# **Worcester Polytechnic Institute ⎯ Department of Electrical and Computer Engineering**

## ECE5341 ⎯ Applied Medical Signal Analysis ⎯ Spring 2019

# **Home Assignment 4: Due Friday, 1 February 2019 (4:00 P.M.)**

|  |
| --- |
| **Write your name and ECE box at the top of each page.** |

**Automated ECG Complex Detection “Mini-Project”**

For this home assignment, you will develop MATLAB code to automatically detect (but NOT classify) ECG complexes, and then perform some evaluation of your detector. You will hand in a structured, written report describing your work. This assignment page will describe the technical work that is expected in this home assignment, and then the written report that you must create.

**Introduction:** In class, we discussed a “conventional” ECG complex detector algorithm comprised of the cascade operations of (1) a bandpass filter, (2) a detector (such as an absolute value operation or a squaring operation), and (3) a threshold detection operation (with hysteresis). Each of these three stages has several possible programmable parameter selections, including: the bandpass filter order, highpass cut-off frequency and lowpass cut-off frequency; and the threshold cut-off level and time (and/or amplitude) duration of the hysteresis.

Your assignment will be to develop MATLAB code to perform “conventional” ECG complex detection and then analyze the performance of that code. Note that a basic “conventional” ECG detector should be able to be implemented in less than one page of MATLAB code.

**Data Set for Evaluation:** A public domain data set known as the MIT-BIH Arrhythmia Database contains several recordings from patients being evaluated for cardiac arrhythmias. I have excerpted 30 seconds of one channel of data from each of 23 patients and merged these data into one contiguous WFDB data record, named “mit\_sample.dat” (and corresponding header file “mit\_sample.hea”). I have also created a “gold standard” annotation file “mit\_sample.eaf” which marks each R-wave location with identity “1”. Note that normal and abnormal beats are not distinguished — each is labeled with identity “1” — since you need only perform detection in this mini-project (you need not classify each beat). Each individual subject epoch is normalized in amplitude and re-sampled at 1000 Hz.

Thus, you have a single composite ECG data file, with a total duration of 11.5 minutes, for use in this project. In addition, you are supplied with the “true” annotation results — at least to the best of my ability. Individual subject gains were normalized so that each 30 second data portion would have *similar* amplitude characteristics. Note that the amplitude characteristics are only made similar by the normalization, since differences in ECE complex shape, background noise, etc., alter the effectiveness of the normalization process.

Please download the three database files (mit\_sample.dat, mit\_sample.hea, mit\_sample.eaf) from the “Data” section of the class web page. I recommend that you then review the data in EMGlab to understand its character. Also, download MATLAB function tread\_wfdb.m from the “Software” section of the class web page.

**Primary MATLAB Code to Develop:**

**Note:** MATLAB has a peak detection function “findpeaks()”. You are strictly prohibited from using this MALAB function. The point of this assignment is for you to develop your own signal processing in MATLAB that performs complete peak detection.

1. Write a stand-alone MATLAB function to perform “conventional” ECG detection. The file should accept as input a vector containing the ECG data and any/all parameters that you wish to manipulate within the function (e.g., threshold cut-off level, order of the bandpass filter). As output, the file should return an EMGlab-format structure containing two fields: field “.time” should be a vector of times (in seconds) corresponding to the R-wave peak locations and field “.unit” should be a vector (same dimensions as “.time”) with all values set to “1”. The “.unit” field labels each beat and is needed for subsequent performance comparison. You are also encouraged to return any other information that would help you debug and analyze your function.
2. Write any other functions that are needed to perform analysis of your detector software, as explained below in Analysis. For example, you will be asked to evaluate your detector software as a function of the threshold cut-off level. You might chose to compile the results for this evaluation within a MATLAB “for” loop, and code the entire evaluation as a separate MATLAB function that calls your detector software. For your benefit, a few EMGlab (and related) functions already exist to help you perform evaluation. Be sure you have previously added the EMGlab sub-directories “\readers” and “\readers\extras” to your startup.m file, since these useful EMGlab functions are found in these sub-directories.
   1. Function tread\_wfdb.m: Use this function to read the WFDB file “mit\_sample.dat”. MATLAB code “>>> data = tread\_wfdb(‘mit\_sample.dat’);” will read the data file and return the ECG signal in the variable data. The corresponding .hea file must be in the same directory as the .dat file (and on the MATLAB path). In this case, variable “data” is a vector, since only one ECG channel is included in the file. [This function is available from the “Software” section of the class web page. It is an obsolete version of an EMGlab function, but better for our needs.] Note that tread\_wfdb() *can* have difficulty locating data files if they are not in the current MATLAB directory. Thus, I recommend changing the MATLAB current directory to the directory containing the data files.
   2. EMGlab function load\_eaf.m: Use this function to read the “truth” annotation file “mit\_sample.eaf”. MATLAB code “>>> AnnTruth = load\_eaf(‘mit\_sample.eaf’);” will read the truth annotation file and return all annotations in the structure AnnTruth. In particular, vector AnnTruth.time will list the times, in seconds, of each R-wave occurrence. All entries in the vector AnnTruth.unit will equal one, since we are only performing detection (and not classification) in this project.
   3. EMGlab function save\_eaf.m: You will likely wish to save annotations that you have created to a file. (So that they may be loaded into EMGlab.) To do so, create a structure, for example named AnnTest. One structure member must be the vector AnnTest.time that lists the times, in seconds, of each R-wave occurrence. A second required structure member is the vector AnnTest.unit. For this project, AnnTest.unit must be a vector, of the same dimensions as AnnTest.time, with all values set to one. For example, you could form AnnTest.unit via “>>> AnnTest.unit = ones( size(AnnTest.time) );”.

