Bayesian parameter estimation for the E-Z Reader model using the Python programming language

Lab report

Cognitive Science – Embodied Cognition (M.Sc.)

University of Potsdam, Germany

Department of Psychology

Submitted by: Nils Wendel Heinrich

Matriculation Nr.: 800197

E-Mail: nheinrich@uni-potsdam.de

Abstract

The E-Z Reader model Reichle, Pollatsek, Fisher, and Rayner (1998), Reichle, Rayner, and Pollatsek (1999) is the most prominent computational model of eye-movement control in reading, which drove reading research immensely with its unique predictions. In the last two decades, however, several competitor models were developed which directly challenge the E-Z Reader in its theoretical approach. Therefore, model comparison will play a big role in this domain in the near future. To conduct such a model comparison, compared computational models have to be mathematically validated. The statistical inference validates a model by simply finding out whether the parameter values used to simulate data can be determined through the simulations. Despite being so influential the E-Z Reader model was not challenged in regards of a statistical inference yet. We conducted such a statistical inference for the E-Z Reader model using a synthetic likelihood approach. The results are taken as foundation for a Bayesian parameter estimation. The present report focuses on the documentation of the source code for the statistical inference as well as the Bayesian parameter estimation.

Contents

List of Figures

List of Tables

1 Version control and Python

Using the version control system (VCS) Git gave us full control about small- and large-scale adjustments to the code base. Git records every change to all the files in the parent folder. This way we can recall a specific version of the file. It allows us to revert selected files or even the entire project back to a previous state, compare changes over time, and see when an issue was introduced. Using a VCS also generally means that if we screw things up or lose files, we are able to easily recover the project.

In order to conduct a Bayesian parameter estimation we first translated the Java application which is available on the website of the original author (http://www.erikdreichle.com/downloads.html) to the *Python* 3.8 programming language (Van Rossum & Drake, 2009). There are two reasons behind this: (1) By translating every bit of code of the original application we were able to fully understand the E-Z Reader model. We would have worked with a black box when simply applying the Java code. The degree of insight we received allows us to adjust the model itself at a later stage of this project. (2) Not only the model itself but the whole code base of the project is in *Python* 3.8. This way we did not have to think about handling different type systems from different languages/compilers. The only thing where we moved away from *Python* was `shell` commands to create a programming environment so that the scripts could all be run without having to adjust *Python*-packages on a local machine.

2 Code documentation

This work will focus on the master branch of the project. Other branches are created by the members of the project to adjust the code base and test all changes. Only when the adjusted code works as intended, is the branch merged into the master branch.

All of the scripts discussed here include specific documentation of the respective code. This level of detail is not realized in this report. The following documentation will elaborate the general purpose of each script as well as specific details that are worth mentioning and will serve as an orientation when getting started to work on this code base.

The parent folder contains 5 subfolders, 4 *Python* scripts, and 1 `shell` script. The subfolders are called as specified: `Data`, `Dream_mod`, `EZReader`, `Likelihood_synth_Wood`, and `Scripts`. The Python scripts have the names: `EZ_DREAM_Xpars.py`, `EZ_likelihoodprofile.py`, `save_single_run.py`, and `tst.py`. The `shell` script is called `make_env.sh`.

I will elaborate the individual scripts in chronological order in which they are run in a usual workflow. However, in this report I will not explain the contents of the subfolder `DREAM_mod`. This is because I was not involved in the writing process of the included scripts during my internship.

2.1 Virtual environment

Running `make_env.sh` by simply typing `./make_env.sh` sets up the environment. It installs *Python* 3.8 and creates a virtual environment. Additionally, it upgrades *pip*, *Pythons* native package installer. Subsequently, it installs the *Python* packages *pandas*, *scipy*, and *matplotlib*. The virtual environment has to be sourced by typing source `env_from__make_env/bin/activate` into the `shell`. Within this virtual environment the local machine is ready to run all the scripts of the project regardless of the locally installed software.

