NIC Technical Document # 4 2004-05-05
Robert Frank

EEG Blink Removal Using BlinkAway

Introduction

BlinkAway is a collection of MATLAB m-files, based upon the ICA Toolbox [1, 2], designed to remove eyeblinks from EEG data using independent component analysis (ICA). In ICA, EEG data (x) is presented as the linear mixture of a set of statistically independent streams (s), called the temporal activations of the independent components, or independent components for short. Mathematically,

$$x = As$$

where:

- A = m by m mixing matrix,
- s = m by n matrix of independent components, one per row, and
- x = m by n matrix of EEG data with each row corresponding to the output of a single detector and each column representing one time sample from all the detectors.

The i^{th} column vector of the matrix A projects the temporal activation of the i^{th} independent component onto the array of EEG detectors and thus determines the spatial contribution of that component to each detector. Mathematically speaking, the spatial-temporal contribution of the i^{th} component to the EEG data is the outer product of the i^{th} column of A with the ith row of s:

$$x_{IC} = A_i * s_i.$$

With respect to blink removal, eyeblinks ideally represent a stream of activity statistically independent from other cerebral activity and so amenable to capture by a single[1] independent component. The eyeblink activity at the detectors ($x_{EyeBlink}$) is then computed as the outer product of its independent component ($s_{EyeBlink}$) with its corresponding spatial projector ($A_{EyeBlink}$)

$$x_{EyeBlink} = A_{EyeBlink} * s_{EyeBlink},$$

and subsequently removed from the EEG data by a matrix-matrix subtraction to yield blinkfree EEG data ready for further analysis

$$x_{BlinkFree} = x_{Original} - x_{EyeBlink}.$$

Description of the Program

Component M-Files

BlinkAway consists of the following MATLAB script files, listed below in alphabetical order:

1. AutoDriver.m
2. BlinkAway.m
3. Driver.m
4. ManyRuns.m
5. PreFilter.m
6. ReadRaw.m
7. UserInfo.m
8. WriteEvents.m
9. WriteRaw.m

Note: AutoDriver and ManyRuns are used in batch processing mode only.

Program Execution and Flow

To start the program, place all the script files, EEG data files and corresponding blink templates in a single directory[2] on the MATLAB search path, and then type Driver at the MATLAB command prompt. Driver first calls UserInfo, which queries the user on various runtime parameters such as EEG data file name, ICA decomposition method, bad channel / saturated observation processing and data segmentation. UserInfo also calls ReadRaw, which reads in the EEG data from an EGI raw-format file. Upon completion, UserInfo returns control to Driver.

Driver next calls PreFilter, which computes row pointers to the bad channels, column pointers to the saturated observations and, if the user chose to segment the data, column pointers to the segment breaks. Upon completion, PreFilter returns control to Driver.

At this point, Driver writes the EEG data, and event data, if any, to disk and opens several binary scratch files, which will be deleted when the program has finished running. The workspace is also cleared of variables whose contents are no longer needed in the interest of maximizing available memory. The current EEG data or data segment is then reloaded from disk and subsequently passed to the user-specified ICA algorithm, minus any bad channels or saturated observations.

At present, BlinkAway can use one of two algorithms to ICA decompose the data: FastICA [3], with a hyperbolic tangent contrast function, or Infomax [4], via either MATLAB or its faster C implementation[3]. Regardless of which algorithm is called, a successful ICA decomposition, that is, one that

converges, will produce two matrices: a mixing matrix A, and its inverse, the unmixing matrix W. Upon return of control to Driver, the program saves both the matrix A and the independent components, s = Wx, to disk and calls BlinkAway to determine which, if any, of the columns of the matrix A correlate to the blink template.

As a first step in removing blink activity from the EEG data, the relative polarity of the elements representing the EOG detectors in each column of the mixing matrix A is checked to ascertain if it corresponds to either horizontal or vertical eye movements, since correspondence to vertical eye movement is a necessary condition that blink activity must meet. A normalized covariance is then computed between each column of matrix A and the blink template[4], and all of those columns that both meet the polarity requirement and correlate to the blink template, within a user specified range, are flagged as describing a spatial topography at the detectors consistent with blink activity.

As stated in the introduction, each column vector of the mixing matrix A determines the spatial distribution of its corresponding independent component, and can be thought of as the spatial projector for that component. Since we now know which column vectors of matrix A correspond to blink activity, we therefore know that their corresponding independent components contain the temporal activations of those blinks. The blink activity at all detectors for all time is then computed as the outer product of these spatial projectors (m x 1 column vectors, m = # of detectors) with their independent components (1 x n row vectors, n = # of samples), resulting in an m x n matrix of blinks which is subtracted from the original m x n EEG data matrix to give an m x n blink-free EEG data matrix:

$$EEG_{BlinkFree} = EEG_{Original} - EEG_{PureBlinks}.$$

Before finally returning control to Driver, BlinkAway writes the blink-free and pure-blink data arrays to disk.

Driver now reads from disk the event data, mixing matrix A, independent components s, and blink-free and pure-blink data arrays. It then calls WriteRaw, which writes this data to disk in the following EGI raw format files:

- Mixing Matrix A: <FileName_mix>
- Independent Components s: <FileName_ica>
- Blink-Free EEG Data: <FileName_fltrd>
- Extracted Blinks: <FileName_blnks>

Note: FileName is the name of the original raw format EEG data file, sans extension.

At present, NetStation cannot read in EGI raw format files containing event data and successfully parse the event markers. Therefore, if the original EEG

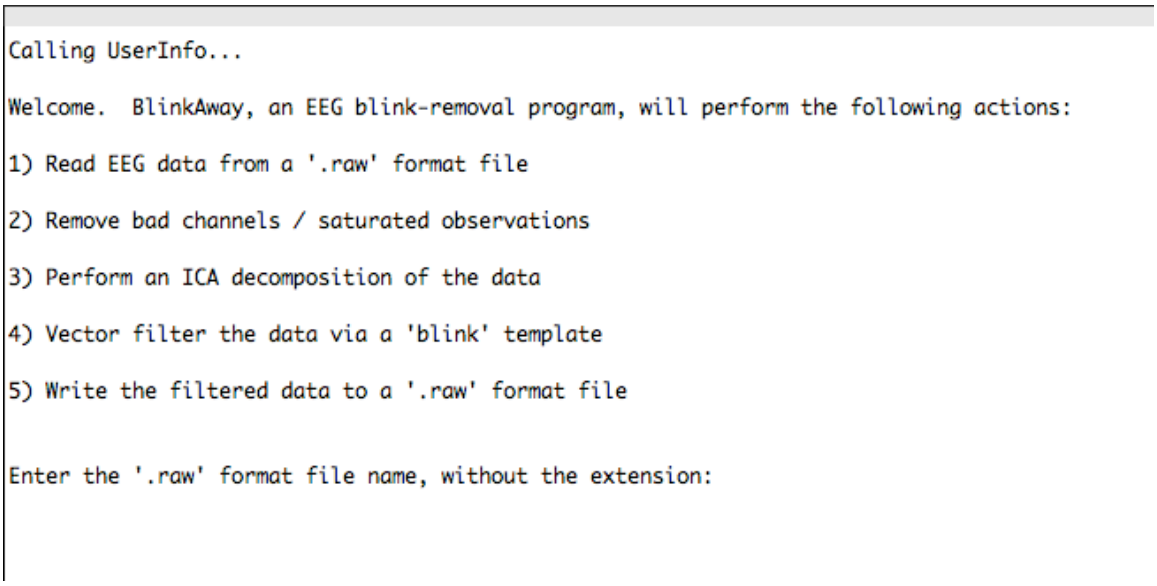
data file contained event markers, they are, at the users request, written to a separate ASCII text file, <FileName_events>, after first removing any event markers corresponding to deleted saturated observations. This separate text file can be read in by Netstation, and its event markers can then be successfully matched to their corresponding events in the EEG data.

Prior to completion, Driver deletes from disk all temporary scratch / working files and creates a directory named <ICA-FileName>, moving into it all the raw format data files that it created.

Command Line Interface

The following is a sequence of images illustrating the questions asked by the program, the ICA algorithm as it is running, and the subsequent decomposition results and blink template matches.

Figure 1: Introduction



Enter the '.raw' format filename, without the extension. Please note that this data file must be in the same directory as the MATLAB script files which implement BlinkAway, and the EEG data signals must be in units of microvolts.

Figure 2: EEG Data File Header Info

```
-------------------------------------------------

ReadRaw Input File: SimBlinkSet5.raw
versionNumber = 4
recordingTimeYear = 0
recordingTimeMonth = 0
recordingTimeDay = 0
recordingTimeHour = 0
recordingTimeMinute = 0
recordingTimeSecond = 0
recordingTimeMillisec = 0
samplingRate = 250
numChan = 34
boardGain = 0
numConvBits = 0
ampRange = 0
numSamples = 59419
numEvents = 0

SimBlinkSet5.raw does not contain event information.

Array eegData: 34 x 59419

< Return > to continue or < Control-C > to quit...
```

Upon entering the file name and pressing <Return>, "BlinkAway" will present you with the header information and the size of the EEG data array.

