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

EEG Blink Removal Using APECS

Introduction

APECS is a collection of MATLAB m-files, based upon Dr. Joseph Dien's ICA Toolbox [1, 2], designed to remove eyeblinks from EEG data using independent component analysis (ICA). In ICA, EEG data (x) is represented by a 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
- x = m by n matrix of EEG data, where each row corresponds to the output of a single detector and each column represents 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 independent component[1]. 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

Correlation to ICA Toolbox

Although based upon Dr. Dien's ICA Toolbox, APECS adds several additional features which you may find of interest:

- The ability to call FastICA, Infomax or Par-FastICA, a parallel implementation of FastICA written in C, as the ICA decomposition protocol;
- The ability to locate and extract blink activity using blink template correlation / VEOG polarity constraints;
- The ability to loop through multiple blink template correlations: For example, extract all blink activity correlating to the blink template with correlation coefficient >= 0.85, >= 0.90 and >= 0.95, etc...

Component M-Files

APECS consists of the following MATLAB script files (.m), listed below in the order in which they are executed:

1.	Driver	(Controlling script that links all routines)
2.	UserInfo	(Queries you for blink removal parameters)
3.	AutoDriver	(Reads blink removal parameters from a file)
4.	ReadRawHeader	(Reads EGI raw format EEG data file header)
5.	PreFilter	(Bad channel / saturated observation removal)
6.	ReadRawData	(Reads EGI raw format EEG data)
7.	RewindFstFrwrd	(Moves file pointer to appropriate data segment)
8.	NicICAForms	(Parameter file for Parallel FastICA)
9.	BlinkAway	(Removes blinks via blink template / VEOG polarity)
10.	WriteRaw	(Writes various EGI raw format data files to disk)
11.	WriteEvents	(Writes event markers, if any, to a text file)

Note: AutoDriver is used in batch processing mode only.

Program Execution and Flow

To start the program, first 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 calls UserInfo, which queries the user regarding EEG data file name, bad channel(s) and saturated obervation(s) removal, data segmentation, ICA decomposition method, and how blink activity is to be determined. UserInfo also calls ReadRawHeader, which reads in the EEG data header from an EGI raw format file. Upon completion, UserInfo returns control to Driver.

Driver next calls PreFilter, which, after reading in the EEG data / data segments, computes row pointers to the bad channels, column pointers to the saturated observations and, if you chose to segment the data, column pointers to the segment breaks. Upon completion, PreFilter returns control to Driver.

At this point, Driver opens several binary scratch files needed by the program, which are 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 passed to the user-specified ICA algorithm, minus any bad channels or saturated observations.

At present, APECS can use one of three algorithms to ICA decompose the data: FastICA [3], with a hyperbolic tangent contrast function, Infomax [4], via either MATLAB or its faster C implementation[3] and Par-FastICA. 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 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. A normalized covariance may also be computed between each column of matrix A and the blink template[4]. Then, in accordance with your specifications, all of those columns that meet the VEOG polarity requirement, correlate sufficiently to the blink template, or both, 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 you now know which column vectors of matrix A correspond to blink activity, you 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, APECS writes the blink-free and pure-blink data arrays to disk.

Driver now reads from disk the 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.

If the user specified more than one blink template correlation, APECS writes one blink-free EEG data file and one set of extracted blinks for each of the tolerances.

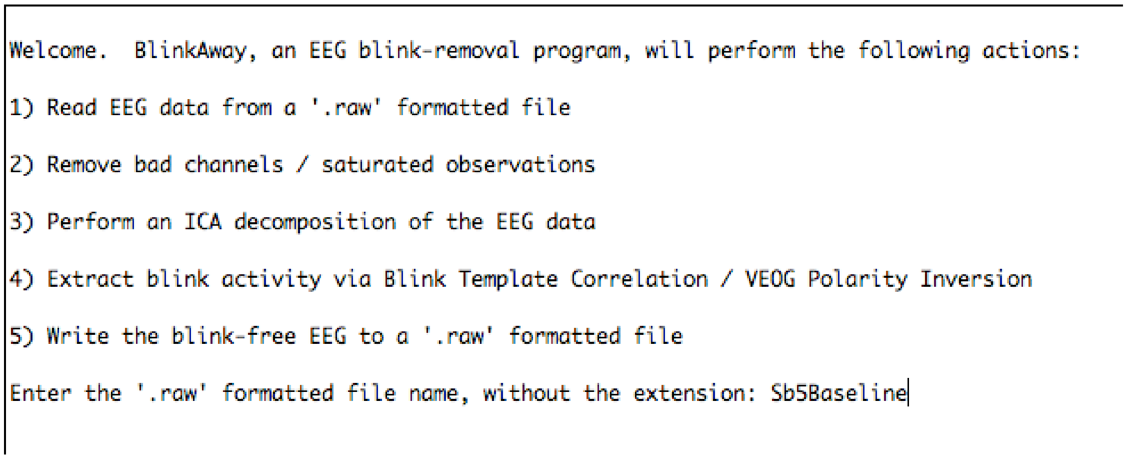
At present, NetStation cannot read in EGI raw format files containing event data and successfully parse the event markers. Therefore, if the original EEG data file contained event markers, they are, at your request, written to a separate ASCII text file, <FileName_Events>, after first removing any event 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 all its temporary scratch / working files, saves all your command line interactions and runtime specifications in .mat files with _Log suffixes, and creates a new folder called "ICA-FileName", into which it moves all these raw format and log data files. It then quits and returns control to the command line.

Command Line Interface

The following sequence of images illustrate the command line interface.

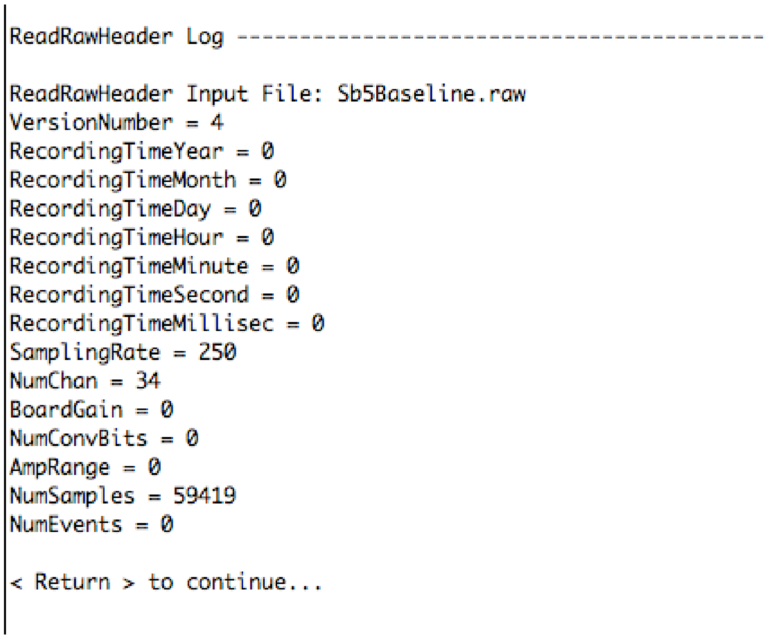
Figure 1: Introduction



```
Welcome.  BlinkAway, an EEG blink-removal program, will perform the following actions:

1) Read EEG data from a '.raw' formatted file

2) Remove bad channels / saturated observations

3) Perform an ICA decomposition of the EEG data

4) Extract blink activity via Blink Template Correlation / VEOG Polarity Inversion

5) Write the blink-free EEG to a '.raw' formatted file

Enter the '.raw' formatted file name, without the extension: Sb5Baseline
```

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

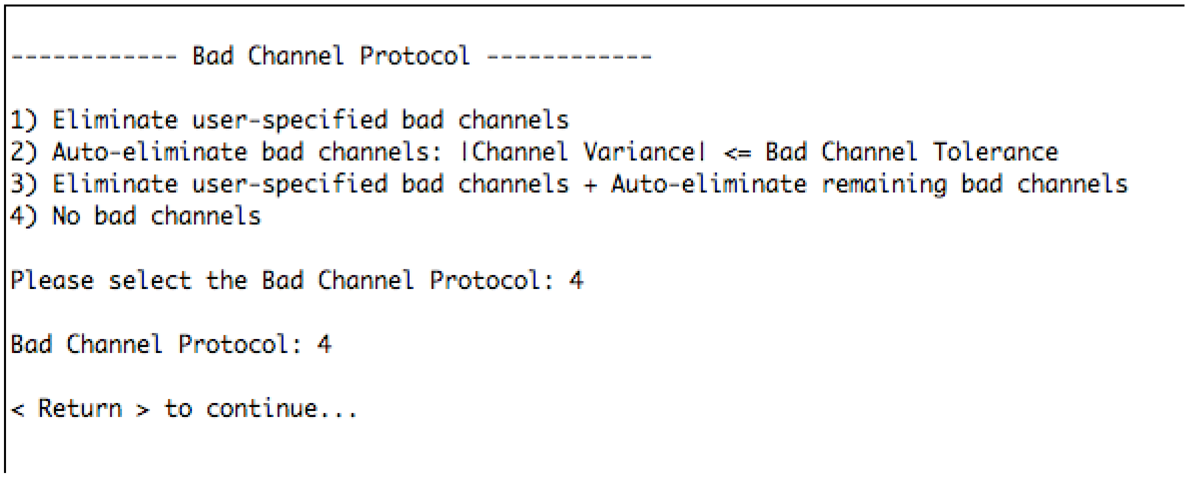
Figure 2: EEG Data File Header Info



```
ReadRawHeader Log ------------------------------------------

ReadRawHeader Input File: Sb5Baseline.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

< Return > to continue...
```

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

Figure 3: Bad Channel Protocol (No Bad Channels)

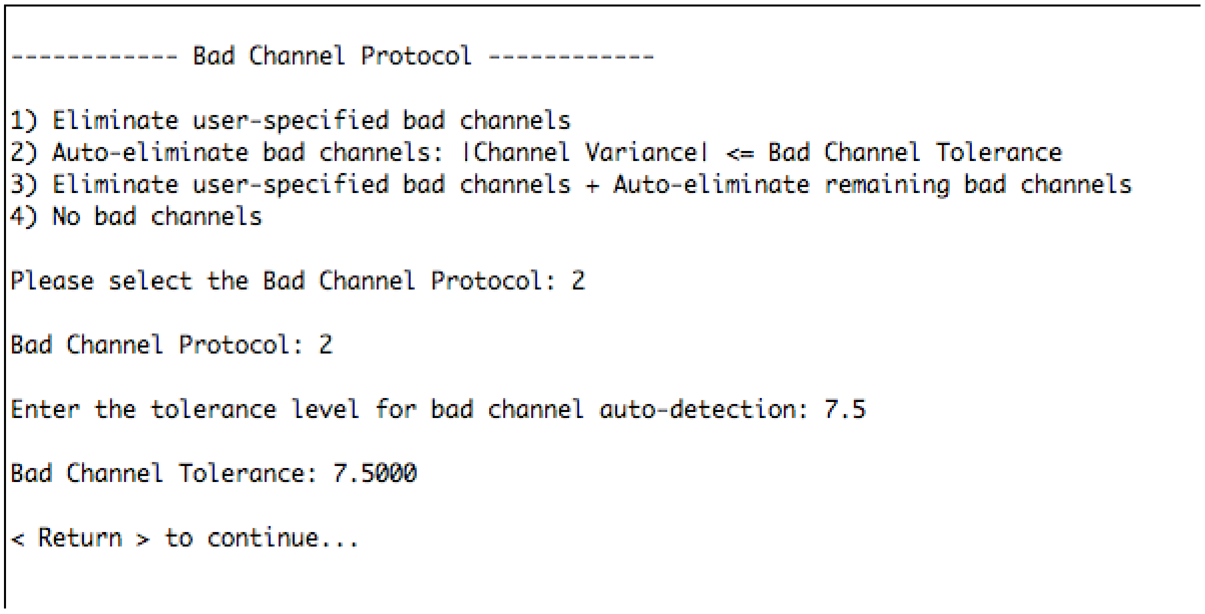


```
------------ Bad Channel Protocol ------------

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...
```

Select the appropriate Bad Channel Protocol, and APECS will confirm your choice.