2.2 The E-Z Reader model of eye-movement control in reading

The subfolder EZReader includes all 7 Python scripts which together make up the E-Z Reader model. The main script Model.py accesses the other scripts, which contain individual functions or object classes. It contains a definition for the function run_EZReader. The function has the variables *parameters*, *corpus*, *NRuns*, *sentences*, and *timeout*. The file parameters.txt, which is located in the Data subfolder, is passed to the variable parameters of Model.py as a path-object. All free parameters and their values are specified here. Also available in Data is the file SRC98Corpus.txt. It contains all words of the corpus that is to be read by the E-Z Reader model, their frequency, the word length in letters and the cloze predictability. This is passed to the object class corpus of Corpus.py, which in turn is passed to Model.py. The Corpus.py script converts the individual words in the .txt file into word objects and sorts them into several sentence objects, as well as a complete corpus object. Both files were taken directly from the original Java application and were not changed. The variable *NRuns* is initially set to 1 and specifies how many complete readings of the corpus should be simulated. With the variable *sentences* the user can specify how many sentences of the corpus should be read by the model (1-48 in our case). The resulting data will only refer to the specified sentences. The variable *timeout* enables the user to specify a maximum time for the simulation in seconds. The function run_EZReader will aboard as soon as the specified time is elapsed. In this case, the fixation data is still returned as well as a statement that timeout happened, but not every reading of the corpus will be completed or not every sentence of the corpus will be read. *timeout* is initially set to 20. Note, that no timeouts happened during the latest stage of the project. The body of Model.py consists of a while-loop. It constantly checks which internal process of the model takes the least time. This means the calculated virtual and not the real physical time. The fastest process is completed and determines which processes are initiated next. Process.py includes all the E-Z Readers internal states. Each state is an individual

Python function which relocates the position of attention or that of the eye, or both, and simultaneously calculates the process-specific time based on the parameter values passed on to it. `Fixation.py` contains the definition of the *Python* object class fixation. Here, important properties like the virtual time of the fixation or its exact position are assigned to the object. `Saccade.py` also contains the definition for an object class, here `saccade`. The purpose of this object class is to calculate the random and the systematic error. Based on the results the final new position of the virtual eye is determined. The *Python* script `Gamma.py` contains the gamma function defined by the authors of the E-Z Reader. This was defined as a *Python* function and is called every time an internal process determines its own virtual time. The last *Python* script to be discussed is `Display.py`. This is loosely based on the original script of the *Java* application. Since we only work with scanpaths of the eyes, we defined a function here which generates a list and appends all fixations with extensive information to it. The complete list after running the `Model.py` script is the output of the program and therefore represents the data that will be further dealt with. Figure 1 shows an example simulation, which was stored in a `.txt` file.

2.3 Scripts subfolder

The Scripts subfolder contains 3 *Python* scripts: `Annotate.py`, `helper_functions.py`, and `Metrics.py`. They represent overall useful functions, which can be called when doing investigative work with reading data.

2.3.1 Annotation

`Annotate.py` includes 3 definitions for *Python* functions. The function `annotate_reading_data` assigns the fixations (resulting data of the E-Z Reader model) passed to it to the respective reading passes and counts the consecutive fixations in each reading pass. The passed reading data has to be a `pandas.DataFrame`. To label the fixations associated to the passes and to indicate the number of the fixation in the respective pass, the function adds 4 more columns. Columns *cnfix1* and *cnfix2* count the fixations in

	fixationN	duration	position	wordN	freq	freqClass	sentenceN	runN
0	1	351.0112526124668	3.0	1	1.0	0	1	1
1	2	220.83902397215772	7.96333478109042		2	181.0	2	1	1
2	3	305.58166523298723	16.228800359692702		3	1789.0	3	1	1
3	4	242.36802700792305	18.77310603256304		4	3036.0	3	1	1
4	5	253.62759932856432	29.497426901331806		6	81.0	2	1	1
5	6	209.17167655804005	35.741794188377646		7	5372.0	4	1	1
6	7	204.05608518528925	41.79476865140399		8	69974.0	5	1	1
7	8	128.06995140427802	39.04071128163464		8	69974.0	5	1	1
8	9	134.0850869019286	48.90784371927423		10	36414.0	5	1	1
9	1	299.3869589390305	1.5	1	69974.0	5	2	1
10	2	170.9166847084601	5.654833804159435		2	92.0	2	2	1
11	3	297.3137137187037	21.599543757388307		3	52.0	2	2	1
12	4	223.13080582562796	32.096876917155186		6	382.0	3	2	1
13	5	227.8905757346035	40.157149897201556		6	382.0	3	2	1
14	1	289.3048554804327	2.0	1	108.0	2	3	1
15	2	412.7799686865453	12.457782367121384		4	213.0	2	3	1
16	3	288.8932780605257	31.986523071690677		7	3942.0	4	3	1
17	4	252.10873305914836	37.6065131749219		8	3727.0	4	3	1
18	5	167.9686498965614	41.572171046765376		9	1.0	0	3	1
19	6	203.35200537123148	45.94603902673377		10	1068.0	3	3	1
20	1	281.20255516415693	2.0	1	83.0	2	4	1
21	2	98.38732018347505	6.331671014142358		2	413.0	3	4	1
22	3	172.04900429265294	10.862869052580951		3	1.0	0	4	1
23	4	96.53346117352572	5.652180369967052		2	413.0	3	4	1
24	5	148.67276000724218	13.15032199551246		3	1.0	0	4	1
25	6	160.54652136780487	18.722821704918534		4	10593.0	4	4	1
26	7	363.18426908189315	27.954035235063387		6	2715.0	3	4	1
27	8	159.06280761004774	32.27158198220096		7	149.0	2	4	1
28	9	139.7365534129977	38.923888618845325		9	1068.0	3	4	1
29	10	218.95145668721415	41.54373259089815		9	1068.0	3	4	1

