Striketime Code

Version 1.2   05Jun2025

Now saves a PDF of output automatically

Designed **for new style TAMU data** (event-time files). NOT backward compatable but similar algorithm. Will need to update Preparams.params file to match new input style where there can be up to three data channels A,B,C. The col values indicate which data column the data is stored in, where 0 means no data for that channel.

datatype null null sampletime driftwinA driftwinB basethreshA basethreshB corrt dofit colA colB colC

Sabreen Update d to store results differently than before.

Run this from the directory containing the data files

Point of this code is to find events and their amplitudes. Input parameters for this include a minimum threshold amplitude. Currently there is the thought of handling spikes in a detected non-zero background. To run the same spike finding algoryhtm one needs to perform a "basethresh" filter where all data below basethresh (a.k.a. cut threeshold in bd_175) are set to zero. This facilitates bracketing events with zeros as would be the case for the "clean" single particle events BAS assumes. This type of application HAS NOT BEEN TESTED. At the very least, need to confirm that adding random noise background to a single particle burst signal leads to recovery of correct distribution.

To use code in this way will be iterative as stands. Try plotting the time stream to determine a basethresh value to place in the parameters.params file. Suggest perhaps 3 to 5 times rms of time stream. We can automate/hardcode this after some exploration.

Output:

datastat = 0   if no data processed for that channel

datastat = 1   if processed w/o Gaussian fit to events (dofit=0)

datastat = complex structure if Gaussian fits done (dofit=1)

handles.base = 0   if no data for the channel

handles .base .strike .strikeamp .d   returned in structure for processed channel

Dependancies:

'fittype' requires Curve Fitting Toolbox.

Recent mods:

%Handle possible NaN values - should be very infrequent ; line 42

Initialize

```
clear               % Clear all workspace saved variables

% UPDATE when code is revised
versionnum= 'v1p1';
disp(['FOR TIME STAMP DATA FILES ONLY. Version information and details of modeling
at bottom of mlx code ' ...
    'below the Information function'])
```

```
FOR TIME STAMP DATA FILES ONLY. Version information and details of modeling at bottom of mlx code below the Informat
```

Choose subdirectory to process

```
[PreFileName,PrePathName] = uigetfile('*.par','Select preprocessing parameter file
');
disp(strcat('Using preprocessing parameter file: ',PrePathName,PreFileName));
```

```
Using preprocessing parameter file:C:\Users\jp\Desktop\BAS2025\100nm_Tetraspecs\Pre_12runs.par
```

Read parameter file

```
%The parameter file is assumed in same directory as data
%Parameter file should have 5 header lines at top

paramfile=strcat(PrePathName,PreFileName);
readmePRE = importdata(paramfile); %autofind text header indepent of number of lines
preproparams=readmePRE.data;  %vector of parameters for processing
catfilenames=readmePRE.textdata(6:end-1,1) %last line is column headings
```

```
catfilenames = 12×1 cell
'100nm_TetraSpec_FS_1.txt'
'100nm_TetraSpec_FS_2.txt'
'100nm_TetraSpec_FS_3.txt'
'100nm_TetraSpec_FS_4.txt'
'100nm_TetraSpec_FS_5.txt'
'100nm_TetraSpec_FS_6.txt'
'100nm_TetraSpec_FS_7.txt'
'100nm_TetraSpec_FS_8.txt'
'100nm_TetraSpec_FS_9.txt'
'100nm_TetraSpec_FS_10.txt'
     ⋮
```

```
colheader=readmePRE.textdata(end,:)  %has column header info
```

```
colheader = 1×15 cell
'datatype'    'TotTime'    'MTclock'    'bintime'    'driftwinA'   'driftwinB'   'd···
```

```

%parameters: datatype sampletime driftwinA driftwinB basethreshA basethreshB corrt
dofit colA colB colC
%some of these are obsolete. Datatype is used in case there is an update to
%formatting. Default is 1 for current TAMU format. Note, data files
%typically have txt header with words (say about first 13 lines).
```

```
datatype=readmePRE.data(1)  %Flag for possible changes in data formatting
```

```
datatype =
1
```

```
Tottime=readmePRE.data(2)  %Total recording time
```

```
Tottime =
10
```

```
MTclock=readmePRE.data(3)    %TAMU photon counting board clock
```

```
MTclock =
8.2310e-11
```

```
tbin=readmePRE.data(4)     %Binning size in seconds to form time series
```

```
tbin =
5.0000e-04
```

```
driftwinA=readmePRE.data(5)   %integer bin factor multiplied by 2000
```

```
driftwinA =
0
```

```
driftwinB=readmePRE.data(6)   %slow drift removal
```

```
driftwinB =
0
```

```
driftwinC=readmePRE.data(7)
```

```
driftwinC =
0
```

```
basethreshA=readmePRE.data(8)     %low amplitude spike hash removal (integer cnts)
```

```
basethreshA =
1
```

```
basethreshB=readmePRE.data(9)     %Does change amplitude of remaining data
```

```
basethreshB =
1
```

```
basethreshC=readmePRE.data(10)
```

```
basethreshC =
1
```

```
corrt=readmePRE.data(11)          %Minimal time acceptable bewteen events
```

```
corrt =
0.0030
```

```
dofit=readmePRE.data(12)          %Flag for Gaussian fit to events (usually 0)
```

```
dofit =
0
```

```
colA=readmePRE.data(13)    %Column number where A data located; 0 if no data
```

```
colA =
1
```

```
colB=readmePRE.data(14)    %Column number where B data located; 0 if no data
```

```
colB =
0
```

```
colC=readmePRE.data(15)    %Column number where C data located; 0 if no data
```

```
colC =
0
```

```
numcatfiles=length(catfilenames); %photon counts in time bins
I_A=[0];
I_B=[0];
I_C=[0];

%Set up time series using tbin and total time
disp(append('Estimated number of samples per file at MT clock rate
',num2str(Tottime/MTclock)));
```

```
Estimated number of samples per file at MT clock rate 121491920787.2677
```

```
disp(append('Estimated timestream dimension per file after requested binning time
',num2str(Tottime/tbin)));
```

```
Estimated timestream dimension per file after requested binning time 20000
```

```
%Determine edges of time bins for photon count per bin histogram to follow
Nafterbin = round(Tottime/tbin);
tedge = [0,1:Nafterbin]*tbin;  %edges vector has Nafterbin+1 elements
```

Concatenate data files into 3 color channel vectors

```
disp('Concatenating files using single baseline offset defined in params file')
```

```
Concatenating files using single baseline offset defined in params file
```

```
disp('Confirm mean and rms of all files similar!!')
```

