BE 521 Final Algorithm Report

Team Ballmer Peaks

1-2 Brews ~ Peak Coding Performance

Joseph Iwasyk and John “Jack” Talley

Ballmer Peaks Algorithm Summary: Our Algorithm maximized efficiency through iterative testing and detailed signal processing before feature application to each subject's ECoG for prediction. Our primary value was derived from the ability to iterate very quickly through different versions of our algorithm by ensuring our model's run-time was only a few minutes at a maximum. This was enabled by the relatively fast computation time of the Linear Decoder Modeling process. To optimize our algorithm, we first educated ourselves on the subject matter through reading a variety of research papers. We factored in useful processing steps suggested by the research, such as incorporating a 37 ms delay for the ECoG signal since it had to match the appropriate DG coordinates. Furthermore, the research gave us guidance on filter selection and to focus more on high-frequency signals. Our features were simple and tested from class: Area, Energy, and Line Length Features. Through iterative testing, we were able to determine each feature or combination of features that predicted the best for each subject. We further reduced noise for each subject by removing certain channels and iterating to predict on our test set. Final steps to increase our accuracy included normalization of the signals and using a Common Average Reference (CAR) as suggested by research literature. Our iterative approach was logged in a spreadsheet with correlation scores for each finger of each subject, enabling us to see exactly what changed after each isolated change we made in the algorithm. Overall, we identified the best features, channels, and filter parameters that worked for each subject and then applied processing steps to further reduce noise of the signal. Our resulting algorithm yielded an average correlation of roughly 54% on the competition set.

Algorithm in Detail:

Research-Oriented Thought Process & Useful Tips: One important & crucial part to our algorithm's success was learning from research that had worked on similar projects. A few papers working with similar ECoG signals, including the Kubánek paper[2] and Bleichner[3] paper , suggesting using a Bandpass Filter with a focus on higher frequency signals. The specific band for each subject would later be optimized, but all bands were between 60 and 500 Hz. The literature [6] also suggested adding a 37 ms delay to the ECOG data as it would be ahead of the

DG coordinate data. We included the 37 ms delay successfully as a phase shift to the ECoG signal. A Common Average Reference (discussed further below) was also used in the literature [2,3] as a method of reducing drift noise from the ECoG. Some papers [2,3] suggested using a feature called the Local Motor Potential (LMP) because it offered good movement discrimination. As discussed below, the LMP feature did not perform well for our tests. Regarding predictive models, there were a variety of papers suggesting more complex Models to predict DG coordinates, such as more educated Deep Learning Models[4] or Linear Switching Models[1]. We decided we wanted to focus on the Linear Decoder Model as it could quickly perform computations.

Linear Decoder Model & Iterative Testing Capabilities: After completing the first part of this final assignment, we were adamant on testing other predictive models such as the Random Forest Predictor. We found that these other models were 1) Not very accurate or 2) Took much longer than the Liner Decoder. Thus, we decided to continue working with the Linear Decoder Model and highlight one of its core value propositions: swift calculation times. Each predictive run for all three subjects on our 30% testing set with 70% training set took a maximum of 2 minutes per run. While first we did not truly embrace the benefits of this quick computation time, we soon realized it would be advantageous in our approach of quickly swapping features, channel combinations, frequency ranges, and other aspects into our algorithm.

Subject-Specific Approach: One of the key aspects was our separation of the three subjects. We hypothesized that each subject would have a unique ECoG/DG coordinate response and thus they should be treated independently. While some overarching trends may persist among the three subjects, other aspects of our algorithm were found to be best optimized for each subject independently. By utilizing the iterative nature of the Linear Decoder Model and our testing design, we were able to quickly and effectively test various combinations of feature, channels, and filter parameters on each subject. By analyzing each subject's score independently and comparing recent scores to previous scores in a spreadsheet (see **Figure 2**), we could ensure each new addition we made to the code would result in an overall increase of algorithm performance. Our next step would have been to investigate how optimizing for each finger of each subject would have performed since we noticed some fingers performed significantly worse than others

under optimized subject conditions. We did not have enough time to fully incorporate this into our model.

Filter Frequency Optimization: As aforementioned, the research guided us to use a second-order Butterworth bandpass filter with a focus on higher frequency ranges. We started with a range of 60 to 150 HZ but soon began realizing that increasing the upper end of the bandwidth range helped increase our scores, indicating that higher frequency components were important for feature extraction. Thus, each subject's max frequency of the bandpass filter was set to 495 Hz, just below the Nyquist frequency of 1000/2 = 500 Hz. The lower cutoff of the bandpass filter, however, was different for each subject: through optimizing the correlation scores by iteratively testing different lower-end filter cutoffs, we found that Subject 1 best performed with lower end of 105 Hz, Subject 2 with 60 Hz, and Subject 3 with 57.5 Hz. There is likely further optimization that could be done with these ranges, but we did not want to waste time or overfit our data.

Channel Selection: Another key aspect of our algorithm that helped increase our correlations was identifying the noisy channels and removing them for each subject. This processing step required us to perform a variety of prediction iterations to identify a channel or range of channels that negatively impacted a subject's score and then remove those channels. The resulting optimized channels for each subject were as follows: Subject 1: [1:5,10,15:28,34:47], Subject 2: [7:13,15:29,31,34:36,38,43,45:46], and Subject 3: [7:29,34:35,41,44,46,48:49,51,54:55,57,61].

Common Average Referencing (CAR): Most papers we examined used common-average referencing to minimize noise and normalize the input ECoG data signals. This method is simple, performing re-referencing of the signal by subtracting the mean signal amplitude of all channels at a given time point from the channel of interest at the same time point. We performed this re-referencing of the signal across the entire session for all three subjects, and noticed a 4-5% aggregate correlation increase when testing in both directions.