Once structure AnnTest is available, it can be written to a file in EAF form with the MATLAB code: “>>> save\_eaf(‘FileName.eaf’, AnnTest);”, where “FileName.eaf” is the name you wish to give this file.

* 1. EMGlab function eaf\_compare.m: This function performs a comparison of two annotation files. The function is written in a very general manner. For this mini-project, you should have available an EAF-format “truth” annotation structure within MATLAB, which I will refer to as AnnTruth. (For this project, AnnTruth is loaded from mit\_sample.eaf.) You will also be creating your own “test” annotations from your software, which I will refer to as EAF-format MATLAB structure AnnTest. Function eaf\_compare will look for successful matches, as well as false positives and missed detections, then compute the overall accuracy. Assuming you have formatted AnnTruth and AnnTest within MATLAB as EAF-style structures (format explained above), then you can perform a comparison with the MATLAB code: “>>> Sp = eaf\_compare(AnnTruth, AnnTest, ‘Window’, 0.15);”. The last two arguments will tell the function to use a comparison window that is 150 ms in width, which is the AAMI/ANSII standard for R-wave detectors.

This function will print a few result lines to the screen. (You can suppress these lines by adding the tail input arguments: “, ‘Print’, ‘off’”.) Assuming that you mark all detections as “.unit” value “1” as instructed, the overall accuracy (in percent) of your detector is given as the value under label “Dacc” in the printed output. For automated analysis, the overall accuracy (expressed as a fraction) for your detector can be found from the function output with the MATLAB statement: “>>> accuracy = Sp.Confuse(1,1) / sum( Sp.Confuse(:) )”.

**Analysis:** My interest in this mini-project is for you to write a “classical” detector, and then compare its performance as you vary the parameters of the algorithm. Thus, you should first write the stand-alone detector program, debugging it in MATLAB (perhaps with the help of EMGlab, MATLAB plotting, etc.). Write your program so that it is easy to vary some of the parameters. (For example, code the relevant parameters as input arguments.) ***At a minimum, you are required to produce results in which the threshold cut-off level is varied and at least one other parameter is varied.*** For every combination of parameters, measure the overall accuracy of your detector using EMGlab function “eaf\_compare”, as described above.

Be sure to vary each parameter over a range of values that are relevant to this problem. For example, threshold cut-off values that are clearly too low should produce many false positive detections. Alternatively, threshold cut-off values that are clearly too high will miss many actual detections. Thus, the range of values evaluated should exhibit this characteristic.

It is your choice if you wish to automate recording of the results from eaf\_compare (e.g., by varying a parameter value within a MATLAB “for” loop) using the “Sp.Confuse” output variable, or read the results manually from the function print-out. Typically, presentation of the results in a graphical format is strongly preferred. Think carefully about *how* to present results data, so that information is clearly conveyed. In addition, you might consider simultaneous variation of one or more parameters — for example, varying the threshold cut-off value and the frequency location of the lowpass cut-off filter in one combined analysis.

As a reminder, you are required to present an analysis of overall *automated* accuracy (as measured by eaf\_compare using a 150 ms match window) vs. threshold cut-off level. You are also required to present an analysis of overall accuracy vs. at least one other parameter. Do not use EMGlab to manually correct inaccurate annotations! On this project, we want to study automated algorithms.

After completing the minimum required work, you are free to evaluate additional parameters and/or explore detection beyond the “classical” detector paradigm. In particular, my own experience is that use of a fixed threshold cut-off is problematic with this collection of diverse subjects that have been concatenated together into one data file. A threshold that varies based on the standard deviation of the last few seconds of data might be more appropriate for more robust detection.

**Report Expectations:** I expect excellent reports for these mini-projects. Your grade will be based on your report. That said, I want all of your writing to be related only to your work. You are NOT expected to restate the project assignment, for example.

