Supplementary Materials

S.1 Despiking method using convolution (*despike.m*)

```
function [data_ds, ns, index] = despike(data, nw, sig, buffer, varargin)

% DESPIKE filters out spikes from a data vector using a Gaussian convolution.
%   INPUTS: data: nx1 vector
%           nw: sample size of “sliding window” used for convolution
%           sig: # of standard deviations considered significant (& removed)
%           buff: number of adjacent samples to remove
%           varargin{1}: 'interp' option interpolates over nans (cubic)
%           varargin{2}: timestamps for data
%           varargin{3}: (optional) passes interpolation method ('cubic', etc.)
%                        w/o 3rd varargin, reverts to default 'linear'
%  OUTPUTS:                     data_ds: despiked data
%           ns: number of (removed) spikes
%           index: logical vector with TRUE = spike
% REQUIRES: setnan.m (function that sets flagged values to NaN for index with buffer)
%
% Jason Kelley NEWAg Lab OSU
% Written 29 FEB 2016
% Last modified 27JUL2016 (Jewell)

% check for pre-existing non-number points (errors) and interpolate during despiking
nn = isnan(data);
xs = 1:length(data);
if nnz(isnan(data))>0
    data = interp1(xs(~nn),data(~nn),xs,'nearest')';
end

w = gausswin(nw,1);             % Matlab function generates Gaussian filter
sw = sum(w);                    % total area under window function
w = w./sw;                      % normalize window

filter = conv(data,w,'same');   % filtered data using _convolution_

ii = true(length(data),1);      % eliminate edge bias: index to original data
    hw = ceil(length(w)/2);     % data to ignore is 50% of filter window size
    ii(1:hw) = false;
    ii(end-hw:end) = false;

mstd = mw_std(data,nw).*sig;    % significance level in terms of standard deviation
fluc = zeros(length(data),1);   % normalize fluctuations by absolute value
% fluc(ii) = (data(ii)-filter(ii))./data(ii); % alternate def for significance
```

```
fluc(ii) = data(ii)-filter(ii); % spikes are fluctuations exceeding signif. threshold
index = abs(fluc)>mstd;         % index the spikes
ns = nnz(index);                % count the spikes

data_ds = setnan(data,index,buffer); % set spikes and adjacent values to NaN
if nnz(nn)>0                    % reset pre-existing NaNs in data vector
    data_ds(nn) = NaN;
    data(nn) = NaN;
end
fprintf('  %i spikes removed ; ',ns)
fprintf('%3.3f%% of data NaN''ed\n',(sum(isnan(data_ds))/length(data))*100)

% optional plotting for visual inspection (not included here for brevity)

% optional interpolation between nan'd points using timestamp for ordinates
   if nargin > 4 && strcmp(varargin{1},'interp')
      ind = isnan(data_ds);
      switch nargin
       case 5
           time = 1:length(data);
       case 6;
           time = varargin{2};
      end % switch
      if nargin == 7
          data_ds(ind) = interp1(time(~ind),data_ds(~ind),time(ind),varargin{3});
      else
          data_ds(ind) = interp1(time(~ind),data_ds(~ind),time(ind));
      end
   end % end interp option

end % end main despike function

% sub function for moving window standard deviation
    function mstd = mw_std(signal,w)
        % adapted from http://matlabtricks.com/post-20/
        % "calculate-standard-deviation-case-of-sliding-window"

        N = length(signal);
        n = conv(ones(N,1),ones(w,1),'same'); % counts no. elements in each window
        s = conv(signal, ones(1, w), 'same'); % s vector
        q = signal .^ 2;
        q = conv(q, ones(1, w), 'same');      % q vector
        mstd = (q - s.^2./n)./(n-1);                 % variance of moving window
        mstd = mstd.^0.5;                     % standard deviation
    end % moving window mw_std sub-function
```