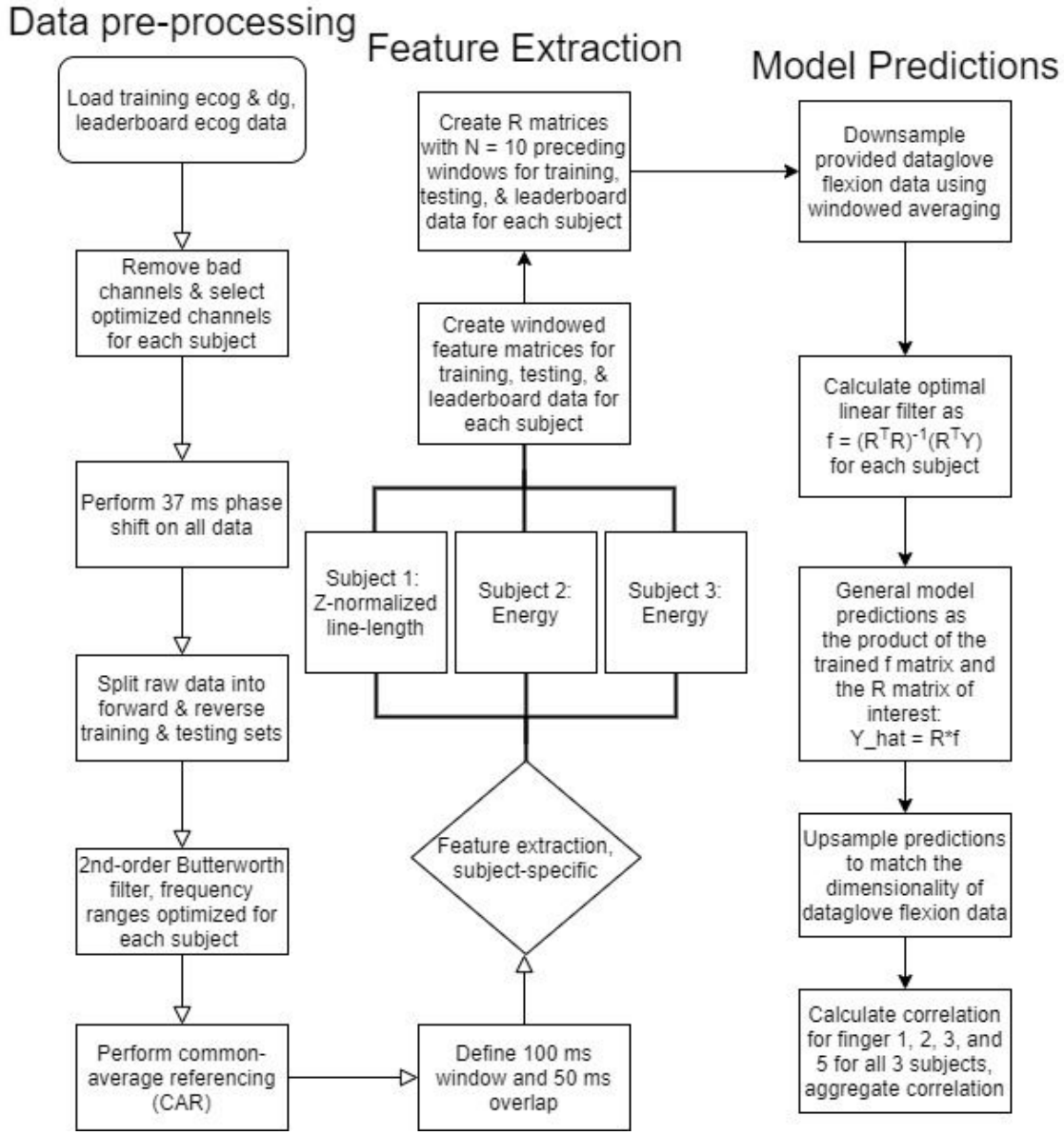
Feature Selection & Z-Scoring: Feature selection was an impactful step in our algorithm design. First, we tried a large combination of averaging, Line Length, Area, Energy, and Frequency Spectrum Analysis Features. However, we began to realize that features in isolation tended to perform better. Furthermore, we investigated and tested each of the aforementioned features alone and in combination on each subject. Our iterative testing yielded the following best

features for each subject: Subject 1 used a Z-Scored Line Length, Subject 2 used the Energy feature, and Subject 3 used the Energy Feature. These findings were perhaps the most counter-intuitive of the entire project as we expected to have better scores with more features, especially with the frequency spectrum analysis suggested by some research. Nonetheless, our best performing features were in isolation and simple ones we have used in class previously.

R-Matrix Creation and Dimensionality: After optimizing our algorithm for data pre-processing and feature extraction, we investigated ways to improve the R-matrix's ability to train the optimal linear decoder model and improve predictions correlations. We increased the number of preceding time windows used in each row of the output R matrix from 3 to 10, and observed a ~3% increase in resulting correlations in both directions. Since we were provided a limited amount of ECoG data with which to train the model, creating an R matrix that contains an increasing number of preceding feature windows improved the training and testing accuracy of our model.

Sampling and Prediction Evaluation: In order to match the dimensionality of the R matrices for each subject, we had to downsample to the provided dataglove finger flexion data. We tried a few different methods to do so, and found that the best results came from using a sliding time window-average downsampling (similar to how the windowed feature matrices were created). After computing the optimal linear decoder predictions, zoInterp was used to upsample the predictions, and the correlation values for fingers 1, 2, 3, and 5 were computed for subjects 1, 2, and 3 along with the aggregated correlation value.

Flowchart of Algorithm:



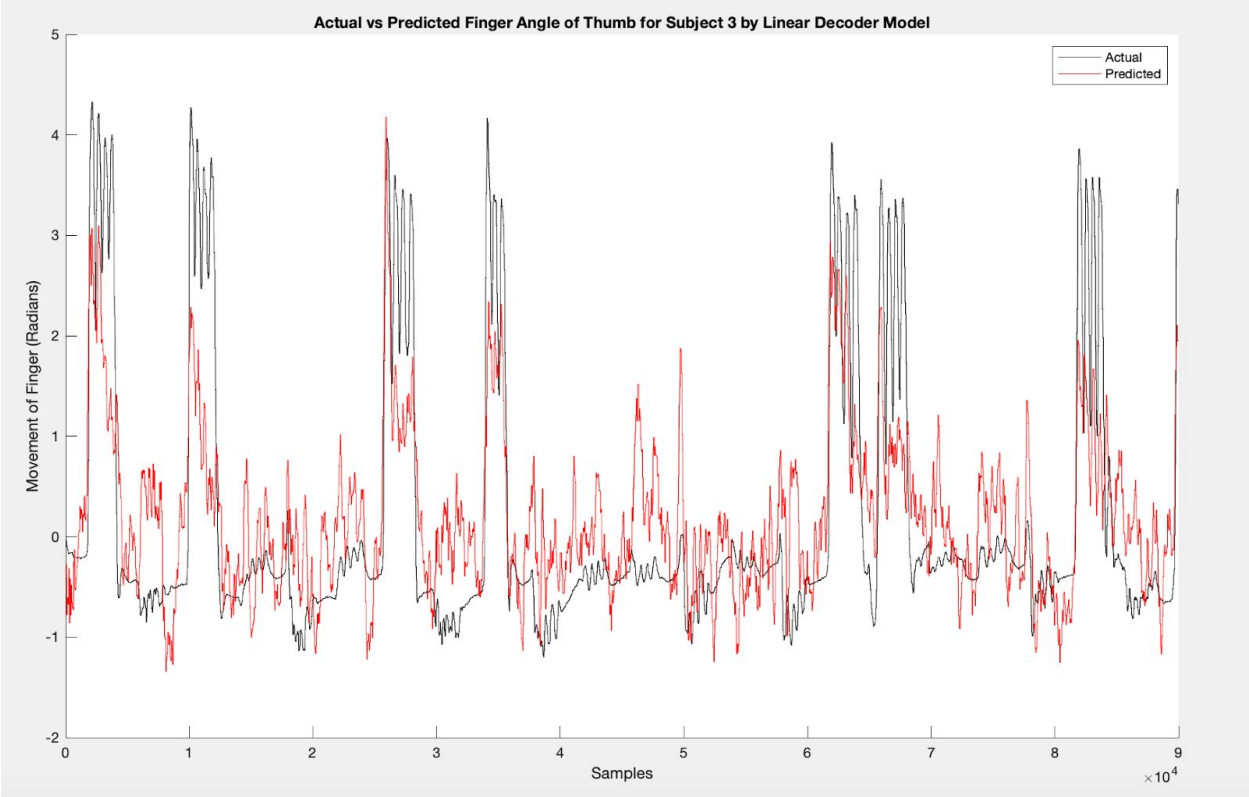


Figure 1 Algorithm Performance Plotted: The plot above shows the predicted vs actual DG coordinate values of Subject 3's thumb over a range of samples. The performance of the correlation was relatively high. This particular plot yielded a roughyl 70% correlation for the thumb. Our team found it useful to frequently observe plots like this to determine the behavior of our predictions relative to the actual DG coordinates.

Subject 1	Condition 1	Direction1 (first 70% train)	Condition 3	Direction1
Finger	Filter: 80:350; Feature: just line length; Channels: [1		Filter [0.21 0.99], just z scored LL, channels: [1:5,10,15:28,34:47]	
1		0.6494125562		0.6596662479
2		0.7539425821		0.7300177468
3		0.2200531459		0.303713074
4		0.6332465014		0.6118664045
5		0.2357048161		0.2774095148
Average		0.4647782751		0.4927016459
Subject 2	**Condition 1**	**Direction1**	**Condition 3**	**Direction1**
Finger	Filter: 65:200; Feature: Line length & Energy; Chann		Filter [0.12 0.99], just energy, channels: [7:13,15:29,31,34:36,38,43,45:46]	
1		0.611238003		0.6327082772
2		0.3691578928		0.3524557204
3		0.3716817858		0.376740487
4		0.5336433579		0.5409847623
5		0.372682421		0.465223122
Average		0.4311900257		0.4567819017
Subject 3	**Condition 1**	**Direction1**		
Finger	Filter: 65:200; Feature: Just Energy; Channels: All C		**Condition 3**	**Direction1**
1		0.6871350518	Filter [0.115 0.99	0.7024199826
2		0.3869448151		0.5142710477
3		0.6147299789		0.7201197235
4		0.6163111404		0.6186021786
5		0.5869762981		0.6310548679
Average		0.568946536		0.6419664054
TOTAL Average:		0.4883049456		0.5304833177

Figure 2 Iterative Testing Log Example: Our team logged iterations of our algorithm in google spreadsheets to track our performance for each subject. The figure above shows "Condition 1" and "Condition 3". Condition 3 changed the features, filter, and channels selected for each subject. As a result, the overall average score increased

Unsuccessful Methods Attempted:

Implementing Other Prediction Models: One method we tried early on in development of our algorithm was to use other predictor models rather than the Linear Decoder. This process took us roughly two days of tests and was largely unsuccessful, setting us back to our original model. For some predictive models, such as trying to use the SVM classifier, we could not reason a way to classify or predict coordinates effectively as we had done in the past. We concluded that a regression model was best as it made the most sense for the non-binary types of predictions we were making. Thus, we tested other types of regression models. One model that yielded some success was the Random Forest Regression Model. This model was able to yield decent predictions, usually 7-10% below our Linear Decoder Model in terms of correlation score. However, the Random Forest Model took nearly 5 to 7 minutes per run, making it roughly 3

times as slow as our Linear Decoder Model. We recognized that this slow processing time would be a severe bottleneck to iterative testing given our team's limited combined computing power. Thus, our attempts at finding an alternate to the Linear Decoder Model were unsuccessful, although we believe there is a better model to predict. One example is the Linear Switching Model that was researched[1]. Another intriguing model that we had thoughts of implementing involved a Convolution Neural Network (CNN) and a recurrent neural network (RNN) called long short term memory (LSTM) to make predictions that outperform the Linear Model [4].

Power Spectrum Features: Some research[1,3] indicated that focusing on specific power spectra of higher frequencies would be a good feature. Taking this advice, we tested the Local Motor Potential (LMP) feature that calculated the average power spectrum across a range of frequencies. This feature did not outperform the Line Length or Energy features though and thus we suspect we either did not implement it properly or the feature simply was not as useful as previously thought. Other frequency domain features that we investigated, but that did not improve our algorithm's performance on training sets over the optimized linear decoder, include MATLAB's spectrogram function, Fast-Fourier Transform (FFT) functions, and bandpower within consecutive frequency bands.

Observations:

Correlation between Flexions of Finger 4 and Fingers 3 & 5: The correlations between flexions of Finger 4 and Fingers 3 & 5 are hypothesized to exist due to the underlying muscle and tendon structure of Finger 4 correlating it with fingers 3 and 5 when the fingers are moved. If an individual moves their ring finger, it is nearly impossible to not move the neighboring fingers 3 or 5. Vice versa, if one moves finger 3 or finger 5, it is near impossible to not move finger 4. Furthermore, it takes muscular effort to attempt to hold finger 4 still while individually moving fingers 3 or 5. Thus, during the experiment for this competition, it was likely that the subjects could not independently move fingers 3 or 5 without moving 4 slightly or vice versa, especially given the length of the experiment at 10 minutes which may have induced muscle fatigue in the finger muscles. Finger 4 likely moved along with fingers 3 or 5 and thus we see high correlations of the flexion for finger 4 with fingers 3 and 5. Our team recognizes that another contributing

factor could simply be that our prediction model is suggests that ECOG signal of finger 3 or 5 correlates highly with finger 4 by coincide; however, it is much more likely that the physical dependence of finger 4 on finger 3 or 5 is a confounding variable and its movement is not intended. Thus, the prediction for movement of finger 4, while possibly true, was not the intended movement of the subject. Nonetheless, to truly replicate human finger movement in, say, a robotic hand in the future, it may be desirable to simulate the slight movement of finger 4 with fingers 3 & 5 for some specific hand function.

Finger Performance in Isolation: Another observation we made was pertaining to our algorithm scores. Upon inspection of the correlations of each finger, we noted that some fingers did not improve scores or even decreased scores despite the overall average of subject increasing. Thus, we hypothesize that a truly optimal algorithm would focus on each finger of each subject and determine the best combination of channels, frequency range, and features to predict with the highest correlation.