Reports must be typed *with a minimum font size of 12 pts*. Formatting must be clear. Number all pages. However, given the short time duration of the mini-projects, it is acceptable to (for example) leave extra space around figures; or even label *x*- and *y*-axes of figures by hand. (Of course, ***every axis should be labeled on every plot!***) You likely will not require many (or even any) citations in this report.

Write the report with the standard structure of an engineering report or journal/conference manuscript using the following guidance (label each section in your report):

* Title Page: Please provide a title page with your name(s), this class and the assignment number (Home Assignment x), and your ECE Box number. Leave the rest of the title page blank, to be available for grading comments from the grader.
* Abstract: Not needed (tells nothing new).
* Introduction: Skip this also, as there is nothing new to tell.
* Methods: Write a methods section so that your “investigation” of ECG detection is ***reproducible*** by other ECE students who have an appropriate technical background, ***but know nothing about this specific project.*** Do ***NOT*** assume that your audience is familiar with ***this*** homework assignment. Give enough detail that another ECE student (i.e., someone with the required background in digital signal processing and the physiologic background provided by this course) could replicate your work based on what you have written. Use mathematical expressions and engineering black box diagrams to explain what you have done, and use the written word. Remember to be precise in your description. Good engineering is nothing if it is not precise. Do not use software code (unless you really have to), rather use mathematical expressions to describe your work. This section should explain which parameters are being varied and your mathematical approach to doing so.
* Results: This section describes the results of your analysis. Typically, you should begin with a sub-section that provides step-by-step plots that illustrate your detection procedure on one small section (perhaps 5–10 seconds) of data. Thereafter, a sub-section providing narrated summary plots of your overall accuracy results vs. the parameters varied might follow. It will probably be helpful (perhaps using EMGlab) to plot sections of the data and associated detections where your code failed to perform correctly; it is terribly important to understand the limitations of your automated algorithm.
* Discussion: Use this section to discuss how well your algorithm performed, particularly as you varied the value of your parameters. Discuss the limitations of the “classical” model as well as the limitations of your analysis. Discuss what might be done in the future to improve upon your methods. This section should provide insight into what you have learned in your analysis of these data and this “classical” detection technique.
* Conclusion: A short, summarizing conclusion may be appropriate.
* Appendix: Put *all* of your MATLAB code in the appendix. In general, your code should consist of a collection of MATLAB functions. Each function should include a description of itself within the function file, using MATLAB comment lines, and definition of all inputs and outputs. If appropriate, you can place other material here.
* References: I doubt you will need references for this mini-project. If you do, place them here.

**Grading Rubrics:**

* Grade of “A”: Received by those who exceed the expectations of the “B” grade. Only available to those who write a strong report that clearly meets the expectation of “reproducible.” Exhibited by an outstanding report write-up; or preferably by a report that meets the “reproducible” standard and either investigates additional parameters, a non-conventional detection method, or provides exceptional analysis of the required parameter evaluation.
* Grade of “B”: Received by those who fulfill the minimum mini-project requirements and write an acceptable (but not outstanding) report. Minimum requirements are a functioning detector and adequate analysis of two parameters (one of which must be the threshold cut-off level).
* Grade of “C (or lower)”: Received by those who do not complete the minimum mini-project requirements and/or write a report that is incomplete, too informal, or does not meet the standard of “reproducible.”

**Important Notes on MATLAB Coding/Design Style**

We **WILL** review the MATLAB code from these mini-projects for proper coding style and in-line documentation. Proper coding style refers to both a logical (easy to understand) flow of operations and routine use of the MATLAB matrix-based language. A few specific coding points are:

* All final analysis software should be coded as MATLAB *functions* (not MATLAB script files). Functions have private scope, which is required for mature software development.
* Do not use a “for” loop to perform routine mathematical operations on a vector or matrix when regular MATLAB expressions/operations are available. For example, if you want to add vector “A” to vector “B”, use the MATLAB expression:

>>> C = A + B;

It would be considered ***incorrect*** to use the code:

>>> for m = 1:length(A), C(m) = A(m) + B(m); end

Similarly, most mathematical (and logical) operations on vectors and matrixes can be performed without looping. There **ARE** appropriate usages for a loop, for example looping through a series of processing steps during which the value of a parameter (e.g., the peak detection threshold) is varied each pass through the loop.

* If the same constant appears more than once in a software module, that value should be set once to a variable and then reference should be made to the variable thereafter.
* Pre-allocate variables when appropriate.
* If a logical operation has multiple outcomes, consider using a “switch” statement, rather than nested “if-then-else” statements. Simple “if-then” statements are appropriate if a logical sequence has only two options.