameters.CSP_channels = [0];
    parameters.MEP_std_or_chngpts = 1; % 0 = std of baseline, 1 = findchangepts
end
```

define analysis parameters

```
%edit these to suit your analysis needs
parameters.sampling_rate = 2000; %5000; % samples per second (Hz)
parameters.emg_burst_threshold = .3; % raw threshold in V to consider for EMG
parameters.emg_onset_std_threshold = 2; % number of std to consider for EMG burst onsets/offs
ets
parameters.tms_artefact_threshold = .0001; % raw threshold magnitude in V to consider for TMS
 artefact
parameters.MEP_window_post_artefact = .1; % time in s after TMS to measure MEP in seconds
parameters.pre_TMS_reference_window = .05;  % time before TMS to serve as reference baseline
for MEP onset
parameters.min_TMS_to_MEP_latency = .015; % number of secs after TMS to begin MEP onset detec
tion
parameters.MEP_onset_std_threshold = .5; % number of std to consider for MEP onsets
```

```
parameters.RMS_preMEP_EMG_tolerance = .05; % root mean square EMG tolerance for including MEP

% parameters for removing TMS artefact & MEP when detecting EMG in same channel as MEP
parameters.time_prior_to_TMS_artefact = .005; % time in s prior to TMS artefact to set to zer
o
parameters.end_of_MEP_relative_to_TMS = .1; % time in s of end of MEP relative to TMS artefac
t
```

Parse input

```
if (nargin < 1)
    % open file with finder/file explore
    [filename, pathname] = uigetfile;
    File = fullfile(pathname, filename);
else
    File = fullfile(pwd, filename);
end
load(File);
```

```
Error using load
Number of columns on line 50 of ASCII file C:\Users\Jessica S. Gilliam\Documents\MATLAB\Sheph
erdCenter\VETA\findEMG.m must be the same as previous lines.

Error in findEMG (line 84)
load(File);
```

set number of channels

```
trials.trial_accept(:,1) = 1;
chs = ["ch1","ch2","ch3","ch4","ch5","ch6","ch7","ch8"];
parameters.num_channels = sum(contains(trials.Properties.VariableNames,chs));
```

Toggles EMG and TMS functionality

```
if use_command_line
    parameters.EMG = input('Do you want to find electromyographic (EMG) bursts? yes(1) or no(
0): ');

    if parameters.EMG & ~isfield(parameters, 'EMG_burst_channels')
        parameters.EMG_burst_channels = input('Enter the channels to be analyzed for EMG burs
ts (e.g. [2] or [1 3 5]): ');
    end

    parameters.MEP = input('Do you want to detect motor evoked potentials (MEPs)? yes(1) or n
o(0): ');
    parameters.CSP = input('Do you want to detect cortical silent period (CSP) epochs? yes(1)
 or no(0): ');

    if parameters.MEP | parameters.CSP
        trials.artloc(:,1) = 0; All rows, column 1 artefact location 0?
        trials.preTMS_period_start(:,1) = 0; All rows, column 1 start the pre-TMS period at 0?
```

```
        if ~isfield(parameters,'artchan_index')
            parameters.artchan_index = input('Enter TMS artefact channel #: ');
        end
    end

    if parameters.MEP & ~isfield(parameters, 'MEP_channels')
        parameters.MEP_channels = input('Enter MEP channels (e.g. [2] or [1 3 5]): ');
    end

    if parameters.CSP & ~isfield(parameters,'CSP_channels')
        parameters.CSP_channels = input('Enter CSP channels (e.g. [2] or [1 3 5]): ');
    end
    parameters.MEP_std_or_chngpts = 1; % 0 = std of baseline, 1 = findchangepts
end
```

Always place zero here, we have no artefact channel

find photodiode event

```
if any(strcmp('photodiode', trials.Properties.VariableNames))
    trials = findDiode(trials,parameters);
else
    trials.stim_onset(:,1) = zeros;
end
```

Find TMS, MEP, and EMG events

```
trials = findEvents(trials,parameters);
```

save file

```
outfile=[File(1:end-4),'_preprocessed'];
uisave({'trials','subject','parameters'},outfile);
```