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

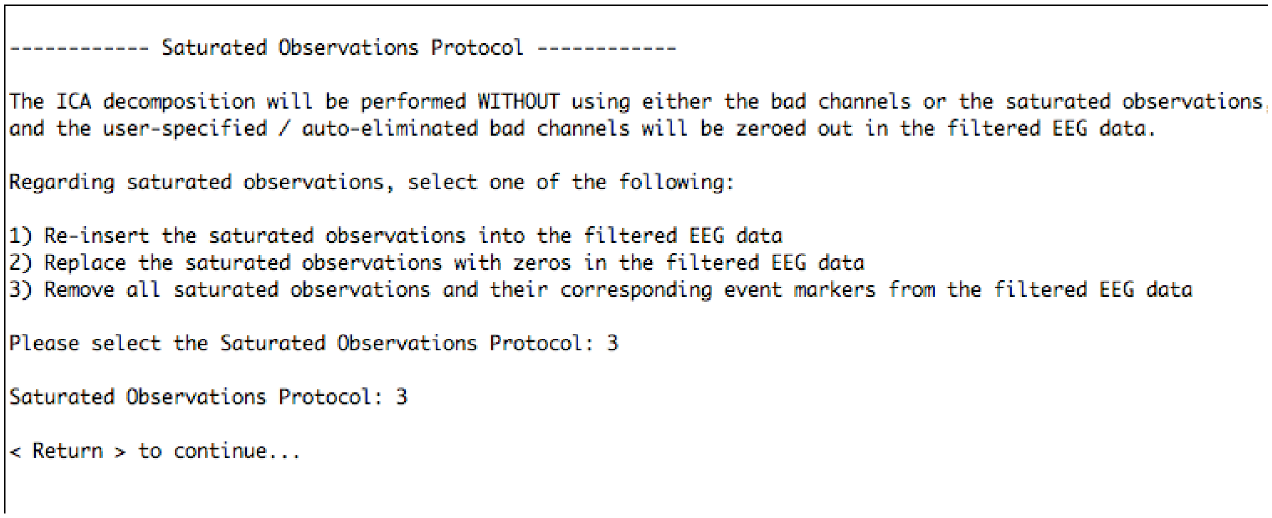
Figure 4: Bad Channel Protocol (Auto-Eliminate Bad Channels)



```
------------ 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...
```

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).

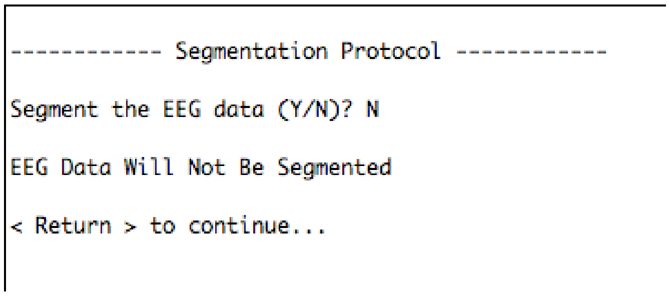
Figure 5: Saturated Observations Protocol



```
------------ Saturated Observations Protocol ------------

The ICA decomposition will be performed WITHOUT using either the bad channels or the saturated observations,
and the user-specified / auto-eliminated bad channels will be zeroed out in the filtered EEG data.

Regarding saturated observations, select one of the following:

1) Re-insert the saturated observations into the filtered EEG data
2) Replace the saturated observations with zeros in the filtered EEG data
3) Remove all saturated observations and their corresponding event markers from the filtered EEG data

Please select the Saturated Observations Protocol: 3

Saturated Observations Protocol: 3

< Return > to continue...
```

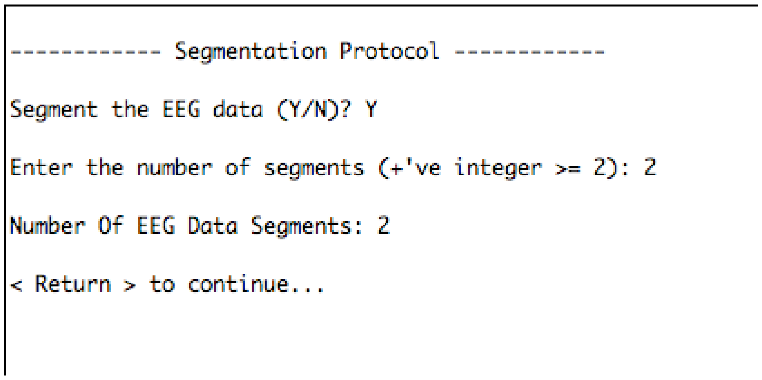
Select the appropriate Saturated Observations Protocol, and APECS will confirm your choice.

Figure 6: Segmentation Protocol



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

Segment the EEG data (Y/N)? N

EEG Data Will Not Be Segmented

< Return > to continue...
```

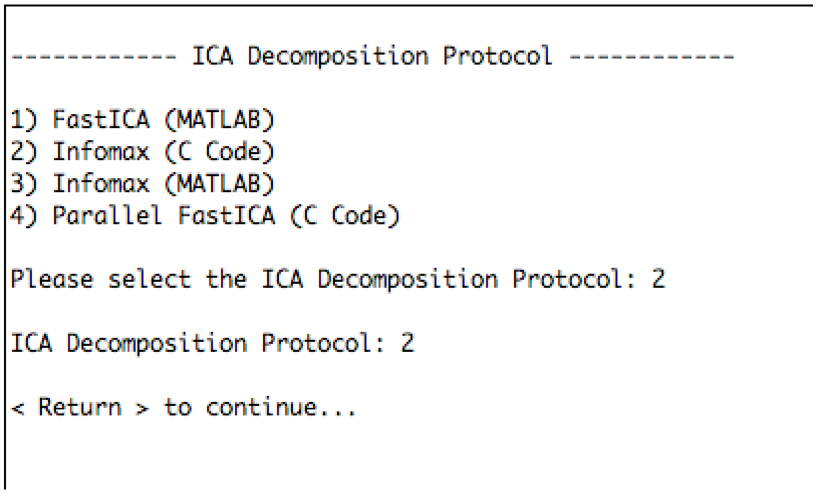
or



```
------------ 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...
```

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

Figure 7: ICA Decomposition Protocol

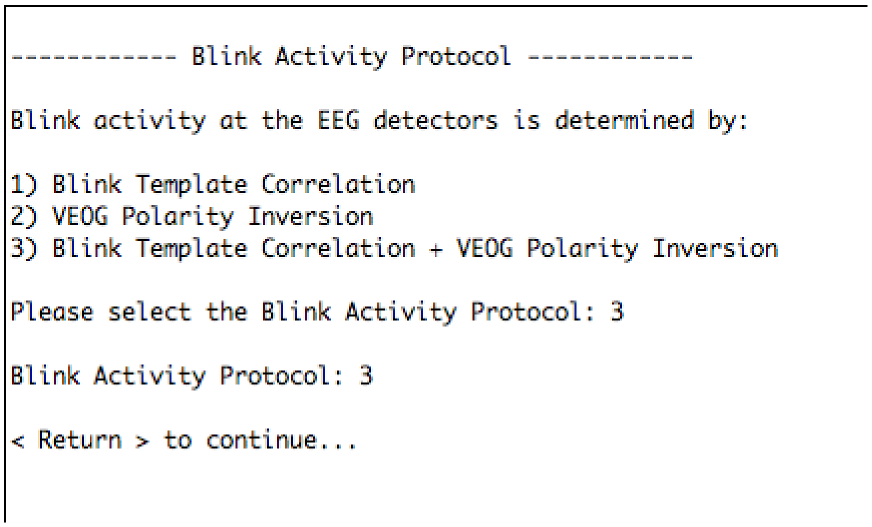


```
------------ ICA Decomposition Protocol ------------

1) FastICA (MATLAB)
2) Infomax (C Code)
3) Infomax (MATLAB)
4) Parallel FastICA (C Code)

Please select the ICA Decomposition Protocol: 2

ICA Decomposition Protocol: 2

< Return > to continue...
```

Select the appropriate ICA Decomposition Protocol, and APECS will confirm your choice.

Note: Parallel FastICA is designed to run simultaneously on multiple processors.

Figure 8: Blink Activity Protocol

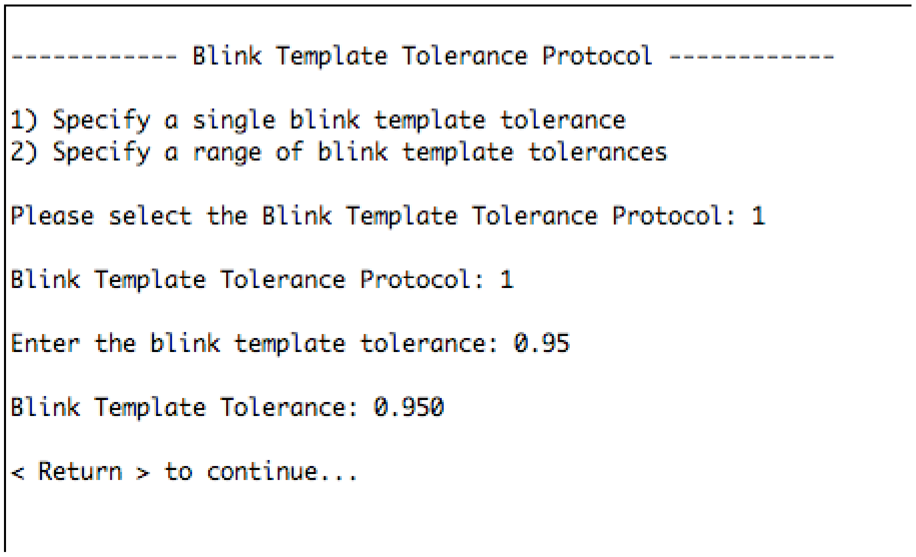


```
------------ Blink Activity Protocol ------------

Blink activity at the EEG detectors is determined by:

1) Blink Template Correlation
2) VEOG Polarity Inversion
3) Blink Template Correlation + VEOG Polarity Inversion

Please select the Blink Activity Protocol: 3

Blink Activity Protocol: 3

< Return > to continue...
```

Select the appropriate Blink Activity Protocol, and APECS will confirm your choice.

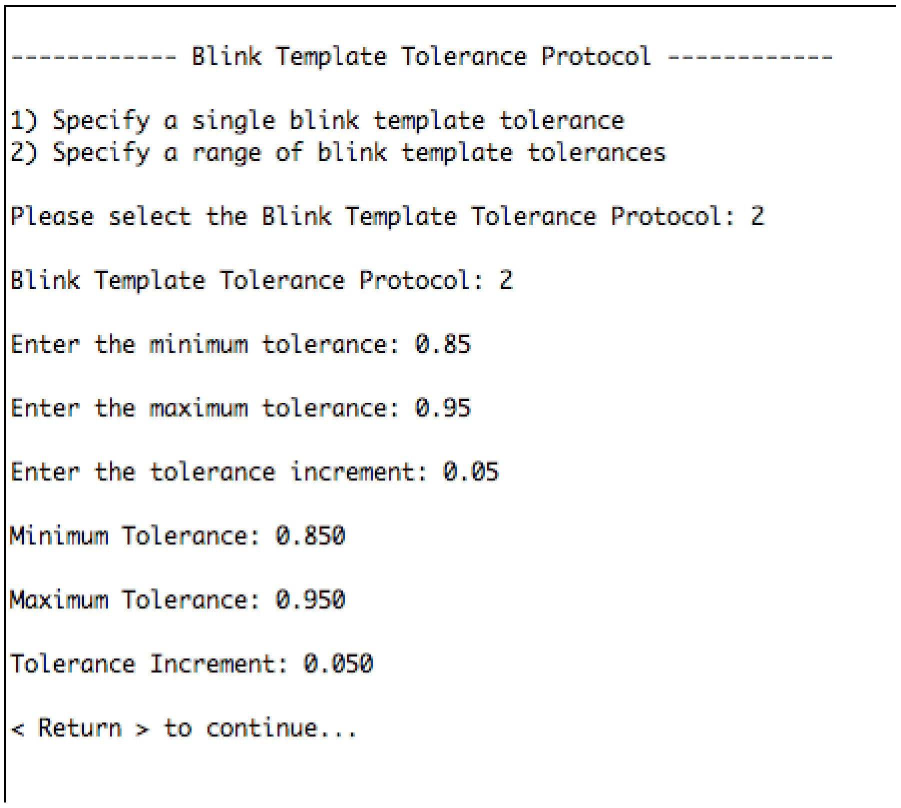
Note: If you choose optiion 1 or 3, you will be prompted to enter the blink template tolerance (Figure 9).

Figure 9: Blink Template Tolerance Protocol