Conclusion: In summary, our team thoroughly enjoyed the process of the competition and feels that it was a comprehensive review of our course in BE 521. We first consulted research in the field to educate our approach, providing us with useful tips on how to process the signals. Then, we spent considerable time drafting our approach in google docs. The most rewarding method of our approach was the iterative capabilities of our model: we were able to test, view results, and alter as needed many times. While we were conscious of overfitting, we continued to optimize our algorithm and increase our performance. Our team applied many course-established concepts, such as preprocessing / filtering data or the simple features such as Area and Line Length. Furthermore, we uncovered new techniques during this project that gave us more insight, such as fascinating predictive models in the research that could be implemented later.

From all-nighters we spent testing various iterations to the excitement of seeing our correlation score improve on the leaderboard, our team feels we performed well on this competition and is eager to apply the knowledge we have gained to other projects and fields. Some projects we hope to have explored include other ECOG datasets, virus tracking projects, and Foreign Currency trading algorithms. We appreciate the professors and TA's who have provided us the feedback and course material to guide us in designing Machine Learning Models.

References: (See in-text numbered citations throughout paper)

1. Flamary, Rémi, and Alain Rakotomamonjy. "Decoding Finger Movements from ECoG Signals Using Switching Linear Models." *Frontiers in Neuroscience*, vol. 6, 2012, doi:10.3389/fnins.2012.00029.
2. Kubánek, J et al. "Decoding flexion of individual fingers using electrocorticographic signals in humans." Journal of neural engineering vol. 6,6 (2009): 066001. doi:10.1088/1741-2560/6/6/066001
3. Bleichner, M.G., Freudenburg, Z.V., Jansma, J.M. et al. Give me a sign: decoding four complex hand gestures based on high-density ECoG. Brain Struct Funct 221, 203–216 (2016). https://doi-org.proxy.library.upenn.edu/10.1007/s00429-014-0902-x
4. Xie, Ziqian, et al. "Decoding of Finger Trajectory from ECoG Using Deep Learning." *Journal of Neural Engineering*, vol. 15, no. 3, 2018, p. 036009., doi:10.1088/1741-2552/aa9dbe.
5. Wang, Z., Miller, K. J., & Schalk, G. (2011). Prior knowledge improves decoding of finger flexion from electrocorticographic signals. *Frontiers in neuroscience*, *5*, 127.
6. Miller, K. J., & Schalk, G. (2008). Prediction of finger flexion: 4th brain-computer interface data competition. *BCI Competition IV*, *1*, 1-2.
7. Warland, D. K., Reinagel, P., & Meister, M. (1997). Decoding visual information from a population of retinal ganglion cells. *Journal of neurophysiology*, *78*(5), 2336-2350.
8. Statistics and Machine Learning Toolbox. (n.d.). Retrieved from https://www.mathworks.com/products/statistics.html

Code Appendix: Ballmer Peaks BE 521 Final Project

Prepared by John Bernabei and Brittany Scheid

```
% Author: John Talley
% Collabrator: Joseph Iwasyk

% One of the oldest paradigms of BCI research is motor planning: predicting
% the movement of a limb using recordings from an ensemble of cells involved
% in motor control (usually in primary motor cortex, often called M1).

% This final project involves predicting finger flexion using intracranial EEG (ECoG) in three human
% subjects. The data and problem framing come from the 4th BCI Competition. For the details of the
% problem, experimental protocol, data, and evaluation, please see the original 4th BCI Competition
% documentation (included as separate document). The remainder of the current document discusses
% other aspects of the project relevant to BE521.
```

Start the necessary ieeg.org sessions (0 points)

```
% username = 'jtalley';
% passPath = 'jta_ieeglogin.bin';

% Load training ecog from each of three patients
% s1_train_ecog = IEEGSession('I521_Sub1_Training_ecog', username, passPath);

% Load training dataglove finger flexion values for each of three patients
% s1_train_dg = IEEGSession('I521_Sub1_Training_dg', username, passPath);
addpath('C:\Users\Jack\Documents\Penn\BE 521\Final_Project_Part1');
load('C:\Users\Jack\Documents\Penn\BE 521\Final_Project_Part1\final_proj_part1_data.mat');
load('C:\Users\Jack\Documents\Penn\BE 521\Final_Project_Part1\testRfunction.mat');
load('C:\Users\Jack\Documents\Penn\BE 521\Final_Project_Part1\leaderboard_data.mat');

dg1 = train_dg{1};
dg2 = train_dg{2};
dg3 = train_dg{3};

ecog1 = train_ecog{1};
ecog1 = ecog1(:,[1:5,10,15:28,34:47]);
ecog2 = train_ecog{2};
ecog2 = ecog2(:,[7:13,15:29,31,34:36,38,43,45:46]);
ecog3 = train_ecog{3};
ecog3 = ecog3(:,[7:29,34:35,41,44,46,48:49,51,54:55,57,61]);

leader_ecog1 = leaderboard_ecog{1};
leader_ecog1(:,55) = []; % remove bad channel 55
leader_ecog1 = leader_ecog1(:,[1:5,10,15:28,34:47]);
leader_ecog2 = leaderboard_ecog{2};
leader_ecog2(:,[21,38]) = []; % remove bad channels 21 and 37
leader_ecog2 = leader_ecog2(:,[7:13,15:29,31,34:36,38,43,45:46]);
leader_ecog3 = leaderboard_ecog{3};
leader_ecog3 = leader_ecog3(:,[7:29,34:35,41,44,46,48:49,51,54:55,57,61]);


% phase shift 37 ms to ECoG to account for amplifier delay
ecog1(38:end,:) = ecog1(1:end-37,:);
ecog2(38:end,:) = ecog2(1:end-37,:);
ecog3(38:end,:) = ecog3(1:end-37,:);

leader_ecog1(38:end,:) = leader_ecog1(1:end-37,:);
leader_ecog2(38:end,:) = leader_ecog2(1:end-37,:);
```

```
leader_ecog3(38:end,:) = leader_ecog3(1:end-37,:);
```

Extract dataglove and ECoG data %1.1

```
% Dataglove should be (samples x 5) array
% ECoG should be (samples x channels) array

%1.1
% 300,000 samples
Fs = 1000; % Hz
% Nyquist: 500 hz

% Splitting Training/Testing at 70/30
%
train_dg1 = dg1(1:210000,:);
train_dg2 = dg2(1:210000,:);
train_dg3 = dg3(1:210000,:);

test_dg1 = dg1(210001:300000,:);
test_dg2 = dg2(210001:300000,:);
test_dg3 = dg3(210001:300000,:);

train_ecog1 = ecog1(1:210000,:);
train_ecog2 = ecog2(1:210000,:);
train_ecog3 = ecog3(1:210000,:);

test_ecog1 = ecog1(210001:300000,:);
test_ecog2 = ecog2(210001:300000,:);
test_ecog3 = ecog3(210001:300000,:);

% REVERSE
% % % %
% train_dg1 = dg1(90001:300000,:);
% train_dg2 = dg2(90001:300000,:);
% train_dg3 = dg3(90001:300000,:);
%
% test_dg1 = dg1(1:90000,:);
% test_dg2 = dg2(1:90000,:);
% test_dg3 = dg3(1:90000,:);
%
% train_ecog1 = ecog1(90001:300000,:);
% train_ecog2 = ecog2(90001:300000,:);
% train_ecog3 = ecog3(90001:300000,:);
%
% test_ecog1 = ecog1(1:90000,:);
% test_ecog2 = ecog2(1:90000,:);
% test_ecog3 = ecog3(1:90000,:);


% Split data into a train and test set (use at least 50% for training)
```