```
end
```

HELPER FUNCTIONS

Find Diode

```
function trials=findDiode(trials,parameters)
% detects large changes in photodiode signal
    for i=1:height(trials)
```

photodiode event

```
        diff_diode = diff(trials.photodiode{i});
        [max_diode_value,max_diode_index] = max(diff_diode);
        trials.stim_onset(i,1) = max_diode_index/parameters.sampling_rate; % location for GUI
```

```
    end
end
```

Find TMS, MEP, CSP, and EMG

```
function trials = findEvents(trials,parameters) % parameterize these
```

initialize trials columns

```
if parameters.EMG
    for chan = 1:length(parameters.EMG_burst_channels)
        trials.(['ch', num2str(parameters.EMG_burst_channels(chan)), '_EMGburst_onset'])(:,1)
 = 0;
        trials.(['ch', num2str(parameters.EMG_burst_channels(chan)), '_EMGburst_offset'])(:,1
) = 0;
        trials.(['ch', num2str(parameters.EMG_burst_channels(chan)), '_EMGburst_area'])(:,1)
= 0;
    end
end

if parameters.artchan_index
    trials.artloc(:,1) = 0;
end

if parameters.MEP
    for chan = 1:length(parameters.MEP_channels)
        trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_time'])(:,1) = 0;
        trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_latency'])(:,1) = 0;
        trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_offset'])(:,1) = 0;
        trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_area'])(:,1) = 0;
    end
end

if parameters.CSP
    for chan = 1:length(parameters.CSP_channels)
        trials.(['ch', num2str(parameters.CSP_channels(chan)), '_CSP_onset'])(:,1) = 0;
        trials.(['ch', num2str(parameters.CSP_channels(chan)), '_CSP_offset'])(:,1) = 0;
    end
end
```

Artefact channel again saying all rows, column 1 that the artefact is 0

This seems to be fine - output parameters show 6 channels

identify MEP and non-MEP channels

```
if parameters.MEP & parameters.EMG
    non_MEP_channels = parameters.EMG_burst_channels(parameters.EMG_burst_channels ~= paramet
ers.MEP_channels); % do not want to detect MEPs as EMG bursts, so ignore MEP channel
elseif parameters.EMG
    non_MEP_channels = parameters.EMG_burst_channels;
end
```

sweep loop

```
for i = 1:height(trials)
```

find TMS artefact and MEP

```
    if parameters.MEP
        for chan = 1:length(parameters.MEP_channels)
            MEPchannel = trials.(['ch', num2str(parameters.MEP_channels(chan))]){i,1};
            if parameters.artchan_index
                artchannel = trials.(['ch', num2str(parameters.artchan_index)]){i,1};
                [artefact_value, TMS_artefact_sample_index] = max(artchannel); %max(abs(artch
annel));

            else
                TMS_artefact_sample_index = find(MEPchannel > parameters.tms_artefact_thresho
ld,1);
                artefact_value = abs(MEPchannel(TMS_artefact_sample_index));
            end

            %set range to look for preMEP EMG activity to calculate RMS
            lower_rms_bound = TMS_artefact_sample_index - (parameters.pre_TMS_reference_windo
w * parameters.sampling_rate);
            upper_rms_bound = TMS_artefact_sample_index;

            %redefine range if it extends beyond lower x limit
            if lower_rms_bound < 0
                lower_rms_bound = 1;
            end
            preTMS_reference_data = MEPchannel(lower_rms_bound:upper_rms_bound);
            RMS_of_preMEP_window = rms(preTMS_reference_data);
            trials.(['ch',num2str(parameters.MEP_channels(chan)),'_RMS_preMEP'])(i,1) = RMS_o
f_preMEP_window;

            % reject trial if RMS is above tolerance threshold
            if RMS_of_preMEP_window < parameters.RMS_preMEP_EMG_tolerance
                trials.trial_accept(i,1)=1;
            else
                trials.trial_accept(i,1)=0;
            end

            if artefact_value > abs(parameters.tms_artefact_threshold) % TMS artefact must ex
ceed a threshold to be classified as an artefact
                trials.artloc(i,1) = TMS_artefact_sample_index/parameters.sampling_rate; %art
efact location scaled for visualization

