**Documentation of EMF\_RP**

Tao Gong, Jonathan Gordils, David Braze

*Haskins Laboratories, New Haven, CT, USA*

Date: 2015/07/08

**EMF\_RP** (available at GitHub: github.com/gtojty/emf\_rp) is a general purpose, stand-alone software package designed to assist with the chores of implementing single/multi-line text reading experiments and analyzing collected eye-movement data. The package consists of a set 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 eye-movement measures.

Developed using Python 2.7, the package depends on some additional Python packages, including *os*, *fnmatch*, *re*, *csv*, *codecs*, *pandas*, *numpy*, *PIL*, *turtle*, *matplotlib*, and *winsound*, all widely available and pre-installed with popular distributions of Python in Windows and Linux (e.g., WinPython, Anaconda, Maverick, etc.).

Functions in the package can be classified as *user functions*, which are intended to be called directly by users, and *helper functions*, which are mainly called by user functions to implement various tasks. Names of helper functions start with ‘**\_**’. To ease processing, we also provide the batch versions of some user functions that allow routinely processing the data of many subjects in a batching fashion. Names of such batching functions end with ‘**\_b**’.

The whole package consists of five sections:

1. Functions for generating bitmaps of single/multi-line texts and region files of word position information in texts.
2. Functions for extracting saccades, fixations, and other information from ASCII data files transformed from EyeLink EDF files.
3. Functions for classifying saccades and fixations into different lines of text, assigning fixations into different word regions, and identifying cross-line saccades and fixations.
4. Functions for visualizing saccades and fixations on bitmaps.
5. Functions for calculating popular eye-movement measures.

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

**Paragraph reading experiment:** The experiment aims to analyze individual reading patterns during paragraph reading. It consisted of three trials. In each trial, subjects were asked to read overtly a short story shown in one paragraph on the screen, and their eye movements were recorded by EyeLink 1000, and their oral reading were recorded by microphone. The three stories are recorded in the three text files in the subfolder ‘bitmap\_regfile’, story01.txt, story02.txt, story03.txt; line breaks specified in text files are respected in created bitmaps. The commenting line (starting with ‘#’) in each text shows meta-data for the story (title and number of words). Passages.txt contains all three stories in one file.

Using the functions in EMF\_RP, users can generate bitmaps of texts, along with region files in csv format. Each region file specifies regions of interest (ROIs) within each bitmap; conventionally, one ROI for each word in the story. Bitmaps are used as stimulus items during eye-movement data collection. Region files are used post-data collection to create region-wise (e.g., word-wise) summaries of eye movements. Additional functions in the package facilitate visualization and classification of saccades and fixations recorded in each trial.

1. **Functions for generating bitmaps and region files**

The screen-based coordinate system for generated bitmaps and region files place the origin in the upper left corner. We develop two user functions for creating bitmaps and region files: **Praster** is the main workhorse, and **Gen\_Bitmap\_RegFile** is a convenience wrapper around **Praster**.

**User functions:**

1. **Praster:** each call of it createsa single bitmap of single/multi-line text according to specified parameters, along with attendant region file. Bitmaps are used as stimuli in eye-tracking based text reading experiments, and region files are for data summary.

*Arguments:*

* ‘direct’: directory storing text files and generated bitmaps and region files.
* ‘fontpath’: fully qualified path to a font file. It must be a font accessible to Python.
* ‘langType’: language of shown texts. It can be ‘English’, or other Latin languages (e.g., ‘German’, ‘French’, ‘Italian’, etc.), or some non-Latin languages (e.g., ‘Chinese’, ‘Korean’, or ‘Japanese’).
* ‘codeMethod’: encoding method of texts; default value is ‘utf\_8’. For displaying English, French, German, and other Latin like languages, ‘utf\_8’ is an efficient coding method. However, for displaying Chinese, Korean, or Japanese, specific codeMethod is required. As an example, we use ‘simsun.ttc’ for displaying Chinese and Japanese, and ‘batang.ttc’ for displaying Korean. The corresponding ttc font files are also put in the example folder.
* ‘text’: a list, containing one or more lines of text 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.’]. Text is encoded in unicode.
* ‘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. It corresponds to the monitor resolution to be used for displaying bitmaps during data collection.
* ‘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 for text in pixels; default value is 86.
* ‘tmargin’: top margin for text in pixels; default value is 86. ‘lmargin’ and ‘tmargin’ determine the starting position of a multi-line text. As for a single-line text, its starting position is at the center of the bitmap (‘lmargin’, ‘y’/2.0).
* ‘linespace’: line spacing in pixels; default value is 43. It is likely to be adjusted if font size is set to a value other than its default value.
* ‘fht’: font height (maximum vertical distance between the highest and lowest painted pixel of each character shown in a particular font) in pixels; default value is 18.
* ‘fwd’: font width in pixels, default value is None. ‘fwd’ takes precedence over ‘fht’.
* ‘bbox’: whether to draw a minimal bounding box, corresponding to a region of interest around each word in the bitmap; default value is False. It is used for testing purposes. Coordinates of the bounding box are stored in region file.
* ‘bbox\_big’; whether to draw an expanded bounding box around each word in the bitmap. Such bounding boxes are expanded vertically per the argument ‘addspace’ below; default value is False (not drawing). It is used for testing purposes. Coordinates of the bounding box are also stored in region file.
* ‘ID’: unique identity for the stimulus item, used to set the filenames of generated bitmap and region file; default value is ‘test’, resulting in a bitmap test.png and a region file test.region.csv, both in the specified folder (‘direct’).
* ‘addspace’: the amount of vertical padding (in pixels) to be added above the top and below the bottom of each word when creating the ‘big’ ROIs; default value is 18.
* ‘log’: whether to record intermediate results in a log file, default value is False. It is used for testing purposes.

*Outputs:*

**Praster** creates two files: a bitmap in PNG format (e.g., ID.png) and a CSV region file (e.g., ID.region.csv). The bitmap is used as stimulus during data collection and background for visualizing saccades and fixations. If ‘regfile’ is True (by default), the region file created is used for generating regional summaries of gaze patterns. Region information of each word in the displayed text is stored as one line of the region file. Such information includes values from 16 columns:

* + ‘Name’: identical to ‘ID’; conventionally, a unique ID for each trial or stimulus item.
  + ‘Language’: type of language.
  + ‘WordID’: numerical index of each word in the text, starting at 1.
  + ‘Word’: individual word in the text, including, if present, the space(s) before the word and punctuation behind.
  + ‘length’: length of a word in characters, including attendant space(s) before the word and punctuation behind.
  + ‘height’: height of a word in pixels; the maximum vertical distance between the highest and lowest painted pixel of each character shown in a particular font.
  + ‘baseline’: baseline position in pixels of each line of words, words on the same line have identical baseline values.
  + ‘line\_no’: numerical index of each line of words, starting at 1.
  + ‘x1\_pos’, ‘y1\_pos’: top left corner of the minimal bounding box containing a word (including attendant space(s) before the word and punctuation behind), in pixels.
  + ‘x2\_pos’, ‘y2\_pos’: bottom right corner of the minimal bounding box containing a word, in pixels.
  + ‘b\_x1’, ‘b\_y1’: top left corner of the expanded bounding box for a word (including attendant space(s) before the word and punctuation behind), in pixels.
  + ‘b\_x2’, ‘b\_y2’: bottom right corner of the expanded bounding box containing a word, in pixels. Among the position values, ‘b\_x1’ and ‘b\_x2’ of the expanded bounding box are identical to ‘x1\_pos’ and ‘x2\_pos’ of the minimal bounding box. However, unlike the minimal bounding box, the top and bottom values of the expanded bounding box (‘b\_y1’ and ‘b\_y2’) are aligned by lines; all words on the same line have the same ‘b\_y1’ (minimum values of ‘y1\_pos’ of all minimal bounding boxes of these words – ‘addspace’) and ‘b\_y2’ (maximum values of ‘y2\_pos’ of these minimal boxes + ‘addspace’). If ‘addspace’ is 0 (not adding extra space above and below each word), ‘b\_y1’ and ‘b\_y2’ of the expanded bounding box correspond to the top and bottom boundaries of all words on the same line.

*Algorithm:*

**Praster** reads the single/multi-line text in ‘text’. Spaces are associated with the word that they precede, and punctuation is associated with the word that it follows. **Praster** calculates the height and length of each word (including associated spaces and punctuation), and the maximum height for all words on the same line. Then, it sets up the baseline according to ‘tmargin’, ‘lmargin’, and ‘linespace’. Words on a given line have the same baseline. Bounding box (region) position values are calculated for each word and recorded in the region file. If font size or ‘tmargin’ and ‘dim’ values are such that any words overflow the bitmap boundaries, it throws a warning message to the screen and stops.

The function can generate single-/multi-line text of English, German, French, Dutch, Italian, Spanish, Greek, as well as languages in East Asia, such as Chinese, Korean, and Japanese. Displaying the latter three languages require specific font files as input, and these languages have different print styles from those of the other Latin like languages. For example, in English, words are regularly separated by space. However, in Chinese, there is no space between words; in Japanese or Korean, it is optional to have a space between words, but there is no space between a Chinese-style punctuation (e.g., ‘，’, ‘。’, or ‘、’) and the following word.