```
------------ Blink Template Tolerance Protocol ------------

1) Specify a single blink template tolerance
2) Specify a range of blink template tolerances

Please select the Blink Template Tolerance Protocol: 1

Blink Template Tolerance Protocol: 1

Enter the blink template tolerance: 0.95

Blink Template Tolerance: 0.950

< Return > to continue...
```

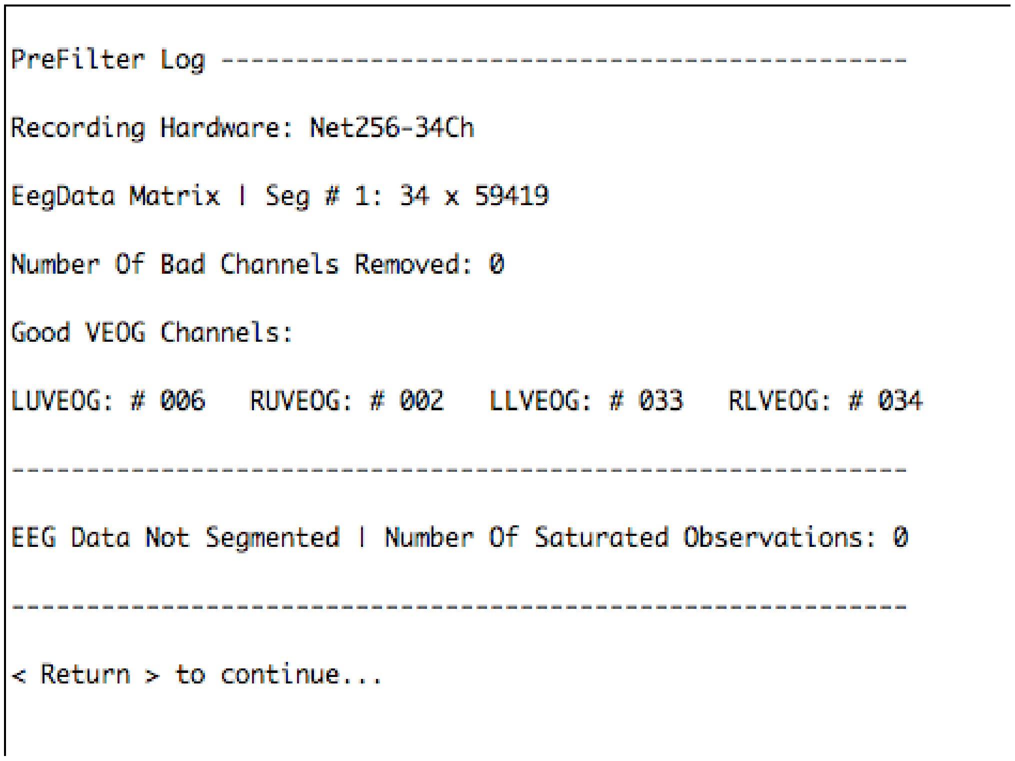
or



```
------------ Blink Template Tolerance Protocol ------------

1) Specify a single blink template tolerance
2) Specify a range of blink template tolerances

Please select the Blink Template Tolerance Protocol: 2

Blink Template Tolerance Protocol: 2

Enter the minimum tolerance: 0.85

Enter the maximum tolerance: 0.95

Enter the tolerance increment: 0.05

Minimum Tolerance: 0.850

Maximum Tolerance: 0.950

Tolerance Increment: 0.050

< Return > to continue...
```

Select the appropriate Blink Template Tolerance Protocol, and APECS will confirm your choice.

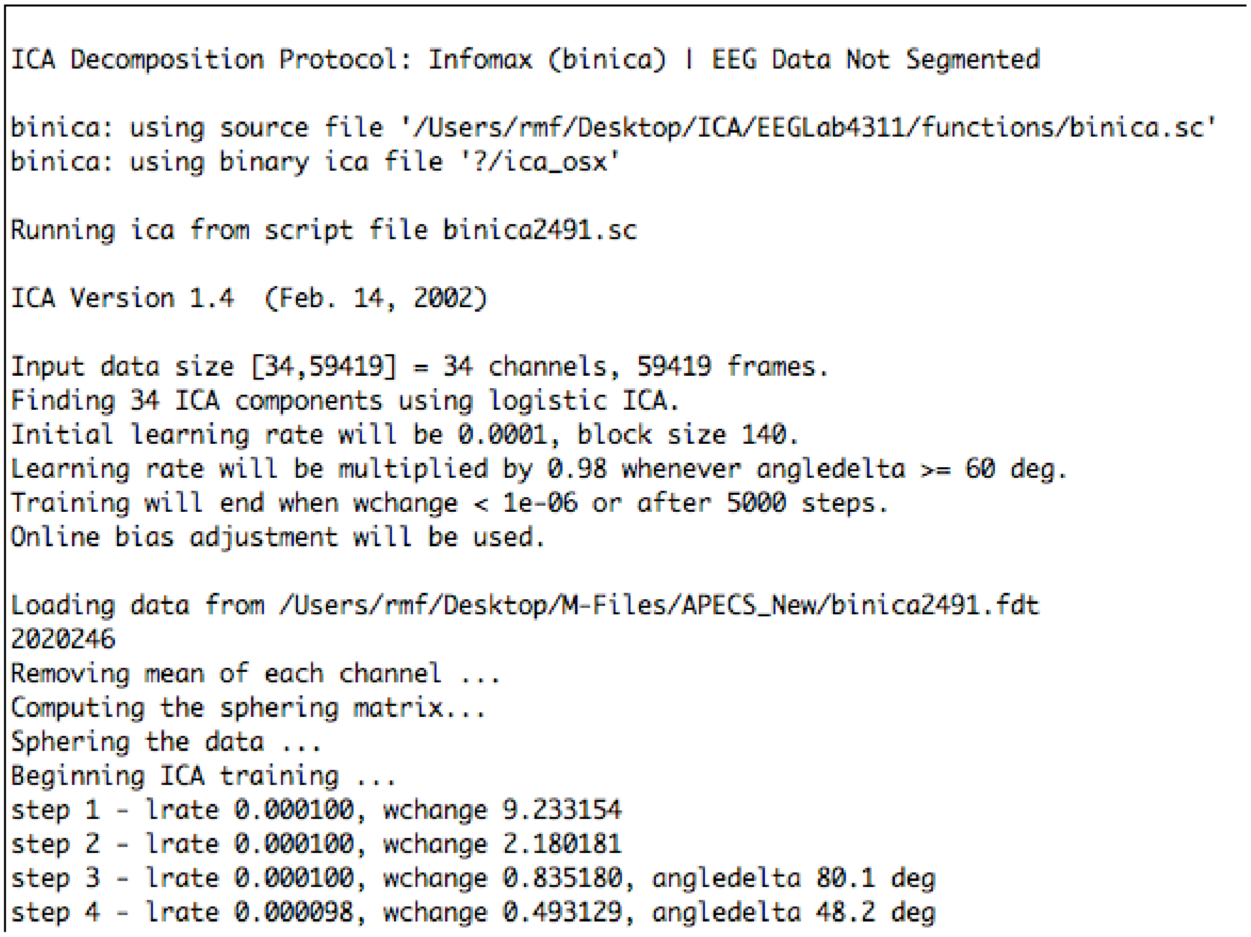
Figure 10: PreFilter Log



```
PreFilter Log ---------------------------------------------------

Recording Hardware: Net256-34Ch

EegData Matrix | Seg # 1: 34 x 59419

Number Of Bad Channels Removed: 0

Good VEOG Channels:

LUVEOG: # 006   RUVEOG: # 002   LLVEOG: # 033   RLVEOG: # 034

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

EEG Data Not Segmented | Number Of Saturated Observations: 0

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

< Return > to continue...
```

The PreFilter log informs you of the recording hardware[5] used to acquire the EEG, the number of bad channels removed, the segmentation protocol and the number of saturated observations.

Figure 11: ICA Algorithm Status

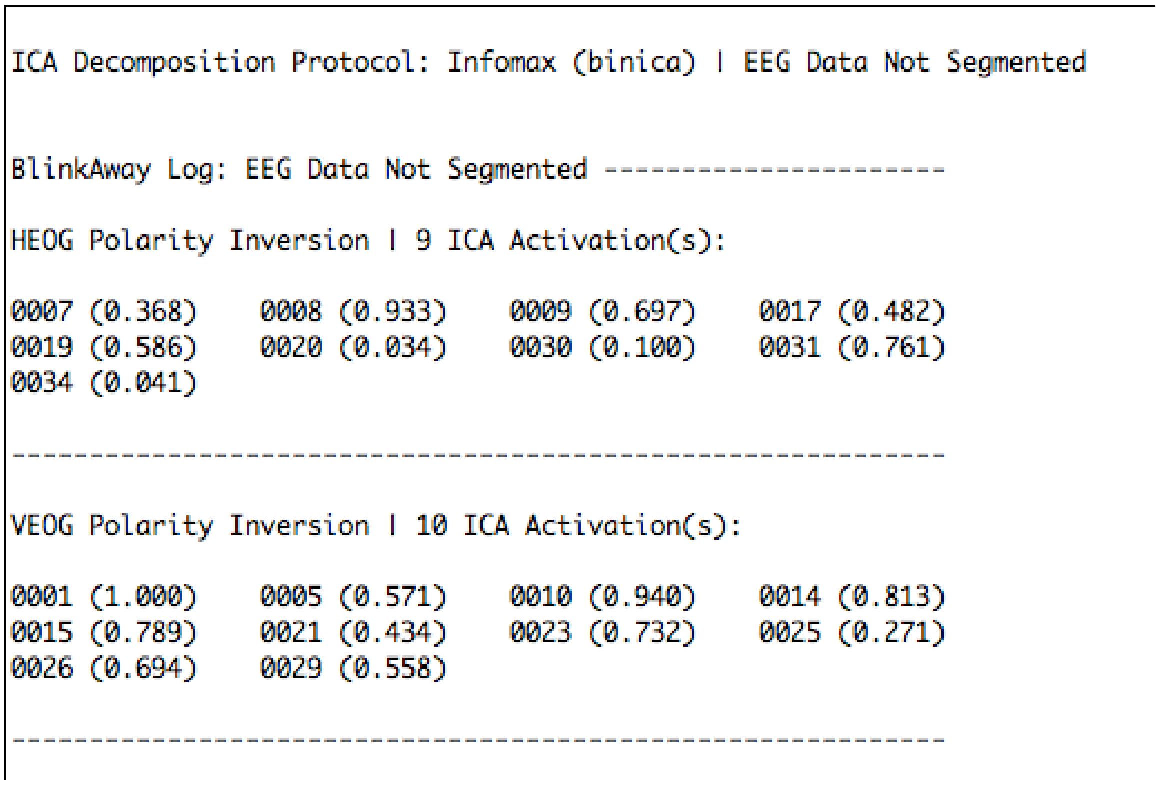


```
ICA Decomposition Protocol: Infomax (binica) | EEG Data Not Segmented

binica: using source file '/Users/rmf/Desktop/ICA/EEGLab4311/functions/binica.sc'
binica: using binary ica file '?/ica_osx'

Running ica from script file binica2491.sc

ICA Version 1.4  (Feb. 14, 2002)

Input data size [34,59419] = 34 channels, 59419 frames.
Finding 34 ICA components using logistic ICA.
Initial learning rate will be 0.0001, block size 140.
Learning rate will be multiplied by 0.98 whenever angledelta >= 60 deg.
Training will end when wchange < 1e-06 or after 5000 steps.
Online bias adjustment will be used.