Figure 3: ICA Decomposition Protocol

```
------------ ICA Decomposition Protocol ------------

1) Fast ICA
2) Infomax ICA (binica)
3) Infomax ICA (runica)

Please select the ICA Decomposition Protocol: 1

ICA Decomposition Protocol: 1

< Return > to continue or < Control-C > to quit...
```

Specify the appropriate ICA Decomposition Protocol, and "BlinkAway" will confirm your choice.

Figure 4: Bad Channel Protocol (No Bad Channels)

```
------------ Bad Channel Protocol ------------

1) Eliminate user-specified bad channels
2) Auto-eliminate bad channels: |Channel Variance| <= Bad Channel Tolerance
3) Eliminate user-specified bad channels + Auto-eliminate remaining bad channels
4) No bad channels

Please select the Bad Channel Protocol: 4

Bad Channel Protocol: 4

< Return > to continue or < Control-C > to quit...
```

Specify the appropriate Bad Channel Protocol, and "BlinkAway" will confirm your choice.

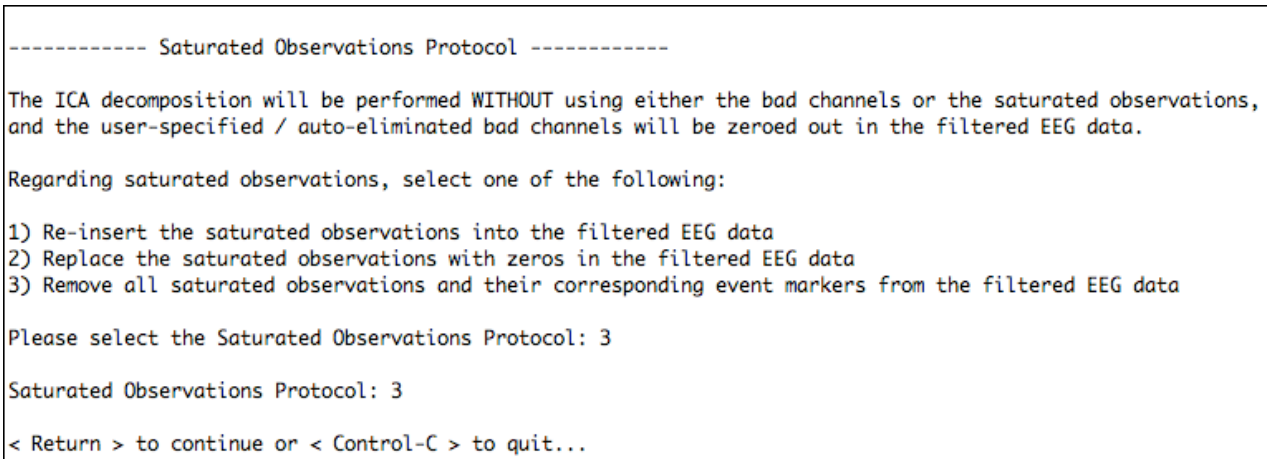
Note: Data pre-processing is typically performed within NetStation.

Figure 5: Bad Channel Protocol (Auto-Eliminate Bad Channels)

```
------------ Bad Channel Protocol ------------

1) Eliminate user-specified bad channels
2) Auto-eliminate bad channels: |Channel Variance| <= Bad Channel Tolerance
3) Eliminate user-specified bad channels + Auto-eliminate remaining bad channels
4) No bad channels

Please select the Bad Channel Protocol: 2

Bad Channel Protocol: 2

Enter the tolerance level for bad channel auto-detection: 7.5

Bad Channel Tolerance: 7.5000

< Return > to continue or < Control-C > to quit...
```

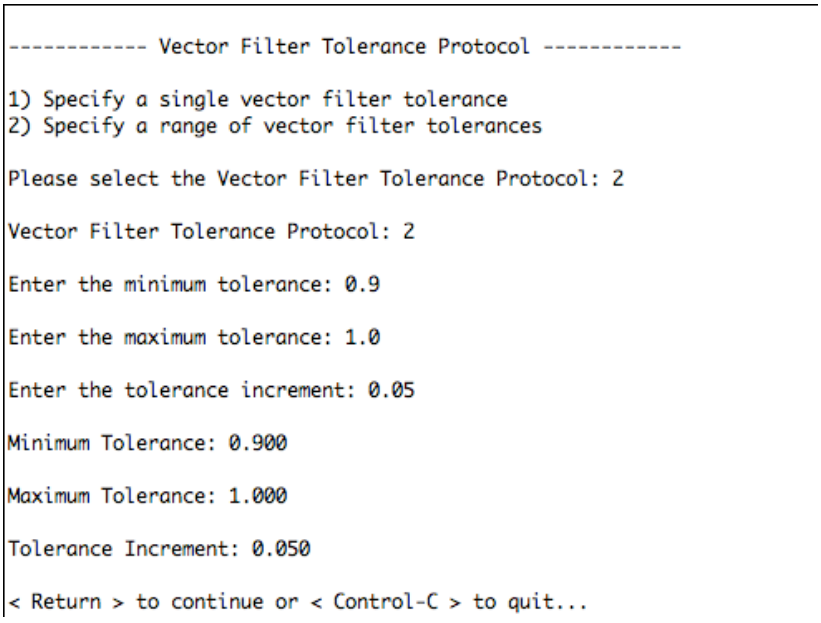
If you choose to select / scan for bad channels, the program will query you for additional information, such as the vector of bad channel markers or the bad channel tolerance (in microvolts).

Figure 6: Saturated Observations Protocol



Specify the appropriate Saturated Observations Protocol, and "BlinkAway" will confirm your choice.

Figure 7: Vector Filter Tolerance Protocol



Specify the appropriate Vector Filter Tolerance Protocol, and "BlinkAway" will confirm your choice. Depending on your selection, the program will query you for the necessary additional information.

Figure 8: Segmentation Protocol

```
------------ Segmentation Protocol ------------

Segment the EEG data (Y/N) ? N

The EEG data will not be segmented.

< Return > to continue or < Control-C > to quit...
```

```
------------ Segmentation Protocol ------------

Segment the EEG data (Y/N) ? Y

Enter the number of segments (+'ve integer >= 2): 2

Number Of EEG Data Segments: 2

< Return > to continue or < Control-C > to quit...
```

If you choose to segment the data, "BlinkAway" will prompt you to enter the number of segments as a positive integer >= 2.

Figure 9: PreFilter Output Log

```
Calling PreFilter...

PreFilter Output Log:

Recording Hardware: TestData.

-------------------------------------------------

Number Of Bad Channels Removed: 0

Good VEOG Channels:
LUVEOG: # 6
RUVEOG: # 2
LLVEOG: # 33
RLVEOG: # 34

-------------------------------------------------

EEG Data Not Segmented | Number Of Saturated Observations: 0

-------------------------------------------------

< Return > to continue or < Control-C > to quit...
```

The PreFilter output log informs you of the recording hardware[5], # of bad channels removed, segmentation protocol and # of saturated observations.

Figure 10: ICA Algorithm Status

```
Calling FastICA (Tanh): EEG Data Not Segmented
-------------------------------------------------------------------------
Number of signals: 34
Number of samples: 59419
Calculating covariance...
Dimension not reduced.
Selected [ 34 ] dimensions.
Smallest remaining (non-zero) eigenvalue [ 0.318416 ]
Largest remaining (non-zero) eigenvalue [ 33005.2 ]
Sum of removed eigenvalues [ 0 ]
[ 100 ] % of (non-zero) eigenvalues retained.
Whitening...
Check: covariance differs from identity by [ 4.46376e-12 ].
Used approach [ defl ].
Used nonlinearity [ tanh ].
Starting ICA calculation...
IC 1 ...................computed ( 20 steps )
IC 2 ..........................................computed ( 44 steps )
IC 3 ......................computed ( 24 steps )
```

The ICA algorithm status screen informs you of the running ICA algorithm, current EEG data segment and the current state (number of independent components found, etc...) of the decomposition.

Figure 11: BlinkAway Results (EOG Polarity Constraints)

```
Calling BlinkAway: EEG Data Not Segmented

BlinkAway Output Log:

Criteria # 1 (HEOG Polarity Inversion) | 8 ICA Activation(s):

003 (0.364)     005 (0.836)     007 (0.374)     008 (0.039)     014 (0.412)
021 (0.418)     024 (0.276)     034 (0.356)

-----------------------------------------------------------

Criteria # 2 (VEOG Polarity Inversion) | 6 ICA Activation(s):

001 (0.906)     004 (0.705)     006 (1.000)     009 (0.223)     023 (0.433)
029 (0.447)

-----------------------------------------------------------
```

The BlinkAway results screen informs you of the channels which met the EOG polarity constraints, both horizontal and vertical, and their correlation to the blink template.