                %define MEP search range
                lower_limit_MEP_window = TMS_artefact_sample_index + (parameters.min_TMS_to_M
EP_latency * parameters.sampling_rate);
                upper_limit_MEP_window = TMS_artefact_sample_index + (parameters.MEP_window_p
ost_artefact * parameters.sampling_rate);
                if lower_limit_MEP_window>length(MEPchannel)
                    lower_limit_MEP_window = 1;
                end
                if upper_limit_MEP_window>length(MEPchannel)
                    upper_limit_MEP_window = length(MEPchannel);
                end
```

Initializing what MEPChannel is being pulled (rows, columns)

artchannel = 0??

Verification that MEP Channel is larger than tms threshold
The threshold is 1.0 e -4

```
                MEPsearchrange = abs(MEPchannel(lower_limit_MEP_window:upper_limit_MEP_window
));

                % detect MEP onset and offset point;
                MEP_onset_from_TMS = find(MEPsearchrange > parameters.MEP_onset_std_threshold
 * std(abs(preTMS_reference_data)),1); % first value that exceeds std threshold within rectif
ied MEP search range
                ipoints = findchangepts(MEPsearchrange, 'MaxNumChanges', 10, 'Statistic', 'me
an'); % fewer change points may suffice

                % select option for determining MEP onset
                if parameters.MEP_std_or_chngpts & ipoints
                    MEP_onset_index = ipoints(1) + lower_limit_MEP_window; % use findchangept
s value
                else
                    MEP_onset_index = MEP_onset_from_TMS + lower_limit_MEP_window; % use num
std of baseline
                end

                if ipoints
                    MEP_offset_index = ipoints(end) + lower_limit_MEP_window;
                else
                    MEP_offset_index = MEP_onset_index;
                end

                % define lower limit of pre-TMS artefact reference window
                % for determining threshold for MEP.
                %preTMS_reference_window_lower_limit = TMS_artefact_sample_index - (parameter
s.pre_TMS_reference_window * parameters.sampling_rate);

                %redefine range if preTMS reference range extends beyond lower x limit
               % if preTMS_reference_window_lower_limit < 1
                %    preTMS_reference_window_lower_limit = 1;
               % end
               trials.preTMS_period_start(i,1) = lower_rms_bound/parameters.sampling_rate;

                %redefine range if it extends beyond upper x limit

                if MEP_offset_index > length(trials.ch1{1,1})
                    MEP_offset_index=length(trials.ch1{1,1})-1;
                end

                %look only in range after artefact
                %preTMS_MEP_reference_data = MEPchannel(preTMS_reference_window_lower_limit:T
MS_artefact_sample_index);
                MEPsearchrange = MEPchannel(lower_limit_MEP_window:MEP_offset_index);
                [max_MEP_value,MEP_max_sample_point] = max(MEPsearchrange);
                [min_MEP_value,MEP_min_sample_point] = min(MEPsearchrange);
                MEParea = sum(abs(MEPchannel(MEP_onset_index:MEP_offset_index)))/(MEP_max_sam
ple_point - MEP_min_sample_point);

                % identify MEP onset
                if ~isempty(MEP_onset_index)
                    trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_time'])(i,1)
 = MEP_onset_index/parameters.sampling_rate;
                    trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_latency'])(i
,1) = (MEP_onset_index/parameters.sampling_rate) - trials.artloc(i,1);
```

```
                    trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_offset'])(i,
1) = (MEP_offset_index/parameters.sampling_rate) - trials.artloc(i,1);
                    trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_duration'])(
i,1) = trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_offset'])(i,1) - trials.(
['ch', num2str(parameters.MEP_channels(chan)), '_MEP_latency'])(i,1);
                    trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_amplitude'])
(i,1) = max_MEP_value - min_MEP_value;
                    trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_area'])(i,1)
 = MEParea;
                end

            end
        end
    end
```

find EMG bursts

if EMG burst detection is required in MEP channel

```
    if parameters.EMG & parameters.MEP & sum(ismember(parameters.EMG_burst_channels, paramete
rs.MEP_channels))
        for chan = 1:length(parameters.MEP_channels(chan))