Filter raw ECoG data %1.2

```
function clean_data = filter_data(raw_eeg)
    %
    % filter_data_release.m
    %
    % Instructions: Write a filter function to clean underlying data.
```

```
    %               The filter type and parameters are up to you.
    %               Points will be awarded for reasonable filter type,
    %               parameters, and correct application. Please note there
    %               are many acceptable answers, but make sure you aren't
    %               throwing out crucial data or adversely distorting the
    %               underlying data!
    %
    % Input:    raw_eeg (samples x channels)
    %
    % Output:   clean_data (samples x channels)
    %
%% Your code here (2 points)
    [b,a] = butter(3,[0.0003 0.4],'bandpass');
    clean_data = filtfilt(b,a,raw_eeg);

    if size(raw_eeg,2) == 34
        [b,a] = butter(2,[0.21 0.99],'bandpass');
        clean_data = filtfilt(b,a,raw_eeg);
    elseif size(clean_data,2) == 30
        [b,a] = butter(2,[0.12 0.99],'bandpass');
        clean_data = filtfilt(b,a,raw_eeg);
    elseif size(clean_data,2) == 35
        [b,a] = butter(2,[0.115 0.9999],'bandpass');
        clean_data = filtfilt(b,a,raw_eeg);
    else
        [b,a] = butter(2,[0.0003 0.4],'bandpass');
        clean_data = filtfilt(b,a,raw_eeg);
    end

end
```

```
%1.2

% train_ecog1_bp = bandpass(train_ecog1, [0.15 200], Fs);
% train_ecog2_bp = bandpass(train_ecog2, [0.15 200], Fs);
% train_ecog3_bp = bandpass(train_ecog3, [0.15 200], Fs);
%
% test_ecog1_bp = bandpass(test_ecog1, [0.15 200], Fs);
% test_ecog2_bp = bandpass(test_ecog2, [0.15 200], Fs);
% test_ecog3_bp = bandpass(test_ecog3, [0.15 200], Fs);

% butter

% [b,a] = butter(3,[0.0003 0.4],'bandpass'); %3rd order, 0.15/Nyquist:200/Nyquist

% 0.15 Hz to 200 Hz used as in BCI competition doc

train_ecog1_ff = filter_data(train_ecog1);
train_ecog2_ff = filter_data(train_ecog2);
train_ecog3_ff = filter_data(train_ecog3);

test_ecog1_ff = filter_data(test_ecog1);
test_ecog2_ff = filter_data(test_ecog2);
test_ecog3_ff = filter_data(test_ecog3);
```

Get Features

run getWindowedFeats_release function

2.1

```
window_length = 0.100; % seconds
window_overlap = 0.050; % seconds
len = 300000;

NumWins = @(xLen, Fs, winLen, winDisp) round((xLen/Fs - winLen)/winDisp + 1);
numWindows = NumWins(len, Fs, window_length, window_overlap);
numTrainWindows = NumWins(size(train_dg1,1), Fs, window_length, window_overlap);
numTestWindows = NumWins(size(test_dg1,1), Fs, window_length, window_overlap);

% Features:
% 1. LMP Feature
% 2. Frequency Ranges
% 3. CAR (all papers)
% 4. Spectrogram
% 5. Conv
% 6. Power in different freq bands
% 7. LL
% 8. Area
% 9. Energy
% 10. Zx
% 11. Average time-domain voltage
% 12. Average freq-domain magnitude in consecutive 15 Hz bands
% 13. Spike counts (paper)
```

2.2

```
function [features] = get_features(clean_data,fs)
    %
    % get_features_release.m
    %
    % Instructions: Write a function to calculate features.
    %               Please create 4 OR MORE different features for each channel.
    %               Some of these features can be of the same type (for example,
    %               power in different frequency bands, etc) but you should
    %               have at least 2 different types of features as well
    %               (Such as frequency dependent, signal morphology, etc.)
    %               Feel free to use features you have seen before in this
    %               class, features that have been used in the literature
    %               for similar problems, or design your own!
    %
    % Input:    clean_data: (samples x channels) (210000,90000 x 61,46,64)
    %           fs:         sampling frequency
    %
    % Output:   features:   (1 x (channels*features))
    %
%% Your code here (8 points)

LLFn = @(x) sum(abs(diff(x))); % Line Length
Area = @(x) sum(abs(x)); % Area
Energy = @(x) sum(x.^2); % Energy
ZX = @(x) sum((x(1:length(x)-1) > mean(x) & x(2:length(x)) < mean(x)) | (x(1:length(x)-1) < mean(x) & x(2:length(
Var = @(x) var(x); % same as energy
SSC = @(x) sum(diff(sign(diff(x)))~=0);
RMS = @(x) (1/size(x,1))*sum(x.^2);
% Activity = @(x) sum((x - mean(x)).^2);
% Mobility = @(x) (sqrt(var(diff(x)))./(sqrt(var(x)));
% Complexity = @(x) ((sqrt(var(diff(diff((x))))))./(sqrt(var(diff(x))))./((sqrt(var(diff((x))))))./(sqrt(var(x)))
% Kurtosis = @(x) ((1/size(x,1))*sum((x - mean(x)).^4))./(((1/size(x,1))*sum((x - mean(x)).^2)).^2);
% SpikeCounts = @(x) length(findpeaks(x,'MinPeakDistance',10,'MinPeakHeight',0));
% Mean = @(x) mean(x); % Average Time-domain Voltage
% Spec1 = @(x) mean(abs(x(0:fs/length(x):fs/2 >= 75 & 0:fs/length(x):fs/2 <= 100))); % Avg freq domain magnitude
```

```
% Spec2 = @(x) mean(abs(x(0:fs/length(x):fs/2 >= 100 & 0:fs/length(x):fs/2 <= 125))); % Avg freq domain magnitude

feat1 = LLFn(clean_data); % length
feat1Z = (feat1 - mean(feat1))./std(feat1); % Z-normalize
feat2 = Area(clean_data);
feat2Z = (feat2 - mean(feat2))./std(feat2);
feat3 = Energy(clean_data);
feat3Z = (feat3 - mean(feat3))./std(feat3); % Z-normalize
% feat4 = zeros(1,size(clean_data,2));
feat5 = Var(clean_data);
feat5Z = (feat5 - mean(feat5))./std(feat5);
% feat6 = SSC(clean_data);
% feat7 = RMS(clean_data);
% feat8 = Activity(clean_data);
% feat9 = Mobility(clean_data);
% feat10 = Complexity(clean_data);
% feat11 = Kurtosis(clean_data);
% feat12 = range(clean_data);

% %
if size(clean_data,2) == 34
    features = feat1Z; % 1Z

elseif size(clean_data,2) == 30
    features = feat3; % 3
elseif size(clean_data,2) == 35
    features  = feat3; %3
else
    features = feat3;
end

% if size(clean_data,2) == 35
%     features = feat1;
% elseif size(clean_data,2) == 18
%     features = [feat1 feat3];
% else
%     features = feat3; % try only energy
end
```

```
features_1 = get_features(ecog1, Fs);
```