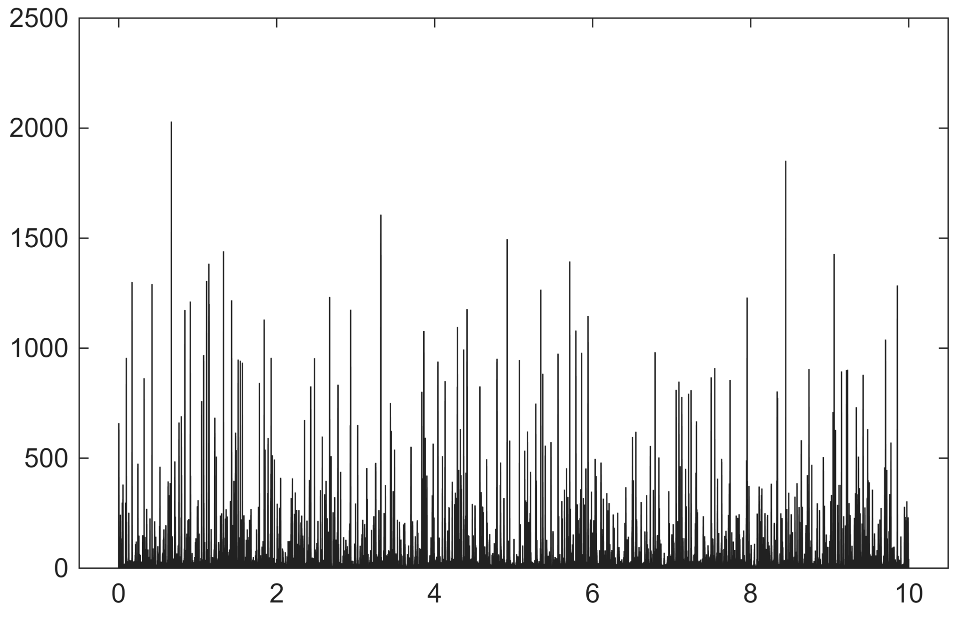
```
Confirm mean and rms of all files similar!!
```

```
for i=1:numcatfiles
```

```
    temp = importdata(char(strcat(PrePathName,catfilenames(i))),'\t'); %Tab
delimited
%Handle possible NaN values - should be very infrequent
    ss=mean(temp.data,2); %use to find rows with NaN's then drop those rows
    allnans =find(~isfinite(ss)); %rows where NaN's occur
    numnans = length(allnans);
    disp(strcat('Number of NaN containing rows removed:',num2str(numnans)))
     temp.data(allnans,:)=[];

% Form up to 3 color channels
    if (colA ~= 0)
        disp('Add a data file for A channel');
        tempA = temp.data(:,colA)'*MTclock*2;  %arrival times in seconds for one
file;
        % factor of 2 is a feature of the BH board
        tempA = tempA(tempA(:) ~= 0);  %Remove 0's that pad the data columns
        % Apply binning set by valiable tbin=readmePRE.data(4)
        clf; % Clear current figure
        h = histogram(tempA,tedge);
        I_A=[I_A,h.Values];

%         disp('Press any key to continue...');
        pause(1); % or pause; to wait for a key press

    end
    if (colB ~= 0)
        disp('Add a data file for B channel');
        tempB = temp.data(:,colB)'*MTclock*2;  %arrival times in seconds for one
file;
        % factor of 2 is a feature of the BH board
        tempB = tempB(tempB(:) ~= 0);  %Remove 0's that pad the data columns
        % Apply binning set by valiable tbin=readmePRE.data(4)
        clf; % Clear current figure
        h = histogram(tempB,tedge);
        I_B=[I_B,h.Values];
%         disp('Press any key to continue...');
        pause(1); % or pause; to wait for a key press

    end
    if (colC ~= 0)
        disp('Add a data file for C channel');
        tempC = temp.data(:,colC)'*MTclock*2;  %arrival times in seconds for one
file;
        % factor of 2 is a feature of the BH board
        tempC = tempC(tempC(:) ~= 0);  %Remove 0's that pad the data columns
        % Apply binning set by valiable tbin=readmePRE.data(4)
        clf; % Clear current figure
        h = histogram(tempC,tedge);
        I_C=[I_C,h.Values];
%         disp('Press any key to continue...');
```

```
        pause(1); % or pause; to wait for a key press

    end

end
```

```
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
Number of NaN containing rows removed:0
Add a data file for A channel
```



```
%Remove first null data point and initialize original timestream output and
%find spike-free rms, mean and median for possible offset correction in BAS code
if (colA ~= 0)
    I_A=I_A(2:end);
    handlesA.base=I_A;
    rms1=std(I_A);
    qrms1=(I_A < 5*rms1);   % Data that isn't part of major spike activity
    handlesA.rms=std(I_A(qrms1));
    handlesA.median=median(I_A(qrms1));
    handlesA.mean=mean(I_A(qrms1));
else
    disp('No A Channel data specified')
    handlesA.base=0;
    datastatA=0;
    peaklocA=0;
end
if (colB ~= 0)
    I_B=I_B(2:end);
    handlesB.base=I_B;
    rms1=std(I_B);
    qrms1=(I_B < 5*rms1);   % Data that isn't part of major spike activity
    handlesB.rms=std(I_B(qrms1));
    handlesB.median=median(I_B(qrms1));
    handlesB.mean=mean(I_B(qrms1));
else
    disp('No B Channel data specified')
    handlesB.base=0;
    datastatB=0;
    peaklocB=0;
end
```

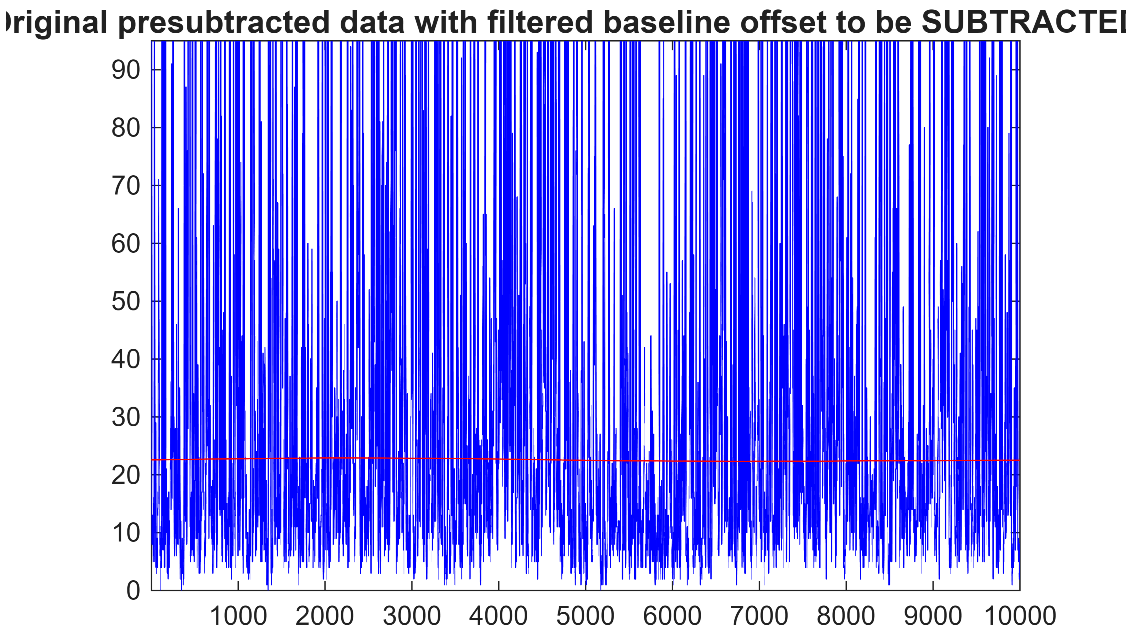
```
No B Channel data specified
```

```
if (colC ~= 0)
    I_C=I_C(2:end);
    handlesC.base=I_C;
    rms1=std(I_C);
    qrms1=(I_C < 5*rms1);   % Data that isn't part of major spike activity
    handlesC.rms=std(I_C(qrms1));
    handlesC.median=median(I_C(qrms1));
    handlesC.mean=mean(I_C(qrms1));
else
    disp('No C Channel data specified')
    handlesC.base=0;
    datastatC=0;
    peaklocC=0;
end
```

```
No C Channel data specified
```

```


%Apply slow drift correction to baseline (paramsfile parameters 5&6)
if driftwinA && colA
    [I_A,basedrift]=Basedrift_Data(I_A,driftwinA,'A');
    handlesA.basedriftA=basedrift;
end
if driftwinB && colB
```

```
    [I_B,basedrift]=Basedrift_Data(I_B,driftwinB,'B');
    handlesB.basedriftB=basedrift;
end
if driftwinC && colC
    [I_C,basedrift]=Basedrift_Data(I_C,driftwinC,'C');
    handlesC.basedriftC=basedrift;
end


%Apply baseline threshold filter (paramsfile parameters 7&8) to subtract
%offset
if basethreshA && colA
    [I_A,baseoffset]=Basethresh_Data(I_A,basethreshA,'A');
    handlesA.baseoffsetA=baseoffset;
end
```



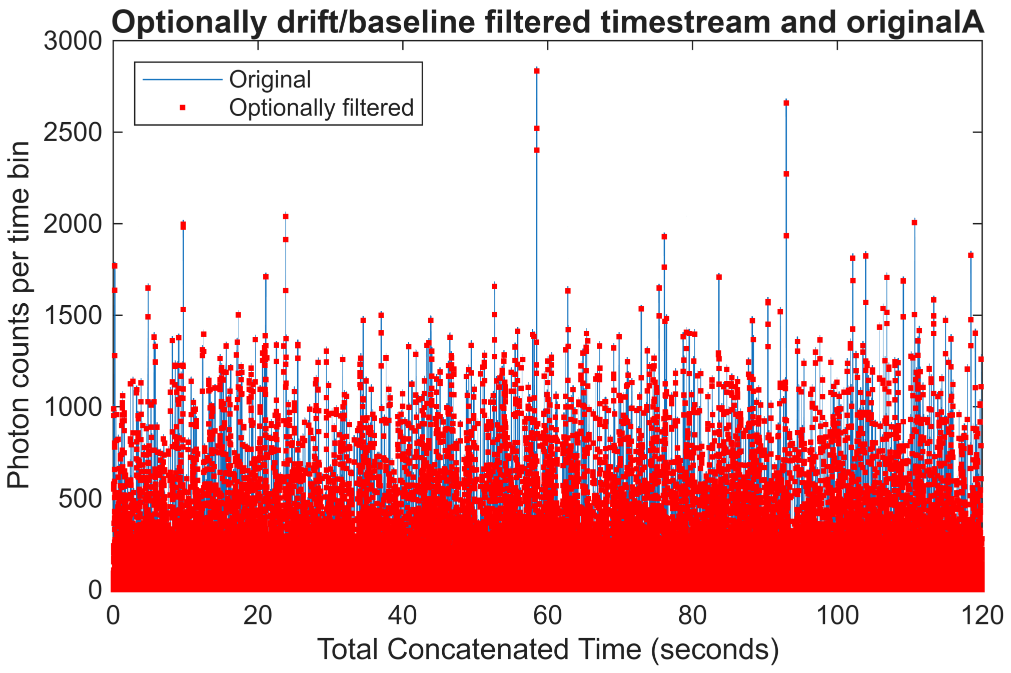
```
if basethreshB && colB
    [I_B,baseoffset]=Basethresh_Data(I_B,basethreshB,'B');
    handlesB.baseoffsetB=baseoffset;
end
if basethreshC && colC
    [I_C,baseoffset]=Basethresh_Data(I_C,basethreshB,'C');
    handlesC.baseoffsetC=baseoffset;
end
```

Determine location and amplitudes of spikes in concatenated data set

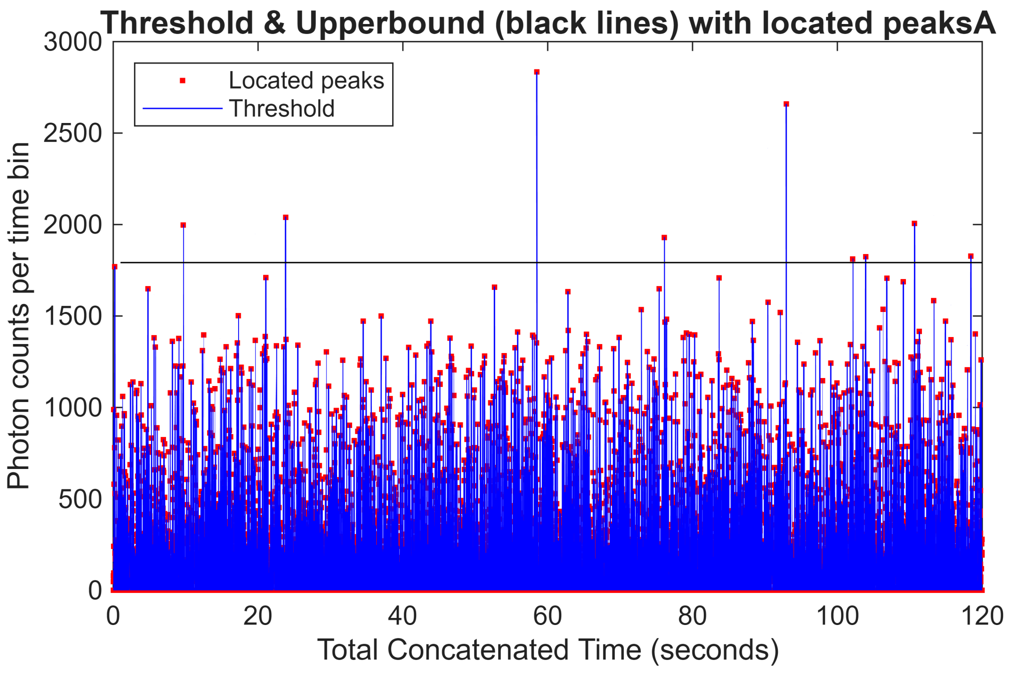
```
if (colA ~= 0)
     [datastatA,handlesA,peaklocA]=burst_data(I_A,tbin,corrt, dofit, 'A', handlesA);
```

```
end
```

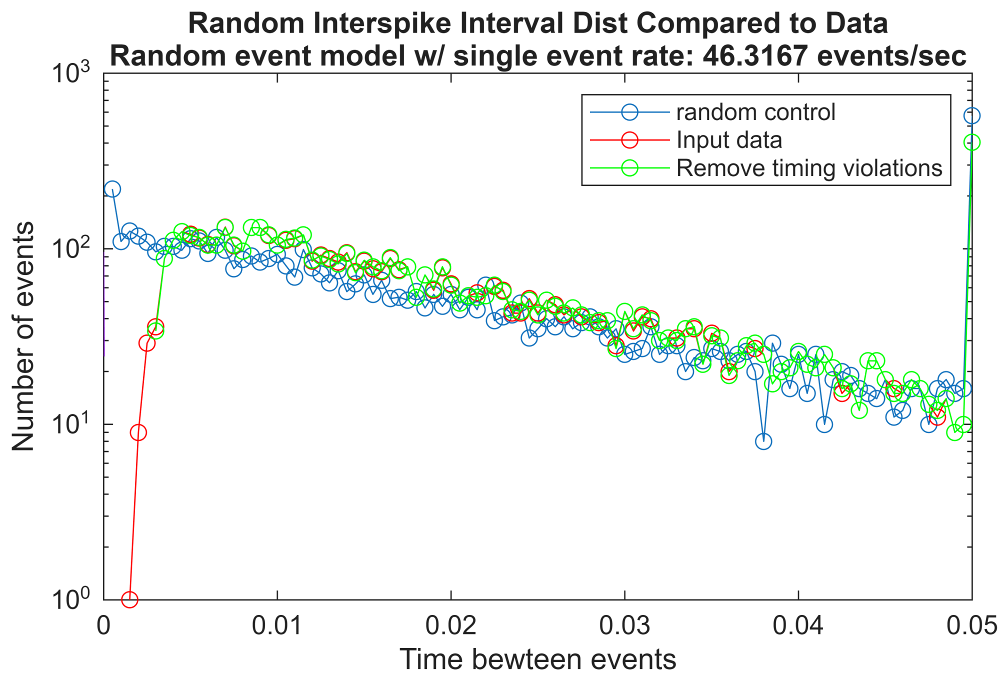
```
***** Processing channel :A
```



```
handles = struct with fields:
            base: [64 112 102 89 60 33 32 26 8 9 8 8 8 13 10 11 5 12 7 15 11 7 12 16 12 7 6 10 7 6 5 6 12 45 89 103
             rms: 76.9307
          median: 19
            mean: 47.2537
     baseoffsetA: [22.5800 22.5801 22.5801 22.5802 22.5803 22.5803 22.5804 22.5804 22.5805 22.5805 22.5806 22.5806
               d: [2×240000 double]
    spthreshtype: "mean"
spikethresh =
94.5073
Chosen spike finding threshold setting is the 2*mean of 5-sigma filtered data set
Setting lower spike acceptance threshold to:94.5073
Retaining subthreshold spikes between47.2537 and94.5073 in Handles structure
```



```
Checking interspike timing distribution and removing events closer than corrt
----------------------------------------
Removing39 intertime violation events from5519 total events
Warning: Start point not provided, choosing random start point.
```



```
if (colB ~= 0)
    [datastatB,handlesB,peaklocB]=burst_data(I_B,tbin,corrt, dofit, 'B', handlesB);
end
if (colC ~= 0)
    [datastatC,handlesC,peaklocC]=burst_data(I_C,tbin,corrt, dofit, 'C', handlesC);
end
```

Check rate of strikes and single particle limit

Use a few stats to determine if single particle limit holds for this dataset

```
do_strikestats=true
```

```
do_strikestats = logical
   1
```

```

if do_strikestats
% Determine some statistics about detected spikes

if (colA ~= 0)
    numbins = length(handlesA.strike)
    disp('>>>>>>>Channel A')
    numspA = sum(handlesA.strike);
    disp(strcat('Number events above A threshold: ',num2str(numspA)));
% If random, prob of spikes in single bin and multiple adjacent bins
    p0bin1A= (numbins-numspA) / numbins;
    p1bin1A= numspA / numbins;
    disp(strcat('Prob of 0 event in a bin: ',num2str(p0bin1A)));
    disp(strcat('Prob of 1 event in a bin: ',num2str(p1bin1A)));
    disp(strcat('Prob of 2 events in a bin based on P(n=1)^2:
',num2str(p1bin1A^2)));

    disp(' ')
    disp(strcat('Fraction of events that are double strikes
',num2str((p1bin1A^2*numbins)/numspA) ));

    %average value and prob of a subthreshold event in subthresh band
    avgsubampA=mean(handlesA.subthreshstrikeamp);
    probsubampA= sum(handlesA.subthreshstrike)/numbins;
    disp('avg A channel contribution of subthresh events is')
    overampsubA=probsubampA*avgsubampA



% Prob of subthreshold events contributing to more than 10%

end


if (colB ~= 0)
    numbins = length(handlesB.strike)
    disp('>>>>>>>Channel B')
    numspB = sum(handlesB.strike);
    disp(strcat('Number events above B threshold: ',num2str(numspB)));
% If random, prob of spikes in single bin and multiple adjacent bins
    p0bin1B= (numbins-numspB) / numbins;
    p1bin1B= numspB / numbins;
    disp(strcat('Prob of 0 event in a bin: ',num2str(p0bin1B)));
    disp(strcat('Prob of 1 event in a bin: ',num2str(p1bin1B)));
    disp(strcat('Prob of 2 events in a bin based on P(n=1)^2:
',num2str(p1bin1B^2)));

    disp(' ')
    disp(strcat('Fraction of events that are double strikes
',num2str((p1bin1B^2*numbins)/numspB) ));

    %average value and prob of a subthreshold event in subthresh band
    avgsubampB=mean(handlesB.subthreshstrikeamp);
    probsubampB= sum(handlesB.subthreshstrike)/numbins;
    disp('avg B channel contribution of subthresh events is')
    overampsubB=probsubampB*avgsubampB


end

if (colC ~= 0)
    numbins = length(handlesC.strike)
    disp('>>>>>>>Channel C')
    numspC = sum(handlesC.strike);
    disp(strcat('Number events above C threshold: ',num2str(numspC)));
% If random, prob of spikes in single bin and multiple adjacent bins
    p0bin1C= (numbins-numspC) / numbins;
    p1bin1C= numspC / numbins;
    disp(strcat('Prob of 0 event in a bin: ',num2str(p0bin1C)));
    disp(strcat('Prob of 1 event in a bin: ',num2str(p1bin1C)));
    disp(strcat('Prob of 2 events in a bin based on P(n=1)^2:
',num2str(p1bin1C^2)));

    disp(' ')
    disp(strcat('Fraction of events that are double strikes
',num2str((p1bin1C^2*numbins)/numspB) ));

    %average value and prob of a subthreshold event in subthresh band
    avgsubampC=mean(handlesC.subthreshstrikeamp);
    probsubampC= sum(handlesC.subthreshstrike)/numbins;
    disp('avg C channel contribution of subthresh events is')
    overampsubC=probsubampC*avgsubampC


end

end
```

```
numbins =
240000
>>>>>>>Channel A
Number events above A threshold:5558
Prob of 0 event in a bin:0.97684
Prob of 1 event in a bin:0.023158
Prob of 2 events in a bin based on P(n=1)^2:0.00053631

Fraction of events that are double strikes0.023158
avg A channel contribution of subthresh events is
overampsubA =
0.0050
```

Cross-correlation of data sets

Determine fraction of events common to both channels using raw signal

```
do_xcorr=true
```

```
do_xcorr = logical
   1
```

```
if (colA*colB*do_xcorr ~= 0)
    disp('Both channels active: finding fractional overlap of events ...')
    %Find typical upper bound using 10 largest strikes
    hold off;
    [c,lags]=xcorr(handlesA.base,handlesB.base,5,'normalized');
    stem(lags,c);
    title('Normailized Cross-correlation of RAW SIGNAL vs bin lag','FontSize', 12)
    figure
    hold off;
    [c,lags]=xcorr(handlesA.strike,handlesB.strike,5,'normalized');
    stem(lags,c);
    title('Normailized Cross-correlation of EVENT TIMES vs bin lag','FontSize', 12)


    sortA=sort(handlesA.strikeamp,'descend');  %sort descending
    sortB=sort(handlesB.strikeamp,'descend');
    UBA= median(sortA(1:10));
    UBB= median(sortB(1:10));
    AQ1=handlesA.strikeamp > 0.5*UBA;
    BQ1=handlesB.strikeamp > 0.5*UBB;


    hold on
    Atempstrike=handlesA.strike.*AQ1;
    Btempstrike=handlesB.strike.*AQ1;
    [c,lags]=xcorr(Atempstrike,Btempstrike,5,'normalized');
    stem(lags,c);
    Atempstrike=handlesA.strike.*BQ1;
    Btempstrike=handlesB.strike.*BQ1;
    [c,lags]=xcorr(Atempstrike,Btempstrike,5,'normalized');
    stem(lags,c);

    legend('All events','Top 50%; A-triggered','Top 50%; B-triggered')

    numberA=sum(handlesA.strike(AQ1));
    numberB=sum(handlesB.strike(BQ1));

    disp('Event zero lag A-triggered, B-triggered, and auto-correlation AA, BB = 1
check')
    fracoverlapAB= sum(handlesA.strike.*handlesB.strike.*AQ1)/numberA
    fracoverlapBA= sum(handlesB.strike.*handlesA.strike.*BQ1)/numberB

    fracoverlapAA= sum(handlesA.strike.*handlesA.strike.*AQ1)/numberA
    fracoverlapBB= sum(handlesB.strike.*handlesB.strike.*BQ1)/numberB
    handlesA.overlapAB=fracoverlapAB;
    handlesB.overlapAB=fracoverlapAB;
end

if (colC*colB*do_xcorr ~= 0)
    disp('Both channels active: finding fractional overlap of events ...')
    %Find typical upper bound using 10 largest strikes
    hold off;
    [c,lags]=xcorr(handlesC.base,handlesB.base,5,'normalized');
    stem(lags,c);
    title('Normailized Cross-correlation of RAW SIGNAL vs bin lag','FontSize', 12)
    figure
    hold off;
    [c,lags]=xcorr(handlesC.strike,handlesB.strike,5,'normalized');
    stem(lags,c);
    title('Normailized Cross-correlation of EVENT TIMES vs bin lag','FontSize', 12)


    sortC=sort(handlesC.strikeamp,'descend');  %sort descending
    sortB=sort(handlesB.strikeamp,'descend');
    UBC= median(sortC(1:10));
    UBB= median(sortB(1:10));
    CQ1=handlesC.strikeamp > 0.5*UBC;
    BQ1=handlesB.strikeamp > 0.5*UBB;


    hold on
    Ctempstrike=handlesC.strike.*CQ1;
    Btempstrike=handlesB.strike.*AQ1;
    [c,lags]=xcorr(Ctempstrike,Btempstrike,5,'normalized');
    stem(lags,c);
    Ctempstrike=handlesC.strike.*BQ1;
    Btempstrike=handlesB.strike.*BQ1;
    [c,lags]=xcorr(Ctempstrike,Btempstrike,5,'normalized');
    stem(lags,c);

    legend('All events','Top 50%; C-triggered','Top 50%; B-triggered')

    numberC=sum(handlesC.strike(CQ1));
    numberB=sum(handlesB.strike(BQ1));

    disp('Event zero lag C-triggered, B-triggered, and auto-correlation CC, BB = 1
check')
    fracoverlapCB= sum(handlesC.strike.*handlesB.strike.*CQ1)/numberC
    fracoverlapBC= sum(handlesB.strike.*handlesC.strike.*BQ1)/numberB

    fracoverlapCC= sum(handlesC.strike.*handlesC.strike.*CQ1)/numberC
    fracoverlapBB= sum(handlesB.strike.*handlesB.strike.*BQ1)/numberB
    handlesC.overlapCB=fracoverlapCB;
    handlesB.overlapCB=fracoverlapCB;
end

if (colA*colC*do_xcorr ~= 0)
    disp('Both channels active: finding fractional overlap of events ...')
    %Find typical upper bound using 10 largest strikes
    hold off;
    [c,lags]=xcorr(handlesA.base,handlesC.base,5,'normalized');
    stem(lags,c);
    title('Normailized Cross-correlation of RAW SIGNAL vs bin lag','FontSize', 12)
    figure
    hold off;
    [c,lags]=xcorr(handlesA.strike,handlesC.strike,5,'normalized');
    stem(lags,c);
    title('Normailized Cross-correlation of EVENT TIMES vs bin lag','FontSize', 12)


    sortA=sort(handlesA.strikeamp,'descend');  %sort descending
    sortC=sort(handlesC.strikeamp,'descend');
    UBA= median(sortA(1:10));
    UBC= median(sortC(1:10));
    AQ1=handlesA.strikeamp > 0.5*UBA;
    CQ1=handlesC.strikeamp > 0.5*UBC;


    hold on
    Atempstrike=handlesA.strike.*AQ1;
    Ctempstrike=handlesC.strike.*AQ1;
    [c,lags]=xcorr(Atempstrike,Ctempstrike,5,'normalized');
    stem(lags,c);
    Atempstrike=handlesA.strike.*CQ1;
    Ctempstrike=handlesC.strike.*CQ1;
    [c,lags]=xcorr(Atempstrike,Ctempstrike,5,'normalized');
    stem(lags,c);

    legend('All events','Top 50%; A-triggered','Top 50%; C-triggered')

    numberA=sum(handlesA.strike(AQ1));
    numberC=sum(handlesC.strike(CQ1));

    disp('Event zero lag A-triggered, C-triggered, and auto-correlation AA, CC = 1
check')
    fracoverlapAC= sum(handlesA.strike.*handlesC.strike.*AQ1)/numberA
    fracoverlapCA= sum(handlesC.strike.*handlesA.strike.*CQ1)/numberC

    fracoverlapAA= sum(handlesA.strike.*handlesA.strike.*AQ1)/numberA
    fracoverlapCC= sum(handlesC.strike.*handlesC.strike.*CQ1)/numberC
    handlesA.overlapAC=fracoverlapAC;
    handlesC.overlapAC=fracoverlapAC;
end
```

Save results to .mat file and a pdf report

```
%Save processing and create PDF report
savebasresults=true
```

```
savebasresults = logical
   1
```

```
%Save processing and create PDF report
if savebasresults

savepredone=Save_Preprocessed(PrePathName,PreFileName,datastatA,datastatB,datastatC,
handlesA,handlesB,handlesC,peaklocA,peaklocB,peaklocC,readmePRE,versionnum)
end
```

```
tstr = datetime
   04Jun25-124837
version number:v1p1
newdir =
"Pre_v1p1_04Jun25-124837"
savepredone =
1
```

Part II

FUNCTIONS

Rolling window filter to remove baseline wander but retain overall mean of timestream

```
function [cI,basedrift]=Basedrift_Data(I,driftwin,channel)

% Removes a smoothed baseline from timestream but retains overall mean of
% timestream; Use a 50 sample median filter (slow and memory intensive but largely
insensitive to spikes)
% followed by a 12000 sample averaging filter on that result (fast and
% roughly 6 second time constant with nominal 0.5 ms sampling

medianI=median(I);
qq=I-medianI;  %Removes median to roughly center baseline on zero
MF=medfilt2(qq,[1 fix(driftwin*50)]); % sets window size and filters
basedrift = MF+medianI;  %restores original amplitude
basedrift = smoothdata(basedrift,'gaussian',12000);

cI=I-basedrift;  %Remove drift but now there are negative values
cI=cI + medianI;  %non-drifting time stream possibly with some zeros
cI(cI<0)=0;       %zero any negative values


%Overplot data and baseline in zoomed in fashion
plot(I,'b');
hold on
plot(basedrift,'r');
xlim([1,10000]);
ylim([0,medianI*5]);
title(strcat('Original data with drift overplotted :',channel),'FontSize', 12);
```

```
hold off
%pause(3)


end
```

DEPRECATED: Low-amp noise suppression (a.k.a. "cut threshold" or "basethresh")

In bd_175.m this is called "Cut threshold". This can be used to supress (but not remove) a background signal before finding spikes and spike amps.

```
% function [cI]=Basethresh_Data(I,basethresh,channel)
% % Used same median filter as used in Basedrift to fine a baseline which is then
removed. Zeroes data
% % below this base threshold curve in each channel but does
%
% cI=I-basethresh;
% zpoints=find(cI < 0);
% if ~isempty(zpoints)
%     cI(zpoints)=-basethresh;
%     cI=cI+basethresh;
% end
%
% %Plot data w/ hash removed
% plot(cI,'.b');
% title(strcat('Data with base threshold applied to suppress low ampl.
background',channel))
% pause(3)
%
%
% end
```

Offset removal

In bd_175.m this is also called "Cut threshold". This routine can be used to subtract a background before finding spikes and spike amps. This subtracts a smoothed drifting function from the raw data and sets any values below zero to zero. The idea is to remove an unresolved, ubiquitous, luminous background that is adding to the event amplitudes.

```
function [cI,baseoffset]=Basethresh_Data(I,basethresh,channel)
% Used same median filter as used in Basedrift to find a baseline but is then
removed. Zeroes out any data
% below zero value in each channel.
% Use a 50 sample median filter (slow and memory intensive but largely insensitive
to spikes)
% followed by a 12000 sample averaging filter on that result (fast and
% roughly 6 second time constant with nominal 0.5 ms sampling
```

```
medianI=median(I);
qq=I-medianI;  %Removes median to roughly center baseline on zero
MF=medfilt2(qq,[1 fix(basethresh*50)]); % sets window size and filters
baseoffset = MF+medianI;  %restores original amplitude
baseoffset = smoothdata(baseoffset,'gaussian',12000);

cI=I-baseoffset;  %Remove drift but now there are negative values

cI(cI<0)=0;       %zero any negative values


%Overplot data and baseline in zoomed in fashion
plot(I,'b');
hold on
plot(baseoffset,'r');
xlim([1,10000]);
ylim([0,max([medianI*5,10]) ]);  %Handles cases with zero median value
title(strcat('Original presubtracted data with filtered baseline offset to be
SUBTRACTED :',channel),'FontSize', 12);
hold off
%pause(3)

end
```

Save preprocessed results to .mat and PDF report

```
function
savepredone=Save_Preprocessed(PrePathName,PreFileName,datastatA,datastatB,datastatC,
handlesA,handlesB,handlesC,peaklocA,peaklocB,peaklocC,readmePRE,versionnum)
% save data and results

%Get information about this data run and create a director for this data store
[filebase, tstr]=Information(versionnum);
timestr=string(tstr);
newdir=strcat(filebase,'_',timestr)
mkdir(newdir)

%prefilenamereport=strcat(PrePathName,filebase,PreFileName,timestr,'Report.pdf');
prefilenamesave=strcat(PrePathName,newdir,'/',filebase,PreFileName,timestr,'.mat');

%Save all Matlab variables and states
save(prefilenamesave)
savepredone=1;

% %Save a PDF of the output
```

```
%export('../New_Pre_v1p2.mlx', 'pdf');

end
```

Find burst location times and amplitudes (preprocess)

```
function [datastat,handles,peakloc]=burst_data(d,tbin, corrt, dofit, channel,
handles)
%This is original 2013 preprocessing code used in MegaMan_V1.m and bd_175
%Some of the commented code is left to show comparison to what was - if
%commented it was not doing anything even in old code
%Channel data are mapped to appropriate old variables like handles
%Input variable channel is string to label channel being processed
%handles.medwin=5;    Not used in bd_175 either

handles.d=zeros(2,length(d));
handles.d(1,:)=[0:length(d)-1]*tbin;  %make time vector in seconds
handles.d(2,:)=d;   %Time data filtered by drift and basethresh is read in
%handles.base=d;     %Original time data unfiltered is read in


disp(strcat('***** Processing channel : ',channel))

%Show original data
figure
plot(handles.d(1,:),handles.base);  %show original time steam data
hold on;
plot(handles.d(1,:),handles.d(2,:),'.r');
title(strcat('Optionally drift/baseline filtered timestream and original
',channel),'FontSize', 12)
legend('Original','Optionally filtered','Location','northwest')
xlabel('Total Concatenated Time (seconds)')
ylabel('Photon counts per time bin')
hold off


%%%%%%%%%
% Consider using a threshold level to define when burst boundary happens
% rather than always using zero
%
% Set an amplitude threshold for base of spike to 0, mean, median, rms, or
% dynamic range (1/100) of large spike
%

%%%% Determine upper range of spike activity ; 30 is a heuristic parameter
    sortbase=sort(handles.base);
    UB= median(sortbase(end-30:end));
```

```
%%%% Determine lower bound (threshold) of trustable spike activity
%% Default to mean for most cases
handles.spthreshtype = "mean"
dynrange = 100;
switch handles.spthreshtype
    case 'zero'
        spikethresh = 0
        disp('Chosen spike finding threshold setting is zero' )
    case 'mean'
        spikethresh = 2*handles.mean
        disp('Chosen spike finding threshold setting is the 2*mean of 5-sigma
filtered data set ' )
    case 'median'
        spikethresh = 2*handles.median
        disp('Chosen spike finding threshold setting is the 2*median of 5-sigma
filtered data set ' )
    case 'rms'
        spikethresh = 2*handles.rms
        disp('Chosen spike finding threshold setting is the 2*rms of 5-sigma
filtered data set ' )
    case 'dynamicrange'
  % Estimate upper bound on events in a channel
        spikethresh = round(UB/dynrange);
        disp(strcat('Applying a 1/100 dynamic range threshold : ',
num2str(spikethresh)))
        disp('Chosen spike finding threshold setting is UB/100 of data set '
)
end

%%% Bound lower threshold has some constraints

if spikethresh < round(UB/dynrange)  % the dynamic range is minimum possible
    spikethresh = round(UB/dynrange)
    disp('**** OVERRIDE: Restricted to Upper Bound / 100 as lower limit ****' )
end
if spikethresh < 10   % set 10 as a lower limit in all cases due to shot noise
    spikethresh = 10
    disp('**** OVERRIDE: Restricted to 10 cnts as lower limit - arbitrary Poisson
noise limit ****' )

end

%%%% Store parameters in structure for later use
    handles.spthresh = spikethresh;
    handles.UB = UB;

%%%%%%%%%

strike=d;  %data imported to function (not baseline)
```

```
% Set a lower bound for spike finding below the threshold as a way of estimating
subthreshold
% activity later
disp(strcat('Setting lower spike acceptance threshold to: ',num2str(spikethresh)));
disp(strcat('Retaining subthreshold spikes between ',num2str(spikethresh/2),' and
',num2str(spikethresh),' in Handles structure'));
strike(strike < spikethresh/2)=0;   % Set lower amplitude to find the edge of spikes

%Guarentee zeroed end points
strike(1)=0; strike(2)=0; strike(end)=0; strike(end-1)=0;
again1=1; again2=1;
while(again2 == 1)
    t=find(strike==max(strike));
    t=max(t);  %This could be 2nd pnt, 2nd to last pnt or in middle of data
    i=1; rightend=1; leftend=1; again1=1;
%    if (t==19997)
 %       t
 %   end;
    strike(t) = -1; %tip of peak in single event set to -1
    while (again1 >= 1)
        ti_index=max(max([mod(t+i,size(strike,2)),1])); %Because nonzero index
needed
        if (abs(strike(ti_index))>0 && rightend)    %  See comment above about
threshold
            strike(ti_index)=0;
        else
            rightend=0;
        end
        ti_index=max([mod(t-i,size(strike,2)),1]);
        if (abs(strike(ti_index))>0 && leftend)    %  See comment above about
threshold
            strike(ti_index)=0;
        else
            leftend=0;
        end

        i=i+1;
        again1=rightend+leftend;
    end % again1


    if (max(strike)<=0)
        again2=0;
    end
end  % again2

handles.strike=-strike;
%handles.strikeamp=-strike.*handles.base;   changed 24 June 2020
handles.strikeamp=-strike.*handles.d(2,:);
```

```
%axes(handles.Event_data);
figure
plot(handles.d(1,:),handles.strikeamp,'r.');
%axis(handles.lim);
hold on;
%plot(handles.d(1,:),handles.base,'b');
plot(handles.d(1,:),handles.d(2,:),'b');
plot([1,handles.d(1,end)], [UB,UB],'k-')
title(strcat('Threshold & Upperbound (black lines) with located peaks
',channel'),'FontSize', 12)
legend('Located peaks','Threshold','Location','northwest')
xlabel('Total Concatenated Time (seconds)')
ylabel('Photon counts per time bin')
hold off;

% Store some of the spike data below the threshold for later noise and
% single event prob analysis; zero these data in main propagated data
% variables
handles.subthreshstrike=handles.strike.*(handles.strikeamp < spikethresh);
handles.subthreshstrikeamp=handles.strikeamp.*(handles.strikeamp < spikethresh);
handles.strike=handles.strike.*(handles.strikeamp > spikethresh);
handles.strikeamp = handles.strikeamp.*(handles.strikeamp > spikethresh);

%Consider peaks away from very edge of data set; 15 bin buffer is
%sized for later Gaussian model fit to events
peakloc=find(handles.strikeamp > 0);   %All burst events found
peakloc=peakloc(find(peakloc > 15));
peakloc=peakloc(find(peakloc < size(handles.strikeamp,2)-15));
npeaks=size(peakloc,2);
params=struct('mean',[1:npeaks],'sig',[1:npeaks],'A',[1:npeaks]);


% Check interspike timing statistics and remove events too closely spaced
disp('Checking interspike timing distribution and removing events closer than
corrt')
ISI_check(handles,tbin,corrt)


%Do fit of Gaussian model to each event if dofit flag set in parameter
while (dofit)  %%%%%%Hook to turn off Gaussian fitting - remove datastat=1 too
%%%%%
for i=1:npeaks   *handles.d(2,:)
%    fitme=handles.base(peakloc(i)-10:peakloc(i)+10); %start with general spike zone
    fitme=handles.d(2,peakloc(i)-10:peakloc(i)+10); %start with general spike zone
    dzone=fix(2*sum(fitme)/max(fitme)); % refine zone assuming Gaussian peak
%    fitme=handles.base(peakloc(i)-dzone:peakloc(i)+dzone);
    fitme=handles.d(2,peakloc(i)-dzone:peakloc(i)+dzone);
    x=[peakloc(i)-dzone:peakloc(i)+dzone];
    [A, mu, sigma] = fitgauss(fitme, x, max(fitme), peakloc(i), 3);
    params.mean(i)=mu;
    params.sig(i)=sigma;
    params.A(i)=A/sqrt(2*3.1415*sigma^2); %A is now amplitude
%     hold off; plot(x,fitme,'+');
%     hold on; plot(x,params.A(i)*exp(-(x-mu).^2/2/(params.sig(i)^2)),'r');
%     pause(1);
end %gauss fit
q=find((params.sig<10) &
(params.sig>0)&(params.A<2*max(handles.strikeamp))&(params.A>0));
datastat=struct('mean',params.mean(q),'sig',params.sig(q),'A',params.A(q));
dofit=0;
end %while
datastat=1;

%Reassign large, unused variables
qq=1; d=1; strike=1; zpoints=1; IX=1; MF=1; baseline=1;


end
```

Check intervals between events (ISI)

```
function ISI_check(handles,tbin,corrt)
% Uses events locations in peakloc and timestream stored in handles structure
% to generate interspike interval plots compared to simple single-rate
% poisson model. The sampling time tbin and the correlation time (~width of a
% burst) corrt are used to exclude events that are too close in time.

nsamples=length(handles.strikeamp(:));  %number of sample bins in concatenated data
nhistbins=100; %consider sampletime*nhistbins seconds of event spacing (typically
100ms)
%Check interspike interval using generated time stream and a single
%population model prediction based on random times and same avg rate
striketimesm=randi([1,nsamples],sum(handles.strike),1);
rtm=sort(striketimesm); %chronological order of event bins
peakshm=circshift(rtm,[1,0]);
peakdiffm=rtm - peakshm;  % This spacing (ISI) gives statistical measure of

%For reference, model of single population distribution with same avg event rate
% If multiple species, events should still be uncorrelated like 1 species
% of higher rate
xoutm=[1:nhistbins];
nm=hist(peakdiffm(2:end-1),xoutm);   %hist of random samples for control
figure
semilogy(xoutm*tbin,nm,'-o')
subtitle=strcat("Random event model w/ single event rate: ",...
    num2str(length(rtm)/(tbin*nsamples))," events/sec");
title({'Random Interspike Interval Dist Compared to Data';subtitle})
```

```
xlabel('Time bewteen events')
ylabel('Number of events')
hold on
%Do again with real data shifted for comparison; should yield same powerlaw
%slope; Overplot distributions
qq=(handles.strikeamp(:) > 0);
striketimesd=[1:nsamples]';  %need transpose to switch index order
striketimesd=striketimesd(qq); %indices where strikes occurred
rtd=sort(striketimesd);
peakshd=circshift(rtd,[1,0]);
peakdiffd=rtd - peakshd;
xoutd=[1:nhistbins];
nd=hist(peakdiffd(2:end-1),xoutd);   %hist of random samples for control
semilogy(xoutd*tbin,nd,'-ro')
legend('Noiseless Model','Simulated Data','Location','northeast')


%Find real data interspike intervals that don't violate correlation time
%parameter corrt; keep first event of two that are too close; arbitrary
%Would be better to keep larger amplitude event
dontkeep=(abs(peakdiffd) < round(corrt/tbin)); %logical indexing
rr=striketimesd(dontkeep);
handles.strikeamp(rr)=0;    %zero timing violation amplitude
handles.strike(rr)=0;       % zero the strike flag as well


% Now show with sub correlation time events removed
disp('-----------------------------------------');
ntotevents=sum(handles.strike);
displine=strcat('Removing ',num2str(length(rr)),' intertime violation events from
', ...
num2str(ntotevents),' total events');
disp(displine);
qq=(handles.strikeamp(:) > 0);
striketimesd=[1:nsamples]';  %need transpose to switch index order
striketimesd=striketimesd(qq); %indices where strikes occurred
rtd=sort(striketimesd);
peakshd=circshift(rtd,[1,0]);
peakdiffd=rtd - peakshd;
xoutd=[1:nhistbins];
nd=hist(peakdiffd(2:end-1),xoutd);   %hist of random samples for control
semilogy(xoutd*tbin,nd,'-go')



%Fit exponential model to distribution
mymodel = fittype('a*exp(b*n*x)','problem','n');
opts = fitoptions(mymodel);
set(opts,'normalize','on')
xdata=xoutd*tbin; ydata=nd;
```

```
[fit4,gof4,out4] = fit(xdata(1:end-1)',ydata(1:end-1)',mymodel,opts,'problem',{-1});
hold on;
semilogy(xdata*tbin,fit4(xdata));
hold off;
legend('random control','Input data','Remove timing violations',
'Location','northeast')

hold off
% 'Predicted fraction less than ISI threshold',nm(end)/sum(nm)
% 'Data fraction less than ISI threshold',n(end)/sum(n)


end
```

Outputs code version information

```
function [prepfile, tstr]=Information(versionnum)
% Display version and hold introductory information


prepfile=strcat('Pre_',versionnum);

tstr=datetime('now');
tstr.Format = 'ddMMMyy-HHmmss'

disp(strcat('version number: ',versionnum))
%disp(strcat('Output file name: ',strcat(prepfile,'-Report',string(tstr),'.pdf')))

end
```