***Note:* due to mismatch between Windows and Linux versions of Python and internal bugs of the Image package in Python, the generated region files are slightly different in ‘y1\_pos’ and ‘y2\_pos’ of certain words; nonetheless, ‘x1\_pos’ and ‘x2\_pos’ are accurate for calculating eye-movement measures.**

1. **Gen\_Bitmap\_RegFile:** aconvenience wrapper around **Praster**, primarily to simplify the task of generating many bitmaps and region files at once. 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 accessible to Python, e.g., ‘LiberationMono’, a mono-space font. Mono-space font is commonly used in text reading experiments, due to the ease of handling word positions.
* ‘textFileNameList’: list of names of the text files for generating bitmaps and region files. Based on the argument ‘genmethod’ below, the list can contain no, one, or several text files.
* ‘genmethod’: method to generate bitmaps and region files. If ‘genmethod’ is 0, it is used for testing purposes; if ‘genmethod’ is 1, the function reads one text file in ‘textFileNameList’, which contains one or more single/multi-line texts; if ‘genmethod’ is 2, it reads one or more text files in ‘textFileNameList’, each of which contains only one single/multi-line text, here, ‘textFileNameList’ can also be empty, and in this case, it reads all text files in ‘direct’; default value is 2.
* Additional arguments passed to **Praster** (‘langType’, ‘codeMethod’, ‘dim’, ‘fg’, ‘bg’, ‘lmargin’, ‘tmargin’, ‘linespace’, ‘fht’, ‘fwd’, ‘bbox’, ‘bbox\_big’, ‘addspace’, and ‘log’): see **Praster**.

*Outputs:*

**Gen\_Bitmap\_RegFile** generates a bitmap and a region file for each text file in ‘textFileNameList’.

*Algorithm:*

**Gen\_Bitmap\_RegFile**provides two ways of handling text files. If ‘genmethods’ is 1, it reads one text file in ‘textFileNameList’. The text file can contain one or more single/multi-line texts, and it generates a bitmap and a region file for each text, whose names are story01.png, story01.region.csv, story02.png, story02.region.csv, etc. If ‘genmethods’ is 2, it reads one or more text files in ‘textFileNameList’, each containing one single/multi-line text, and generates corresponding bitmaps and region files, whose names are determined by the names of the text files. If some text files are missing in the specified directory (‘direct’), it throws a warning message and stops. Here, if ‘textFileNameList’ is empty, it reads all text files in ‘direct’, each containing one single/multi-line text, and generates corresponding bitmaps and region files, whose names are determined by the names of the text files.

In a text file, users can separate the text into different lines using the delimiter ‘\r\n’; in a text file of many single/multi-line texts, users can use the delimiter ‘\r\n\r\n’ (an empty line) to separate different texts. Users can insert comments (starting with ‘#’) to record meta-data of the text (e.g., paragraph title and number of words), which are squeezed out by the function.

For English and other Latin-like texts, words are separated by spaces. However, for Chinese, since there are no spaces between words, to let the region files to correctly record each word’s boundary, users can use ‘|’ to mark word boundaries in the texts, and such separator will be removed in generated bitmaps. For Korean and Japanese, since spaces between words are optional, users can freely insert spaces between words.

**Examples:**

1. **Praster(‘.’, fontpath, ‘English’, text=[u‘The quick brown fox jumps over the lazy dog.’]):** generate a bitmap (test.png) displaying the sentence in the middle of it and a region file (test.region.csv) recording the position information of each word in the sentence (fontpath is set according to the system used, see examples of using Gen\_Bitmap\_RegFile below). The results are stored in the current folder (‘.’).
2. **Praster(‘./example’, fontpath, ‘English’, 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 at the top left corner of the screen (according to ‘lmargin’ and ‘tmargin’, here, using the default values), and a region file storing the position information of each word in the two sentences. These results are stored in the pre-existent folder ‘example’.
3. **Praster(‘./bitmap\_regfile’, ‘simsun.ttc’, ‘Chinese’, text = [u’‘我们|爱|你！’]):** generate a bitmap showing the Chinese sentence, starting in the center of the screen, and a region file storing the position information of each word (separated by ‘|’) in the sentence. These results are stored in the folder ‘bitmap\_regfile’.
4. **Gen\_Bitmap\_RegFile(‘./bitmap\_regfile’, ‘LiberationMono’, ‘English’, [‘Passages.txt’], 1):** read Passages.txt in the subfolder bitmap\_regfile. The text file contains three multi-line texts. It 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, it also summaries the number of lines in each text displayed in the bitmaps.
5. **Gen\_Bitmap\_RegFile(‘./bitmap\_regfile’, ‘LiberationMono’, ‘English’, [‘story01.txt’, ‘story02.txt’, ‘story03.txt’], 2):** read three text files (story01.txt, story02.txt, story03.txt) in the subfolder ‘bitmap\_regfile’, and generates the same bitmaps and region files as in the previous example. If ‘textFileNameList’ is [], it reads all text files in the subfolder, and generates the same bitmaps and region files as in the previous example.
6. **Gen\_Bitmap\_RegFile(‘./bitmap\_regfile’, ‘simsun.ttc’, ‘Chinese’, [‘ChineseStory.txt’], 2):** read the text file ChineseStory.txt in the subfolder ‘bitmap\_regfile’, and generates the corresponding bitmap and region file as in the previous example. Here, the encoding method has to be specified using the particular font file ‘simsun.ttc’, which has to be in the same folder with the text file. If ‘textFileNameList’ is [], it reads all text files in the subfolder, and generates the same bitmaps and region files as in the previous example.
7. **Gen\_Bitmap\_RegFile(‘./bitmap\_regfile’, ‘simsun.ttc’, ‘Japanese’, [‘JapaneseStory.txt’], 2):** read the Japanese text file and generates the corresponding bitmap and region file as in the previous example.
8. **Gen\_Bitmap\_RegFile(‘./bitmap\_regfile’, ‘batang.ttc’, ‘Korean’, [‘KoreanStory.txt’], 2):** read the Korean text file and generates the corresponding bitmap and region file as in the previous example.

1. **Functions for extracting saccades, fixations, and other information in data files**

Functions in this section read eye-tracker generated data files to recover eye-movement events (blinks, saccades, fixations, etc.) as well as other session and trial information, and store all this information in useful data structures for subsequent processing.

The functions require not only the region files created in the previous section but also the eye-tracker generated data file. The data files used in the EMF\_RP package are the ASCII data files converted from EyeLink EDF files by way of SR Research’s ‘edf2asc.exe’ utility. To be parsed properly by the functions, the ASCII data files must contain certain ‘flags’ or ‘events’ (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, ‘START’ for the starting time of each trial, ‘END’ for the ending time of each trial, ‘ARECSTART’ for the starting time of recording, ‘ARECSTOP’ for the ending time of recording, etc. Please refer to the EyeLink manual for details of EDF/ASCII file content; users focusing only on extracted results do not need to understand such detail). We provide helper functions to extract this information from data files and store it in a Python Numpy data frame for subsequent processing. In the subfolder ‘asc\_example’, we put three example ASCII data files collected in the paragraph reading experiment. Each data file records the eye-movement data of one subject (the name of the file is the subject ID) reading the three stories shown in the bitmaps stored in the subfolder ‘bitmap\_regfile’. 1950024.asc contains the reading data of the left eye; 1950088.asc contains the reading data of the right eye; and 1950099.asc contains the reading data of both eyes. To properly use the functions in this section, users need to put the ASCII files and relevant region files in the same folder. Apart from ASCII data files, users can write their own user functions to read data files in other eye-tracker formats.

The functions in this section extract raw data in ASCII data files (e.g., basic information of eye-tracker setting and physical locations and durations of detected saccades and fixations) and put them into user-friendly data frames. During extraction of fixations, we provide the option of lumping (merging) fixations shorter than 50 ms. We also provide user functions to store such data frames into csv files as saccade and fixation reports. Users can write their own functions to manipulate these data frames and generated reports. For example, during the paragraph reading experiment, before reading the paragraph line by line, some subjects may browse the whole paragraph, and the saccades and fixations during such browsing need to be removed; without clearly grasping the instructions, some subjects may stop during the reading and interact with the experimenter, and the saccades and fixations during such interruptions need to be removed. Considering that there could be many ASCII data files for extracting, we provide a batch user function allowing routinely processing many ASCII data files at once.

**User functions:**

1. **read\_SRRasc:** read an ASCII data file and returns extracted saccades and fixations in data frames.