2.3

```
function [all_feats]=getWindowedFeats(raw_data, fs, window_length, window_overlap)
    % This is similar MovingWinFeats.m, but with multiple feature types and
    % many channels / subjects
    % getWindowedFeats_release.m
    %
    % Instructions: Write a function which processes data through the steps
    %               of filtering, feature calculation, creation of R matrix
    %               and returns features.
    %
    %               Points will be awarded for completing each step
    %               appropriately (note that if one of the functions you call
    %               within this script returns a bad output you won't be double
    %               penalized)
    %
    %               Note that you will need to run the filter_data and
    %               get_features functions within this script. We also
```

```
    %               recommend applying the create_R_matrix function here
    %               too.
    %
    % Inputs:   raw_data:       The raw data for all patients
    %           fs:             The raw sampling frequency
    %           window_length:  The length of window
    %           window_overlap: The overlap in window
    %
    % Output:   all_feats:      All calculated features
    %
%% Your code here (3 points)

% First, filter the raw data
clean_data = filter_data(raw_data);

% CAR (Common-average reference)
ref = mean(clean_data');
for j = 1:size(clean_data,2) % all channels
    for i = 1:size(clean_data,1) % all rows
        clean_data(i,j) = clean_data(i,j) - ref(i);
    end
end

% clean_data = (clean_data - mean(clean_data))./std(clean_data); % Z-normalize the data
% Then, loop through sliding windows

    % Within loop calculate feature for each segment (call get_features)

 % MovingWinFeats.m

num_windows = round((length(clean_data)/fs - window_length)/window_overlap + 1);

x_start = 1;
winLen = fs*window_length;
winDisp = fs*window_overlap;

all_feats = zeros(num_windows, length(get_features(clean_data,fs))); % windows x channels*features

for i = 1:num_windows
    x_end = winLen + (i-1)*winDisp;
    feature = clean_data(x_start:x_end, :);
    all_feats(i,:) = get_features(feature,fs);
    x_start = x_end - (winLen - winDisp) + 1;
end
% Finally, return feature matrix

end
```

```
windowed_feats1 = getWindowedFeats(ecog1, Fs, window_length, window_overlap);
windowed_feats2 = getWindowedFeats(ecog2, Fs, window_length, window_overlap);
windowed_feats3 = getWindowedFeats(ecog3, Fs, window_length, window_overlap);

train_windowed_feats1 = getWindowedFeats(train_ecog1, Fs, window_length, window_overlap);
train_windowed_feats2 = getWindowedFeats(train_ecog2, Fs, window_length, window_overlap);
train_windowed_feats3 = getWindowedFeats(train_ecog3, Fs, window_length, window_overlap);

test_windowed_feats1 = getWindowedFeats(test_ecog1, Fs, window_length, window_overlap);
test_windowed_feats2 = getWindowedFeats(test_ecog2, Fs, window_length, window_overlap);
test_windowed_feats3 = getWindowedFeats(test_ecog3, Fs, window_length, window_overlap);

% Leaderboard data windowed features
leader_windowed_feats1 = getWindowedFeats(leader_ecog1, Fs, window_length, window_overlap);
leader_windowed_feats2 = getWindowedFeats(leader_ecog2, Fs, window_length, window_overlap);
```

```
leader_windowed_feats3 = getWindowedFeats(leader_ecog3, Fs, window_length, window_overlap);
```

3.1

```
% M (total number of time bins) x (features*channels* N (time bins))
% = 5999 x (6*61*3) + 1 for leftmost column of 1s
% = 5999 x 1099
```

Create R matrix 3.2

```
function [R]=create_R_matrix(features, N_wind)
    %
    % get_features_release.m
    %
    % Instructions: Write a function to calculate R matrix.
    %
    % Input:    features:   (samples x (channels*features))
    %           N_wind:     Number of windows to use
    %
    % Output:   R:          (samples x (N_wind*channels*features))
    %
%% Your code here (5 points)

R = zeros(size(features,1), N_wind*size(features,2) + 1);
R(:,1) = 1; % first column all 1's

% manually fix time window 1
R(1,2:end) = [features(1,:) features(2,:) features(3,:) features(4,:) features(5,:) features(6,:) features(7,:) f

% manually fix time window 2
R(2,2:end) = [features(2,:) features(3,:) features(4,:) features(5,:) features(6,:) features(7,:) features(8,:) f

% manually fix time window 3
R(3,2:end) = [features(3,:) features(4,:) features(5,:) features(6,:) features(7,:) features(8,:) features(9,:) f

% manually fix time window 4
R(4,2:end) = [features(4,:) features(5,:) features(6,:) features(7,:) features(8,:) features(9,:) features(1,:) f

% manually fix time window 5
R(5,2:end) = [features(5,:) features(6,:) features(7,:) features(8,:) features(9,:) features(1,:) features(2,:) f

% manually fix time window 6
R(6,2:end) = [features(6,:) features(7,:) features(8,:) features(9,:) features(1,:) features(2,:) features(3,:) f

% manually fix time window 7
R(7,2:end) = [features(7,:) features(8,:) features(9,:) features(1,:) features(2,:) features(3,:) features(4,:) f

% manually fix time window 8
R(8,2:end) = [features(8,:) features(9,:) features(1,:) features(2,:) features(3,:) features(4,:) features(5,:) f

% manually fix time window 9
R(9,2:end) = [features(9,:) features(1,:) features(2,:) features(3,:) features(4,:) features(5,:) features(6,:) f


for M = 10:size(features,1)
    R(M, 2:end) = [features(M-9,:) features(M-8,:) features(M-7,:) features(M-6,:) features(M-5,:) features(M-4,:
end
end
```

run create_R_matrix

```
N_wind = 10;
R = create_R_matrix(testR_features, N_wind);

% test construction of Response Matrix

r_mean = mean(mean(R)); % 25.4668 confirmed

R1 = create_R_matrix(windowed_feats1, N_wind);
R1_train = create_R_matrix(train_windowed_feats1, N_wind);
R1_test = create_R_matrix(test_windowed_feats1, N_wind);
R1_leader = create_R_matrix(leader_windowed_feats1, N_wind);

R2 = create_R_matrix(windowed_feats2, N_wind);
R2_train = create_R_matrix(train_windowed_feats2, N_wind);
R2_test = create_R_matrix(test_windowed_feats2, N_wind);
R2_leader = create_R_matrix(leader_windowed_feats2, N_wind);

R3 = create_R_matrix(windowed_feats3, N_wind);
R3_train = create_R_matrix(train_windowed_feats3, N_wind);
R3_test = create_R_matrix(test_windowed_feats3, N_wind);
R3_leader = create_R_matrix(leader_windowed_feats3, N_wind);

% try upsampling R—matrices before linear decoding
```