Figure 1: Exemplary illustration of a simulation output stored in a `.txt` file. Each line represents one fixation in the simulated scanpath with all its important properties captured in the various columns.

the first or the second and higher reading pass. Columns *xnfix1* and *xnfix2* enumerate the fixations in the respective pass. The function `add_wn` is a one line function which simply counts the values of the *wordN* column of a `pandas.DataFrame` that is passed to it. This function is implemented in the following function, `initiate_corpus`. The last python function in `Annotate.py` is `initiate_corpus`. When passing a file which resembles the `SRC98Corpus.txt` in the `Data` subfolder to it, this function will generate a `pandas.DataFrame` with the included words. The data frame will have the columns *freq* (word frequency), *wlen* (word length), *cloze* (cloze predictability), *word* (the actual word spelled out), *sentenceN* (the number of the sentence the current word is part of), and *wordN* (the number of the word within the given sentence). The function `add_wn` is used to put the appropriate values in this last column *wordN*. Therefore, it will never be called on its own by a user but will always automatically be called when initiating the corpus.

2.3.2 Helper functions

The script helper_functions.py , again, contains 3 definitions for *Python* functions. The function load_parameters_pandas will create a pandas.DataFrame including the parameter labels and their values when a file resembling parameter.txt is passed to it. Likewise, the function load_parameters_dict will create a *Python* dictionary including parameter labels and the appropriate values. This function (and not load_parameters_pandas) is currently implemented in the workflow. The last defined function of this script mkdir_p will create a directory at whichever location is passed to it in the form of a path. Therefore, it is a direct *Python* copy of the shell command mkdir -p .

2.3.3 Scripts

The last *Python* script in the subfolder Scripts is Metrics.py . This script includes every definition of functions used to calculate the summary statistics that are utilized in this project. The summary statistics will be calculated word-based. Therefore, together with the fixation pandas.DataFrame a corpus object from initiate_corpus has to be passed to the appropriate function. Table 1 illustrates the function calls and which summary statistic will be calculated with the according function. For every fixation pandas.DataFrame that is passed, the functions will return one single value representing the appropriate summary statistic.

2.3.4 Synthetic likelihood function

The subfolder Likelihood_synth_Wood contains a single *Python* script, synthetic_Likelihood.py , together with a ReadMe.txt file. The latter only states that the enclosed *Python* script includes the definition of the function which calculates the synthetic likelihood proposed by Wood Wood (2010), calc_likelihood . Two fixation pandas.DataFrame have to be passed to the function, representing $s^\star$ and s. Furthermore, the function has a variable *debugme* which is initially set to *"nope"*. If *debugme* is set to *"debug"*, every component of the likelihood is printed, enabling to check for inconsistencies.

Function	Summary statistic
Durations	
calc_SFD :	calculates mean Single Fixation Duration
calc_SFD_sd :	calculates standard deviation of Single Fixation Duration
calc_FMFD :	calculates mean First of Multiple Fixation Duration
calc_FMFD_sd :	calculates standard deviation of First of Multiple Fixation Duration
calc_SMFD :	calculates mean Second of Multiple Fixation Duration
calc_SMFD_sd :	calculates standard deviation of Second of Multiple Fixation Duration
calc_TVT :	calculates mean Total Viewing Time
calc_TVT_sd :	calculates standard deviation of Total Viewing Time
calc_gazeDur :	calculates mean Gaze Duration
calc_gazeDur_sd :	calculates standard deviation of Gaze Duration
Probabilities	
calc_SFP :	calculates Probability of Single Fixation
calc_SP :	calculates Skipping Probability
calc_refixProb :	calculates Refixation Probability
Other	
calc_meanFixPerSentence :	calculates mean number of fixations per sentence
calc_meanFixPerSentence_sd :	calculates standard deviation of mean number of fixations per sentence

Table 1: Enumeration of all *Python* commands which are defined in Metrics.py and which summary statistic they calculate respectively. The commands are grouped into three categories. They are *Durations*, *Probabilities*, and *Other*.

calc_likelihood will return a single value for every set of pandas.DataFrames which is passed to it.