*Arguments:*

* ‘direct’: directory of the ASCII file and region file(s).
* ‘datafile’: name of the ASCII file, e.g., ‘1950024.asc’.
* ‘regionfileNameList’: list of names of region files. If the ASCII data file contains the data of many trials, the corresponding region files should be sorted accordingly. As in the paragraph reading experiment, the ASCII file stores the data of three trials, and ‘regionfileNameList’ is [‘story01.region.csv’, ‘story02.region.csv’, ‘story03.region.csv’]. If ‘regfileNameList’ is empty, the function reads all \*.region.csv files in ‘direct’ as region files.
* ‘ExpType’: type of experiment, it can be set as ‘RP’, meaning ‘reading paragraph’, in the paragraph reading experiment.
* ‘rec\_lastFix’: whether set the last fixation in each trial as a valid fixation; default value is False (not setting). Conventionally, the last fixation in each trial is not treated as a valid fixation.
* ‘lump\_Fix’: whether conduct lumping operation (see more description in *Algorithm*) on extracted fixations; default value is True (conducting lumping operation). If ‘lump\_Fix’ is True, the following three arguments will be used.
* ‘ln’: threshold duration in milliseconds for a fixation to be lumped; default value is 50. Fixations shorter than ‘ln’ are subject to lumping with adjacent and near enough (determined by the argument ‘zn’) fixations.
* ‘zn’: threshold distance in pixels between two fixations for lumping; default value is 50, roughly 1.5 character. Fixations closer than ‘zn’ are subject to the lumping operation.
* ‘mn’: minimum legal fixation duration; default value is 50. Any fixation (with or without lumping) shorter than this value is marked as an invalid fixation.

*Outputs:*

**read\_SRRasc**creates two data frames (SacDF and FixDF), respectively storing saccades and fixations from the ASCII data file, the information of each saccade and fixation is stored in different columns of the data frames.

SacDF has 24 columns:

* + ‘subj’: subject ID for each session.
  + ‘trial\_id’: numerical index of trials, starting from 0. It is the unique identifier for each trial. Note that in ASC data file, the trial index starting from 0, corresponding to trial 1.
  + ‘trial\_type’: trial type, a unique label for the trial condition. In the paragraph reading 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 and configuration of the eye-tracker.
  + ‘script’: name of script used in the experiment.
  + ‘sessdate’: date of the experiment.
  + ‘srcfile’: EDF file used for generating the ASCII data file.
  + ‘trialstart’: starting time of each trial.
  + ‘trialend’: ending time of each trial.
  + ‘tdur’: total duration of each trial, equal to ‘trialend’ – ‘trialstart’.
  + ‘recstart’: starting time of recording. This value is important for aligning the recorded sound file and the recorded eye movements (see section 4.2 for generating animations of showing fixations and playing sounds).
  + ‘recend’: ending time of recording.
  + ‘blinks’: total number of blinks detected in each trial.
  + ‘eye’: which eye the EM data are recorded from, e.g., ‘L’ (left eye), ‘R’ (right eye), or ‘LR’ (both eyes).
  + ‘start\_time’: starting time of the saccade in milliseconds.
  + ‘end\_time’: ending time of the saccade in milliseconds.
  + ‘duration’: duration of the saccade, equal to ‘end\_time’ – ‘start\_time’ + 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\_no’: on which line of the text the saccade occurs. Before the saccade is classified into a line of the text, it is set as ‘NaN’ (missing value). Functions in the next section help classify the saccade into an appropriate line.

FixDF contains 23 columns:

* + ‘subj’, ‘trial\_id’, ‘trial\_type’, ‘sampfreq’, ‘script’, ‘sessdate’, ‘srcfile’, ‘trialstart’, ‘trialend’, ‘tdur’, ‘recstart’, ‘recend’, ‘blinks’, ‘eye’: same as those of SacDF. Since the follow-up analysis may focus on either saccades or fixations, both data frames share some basic information.
  + ‘start\_time’, ‘end\_time’: starting and ending times 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’: the radius of the pupil recorded during the fixation.
  + ‘valid’: whether it is a valid (‘yes’) or invalid (‘no’) fixation.
  + ‘line\_no’: on which line of the text the fixation occurs. Before the fixation is classified into a line of the text, it is set as ‘NaN’ (missing value). Functions in the next section help classify the fixation into an appropriate line.
  + ‘region\_no’: on which word region of the text the fixation occurs. Before the fixation is classified into a line of the text, it is set as ‘NaN’ (missing value). Functions in the next section help classify the fixation into an appropriate word region.

*Algorithm:*

**read\_SRRasc** first checks whether the ASCII data file and associated region files exist in the specified directory (‘direct’). If not, it throws a warning message and stops. If the necessary files do exist, **read\_SRRasc** reads the ASCII file and region files, extracts relevant information from the ASCII data file, and creates two data frames respectively storing the saccades and fixations for all trials found in the data file. As for the data file storing saccades and fixations of both eyes, in each trial, it first extracts and stores saccades and fixations of the left eye, and then the right eye.

An optional operation during extraction of fixations is lumping (merging) fixations. To apply the operation, users can set ‘lump\_Fix’ as True. In general, a fixation shorter than 50 ms (marked by the argument ‘ln’) is deemed short and unreliable for follow-up analysis.

The lumping operation proceeds as follows. First, for each short fixation, the operation checks its previous and next fixations, and if those fixations are also short, they are merged 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 of those fixations. Second, if these fixations are not short or the duration of the merged one is still short, the operation keeps checking the distance (in terms of ‘x\_pos’) between this fixation and any other ones physically close to it. If the distance between it and one of its neighbors is smaller than 50 pixels (marked by the argument ‘zn’), the two fixations are merged into one. If the distances between it and more than one of its neighbor are within 50 pixels, the operation merges it with its closest neighbor. Finally, if the distances between it and all its neighbors are beyond ‘zn’, no operation is done. As for the data file storing fixations of both eyes, the lumping operation is applied to fixations of the same eye.

No matter whether the lumping operation is conducted, before storing extracted fixations, the function marks any fixation shorter than 50 ms (marged by the argument ‘mn’) as invalid fixations. In addition, if ‘rec\_lastFix’ is False, the last fixation in each trial is also marked as invalid.

1. **write\_Sac\_Report:** store the data frame of extracted saccades (SacDF) into a csv file as the generated saccade report. The csv file has the name \*\_Sac.csv, ‘\*’ is the subject ID. It has two arguments: ‘direct’, directory of the generated csv file; and ‘SacDF’, data frame of extracted saccades.
2. **write\_Fix\_Report:**store the data frame of extracted fixations (FixDF) into a csv file as the generated fixation report. The csv file has the name \*\_Fix.csv, ‘\*’ is the subject ID. It has two arguments: ‘direct’, directory of the generated csv file; and ‘FixDF’, data frame of extracted fixations.
3. **read\_write\_SRRasc:**read a particular ASCII data file, extract saccades and fixations in it, and store corresponding data frames into saccade and fixation reports. It first checks whether the data file and relevant region files exist in the specified folder; if not, it throws a warning message and stops.

*Arguments:*

* ‘direct’: directory of the ASCII file and region file(s).
* ‘datafile’: name of the ASCII file.
* ‘regfileNameList’: list of names of region files. If it is empty, the function automatically gets all csv region files from ‘direct’. If the ASCII data file contains the data of many trials, the corresponding region files should be sorted accordingly.
* ‘ExpType’: type of experiment, it can be set as ‘RP’, meaning ‘reading paragraph’, in the paragraph reading experiment.
* Additional arguments passed to **read\_SSRasc** (‘rec\_lastFix’, ‘lump\_Fix’, ‘ln’, ‘zn’, ‘mn’): see **read\_SSRasc**.

1. **read\_write\_SRRasc\_b:** the batch version of **read\_write\_SRRasc**. It reads one or more ASCII data files from the same experiment, extract saccades and fixations in those data files, and store corresponding data frames into saccade and fixation reports. It first collects all ASCII data files and relevant region files in the specified folder; if there is no such file, it throws a warning message and stops. It has no argument of ‘datafile’, and the other arguments are identical to those of **read\_write\_SRRasc**.

**Examples:**

1. **SacDF, FixDF = read\_SRRasc(‘./asc\_example’, ‘1950024.asc’, [‘story01.region.csv’, ‘story02.region.csv’ , ‘story03.region.csv’], ‘RP’):** read 1950024.asc in the subfolder ‘asc\_example’, and extract saccades and fixations into SacDF and FixDF. In this example, during fixation extraction, the lumping operation is conducted based on the default values of relevant parameters. ‘regfileNameList’ can be empty; in this case, all \*.region.csv files in the subfolder are recognized as region files.
2. **write\_Sac\_Report(‘./asc\_example’, SacDF):** generate a saccade report (1950024\_Sac.csv) based on SacDF created in the first example. The generated report is in the subfolder ‘asc\_example’.
3. **write\_Fix\_Report(‘./asc\_example’, FixDF):** generate a saccade report (1950024\_Fix.csv) based on FixDF created in the first example. The generated report is in the subfolder ‘asc\_example’.
4. **Read\_write\_SRRasc(‘./asc\_example’, ‘1950024.asc’, [‘story01.region.csv’, ‘story02.region.csv’, ‘story03.region.csv’], ‘RP’):** read 1950024.asc in the subfolder ‘asc\_example’, extract saccades and fixations therein, and generate associated reports. ‘regfileNameList’ can be empty; in this case, all \*.region.csv files in the subfolder are recognized as region files.
5. **read\_write\_SRRasc\_b(‘./asc\_example’, [], ‘RP’):** read all ASCII data files (1950024.asc, 1950088.asc, 1950099.asc) in the subfolder ‘asc\_example’, extract saccades and fixations therein, and generate associated reports. Note that 1950099.asc records saccades and fixations of both eyes, the generated reports for it list the saccades and fixations of the left eye, and then the right eye, in each trial.
6. **Functions for classifying fixations and saccades into different lines of texts, assigning fixations into different word regions, and identifying cross-line saccades and fixations**

The purpose of the functions in this section is to address the occasional messiness encountered in data for eye-movements over print, especially with regard to the vertical positions of fixations, to classify saccades and fixations as corresponding to different lines of a multi-line text stimulus, and to classify fixations as corresponding to different word regions of a multi-line text stimulus. The classifications conducted by these functions can then be used to facilitate calculation of regional summaries of gaze data.

In eye-movement multi-line text reading experiments, analysis of reading patterns is usually based on region-wise (e.g., word-wise) summaries of eye movements, and how to reliably classify saccades, and especially fixations, belonging to specific words in different lines of text is critical for the analysis. This chore is difficult in the context of multi-line text presentation, given that in practice accurate measurement of vertical positions of fixations can be challenging. In addition, gaze data of some subjects during multi-line text reading are not well-aligned with the baselines of displayed text. Furthermore, some subjects tend to jump forward and backward during multi-line text reading, especially at the beginning or the end of a line, and such long eye-movement may not be achieved by a single saccade, which adds difficulty in correctly assigning fixations into different lines of the text. Noting these, extending from line-by-line presentation to multi-line presentation is by no means trivial, but requires technical and psychological considerations. Yet, reading multi-line texts is common in everyday life, and in the experimental context, such behavior may afford the opportunity of asking interesting research questions about text comprehension in a way that is more nature than what can be done with line-by-line presentation of multi-line texts.

The functions in this section aim to overcome these difficulties and enable an efficient and reliable way of classifying saccades and fixations into different lines of text, assigning fixations into different word regions, and capturing forward and backward cross-line jumping during reading. Based on the experiences of conducting and observing multi-line text reading, we design our algorithms based on the following assumptions.

First, we rely on long cross-line saccade or fixation transitions from one line of text to the next (or previous) line to locate the lines of text that subjects are currently reading. A saccade is *a cross-line saccade*, if it starts at a line of text and stops at the next (forward cross-line saccade) or the previous (backward cross-line saccade) line. A fixation is *a cross-line fixation*, if it occurs at a line of text and the fixation immediately before it occurs at the previous (forward cross-line fixation) or the next (backward cross line fixation) line. Such transition usually occurs at the beginning or the end of a line, but due to parafoveal reading and reader choices about the optimal viewing position within a word, it may not necessarily locate at the first or last word of a line.

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

Third, typical regressive eye-movements within a line of text are usually not big in terms of horizontal movement (jumping only a few words), whereas a cross-line saccade tends to jump a large horizontal distance (more than a half of total words in the same line). Therefore, a large, forward jump in terms of horizontal position can indicate that subjects start to read a new line, whereas a large, backward jump can reflect that subjects go back to the previous line.

Finally, at the end of reading, subjects may rescan some parts of the whole paragraph. Saccades and fixations detected during such wrap-up behavior are often not useful for analysis, and need to be removed. Large change in vertical position can indicate such wrap-up behavior. Therefore, vertical position, though not accurate enough to classify fixations across lines, is still informative for the end of reading.

These assumptions are implemented in the relevant helper functions described below. Based on these assumption, saccades having large (and mostly) horizontal displacement and temporally concatenate fixations inducing large horizontal distance are used to identify which line of text subjects are reading, and saccades having largely vertical displacement and temporally concatenate fixations inducing large vertical distance are used to identify the end of reading. Considering that some of the assumptions may not hold in extreme cases and the classified results in those cases could be questionable, the functions in this section may generate a log file to record the information of such questionable data, which guides users to do additional check.

We provide user functions to read the saccade and fixation reports (\*\_Sac.csv and \*\_Fix.csv), and based on the above assumptions and relevant region files, the functions call relevant helper functions to classify saccades and fixations into different lines of the text and assign fixations into different word regions, and create data frames respectively storing classified saccades and fixations and identified cross-line saccades and fixations. We also provide user functions to generate reports of classified and cross-line saccades and fixations. To ease processing, we further provide batching functions to routinely handle saccade and fixation reports of many subjects at once. Apart from these functions, we also provide two user functions that unify all the processing from reading ASCII data files to generating csv files storing classified and cross-line saccades and fixations.

**User functions:**

1. **cal\_crlSacFix:** read saccade and fixation reports of a subject, classify saccades and fixations into different lines of text and assigns fixations into different word regions, and create data frames storing classified and cross-line saccades and fixations.

*Arguments:*

* ‘direct’: directory of saccade and fixation reports (\*\_Sac.csv and \*\_Fix.csv) and the relevant region file.
* ‘subj’: subject ID, identical to ‘\*’ in the report files.
* ‘regfileNameList’: list of names of region files. Region files are sorted according to the trials of the experiment. If ‘regfileNameList’ is empty, the function automatically reads all \*.region.csv files in ‘direct’ as region files.
* ‘ExpType’: type of experiment.
* ‘recStatus’: whether record questionable saccades and fixations into a log file; default value is True (recording). See more description in *Algorithm*.
* ‘diff\_ratio’: the ratio of the maximum horizontal distance between the center of the last word and the center of the first word in a line of text; default value is 0.6. It determines the cross-line jumping threshold for identifying a cross-line saccade and fixation. See more description in *Algorithm*.
* ‘frontrange\_ratio’: the ratio of the maximum horizontal distance between the center of the last word and the center of the first word in a line of text; default value is 0.2. It is used to identify backward cross-line saccades and fixations. See more description in *Algorithm*.
* ‘y\_range’: threshold of vertical position change in pixels; default value is 60. Saccade or fixation that induces a vertical position change greater than this threshold can indicate the end of reading. See more description in *Algorithm*.
* ‘fix\_method’: method to classify fixations into different lines of text; default value is ‘DIFF’. If ‘fix\_method’ is ‘DIFF’, fixations are directly classified based on identified cross-line fixations; if ‘fix\_method’ is ‘SAC’, we first identify cross-line saccades, and then classify fixations into different lines of text based on identified cross-line saccades. See more description in *Algorithm*. Note that since the logics of identifying cross-line fixations and cross-line saccades are similar, the classification results using either method are usually the same. However, in cases where some saccades are not captured by the eye-tracker (due to blinking or other reason), the classification results based on ‘DIFF’ could be more reliable than those based on ‘SAC’.
* ‘addCharSp’: number of single character space added to capture overshoot fixations; default value is 1.

*Outputs:*

**cal\_crlSacFix** reads the saccade and fixation reports into data frames SacDF and FixDF, identifies cross-line saccades and fixations among these eye-movement events and stores identified events respectively in crlSacDF and crlFixDF, and updates the ‘line\_no’ columns in SacDF and FixDF, and the ‘region\_no’ columns in FixDF accordingly. Finally, it returns the four data frames. For the reports of saccades and fixations of both eyes, it first classifies the left eye saccades and fixations, and then the right eye.

After update, the ‘line\_no’ column in SacDF is changed from ‘NaN’ (missing value before classification) to either a number corresponding to the index (starting from 1) of a line in the text, or a combination of two line indices linked by ‘\_’, e.g., ‘2\_3’, indicating that the saccade is a cross-line saccade from line 2 to line 3 of the text. Some ‘line\_no’ values remain as ‘NaN’, if those saccades occur after the end of reading. Similarly, the ‘line\_no’ column in FixDF is changed from ‘NaN’ to a line index, indicating the line of the text that fixation belongs to. Fixations having different ‘line\_no’ values from their previous neighbors are cross-line fixations. Some ‘line\_no’ values remain as ‘NaN’, if those fixations occur after the end of reading. In addition, the ‘region\_no’ column in FixDF is also changed from ‘NaN’ to a word region index according to the region file. Some ‘region\_no’ values remain as ‘NaN’, if those fixations occur after the end of reading.

crlSacDF records the information of cross-line saccades. It has 15 columns:

* + ‘subj’, ‘trial\_id’, ‘eye’: see the corresponding columns in SacDF.
  + ‘startline’, ‘endline’: the starting and ending lines of the cross-line saccade.
  + ‘Saclineindex’: numerical index of the saccade in SacDF, starting from the first saccade of the trial. It helps locate the saccade in SacDF.
  + ‘start\_time’, ‘end\_time’, ‘duration’, ‘x1\_pos’, ‘y1\_pos’, ‘x2\_pos’, ‘y2\_pos’, ‘ampl’, ‘pk’: see the corresponding columns in SacDF.

crlFixDF stores the information of identified cross-line fixations, it has 13 columns:

* + ‘subj’, ‘trial\_id’, ‘eye’: see the corresponding columns in FixDF.
  + ‘startline’, ‘endline’: the starting and ending lines of the cross-line fixation. The current fixation is at ‘endline’, and the immediately previous fixation is at ‘startline’.
  + ‘Fixlineindex’: numerical index of the fixation in FixDF, starting from the first fixation of the trial. It helps locate the saccade in FixDF.
  + ‘start\_time’, ‘end\_time’, ‘duration’, ‘x\_pos’, ‘y\_pos’, ‘pup\_size’, ‘valid’: see the corresponding columns in FixDF.

Apart from the four data frames, **cal\_crlSacFix** may also generate a text file (log.txt) to record the information (subject ID, trial ID, saccade or fixation) of questionable data, e.g., ‘Subj: 1950088 Trial 2 crlSac start/end need check!’ or ‘Subj: 1950088 Trial 2 crlFix start/end need check!’, indicating that the data of a particular subject (1950088) in a particular trial (2) need additional check. In general, if the first identified cross-line saccade or fixation is a backward one (indicating that some data are missing or the first part of the data are messy), or the last identified cross-line saccade or fixation does not cover the last line of text (indicating that the data are incomplete or some cross-line saccades or fixations are incorrectly identified), the data in this trial are marked as questionable, and its information is passed to the log file.

*Algorithm:*

**cal\_crlSacFix** first checks whether the relevant reports and region files exist in the specified folder (‘direct’); if not, it throws a warning message and stops. If all the files exist, **cal\_crlSacFix** reads the reports, calls helper functions to classify saccades and fixations, and stores the results into data frames.

Identification of cross-line saccades and fixations proceeds as follows. Here, we only describe the identification of cross-line fixations, and the same logic can be applied to identify cross-line saccades.

First, based on the region file, we measure the horizontal distance between the center of the first word and the center of the last word in each line of the text, which can be viewed as the maximum distance for that line of the text. Based on these distances, we set up the cross-line jumping thresholds (‘diff\_ratio’ times the maximum distance) for every line of the text, and the frontal regions (‘frontrange\_ratio’ times the maximum distance) for every line of the text, both of which are used for identifying forward and backward cross-line fixations.

Second, we reorganize the fixations by grouping together concatenate fixations that all move leftward (*a leftward moving fixation* is a fixation whose ‘x\_pos’ is smaller than that of the previous fixation). As for rightward moving fixations (*a rightward moving fixation* is a fixation whose ‘x\_pos’ is bigger than that of the previous fixation), we treat each of them as a group. Here, we use ‘leftward’ or ‘rightward’, instead of ‘regressive’, because in the context of multi-line text reading, a leftward moving fixation is not necessarily regressive in the sense of moving to an earlier part of the text; consider the case of a typical ‘retrace’ saccade from the end of one line to the beginning of the next, although its direction is leftward, its direction is forward, as it moves to the word on the next line.

Third, after grouping, we examine each group in sequence.

For each group of rightward moving fixation, if its previous fixation occurs in the frontal region of the line and the horizontal distance between it and its previous fixation is greater than the cross-line jumping threshold of the line, the fixation can be identified as a backward cross-line fixation.

For each group of leftward moving fixation, there are two cases. First, there is only one fixation in the group. Then, if the distance between the fixation and its previous one is greater than the cross line jumping threshold of the line, the fixation can be identified as a forward cross-line fixation. Second, the group contains more than one fixation. Then, if the horizontal distance between the last fixation and the first one in the group is greater than the cross-line jumping threshold, the group of fixations achieve a forward cross-line eye-movement.

In the second case (a group of fixations collectively achieve a forward cross-line eye-movement), we need to identify which fixation in the group can be viewed as a cross-line fixation. If there exists a fixation in the group such that the horizontal distance between it and its previous fixation (this fixation could be in the same group or the previous group) is greater than the cross-line jumping threshold of the line, the fixation can be identified as the forward cross-line fixation. If there is no such fixation in the group, we look for the fixation(s) that has the greatest change in either horizontal position (difference in ‘x\_pos’) or vertical position (difference in ‘y\_pos’) compared with its previous fixation. If the fixation having the greatest change in horizontal position is the same as the one having the greatest change in vertical position, that fixation can be identified as the forward cross-line fixation. Otherwise, the fixation having a smaller index (occurring earlier in time) can be identified as the forward cross-line fixation. This decision is based on the assumption that a cross-line fixation should be the earliest fixation that induces a great change in either horizontal or vertical position.

Once a cross-line fixation is identified, we update the cross-line jumping threshold and the frontal region to the values of the next or previous line of the text, dependent on whether the cross-line fixation is forward or backward. After the threshold and the frontal region of the last line of the text are used for identifying cross-line fixations, if there are still fixations unclassified, we classify them into the last line of the text. Among these fixations, if there exists one fixation such that the vertical distance between it and its previous one is greater than ‘y\_range’, we assume that the reading ends before that fixation, and the ‘line\_no’ values of that fixation and those afterward are set to ‘NaN’. On the other hand, if all fixations are classified before the threshold and the frontal region of the last line of the text are used for identification, the data of the trial are flagged as questionable, and their information is passed to the log file (if ‘recStatus’ is True).

Finally, after cross-line fixations are identified and stored in crlFix, the other fixations lying between two cross-line fixations can be classified into the corresponding line. As for the data containing fixations of both eyes, we first classify the left eye cross-line fixations, and then the right eye.

Note that if ‘fix\_method’ is ‘SAC’, we first use the above logic to identify cross-line saccades, and then classify fixations based on the starting and ending times of the cross-line saccades; fixations whose starting and ending times are between the starting (or ending) times of two concatenate cross-line saccades in crlSac can be classified into the corresponding line, and the ‘line\_no’ values of the fixations occurring after the starting time of the saccade indicating the end of reading (inducing vertical position change greater than ‘y\_range’) are set to ‘NaN’.

After classifying the saccades and fixations into different lines of the text, the function further assigns fixations into specific word regions of the text. For word regions at the beginning or the end of a line of text, fixations may overshoot and fall slightly outside the word regions. To capture overshoot fixations in the boundary words, we extend the left boundaries of the starting words of each line and the right boundaries of the ending words of each line by one space (determined by ‘addcharSp’, see *Arguments*). ‘addCharSp’ helps to capture such overshoot fixations and correctly assign them into appropriate word regions for calculation of eye-movement measures. For words in the middle of a line, no such adjustment is needed, fixations are assigned to appropriate word regions mainly based on the bounding boxes around the words and the line indices of the fixations.

1. **write\_Sac\_crlSac:** store the data frames of classified saccades (SacDF) and cross-line saccades (crlSac) into csv files (\*\_Sac.csv and \*\_crlSac.csv), ‘\*’ is the subject ID. It has four arguments: ‘direct’, directory of the generated csv file; ‘subj’, subject ID; ‘SacDF’ and ‘crlSac’, data frames of classified saccades and cross-line saccades.
2. **write\_Fix\_crlFix:** store the data frames of classified fixations (FixDF) and cross-line fixations (crlFix) into csv files (\*\_Fix.csv and \*\_crlFix.csv), ‘\*’ is the subject ID. It has four arguments: ‘direct’, directory of the generated csv file; ‘subj’, subject ID; ‘FixDF’ and ‘crlFix’, data frames of classified fixations and cross-line fixations.
3. **cal\_write\_SacFix\_crlSacFix:**read the saccade and fixation reports from a particular subject and relevant region files, classify these saccades and fixations, and store classified and cross-line saccades and fixations into csv files. It may also generate a log file storing the information of questionable data. It first checks whether the reports of the subject and relevant region files exist in the specified folder; if not, it throws a warning message and stops. It calls **cal\_crlSacFix**, **write\_Sac\_crlSac**, and **write\_Fix\_crlFix** to work.

*Arguments:*

* ‘direct’: directory of the saccade and fixation reports from one or more subjects and relevant region files, the generated csv files are also stored there.
* ‘subj’: subject ID.
* ‘regfileNameList’: list of names of region files. Region files are sorted according to the trials of the experiment. If ‘regfileNameList’ is empty, the function automatically reads all \*.region.csv files in ‘direct’ as region files.
* ‘ExpType’: type of experiment.
* Additional arguments (‘recStatus’, ‘diff\_ratio’, ‘frontrange\_ratio’, ‘y\_range’, ‘fix\_method’, ‘addCharSp’) passed to **cal\_crlSacFix**: see **cal\_crlSacFix**.

1. **cal\_write\_SacFix\_crlSacFix\_b:**the batch version of **cal\_write\_SacFix\_crlSacFix**. It reads the saccade and fixation reports from one or more subjects and relevant region files, classifies these saccades and fixations, and stores classified and cross-line saccades and fixations into csv files. It may also generate a log file storing the information of questionable data. It first collects the reports and relevant region files in the specified folder; if there is no such file, it throws a warning message and stops. It has no argument of ‘subj’, and the other arguments are identical to those of **cal\_write\_SacFix\_crlSacFix**.
2. **read\_cal\_SRRasc:**read ASCII data file of a subject, extract saccades and fixations therein, classify them, and return data frames respectively storing classified saccades (SacDF), cross-line saccades (crlSac), classified fixations (FixDF), and cross-line fixations (crlFix).

*Arguments:*

* ‘direct’: directory of the ASCII data file and relevant region files.
* ‘datafile’: name of the ASCII data file.
* ‘regfileNameList’: list of names of region files. Region files are sorted according to the trials of the experiment. If ‘regfileNameList’ is empty, the function automatically reads all \*.region.csv files in ‘direct’ as region files.
* ‘ExpType’: type of experiment.
* Arguments for extracting saccades and fixations (‘rec\_lastFix’, ‘lump\_Fix’, ‘ln’, ‘zn’, ‘mn’): see **read\_SRRasc**.
* Arguments for classifying saccades and fixations (‘recStatus’, ‘diff\_ratio’, ‘frontrange\_ratio’, ‘y\_range’, ‘fix\_method’, ‘addCharSp’): see **cal\_crlSacFix**.

1. **read\_cal\_write\_SRRasc:**read the ASCII data file of a particular subject, extract saccades and fixations therein, classify them, and generate csv reports of classified and cross-line saccades and fixations. It first checks whether the ASCII data file and relevant region files exist in the specified folder; if not, it throws a warning message and stops. It calls **read\_cal\_SRRasc**, **write\_Sac\_crlSac**, and **write\_Fix\_crlFix** to work.

*Arguments:*

* ‘direct’: directory of the ASCII data file and relevant region files, the generated csv files are also stored there.
* ‘datafile’: name of the ASCII data file.
* ‘regfileNameList’: list of names of region files. Region files are sorted according to the trials of the experiment. If ‘regfileNameList’ is empty, the function automatically reads all \*.region.csv files in ‘direct’ as region files.
* ‘ExpType’: type of experiment.
* Additional arguments for extracting and classifying saccades and fixations (‘rec\_lastFix’, ‘lump\_Fix’, ‘ln’, ‘zn’, ‘mn’, ‘recStatus’, ‘diff\_ratio’, ‘frontrange\_ratio’, ‘y\_range’, ‘fix\_method’): see **read\_cal\_SRRasc**.

1. **read\_cal\_write\_SRRasc\_b:**the batch version of **read\_cal\_write\_SRRasc**. It reads ASCII data files of one or more subjects, extracts saccades and fixations therein, classify them, and generate csv reports of classified and cross-line saccades and fixations. It first collects the ASCII data files and relevant region files exist in the specified folder; if there is no such file, it throws a warning message and stops. It has no argument of ‘datafile’, and the other arguments are identical to those of **read\_cal\_write\_SRRasc**.

**Examples:**

1. **SacDF, crlSac, FixDF, crlFix = cal\_crlSacFix(‘./asc\_example’, ‘1950088’, [‘story01.region.csv’, ‘story02.region.csv’, ‘story03.region.csv’], ‘RP’):** read the saccade and fixation reports (1950088\_Sac.csv and 1940088\_Fix.csv) in the subfolder ‘asc\_example’, classify those saccades and fixations into different lines, and return data frames respectively storing classified and cross-line saccades and fixations. The data of one trial are questionable, a log file is generated in the same subfolder, which stores the information of the questionable data. ‘regfileNameList’ can be empty; in this case, all \*.region.csv files in the subfolder are recognized as region files.
2. **write\_Sac\_crlSac(‘./asc\_example’, ‘1950088’, SacDF, crlSac):** generate two csv reports respectively for classified saccades (1950088\_Sac.csv) and cross-line saccades (1950088\_crlSac.csv) of subject 1950088 in the subfolder ‘asc\_example’.
3. **write\_Fix\_crlFix(‘./asc\_example’, ‘1950088’, FixDF, crlFix):** generate two csv reports respectively for classified fixations (1950088\_Fix.csv) and cross-line fixations (1950088\_crlFix.csv) of subject 1950088 in the subfolder ‘asc\_example’.
4. **cal\_write\_SacFix\_crlSacFix(‘./asc\_example’, ‘1950024’, [‘story01.region.csv’, ‘story02.region.csv’, ‘story03.region.csv’], ‘RP’):** read the saccade and fixation reports of the subject ‘1950024’ in the subfolder ‘asc\_example’, extract saccades and fixations therein, classify those saccades and fixations, and generate csv reports storing classified and cross-line saccades and fixations. It may also generate a log file storing the information of some questionable data. ‘regfileNameList’ can be empty; in this case, all \*.region.csv files in the subfolder are recognized as region files.
5. **cal\_write\_SacFix\_crlSacFix\_b(‘./asc\_example’, [], ‘RP’):** automatically read the saccade and fixation reports of the three subjects in the subfolder ‘asc\_example’, extract saccades and fixations therein, classify those saccades and fixations, and generate csv reports storing classified and cross-line saccades and fixations. It may also generate a log file storing the information of some questionable data.
6. **SacDF, crlSac, FixDF, crlFix = read\_cal\_SRRasc(‘./asc\_example’, ‘1950099.asc’, [‘story01.region.csv’, ‘story02.region.csv’, ‘story03.region.csv’], ‘RP’):** read 1950099.asc in the subfolder ‘asc\_example’ to extract saccades and fixations, classify them, and return four data frames storing classified and cross-line saccades and fixations. ‘regfileNameList’ can be empty; in this case, all \*.region.csv files in the subfolder are recognized as region files.
7. **read\_cal\_write\_SRRasc(‘./asc\_example’, ‘1950024.asc’, [‘story01.region.csv’, ‘story02.region.csv’, ‘story03.region.csv’], ‘RP’):** read 1950024.asc in the subfolder ‘asc\_example’, extract saccades and fixations therein, classify them into different lines of the texts, and generate csv reports for classified and cross-line saccades and fixations of different subjects.A log file is also generated to store information about questionable data. ‘regfileNameList’ can be empty; in this case, all \*.region.csv files in the subfolder are recognized as region files.
8. **read\_cal\_write\_SRRasc\_b(‘./asc\_example’, [], ‘RP’):** automatically read all ASCII data files of the three subjects in the subfolder ‘asc\_example’, extract saccades and fixations therein, classify them into different lines of the texts, and generate csv reports for classified and cross-line saccades and fixations of different subjects.A log file is also generated to store information about questionable data.
9. **Functions for visualizing saccades and fixations on bitmaps**

To better visualize the experimental data, we provide two sets of functions in this section to draw saccades and fixations upon actual bitmaps that subjects read in the experiment. The first set of functions can generate bitmaps with saccades and fixations. The second set of functions can generate animations that dynamically illustrate fixations along oral reading sounds. All these functions are for the demonstration purpose.

* 1. ***Functions for showing saccades and fixations on bitmaps***

To use these functions, users need to put the bitmaps, region files, and csv reports (classified saccades and fixations) generated by the functions in the other sections into the same folder. As for the reports containing saccades and fixations of both eyes, the functions first show saccades and fixations of the left eye, and then the right eye. In the figures created by our functions, the single/multi-line text is shown as the background. Saccades are denoted by solid lines, and fixations are denoted by solid circles, whose radiuses are scaled to the durations of fixations. A leftward moving saccade is shown in blue, and a rightward moving saccade in red. A left eye fixation is shown in a green circle, and a right eye fixation in a red circle. In addition, the functions have the option of showing the actual durations of fixations near the circles in blue texts, and the option of showing all saccades and fixations or only those before the end of normal reading.

**User functions:**

1. **draw\_SacFix:** display classified and/or cross-line saccades and fixations of a particular subject on corresponding bitmaps.

*Arguments:*

* ‘direct’: directory of the bitmaps, region files, and csv reports of a subject, the generated figures are also stored there.
* ‘subj’: subject ID.
* ‘regfileNameList’: list of names of region files. If ‘regfileNameList’ is empty, the function automatically reads all \*.region.csv files in ‘direct’ as region files.
* ‘bitmapNameList’: list of names of bitmaps used in each trial of the experiment. Names of the bitmaps and corresponding region files in ‘regfileNameList’ and ‘bitmapNameList’ are sorted according to the trials of the experiment. If ‘PNGmethod’ (see below) is 0, ‘bitmapNameList’ can be empty; in this case, the function automatically reads all \*.png files in ‘direct’ as bitmap files.
* ‘medthod’: if ‘method’ is ‘ALL’, the function draws two figures (\*\_FixSac\_trial#.png and \*\_crlFixSac\_trial#.png, ‘\*’ is subject ID, and ‘#’ is trial ID), one showing all saccades and fixations during reading (fixations and saccades occurring after the end of reading are not shown), and the other showing all identified cross-line saccades and fixations; if ‘method’ is ‘Sac’, it draws two figures (\*\_Sac\_trial#.png and \*\_Fix\_trial#.png) respectively showing all saccades during reading and identified cross-line saccades; and if ‘method’ is ‘Fix’, it draws two figures (\*\_Fix\_trial#.png and \*\_crlFix\_trial#.png) respectively showing all fixations during reading and identified cross-line fixations.
* ‘max\_FixRadius’: maximum radius of fixation circles in pixels; default value is 30.
* ‘drawFinal’: whether draw saccades and fixations after the end of normal reading; default value is False (only drawing the saccades and fixations during normal reading).
* ‘showNum’: whether show actual durations of fixations; default value is False (not showing durations).
* ‘PNGmethod’: whether draw saccades and fixations based on existing bitmaps or re-generate bitmaps from relevant region files; default value is 0. If ‘PNGmethod’ is 0, saccades and fixations are drawn upon existing bitmaps; if ‘PNGmethod’ is 1, the function regenerates bitmaps based on region files, and then, draw saccades and fixations on these newly-created bitmaps. Due to different monitor setting, regenerated bitmaps may be slightly different from the ones used in the experiment.

1. **draw\_SacFix\_b:** the batch version of **draw\_SacFix**. It calls **draw\_SacFix** to draw relevant figures for the data files of each subject. It first collects the relevant files exist in specified folder; if there is no such file, it throws a warning message and stops. It has no argument of ‘subj’, and the other arguments are identical to those of **draw\_SacFix**.

**Examples:**

1. **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 csv reports of subject 1950024 (1950024\_Sac.csv, 1950024\_crlSac.csv, 1950024\_Fix.csv, 1950024\_crlFix.csv), draw figures showing all saccades and fixations during reading (1950024\_FixSac\_trial0.png, 1950024\_FixSac\_trial1.png, 1950024\_FixSac\_trial2.png) and figures showing only cross-line saccades and fixations (1950024\_crlFixSac\_trial0.png, 1950024\_crlFixSac\_trial1.png, 1950024\_crlFixSac\_trial2.png). To properly use the 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’. ‘regfileNameList’ can be empty; in this case, all \*.region.csv files in the subfolder are recognized as region files. If ‘PNGmethod’ is 0 (default), ‘bitmapNameList’ can be empty; in this case, all \*.png files in the subfolder are recognized as bitmap files.
2. **draw\_SacFix\_b(‘./asc\_example’, [], [], ‘ALL’):** for the csv reports of all three subjects,generate relevant figures showing classified and cross-line saccades and fixations.
   1. ***Functions for showing saccades and fixations on bitmaps***

To use these functions, users need to put the bitmaps, sound files (in the subfolder ‘sounds’), region files, and CSV reports (raw or classified fixations) generated by the functions in the other sections into the same folder. Due to restrictions of the turtle package in Python, bitmaps as animation background have to be GIF files. We provide a helper function, **changePNG2GIF**, to transform PNG bitmaps into GIF bitmaps (also stored in the subfolder ‘bitmap\_regfile’) for animation. The sound files are WAV files, whose filenames contain the subject ID and trail index (starting from 1).

***Note:* the animation function requires winsound package, which is available only in Windows Python distributions, so it cannot be run properly in Linux Python distributions.**

**User functions:**

1. **changePNG2GIF:** transform PNG bitmaps into GIF bitmaps for the purpose of animation. The GIF bitmaps have the same names of the PNG bitmaps.

*Arguments:*

* ‘direct’: directory of the PNG bitmaps. The generated GIF bitmaps are also stored here.

1. **animate:** generate an animation showing the eye-movements and oral reading in a particular trial of a particular subject. It first checks whether the relevant files (including the sound files, GIF bitmaps, CSV fixation reports) exist in specified folder; if not, it throws a warning message and stops. It generates a screen to display the bitmaps and the fixations along reading. The title of the screen shows the subject ID, trial number, and which eye’s fixations are shown. Fixations are denoted by green circles, whose radiuses are scaled to the durations of fixations. As for the reports containing fixations of both eyes, fixations of the left eye are shown in green circles, and those of the right eye are shown in red circles. Both types of fixations are updated simultaneously during the animation. Along with the animation, it plays the sound files in the background.

*Arguments:*

* ‘direct’: directory of the PNG bitmaps.
* ‘subj’: subject ID.
* ‘trialID’: trial ID (starting from 0, corresponding to trial 1).

*Algorithm:*

**animate** first checks whether required files are in the specified folder ‘direct’; if not, it throws a warning and stops. If all required files are there, based on ‘trialID’, it extracts the fixations in that particular trial from the CSV file, and checks whether the data file contains fixations of a single eye or both eyes. Then, it creates a turtle screen using the bitmap as the background, and one or two turtle players for displaying fixations. If the data file contains fixations of a single eye, it generates one turtle player (green circle for left eye fixations and red circle for right eye fixations); if the data file contains fixations of both eyes, it generates two turtle players respectively for fixations of each eye. After that, it listens to key input to start or terminate the animation. Once the key ‘s’ is pressed, based on winsound.PlaySound(), it plays the sound file in the background, and displays each fixation in the extracted data file in time. The logic of displaying fixations is as follows. First, it calculates the time difference between the starting times of two consecutive fixations (or between the starting of the sound file and the starting time of the first fixation), uses time.sleep() to wait for such time, and then, displays the corresponding fixation. Once the key ‘e’ is pressed, it stops the animation. If the key ‘s’ is pressed again, it restarts the whole animation. If one clicks on the animation screen, it terminates the animation and quits.

It adds additional timers to compensate the time lags due to imprecision of time.sleep() function and running times of Python codes for moving and displaying turtle players, making sure a largely aligned display of fixations along with the sound. However, for a long time paragraph reading, the accumulated errors may still cause mismatch between the sounds and fixations at the later part of the animation. In addition, due to restrictions of the turtle and winsound packages, the animation and the sound played cannot be paused or resumed, and the whole animation cannot be stored as a separate file.

**Examples:**

1. **change(‘./asc\_example’):** transform all PNG bitmaps in the subfolder ‘asc\_example’ into GIF bitmaps.
2. **animate(‘./asc\_example’, ‘1950024’, 0):** generate an animation showing the eye-movements and oral reading of subject 1950024 in the first trial of the experiment. Once it is called, a screen is generated, whose background is the story bitmap of that trial. Due to the resolution of the monitor, the whole screen may not cover the whole bitmap. One can manually adjust the screen size and scroll down or up the screen, without affecting displayed fixations. One can start or restart the animation by pressing the key ‘s’, stop the animation by pressing the key ‘e’, and quit the animation by clicking on the animation screen.
3. **Functions for calculating eye-movement measures**

Based on region-wised summaries, we provide functions in this section to calculate widely-adopted eye-movement measures. These functions require the region files storing position information of words in the single/multi-line text and the CSV reports storing classified fixations and saccades in different lines of the text. All files are generated by the functions in the other sections.

Referring to previous eye-movement literature, we focus on three groups of eye-movement measures, including eight first-pass fixation and regression measures, four regression path measures, and two second-pass fixation measures. Apart from these region-based measures, we also calculate four whole-text-based measures (the definitions of all these measures are shown below in *Outputs* of the user function **cal\_write\_EM**). All the measures are calculated by appropriate helper functions, and the values are stored in corresponding columns of the CSV reports.

**User functions:**

1. **cal\_write\_EM:** calculate the eye-movement measures based on the saccade and fixation reports of one subject, and store the results in CSV files. If some reports are missing in the specified folder, it throws a warning message and stops. As for the reports containing saccades and fixations of two eyes, it first calculates the eye-movement measures based on the left eye data and stores the result in the CSV files, and then the right eye.

*Arguments:*

* ‘direct’: directory of the saccade and fixation reports (\*\_Sac.csv and \*\_Fix.csv, ‘\*’ is the subject ID) of a subject and the relevant region files, the generated CSV files are also stored there.
* ‘subj’: subject ID, identical to ‘\*’ of the CSV files.
* ‘regfileNameList’: list of names of the region files relative to the data files. If ‘regfileNameList’ is empty, it reads all \*.region.csv files in ‘direct’ as region files.
* ‘addCharSp’: number of single character space added to capture overshoot fixations; default value is 1.

*Outputs:*

Based on the number of trials, **cal\_write\_EM** generates a number of CSV files (\*\_EM\_trial#.csv, ‘\*’ is the subject ID and ‘#’ is the trail index starting at 0) respectively storing the values of the eye-movement measures based on the CSV files in each trial. If the saccade and fixation reports contain data of both eyes, it generates two sets of CSV files: \*\_EM\_trial#\_L.csv records the values of the eye-movement measures based on the left eye data, and \*\_EM\_trial#\_R.csv records the values of the eye-movement measures based on the right eye data.

\*\_EM\_trial#.csv, \*\_EM\_trial#\_L.csv, or \*\_EM\_trial#\_R.csv has 32 columns. Some columns record the general information of the reading data (extracted from the saccade and fixation reports), some record word information of the single/multi-line text (extracted from the region files), and the others record the values of eye-movement measures (calculated by the relevant helper functions):

* + ‘subj’, ‘trial\_id’, ‘trial\_type’, ‘trialstart’, ‘trialend’, ‘tdur’, ‘recstart’, ‘recend’, ‘blinks’, ‘eye’: same as the corresponding 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 eye-movement measure).
  + ‘tffixurt’: total duration of the first pass fixation of each word in the text (whole-text eye-movement measure).
  + ‘tffixcnt’: total number of valid fixations in the trial (whole-text eye-movement measure).
  + ‘tregrcnt’: total number of *regressive saccades* (a saccade is a regressive 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 spaces before the word and punctuation afterward; identical to the ‘length’ column in the region file.
  + ‘word’: word in each region (including spaces before the word and punctuation afterward); identical to the ‘Word’ column in the region file.
  + ‘line\_no’: line index of the word in the whole text; identical to the ‘line\_no’ column in the region file.
  + ‘x1\_pos’, ‘x2\_pos’: the left and right boundaries in pixels of the minimal bounding box of each word; identical to the ‘x1\_pos’ and ‘x2\_pos’ columns in 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 a line, ‘mod\_x1’ is ‘x1\_pos’ – ‘addCharSp’ (see *Arguments*); for words at the end of a line, ‘mod\_x2’ is ‘x2\_pos’ + ‘addCharSp’. We use ‘mod\_x1’ and ‘mod\_x2’ to capture overshoot fixations.
  + ‘fpurt’: first-pass fixation time. It is the sum of the durations of one or more first-pass fixations falling into the word region. By default, we only record fixations of 50 ms or longer; shorter fixations are subject to the lumping operation. If there is no first-pass fixation laying within the word region, ‘fpurt’ is ‘NaN’ (missing value) (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 ‘NaN’ (first-pass fixation measure).
  + ‘fpregres’: whether there is a first-pass regression starting from the current word region; if so, ‘fpregres’ is 1, otherwise, ‘fpregres’ is 0. If there is no first-pass fixation in the word region, ‘fpregres’ is ‘NaN’ (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. If there is no first-pass fixation in the word region, ‘fpregreg’ is ‘NaN’ (first-pass regression measure).
  + ‘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). If there is no first-pass fixation in the word region, ‘fpregchr’ is ‘NaN’ (first-pass regression measure).
  + ‘ffos’: offset in characters of the first first-pass fixation in the word region from the first character of the region. If there is no first-pass fixation in the word region, ‘ffos’ is ‘NaN’ (first-pass fixation measure).
  + ‘ffixurt’: duration of the first first-pass fixation in the word region. If there is no first-pass fixation in the word region, ‘ffixurt’ is ‘NaN’ (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 ‘NaN’ (first-pass fixation measure).
  + ‘rpurt’: sum of durations of all fixations in the regression path. A 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 in the current region and those outside the current word region but falling into only the word regions before the current region. If there is no first-pass regression (‘fpregres’ is 0), ‘rpurt’ equals to ‘fpurt’. If there is no first-pass fixation in the word region, ‘rpurt’ is ‘NaN’ (regression path measure).
  + ‘rpcount’: number of fixations in the regression path. If there is no first-pass fixation in the word region, ‘rpcount’ is ‘NaN’ (regression path measure).
  + ‘rpregreg’: the smallest index of the word region visited by the regression path. If there is no regression path (‘fpregres’ is 0), ‘rpregreg’ is 0. If there is no first-pass fixation in the word region, ‘rpregreg’ is ‘NaN’ (regression path measure).
  + ‘rpregchr’: offset in characters in the smallest word region visited by the regression path. If there is no first-pass regression (‘fpregres’ 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). If there is no first-pass fixation in the word region, ‘rpregreg’ is ‘NaN’ (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 ‘NaN’ (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\_write\_EM** first checks whether the relevant CSV reports (\*\_Sac.csv, \*\_Fix.csv, and the region files) exist in the specified subfolder; if not, it throws a warning message and stops. If the files exist, it 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, it calls relevant helper functions to calculate the first-pass fixation and regression measures, regression path measures, and second-pass fixation measures. After calculating these region-based measures, it sums up the whole-text-based measures. Finally, it stores the results into CSV files.

Compared with single-line text, there are several difficulties in calculating the eye-movement 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, ‘mod\_x1’ and ‘mod\_x2’ of the word region, and ‘line\_no’ in which the fixation is classified into. For words at the beginning or the end of a line, we need to use slightly bigger word region to capture overshoot fixations. In addition, physical position of a word region is insufficient to indicate whether this region is before or after the target word region. Therefore, we have to use ‘region’, a numerical index of each word in the text, to clarify regression probability and regression path.

1. **cal\_write\_EM\_b:** the batch version of **cal\_write\_EM**. For each set of saccade and fixation reports of a subject, it calls **cal\_write\_EM** to work. It has no argument of ‘subj’, and the other arguments are identical to those of **cal\_write\_EM**.

**Examples:**

1. **cal\_write\_EM(‘./asc\_example’, ‘1950024’, [‘story01.region.csv’, ‘story02.region.csv’, ‘story03.region.csv’]):** calculate the eye-movement measures based on 1950024\_Fix.csv and 1950024\_Sac.csv, and store the results into 1950024\_EM\_trial0\_L.csv, 1950024\_EM\_trial1\_L.csv, and 1950024\_EM\_trial2\_L.csv. The saccade and fixation reports contain only the left eye data. ‘regfileNameList’ can be empty; in this case, all \*.region.csv files in the subfolder are recognized as region files.
2. **cal\_write\_EM\_b(‘./asc\_example’, []):** calculate the eye-movement measures based on the CSV reports of many subjects in the sub folder ‘asc\_example’, and store the results into corresponding CSV files. Note that if some subject’s CSV reports (say, subject 1950088) are not correctly classified, it will report errors when calculating the eye-movement measures.
3. **Summary and future directions**

The EMF\_RP package provide useful functions covering the major steps in designing and analyzing an eye-movement reading experiment, including generating and displaying stimuli, collecting eye-movement data, and calculating and storing eye-movement measures for subsequent analysis. Many of these functions are flexible and extendable to other relevant eye-movement experiments.

In addition, the package designs flexible ways to generate bitmaps showing single/multi-line text and region files recording position information of words in bitmaps. These functions can not only display English texts, but also elegantly show many Latin family languages, including French, German, Dutch, Spanish, Italian, and Greek, as well as other popular languages in the World, including Chinese, Korean, and Japanese. This feature greatly extends the applications of the package, and makes it suitable for cross-language or multi-language experimental studies.

Furthermore, the package implements a series of efficient algorithms to lump short fixations, identify cross-line saccades and fixations, and classify saccades and fixations into different lines of text. Identifying cross-line saccades and fixations are the most challenging task in analyzing eye-movement data of multi-line text reading. To our knowledge, our distance-based algorithms for classifying cross-line saccades and fixations have not been used in any previous data analysis of eye-movement reading experiments. These algorithms recapitulate an online reading, and such online-based algorithms are simpler and more efficient than some offline algorithms that rely on identifying particular trajectories of saccades that resemble ideal regressions from the end (or the beginning) of a line of text to the beginning (or the end) of another. Our algorithms have been proved to be reliable for capturing not only forward and backward cross-line eye movements but also a mixture of both types of cross-line eye movements during paragraph reading. All these help overcome the most challenging tasks in this line of research.

Finally, the package calculates a wide scope (first-pass fixation, regression, and second-pass fixation) of eye-movement measures used in previous eye-movement literature, which allows analysis of reading patterns from different perspectives. All these features make the package necessary and very useful to the current reading research based on eye-movements.

Despite of these advantages, 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, the saccades and fixations of some subjects cannot be accurately classified by the current functions. With more experimental studies in multi-line text reading, we can update the available algorithms to automatically and efficiently capture various reading patterns and classify relevant saccades and fixations into different lines of text. In addition, when a cross-line reading is achieved not by a single jump, but by a group of saccades and fixations, the current function estimates the fixation or saccade that can induce the greatest change in either horizontal or vertical position as the cross-line fixation or saccade. However, such criterion may not always hold, and we have not clearly understood what cognitive process induces such slow cross-line reading. Such phenomenon is not shown in line-by-line presentation of multi-line text, and may serve as an entry point to better understand reading behavior and its cognitive bases.

Other aspects concern some practical issues, and can be improved by means of advanced techniques. For example, the current functions in the package can only generate bitmaps displaying some major World languages. We can further modify these functions to display texts of other academically interesting languages, such as Arabic or Hebrew, and to show other types of print style, such as displaying texts from right to left, as in Arabic, or displaying texts vertically, as in traditional ways of Chinese writing. Probably following the same logic, we can extend these functions to capture cross-line or cross-column saccades or fixations during reading of texts in such displaying formats. In addition, the total number of words in multi-line text (usually over 100 and more) is much bigger than that of single-line text (usually less than 20). Accordingly, calculating eye-movement measures in multi-line text is much slower than calculating the measures in single-line text. As for the current package, it takes several seconds to extract and classify one subject’s data, and one or two minutes to calculate one subject’s eye-movement measures, but much longer to automatically handle many subjects’ data. There is a necessity to increase the efficiency of calculation.

All the above-mentioned theoretical and practical issues pave the future directions of the EMP\_RP package.