4.1

calculate linear filter f f = (R'*R)^-1*(R'*Y)

```
% downsample method 1 (use MovingWinFeats method)

x_start = 1;

winLen = Fs*window_length;
winDisp = Fs*window_overlap;

dg1_ds_avg = zeros(numWindows, size(dg1,2));
dg2_ds_avg = zeros(numWindows, size(dg2,2));
dg3_ds_avg = zeros(numWindows, size(dg3,2));

for i = 1:numWindows
    x_end = winLen + (i—1)*winDisp;

    dg1_wind = dg1(x_start:x_end, :);
    dg1_ds_avg(i,:) = mean(dg1_wind);

    dg2_wind = dg2(x_start:x_end, :);
    dg2_ds_avg(i,:) = mean(dg2_wind);

    dg3_wind = dg3(x_start:x_end, :);
    dg3_ds_avg(i,:) = mean(dg3_wind);

    x_start = x_end — (winLen — winDisp) + 1;
end

x_start = 1;

train_dg1_ds_avg = zeros(numTrainWindows, size(dg1,2));
train_dg2_ds_avg = zeros(numTrainWindows, size(dg2,2));
train_dg3_ds_avg = zeros(numTrainWindows, size(dg3,2));
```

```
for i = 1:numTrainWindows
    x_end = winLen + (i-1)*winDisp;

    dg1_wind = train_dg1(x_start:x_end, :);
    train_dg1_ds_avg(i,:) = mean(dg1_wind);

    dg2_wind = train_dg2(x_start:x_end, :);
    train_dg2_ds_avg(i,:) = mean(dg2_wind);

    dg3_wind = train_dg3(x_start:x_end, :);
    train_dg3_ds_avg(i,:) = mean(dg3_wind);

    x_start = x_end - (winLen - winDisp) + 1;
end

x_start = 1;

test_dg1_ds_avg = zeros(numTestWindows, size(dg1,2));
test_dg2_ds_avg = zeros(numTestWindows, size(dg2,2));
test_dg3_ds_avg = zeros(numTestWindows, size(dg3,2));

for i = 1:numTestWindows
    x_end = winLen + (i-1)*winDisp;

    dg1_wind = test_dg1(x_start:x_end, :);
    test_dg1_ds_avg(i,:) = mean(dg1_wind);

    dg2_wind = test_dg2(x_start:x_end, :);
    test_dg2_ds_avg(i,:) = mean(dg2_wind);

    dg3_wind = test_dg3(x_start:x_end, :);
    test_dg3_ds_avg(i,:) = mean(dg3_wind);

    x_start = x_end - (winLen - winDisp) + 1;
end

Y1 = dg1_ds_avg;
Y2 = dg2_ds_avg;
Y3 = dg3_ds_avg;

Y1_train = train_dg1_ds_avg;
Y2_train = train_dg2_ds_avg;
Y3_train = train_dg3_ds_avg;


% downsample method 2: decimate

% for i = 1:5
%     Y1(:,i) = decimate(dg1(:,i),window_overlap*Fs);
%     Y1_train(:,i) = decimate(train_dg1(:,i),window_overlap*Fs);
%
%     Y2(:,i) = decimate(dg2(:,i),window_overlap*Fs);
%     Y2_train(:,i) = decimate(train_dg2(:,i),window_overlap*Fs);
%
%     Y3(:,i) = decimate(dg3(:,i),window_overlap*Fs);
%     Y3_train(:,i) = decimate(train_dg3(:,i),window_overlap*Fs);
% end
%
% Y1(size(Y1,1),:) = [];
% Y1_train(size(Y1_train,1),:) = [];
% Y2(size(Y2,1),:) = [];
% Y2_train(size(Y2_train,1),:) = [];
% Y3(size(Y3,1),:) = [];
% Y3_train(size(Y3_train,1),:) = [];
```

```
% Y1_test = test_dg1_ds_avg;
% Y2_test = test_dg2_ds_avg;
% Y3_test = test_dg3_ds_avg;

f1 = mldivide(R1'*R1, R1'*Y1);
f2 = mldivide(R2'*R2, R2'*Y2);
f3 = mldivide(R3'*R3, R3'*Y3);

Model = cell(3,1);
Model{1} = f1;
Model{2} = f2;
Model{3} = f3;
%
f1_train = mldivide(R1_train'*R1_train, R1_train'*Y1_train);
f2_train = mldivide(R2_train'*R2_train, R2_train'*Y2_train);
f3_train = mldivide(R3_train'*R3_train, R3_train'*Y3_train);

Y1_hat = R1_test*f1_train;
Y1_hat(size(R1_test,1)+1,:) = Y1_hat(size(R1_test,1),:);
% Y1_hat(size(R1_test,1)+2,:) = Y1_hat(size(R1_test,1),:);

Y2_hat = R2_test*f2_train;
Y2_hat(size(R2_test,1)+1,:) = Y2_hat(size(R2_test,1),:);
% Y2_hat(size(R2_test,1)+2,:) = Y2_hat(size(R2_test,1),:);

Y3_hat = R3_test*f3_train;
Y3_hat(size(R3_test,1)+1,:) = Y3_hat(size(R3_test,1),:);
% Y3_hat(size(R3_test,1)+2,:) = Y3_hat(size(R3_test,1),:);

% % Test leaderboard_data.mat data

% Y1_hat = R1_leader*f1;
% Y1_hat(size(R1_leader,1)+1,:) = Y1_hat(size(R1_leader,1),:);
%
% Y2_hat = R2_leader*f2;
% Y2_hat(size(R2_leader,1)+1,:) = Y2_hat(size(R2_leader,1),:);
%
% Y3_hat = R3_leader*f3;
% Y3_hat(size(R3_leader,1)+1,:) = Y3_hat(size(R3_leader,1),:);

% try upsampling R matrix first and going straight to ML model
```

Train classifiers (8 points)

```
% Classifier 1: Get angle predictions using optimal linear decoding. That is,
% calculate the linear filter (i.e. the weights matrix) as defined by
% Equation 1 for all 5 finger angles.


% Try at least 1 other type of machine learning algorithm, you may choose
% to loop through the fingers and train a separate classifier for angles
% corresponding to each finger



% Try a form of either feature or prediction post-processing to try and
% improve underlying data or predictions.
```