Loading data from /Users/rmf/Desktop/M-Files/APECS_New/binica2491.fdt
2020246
Removing mean of each channel ...
Computing the sphering matrix...
Sphering the data ...
Beginning ICA training ...
step 1 - lrate 0.000100, wchange 9.233154
step 2 - lrate 0.000100, wchange 2.180181
step 3 - lrate 0.000100, wchange 0.835180, angledelta 80.1 deg
step 4 - lrate 0.000098, wchange 0.493129, angledelta 48.2 deg
```

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

Note: The appearance of this screen and exact nature of the information displayed will depend on the chosen decomposition protocol.

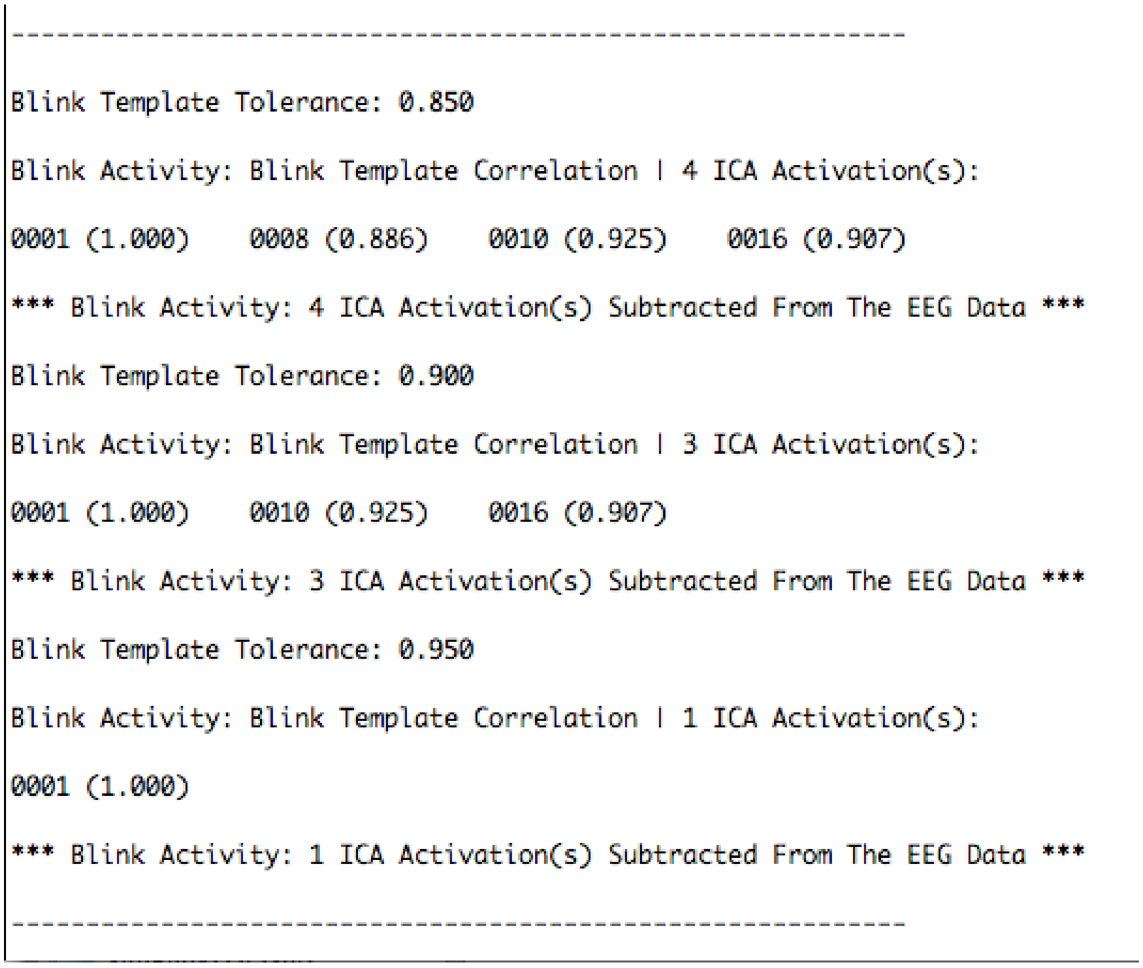
Figure 12: BlinkAway (EOG Polarity Constraints)



```
ICA Decomposition Protocol: Infomax (binica) | EEG Data Not Segmented


BlinkAway Log: EEG Data Not Segmented ----------------------

HEOG Polarity Inversion | 9 ICA Activation(s):

0007 (0.368)    0008 (0.933)    0009 (0.697)    0017 (0.482)
0019 (0.586)    0020 (0.034)    0030 (0.100)    0031 (0.761)
0034 (0.041)

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

VEOG Polarity Inversion | 10 ICA Activation(s):

0001 (1.000)    0005 (0.571)    0010 (0.940)    0014 (0.813)
0015 (0.789)    0021 (0.434)    0023 (0.732)    0025 (0.271)
0026 (0.694)    0029 (0.558)

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

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

Figure 13: BlinkAway (Blink Activation Via Blink Template Correlation)

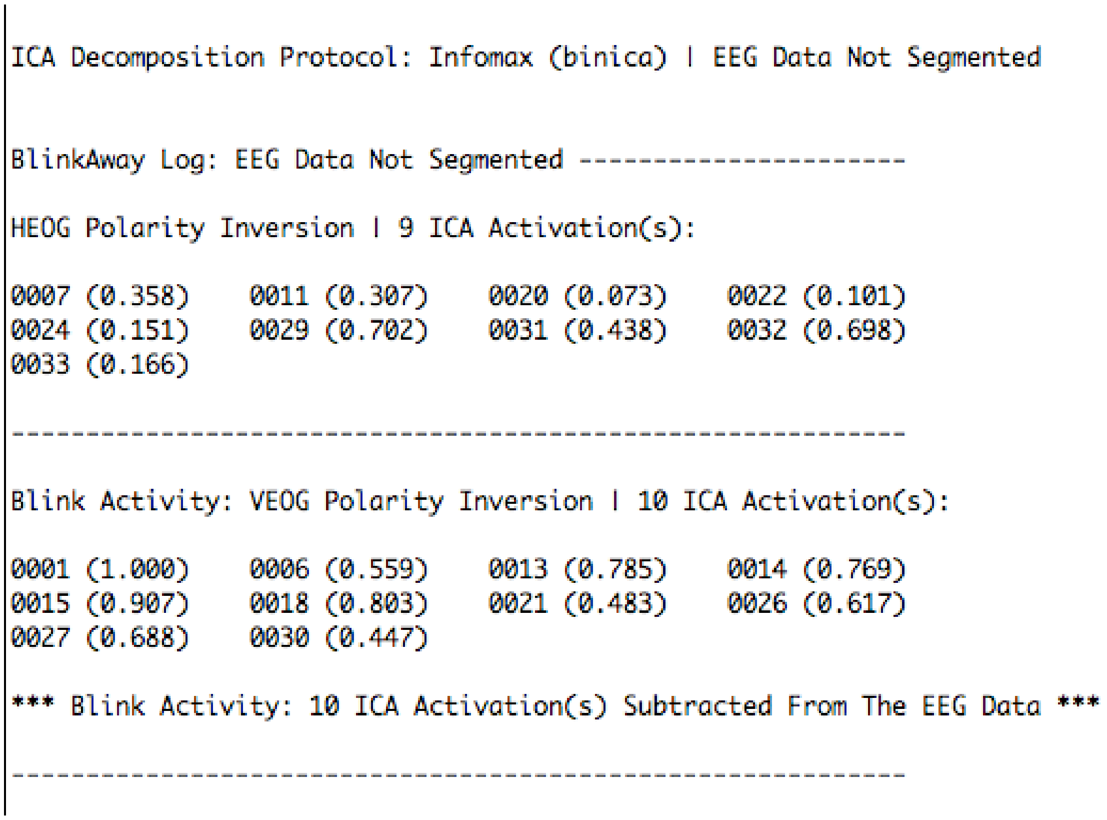


```
------------------------------------------------------------

Blink Template Tolerance: 0.850

Blink Activity: Blink Template Correlation | 4 ICA Activation(s):

0001 (1.000)    0008 (0.886)    0010 (0.925)    0016 (0.907)

*** Blink Activity: 4 ICA Activation(s) Subtracted From The EEG Data ***

Blink Template Tolerance: 0.900

Blink Activity: Blink Template Correlation | 3 ICA Activation(s):

0001 (1.000)    0010 (0.925)    0016 (0.907)

*** Blink Activity: 3 ICA Activation(s) Subtracted From The EEG Data ***

Blink Template Tolerance: 0.950

Blink Activity: Blink Template Correlation | 1 ICA Activation(s):

0001 (1.000)

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

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

The BlinkAway Log screen then informs you of which independent components and their corresponding spatial projectors met the specified Blink Activity Protocol, and confirms that their blink activity has been subtracted from the EEG.

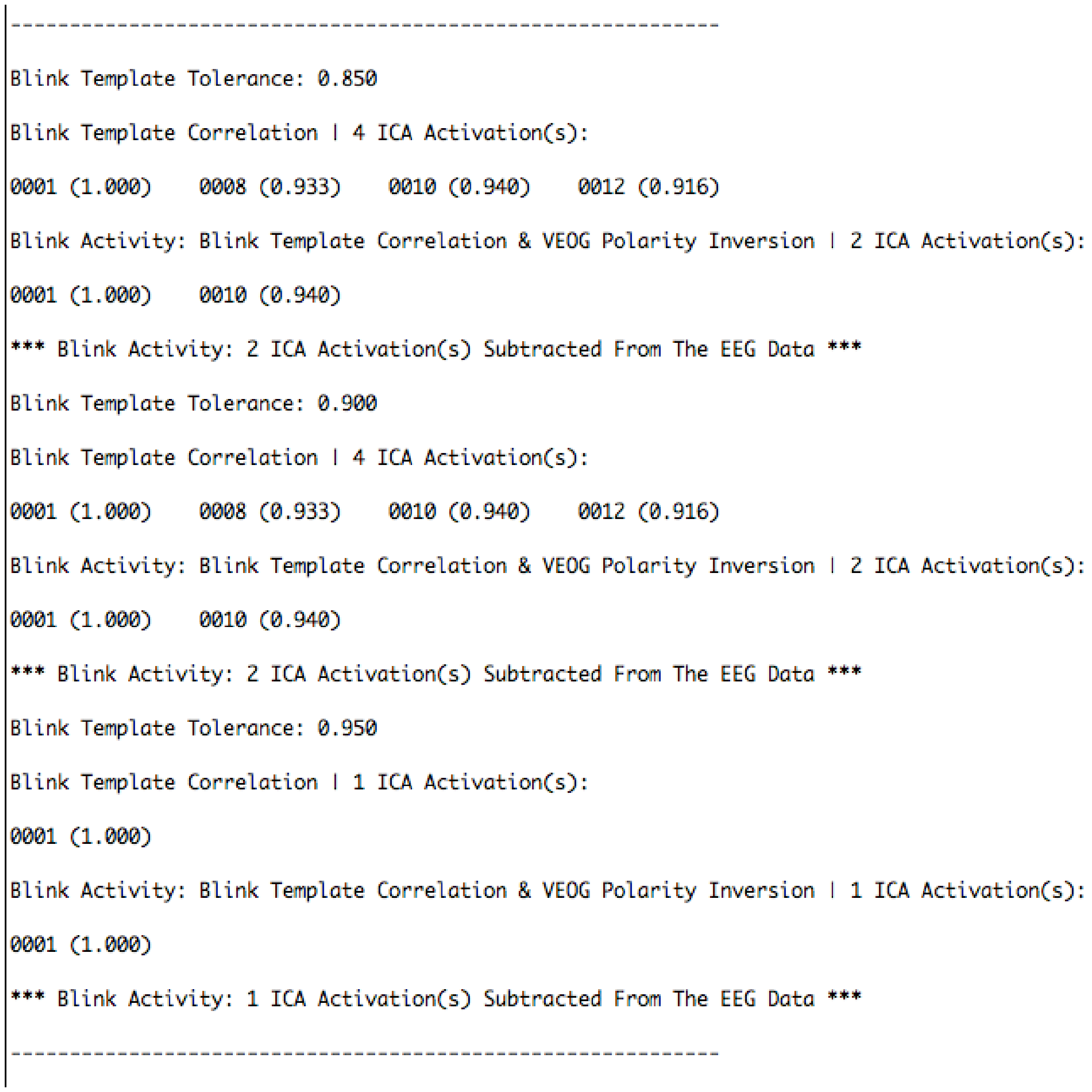
Note: See figures 14 and 15 for results corresponding to other blink activity criteria.

Figure 14: BlinkAway (Blink Activation Via VEOG Polarity Constraints)



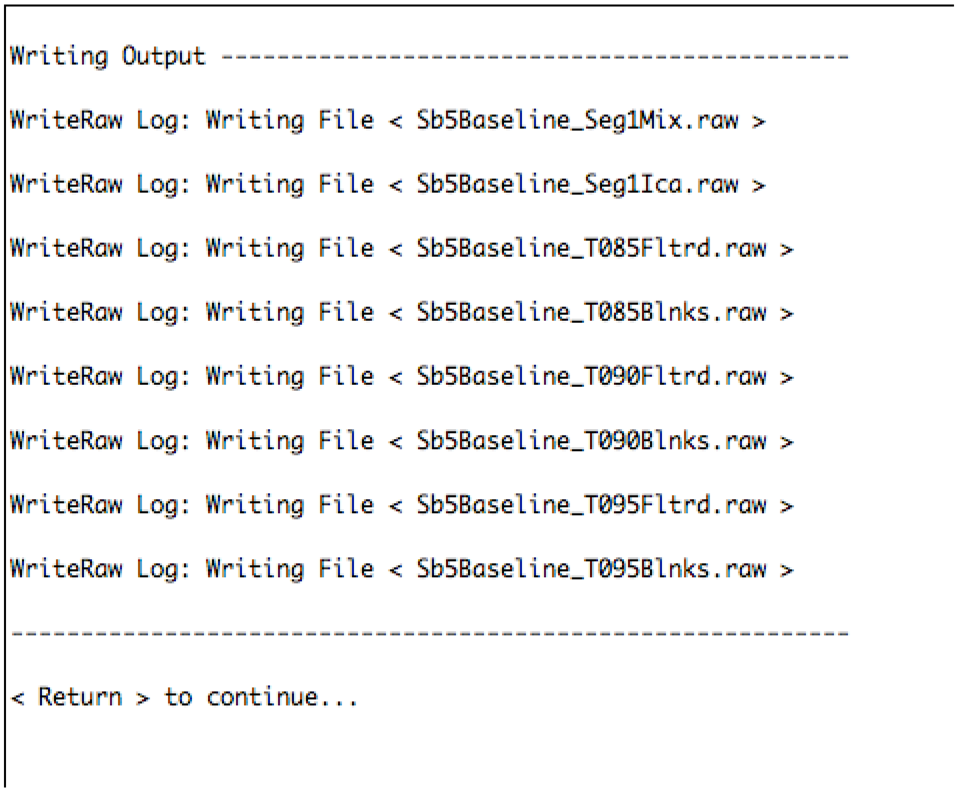
```
ICA Decomposition Protocol: Infomax (binica) | EEG Data Not Segmented