            % signal_burst = MEPchannel;
            signal_burst= trials.(['ch', num2str(parameters.MEP_channels(chan))]){i,1};

            if trials.artloc(i,1)
                % remove MEP to evaluate EMG activity in the absence of TMS and MEP
                % influence on EMG trace
                preTMS_timepoint = (trials.artloc(i,1) - parameters.time_prior_to_TMS_artefac
t) * parameters.sampling_rate; %weird sci notation issue, round necessary
                MEP_end_timepoint = preTMS_timepoint + (parameters.end_of_MEP_relative_to_TMS
 * parameters.sampling_rate); % end of MEP

                %clear MEP activity from channel before looking for bursts
                signal_burst(round(preTMS_timepoint):round(MEP_end_timepoint)) = mean(MEPchan
nel);
            end


            if max(signal_burst) > parameters.emg_burst_threshold
                emg_burst_onset_time = (find(abs(signal_burst) > parameters.emg_onset_std_thr
eshold * std(signal_burst),1)) / parameters.sampling_rate; % find first deviation greater tha
n # std.
                emg_burst_offset_time_from_end = find(abs(signal_burst(end:-1:1)) > parameter
s.emg_onset_std_threshold * std(signal_burst),1);
                emg_burst_offset_time_from_start = (length(signal_burst) - emg_burst_offset_t
ime_from_end)/parameters.sampling_rate; % find last deviation greater than # std.
                trials.(['ch', num2str(parameters.MEP_channels(chan)) '_EMGburst_onset'])(i,1
) = emg_burst_onset_time; % EMG burst onset
                trials.(['ch', num2str(parameters.MEP_channels(chan)) '_EMGburst_offset'])(i,
1) = emg_burst_offset_time_from_start; % EMG burst offset
                trials.(['ch', num2str(parameters.MEP_channels(chan)) '_EMGburst_area'])(i,1)
 = ...
                    sum(abs(signal_burst(round(emg_burst_onset_time * parameters.sampling_rat
```

```
e):round(emg_burst_offset_time_from_start * parameters.sampling_rate))))/ ...
                    (round(emg_burst_offset_time_from_start * parameters.sampling_rate) - rou
nd(emg_burst_onset_time * parameters.sampling_rate)); % EMG burst area

                if trials.stim_onset(i,1)
                    trials.(['ch', num2str(parameters.MEP_channels(chan)) '_EMG_RT'])(i,1) =
trials.(['ch', num2str(parameters.MEP_channels(chan)) '_EMGburst_onset'])(i,1) - trials.stim_
onset(i,1);
                end
            end
        end
    end

    if parameters.EMG & ~isempty(non_MEP_channels)

        for n = 1:length(non_MEP_channels)
            emgcomp(i,n) = max(abs(trials.(['ch', num2str(non_MEP_channels(n))]){i,1}));
        end

        [burstsize,burstchan_index] = max(emgcomp(i,:));
        burstchan = non_MEP_channels(burstchan_index);

        % Detect bursts in non-MEP channels
        for chan = 1:length(non_MEP_channels) %loop through and see which channels surpass th
e threshold
            if emgcomp(i,chan) > parameters.emg_burst_threshold
                signal_burst = trials.(['ch', num2str(non_MEP_channels(chan))]){i,1};
                emg_burst_onset_time = (find(abs(signal_burst) > parameters.emg_onset_std_thr
eshold * std(signal_burst),1)) / parameters.sampling_rate; % find first deviation greater tha
n # std.
                emg_burst_offset_time_from_end = find(abs(signal_burst(end:-1:1)) > parameter
s.emg_onset_std_threshold * std(signal_burst),1);
                emg_burst_offset_time_from_start = (length(signal_burst) - emg_burst_offset_t
ime_from_end)/parameters.sampling_rate; % find last deviation greater than # std.
                trials.(['ch', num2str(non_MEP_channels(chan)) '_EMGburst_onset'])(i,1) = emg
_burst_onset_time;
                trials.(['ch', num2str(non_MEP_channels(chan)) '_EMGburst_offset'])(i,1) = em
g_burst_offset_time_from_start;
                trials.(['ch', num2str(non_MEP_channels(chan)) '_EMGburst_area'])(i,1) = ...
                    sum(abs(signal_burst(round(emg_burst_onset_time * parameters.sampling_rat
e):round(emg_burst_offset_time_from_start * parameters.sampling_rate))))/ ...
                    (round(emg_burst_offset_time_from_start * parameters.sampling_rate) - rou
nd(emg_burst_onset_time * parameters.sampling_rate)); % EMG burst area