Correlate data to get test accuracy and make figures (2 point)

```
function x_interp = zoInterp(x, numInterp)
    x_interp = reshape(repmat(x,numInterp,1),1,length(x)*numInterp);
end
```

```
% Calculate accuracy by correlating predicted and actual angles for each
% finger separately. Hint: You will want to use zointerp to ensure both
% vectors are the same length.

% upsample predictions

zoInterp = @(x,numInterp) reshape(repmat(x,numInterp,1),1,length(x)*numInterp);
Y1_hat_up = zeros(size(test_ecog1,1),size(test_dg1,2));
Y2_hat_up = zeros(size(test_ecog2,1),size(test_dg2,2));
Y3_hat_up = zeros(size(test_ecog3,1),size(test_dg3,2));

for i = 1:5
    Y1_hat_up(:,i) = zoInterp(Y1_hat(:,i)', window_overlap*Fs);
    Y2_hat_up(:,i) = zoInterp(Y2_hat(:,i)', window_overlap*Fs);
    Y3_hat_up(:,i) = zoInterp(Y3_hat(:,i)', window_overlap*Fs);
end

[rho1, pval1] = corr(Y1_hat_up, test_dg1);
[rho2, pval2] = corr(Y2_hat_up, test_dg2);
[rho3, pval3] = corr(Y3_hat_up, test_dg3);

rho1_avg = (rho1(1,1)+rho1(2,2)+rho1(3,3)+rho1(5,5))/4;
rho2_avg = (rho2(1,1)+rho2(2,2)+rho2(3,3)+rho2(5,5))/4;
rho3_avg = (rho3(1,1)+rho3(2,2)+rho3(3,3)+rho3(5,5))/4;
rho_avg = (rho1(1,1)+rho1(2,2)+rho1(3,3)+rho1(5,5)+rho2(1,1)+rho2(2,2)+rho2(3,3)+rho2(5,5)+rho3(1,1)+rho3(2,2)+rh

% Get Leaderboard predictions

% zoInterp = @(x,numInterp) reshape(repmat(x,numInterp,1),1,length(x)*numInterp);
% Y1_hat_up = zeros(size(leader_ecog1,1),size(dg1,2));
% Y2_hat_up = zeros(size(leader_ecog2,1),size(dg2,2));
% Y3_hat_up = zeros(size(leader_ecog3,1),size(dg3,2));
%
% for i = 1:5
%     Y1_hat_up(:,i) = zoInterp(Y1_hat(:,i)', 50);
%     Y2_hat_up(:,i) = zoInterp(Y2_hat(:,i)', 50);
%     Y3_hat_up(:,i) = zoInterp(Y3_hat(:,i)', 50);
% end
%
% % create predicted_dg{} array
% predicted_dg = cell(3,1);
% predicted_dg{1} = Y1_hat_up;
% predicted_dg{2} = Y2_hat_up;
% predicted_dg{3} = Y3_hat_up;
```

```
function [predicted_dg] = make_predictions(test_ecog)

% INPUTS: test_ecog – 3 x 1 cell array containing ECoG for each subject, where test_ecog{i}
% to the ECoG for subject i. Each cell element contains a N x M testing ECoG,
% where N is the number of samples and M is the number of EEG channels.

% OUTPUTS: predicted_dg – 3 x 1 cell array, where predicted_dg{i} contains the
% data_glove prediction for subject i, which is an N x 5 matrix (for
```

```
% fingers 1:5)

% Run time: The script has to run less than 1 hour.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load('Model.mat');
f1 = Model{1};
f2 = Model{2};
f3 = Model{3};

leader_ecog1 = test_ecog{1};
leader_ecog1(:,55) = []; % remove bad channel 55
leader_ecog1 = leader_ecog1(:,[1:5,10,15:28,34:47]);
leader_ecog2 = test_ecog{2};
leader_ecog2(:,[21,38]) = []; % remove bad channels 21 and 37
leader_ecog2 = leader_ecog2(:,[7:13,15:29,31,34:36,38,43,45:46]);
leader_ecog3 = test_ecog{3};
leader_ecog3 = leader_ecog3(:,[7:29,34:35,41,44,46,48:49,51,54:55,57,61]);

leader_ecog1(38:end,:) = leader_ecog1(1:end-37,:);
leader_ecog2(38:end,:) = leader_ecog2(1:end-37,:);
leader_ecog3(38:end,:) = leader_ecog3(1:end-37,:);

leader_ecog1 = leader_ecog1(1:13455,:);
leader_ecog2 = leader_ecog2(1:13455,:);
leader_ecog3 = leader_ecog3(1:13455,:);

Fs = 1000; % Hz
window_length = 0.100; % seconds
window_overlap = 0.050; % seconds

leader_windowed_feats1 = getWindowedFeats(leader_ecog1, Fs, window_length, window_overlap);
leader_windowed_feats2 = getWindowedFeats(leader_ecog2, Fs, window_length, window_overlap);
leader_windowed_feats3 = getWindowedFeats(leader_ecog3, Fs, window_length, window_overlap);

N_wind = 10;
R1_leader = create_R_matrix(leader_windowed_feats1, N_wind);
R2_leader = create_R_matrix(leader_windowed_feats2, N_wind);
R3_leader = create_R_matrix(leader_windowed_feats3, N_wind);

Y1_hat = R1_leader*f1;
Y1_hat(size(R1_leader,1)+1,:) = Y1_hat(size(R1_leader,1),:);

Y2_hat = R2_leader*f2;
Y2_hat(size(R2_leader,1)+1,:) = Y2_hat(size(R2_leader,1),:);

Y3_hat = R3_leader*f3;
Y3_hat(size(R3_leader,1)+1,:) = Y3_hat(size(R3_leader,1),:);

% Get Leaderboard predictions

zoInterp = @(x,numInterp) reshape(repmat(x,numInterp,1),1,length(x)*numInterp);
% Y1_hat_up = zeros(size(leader_ecog1,1),5);
% Y2_hat_up = zeros(size(leader_ecog2,1),5);
% Y3_hat_up = zeros(size(leader_ecog3,1),5);

for i = 1:5
    Y1_hat_up(:,i) = zoInterp(Y1_hat(:,i)', 50);
    Y2_hat_up(:,i) = zoInterp(Y2_hat(:,i)', 50);
    Y3_hat_up(:,i) = zoInterp(Y3_hat(:,i)', 50);
end

Y1_hat_up(end+1:end+5,:) = Y1_hat_up(end-4:end,:);
Y2_hat_up(end+1:end+5,:) = Y2_hat_up(end-4:end,:);
```

```
Y3_hat_up(end+1:end+5,:) = Y3_hat_up(end-4:end,:);

% create predicted_dg{} array
predicted_dg = cell(3,1);
predicted_dg{1} = Y1_hat_up;
predicted_dg{2} = Y2_hat_up;
predicted_dg{3} = Y3_hat_up;



end
```