Figure 12: BlinkAway Results (Blink Template Correspondence)

```
Vector Filter Tolerance: 0.900

Criteria # 3 (Blink Template Correlation) | 2 ICA Activation(s):

001 (0.906)     006 (1.000)

Criteria # 2 AND Criteria # 3 (Blink Activity) | 2 ICA Activation(s):

001 (0.906)     006 (1.000)

*** 2 ICA Activation(s) Subtracted From The EEG Data ***


------------------------------------------------------------


Vector Filter Tolerance: 0.950

Criteria # 3 (Blink Template Correlation) | 1 ICA Activation(s):

006 (1.000)

Criteria # 2 AND Criteria # 3 (Blink Activity) | 1 ICA Activation(s):

006 (1.000)

*** 1 ICA Activation(s) Subtracted From The EEG Data ***


------------------------------------------------------------


Vector Filter Tolerance: 1.000

Criteria # 3 (Blink Template Correlation) | 1 ICA Activation(s):

006 (1.000)

Criteria # 2 AND Criteria # 3 (Blink Activity) | 1 ICA Activation(s):

006 (1.000)

*** 1 ICA Activation(s) Subtracted From The EEG Data ***


------------------------------------------------------------


< Return > to continue or < Control-C > to quit...
```

The BlinkAway results screen also informs you of which channels met both the specified template correspondence and the VEOG polarity condition, which together indicate the component(s) representing blink activity.

Figure 13: File Output

```
Preparing to write to .raw formatted output files...
------------------------------------------------------------

Output File: SimBlinkSet5_RawEeg_mix.raw
Output File: SimBlinkSet5_ica.raw

------------------------------------------------------------

Preparing to write to .raw formatted output files...
------------------------------------------------------------

Output File: SimBlinkSet5_tol0.9_fltrd.raw
Output File: SimBlinkSet5_tol0.9_blnks.raw

------------------------------------------------------------

Preparing to write to .raw formatted output files...
------------------------------------------------------------

Output File: SimBlinkSet5_tol0.95_fltrd.raw
Output File: SimBlinkSet5_tol0.95_blnks.raw

------------------------------------------------------------

Preparing to write to .raw formatted output files...
------------------------------------------------------------

Output File: SimBlinkSet5_tol1_fltrd.raw
Output File: SimBlinkSet5_tol1_blnks.raw

------------------------------------------------------------


Please press < Return > to quit...
```

The file output screen informs you of the various data files written to disk. Upon pressing return, the program will move these data output files into a new folder, delete all the temporary scratch files and then quit.

Notes:

1. The blink activity can be contained in two or more components, although optimal results have been achieved when it is captured by just one.

2. In future revisions, the files will not all need to be placed within the same directory.

3. The directories containing the FastICA and EEGLAB script files must be added to the MATLAB search path.

4. A blink template (NicTR2004-003) must be saved in a text file named vctrFltr.txt in the working directory.

5. The recording hardware (Ex: Net256) must be defined in PreFilter.m and referenced in UserInfo.m. (See these .m files for details.)

Script M Files

Driver.m

```
% ICA Blink Removal - Driver Program

global header eegData outputLog_bA outputLog_pF outputLog_uI outputLog_wR outputLog_wE;
numBlinkPntrs = []; outputLog_ICA = []; outputLog_oF = [];

% Fetch various runtime parameters.-----------------------------------------------------------------------------------

clc; usrMsg = sprintf('\nCalling UserInfo...'); disp(usrMsg);

[fileName, EEG, precision, badChanProtocol, badChanPntr, badChanTol, ...
   satObsProtocol, icaProtocol, vctrFltrTol, segment, numSegment, ...
   fileStem, messageStem, vftFileStem, numVctrFltrTol, autoInputMode] = userInfo;

usrMsg = sprintf('\nUserInfo Output Log:\n');
outputLog_uI = strvcat(usrMsg, outputLog_uI);

if (~autoInputMode)
   usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause;
end

% Remove bad channels / saturated observations. ----------------------------------------------------------------------

clc; usrMsg = sprintf('\nCalling PreFilter...'); disp(usrMsg);

[goodChanPntr, badChanPntr, numGoodChan, goodObsPntr, badObsPntr, numGoodObs, ...
   segStart, segStop, segSize, VEOG, numVEOG] = preFilter(fileName, EEG, ...
   badChanProtocol, badChanTol, badChanPntr, ...
   segment, numSegment, messageStem);

usrMsg = sprintf('\nPreFilter Output Log:\n');
outputLog_pF = strvcat(usrMsg, outputLog_pF);
disp(outputLog_pF);

if (~autoInputMode)
   usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause;
end

% Housekeeping. ------------------------------------------------------------------------------------------------------

numChan = header.numChan;
numSamples = header.numSamples;

logFile = [fileName '_Logs'];
save (logFile, 'outputLog_uI', 'outputLog_pF');
clear global outputLog_uI; clear global outputLog_pF;

% Save EEG data to disk and open binary scratch files. ---------------------------------------------------------------

for i = 1 : numSegment

   fidTmp = fopen([fileName fileStem{i}], 'w');

   if (fidTmp == -1)
      errMsg = sprintf('\n!!! File Creation Error: %s !!!\n', [fileName fileStem{i}]); error(errMsg);
   end

   fwrite(fidTmp, eegData(goodChanPntr, segStart(i):segStop(i)), precision);

   fclose(fidTmp);

end

clear global eegData;
```

```
for i = 1 : numVctrFltrTol

   fid(i) = fopen([fileName vftFileStem{i}], 'w+');

   if (fid(i) == -1)
      errMsg = sprintf('\n!!! File Creation Error: %s !!!\n', [fileName vftFileStem{i}]); error(errMsg);
   end

end

fid(numVctrFltrTol + 1) = fopen([fileName '_mix'], 'w+');
if (fid(numVctrFltrTol + 1) == -1)
   errMsg = sprintf('\n!!! File Creation Error: %s !!!\n', [fileName '_mix']); error(errMsg);
end

fid(numVctrFltrTol + 2) = fopen([fileName '_ica'], 'w+');
if (fid(numVctrFltrTol + 2) == -1)
   errMsg = sprintf('\n!!! File Creation Error: %s !!!\n', [fileName '_ica']); error(errMsg);
end


% Call ICA decomposition and vector filtering algorithm. ----------------------------------------------------------------


switch icaProtocol

   case 1

      for i = 1 : numSegment

         clc; usrMsg = sprintf('\nCalling FastICA (Tanh): %s', messageStem{i}); disp(usrMsg);
         disp('-------------------------------------------------------------------------');
         outputLog_ICA = strvcat(outputLog_ICA, usrMsg);

         fidTmp = fopen([fileName fileStem{i}]);

         if (fidTmp == -1)
            errMsg = sprintf('\n!!! File Access Error: %s !!!\n', [fileName fileStem{i}]); error(errMsg);
         end

         eegDataLocal = fread(fidTmp, [numGoodChan, segSize(i)], precision);

         fclose(fidTmp);

         [A, W] = fastica(eegDataLocal(:, goodObsPntr{i}), 'displayMode', 'off', 'g', 'tanh');

         fwrite(fid(numVctrFltrTol + 1), A, precision);
         fwrite(fid(numVctrFltrTol + 2), W * eegDataLocal(:, goodObsPntr{i}), precision);

         clear eegDataLocal;

         clc; usrMsg = sprintf('\nCalling BlinkAway: %s', messageStem{i}); disp(usrMsg);

         [numBlinkPntrsSeg] = blinkAway(fileName, fileStem{i}, fid, precision, ...
            goodChanPntr, badChanPntr, numGoodChan, goodObsPntr{i}, badObsPntr{i}, ...
            satObsProtocol, vctrFltrTol, segSize(i), segStart(i), segStop(i), ...
            A, W, VEOG, numVEOG, numVctrFltrTol, vftFileStem);

         numBlinkPntrs = [numBlinkPntrs numBlinkPntrsSeg];

         usrMsg = sprintf('\nBlinkAway Output Log:\n'); disp(usrMsg); disp(outputLog_bA);
         outputLog_ICA = strvcat(outputLog_ICA, usrMsg, outputLog_bA);

         if (~autoInputMode)
            usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause;
         end

      end
```

```
   case {2, 3}

      for i = 1 : numSegment

         clc; usrMsg = sprintf('\nCalling InfoMax ICA: %s', messageStem{i}); disp(usrMsg);
         disp('-------------------------------------------------------------------------------');
         outputLog_ICA = strvcat(outputLog_ICA, usrMsg);

         fidTmp = fopen([fileName fileStem{i}]);

         if (fidTmp == -1)
            errMsg = sprintf('\n!!! File Access Error: %s !!!\n', [fileName fileStem{i}]); error(errMsg);
         end

         eegDataLocal = fread(fidTmp, [numGoodChan, segSize(i)], precision);

         fclose(fidTmp);

         if icaProtocol == 2
            [Wght, Sph] = binica(eegDataLocal(:, goodObsPntr{i}));
         else
            [Wght, Sph] = runica(eegDataLocal(:, goodObsPntr{i}));
         end

         W = Wght * Sph;
         A  = inv(W);

         fwrite(fid(numVctrFltrTol + 1), A, precision);
         fwrite(fid(numVctrFltrTol + 2), W * eegDataLocal(:, goodObsPntr{i}), precision);

         clear eegDataLocal;

         clc; usrMsg = sprintf('\nCalling BlinkAway: %s', messageStem{i}); disp(usrMsg);

         [numBlinkPntrsSeg] = blinkAway(fileName, fileStem{i}, fid, precision, ...
            goodChanPntr, badChanPntr, numGoodChan, goodObsPntr{i}, badObsPntr{i}, ...
            satObsProtocol, vctrFltrTol, segSize(i), segStart(i), segStop(i), ...
            A, W, VEOG, numVEOG, numVctrFltrTol, vftFileStem);

         numBlinkPntrs = [numBlinkPntrs numBlinkPntrsSeg];

         usrMsg = sprintf('\nBlinkAway Output Log:\n'); disp(usrMsg); disp(outputLog_bA);
         outputLog_ICA = strvcat(outputLog_ICA, usrMsg, outputLog_bA);

         if (~autoInputMode)
            usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause;
         end

      end

end

% Housekeeping. -----------------------------------------------------------------------------------------------

save (logFile, 'outputLog_ICA', '-append');
clc; clear global outputLog_bA; clear outputLog_ICA;

% Load event data, if any. ------------------------------------------------------------------------------------

if (header.numEvents ~= 0)
   load([fileName '_events.mat']);
else
   eventData = [];
end

% Deleted bad observations? Recalculate segStart, segStop and numSamples. -------------------------------------

if (satObsProtocol == 3)

   segStart = cumsum([1, numGoodObs(1 : numSegment - 1)]); segStop = cumsum(numGoodObs(1 : numSegment));
```

```
   segSize = numGoodObs; numSamples = sum(numGoodObs); header.numSamples = numSamples;

   if (header.numEvents ~= 0)

      for i = 1 : numSegment
         eventData(:, sum(segSize(1 : i - 1)) + badObsPntr{i}) = [];
      end

   end

end

% Read in mixing matrix A & IC from disk and then write to .raw format. ----------------------------------------------------

A = zeros(numChan, numChan); s = zeros(numChan, sum(numGoodObs));

filStatus1 = fseek(fid(numVctrFltrTol + 1), 0, 'bof'); filStatus2 = fseek(fid(numVctrFltrTol + 2), 0, 'bof');

if (filStatus1 == 0) && (filStatus2 == 0)
   usrMsg = sprintf(['\nPreparing to write to .raw formatted output files...' ...
                     '\n-------------------------------------------------------------\n']); disp(usrMsg);
else
   errMsg = sprintf('\n!!! BOF Seek Failure: A / IC Data Arrays !!!\n'); error(errMsg);
end

for i = 1 : numSegment

   A(goodChanPntr, goodChanPntr) = fread(fid(numVctrFltrTol + 1), [numGoodChan, numGoodChan], precision);

   writeRaw([fileName fileStem{i} '_mix'], [], A, [], 0, numChan, numChan, header.samplingRate, header.versionNumber);

   disp(outputLog_wR); outputLog_oF = strvcat(outputLog_oF, outputLog_wR);

end

s(goodChanPntr, :) = fread(fid(numVctrFltrTol + 2), [numGoodChan, sum(numGoodObs)], precision);

writeRaw([fileName '_ica'], [], s, [], 0, numChan, sum(numGoodObs), header.samplingRate, header.versionNumber);

disp(outputLog_wR);  outputLog_oF = strvcat(outputLog_oF, outputLog_wR);

usrMsg = sprintf('\n-------------------------------------------------------------\n'); disp(usrMsg);

% Housekeeping. ------------------------------------------------------------------------------------------------------------

clear s;

% Read in filtered EEG data & extracted 'blinks' from disk and then write to .raw format. ----------------------------------

for i = 1 : numVctrFltrTol

   filStatus = fseek(fid(i), 0, 'bof');

   if (filStatus == 0)
      usrMsg = sprintf(['Preparing to write to .raw formatted output files...' ...
                        '\n-------------------------------------------------------------\n']); disp(usrMsg);
   else
      errMsg = sprintf('\n!!! BOF Seek Failure: Filtered EEG / Pure Blinks Data Arrays !!!\n'); error(errMsg);
   end

   if all(numBlinkPntrs(i, :))

      filteredEeg = zeros(numChan, numSamples); pureBlinks = zeros(numChan, numSamples);

      for j = 1 : numSegment
         filteredEeg(:, segStart(j):segStop(j)) = fread(fid(i), [numChan, segSize(j)], precision);
         pureBlinks(:, segStart(j):segStop(j)) = fread(fid(i), [numChan, segSize(j)], precision);
      end
```

```
        writeRaw([fileName vftFileStem{i} '_fltrd'], header, filteredEeg, eventData);
        disp(outputLog_wR);  outputLog_oF = strvcat(outputLog_oF, outputLog_wR);
        writeRaw([fileName vftFileStem{i} '_blnks'], header, pureBlinks, eventData);
        disp(outputLog_wR);  outputLog_oF = strvcat(outputLog_oF, outputLog_wR);

    else

        usrMsg = sprintf(['Vector Filter Tolerance: %6.4f\n' ...
                    'No ICA activations met the filtering criteria in 1 (or more) segment(s): ' ...
                    'No filtered data written out.'], vctrFltrTol(i));
        disp(usrMsg);  outputLog_oF = strvcat(outputLog_oF, usrMsg);

    end

    usrMsg = sprintf('\n---------------------------------------------------------------\n'); disp(usrMsg);

end

if (header.numEvents ~= 0)
    if (~autoInputMode)
        netStationEvents = input('Write event data to NetStation formatted text file (Y/N)? ','s');
    else
        netStationEvents = 'Y';
    end
    if ((netStationEvents == 'y') || (netStationEvents == 'Y'))
        usrMsg = sprintf('\nPreparing to write NetStation formatted event data to disk...'); disp(usrMsg);
        writeEvents([fileName '_events'], header, eventData);
        disp(outputLog_wE); outputLog_oF = strvcat(outputLog_oF, outputLog_wE);
    end
end

% Housekeeping. ------------------------------------------------------------------------------------------------

filStatus = fclose('all');

if (filStatus == -1)
    errMsg = sprintf('\n!!! File Error: Files Failed To Close !!!\n'); error(errMsg);
end

save (logFile, 'outputLog_oF', '-append');

for i = 1 : numSegment
    eval(['! rm ' fileName fileStem{i}]);
end

for i = 1 : numVctrFltrTol
    eval(['! rm ' fileName vftFileStem{i}]);
end

if (header.numEvents ~= 0)
    eval(['! rm ' fileName '_events.mat']);
end

if (icaProtocol == 2)
    eval('! rm binica*');
    eval('! rm bias_after_adjust');
end

eval(['! rm ' fileName '_mix']);
eval(['! rm ' fileName '_ica']);

eval(['! mkdir ICA-' fileName]);
eval(['! mv ' fileName '* ICA-' fileName]);

if (~autoInputMode)
    usrMsg = sprintf('\nPlease press < Return > to quit...'); disp(usrMsg); pause;
end

clc; clear outputLog_oF; clear global outputLog_wE; clear global outputLog_wR;
```