BlinkAway Log: EEG Data Not Segmented ----------------------

HEOG Polarity Inversion | 9 ICA Activation(s):

0007 (0.358)    0011 (0.307)    0020 (0.073)    0022 (0.101)
0024 (0.151)    0029 (0.702)    0031 (0.438)    0032 (0.698)
0033 (0.166)

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

Blink Activity: VEOG Polarity Inversion | 10 ICA Activation(s):

0001 (1.000)    0006 (0.559)    0013 (0.785)    0014 (0.769)
0015 (0.907)    0018 (0.803)    0021 (0.483)    0026 (0.617)
0027 (0.688)    0030 (0.447)

*** Blink Activity: 10 ICA Activation(s) Subtracted From The EEG Data ***

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

Figure 15: BlinkAway (Blink Activation Via Blink Template & VEOG)



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

Blink Template Tolerance: 0.850

Blink Template Correlation | 4 ICA Activation(s):

0001 (1.000)    0008 (0.933)    0010 (0.940)    0012 (0.916)

Blink Activity: Blink Template Correlation & VEOG Polarity Inversion | 2 ICA Activation(s):

0001 (1.000)    0010 (0.940)

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

Blink Template Tolerance: 0.900

Blink Template Correlation | 4 ICA Activation(s):

0001 (1.000)    0008 (0.933)    0010 (0.940)    0012 (0.916)

Blink Activity: Blink Template Correlation & VEOG Polarity Inversion | 2 ICA Activation(s):

0001 (1.000)    0010 (0.940)

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

Blink Template Tolerance: 0.950

Blink Template Correlation | 1 ICA Activation(s):

0001 (1.000)

Blink Activity: Blink Template Correlation & VEOG Polarity Inversion | 1 ICA Activation(s):

0001 (1.000)

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

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

Figure 16: File Output



```
Writing Output ---------------------------------------------

WriteRaw Log: Writing File < Sb5Baseline_Seg1Mix.raw >

WriteRaw Log: Writing File < Sb5Baseline_Seg1Ica.raw >

WriteRaw Log: Writing File < Sb5Baseline_T085Fltrd.raw >

WriteRaw Log: Writing File < Sb5Baseline_T085Blnks.raw >

WriteRaw Log: Writing File < Sb5Baseline_T090Fltrd.raw >

WriteRaw Log: Writing File < Sb5Baseline_T090Blnks.raw >

WriteRaw Log: Writing File < Sb5Baseline_T095Fltrd.raw >

WriteRaw Log: Writing File < Sb5Baseline_T095Blnks.raw >

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

< Return > to continue...
```

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 eegNetType.m. (See this .m files for details.)

Script M Files

Driver.m

```
% ICA Blink Removal - Driver Program

global eegData metaData logFile logFileIx;
numBlinkPntrs = []; logFileIx = 0;
clc;


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


[fileName, fileExt, dataFileFid, eegDataStart, numBytes, precision, badChanProtocol, badChanPntr, ...
 badChanTol, satObsProtocol, icaProtocol, blinkProtocol, vctrFltrTol, numVctrFltrTol, numProc, ...
 segment, numSegment, messageStem, vftFileStem, autoInputMode] = userInfo;

numChan = metaData.numChan;
numSamples = metaData.numSamples;
numBlinkPntrs = zeros(numSegment, numVctrFltrTol);

save([fileName '_UserInfoLog'], 'logFile', '-mat');
logFile = {}; logFileIx = 0;


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


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

save([fileName '_PreFilterLog'], 'logFile', '-mat');
disp(char(logFile)); logFile = {}; logFileIx = 0;

if (~autoInputMode)
   disp(sprintf('\n< Return > to continue...'));
   pause;
end


% Open binary scratch files. -----------------------------------------------------------------------------------------------


for i = 1 : numVctrFltrTol

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

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

end

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

scratchFid(numVctrFltrTol + 2) = fopen([fileName '_Ica'], 'w+');
if (scratchFid(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
     icaMsg = 'ICA Decomposition Protocol: FastICA (Tanh) | ';
  case 2
     icaMsg = 'ICA Decomposition Protocol: Infomax (binica) | ';
  case 3
     icaMsg = 'ICA Decomposition Protocol: Infomax (runica) | ';
  case 4
     icaMsg = 'ICA Decomposition Protocol: NIC FastICA | ';
end

for i = 1 : numSegment

   if (numSegment ~= 1)
      readRawData(dataFileFid, numBytes, precision, segSize, i, eegDataStart, (i == 1), 0, 0);
   end

   usrMsg = sprintf('\n%s%s\n', icaMsg , messageStem{i});
   clc; disp(usrMsg); logFileIx = logFileIx + 1; logFile{logFileIx} = usrMsg;

   switch icaProtocol
      case 1
         [A, W] = fastica(eegData(goodChanPntr, goodObsPntr{i}), 'displayMode', 'off', 'g', 'tanh');
      case 2
         [Wght, Sph] = binica(eegData(goodChanPntr, goodObsPntr{i}), 'maxsteps', 5000);
         W = Wght * Sph;
         A = inv(W);
      case 3
        [Wght, Sph] = runica(eegData(goodChanPntr, goodObsPntr{i}), 'maxsteps', 5000);
         W = Wght * Sph;
         A = inv(W);
      case 4
         [dataFile, unMixFile, wgtFile, sphFile, currDir, dataDir, execDir] = nicICAForms(numChan, numSamples);
         fid = fopen([dataDir dataFile], 'wb');
         fwrite(fid, eegData(goodChanPntr, goodObsPntr{i}), 'real*8');
         fclose(fid);
         cd(execDir);
         eval(['!mpirun -np ' int2str(numProc) ' -machinefile cpus signal_cleaner ']);
         fid = fopen(unMixFile, 'rb');
         W = fread(fid, [numGoodChan, numGoodChan], 'real*8');
         fclose(fid);
         cd(currDir);
         A = inv(W);
   end

   fwrite(scratchFid(numVctrFltrTol + 1), A, precision);
   fwrite(scratchFid(numVctrFltrTol + 2), W * eegData(goodChanPntr, goodObsPntr{i}), precision);

   [numBlinkPntrs(i,:)] = blinkAway(scratchFid, dataFileFid, numBytes, precision, eegDataStart, goodChanPntr, ...
      badChanPntr, numGoodChan, goodObsPntr{i}, badObsPntr{i}, satObsProtocol, blinkProtocol, segSize(i), ...
      A, W, VEOG, numVEOG, vctrFltrTol, numVctrFltrTol, messageStem{i}, vftFileStem);

   save([fileName '_ICALogSeg' int2str(i)], 'logFile', '-mat');
   clc; disp(char(logFile)); logFile = {}; logFileIx = 0;

   if (~autoInputMode)
      disp(sprintf('\n------------------------------------------------------------\n\n< Return > to continue...'));
      pause;
   end

end


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

```
if (metaData.numEvents ~= 0)
   [eventData] = readRawData(dataFileFid, numBytes, precision, [], [], eegDataStart, 0, 1, 0);
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;
   metaData.numSamples = sum(numGoodObs);
   numSamples = metaData.numSamples;

   for i = 1 : numSegment
      goodObsPntr{i} = 1 : segSize(i);
   end

   if (metaData.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; write to .raw format. -------------------------------------------------------------


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

if (filStatus1 == 0) & (filStatus2 == 0)
   clc; disp(sprintf('\nWriting Output ---------------------------------------------'));
else
   errMsg = sprintf('\n!!! BOF Seek Failure: A / IC Data Arrays !!!\n'); error(errMsg);
end

for i = 1 : numSegment

   A = zeros(numChan, numChan); s = zeros(numChan, segSize(i));

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

   writeRaw([fileName '_Seg' int2str(i) 'Mix'], A, [], 0, [], numChan, numChan, ...
                                             metaData.samplingRate, metaData.versionNumber);

   s(goodChanPntr, goodObsPntr{i}) = fread(scratchFid(numVctrFltrTol + 2), [numGoodChan, numGoodObs(i)],
            precision);

   writeRaw([fileName '_Seg' int2str(i) 'Ica'], s, [], 0, [], numChan, segSize(i), ...
                                             metaData.samplingRate, metaData.versionNumber);

end

clear A s;


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


for i = 1 : numVctrFltrTol

   filStatus = fseek(scratchFid(i), 0, 'bof');
```

```
    if (filStatus ~= 0)
        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(goodChanPntr, segStart(j):segStop(j)) = fread(scratchFid(i), [numGoodChan, segSize(j)],
                precision);
            pureBlinks(goodChanPntr, segStart(j):segStop(j)) = fread(scratchFid(i), [numGoodChan, segSize(j)],
                precision);
        end

        writeRaw([fileName vftFileStem{i} 'Fltrd'], filteredEeg, metaData, 0, eventData);
        writeRaw([fileName vftFileStem{i} 'Blnks'], pureBlinks, metaData, 0, eventData);

    else

        usrMsg = sprintf(['\nVector Filter Tolerance: %6.4f\n' ...
                         'No ICA blink activations in 1 (or more) segments: EEG data not filtered.'], vctrFltrTol(i));
        logFileIx = logFileIx + 1; logFile{logFileIx} = usrMsg;

    end

end

if (metaData.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')
        writeEvents([fileName '_Events'], eventData);
    end
end

save([fileName '_WriteRawLog'], 'logFile', '-mat');
disp(char(logFile)); logFile = {}; logFileIx = 0;

if (~autoInputMode)
    disp(sprintf('\n------------------------------------------------------------\n\n< Return > to continue...'));
    pause;
end


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


filStatus = fclose('all');

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

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

eval(['! rm ' fileName '_Mix']);
eval(['! rm ' fileName '_Ica']);

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

```
if (~autoInputMode)
   eval(['! mkdir ICA-' fileName]);
   eval(['! mv ' fileName '* ICA-' fileName]);
end

clc;
```

