EMF_RP: Documentation

Tao Gong, Jonathan Gordils, Dave Braze

Haskins Laboratories, New Haven, CT, USA

Date 2015/05/13

EMF_RP (available at GitHub: github.com/gtojty/emf_rp) is a general, stand-alone software package designed for assisting single/multi-line text reading experiments and analyzing collected eye movement (EM) data. The package consists of a series of functions to generate pixel-based bitmaps of single/multi-line texts for reading, extract and visualize saccades and fixations detected by eye trackers during reading, and calculate widely-adopted EM measures. Developed using Python 2.7, the package requires some Python packages, such as *os*, *fnmatch*, *re*, *csv*, *codecs*, *pandas*, *numpy*, *PIL*, and *matplotlib*, most of which are pre-installed in many free, open-source portable distributions of Python in Windows or Linux (e.g., WinPython, Anaconda, Maverick, etc.). To use it in High Performance Computing (HPC) clusters, users need to make sure that those required packages are installed properly and can be accessed to by the functions in the package.

The functions in the package can be classified as *user functions*, which can be called directly by users, and *helper functions*, which are mainly called by the user functions to implement various tasks. The package consists of four sections, each containing a number of user and helper functions achieving different tasks:

(a) Functions generating bitmaps and region files for text reading experiments.

(b) Functions extracting information from data files of EyeLink eye tracker and classifying cross line saccades and fixations.

(c) Functions drawing saccades and fixations upon bitmaps.

(d) Functions calculating EM measures.

In the following sections, based on a paragraph reading experiment (the relevant files are in the subfolders at GitHub), we introduce the arguments and outputs of the user functions in different sections, describe involved algorithms, and show examples of using those functions.

Paragraph reading experiment: The experiment aims to analyze individual reading patterns during paragraph reading. In each of the three trials of the experiment, subjects were asked to read overtly a short story shown on the screen, and their eye movements were recorded by EyeLink 1000. The three stories for reading are in the text files in the subfolder bitmap_regfile, story01.txt, story02.txt, story03.txt, which are arbitrarily divided into various lines. The first line in each text file shows information of the story (story name and number of words). Using the functions in EMF_RP, one can generate the bitmaps showing these paragraphs for data collection, classify and visualize recorded saccades and fixations in different lines of text, and calculate general EM measures for analysis.

Functions generating region files and bitmaps: Both generated bitmaps and position information in region files are pixel based, which makes these files less restricted to particular monitors. There are two user functions in this section:

(1) ***Praster*:** to rasterize text using a particular font according to specified parameters. It is intended for displaying single/multi-line text in a bitmap. For a single-line text, it automatically appears in the center of the bitmap; for a multi-line text, it appears, line by line, at a top left position in the bitmap.

Arguments:

- 'direct': directory storing the bitmap and/or region file.
- 'fontpath': fully qualified path to font file.
- 'codeMethod': in Linux, one can use 'utf_8'; in Windows, one can use 'cp1252' or 'utf_8'.
- 'text': a list, containing one or many lines of words for rasterizing, e.g., [u'The quick brown fox jumps over the lazy dog.', u'The lazy tabby cat sleeps in the sun all afternoon.'] (encoded by 'utf_8').
- 'dim': (x, y) dimension of the bitmap, default value is (1280,1024), the origin (0,0) is on the top left corner of the bitmap.
- 'fg': font color in RGB format, default value is (0,0,0).
- 'bg': background color in RGB format, default value is (232,232,232).
- 'regfile': whether to generate the region file together with the bitmap, default value is True.
- 'lmargin': left margin in pixels, default value is 86.
- 'tmargin': top margin in pixels, default value is 86. 'lmargin' and 'tmargin' set the starting position to show multi-line text. For single-line text, the starting position is at the center of the bitmap ('lmargin', 'y'/2.0).
- 'linespace': line spacing in pixels, default value is 43.
- 'xsz': font height in pixels (maximum vertical distance between the highest and lowest painted pixel of each character shown in a particular font), default value is 18.
- 'ysz': font width in pixels, default value is None. 'ysz' takes precedence over 'xsz'.
- 'bbox': whether to draw bounding box around each word, the default value is False. It is for testing purpose.
- 'bbox_big'; whether to draw bounding box around the whole line of word, default value is False. It is for testing purpose.
- 'ID': unique id of the stimuli, used to set the directory and filename of generated bitmaps and region files, default value is 'test' (then, the created bitmap is test.png, the region file is test.region.csv, both in the same folder of the package).
- 'addspace': extra pixels to be added above the top and below the bottom of each line of words, default value is 18. Boundary values of each line of words after adding extra pixels are also stored in the region files.
- 'log': whether to record intermediate results in a log file, default value is False. It is for testing purpose.

Outputs:

Praster generates a bitmap (id.png) displaying the single/multi-line of words in 'text', which can be referred to by functions drawing saccades and fixations. If 'regfile' is 'True', ***Praster*** also generates a region file, which can be referred to by functions in other sections. Information of each word is stored in one line of the region file. The region file contains 15 columns:

- 'Name': identical to 'ID'.
- 'WordID': numerical index of each word in the whole text, starting at 1.
- 'Word': individual word in the whole text, including added space before the word and punctuation afterward.
- 'length': length of a word in characters, including added space before the word and punctuation afterward.
- 'height': height of a word in pixels.
- 'baseline': baseline in pixels of each line of words, words in the same line have identical baseline.
- 'line': numerical index of each line of words, starting at 1.
- 'x1_pos', 'y1_pos': top left position in pixels of each word.
- 'x2_pos', 'y2_pos': bottom right position in pixels of each word.
- 'b_x1', 'b_y1': top left position in pixels of each word, after adding extra pixels ('addspace') above the word.
- 'b_x2', 'b_y2': bottom right position in pixels of each word, after adding extra space below the word. For each word, 'b_x1', 'b_x2' are identical to 'x1_pos', 'x2_pos'. If 'addspace' is 0, 'b_y1', 'b_y2' record the top and bottom positions of all words in the same line.

Algorithm:

Praster reads multiple lines of words in 'text'. For each line of words, except the first word, it arbitrarily adds a space (with length of one character) in front of the other words, and calculates the height and length of each word (including added space before the word and punctuation afterward), and the maximum height among all words in the same line. Then, it sets up the baseline according to 'tmargin', 'lmargin', and 'linespace', displays each word aligned to the baseline, calculates position values of each word, and records these values in the region file. Due to font size and the values of 'tmargin' and 'dim', all lines of words may not be fully shown in the bitmap; in this case, a warning message is thrown to the screen.

Examples:

Praster('.', fontpath, text=[u'The quick brown fox jumps over the lazy dog.']): generate a bitmap (test.png) showing the sentence in the middle and a region file (test.region.csv) recording position information of each word in the sentence (fontpath is set based on the system used). These results are stored in the current folder.

Praster('.', fontpath, text=[u'The quick brown fox jumps over the lazy dog.', u'The lazy tabby cat sleeps in the sun all afternoon.']): generate a bitmap showing the two sentences, line by line, starting from the top left corner of the screen, and a region file recording position information of each word in the two sentences. These results are stored in the current folder.

(2) ***Gen_Bitmap_RegFile*:** to generate the bitmaps and region files from one or many text files. It calls ***Praster*** to work.

Arguments:

- ‘direct’: directory of text files for generating bitmaps and region files, the generated files are also stored there.
- ‘fontName’: name of a chosen font, e.g., ‘LiberationMono’.
- ‘textFileNameList’: list of text file names.
- ‘genmethods’: method to generate bitmaps and region files. If ‘genmethods’ is 0, it is for testing purpose; if ‘genmethods’ is 1, it reads one text file in ‘textFileNameList’; and if ‘genmethods’ is 2, it reads many text files in ‘textFileNameList’, default value is 2.
- ‘codeMethod’, ‘dim’, ‘fg’, ‘bg’, ‘lmargin’, ‘tmargin’, ‘linespace’, ‘xsz’, ‘ysz’, ‘bbox’, ‘bbox_big’, ‘addspace’, ‘log’: identical to those of ***Praster***.

Outputs:

For each text file in ‘textFileNameList’, ***Gen_Bitmap_RegFile*** generates a bitmap and a region file.

Algorithm:

If ‘genmethods’ is 1, ***Gen_Bitmap_RegFile*** reads one text file in ‘textFileNameList’. If the text file contains one or many single/multi-line texts, the function generates a bitmap and a region file for each text, the names of which are story01.png, story01.region.csv, story02.png, story02.region.csv, etc. If ‘genmethods’ is 2, each file in ‘textFileNameList’ contains only one single/multi-line text. ***Gen_Bitmap_RegFile*** reads each file and generates a bitmap and a region file accordingly, the names of which are determined by the names of the text files. If some of the relevant files do not exist, the function throws warning messages and stops.

In a text file containing one multi-line text, users can arbitrary separate the text into different lines using the delimiter ‘\r\n’. In a text file contains many multi-line texts, users can use the delimiter ‘\r\n\r\n’ (an empty line) to separate different multi-line texts. Users can also insert commenting line (starting with ‘#’) to involve information of the text (e.g., story name and number of words). ***Gen_Bitmap_RegFile*** can squeeze out commenting lines and display only texts in bitmaps.

Examples:

Gen_Bitmap_RegFile(‘./bitmap_regfile’, ‘LiberationMono’, [‘Passages.txt’], 1, ‘utf_8’): read Passages.txt in the subfolder bitmap_regfile. The text file contains three multi-line texts. The function generates three bitmaps (story01.png, story02.png, story03.png) and three region files (story01.region.csv, story02.region.csv, story03.region.csv) in the same subfolder. On the screen, the function also summarizes the number of words in each text shown in the bitmaps.

Gen_Bitmap_RegFile(‘./bitmap_regfile’, ‘LiberationMono’, [‘story01.text’, ‘story02.txt’, ‘story03.txt’], 2, ‘utf_8’): read three text files (story01.txt, story02.txt, story03.txt, each containing one multi-line text) in the subfolder bitmap_regfile. The function generates the same bitmaps and region files as in the first example.

Functions extracting basic information and classifying cross line saccades and fixations: Functions in this section require not only the region files created in the previous section but also the EM data file. The data files are the ascii files converted from EyeLink EDF files by way of SR Research's 'edf2asc.exe' utility. To be parsed properly, the ascii files must contain appropriate 'flags' (e.g., 'DATE:' indicates the date of data collection, 'TRIALID' indicates the ID of different trials, 'EBLINK' for a blink event, 'EFIX' for a fixation, 'ESACC' for a saccade, etc. Please refer to the EyeLink manual for relevant flags. We design relevant helper functions to extract this information. Users focusing only on extracted results do not need to understand such detailed information). In the subfolder asc_example, there is an example ascii file. It records the EM data of a subject (the name of the file is the subject ID) reading the three stories shown in the three bitmaps in the subfolder bitmap_regfile. To properly use the functions in this section, users need to put the region files and ascii file(s) in the same folder.

The functions in this section transform raw data in ascii files (basic information of eye-tracker setting and physical locations and durations of detected saccades and fixations) into user-friendly data files (saccades and fixations reliably classified into different words of a single-line text or different lines of words of a multi-line text) for follow-up analysis. Due to different experimental situations, there could be additional preprocessing before transforming an ascii file into classified data files. For example, before reading the paragraph line by line, some subjects may browse the whole paragraph, and the saccades or fixations during such browsing need to be removed; without clearly grasping the instructions, there could be interruptions during reading, and the saccades or fixations detected during such interruptions need to be removed; the experiment may be terminated due to some reasons, and recorded data may become messy in such situation.

Noting these, we design three sets of user functions for extracting information in the EM data file, classifying saccades and fixations into various lines, and notifying users if some data files are inappropriate for follow-up analyses. In addition, there are two versions of user functions in each set: the first version handles only the data file of one subject, whereas the second allows processing data files of many subjects in a batching fashion.

The first set of user functions, ***rec_Sac_Fix*** and ***rec_Sac_Fix_Batch***, only extract information from the ascii data file, without classifying saccades or fixations. Users can remove some questionable saccades or fixations from generated data files, and call the second set of user functions to classify saccades or fixations.

(1) ***rec_Sac_Fix*:** to extract the basic information from the ascii data file and store the results into csv data file.

Arguments:

- 'direct': directory of the ascii file and region files, the generated csv files are also stored there.
- 'datafile': complete name of the ascii file, e.g., '1950024.asc'.

- ‘regionfileNameList’: list of the region file names. If the ascii data file contains the data of many trials, the corresponding region files should be sorted accordingly. As in our experiment, the ascii file stores the reading data of three trials, so ‘regionfileNameList’ should be [‘story01.region.csv’, ‘story02.region.csv’, ‘story03.region.csv’].
- ‘ExpType’: type of experiment, in our experiment, it can be set as ‘RP’, meaning ‘reading paragraph’.

Outputs:

rec_Sac_Fix generates two csv files. *_Sac.csv records extracted saccades in different trials of reading, and *_Fix.csv records extracted fixations in different trials. ‘*’ is identical to the name of the ascii file.

*_Sac.csv contains 20 columns:

- ‘subj’: subject ID.
- ‘trial_id’: numerical indexing of trials, starting at 0.
- ‘trial_type’: type of trials, in our example experiment, it is a combination of ‘ExpType’ and corresponding region file, e.g., ‘RP_story01’.
- ‘sampfreq’: sampling frequency of the eye-tracker, e.g., 250, 500, or 1000, dependent on the model of eye-tracker.
- ‘script’: script used in the experiment.
- ‘sessdate’: date of the experiment.
- ‘srcfile’: EDF file used for generating the ascii file.
- ‘tdur’: total duration of the eye tracking in one trial.
- ‘blinks’: total number of blinks detected in one trial.
- ‘eye’: which eye the EM data are recorded, e.g., ‘L’ (left eye), ‘R’ (right eye), or ‘LR’ (both eyes).
- ‘start’: starting time of the saccade in milliseconds.
- ‘end’: ending time of the saccade in milliseconds.
- ‘duration’: duration of the saccade, equal to ‘end’ – ‘start’ + 1/ ‘sampfreq’.
- ‘x1_pos’, ‘y1_pos’: starting position in pixels of the saccade.
- ‘x2_pos’, ‘y2_pos’: ending position in pixels of the saccade.
- ‘ampl’: amplitude in degrees of the saccade.
- ‘pk’: peak velocity of the saccade.
- ‘line’: on which line of words the saccade occurs. Before classifying, it is 0.

*_Fix.csv contains 18 columns:

- ‘subj’, ‘trial_id’, ‘trial_type’, ‘sampfreq’, ‘script’, ‘sessdate’, ‘srcfile’, ‘tdur’, ‘blinks’, ‘eye’: identical to those of *_Src.csv. Since the follow-up analyses may focus only on saccades or fixations, we let both csv files share basic information.

- ‘start’, ‘end’: starting and ending time of the fixation.
- ‘duration’: duration of the fixation, equal to ‘end’ – ‘start’ + 1/‘sampfreq’.
- ‘x_pos’, ‘y_pos’: position in pixels of the fixation.
- ‘pup_size’: pupil size of the fixation.
- ‘valid’: whether it is a valid (‘yes’) or invalid (‘no’) fixation. In some eye-tracker, the last fixation of a trial is set to be invalid, and much shorter fixations (e.g., less than 50 ms) are also assumed invalid.
- ‘line’: on which line of text the fixation occurs during reading. Before classifying, it is 0.

Algorithm:

rec_Sac_Fix first checks whether the ascii file and relevant region files exist in the specified folder. If not, the function throws warning messages and stops. If all these files exist, ***rec_Sac_Fix*** reads the ascii file, and uses helper functions to extract relevant information of the data file. Then, it creates data frames respectively storing detected saccades and fixations across different trials recorded in the data file.

An important operation here is merging (or lumping) fixations. Generally speaking, a fixation shorter than 50 ms (marked by the global variable **LN**; to change it, use Python codes: global LN; LN = ‘new value’) is deemed short and unreliable for follow-up analysis. For each short fixations, we first check its immediate neighboring fixations, and if those fixations are also short, we merge all of them into one fixation (whose starting and ending times are the earliest and latest times of those fixations, whose duration is the sum of the durations of those fixations, and whose position is the centroid position of those fixations). If the neighboring fixations are not short or the duration of the merged fixation is still short, we check the horizontal distance between this fixation and its two neighboring fixations. If the horizontal distance between this fixation and one of its neighbors is smaller than 50 pixels (about 1.5 character length) (marked by the global variable **ZN**), we merge these two fixations into one. If the horizontal distances between this fixation and its two neighbors are both within 50 pixels, merge this fixation with the closer neighbor in terms of horizontal distance. If the distances between this fixation and its two neighbors are beyond 50 pixels, we either delete this fixation or mark it as an invalid fixation. After the merging operation, the function stores the two data frames into csv files.

(2) ***rec_Sac_Fix_Batch*:** to extract the basic information from many EM data files and store the results into csv files.

Arguments:

- ‘direct’: directory of the ascii files and the region files, the generated csv files are also stored there.
- ‘regionfileNameList’: list of the region file names.
- ‘ExpType’: type of experiment, in our experiment, it can be set as ‘RP’, meaning ‘reading paragraph’.

Outputs:

For each ascii file, ***rec_Sac_Fix_Batch*** generates two csv files: *_Sac.csv and *_Fix.csv. ‘*’ is the same as the ascii file name.

Algorithm:

rec_Sac_Fix_Batch lists all ascii files in 'direct'. Then, for each ascii file, it calls ***rec_Sac_Fix*** to generate corresponding csv files. For the ascii file recording saccades and fixations of both eyes, it first processes and stores the left eye saccades and fixations, and then, the right eye. If the data of either eye are empty, a warning message is thrown to the screen.

Examples:

rec_Sac_Fix('./asc_example', '1950024.asc', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv'], 'RP'): generate 1950024_Sac.csv and 1950024_Fix.csv based on the ascii file 1950024.asc. The csv files contain extracted saccades and fixations detected in the three trials of reading of the three stories.

rec_Sac_Fix_Batch('./asc_example', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv'], 'RP'): generate the csv files based on each of the ascii files in the subfolder. Each ascii file records the reading data of a particular subject reading the three stories in three trials.

The second set of user functions, ***crl_Sac_Fix*** and ***crl_Sac_Fix_Batch***, classify saccades, and especially fixations, into different lines of text. Given classified fixations and saccades, users can calculate the EM measures and conduct follow-up analysis.

In data analysis of an EM reading experiment, reliably classifying detected fixations into different words (single-line text) or different lines of words (multi-line text) is the most critical part. Compared with single-line text reading, reading multi-line text is more natural and frequent in everyday life, but how to classify fixations in multi-line text appears more difficult. First, horizontal positions of fixations are the most reliable indicators of which words subjects are reading, but unlike horizontal dimension, eye-tracker usually has a relatively lower resolution in vertical dimension. In addition, due to individual difference, subjects tend to show various patterns of line reading, which are usually not well aligned with the baselines of shown text. Therefore, even if an eye-tracker has a good resolution in vertical dimension, y-values are not much reliable to classify which lines subjects are reading. Furthermore, some subjects tend to jump forward and backward during reading, especially at the beginning or ending of a line of text, and such jumping may not be achieved within one saccade, which adds difficulty in correctly assigning fixations into different lines of text. Noting these, extension from single-line text to multi-line text is by no means trivial, but requires both technical and psychological considerations.

The second set of user functions aim to overcome these difficulties and enable an efficient and reliable way of classifying saccades and fixations into different lines of text and capturing forward and backward line jumping during reading. Based on previous experiences of doing and observing multi-line text reading, we design our algorithms based on four assumptions:

First, we rely on cross line saccades or cross line fixations to locate lines of text that subjects are reading. A saccade is *a cross line saccade*, if it starts at a line of text and stops at the immediately next line (forward cross line saccade), or it starts at a line of text and stops at the immediately previous line (backward cross line saccade); a fixation is *a cross line fixation*, if it occurs at a line

of text and the immediate fixation before it occurs at the immediately previous line (forward cross line fixation), or it occurs at a line of text and the immediate fixation before it occurs at the immediately next line (backward cross line fixation). Due to parafoveal or peripheral reading, cross line saccades or fixations may not necessary locate at the very beginning or ending of a line.

Second, we assume that during a normal reading of multi-line text, subjects would not jump, forward or backward, more than one line of text. In addition, a backward cross line eye movement is usually achieved by one fixation or saccade, whereas a forward cross line eye movement may be achieved by one or a series of fixations or saccades. This assumption is based on the observations of many subjects in our multi-line text reading experiment.

Third, normal regression in reading of a single line of text is usually not big in terms of horizontal position (jumping only a few words), whereas a cross line saccade (or fixation) tends to jump a big amount (more than a half of total words in the same line) of horizontal position; a big, forward jump in terms of horizontal position indicates that subjects start to read a new line, whereas such a big, backward jump indicates that subjects start to read the previous line.

Finally, at the end of reading, subjects may move their eyes back to the center or beginning of the paragraph. Saccades or fixations detected during such wrap-up behavior are not useful for analysis, and need to be removed. Big change in vertical position may indicate such behavior. Therefore, vertical position, though not accurate enough to classify fixations into different lines of text, is still informative for the ending of reading.

Based on big horizontal jumps, we can trace which line of text subjects are reading, and classify fixations into appropriate words in those lines. Based on big vertical jumps, we can remove saccades and fixations after the ending of reading. These assumptions and operations are implemented by relevant helper functions.

The user functions in this set, ***crl_Sac_Fix*** and ***crl_Sac_Fix_Batch***, read *_Sac.csv and *_Fix.csv created by the first set of user functions, call relevant helper functions to classify saccades and fixations into different lines of the text, and store the results into csv files. Noting that some assumptions may be violated in extreme cases, the user functions also generate a log file recording the information of the data that could be questionable and guiding users to do additional check.

(3) ***crl_Sac_Fix*:** to classify saccades and fixations into different lines of text, and stores the results in four csv files.

Arguments:

- 'direct': directory of *_Sac.csv and *_Fix.csv and the region file, the generated csv files are also stored there.
- 'subj': subject ID, identical to '*' in the csv files.
- 'regfileNameList': list of region files.
- 'ExpType': type of experiment.

Outputs:

crl_Sac_Fix modifies the 'line' column in *_Sac.csv and *_Fix.csv. After modification, the 'line' column in *_Sac.csv becomes either '0' or two values linked by '_', e.g., '2_3', indicating that the saccade is a cross line jumping from line 2 to

line 3 of text. The ‘line’ column in *_Fix.csv becomes a numerical number, starting at 1, indicating which line of text the fixation occurs. If the ‘line’ column of a fixation remains as 0, it means that the fixation happens after the ending of reading. Apart from these two csv files, ***crl_Sac_Fix*** generate two new csv files, *_crlSac.csv and *_crlFix.csv, which respectively record cross line saccades and fixations. Such selective data help users quickly check whether the results are appropriate. For the data file recording saccades and fixations of both eyes, the function first classifies the left eye cross line saccades and fixations, and then, the right eye.

*_crlSac.csv contains 15 columns:

- ‘subj’, ‘trial_id’, ‘eye’: same as those columns in *_Sac.csv.
- ‘startline’, ‘endline’: starting and ending lines of the cross line saccade.
- ‘Saclineindex’: numerical index of the saccade in *_Sac.csv, starting from the first saccade of the trial.
- ‘start’, ‘end’, ‘duration’, ‘x1_pos’, ‘y1_pos’, ‘x2_pos’, ‘y2_pos’, ‘ampl’, ‘pk’: same as those columns in *_Sac.csv, recording basic information of the saccade.

*_crlFix.csv contains 13 columns:

- ‘subj’, ‘trial_id’, ‘eye’: same as those columns in *_Fix.csv.
- ‘startline’, ‘endline’: starting and ending lines of the cross line fixation.
- ‘Fixlineindex’: numerical index of the fixation in *_Fix.csv, starting from the first fixation of the trial.
- ‘start’, ‘end’, ‘duration’, ‘x_pos’, ‘y_pos’, ‘pup_size’, ‘valid’: same as those columns in *_Fix.csv, recording basic information of the fixation.

Apart from these csv files, ***crl_Sac_Fix*** also generates a text file, log.txt, each line of which records the information of questionable data, e.g., ‘Subj: 1950021 Trial 0 crlSac start/end need check!’ or ‘Subj: 1950072 Trial 2 crlFix start/end need check!’, indicating that the data of a particular subject in a particular trial need check. If the first identified cross line saccade or fixation is a backward one, or the last identified cross line saccade or fixation does not cover the last line of text, the data in this trial are marked as questionable, and its information (subject ID, trial ID, saccade or fixation) is put into the log file.

Algorithm:

crl_Sac_Fix first checks whether the relevant csv files and region files exist in the specified folder; if not, the function throws warning messages and stops. If all these files exist, ***crl_Sac_Fix*** reads the data from the csv files, calls the helper functions to classify saccades and fixations into different lines of text, and stores the results into the four csv files.

Identification of cross line saccades or fixations is done as follows.

First, based on the region file, for each line of words, we calculate the horizontal distance between the center of the first word and the center of the last word, and set the cross line jumping threshold of this line of text as 60% (marked by the global variable **FIXDIFF_RATIO**) of this biggest horizontal distance. Considering that a backward cross line saccade or fixation

may not necessary start at the very beginning of a line, we set up a frontal region between the starting position of the line to 20% (marked by the global variable **FRONT_RANGE_RATIO**) of the biggest horizontal distance in that line, and assume that if there is a saccade starting in this region and undergoing a jump larger than the cross line jumping threshold, the saccade is identified as a backward cross line saccade. Similarly, if a fixation lies in the frontal region, and the 'x_pos' difference between it and its immediately next fixation is bigger than the cross line jumping threshold, the next fixation is identified as a backward cross line fixation.

Second, we reorganize the saccades and fixations by grouping concatenate saccades or fixations all moving leftward (*a leftward moving saccade* is a saccade whose x2_pos is smaller than x1_pos; *a leftward moving fixation* is a fixation whose x_pos is smaller than the x_pos of the immediately previous fixation; the reason of using 'leftward' or 'rightward', instead of 'regression' or 'retrace' as in many EM literature is that in the situation of multi-line reading, a 'leftward' or 'rightward' moving saccade is not necessarily a regression or retrace saccade as in single-line text reading).

Third, for each rightward moving saccade, if it starts in the frontal region and its change in horizontal position ('x2_pos' – 'x1_pos') is bigger than the cross line jumping threshold, it is identified as a backward cross line saccade. Similarly, for each rightward moving fixation, if the immediately previous fixation is in the frontal region, and the 'x_pos' distance between the current fixation and the immediately previous fixation is bigger than the cross line jumping threshold, the current fixation is identified as a backward cross line fixation. At the same time, for each leftward moving single saccade or fixation, if its change in horizontal distance is bigger than the cross line jumping threshold, it is identified as a forward cross line saccade or fixation.

If there is a group of saccades or fixations whose accumulated change in horizontal position (for a group of saccades, 'x1_pos' of the first saccade – 'x2_pos' of the last saccade; for a group of fixations, 'x_pos' of the first saccade – 'x_pos' of the last fixation) matches this criterion, the situation is a little bit complex. Here, we use two steps to determine which saccade or fixation in the group is the cross line saccade or fixation. First, if there is one saccade or fixation in the group whose change in horizontal position is bigger than the cross line jumping threshold, that saccade or fixation is identified as the cross line saccade or fixation. Second, if there is no such saccade or fixation in the group, we look for the saccade or fixation that has the biggest change in horizontal position and the biggest change in vertical position (for a saccade, its change in vertical position is 'x2_pos' – 'x1_pos'; for a fixation, its change in vertical position is its 'y_pos' – 'y_pos' of its immediately previous fixation). If the saccades or fixations having the biggest changes in horizontal and vertical positions are the same, this saccade or fixation is identified as the cross line saccade or fixation. If they are not the same, the saccade or fixation having a smaller index (taking place earlier in time) is identified as the cross line saccade or fixation. This setting is based on the assumption that a cross line saccade or fixation should be the earliest saccade or fixation that can induce a big change in either horizontal or vertical position.

Once a cross line saccade or fixation is identified, the cross line jumping threshold and frontal region are updated based on the next or previous line of text.

Fourth, after the information of the last line of text is used for setting the cross line jumping threshold, if there are still saccades or fixations unclassified those saccades or fixations are classified into the last line. Then, if some saccade or fixation shows a big change in vertical position, say 60 pixels (marked by the global variable **Y_RANGE**), we assume that the reading ends after that saccade or fixation, and the saccades or fixations afterward are all classified into line 0. However, if all saccades or fixations are classified before the last line of text is used for setting the cross line jumping threshold, it means that the reading data are incomplete to cover all lines of text. In this situation, the data are flagged as questionable.

Finally, after identifying cross line saccades and fixations, the other saccades and fixations lying between the cross line saccades and fixations are classified into corresponding lines. For fixations, we mark all their lines; for saccades, we only mark cross line saccades.

(4) ***crl_Sac_Fix_Batch*:** for each pair of *_Sac.csv and *_Fix.csv, classify saccades and fixations into different lines of text.

Arguments:

- 'direct': directory of *_Sac.csv and *_Fix.csv and region file, the generated csv files are also stored here.
- 'regfileNameList': list of region files.
- 'ExpType': type of experiment.

Outputs:

For each pair of *_Sac.csv and *_Fix.csv, ***crl_Sac_Fix_Batch*** calls ***crl_Sac_Fix*** to modify the 'line' columns in *_Sac.csv and *_Fix.csv, generate corresponding *_crlSac.csv and *_crlFix.csv, and records questionable data (if any) in log.txt.

Examples:

crl_Sac_Fix('./asc_example', '1950024', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv'], 'RP'): modify the 'line' columns in 1950024_Sac.csv and 1950024_Fix.csv, and generate 1950024_crlSac.csv and 1950024_crlFix.csv for cross line saccades and fixations.

crl_Sac_Fix_Batch('./asc_example', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv'], 'RP'): modify the 'line' columns in each pair of *_Sac.csv and *_Fix.csv files, and generate corresponding *_crlSac.csv and *_crlFix.csv files.

The third set of user functions, ***reccrl_Sac_Fix*** and ***reccrl_Sac_Fix_Batch***, unifies the previous two sets of user functions. They can automatically check whether the relevant ascii and region files exist in the specified folder, and if so, read the ascii file, extract the information of reading data in different trials, record saccades and fixations into *_Sac.csv and *_Fix.csv, identify cross line saccades and fixations and store them into *_crlSac.csv and *_crlFix.csv, and record the information of questionable data (if any) in log.txt. If some of the relevant files do not exist, the function throws warning messages and stops.

(5) ***reccrl_Sac_Fix*:** to read ascii data file and generate csv files storing detected saccades and fixations.

Arguments:

- 'direct': directory of the ascii file and the region files, the generated csv files are also stored there.
- 'datafile': complete name of the ascii data file.
- 'regfileNameList': list of region files.
- 'ExpType': type of experiment.

Outputs:

reccrl_Sac_Fix generates four csv files, two (*_Sac.csv and *_Fix.csv) for detected saccades and fixations classified into different lines of the text, and two (*_crlSac.csv and *_crlFix.csv) for cross line saccades and fixations. It may also generate a log file to store the information of questionable data (if any).

(6) ***reccrl_Sac_Fix_Batch*:** for each ascii file, generate csv files storing detected saccades and fixations.

Arguments:

- 'direct': directory of the ascii data files and the region files, the generated csv files are also stored there.
- 'regfileNameList': list of region files.
- 'ExpType': type of experiment.

Outputs:

reccrl_Sac_Fix_Batch calls ***reccrl_Sac_Fix*** to generates relevant csv files for each ascii data file, and probably a log file storing the information of questionable data (if any).

Example:

reccrl_Sac_Fix('./asc_example', '1950024.asc', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv'], 'RP'): generate 1950024_Sac.csv, 1950024_Fix.csv, 1950024_crlSac.csv, 1950024_crlFix.csv, based on the ascii file 1950024.asc.

reccrl_Sac_Fix_Batch('./asc_example', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv'], 'RP'): generate the four relevant csv files based on each of the ascii files in the subfolder. Each ascii file records the reading data of a particular subject reading the same three stories in three trials.

Functions drawing saccades and fixations on bitmaps: The EDF file created by the Eye Link eye-tracker can only show saccades and fixations on an empty background, which cannot clearly reflect at which lines of text these saccades and fixations occur. To better visualize the experimental data, we design user functions in this section to show saccades and fixations upon actual bitmaps that subjects read in the experiment. Two user functions in this section, ***draw_Sac_Fix*** and ***draw_Sac_Fix_Batch***, help draw relevant figures based on a particular subject's data or relevant figures based on many subjects' data in a batching fashion. These

functions require the bitmaps, region files, and csv files generated by functions in the other sections. If these files do not exist in the specified folder, the functions throw warning messages and stop.

(1) ***draw_Sac_Fix*:** to draw either all saccades and fixations and/or cross line saccades and fixations of a particular subject on corresponding bitmaps.

Arguments:

- 'direct': directory of the bitmaps, region files, and csv files of a subject, the generated figures are also stored there.
- 'subj': subject ID.
- 'regfileNameList': list of region files.
- 'bitmapNameList': list of bitmaps used in each trial of the experiment.
- 'medthod': if 'method' is 'ALL', the function draws two figures, one showing all saccades and all fixations during normal reading (fixations taking place after the ending of reading are not shown), and the other showing only cross line saccades and fixations; if 'method' is 'Sac', it draws two figures respectively showing all saccades and cross line saccades; and if 'method' is 'Fix', it draws two figures respectively showing all fixations during normal reading and cross line fixations.
- 'PNGmethod': whether draw saccades and fixations based on existing bitmaps or re-generate bitmaps from region files. If 'PNGmethod' is 0, saccades and fixations are drawn on existing bitmaps; if 'PNGmethod' is 1, the function regenerates bitmaps based on region files, and then, draw saccades and fixations on these bitmaps. Default value is 0. Due to different monitor setting, regenerated bitmaps may be slightly different from the ones used in the experiment.

Outputs:

If 'method' is 'ALL', ***draw_Sac_Fix*** generates two figures (*_FixSac_trial#.png and *_crlFixSac_trial#.png, '*' is subject ID, and '#' is trial ID); if 'method' is 'Sac', it generates two figures (*_Sac_trial#.png and *_Fix_trial#.png); if 'method' is 'Fix', it generates two figures (*_Fix_trial#.png and *_crlFix_trial#.png). In these figures, single/multi-line text used in the trials is shown as the background; saccades are denoted by solid lines, among which rightward moving saccades are in green and leftward in red; fixations are denoted by blue circles, whose radius are determined by durations of fixations, and green numbers near circles show durations in milliseconds. We use the global variable **MAX_FIXATION_RADIUS** to mark the radius of maximum fixation in the whole trial, the default value is 30 pixels.

(2) ***draw_Sac_Fix_Batch*:** for the data files of each subject, call ***draw_Sac_Fix*** to draw relevant figures.

Arguments:

- 'direct', 'regfileN ameList', 'bitmapNameList', 'method', 'PNGmetghod': all these arguments are identical to those of ***draw_Sac_Fix***.

Outputs:

draw_Sac_Fix_Batch calls draw_Sac_Fix to read the data files of each subject, and generate relevant figures.

Example:

draw_Sac_Fix('./asc_example', '1950024', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv'], ['story01.png', 'story02.png', 'story03.png'], 'ALL'): based on the data files of subject 1950024, draw 1950024_FixSac_trial0.png, 1950024_FixSac_trial1.png, 1950024_FixSac_trial2.png, and 1950024_crlFixSac_trial0.png, 1950024_crlFixSac_trial1.png, 1950024_crlFixSac_trial2.png. To successfully use this function, users need to move the bitmaps (story01.png, story02.png, story03.png) and region files (story01.region.csv, story02.region.csv, story03.region.sv) from the subfolder bitmap_regfile to the subfolder asc_example.

draw_Sac_Fix_Batch('./asc_example', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv'], ['story01.png', 'story02.png', 'story03.png'], 'ALL'): generate the same figures as the previous example. If there are data files of other subjects, the function also generates the corresponding figures of those subjects.

Apart from these functions, we also define another user function, ***draw_blinks***, to show histograms of subjects' blinks in the same trials. It has two arguments: 'direct' (directory of csv files of all subjects) and 'trialNum' (number of trials in the experiment), and shows a number (equal to 'trialNum') of histogram of blinks. It aims to help users to determine the average numbers of blinks subjects normally have in different trials, which serve as a criterion for removing data containing too many blinks.

Functions calculating EM measures: Based on the fixation and saccade data classified into different lines of text, functions in this section help calculate many widely-adopted EM measures. These functions require the region files storing position information of words in the single/multi-line text and the csv files storing classified fixations and saccades in different lines of text, both of which are generated by functions in the other sections. Based on previous literature of EM, we focus on three groups of EM measures, including eight first-pass fixation and regression measures, four regression path measures, and two second-pass fixation measures. Apart from these measures around particular words, we also calculate four whole-text-based measures. All the measures are calculated by appropriate helper functions, and the values are stored in corresponding columns of result csv files. These measures are sufficient for follow-up analysis of reading patterns. There are two user functions, ***cal_EMF*** and ***cal_EMF_Batch***, which calculate either the results of one subject or those of many subjects.

(1) ***cal_EMF*:** to calculate the EM measures based on the data of each trial of one subject, and store the results in csv files.

Arguments:

- 'direct': directory of *_Sac.csv and *_Fix.csv of a subject, and the region files, the generated result csv files are also stored there.
- 'subj': subject ID, identical to '*' of the csv files.
- 'regfileNameList': list of the region files corresponding to the data files.

Outputs:

Based on the number of trials, ***cal_EMF*** generates a number of csv files (*_EMF_trial#.csv, '*' is the subject ID and '#' is the trail index starting at 0) respectively storing the values of the EM measures based on the csv files in each trial.

*_EMF_trial#.csv has 32 columns, some of which record the general information of the reading data (extracted from *_Sac.csv or *_Fix.csv), some of which record word information of the single/multi-line text (extracted from the region files), and the other of which record the values of EM measures (calculated by corresponding helper functions):

- 'subj', 'trial_id', 'trial_type', 'tdur', 'blinks', 'eye': identical to those columns in *_Sac.csv or *_Fix.csv.
- 'tffixos': total offset of the first pass fixation of each word from the beginning of the first sentence of the text (whole-text ME measure).
- 'tffixurt': total duration of the first pass fixation of each word in the text (whole-text EM measure).
- 'tffixcnt': total number of valid fixations in the trial whole-text EM measure (whole-text EM measure).
- 'tregrcnt': total number of *regression saccades* (a saccade is a regression saccade if it starts at one word region in the text and ends at an earlier word region) in the trial (whole-text EM measure).
- 'region': numerical index of each word in the text, starting at 1; identical to the 'WordID' column in the region file.
- 'reglen': length of the region, including the added space before the word and punctuation afterward; identical to the 'length' column in the region file/
- 'word', 'line', 'x1_pos', 'x2_pos': the actual word in each region (including the added space before the word and punctuation afterward), the line index of this word in the whole text, and the left and right boundaries in pixels of this word. All this information can be extracted from the region file.
- 'mod_x1', 'mod_x2': for words in the middle of a line, 'mod_x1' and 'mod_x2' are identical to 'x1_pos' and 'x2_pos'; for words at the beginning of each line, 'mod_x1' is slightly smaller than 'x1_pos'; for words at the ending of each line, 'mod_x2' is slightly bigger than 'x2_pos'. We use 'mod_x1' and 'mod_x2' to determine whether a saccade or fixation lies in a word region.
- 'fpurt': first-pass fixation time. It is the sum of the durations of one or more first-pass fixations falling into the word region. We only record fixation of 50 ms or longer by default; any shorter fixations will undergo the merging operation. If there is no first pass fixation lying within the word region, 'fpurt' is 'NA' (first-pass fixation measure).
- 'fpcount': number of first-pass fixations falling into the word region. If there is no first-pass fixation in the word region, 'fpcount' is 'NA' (first-pass fixation measure).
- 'fpregres': whether (1) or not (0) there is a first-pass regression starting from the current word region (first-pass regression measure).
- 'fpregreg': word region where the first-pass regression ends. If there is no first-pass regression ('fpregres' is 0), 'fpregreg' is 0.

- ‘fpregchr’: offset in characters in the word region where the first-pass regression ends. If there is no first-pass regression (‘fpregres’ is 0), ‘fpregchr’ is set to a value large enough to be out of boundaries of any possible string (in the current version, it is set as the total number of characters of the text).
- ‘ffos’: offset in characters of the first first-pass fixation in the word region from the first character of the region (first-pass fixation measure).
- ‘ffixurt’: duration of the first first-pass fixation in the word region. If there is no first-pass fixation, ‘ffixurt’ is ‘NA’ (first-pass fixation measure).
- ‘spilover’: duration of the first fixation falling beyond (either left or right) the word region. If there is no first-pass fixation in the word region, ‘spilover’ is ‘NA’ (first-pass fixation measure).
- ‘rpurt’: sum of durations of all fixations in the regression path. The regression path starts from the first fixation falling into the current word region and ends at the first fixation falling into the immediately next word region. If there is a first-pass regression (‘fpregres’ is 1), the regression path includes the fixations inside the current word region and those outside the current word region but falling into only the word regions before the current word region. If there is no first-pass regression (‘fpregres’ is 0), ‘rpurt’ equals to ‘fpurt’ (regression path measure).
- ‘rpcount’: number of fixations in the regression path (regression path measure).
- ‘rpregreg’: the smallest word region visited by the regression path. If there is no regression path (‘fpregres’ is 0) or there is no first-pass fixation (‘fpcount’ is ‘NA’), ‘rpregreg’ is 0 (regression path measure).
- ‘rpregchr’: offset in characters in the smallest word region visited by the regression path. If there is no first-pass fixation (‘fpcount’ is ‘NA’), ‘rpregchr’ is set to a value large enough to be out of boundaries of any possible string (in the current version, it is set as the total number of characters of the text) (regression path measure).
- ‘spurt’: second-pass fixation time. It is the sum of durations of all fixations falling again into the current word region after the first-pass reading. If there is no second-pass fixation, ‘spurt’ is ‘NA’ (second-pass fixation measure).
- ‘spcount’: number of second-pass fixations. If there is no second-pass fixation, ‘spcount’ is ‘NA’ (second-pass fixation measure).

Algorithms:

cal_EMF first checks whether the relevant csv files (*_Sac.csv, *_Fix.csv, and the region files) exist in the specified subfolder; if not, the function throws warning messages to the screen and stops. If the files exist, ***cal_EMF*** reads these files, assigns fixations to particular words in the text based on the position information of the words in the region files. Then, for each word region, the function calls relevant helper functions to calculate the first-pass fixation and regression measures, regression path measures, and second-pass fixation measures. After calculating these word-based measures, the function sums up the whole-text-based measures. Finally, the function stores the results into csv files.

Compared with single-line text reading, there are several difficulties in calculating the EM measures in multi-line text reading. For example, to check which word region a fixation falls into, we need to refer to 'x_pos' of the fixation, 'x1_pos' and 'x2_pos' of the words in the text, and 'line' in which the fixation is classified into. In addition, when subjects are reading multi-line text, at line boundaries (beginning or ending), some of their fixations may 'overshoot', lying slightly out of the regions of the first or the last words of different lines. Therefore, we need to use 'mod_x1' and 'mod_x2' as the region boundaries of beginning and ending words of different lines. The difference between 'mod_x1' and 'x1_pos' and that between 'mod_x2' and 'x2_pos' is one character (marked by the global variable **ADD_TIMES**). Furthermore, in multi-line reading, physical position of a word region is insufficient to indicate whether this region is before or after the target word region. Therefore, we use 'region', a numerical index of each word in the text, to clarify regression probability and regression path.

(2) ***cal_EMF_Batch*:** to calculate the EM measures based on the data of each trial of many subjects, and store the results in corresponding csv files.

Arguments:

- 'direct': directory of *_Sac.csv and *_Fix.csv of many subjects, and the region files, the generated result csv files are also stored there.
- 'regfileNameList': list of the region files corresponding to the data files.

Outputs:

cal_EMF_Batch first lists all subject IDs that have csv data files. Then, for each subject's data, it calls ***cal_EMF*** to calculate the EM measures and store the results into corresponding csv files.

Examples:

cal_EMF('./asc_example', '1950024', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv']): calculate the EM measures based on 1950024_Fix.csv and 1950024_Sac.csv, and store the results into 1950024_EMF_trial0.csv, 1950024_EMF_trial1.csv, and 1950024_EMF_trial2.csv.

cal_EMF_Batch('./asc_example', ['story01.region.csv', 'story02.region.csv', 'story03.region.csv']): calculate the EM measures based on the result csv files of many subjects, and store the results into corresponding *_EMF_trial#.csv.

Summary and future directions: The EMF_RP package provides flexible ways of generating bitmaps showing multi-line text and region files recording position information of words in bitmaps. Based on observations of online reading patterns, the package designs reliable algorithms to classify detected saccades and fixations into different lines of text, which help overcome the greatest difficulty in this line of research. Furthermore, the package calculates a rich number of EM measures widely used in previous EM literature, thus allowing analysis of reading patterns from different perspectives (e.g., first-pass fixation, regression, and second-pass fixation). All these characteristics make the package useful to a wide range of EM reading research.

There are several aspects in the current package that deserve improvement.

Some aspects concern theoretical issues about reading behavior, and ask for more experimental research to resolve. For example, due to individual differences, some subjects may show patterns that cannot be accurately captured by the current functions in classifying saccades and fixations, especially cross line reading patterns, and their data may be accidently flagged questionable by the functions. With more experimental studies in multi-line text reading, we can update the relevant algorithms to automatically and efficiently capture various reading patterns and accurately classify detected saccades and fixations into different lines of text. In addition, when a cross line reading is achieved by a group of saccades and fixations, rather than a single big jump, we estimate the fixation or saccade having the biggest change in either horizontal or vertical position among the group as the cross line fixation or saccade. However, such criterion may not always hold, and we have not clearly understood what is the cognitive process involved during such slow cross line reading.

Other aspects concern practical issues, and can be improved by advanced techniques. For example, in the user functions drawing saccades and fixations upon bitmaps, apart from still images showing all saccades and fixations, animations appear to be more effective to illustrate the whole reading process. Meanwhile, if some experiments also record the oral reading of subjects, animations can be played together with audio files to reveal the online reading patterns. Apart from the animation package in Python, 'SimpleGUI' package can be used to achieve the task of creating accurately timed animations based on the starting and ending time of fixations and saccades. In addition, generally speaking, the total number of words in multi-line text (over 100 and more) is much bigger than that of single-line text (usually less than 20). Therefore, calculating EM measures in multi-line text remains much slower than calculating the measures in single-line text (as shown in the functions of the current package). Increasing the efficiency of the algorithms (e.g., reduce the number of looping) is needed to accelerate the calculation process.

All these aspects will be addressed in the future version of the package.