                if trials.stim_onset(i,1)
                    trials.(['ch', num2str(non_MEP_channels(chan)) '_EMG_RT'])(i,1) = trials.
(['ch', num2str(non_MEP_channels(chan)) '_EMGburst_onset'])(i,1) - trials.stim_onset(i,1);
                end
            end
        end
    end
```

find CSP

```
    if parameters.CSP
        for chan = 1:length(parameters.CSP_channels)
```

```
            if trials.artloc(i,1)
                csp_signal = trials.(['ch', num2str(parameters.CSP_channels(chan))]){i,1};
                [IUPPER, ILOWER, UPPERSUM] = cusum(abs(csp_signal));
                if sum(ismember(parameters.CSP_channels, parameters.MEP_channels))
                    trials.(['ch', num2str(parameters.CSP_channels(chan)) '_CSP_onset'])(i,1)
 = trials.artloc(i,1) + ...

   trials.(['ch', num2str(parameters.MEP_channels(chan)), '_MEP_offset'])(i,1);
                    csp_start_loc = trials.(['ch', num2str(parameters.CSP_channels(chan)) '_C
SP_onset'])(i,1) * parameters.sampling_rate;
                else
                    [peak1, csp_start_loc] = findpeaks(UPPERSUM, 'MinPeakProminence', 9, 'NPe
aks', 1);
                    trials.(['ch', num2str(parameters.CSP_channels(chan)) '_CSP_onset'])(i,1)
 = csp_start_loc/parameters.sampling_rate;
                end
                if csp_start_loc
                    [peak2, csp_end_position] = findpeaks(-1*UPPERSUM(round(csp_start_loc):en
d), 'MinPeakProminence', 5, 'NPeaks', 1); % default 5, lower may suffice
                    csp_end_loc = csp_end_position + csp_start_loc;
                end
                if csp_end_loc
                    trials.(['ch', num2str(parameters.CSP_channels(chan)) '_CSP_offset'])(i,1
) = csp_end_loc/parameters.sampling_rate;
                end
            else
                artchannel = trials.(['ch', num2str(parameters.artchan_index)]){i,1};
                [artefact_value, TMS_artefact_sample_index] = max(abs(artchannel));

                if artefact_value > abs(parameters.tms_artefact_threshold) % TMS artefact mus
t exceed a threshold to be classified as an artefact
                    trials.artloc(i,1) = TMS_artefact_sample_index/parameters.sampling_rate;%
artefact location scaled for visualization
                    csp_signal = trials.(['ch', num2str(parameters.CSP_channels(chan))]){i,1}
;
                    [IUPPER, ILOWER, UPPERSUM] = cusum(abs(csp_signal));
                    [peak1, csp_start_loc] = findpeaks(UPPERSUM,'MinPeakProminence', 9, 'NPea
ks', 1);
                    if csp_start_loc
                        [peak2, csp_end_position] = findpeaks(-1*UPPERSUM(csp_start_loc:end),
 'MinPeakProminence', 5, 'NPeaks', 1); % default 5, lower may suffice
                        csp_end_loc = csp_end_position + csp_start_loc;
                    end

                    if csp_start_loc & csp_end_loc
                        trials.(['ch', num2str(parameters.CSP_channels(chan)) '_CSP_onset'])(
i,1) = csp_start_loc/parameters.sampling_rate;
                        trials.(['ch', num2str(parameters.CSP_channels(chan)) '_CSP_offset'])
(i,1) = csp_end_loc/parameters.sampling_rate;
                    end
                end
            end
        end
    end
```

```
end % end trial loop
```

```
end % end findEvent function
```

Published with MATLAB® R2019b