UserInfo.m

```
function [fileName, fileExt, dataFileFid, eegDataStart, numBytes, precision, badChanProtocol, ...
        badChanPntr, badChanTol, satObsProtocol, icaProtocol, blinkProtocol, vctrFltrTol, ...
        numVctrFltrTol, numProc, segment, numSegment, messageStem, vftFileStem, autoInputMode] = userInfo;

fileExt = '.raw'; global metaData logFile logFileIx;


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


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

   AutoDriver;
   [dataFileFid, eegDataStart, numBytes, precision] = readRawHeader(fileName, fileExt);

   save([fileName '_ReadRawHeaderLog'], 'logFile', '-mat');
   logFile = {}; logFileIx = 1;
   logFile{logFileIx} = sprintf(['\nUserInfo Log: See AutoDriver.m For User Settings\n']);

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

      segment = 1;

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

   else

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

   end

   switch blinkProtocol

      case{1, 3}

         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} = ['_T0' int2str(round(100 * vctrFltrTol(i)))];
         end

      case 2

         vctrFltrTol = 0;
         numVctrFltrTol = 1;
         vftFileStem{1} = '_VEOG';

   end
```

```
    autoInputMode = 1;
    return;

else

    autoInputMode = 0;

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" formatted file\n' ...
        '\n2) Remove bad channels / saturated observations\n' ...
        '\n3) Perform an ICA decomposition of the EEG data\n' ...
        '\n4) Extract blink activity via Blink Template Correlation / VEOG Polarity Inversion \n' ...
        '\n5) Write the blink-free EEG to a ".raw" formatted file']);
disp(usrMsg);

fileName = input('\nEnter the ".raw" formatted 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 readRawHeader ----------------------------------------------------------------------------------------------------------


[dataFileFid, eegDataStart, numBytes, precision] = readRawHeader(fileName, fileExt);

save([fileName '_ReadRawHeaderLog'], 'logFile', '-mat');
disp(char(logFile)); logFile = {}; logFileIx = 1;
logFile{logFileIx} = sprintf('\nUserInfo Log ----------------------------------------------');


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


disp(sprintf('\n< Return > to continue...')); 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);
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = 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);
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = 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(sprintf([usrMsg '\n']));
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = 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);
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = usrMsg;

    case 4

        disp(' ');
        badChanPntr = []; badChanTol = 0;

end


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


disp(sprintf('< Return > to continue...')); pause; clc;

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 + 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);
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = usrMsg;

    otherwise

        errMsg = sprintf('\n!!! Next Time, Please Select Protocol # 1, 2 Or 3 !!!\n');
        error(errMsg);

end
```

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


disp(sprintf('\n< Return > to continue...')); pause; clc;

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

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

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

    segment = 1;
    numSegment = input('\nEnter the number of segments (+''ve integer >= 2): ');

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

        usrMsg = sprintf('\nNumber Of EEG Data Segments: %d', numSegment);
        disp(usrMsg);
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = usrMsg;

        for i = 1 : numSegment
            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('\nEEG Data Will Not Be Segmented');
    disp(usrMsg);
    logFileIx = logFileIx + 1;
    logFile{logFileIx} = usrMsg;

    messageStem{1} = 'EEG Data Not Segmented';

end


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


disp(sprintf('\n< Return > to continue...')); pause; clc;

usrMsg = sprintf(['\n------------ ICA Decomposition Protocol ------------\n' ...
        '\n1) FastICA (MATLAB)' ...
        '\n2) Infomax (C Code)' ...
        '\n3) Infomax (MATLAB)' ...
        '\n4) Parallel FastICA (C Code)']);
disp(usrMsg);

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

switch icaProtocol

    case {1, 2, 3, 4}

        usrMsg = sprintf('\nICA Decomposition Protocol: %d', icaProtocol);
        disp(usrMsg);
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = usrMsg;
    otherwise

        errMsg = sprintf('\n!!! Next Time, Please Select Protocol # 1, 2, 3, Or 4 !!!\n');
        error(errMsg);

end

if icaProtocol == 4

    numProc = round(abs(input('\nParallel Processor Implementation | Enter the # of processors: ')));
    usrMsg = sprintf('\n# of Processors: %d', numProc);
    disp(usrMsg);
    logFileIx = logFileIx + 1;
    logFile{logFileIx} = usrMsg;

else

    numProc = 1;

end


% Blink Activity Protocol ------------------------------------------------------------------------------------------------


disp(sprintf('\n< Return > to continue...')); pause; clc;

usrMsg = sprintf(['\n------------ Blink Activity Protocol ------------\n' ...
                  '\nBlink activity at the EEG detectors is determined by:\n' ...
                  '\n1) Blink Template Correlation' ...
                  '\n2) VEOG Polarity Inversion' ...
                  '\n3) Blink Template Correlation + VEOG Polarity Inversion']);
disp(usrMsg);

blinkProtocol = input('\nPlease select the Blink Activity Protocol: ');

switch blinkProtocol

    case {1, 2, 3}

        usrMsg = sprintf('\nBlink Activity Protocol: %d', blinkProtocol);
        disp(usrMsg);
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = usrMsg;

    otherwise

        errMsg = sprintf('\n!!! Next Time, Please Select Protocol # 1, 2 Or 3 !!!\n');
        error(errMsg);

end


% Blink Template Tolerance Protocol ------------------------------------------------------------------------------------


disp(sprintf('\n< Return > to continue...')); pause; clc;

switch blinkProtocol

    case{1, 3}

        usrMsg = sprintf(['\n------------ Blink Template Tolerance Protocol ------------\n' ...
                          '\n1) Specify a single blink template tolerance' ...
                          '\n2) Specify a range of blink template tolerances']);
        disp(usrMsg);

        vctrFltrProtocol = input('\nPlease select the Blink Template Tolerance Protocol: ');

        switch vctrFltrProtocol
        case {1, 2}

            usrMsg = sprintf('\nBlink Template Tolerance Protocol: %d', vctrFltrProtocol);
            disp(usrMsg);
            logFileIx = logFileIx + 1;
            logFile{logFileIx} = 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 blink template tolerance: '));
            usrMsg = sprintf('\nBlink Template Tolerance: %4.3f', vctrFltrTol);
            disp(usrMsg);
            logFileIx = logFileIx + 1;
            logFile{logFileIx} = 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 Increment !!!\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);
            logFileIx = logFileIx + 1;
            logFile{logFileIx} = usrMsg;
            usrMsg = sprintf('\nMaximum Tolerance: %4.3f', vctrFltrTolMax);
            disp(usrMsg);
            logFileIx = logFileIx + 1;
            logFile{logFileIx} = usrMsg;
            usrMsg = sprintf('\nTolerance Increment: %4.3f', vctrFltrTolInc);
            disp(usrMsg);
            logFileIx = logFileIx + 1;
            logFile{logFileIx} = usrMsg;

    end

    numVctrFltrTol = length(vctrFltrTol);

    for i = 1 : numVctrFltrTol
        vftFileStem{i} = ['_T0' int2str(round(100 * vctrFltrTol(i)))];
    end

    disp(sprintf('\n< Return > to continue...')); pause; clc;

case 2

    clc;
    vctrFltrTol = 0;
    numVctrFltrTol = 1;
    vftFileStem{1} = '_VEOG';
```

```
end
```

ReadRawData.m

```
function [eventData] = readRawData(dataFileFid, numBytes, precision, segSize, segNum, eegDataStart,
                rewindToBof, readEventData, verbose);

% readRawData reads an EGI epoch-marked raw format file.
%
% Epoch-marked raw format: Unsegmented simple binary format, version # 2, 4 or 6.
%
% The single sample records (SSR) are extracted to form the eegData matrix,
% with one SSR per column.
%
% The corresponding event records, one per SSR, are extracted to form the
% eventData matrix, with one event record per column.
%
% Input Arguments: dataFileFid - EEG data file ID from fopen.
%
%                  numBytes - # of bytes required by precision.
%
%                  precision - Numeric precision string.
%
%                  eegDataStart - File position indicator for 1st EEG data byte.
%
%                  readEventData - Logical 1 --> read event data.
%
%                  rewindToBof - Logical 1 --> rewind file pointer to 1st EEG data byte.
%
%                  segNum - Current segment (segment #1, segment #2, ...).
%
%                  segSize - Vector of samples in each segment.
%
%                  verbose - generate information for log file.
%
% Output Arguments: eegData - Array of EEG data, channels x samples.
%
%                   eventData - Array of corresponding event data codes.
%
%                   logFile - Cell array of program and data information.
%
%                   logFileIx - Pointer into logFile.
%
%                   metaData - Structure array containing the files MetaData.

global eegData metaData logFile logFileIx;

% Read SSRs into array eegData, corresponding events into array eventData. --------------


if readEventData

    if (metaData.numEvents == 0)

        errMsg = sprintf('\n!!! Error: EGI Raw Format File Contains No Event Data !!!\n');
        error(errMsg);

    end

    numEventData = 0;
    eventData = zeros(metaData.numEvents, metaData.numSamples);

    fseek(dataFileFid, eegDataStart, 'bof');

    for j = 1:metaData.numSamples

        fseek(dataFileFid, metaData.numChan * numBytes, 'cof');
        [eventData(:,j), numEventTemp] = fread(dataFileFid, metaData.numEvents, precision);
        numEventData = numEventData + numEventTemp;

    end

    fseek(dataFileFid, eegDataStart, 'bof');

    if numEventData ~= (metaData.numEvents * metaData.numSamples)

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

    end

    if verbose
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = sprintf('\nEventData Matrix: %d x %d', metaData.numEvents, metaData.numSamples);
    end

else

    eegData = zeros(metaData.numChan, segSize(segNum));

    if rewindToBof
       fseek(dataFileFid, eegDataStart, 'bof');
    end

    if (metaData.numEvents ~= 0)

        numEegData = 0;

        for j = 1:segSize(segNum)

            [eegData(:,j), numEegTemp] = fread(dataFileFid, metaData.numChan, precision);
            fseek(dataFileFid, metaData.numEvents * numBytes, 'cof');
            numEegData = numEegData + numEegTemp;

        end

    else

        [eegData, numEegData] = fread(dataFileFid, [metaData.numChan, segSize(segNum)], precision);

    end

    if (numEegData ~= metaData.numChan * segSize(segNum))

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

    end

    if verbose
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = sprintf('\nEegData Matrix | Seg # %d: %d x %d', segNum, metaData.numChan, ...
                segSize(segNum));
    end

end

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

ReadRawHeader.m

```
function [dataFileFid, eegDataStart, numBytes, precision] =  readRawHeader(fileName, fileExt);

% readRawHeader reads an EGI epoch-marked raw format file.
%
```

```
% Epoch-marked raw format: Unsegmented simple binary format, version # 2, 4 or 6.
%
% Input Arguments: fileName - Name of EGI raw format file, sans extension (string).
%
%                  fileExt - Extension of file (string).
%
% Output Arguments: metaData - Structure array containing the files MetaData.
%
%                   logFile - Cell array of program and data information.
%
%                   logFileIx - Pointer into logFile.
%
%                   dataFileFid - EEG data file ID from fopen.
%
%                   eegDataStart - File position indicator for 1st EEG data byte.
%
%                   numBytes - # of bytes required by precision.
%
%                   precision - Numeric precision string.

global metaData logFile logFileIx;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\nReadRawHeader Log ------------------------------------------');


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


dataFileFid = fopen([fileName fileExt], 'r', 'b');

if (dataFileFid == -1)

   errMsg = sprintf('\n!!! File Access Error: %s !!!\n', [fileName fileExt]);
   error(errMsg);

end

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\nReadRawHeader Input File: %s', [fileName fileExt]);


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


metaData.versionNumber = fread(dataFileFid, 1, 'integer*4');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('VersionNumber = %d', metaData.versionNumber);

switch metaData.versionNumber

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

end

metaData.recordingTimeYear = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('RecordingTimeYear = %d', metaData.recordingTimeYear);

metaData.recordingTimeMonth = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('RecordingTimeMonth = %d', metaData.recordingTimeMonth);

metaData.recordingTimeDay = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('RecordingTimeDay = %d', metaData.recordingTimeDay);

metaData.recordingTimeHour = fread(dataFileFid, 1, 'integer*2');
```

```
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('RecordingTimeHour = %d', metaData.recordingTimeHour);

metaData.recordingTimeMinute = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('RecordingTimeMinute = %d', metaData.recordingTimeMinute);

metaData.recordingTimeSecond = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('RecordingTimeSecond = %d', metaData.recordingTimeSecond);

metaData.recordingTimeMillisec = fread(dataFileFid, 1, 'integer*4');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('RecordingTimeMillisec = %d', metaData.recordingTimeMillisec);

metaData.samplingRate = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('SamplingRate = %d', metaData.samplingRate);

metaData.numChan = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('NumChan = %d', metaData.numChan);

metaData.boardGain = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('BoardGain = %d', metaData.boardGain);

metaData.numConvBits = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('NumConvBits = %d', metaData.numConvBits);

metaData.ampRange = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('AmpRange = %d', metaData.ampRange);

metaData.numSamples = fread(dataFileFid, 1, 'integer*4');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('NumSamples = %d', metaData.numSamples);

metaData.numEvents = fread(dataFileFid, 1, 'integer*2');
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('NumEvents = %d', metaData.numEvents);



for i = 1:metaData.numEvents

    metaData.eventCodes{i} = fread(dataFileFid, [1, 4], 'uchar');
    logFileIx = logFileIx + 1;
    logFile{logFileIx} = sprintf('EventCode #%d = %s', i, char(metaData.eventCodes{i}));

end

eegDataStart = ftell(dataFileFid);


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


switch metaData.versionNumber

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