Published with MATLAB® R2016a

S.2 Structure function calculation using convolution (*strfnc.m*)

```
function [S, max_i, fluxdir] = strfnc( trace, freq, maxlag )
%STRFNC Structure function calculation (following Van Atta, 1977)
%   Last modified 09mar16
% INPUTS 'trace'    data to analyze (Nx1 vector array)
%        'freq'     sampling frequency (Hz)
%        'maxlag'   maximum lag time, (seconds)
% OUTPUTS 'S'       structure functions S(r)^n and lag r (in seconds) for rows
%                   only calculates 2nd 3rd 5th order to save memory,
%                   column order corresponds to SFs, also calculates -S^3(r)/r
%                   format: [r S^2(r) S^3(r) -S^3(r)/r S^5(r)]
%        ‘max_i’    relative location (ieration) at which S^3(r)/r is maximum
%        ‘fluxdir’  sign of S^3(r)/r, used to determine vertical flux direction

    m = length(trace);
 lags = 1:maxlag*freq;                      % vector of lags from 1 to maxlag
   rn = length(lags);
    S = zeros(rn,5);                        % initialize array S to store str funcs

% method by nested iterative loops ------------------------------------------
      for j = lags
            r = lags(j); early  = trace(1:end-r);     later  = trace(r+1:end);
            diffs  = later-early;
            for i = [2 3 5]
                  S(j,i) = sum((diffs).^i)/(m-r);
            end %structure functions at lag j
      end % lags j
      S(:,6) = lags./freq;

% method using convolution --------------------------------------------------
filt = [ones(1,rn); -eye(rn)];  % singleton comparators at 1:rn lags  e.g. [1 0 0 -1]
  cT = conv2(trace,filt);       % conv filter with trace to get all lags
  cT = cT(rn+1:end-rn,:);       % trim edges.  'same' does not work as with conv1.m
cTp(:,:,1) = power(cT,ones(m-rn,rn).*2);    % for second order SF
cTp(:,:,2) = cTp(:,:,1).*cT;                % third order SF
cTp(:,:,3) = cTp(:,:,1).*cTp(:,:,2);        % fifth order SF
         w = (m-rn-(1:rn))-1;               % unbiased weighting vector 1/(N-1)
    S(:,2) = sum(cTp(:,:,1),1)./w;          % column order corresponds to SF order
    S(:,3) = sum(cTp(:,:,2),1)./w;          % i.e. 3rd order SF is S(:,3)
    S(:,5) = sum(cTp(:,:,3),1)./w;
    S(:,1) = lags./freq;                    % sample N -> dt
    S(:,4) = -S(:,3)./S(:,1);               % ratio used to detect lag max’ing S3r
% identify time lag at which S_3(r)/r is maximized, flux direction by +/- S^3(r)/r
  fluxdir = sign(nanmean(S(:,4),1));
[~,max_i] = max(fluxdir.*(S(:,4)));

end %function
Published with MATLAB® R2016a
```

S.3 Cardanos Method for roots of depressed cubic polynomial (*cardanos.m*)

```
function [REALrts, ALLrts] = cardanos(p,q)
% CARDANOS(p,q) root finding algorithm for depressed cubic polynomial with real
   valued p and q.  This has reduced functionality of CardanRoots.m for limited
   cases required for the surface renewal method.  Vectors p and q (from structure
   functions) used to determine ramp Amplitudes. polynomial should be of form A^3 +
   p*A + q = 0,  p & q real valued
% RETURNS REALrts: only positive real valued solutions, complex and negative
   solutions replaced with NaN
  ALLrts: includes positive and negatively valued and complex solutions
   % References
%refs:<ahref="matlab:web('https://en.wikipedia.org/wiki/Cubic_function#Cardano.27s_me
   thod','-browser')">Wiki</a>
   %<ahref="matlab:web('https://www.mathworks.com/matlabcentral/newsreader/view_thre
   ad/165013?requestedDomain=www.mathworks.com','-browser')">Source Code</a>

   D = q.^2 + (4/27)*p.^3;                          % the discriminant
    Dneg = D<0;
    Dpos = ~Dneg;
           n = size(D,1);
         rts = zeros(n, 3);                         % initialization
           a = -q(Dneg);       b = sqrt(-D(Dneg));
          r2 = a.^2-D(Dneg);
         rho = (4^(1/3))*exp(log(r2)/6);
       theta = atan2(b,a)/3;
           a = rho.*cos(theta);   b = rho.*sin(theta);
          S1 = a;
           x = (-0.5)*a;          y = (sqrt(3)/2)*b;
          S2 = x-y;              S3 = x+y;

rts(Dneg,1:3) = [S1 S2 S3];
        E = sqrt(D(Dpos));
       u3 = (-q(Dpos)+E)/2;
       v3 = (-q(Dpos)-E)/2;
        u = sign(u3).*exp(log(abs(u3))/3);          % Cubic roots of u3 and v3
        v = sign(v3).*exp(log(abs(v3))/3);
       S1 = u+v;
        j = complex(-0.5,sqrt(3)/2);                % Complex solutions
       j2 = complex(-0.5,-sqrt(3)/2);
       S2 = j*u+j2*v;        S3 = conj(S2);

rts(Dpos,1:3) = [S1 S2 S3];
   ALLrts = rts;
   rts(imag(rts)~=0) = NaN;
   REALrts = rts;
   end
Published with MATLAB® R2016a
```