HIPPOCAMPAL CIRCADIAN RHYTHMS IN TEMPORAL LOBE EPILEPSY

By

DAVID ARTHUR STANLEY

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2013

To my parents, Jan and Leonard, and my brother, Geoffrey

APPENDIX C
DATA FILTERING CODE

This appendix presents the “smartfilter” code that was used to automatically identify and remove files containing corrupt data. These bad files usually consisted of large chunks of extremely high amplitude or extremely low amplitude data. Bad data usually resulted from technical issues with recording, such as drained pre-amplifier batteries or intermittent electrical connections.

This code acts, generally, to identify exceedingly high amplitude data points and to mark as bad any file that contains too many such data points. We supply as input to the function the mean power in the 1 – 100 Hz frequency range for each non-overlapping 2-second epoch of data. The frequency range 1 – 100 Hz was chosen to capture the majority of the raw signal’s power; we chose 100 Hz as the upper bound, rather than the Nyquist frequency, to avoid having the filter’s parameters be a function of the sampling rate. In addition to the time (t) and mean PSD power ($d0$), the filter also takes in the file number (*fnum*) associated with each data point. *invert* is a flag for operating in inversion mode, described below, and *ratN* is the number associated with the rat under investigation. The function returns the indices of the originally supplied data points that are associated with bad files.

The filter operates by first applying an initial threshold (defined by variable *prefilter_threshold*) to identify data that is *prefilter_threshold* standard deviations above the mean value of the entire dataset. Secondly, it defines an envelope function that is *envelope_threshold* standard deviations above the local mean value of the data. Local means and standard deviations are based on 5 day bins of data (determined by *bin_size*). Files that contain more than *max_allowed_badpoints* data points exceeding this envelope are marked as bad (Figure C-1). The final portion of the code acts to manually remove some bad files that failed to be detected by the filter.

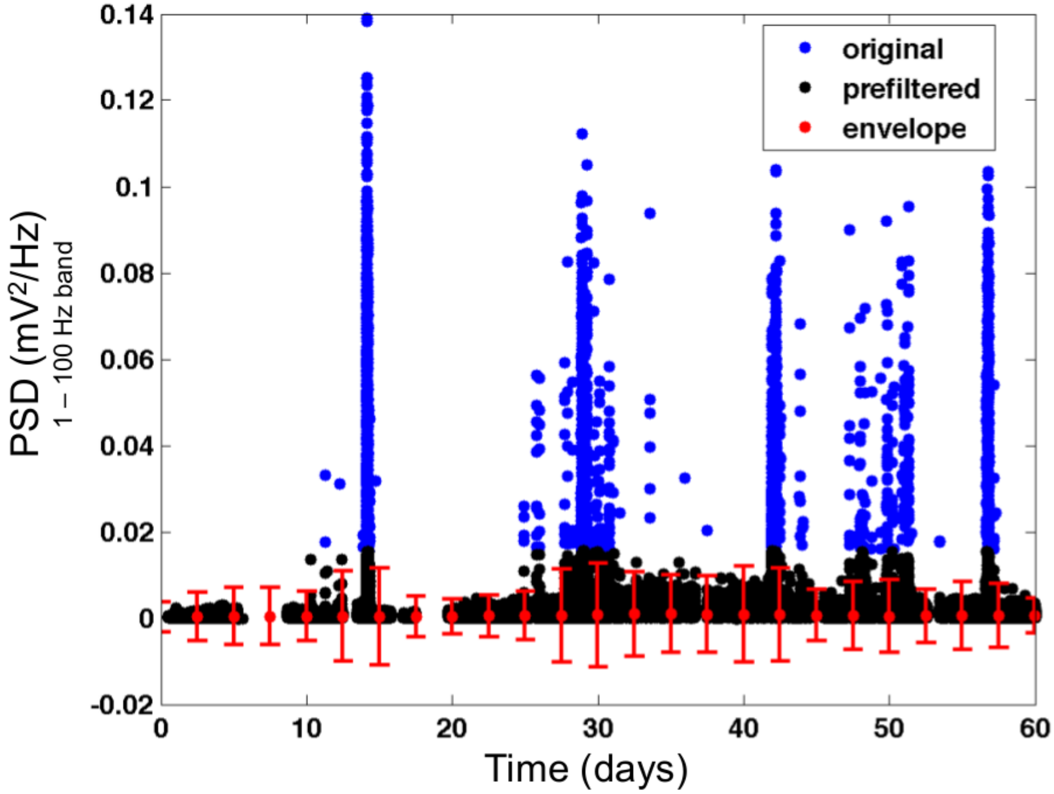

Figure C-1. Rat 4 raw data, prior to any smoothing (blue). Each data point represents the mean power from 1 – 100 Hz in a 2-second epoch of data. Exceedingly high amplitude data was first removed via pre-filtering (black). An envelope function was then defined to identify remaining regions of high amplitude data (red). Files that contained too many data points exceeding this envelope function were eliminated.

This code also contains an optional inversion mode, whereby it automatically inverts all of the supplied data prior to performing filtering. This allows the algorithm to identify and remove the bad files that contain large amounts of extremely low amplitude data.

smartfilter

```
function bad_indices = smartfilter_files (t,d0,fnum,invert,ratN)

%%%%%%%%%
% Description
%%%%%%%%%
```

```
% This code acts, generally, to identify exceedingly high
% amplitude data  points and marks as bad any file that contains
% too many such data points.
%
% David A Stanley
% University of Florida
% dastanley@ufl.edu
%
%%%%%%%%%
% Inputs
%%%%%%%%%
% t - time values of the input time series
%
% d0 - amplitude of the input time series (must be same
%      length as t). Generally, we have used the PSD in the
%      1 - 100 Hz frequency band
%
% fnum - file associated with each data point (must be same
%      length as t)
%
% invert - flag 0 or 1. If set to 1, data in d0 is inverted
%      prior to analysis. This allows the algorithm to remove
%      extremely low amplitude data points
%
% ratN - Rat number.
%
%%%%%%%%%
```

```
% Outputs
% % % % % % % % %
% bad_indices - returns the indices of each data point in
%       the originally supplied time series that is associated
%       with a bad file




if ~invert
    plot_on = 0;
    bin_size = 5;
    prefilter_threshold = 10;
    envelope_threshold = 20;
    max_allowed_badpoints = 10;
else
    plot_on = 0;
    bin_size = 5;
    prefilter_threshold = 0.2;
    envelope_threshold = 15;
    max_allowed_badpoints = 10;
    invertmax = 1/5;
    invertmax = 1/5 * 3276700^2 / (1e3)^2;
    if ratN == 1
        max_allowed_badpoints = 500;
    end
end
```

```

if invert
    d = 1./d0;
    d(d>invertmax) = invertmax;
else
    d = d0;
end

% Pad the data prior to generating the smoothing that will be used
% for calculating the envelope function
ind = find(t > bin_size,1,'first');
ind2 = find(t > t(end) - bin_size,1,'first');
d_temp = [fliplr(d(1:ind)) d fliplr(d(ind2:end))];
t_temp = [-1*fliplr(t(1:ind)) t t(end) + ...
    -1*(fliplr(t(ind2:end))-t(end)) ];
ind = d_temp < (mean(d_temp) + std(d_temp)*prefilter_threshold);

% Smooth the data
[t_sm d_sm d_std] = daveMVAVG_bin (t_temp(ind), d_temp(ind), ...
    bin_size, 0.5, 0.9,0);

if plot_on
    figure; plot(t,d,'b.')
    hold on; plot(t_temp(ind), d_temp(ind),'k.')
    if ~invert hold on; errorbar(t_sm,d_sm, ...
            d_std*envelope_threshold,'r.')
    else hold on; errorbar(t_sm,d_sm, ...
```

```
            d_sm*envelope_threshold,'r.'); end
    legend('original','prefiltered','envelope');
    xlabel('time days');
end
clear ind ind2

temp = interp1(t_sm,d_sm,t);
if ~invert envelope = temp + ...
        interp1(t_sm,d_std,t)*envelope_threshold;
else envelope = temp + temp*envelope_threshold; end
ind = d > envelope;
bad_files_candidates = fnum(ind);

filebins = unique(fnum);
[nfiles] = hist(fnum,filebins);
[nbad] = hist(bad_files_candidates,filebins);


ind = (nbad) >= max_allowed_badpoints;
bad_files = (filebins(ind));
if ratN == 1; bad_files = [bad_files 188]; end
if ratN == 1; bad_files = [bad_files 187 151 152 ...
        153 201 202 218 250 ...
        251 252 253 264 267 115 129 141 149 151:154]; end
if ratN == 10; bad_files = [bad_files 5:11]; end
    % Added to remove discontinuity during start.

```

```
%%% Remove bad files
bad_indices = logical(zeros(1,length(fnum)));
for i = 1:length(bad_files)
    bad_indices = bad_indices | (fnum == bad_files(i));
end


if plot_on
    figure; plot(t,d,'b.');
    hold on; plot(t(~bad_indices),d(~bad_indices),'k.')
end


end


function [t2 x2 x2_std] = daveMVAVG_bin (t, x, time_bin, ...
    fract_overlap, fract_maxgap, use_tcell_search)


use_tcell_search = 0;

eliminate_gaps = 1;

if nargin < 5; fract_maxgap=0.50; end
if nargin < 4; fract_overlap = 0.50; end
```

```
if fract_maxgap > 1.0; fract_maxgap=fract_maxgap/100; end
        % Convert percent to a fraction
if (fract_overlap > 1); fract_overlap = fract_overlap / 100; end
        % Convert percent to a fraction if need be.


if use_tcell_search          % Not used here.
else
    % Sort data
    pairs = [t(:) x(:)];
    pairs = sortrows(pairs);
    t = pairs(:,1); x = pairs(:,2);

    tmax = max(t);
    tmin = min(t);
    tlen = tmax-tmin;
    dt = t(end)-t(end-1);
    N = length(x);
    fract_shift = 1.0-fract_overlap;

    shift = time_bin*fract_shift;
    if (shift <=0); shift = dt; end

    tbin_starts = tmin:shift:(tmax-time_bin);
    tbin_centres = tbin_starts + time_bin/2;
    nbins = length(tbin_centres);

```

```
t2 = zeros(1,nbins);
x2 = t2;
x2_std = t2;
for i = 1:nbins

    index1 = find ( (tbin_starts(i) <= t),1,'first');
    index2 = find ( ((tbin_starts(i)+time_bin) <= t),1, ...
        'first' ) -1;
    index = index1:index2;
    t2(i) = tbin_centres(i);
    if isempty(index)
        x2(i) = NaN;
        x2_std(i) = NaN;
        fprintf ('Gap too large \n');
    else
        xtemp = x(index); ttemp = t(index);
        if (((1 - (length(index)*dt) / time_bin) > ...
                fract_maxgap) && dt > 1e-9)
            x2(i) = NaN;
            x2_std(i) = NaN;
            fprintf ('Gap too large \n');
        else
            x2(i) = mean(xtemp);
            x2_std(i) = std(xtemp);
        end
    end
    clear xtemp ttemp
```

```
    end
end

if eliminate_gaps
    good_index = find(~isnan(x2));
    x2 = x2(good_index);
    x2_std = x2_std(good_index);
    t2 = t2(good_index);
end

end
```