```
end
```

PreFilter.m

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

global eegData metaData logFile logFileIx;
numChan = metaData.numChan; numSamples = metaData.numSamples;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\nPreFilter Log ---------------------------------------------');


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


[EEG, VEOG, VEOGstr, numVEOG, minAmp, maxAmp] = eegNetType;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\nRecording Hardware: %s', EEG);


% 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


% Read 1st segment of EEG Data. ------------------------------------------------------------------------------------------------


readRawData(dataFileFid, numBytes, precision, segSize, 1, eegDataStart, 0, 0, 1);


% Determine 'bad' channels using 1st segment of EEG Data. ----------------------------------------------------------------------


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));
      logFileIx = logFileIx + 1;
      logFile{logFileIx} = 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\n', ...
            badChanTol, numAutoEtdBadChan);
      if numAutoEtdBadChan
         usrMsg = strvcat(usrMsg, sprintf('Bad Channel(s):'), sprintf('Channel # %d\n', badChanPntr));
      end
      logFileIx = logFileIx + 1;
      logFile{logFileIx} = 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) -
                numUsrMrkdBadChan;
        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));
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = usrMsg;
    case 4
        goodChanPntr = 1:numChan; badChanPntr = [];
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = sprintf('\nNumber Of Bad Channels Removed: 0\n');
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 = [];
        for i = 1:numVEOG
            usrMsg = [usrMsg sprintf('%s%03d\t', VEOGstr{i}, VEOG(i))];
        end
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = sprintf(['Good VEOG Channels:\n\n' usrMsg '\n\n-------------------------------------------------------']);
    case 1
        usrMsg = [];
        for i = find(badVEOG)
            usrMsg = [usrMsg sprintf('%s%03d\t', VEOGstr{i}, VEOG(i))];
        end
        disp(sprintf(['Bad VEOG Channels:\n\n' usrMsg '\n\n-------------------------------------------------------------']));
        errMsg = sprintf('\n!!! Program Terminated In PreFilter: Bad VEOG Channels !!!\n');
        error(errMsg);
end

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


for i = 1 : numSegment

    if (i > 1)
        readRawData(dataFileFid, numBytes, precision, segSize, i, eegDataStart, 0, 0, 1);
    end

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

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

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

    logFileIx = logFileIx + 1;
    logFile{logFileIx} = sprintf('\n%s | Number Of Saturated Observations: %d', messageStem{i}, numBadObs(i));

end

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\n-------------------------------------------------------------');
```

BlinkAway.m

```
function [numBlinkPntrs] = blinkAway(scratchFid, dataFileFid, numBytes, precision, eegDataStart, ...
                                goodChanPntr, badChanPntr, numGoodChan, goodObsPntr, badObsPntr, satObsProtocol, ...
                                blinkProtocol, segSize, A, W, VEOG, numVEOG, vctrFltrTol, numVctrFltrTol, messageStem, ...
                                vftFileStem);

% Inputs:
%
%       eegData - m x n EEG data matrix: n time samples from m detectors.
%
%       metaData - structure array of EEG data information.
%
%       A - mixing matrix (numGoodChan by numGoodChan).
%
%       W - separating matrix (numGoodChan by numGoodChan).
%
%       VEOG - (vector) of ocular channel markers.
%
%       scratchFid - (scalar / vector) file id numbers for binary scratch files.
%
%       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.
%
%       blinkProtocol - (scalar) determines the protocol for finding blinks: Blink template
%                       match, VEOG polarity match or both.
%
%       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_T#>. ----------------------------------------------
%
%        eegData - eegData array, new & improved, with fewer blinks.
%
%        pureBlinks - (array) contains the cumulative projections of blink activations
%                     onto the scalp.
%-------------------------------------------------------------------------------------------------------------------------
%
%        numBlinkPntrs - (scalar / vector) number of blink activations per segment.
```

```
global eegData metaData logFile logFileIx;
numChan = metaData.numChan;
epsilon = 0.0001;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\nBlinkAway Log: %s ---------------------', messageStem);

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


% 1) Left-concatenate the blink template (vctrFltr) to mixing matrix A.
% 2) Determine correlations between the blink template and the columns of A (Spatial Projectors / ICA weights).


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


% Recompute VEOG pointers if bad channels were removed.


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

switch blinkProtocol


case 1  %---------------------------------------------------------------------------------------------------------------


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

veogPntr = find((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 = length(veogPntr);

usrMsg = sprintf('\nHEOG Polarity Inversion | %d ICA Activation(s):', numHeogPntrs);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;
usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [heogPntr; corrVctr(heogPntr)]);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\n------------------------------------------------------------');

usrMsg = sprintf('\nVEOG Polarity Inversion | %d ICA Activation(s):', numVeogPntrs);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;
usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [veogPntr; corrVctr(veogPntr)]);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\n------------------------------------------------------------');

for i = 1 : numVctrFltrTol

    logFileIx = logFileIx + 1;
    logFile{logFileIx} = sprintf('\nBlink Template Tolerance: %4.3f', vctrFltrTol(i));


    % Find pointer to columns of A which correlate to vctrFltr such that CorrCoeff >= vFt.


    blinkPntr = find(corrVctr >= (vctrFltrTol(i) - epsilon));
    numBlinkPntrs(i) = length(blinkPntr);

    usrMsg = sprintf('\nBlink Activity: Blink Template Correlation | %d ICA Activation(s):', numBlinkPntrs(i));
    logFileIx = logFileIx + 1;
    logFile{logFileIx} = usrMsg;
    usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [blinkPntr; corrVctr(blinkPntr)]);
    logFileIx = logFileIx + 1;
    logFile{logFileIx} = usrMsg;

    if ~isempty(blinkPntr)  % Blink template correspondence.


        % Generate ICA activations and "pure" blinks.  Extract blinks from the EEG data.


        pureBlinks = zeros(numGoodChan, segSize);
        pureBlinks(:, goodObsPntr) = A(:, blinkPntr) * W(blinkPntr, :) * eegData(goodChanPntr, goodObsPntr);
        eegData(goodChanPntr, goodObsPntr) = eegData(goodChanPntr, goodObsPntr) - pureBlinks(:, goodObsPntr);

        usrMsg = sprintf('\n*** Blink Activity: %d ICA Activation(s) Subtracted From The EEG Data ***', ...
                numBlinkPntrs(i));
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = usrMsg;


        % Save arrays eegData & pureBlinks to a binary file.


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

    else  % No blink template correspondence.

        logFileIx = logFileIx + 1;
        logFile{logFileIx} = sprintf('\n*** No Blink Activity: Zero ICA Activations Subtracted From The EEG Data ***');

    end

    if (i < numVctrFltrTol)
        rewindFastForward(dataFileFid, numBytes, precision, segSize, 1, eegDataStart, 0, 0, 1);
        readRawData(dataFileFid, numBytes, precision, segSize, 1, eegDataStart, 0, 0, 0);
    end
end


case 2  %--------------------------------------------------------------------------------------------------------------------


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

veogPntr = find((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),:))));
numBlinkPntrs = length(veogPntr);

usrMsg = sprintf('\nHEOG Polarity Inversion | %d ICA Activation(s):', numHeogPntrs);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;
usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [heogPntr; corrVctr(heogPntr)]);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;
```

```
logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\n------------------------------------------------------------');

usrMsg = sprintf('\nBlink Activity: VEOG Polarity Inversion | %d ICA Activation(s):', numBlinkPntrs);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;
usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [veogPntr; corrVctr(veogPntr)]);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;

if ~isempty(veogPntr)  % VEOG polarity match.


   % Generate ICA activations and "pure" blinks. Extract blinks from the EEG data.


   pureBlinks = zeros(numGoodChan, segSize);
   pureBlinks(:, goodObsPntr) = A(:, veogPntr) * W(veogPntr, :) * eegData(goodChanPntr, goodObsPntr);
   eegData(goodChanPntr, goodObsPntr) = eegData(goodChanPntr, goodObsPntr) - pureBlinks(:, goodObsPntr);

   usrMsg = sprintf('\n*** Blink Activity: %d ICA Activation(s) Subtracted From The EEG Data ***', numBlinkPntrs);
   logFileIx = logFileIx + 1;
   logFile{logFileIx} = usrMsg;


   % Save arrays eegData & pureBlinks to a binary file.


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

else  % No VEOG polarity match.

   logFileIx = logFileIx + 1;
   logFile{logFileIx} = sprintf('\n*** No Blink Activity: Zero ICA Activations Subtracted From The EEG Data ***');

end


case 3  %-------------------------------------------------------------------------------------------------------------------


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);

usrMsg = sprintf('\nHEOG Polarity Inversion | %d ICA Activation(s):', numHeogPntrs);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;
usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [find(heogPntr);
corrVctr(find(heogPntr))]);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\n------------------------------------------------------------');

usrMsg = sprintf('\nVEOG Polarity Inversion | %d ICA Activation(s):', numVeogPntrs);
logFileIx = logFileIx + 1;
```

```
logFile{logFileIx} = usrMsg;
usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [find(veogPntr);
corrVctr(find(veogPntr))]);
logFileIx = logFileIx + 1;
logFile{logFileIx} = usrMsg;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\n------------------------------------------------------------');

for i = 1 : numVctrFltrTol

  logFileIx = logFileIx + 1;
  logFile{logFileIx} = sprintf('\nBlink Template Tolerance: %4.3f', vctrFltrTol(i));


  % Find pointer to columns of A which correlate to vctrFltr such that CorrCoeff >= vFt.


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

  usrMsg = sprintf('\nBlink Template Correlation | %d ICA Activation(s):', numBlinkTmpltPntrs);
  logFileIx = logFileIx + 1;
  logFile{logFileIx} = usrMsg;
  usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [find(blinkTmpltPntr); ...
        corrVctr(find(blinkTmpltPntr))]);
  logFileIx = logFileIx + 1;
  logFile{logFileIx} = usrMsg;

  if ~isempty(blinkPntr)  % Blink template correspondence.

    usrMsg = sprintf('\nBlink Activity: Blink Template Correlation & VEOG Polarity Inversion | %d ICA Activation(s):', ...
        numBlinkPntrs(i));
    logFileIx = logFileIx + 1;
    logFile{logFileIx} = usrMsg;
    usrMsg = sprintf('\n%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)\t%04d (%4.3f)', [blinkPntr; corrVctr(blinkPntr)]);
    logFileIx = logFileIx + 1;
    logFile{logFileIx} = usrMsg;


    % Generate ICA activations and "pure" blinks. Extract blinks from the EEG data.


    pureBlinks = zeros(numGoodChan, segSize);
    pureBlinks(:, goodObsPntr) = A(:, blinkPntr) * W(blinkPntr, :) * eegData(goodChanPntr, goodObsPntr);
    eegData(goodChanPntr, goodObsPntr) = eegData(goodChanPntr, goodObsPntr) - pureBlinks(:, goodObsPntr);

    usrMsg = sprintf('\n*** Blink Activity: %d ICA Activation(s) Subtracted From The EEG Data ***', ...
        numBlinkPntrs(i));
    logFileIx = logFileIx + 1;
    logFile{logFileIx} = usrMsg;


    % Save arrays eegData & pureBlinks to a binary file.


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

  else  % No blink template correspondence.
```

```
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = sprintf('\n*** No Blink Activity: Zero ICA Activations Subtracted From The EEG Data ***');

    end

    if (i < numVctrFltrTol)
        rewindFastForward(dataFileFid, numBytes, precision, segSize, 1, eegDataStart, 0, 0, 1);
        readRawData(dataFileFid, numBytes, precision, segSize, 1, eegDataStart, 0, 0, 0);
    end

end

end
```