2.4 *Python* scripts

Testing

The remaining content of the parent folder are 4 *Python* scripts. The *Python* script tst.py has simple testing purposes. When called, it will initiate the run_EZReader function (Sec. 2.2) with the default corpus and parameters object, and *NRuns* set to 1. The variable *sentences* is set to 5, meaning that only the first 5 sentences of the corpus object will be read by the E-Z Reader model. tst.py will return an object called fixations and print it automatically. This script enables to check for inconsistencies when running the code for the E-Z Reader model.

Generate reference data

Running the *Python* script save_single_run.py will again initiate the run_EZReader function (Sec. 2.2). Here, also, the default corpus Data/SRC98Corpus.txt and the default parameters Data/parameters.txt are passed to the function. The variable *NRuns* is set to the initial value (= 1), as is *timeout*. The returned pandas.DataFrame object is saved to a .txt file which is called tmp.txt. This file and the fixation data stored in it represent s, one of the pandas.DataFrames that must be passed to the *Python* function calc_likelihood. Therefore, this file is crucial for the ongoing project. For this data we try to recover the true parameter values used to simulate it. Figure 1 actually shows the first few lines of a tmp.txt file.

Generate likelihood profiles

The *Python* script EZ_likelihoodprofile.py is used to determine if individual combinations of parameters and summary statistics are capable of recovering the true value underlying s. Here, all parameters and their ranges are specified. tmp.txt is loaded and converted to a pandas.DataFrame, *stat_s*.

Data/SRC98Corpus.txt is loaded into an object called *ancorp*. This script will run through every parameter individually, simulating overall 200 fixation objects, with varying parameter values, 20 runs per simulation. The simulations are each individually stored in a pandas.DataFrame object called *s_star*. For each simulation, the synthetic likelihood is calculated using *stat_s*. All resulting likelihood values are plotted against the parameter values used to simulate *s_star*. Additionally, the true parameter value of *stat_s* is indicated via a black vertical line. This procedure is executed for each possible summary statistic. Each plot is saved to a PNG file twice, the only difference being the name of the file. Once the summary statistic is listed first, the other time the name of the parameter. But both times, both, the summary statistic as well as the parameter label are included in the file name. The script also creates two subfolders, Plots_parsfirst and Plots_metricsfirst in which the respective PNG files are stored. The plots are used to visually determine the performance of the parameter-summary statistic combination.

Bayesian parameter estimation

The last Python script to be covered is EZ_DREAM_Xpars.py, where X stands for the number of parameters included in the script. Here a full fledged Bayesian parameter estimation is conducted for the E-Z Reader model. This script utilizes several functions defined in the DREAM_mod subfolder which is not covered in this report. tmp.txt is loaded and stored to an object called *y*. For this object all summary statistics specified in the script are calculated and stored to a vector labeled *stat_s*. Data/SRC98Corpus.txt is loaded and stored to an object called *corpus*. Data/parameters.txt is loaded into an object called parameters. For every parameter that is focused in the Bayesian parameter estimation, parameter ranges are specified and stored in a *Python* dictionary, *par_ranges*. The number of iterations of the procedure is specified in the variable *niter*. Also, the number of chains for the procedure is specified in the variable *nchains*. Another variable is *EZ_runs*. This variable holds the number of readings of the corpus for the simulations and this variable is passed to the *NRuns* variable of the run_EZReader function (Sec. 2.2).

`EZ_DREAM_Xpars.py` contains the definition for the function `trunc_gauss`. The lower and upper bounds of the parameter ranges (*par_ranges*) are passed to this function which calculates the probability density from which the actual parameter values are drawn during the procedure. Running this script will create an amount of result `.txt` files, where the amount is equal to the number of chains specified in nchains. These files are labeled `SynthL_Xpars_Y`, where `X` is again the number of parameters focused in the script and `Y` is the number of the chain from which this file resulted. These `.txt` files will include the values for all parameters, that means all the coordinates for a specific point in the parameter space and its likelihood value (the likelihood that *stat_s* was simulated under this specific set of parameter values). The number of rows in each file (the number of tested points in the parameter space) are equal to the number specified in *niter*. Figure 2 shows an exemplary output of a single chain of this procedure, which was stored to a `.txt` file.