UserInfo.m

```
function [fileName, EEG, precision, badChanProtocol, badChanPntr, badChanTol, ...
          satObsProtocol, icaProtocol, vctrFltrTol, segment, numSegment, ...
          fileStem, messageStem, vftFileStem, numVctrFltrTol, autoInputMode] = userInfo;

global header eegData outputLog_uI outputLog_rR;

% Auto Mode ---------------------------------------------------------------------------------------------------------

if (exist('AutoDriver.m', 'file') == 2)

   AutoDriver;
   readRaw(fileName);

   outputLog_uI = outputLog_rR; clear global outputLog_rR;

   switch header.versionNumber
     case 2
        precision = 'integer*2';  numBytes = 2;  % Integer
     case 4
        precision = 'real*4';  numBytes = 4;  % Single Precision Real
     case 6
        precision = 'real*8';  numBytes = 8;  % Double Precision Real
   end

   if (vctrFltrProtocol == 2)
      vctrFltrNumInc = round((vctrFltrTolMax - vctrFltrTolMin) / vctrFltrTolInc);
      vctrFltrTolInc = (vctrFltrTolMax - vctrFltrTolMin) / vctrFltrNumInc;
      vctrFltrTol = vctrFltrTolInc * [0 : vctrFltrNumInc] + vctrFltrTolMin;
   end

   numVctrFltrTol = length(vctrFltrTol);

   for i = 1 : numVctrFltrTol
      vftFileStem{i} = ['_tol' num2str(vctrFltrTol(i))];
   end

   if ((segment =='y') | (segment == 'Y'))

      segment = 1;

      for i = 1 : numSegment
         fileStem{i} = ['_RawEegSeg' int2str(i)];
         messageStem{i} = ['EEG Data Segment # ' int2str(i)];
      end

   else

     segment = 0; numSegment = 1;
     fileStem{1} = '_RawEeg';
     messageStem{1} = 'EEG Data Not Segmented';

   end

   autoInputMode = 1;
   return;

else

   autoInputMode = 0;
   EEG = 'TestData';
end

% ------------------------------------------------------------------------------------------------------------------
%
% Blink Away Intro --------------------------------------------------------------------------------------------------
%
% ------------------------------------------------------------------------------------------------------------------
```

```
usrMsg = sprintf(['\nWelcome.  BlinkAway, an EEG blink-removal program, will perform the following actions:\n' ...
       '\n1) Read EEG data from a ".raw" format file\n' ...
       '\n2) Remove bad channels / saturated observations\n' ...
       '\n3) Perform an ICA decomposition of the data\n' ...
       '\n4) Vector filter the data via a "blink" template\n' ...
       '\n5) Write the filtered data to a ".raw" format file\n']);
disp(usrMsg);

fileName = input('\nEnter the ".raw" format file name, without the extension: ', 's');

if ~isempty(findstr('.', fileName))
   errMsg = sprintf('\n!!! Next Time, Please Do Not Include A "." In The File Name !!!\n');
   error(errMsg);
end

% Call readRaw ------------------------------------------------------------------------------------------------------

readRaw(fileName);
outputLog_ui = outputLog_rR; clear global outputLog_rR;
usrMsg = sprintf('\n-----------------------------------------------------\n'); disp(usrMsg); disp(outputLog_ui);

switch header.versionNumber
  case 2
     precision = 'integer*2';  numBytes = 2;  % Integer
  case 4
     precision = 'real*4';  numBytes = 4;  % Single Precision Real
  case 6
     precision = 'real*8';  numBytes = 8;  % Double Precision Real
end

% ICA Decomposition Protocol ----------------------------------------------------------------------------------------

usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause; clc;
usrMsg = sprintf(['\n------------ ICA Decomposition Protocol ------------\n' ...
       '\n1) Fast ICA' ...
       '\n2) Infomax ICA (binica)' ...
       '\n3) Infomax ICA (runica)']);
disp(usrMsg);

icaProtocol = input('\nPlease select the ICA Decomposition Protocol: ');

switch icaProtocol
   case {1, 2, 3}
      usrMsg = sprintf('\nICA Decomposition Protocol: %d', icaProtocol);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
   otherwise
      errMsg = sprintf('\n!!! Next Time, Please Select Protocol # 1, 2, Or 3 !!!\n');
      error(errMsg);
end

% Bad Channel Protocol ----------------------------------------------------------------------------------------------

usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause; clc;
usrMsg = sprintf(['\n------------ Bad Channel Protocol ------------\n' ...
       '\n1) Eliminate user-specified bad channels' ...
       '\n2) Auto-eliminate bad channels: |Channel Variance| <= Bad Channel Tolerance' ...
       '\n3) Eliminate user-specified bad channels + Auto-eliminate remaining bad channels' ...
       '\n4) No bad channels']);
disp(usrMsg);

badChanProtocol = input('\nPlease select the Bad Channel Protocol: ');

switch badChanProtocol
   case {1, 2, 3, 4}
      usrMsg = sprintf('\nBad Channel Protocol: %d', badChanProtocol);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
   otherwise
      errMsg = sprintf('\n!!! Next Time, Please Select Protocol # 1, 2, 3 Or 4 !!!\n');
      error(errMsg);
end
```

```
switch badChanProtocol
   case 1
      badChanTol = 0;
      badChanPntr = input('\nEnter the vector of bad channel markers ([Channel# Channel# Channel# ...]): ');
      usrMsg = strvcat(sprintf('\nUser-Specified Bad Channel(s):'), sprintf('Channel # %d\n', badChanPntr));
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
   case 2
      badChanPntr = [];
      badChanTol = abs(input('\nEnter the tolerance level for bad channel auto-detection: '));
      usrMsg = sprintf('\nBad Channel Tolerance: %6.4f', badChanTol);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
   case 3
      badChanPntr = input('\nEnter the vector of bad channel markers ([Channel# Channel# Channel# ...]): ');
      badChanTol = abs(input('\nEnter the tolerance level for bad channel auto-detection: '));
      usrMsg = strvcat(sprintf('\nBad Channel Tolerance: %6.4f', badChanTol), ...
                  sprintf('\nUser-Specified Bad Channel(s):'), sprintf('Channel # %d\n', badChanPntr));
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
   case 4
      badChanPntr = []; badChanTol = 0;
end

% Saturated Observations Protocol ----------------------------------------------------------------------------------

switch badChanProtocol
   case {1, 3}
      usrMsg = sprintf('< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause; clc;
   case {2, 4}
      usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause; clc;
end
usrMsg = sprintf(['\n---------- Saturated Observations Protocol ----------\n' ...
      '\nThe ICA decomposition will be performed WITHOUT using either the bad channels or the saturated
observations,' ...
      '\nand the user-specified / auto-eliminated bad channels will be zeroed out in the filtered EEG data.\n' ...
      '\nRegarding saturated observations, select one of the following:\n' ...
      '\n1) Re-insert the saturated observations into the filtered EEG data' ...
      '\n2) Replace the saturated observations with zeros in the filtered EEG data' ...
      '\n3) Remove all saturated observations and their corresponding event markers from the filtered EEG data']);
disp(usrMsg);

satObsProtocol = input('\nPlease select the Saturated Observations Protocol: ');

switch satObsProtocol
   case {1, 2, 3}
      usrMsg = sprintf('\nSaturated Observations Protocol: %d', satObsProtocol);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
   otherwise
      errMsg = sprintf('\n!!! Next Time, Please Select Protocol # 1, 2 Or 3 !!!\n');
      error(errMsg);
end

% Vector Filter Tolerance Protocol ---------------------------------------------------------------------------------

usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause; clc;
usrMsg = sprintf(['\n---------- Vector Filter Tolerance Protocol ----------\n' ...
      '\n1) Specify a single vector filter tolerance' ...
      '\n2) Specify a range of vector filter tolerances']);
disp(usrMsg);

vctrFltrProtocol = input('\nPlease select the Vector Filter Tolerance Protocol: ');

switch vctrFltrProtocol
   case {1, 2}
      usrMsg = sprintf('\nVector Filter Tolerance Protocol: %d', vctrFltrProtocol);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
   otherwise
      errMsg = sprintf('\n!!! Next Time, Please Select Protocol # 1 Or 2 !!!\n');
      error(errMsg);
end
```

```
switch vctrFltrProtocol
   case 1
      vctrFltrTol = abs(input('\nEnter the vector filter tolerance: '));
      usrMsg = sprintf('\nVector Filter Tolerance: %4.3f', vctrFltrTol);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
   case 2
      vctrFltrTolMin = abs(input('\nEnter the minimum tolerance: '));
      vctrFltrTolMax = abs(input('\nEnter the maximum tolerance: '));
      if (vctrFltrTolMax <= vctrFltrTolMin)
         errMsg = sprintf('\n!!! The Maximum Tolerance Must Be > Minimum Tolerance !!!\n');
         error(errMsg);
      end
      vctrFltrTolInc = abs(input('\nEnter the tolerance increment: '));
      if (vctrFltrTolInc > (vctrFltrTolMax - vctrFltrTolMin))
         errMsg = sprintf('\n!!! Next Time, Please Enter A Smaller Tolerance !!!\n');
         error(errMsg);
      end
      vctrFltrNumInc = round((vctrFltrTolMax - vctrFltrTolMin) / vctrFltrTolInc);
      vctrFltrTolInc = (vctrFltrTolMax - vctrFltrTolMin) / vctrFltrNumInc;
      vctrFltrTol = vctrFltrTolInc * [0 : vctrFltrNumInc] + vctrFltrTolMin;
      usrMsg = sprintf('\nMinimum Tolerance: %4.3f', vctrFltrTolMin);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
      usrMsg = sprintf('\nMaximum Tolerance: %4.3f', vctrFltrTolMax);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
      usrMsg = sprintf('\nTolerance Increment: %4.3f', vctrFltrTolInc);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
end

numVctrFltrTol = length(vctrFltrTol);

for i = 1 : numVctrFltrTol
   vftFileStem{i} = ['_tol' num2str(vctrFltrTol(i))];
end

% Segmentation Protocol ----------------------------------------------------------------------------------------------

usrMsg = sprintf('\n< Return > to continue or < Control-C > to quit...'); disp(usrMsg); pause; clc;
usrMsg = sprintf('\n------------ Segmentation Protocol ------------\n');
disp(usrMsg);

segment = input('Segment the EEG data (Y/N) ? ', 's');

if ((segment =='y') | (segment == 'Y'))

   numSegment = input('\nEnter the number of segments (+''ve integer >= 2): ');

   if ((round(numSegment) == numSegment) && (numSegment >= 2))

      segment = 1;
      usrMsg = sprintf('\nNumber Of EEG Data Segments: %d', numSegment);
      disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);

      for i = 1 : numSegment
         fileStem{i} = ['_RawEegSeg' int2str(i)];
         messageStem{i} = ['EEG Data Segment # ' int2str(i)];
      end

   else

      errMsg = ('\n!!! The Number Of Segments Must Be A Positive Integer >= 2 !!!\n');
      error(errMsg);

   end

else

   segment = 0; numSegment = 1;
   usrMsg = sprintf('\nThe EEG data will not be segmented.');
   disp(usrMsg); outputLog_ui = strvcat(outputLog_ui, usrMsg);
```

```
   fileStem{1} = '_RawEeg';
   messageStem{1} = 'EEG Data Not Segmented';

end
```

ReadRaw.m

```
function readRaw(fileName);

% readRaw(fileName) reads an EGI epoch-marked raw format file.
%
% Epoch-marked raw format: Unsegmented simple binary format (version #
%                                    2, 4 or 6) with event codes <epoc> and <tim0>.
%
% The single sample records (SSR) are extracted to form the eegData matrix, where
% each column of the matrix (array) is one SSR.
% The  corresponding event records, one per SSR, are extracted to form the
% eventData matrix, where each column of the matrix (array) is one event record.
%
% Input Arguments: fileName - Name of the EGI epoch-marked raw format
%                             file, without the .raw extension.
%                             It must be a MATLAB string.
%
% Output Arguments: header - MATLAB structure array containing the header info.
%
%                   eegData - m1 by n array containing the temporal EEG data.
%                             m1 = # of channels, n = # of time samples.
%
%                   eventData - m2 by n array containing the corresponding event info.
%                               m2 = # of event types, n = # of time samples.
%
%                   outputLog_rR - Character array of header, EEG data and event data info.

global header eegData outputLog_rR;

% Open the data file. ----------------------------------------------------------------------------------------------------

fid = fopen([fileName '.raw'], 'r', 'b');

if (fid == -1)
   errMsg = sprintf('\n!!! File Access Error: %s !!!\n', [fileName '.raw']); error(errMsg);
end

usrMsg = sprintf('ReadRaw Input File: %s', [fileName '.raw']); outputLog_rR = usrMsg;

% Read the header info into the structure variable 'header'. ------------------------------------------------------------

header.versionNumber = fread(fid, 1, 'integer*4');
usrMsg = sprintf('versionNumber = %d', header.versionNumber);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

switch header.versionNumber
  case {3, 5, 7}
     errMsg = sprintf('\n!!! Unsegmented EEG Data Only !!!\n'); error(errMsg);
end

header.recordingTimeYear = fread(fid, 1, 'integer*2');
usrMsg = sprintf('recordingTimeYear = %d', header.recordingTimeYear);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.recordingTimeMonth = fread(fid, 1, 'integer*2');
usrMsg = sprintf('recordingTimeMonth = %d', header.recordingTimeMonth);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.recordingTimeDay = fread(fid, 1, 'integer*2');
usrMsg = sprintf('recordingTimeDay = %d', header.recordingTimeDay);
outputLog_rR = strvcat(outputLog_rR, usrMsg);
```

```
header.recordingTimeHour = fread(fid, 1, 'integer*2');
usrMsg = sprintf('recordingTimeHour = %d', header.recordingTimeHour);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.recordingTimeMinute = fread(fid, 1, 'integer*2');
usrMsg = sprintf('recordingTimeMinute = %d', header.recordingTimeMinute);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.recordingTimeSecond = fread(fid, 1, 'integer*2');
usrMsg = sprintf('recordingTimeSecond = %d', header.recordingTimeSecond);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.recordingTimeMillisec = fread(fid, 1, 'integer*4');
usrMsg = sprintf('recordingTimeMillisec = %d', header.recordingTimeMillisec);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.samplingRate = fread(fid, 1, 'integer*2');
usrMsg = sprintf('samplingRate = %d', header.samplingRate);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.numChan = fread(fid, 1, 'integer*2');
usrMsg = sprintf('numChan = %d', header.numChan);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.boardGain = fread(fid, 1, 'integer*2');
usrMsg = sprintf('boardGain = %d', header.boardGain);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.numConvBits = fread(fid, 1, 'integer*2');
usrMsg = sprintf('numConvBits = %d', header.numConvBits);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.ampRange = fread(fid, 1, 'integer*2');
usrMsg = sprintf('ampRange = %d', header.ampRange);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.numSamples = fread(fid, 1, 'integer*4');
usrMsg = sprintf('numSamples = %d', header.numSamples);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

header.numEvents = fread(fid, 1, 'integer*2');
usrMsg = sprintf('numEvents = %d', header.numEvents);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

if (header.numEvents ~= 0)  % File contains event info.

   for i = 1:header.numEvents
      header.eventCodes(i) = {fread(fid, [1 4], 'uchar')};
      usrMsg = sprintf('eventCode # %d  = %s', i, char(header.eventCodes{i}));
      outputLog_rR = strvcat(outputLog_rR, usrMsg);
   end

else  % File does not contain event info.

   usrMsg = sprintf...
     ('\n%s does not contain event information.\n', [fileName '.raw']);
   outputLog_rR = strvcat(outputLog_rR, usrMsg);

end

% Finished reading the header. ---------------------------------------------------------------------------------------

switch header.versionNumber
  case 2
     precision = 'integer*2';  % Integer
  case 4
     precision = 'real*4';  % Single Precision Real
  case 6
     precision = 'real*8';  % Double Precision Real
end
```

```
% Read SSRs into array eegData and corresponding events into array eventData. ----------------------------------------

eegDataCount = 0; eventDataCount = 0;
eegData = zeros(header.numChan, header.numSamples);

if (header.numEvents ~= 0)  % File contains event info.

   eventData = zeros(header.numEvents, header.numSamples);

   for j = 1:header.numSamples
      [eegData(:, j), eegTempCount] = fread(fid, header.numChan, precision);
      [eventData(:, j), eventTempCount] = fread(fid, header.numEvents, precision);
      eegDataCount = eegDataCount + eegTempCount;
      eventDataCount = eventDataCount + eventTempCount;
   end

else  % File does not contain event info.

   eventData = [];
   [eegData, eegDataCount] = fread(fid, [header.numChan, header.numSamples], precision);

end

% Finished reading SSRs and event data. ---------------------------------------------------------------------------

if (header.numEvents ~= 0)  % File contains event info.

   if ((eegDataCount ~= header.numChan * header.numSamples) || ...
        (eventDataCount ~= header.numEvents * header.numSamples))

      errMsg = sprintf('\n!!! Data Read Failure: EEG / Event Data Arrays !!!\n'); error(errMsg);

   end

else  % File does not contain event info.

   if (eegDataCount ~= header.numChan * header.numSamples)

      errMsg = sprintf('\n!!! Data Read Failure: EEG Data Array !!!\n'); error(errMsg);

   end

end

usrMsg = sprintf('Array eegData: %s', [int2str(header.numChan) ' x ' int2str(header.numSamples)]);
outputLog_rR = strvcat(outputLog_rR, usrMsg);

if (header.numEvents ~= 0)  % File contains event info.

   usrMsg = sprintf('Array eventData: %s', [int2str(header.numEvents) ' x ' int2str(header.numSamples)]);
   outputLog_rR = strvcat(outputLog_rR, usrMsg);

end

% Finished error checking. ----------------------------------------------------------------------------------------

fclose(fid);

if (header.numEvents ~= 0)  % File contains event info.

   save([fileName '_events'], 'eventData');
   clear eventData;

end
```

PreFilter.m

```
function [goodChanPntr, badChanPntr, numGoodChan, goodObsPntr, badObsPntr, numGoodObs, ...
          segStart, segStop, segSize, VEOG, numVEOG] = preFilter(fileName, EEG, ...
          badChanProtocol, badChanTol, badChanPntr, ...
          segment, numSegment, messageStem);

global header eegData outputLog_pF;
numChan = header.numChan; numSamples = header.numSamples;

% Determine the type of recording hardware. ----------------------------------------------------------------------

switch EEG
   case 'Net256'
      LUVEOG = 36;
      RUVEOG = 18;
      LLVEOG = 242;
      RLVEOG = 241;
      minAmp = -32768;
      maxAmp = 32767;
   case 'TestData'
      LUVEOG = 6;
      RUVEOG = 2;
      LLVEOG = 33;
      RLVEOG = 34;
      minAmp = -32768;
      maxAmp = 32768;
   otherwise
      errMsg = sprintf('\n!!! Data For %s Is Not Available !!!\n', EEG);
      error(errMsg);
end
usrMsg = sprintf('Recording Hardware: %s.', EEG); outputLog_pF = usrMsg;

usrMsg = sprintf('\n-----------------------------------------------------'); outputLog_pF = strvcat(outputLog_pF, usrMsg);

% Store VEOG channels in character array VEOGstr. ----------------------------------------------------------------

VEOG(1) = LUVEOG; VEOG(2) = RUVEOG; VEOG(3) = LLVEOG; VEOG(4) = RLVEOG;
VEOGstr = strvcat(['LUVEOG: # ' int2str(LUVEOG)], ['RUVEOG: # ' int2str(RUVEOG)], ...
                  ['LLVEOG: # ' int2str(LLVEOG)], ['RLVEOG: # ' int2str(RLVEOG)]);
numVEOG = length(VEOG);

% Determine the rows of the eegData matrix corresponding to 'bad' channels. ---------------------------------------

switch badChanProtocol

   case 1
      numUsrMrkdBadChan = length(badChanPntr);
      goodChanPntr = 1:numChan; goodChanPntr(badChanPntr) = [];
      usrMsg = strvcat(sprintf('\nNumber Of User-Specified Bad Channels: %d', numUsrMrkdBadChan), ...
                       sprintf('\nBad Channel(s):'), sprintf('Channel # %d\n', badChanPntr));
      outputLog_pF = strvcat(outputLog_pF, usrMsg);

   case 2
      badChanPntr = find(std(eegData, 0, 2) <= badChanTol);
      goodChanPntr = 1:numChan; goodChanPntr(badChanPntr) = []; numAutoEtdBadChan = length(badChanPntr);
      usrMsg = sprintf('\nBad Channel Tolerance: %6.4f | Number Of Auto-Eliminated Bad Channels: %d', badChanTol, numAutoEtdBad
      if numAutoEtdBadChan
         usrMsg = strvcat(usrMsg, sprintf('\nBad Channel(s):'), sprintf('Channel # %d\n', badChanPntr));
      else
         usrMsg = sprintf('\n');
      end
      outputLog_pF = strvcat(outputLog_pF, usrMsg);

   case 3
      numUsrMrkdBadChan = length(badChanPntr);
      goodChanPntr = 1:numChan; goodChanPntr(badChanPntr) = [];
      goodChanPntr(find(std(eegData(goodChanPntr, :), 0, 2) <= badChanTol)) = [];
      badChanPntr = 1:numChan; badChanPntr(goodChanPntr) = []; numAutoEtdBadChan = length(badChanPntr) - numUsrMrkdBadCh
```

```
      usrMsg = strvcat(sprintf('\nNumber Of User-Specified Bad Channels: %d' , numUsrMrkdBadChan), ...
         sprintf('\nBad Channel Tolerance: %6.4f | Number Of Auto-Eliminated Bad Channels: %d', badChanTol, numAutoEtdBadChan),
         sprintf('\nBad Channel(s):'), sprintf('Channel # %d\n', badChanPntr));
      outputLog_pF = strvcat(outputLog_pF, usrMsg);

   case 4
      goodChanPntr = 1:numChan; badChanPntr = [];
      usrMsg = sprintf('\nNumber Of Bad Channels Removed: 0\n');
      outputLog_pF = strvcat(outputLog_pF, usrMsg);

end

numGoodChan = length(goodChanPntr);

% Verify that VEOG channels are not amongst bad channels. ----------------------------------------------------------------------

for i =  1 : numVEOG
   badVEOG(i) = any(badChanPntr == VEOG(i));
end

switch any(badVEOG)

   case 0
      usrMsg = sprintf('Good VEOG Channels:');
      outputLog_pF = strvcat(outputLog_pF, usrMsg, VEOGstr);

   case 1
      usrMsg1 = sprintf('\nPreFilter Output Log:\n'); usrMsg2 = sprintf('Bad VEOG Channel(s):');
      disp(strvcat(usrMsg1, outputLog_pF, usrMsg2, VEOGstr(find(badVEOG == 1), :)));
      errMsg = sprintf('\n!!! Program Terminated: Bad VEOG Channels !!!\n'); error(errMsg);

end

usrMsg = sprintf('\n-------------------------------------------------'); outputLog_pF = strvcat(outputLog_pF, usrMsg);

% Compute the size of each segment. ---------------------------------------------------------------------------------------------

if segment

   segSize = floor(numSamples / numSegment);
   lastSegSize = segSize + mod(numSamples, numSegment);
   segStart = [1, segSize * (1 : (numSegment - 1)) + 1];
   segStop = [segSize * (1 : (numSegment - 1)), numSamples];
   segSize = [segSize * ones(1, numSegment - 1), lastSegSize];

else

   segStart = 1;  segStop = numSamples;  segSize = numSamples;

end

% Determine if any of the VEOG channels have saturated observations. ------------------------------------------------------------

for i = 1 : numSegment

   goodObsPntr{i} = find((min(eegData(VEOG, segStart(i):segStop(i))) >= minAmp) & ...
                         (max(eegData(VEOG, segStart(i):segStop(i))) <= maxAmp));

   badObsPntr{i} = 1:segSize(i); badObsPntr{i}(goodObsPntr{i}) = [];

   numGoodObs(i) = length(goodObsPntr{i}); numBadObs(i) = length(badObsPntr{i});

   usrMsg = sprintf('\n%s | Number Of Saturated Observations: %d', messageStem{i}, numBadObs(i));
   outputLog_pF = strvcat(outputLog_pF, usrMsg);

end

usrMsg = sprintf('\n-------------------------------------------------'); outputLog_pF = strvcat(outputLog_pF, usrMsg);
```

BlinkAway.m

```
function[numBlinkPntrs] = blinkAway(fileName, fileStem, fid, precision, ...
   goodChanPntr, badChanPntr, numGoodChan, goodObsPntr, badObsPntr, ...
   satObsProtocol, vctrFltrTol, segSize, segStart, segStop, ...
   A, W, VEOG, numVEOG, numVctrFltrTol, vftFileStem);

% Inputs:
%
%       eegData - m by n matrix consisting of n time samples from m detectors.
%                 Read from binary file(s) <fileName fileStem>.
%
%       header - structure array of EEG data information. Passed globally.
%
%       A - mixing matrix (numGoodChan by numGoodChan).
%
%       W - separating matrix (numGoodChan by numGoodChan).
%
%       VEOG - (vector) of ocular channel markers.
%
%       fid - (scalar / vector) contains the file id numbers of the
%             binary scratch files <fileName_tol#>.
%
%       precision - (string) is the precision of the EEG data.
%
%       goodChanPntr - (vector / cell array) of good channel indices / segment.
%
%       badChanPntr - (vector / cell array) of bad channel indices / segment.
%
%       goodObsPntr - (vector / cell array) of non-saturated observation indices / segment.
%
%       badObsPntr - (vector / cell array) of saturated observation indices / segment.
%
%       satObsProtocol - (scalar) determines if saturated observations are used to
%                        construct the ICA activations and are kept in the filtered
%                        EEG data, or are not used to construct the ICA activations
%                        and are either set = 0 in the filtered EEG data or eliminated.
%
%       vctrFltrTol - (scalar / vector) of tolerances for determining blink (vector filter) matches.
%
%       numVctrFltrTol - (scalar) number of tolerances stored in vctrFltrTol.
%
%       segSize - (scalar / vector) contains the number of good observations per segment.
%
%       segStart - (scalar / vector) of indices to the first observation of each segment.
%
%       segStop - (scalar / vector) of indices to the last observation of each segment.
%
%-------------------------------------------------------------------------------------------------------------------
%
% Data Files: vctrFltr.txt (ASCII file) is the 'blinks' template that is read into the program.
%             It should reside in the same directory as this routine.  The template is a
%             m by 1 array of space seperated data for describing the blink (or whatever) events.
%
%-------------------------------------------------------------------------------------------------------------------
%
% Outputs:
%
%------------------------------------------ Written to the file <fileName_tol#>. -----------------------------------
%
%       eegData - eegData array, new & improved, with fewer blinks.
%
%       pureBlinks - (array) contains the cumulative projections of blink activations
%                    onto the scalp.
%-------------------------------------------------------------------------------------------------------------------
%
%       outputlog_bA - character array of program information.  Passed globally.
%
%       numBlinkPntrs - (scalar / vector) number of blink activations per segment.
```

```
global header outputLog_bA;
numChan = header.numChan; outputLog_bA = []; epsilon = 0.0001;

load vctrFltr.txt -ascii;
vctrFltr = reshape(vctrFltr, numChan, 1);


% 1) Left-concatenate the blinks template (vctrFltr) to mixing matrix A.
% 2) Determine the amount of correlation between vctrFltr and the columns of A via the MATLAB routine corrcoef. ---


corrs = corrcoef([vctrFltr(goodChanPntr) A]);
corrVctr = abs(corrs(1, 2 : size(corrs, 2)));

% Compute EOG pointers. ----------------------------------------------------------------------------------------------

for i = 1 : numVEOG
   VEOG(i) = VEOG(i) - sum(badChanPntr < VEOG(i));
end

heogPntr = (sign(A(VEOG(3),:)) == -1 * sign(A(VEOG(4),:)));
numHeogPntrs = sum(heogPntr);

veogPntr = (sign(A(VEOG(1),:)) == sign(A(VEOG(2),:))) & ...
         (sign(A(VEOG(3),:)) == sign(A(VEOG(4),:))) & ...
         (sign(A(VEOG(1),:)) == -1 * sign(A(VEOG(3),:)));
numVeogPntrs = sum(veogPntr);

usrMsg1 = sprintf('Criteria # 1 (HEOG Polarity Inversion) | %d ICA Activation(s):\n', numHeogPntrs);
usrMsg2 = sprintf('%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\n', [find(heogPntr); corrVctr(find(heogPntr))
outputLog_bA = strvcat(outputLog_bA, usrMsg1, usrMsg2);

usrMsg = sprintf('\n---------------------------------------------------------------\n');
outputLog_bA = strvcat(outputLog_bA, usrMsg);

usrMsg1 = sprintf('Criteria # 2 (VEOG Polarity Inversion) | %d ICA Activation(s):\n', numVeogPntrs);
usrMsg2 = sprintf('%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\n', [find(veogPntr); corrVctr(find(veogPntr))
outputLog_bA = strvcat(outputLog_bA, usrMsg1, usrMsg2);

usrMsg = sprintf('\n---------------------------------------------------------------\n');
outputLog_bA = strvcat(outputLog_bA, usrMsg);

% Step through tolerances. -------------------------------------------------------------------------------------------

for i = 1 : numVctrFltrTol

   usrMsg = sprintf('\nVector Filter Tolerance: %4.3f', vctrFltrTol(i));
   outputLog_bA = strvcat(outputLog_bA, usrMsg);

% Find pointer (blinkPntr) to ICA weights (columns of A) that correlate to vctrFltr with a correlation coefficient >= vFt.

   blinkTmpltPntr = corrVctr >= (vctrFltrTol(i) - epsilon);
   numBlinkTmpltPntrs = sum(blinkTmpltPntr);
   blinkPntr = find(veogPntr & blinkTmpltPntr);
   numBlinkPntrs(i) = length(blinkPntr);

   usrMsg1 = sprintf('\nCriteria # 3 (Blink Template Correlation) | %d ICA Activation(s):\n', numBlinkTmpltPntrs);
   usrMsg2 = sprintf('%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\n', …
      [find(blinkTmpltPntr); corrVctr(find(blinkTmpltPntr))]);

   outputLog_bA = strvcat(outputLog_bA, usrMsg1, usrMsg2);

   if ~isempty(blinkPntr)  % Blink (vector filter) template correspondence.

      usrMsg1 = sprintf('\nCriteria # 2 AND Criteria # 3 (Blink Activity) | %d ICA Activation(s):\n', numBlinkPntrs(i));
      usrMsg2 = sprintf('%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\t%03d (%4.3f)\n', [blinkPntr; corrVctr(blinkPntr)]);
      outputLog_bA = strvcat(outputLog_bA, usrMsg1, usrMsg2);

% Read in EEG data / EEG data segment.
```

```
        eegData = zeros(numChan, segSize);
        pureBlinks = zeros(numChan, segSize);

        fidTmp = fopen([fileName fileStem]);

        if (fidTmp == -1)
           errMsg = sprintf('\n!!! File Access Error: %s !!!\n', [fileName fileStem]); error(errMsg);
        end

        eegData(goodChanPntr, :) = fread(fidTmp, [numGoodChan, segSize], precision);

        fclose(fidTmp);

% Generate ICA activations, "pure" blinks, and extract blinks from EEG data.

        icaActivations = W(blinkPntr, :) * eegData(goodChanPntr, goodObsPntr);
        pureBlinks(goodChanPntr, goodObsPntr) = A(:, blinkPntr) * icaActivations;
        eegData(goodChanPntr, goodObsPntr) = eegData(goodChanPntr, goodObsPntr) - pureBlinks(goodChanPntr, goodObsPntr);

        usrMsg = sprintf('\n*** %d ICA Activation(s) Subtracted From The EEG Data ***\n', numBlinkPntrs(i));
        outputLog_bA = strvcat(outputLog_bA, usrMsg);

% Save arrays eegData & pureBlinks to a binary file.

        switch satObsProtocol
           case 1  % Keep saturated observations.
              fwrite(fid(i), [eegData pureBlinks], precision);
           case 2  % Zero - out saturated observations.
              eegData(goodChanPntr, badObsPntr) = 0;
              fwrite(fid(i), [eegData pureBlinks], precision);
           case 3  % Delete saturated observations.
              fwrite(fid(i), [eegData(:, goodObsPntr) pureBlinks(:, goodObsPntr)], precision);
        end

     else  % No blink (vector filter) template correspondence.

        usrMsg = sprintf('\nNo ICA activations qualified as blink activity: No blinks removed.\n');
        outputLog_bA = strvcat(outputLog_bA, usrMsg);

     end

     usrMsg = sprintf('\n-----------------------------------------------------------------\n');
     outputLog_bA = strvcat(outputLog_bA, usrMsg);

end

% Housekeeping. ---------------------------------------------------------------------------------------------------

numBlinkPntrs = reshape(numBlinkPntrs, i, 1);
clear eegData pureblinks icaActivations;
```

WriteRaw.m

```
function writeRaw(fileName, header, eegData, eventData, ...
   addEvents, numChan, numSamples, samplingRate, versionNumber);

% writeRaw writes filtered EEG data, stored in the array eegData, to the file
% <fileName.raw>, along with its corresponding header and event information.
%
%
% <fileName.raw> will be an epoch-marked raw format file.
%
%
% Epoch-marked raw format: Unsegmented, simple binary format file (version #
%                                  2, 4 or 6) with event codes <epoc> and <tim0>.
%
%
```

```
% Input Arguments: fileName - Name of the EGI epoch-marked raw format
%                           file originally read by readRaw, without the
%                           '.raw' extension.  It must be a MATLAB string.
%                header - MATLAB structure containing the header info.
%
%                eegData - m1 by n array containing the continuous EEG data,
%                             ideally after ICA blink extraction / filtering.
%                             m1 = # of channels, n = # of time samples.
%
%                eventData - m2 by n array containing the corresponding event info.
%                              m2 = # of event types, n = # of time samples.
%
%                addEvents - Logical 1 if user requests default events <epoc> and <tim0>.
%
%                numChan - # of data channels (detectors).
%
%                numSamples - # of time samples.
%
%                samplingRate - The rate of sampling.
%
%                versionNumber - 2: Integer, 4: Single Precion Real,
%                                           6: Double Precision Real
%
% Output Arguments: outputLog_wR - Character array of relevant steps taken
%                           during program execution.

global outputLog_wR;

% Create the EGI epoch-marked raw format data file: <fileName.raw>. ----------------------------------------------------------

fid = fopen([fileName '.raw'],'w','b');

if (fid == -1)
   errMsg = sprintf('\n!!! File Creation Error: %s !!!\n', [fileName '.raw']); error(errMsg);
end

usrMsg = sprintf('Output File: %s', [fileName '.raw']); outputLog_wR = usrMsg;

if (nargin == 9)  % Header file NOT supplied.  Instantiate with user-supplied & default values.

   header.versionNumber = versionNumber;
   header.recordingTimeYear = 0;
   header.recordingTimeMonth = 0;
   header.recordingTimeDay = 0;
   header.recordingTimeHour = 0;
   header.recordingTimeMinute = 0;
   header.recordingTimeSecond = 0;
   header.recordingTimeMillisec = 0;
   header.samplingRate = samplingRate;
   header.numChan = numChan;
   header.boardGain = 0;
   header.numConvBits = 0;
   header.ampRange = 0;
   header.numSamples = numSamples;

   if addEvents
      header.numEvents = 2;
      header.eventCodes(1) = {'epoc'};
      header.eventCodes(2) = {'tim0'};
      eventData = zeros(2,numSamples);
      usrMsg = sprintf('Generated default event codes <epoc> and <tim0> (= 0 for all time samples).');
      outputLog_wR = strvcat(outputLog_wR, usrMsg);
   else
      header.numEvents = 0;
   end

end

% Write the header information to the file. ------------------------------------------------------------------------------
```

```
fwrite(fid, header.versionNumber, 'integer*4');
fwrite(fid, header.recordingTimeYear, 'integer*2');
fwrite(fid, header.recordingTimeMonth, 'integer*2');
fwrite(fid, header.recordingTimeDay, 'integer*2');
fwrite(fid, header.recordingTimeHour, 'integer*2');
fwrite(fid, header.recordingTimeMinute, 'integer*2');
fwrite(fid, header.recordingTimeSecond, 'integer*2');
fwrite(fid, header.recordingTimeMillisec, 'integer*4');
fwrite(fid, header.samplingRate, 'integer*2');
fwrite(fid, header.numChan, 'integer*2');
fwrite(fid, header.boardGain, 'integer*2');
fwrite(fid, header.numConvBits, 'integer*2');
fwrite(fid, header.ampRange, 'integer*2');
fwrite(fid, header.numSamples, 'integer*4');
fwrite(fid, header.numEvents, 'integer*2');

if (header.numEvents ~= 0)  % File contains event info.

   for  i = 1:header.numEvents
      fwrite(fid, header.eventCodes{i}, 'char');
   end

end

% Finished writing the header. ----------------------------------------------------------------------------------------

switch header.versionNumber

  case 2
     precision = 'integer*2';  % Integer
  case 4
     precision = 'real*4';  % Single Precision Real
  case 6
     precision = 'real*8';  % Double Precision Real

end

% Write SSR's and corresponding event records to the file. ------------------------------------------------------------

if (header.numEvents ~= 0)  % File contains event info.

   for j = 1:header.numSamples
      fwrite(fid, eegData(:,j), precision);
      fwrite(fid, eventData(:,j), precision);
   end

else  % File does not contain event info.

   fwrite(fid, eegData, precision);

end

% Close the file. -----------------------------------------------------------------------------------------------------

fclose(fid);
```

WriteEvents.m

```
function writeEvents(fileName, header, eventData);

global outputLog_wE;

fid = fopen([fileName '.txt'], 'w');

if (fid == -1)
   errMsg = sprintf('\n!!! File Creation Error: %s !!!\n', [fileName '.txt']); error(errMsg);
end
```

```
deltaT = (1000 / header.samplingRate);  % Time in milliseconds, sampling rate in Hz.

for j = 1:header.numSamples
    eventPntr = find(eventData(:, j) == 1);
    for i = 1 : length(eventPntr)
        fprintf(fid, '%s\t%s\r', num2str((j-1)*deltaT), char(header.eventCodes{eventPntr(i)}));
    end
end

usrMsg  = sprintf('\nNetStation formatted event data written to file: %s', [fileName '.txt']);
outputLog_wE = usrMsg;

fclose(fid);
```

Bibliography

- [1] ICAToolBox download site. http://people.ku.edu/~jdien/. March 12, 2004.

- [2] Dien, J. (1998). Issues in th eapplicatio of the average referance. Review, critiques, and recommmendations. Behave. Res. Methods, instruments, aand Computers, 30(1), 34 - 43.

- [3] FastICA download site. http://www.cis.hut.fi/projects/ica/fastica/. March 12, 2004.

- [4] EEGLab download site. http://www.sccn.ucsd.edu/eeglab/downloadtoolbox.html. March12, 2004.