WriteRaw.m

```
function writeRaw(fileName, data, metaData, addEvents, eventData, numChan, numSamples, ...
                  samplingRate, versionNumber);

% writeRaw writes data, stored in the array data, to the file <fileName.raw>,
% along with its corresponding MetaData and event information, if any.
%
% <fileName.raw> will be an epoch-marked raw format file.
%
% Epoch-marked raw format: Unsegmented simple binary format, version # 2, 4 or 6.
%
% Input Arguments: fileName - Name of EGI raw format file, sans extension (string).
%
%                  metaData - MATLAB structure containing the MetaData.
%
%                  data - m1 by n array containing the data to be written.
%
%                  eventData - m2 by n array containing the corresponding event info.
%                         m2 = # of event types (rows), n = # of samples (columns).
%
%                  addEvents - Logical 1 if user requests default events <epoc> and <tim0>.
%
%                  numChan - # of data channels.
%
%                  numSamples - # of time samples.
%
%                  samplingRate - The rate of sampling (Hz).
%
%                  versionNumber - 2: Integer, 4: Single Precion, 6: Double Precision.
%
% Output Arguments: logFile - Text log of runtime information.

global logFile logFileIx;


% 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

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\nWriteRaw Log: Writing File < %s >', [fileName '.raw']);

if (nargin == 9)  % MetaData NOT supplied.  Instantiate with user-supplied & default values.
    metaData.versionNumber = versionNumber;
    metaData.recordingTimeYear = 0;
    metaData.recordingTimeMonth = 0;
    metaData.recordingTimeDay = 0;
    metaData.recordingTimeHour = 0;
    metaData.recordingTimeMinute = 0;
    metaData.recordingTimeSecond = 0;
    metaData.recordingTimeMillisec = 0;
    metaData.samplingRate = samplingRate;
    metaData.numChan = numChan;
    metaData.boardGain = 0;
    metaData.numConvBits = 0;
    metaData.ampRange = 0;
    metaData.numSamples = numSamples;
    if addEvents
        metaData.numEvents = 2;
        metaData.eventCodes(1) = {'epoc'};
        metaData.eventCodes(2) = {'tim0'};
        eventData = zeros(2, numSamples);
        logFileIx = logFileIx + 1;
        logFile{logFileIx} = sprintf('Generated default event codes <epoc> and <tim0> (= 0 for all time samples).');
    else
        metaData.numEvents = 0;
    end
end


% Write the MetaData information to the file. ---------------------------------------------------------------------------------


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

if (metaData.numEvents ~= 0)

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

end


% Finished writing the MetaData. ----------------------------------------------------------------------------------------------


switch metaData.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 (metaData.numEvents ~= 0)
```

```
    for j = 1 : metaData.numSamples
        fwrite(fid, data(:,j), precision);
        fwrite(fid, eventData(:,j), precision);
    end

else

    fwrite(fid, data, precision);

end


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


fclose(fid);
```

WriteEvents.m

```
function writeEvents(fileName, eventData);

global metaData logFile logFileIx;

logFileIx = logFileIx + 1;
logFile{logFileIx} = sprintf('\nWriteEvents Log: Writing File < %s >', [fileName '.txt']);

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

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

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

for j = 1 : metaData.numSamples

    eventPntr = find(eventData(:, j) == 1);

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

end

fclose(fid);
```

RewindFastForward.m

```
function rewindFastForward(dataFileFid, numBytes, precision, segSize, segNum, eegDataStart, rewindToBof, ...
                fastForwardSeg, rewindSeg)

global metaData logFile logFileIx;

if rewindSeg
    fseek(dataFileFid, -1 * numBytes * segSize(segNum) * (metaData.numChan + metaData.numEvents), 'cof');
elseif fastForwardSeg
    fseek(dataFileFid, +1 * numBytes * segSize(segNum) * (metaData.numChan + metaData.numEvents), 'cof');
elseif rewindToBof
    fseek(dataFileFid, eegDataStart, 'bof');
end
```

EegNetType.m

```
function [EEG, VEOG, VEOGstr, numVEOG, minAmp, maxAmp] = eegNetType


% Specify data acquisition hardware.


% ------------------------------------

EEG = 'Net256-34Ch';

% ------------------------------------

switch EEG
   case 'Net256'
      VEOG(1) = 36;
      VEOG(2) = 18;
      VEOG(3) = 242;
      VEOG(4) = 241;
      minAmp = -32768;
      maxAmp = 32767;
      VEOGstr{1} = 'LUVEOG: # ';
      VEOGstr{2} = 'RUVEOG: # ';
      VEOGstr{3} = 'LLVEOG: # ';
      VEOGstr{4} = 'RLVEOG: # ';
   case 'Net256-34Ch'
      VEOG(1) = 6;
      VEOG(2) = 2;
      VEOG(3) = 33;
      VEOG(4) = 34;
      minAmp = -32768;
      maxAmp = 32768;
      VEOGstr{1} = 'LUVEOG: # ';
      VEOGstr{2} = 'RUVEOG: # ';
      VEOGstr{3} = 'LLVEOG: # ';
      VEOGstr{4} = 'RLVEOG: # ';
   otherwise
      errMsg = sprintf('\n!!! Data For %s Is Not Available !!!\n', EEG);
      error(errMsg);
end

numVEOG = length(VEOG);
```

NicIcaForms.m

```
function [dataFile, unMixFile, wgtFile, sphFile, currDir, dataDir, execDir] = nicICAForms(numChan, numSamples);

   algorithm = 'FastICA';
   blockType = 'SINGLE';
   contrastFcn = 'tanh';
   errTol = 0.0001;
   showConvergence = 'no';

   dataFile = 'A.dat';
   dataOrientation = 'observations x channels';
   dataType = 'binary';

   excludeChan = '{}';
   excludeSamples = '{}';

   seedType = 'random';
   constantSeedValue = 0.5;
   userMatrixSeed = '{}';
```

```
  spheringFile = '""';
  outputFile = 'NicIcaUnMixMtrx';

  currDir = pwd;
  dataDir = [currDir '/../workbench/data/'];
  execDir = [currDir '/../workbench/tool_box/bin/'];
  formsDir = [currDir '/../workbench/tool_box/models/signal_cleaning/forms/'];

%CLEANING FORM--------------------------------------------------------
%Method refers to the data cleaning method you wish to use. These
%methods include:
%
%ICA algorithms
%InfoMax
%FastICA
%PCA algorithms
%PCA
%
%Algorithm: FastICA
%----------------------------------------------------------------------
%
%Data Source File: "../../data/A.dat"
%----------------------------------------------------------------------
%
%Order refers to the Data Layout:
%observations x channels OR
%channels x observations
%
%Data Layout: observations x channels
%----------------------------------------------------------------------
%
%Type refers to binary or ASCII source data b->binary, A->ASCII
%
%Data Type: binary
%----------------------------------------------------------------------
%
%Total Channels: 7
%Total Samples: 2500
%
%
%Select the range of channels and observations to use in computing
%the decomposition. If the set is empty, then use all
%channels/samples. The set can contain ranges (e.g., {2-5}) or single
%channels (e.g., {9}), or a combination (e.g., {2-5, 8, 10, 14-23}).
%
%Selected Channels: {}
%Selected Samples: {}
%----------------------------------------------------------------------

fid = fopen([formsDir 'cleaning.form'], 'w');

fprintf(fid, 'Algorithm: %s\n', algorithm);
fprintf(fid, 'Data Source File: %s\n', ['../../data/' dataFile]);
fprintf(fid, 'Data Layout: %s\n', dataOrientation);
fprintf(fid, 'Data Type: %s\n', dataType);
fprintf(fid, 'Total Channels: %d\n', numChan);
fprintf(fid, 'Total Samples: %d\n', numSamples);
fprintf(fid, 'Selected Channels: %s\n', excludeChan);
fprintf(fid, 'Selected Samples: %s\n', excludeSamples);

fclose(fid);

%FASTICA FORM---------------------------------------------------------
%For block type --
%SINGLE => one component at a time
%       BLOCK  => all components simultaneously
%
%Error tolerance: 0.0001
%Block type: SINGLE
%----------------------------------------------------------------------
```

```
%
%Contrast function choices are:
%cubic
%gaussian
%tanh
%
%Contrast function: tanh
%----------------------------------------------------------------------
%
%Initialization of vector, random values, constant values or
%user defined
%
%Initial W: random  (random, constant, user)
%Constant value: 0.5
%User defined matrix: {}
%----------------------------------------------------------------------
%
%Sphering matrix source, if no source file is included, the
%process will compute the sphering matrix. The size is assumed to be
%channels x channels, where channels is set in the cleaner form
%
%Sphering file: ""
%Data type: binary
%----------------------------------------------------------------------

fid = fopen([formsDir 'fast_ica.form'], 'w');

fprintf(fid, 'Error tolerance: %08.6f\n', errTol);
fprintf(fid, 'Block type: %s\n', blockType);
fprintf(fid, 'Contrast function: %s\n', contrastFcn);
fprintf(fid, 'Initial W: %s\n', seedType);
fprintf(fid, 'Constant value: %08.6f\n', constantSeedValue);
fprintf(fid, 'User defined matrix: %s\n', userMatrixSeed);
fprintf(fid, 'Sphering file: %s\n', spheringFile);
fprintf(fid, 'Data type: %s\n', dataType);

fclose(fid);

%ICAOUTPUT FORM--------------------------------------------------------
%Show convergence: no
%
%----------------------------------------------------------------------
%Output instructions. For each of the following items, indicate
%whether you want a print of the object. Note: some algorithms don't
%compute all of the available object. If your algorithm selection
%does not compute a request object that object will not be printed.
%
%No print       : n(o/one)
%ascii form     : a(scii)
%binary form    : bi(n)
%both           : bo(th)
%
%Screen ---------------------------------------------------------------
%Weight matrix screen: n
%Mixing matrix screen: n
%Unmixing matrix screen: n
%Q matrix screen: n
%Lambda ^(-1/2) matrix screen: n
%Average vector screen: n
%Covariance matrix screen: n
%Sphering matrix screen: n
%
%File-----------------------------------------------------------------
%Note: if no file name is provided, the data source file will be
%used as a file stem. The output file will have the prefix A
%indicating ASCII data or B indicating a binary file. Each file
%will have a suffix appended to it indicating the file type. For
%example, given a data file xxx.dat, a request of an ASCII version
%of the weight matrix will be A_xxx.dat.sph.
%
```

```
%File name: output
%Weight matrix file: bi                    .wgt
%Mixing matrix file: n                     .mix
%Unmixing matrix file: bi                  .umx
%Q matrix file: bi                         .q
%Lambda ^(-1/2) matrix file: bi            .lam
%Average vector file: bi                   .avg
%Covariance matrix file: bi                .cov
%Sphering matrix file: bi                  .sph
%-------------------------------------------------------------------

fid = fopen([formsDir 'ica_output.form'], 'w');

fprintf(fid, 'Show convergence: %s\n', showConvergence);
fprintf(fid, 'Weight matrix screen: n\n');
fprintf(fid, 'Mixing matrix screen: n\n');
fprintf(fid, 'Unmixing matrix screen: n\n');
fprintf(fid, 'Q matrix screen: n\n');
fprintf(fid, 'Lambda ^(-1/2) matrix screen: n\n');
fprintf(fid, 'Average vector screen: n\n');
fprintf(fid, 'Covariance matrix screen: n\n');
fprintf(fid, 'Sphering matrix screen: n\n');
fprintf(fid, 'File name: %s\n', outputFile);
fprintf(fid, 'Weight matrix file: bi\n');
fprintf(fid, 'Mixing matrix file: n\n');
fprintf(fid, 'Unmixing matrix file: bi\n');
fprintf(fid, 'Q matrix file: n\n');
fprintf(fid, 'Lambda ^(-1/2) matrix file: n\n');
fprintf(fid, 'Average vector file: n\n');
fprintf(fid, 'Covariance matrix file: n\n');
fprintf(fid, 'Sphering matrix file: bi\n');

fclose(fid);

if dataType == 'binary'
   unMixFile = ['B_' outputFile '.umx'];
   wgtFile = ['B_' outputFile '.wgt'];
   sphFile = ['B_' outputFile '.sph'];
elseif dataType == 'ASCII'
   unMixFile = ['A_' outputFile '.umx'];
   wgtFile = ['A_' outputFile '.wgt'];
   sphFile = ['A_' outputFile '.sph'];
end
```

Bibliography

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

- [2] Dien, J. (1998). Issues in the application of the average referance. Review, critiques, and recommendations. Behave. Res. Methods, Instruments, and 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.