Alpha1	Delta	Lambda	loglik
121.20832053676317	1.2442172111932561	0.810779726285488	-62827.726754365896
121.20832053676317	1.2442172111932561	0.810779726285488	-120326.57716239434
121.20832053676317	1.2442172111932561	0.810779726285488	-105540.5422424149
121.20832053676317	1.2442172111932561	0.810779726285488	-82116.81220975242
121.20832053676317	1.2442172111932561	0.810779726285488	-43600.982953194376
121.20832053676317	1.2442172111932561	0.810779726285488	-195186.87107373204
121.20832053676317	1.2442172111932561	0.810779726285488	-46538.741949094445
121.20832053676317	1.2442172111932561	0.810779726285488	-66350.46946339037
121.20832053676317	1.2442172111932561	0.810779726285488	-178746.1691478037
121.20832053676317	0.9566431151857601	0.810779726285488	1.1484898304766784e+17
121.20832053676317	0.9566431151857601	0.810779726285488	-161700.78276735492
121.20832053676317	0.9566431151857601	0.810779726285488	-39188.932198308226
121.20832053676317	0.9566431151857601	0.810779726285488	-150221.40178680175
121.20832053676317	0.9566431151857601	0.810779726285488	-185241.37871502875
136.72311544578267	0.3206905470721874	1.1725606174114864	2.885969326861972e+17
228.07660778053693	0.3206905470721874	1.281768924238228	-57553.75110861798
228.07660778053693	0.3206905470721874	1.281768924238228	-34536.93458694707
228.07660778053693	0.3206905470721874	1.281768924238228	-94061.94124501817
229.56414453668026	0.7563642381528528	1.4351643037196284	-27939.733100528967
229.56414453668026	0.7563642381528528	1.4351643037196284	-80323.08514198531
229.56414453668026	0.7563642381528528	1.4351643037196284	2.5731167553667952e+17
229.56414453668026	0.7563642381528528	1.4351643037196284	1.6807154022665597e+17
229.56414453668026	0.7563642381528528	1.4351643037196284	-51597.57372416577
229.56414453668026	0.7563642381528528	1.4351643037196284	-34662.312816444144
229.56414453668026	0.7563642381528528	1.4351643037196284	-62893.49792086122
229.56414453668026	0.7563642381528528	1.4351643037196284	-26125.182173489775
229.56414453668026	0.7563642381528528	1.4351643037196284	-44515.02557324874
229.56414453668026	0.7563642381528528	1.4351643037196284	-34129.61580031375
238.40943621557665	0.7563642381528528	1.4351643037196284	6037541608108242.0
234.50575263170816	1.0323581876422214	0.974471546543946	-44124.72904304979

Figure 2: Exemplary illustration of an output of a single chain of a Bayesian parameter estimation stored in a `.txt` file. Each line represents one point in the parameter space with its parameter values specified in the first three columns. The calculated synthetic likelihood is given in the last column.

3 Conclusion

The project the Bayesian parameter estimation for the E-Z Reader model of eye-movement control in reading was extremely extensive. Therefore, the code base is also quite extensive. However, the project was able to provide some important insights into the most prominent model of eye-movement control in reading and provides the foundation for future research in this area. In addition, most of the code was written with the intention of applying it to other computational models of eye -movement control to allow for model comparison.

This report contains documentation of the most important programming scripts and is intended to serve as a guide. Please note that the individual programming scripts covered here have an internal documentation of the code, which has a higher level of detail than this report. However, with

its help, future participants of the project can get an easier start with the code base. There is no doubt that the code base will be extended, and scripts will be optimized. Therefore, the code documentation will also have to be adapted. Code documentations fulfill an important role in today's computational modeling research. I personally hope that this report will help others to further develop the project, in which I participated to a large extent, by fully explaining the code base to them.

References

Reichle, E. D., Pollatsek, A., Fisher, D. L., & Rayner, K. (1998). Toward a Model of Eye Movement Control in Reading. *Psychological review*, *105*(1), 125.

Reichle, E. D., Rayner, K., & Pollatsek, A. (1999). Eye Movement Control in Reading: Accounting for Initial Fixation Locations and Refixations within the EZ Reader Model. *Vision research*, *39*(26), 4403–4411.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Wood, S. N. (2010). Statistical Inference for Noisy Nonlinear Ecological Dynamic Systems. *Nature*, *466*(7310), 1102–1104.

Statement of authorship

I hereby assure that I have written this report independently, without unauthorized assistance, and have not used any sources other than those listed in the references. All passages that are taken literally or correspondingly from published or not yet published sources are marked as such. The drawings and illustrations in this work have been created by myself or are accompanied by a corresponding reference. This work has not yet been submitted to any other examination board in the same or a similar form. I further assure that the printed and digital versions of this report are identical.

Signature:

Date:
