Software Reference

Robert Mitchell: r.mitchell@ed.ac.uk

August 18, 2022

1 Introduction

The software should be generally readable but the structure may not be intuitive due to the development cycle. As such, we have prepared this document to provide a quick reference for readers that want to see the code which relates directly to equations and methods from the paper. If you have any further questions, please get in touch.

2 κ approximation

The κ approximation process is spread across different stages in different files.

2.1 Raw R estimators

The raw R estimators are computed in util/models.py. The ReliabilityModel class provides utilities for interacting with the light and wind reliability models and the basic estimators are fit to the data within the constructor. The dataset itself is stored in data/elevation_rvals.csv and data/wind_rvals.csv for light elevation and wind speed respectively.

These estimators are:

$$R_{Wind} = (0.11s + 0.43) \tag{1}$$

$$R_{Light} = \begin{cases} (-0.07\phi + 0.80), & \text{if } \phi \leq 75°, \\ (-1.26\phi + 2.31) & \text{otherwise.} \end{cases} \tag{2}$$

Recall, these estimators produce populations with low concentrations; to fix this we augment the estimators with small positive content.

2.2 Constants c_{Wind}, c_{Light}

These constants are included in world/cue.py; Lines 42-47, under the function __kappa_approximation().

The final estimators are now:

$$R_{Wind} = (0.11s + 0.43) + c_{Wind} \tag{3}$$

$$R_{Light} = \begin{cases} (-0.07\phi + 0.80) + c_{Light}, & \text{if } \phi \leq 75°, \\ (-1.26\phi + 2.31) + c_{Light} & \text{otherwise.} \end{cases} \tag{4}$$

The constants c_{Wind} and c_{Light} were tuned by hand.

2.3 κ approximation from Mardia and Jupp (2009)

As a reminder, the κ approximation expression is:

$$\hat{\kappa} \approx \begin{cases} 2R + R^3 + \frac{5}{6}R^5, & \text{if } R < 0.53 \\ \frac{1}{2(1-R)-(1-R)^2-(1-R)^3}, & \text{if } R \geq 0.85 \\ -0.4 + 1.39R + \frac{0.43}{(1-R)}, & \text{otherwise} \end{cases} \tag{5}$$

This expression can be found in code in world/cue.py; Lines 54 - 62, within the function __kappa_approximation().

3 Integration models

All integration models are available within util/integration_models.py. The model names have changed over development, a lookup is provided in Table 1.

Table 1: Lookup table documenting model names which have changed between the paper and the code.

Paper	**Code**
WVS	CMLE
NVS	NWS
BVS	BWS

Each model is represented by a Simulator class which will have two key functions; simulate_treatment() and compute_integration(). Weights are computed (as per each model specification) within the simulate_treatment() function. The final integration is computed by compute_integration().

BVS Note: As NVS is a subset of BVS (without the biases) we used a parameter flag for BVS (bias_window = -1) which makes the BVS class function as the NVS model. The BVS class also contains many references to and much logic concerning *bias windows*; the concept was removed as one of the last stages of active development, as such they feature prominently in code. They are not used for any of the results presented.

4 Evaluation process

The evaluation process is spread across two files: population_generation.py and model_evaluation.py. As the population generation stage takes a long time to run (especially instances where we search across two dimensions), populations are generated and then stored in csv files for later evaluation.

Population generation is reasonably straightforward: a simulator is created for each parameterisation and then a Treatment (see util/treatment.py) is created for each of the behavioural under which the beetles were tested. 1,000,000 simulated beetles are tested on each simlator and the results are then collected into 5° bins. This forms the probability mass function which works as the basis for the evaluation procedure.

The evaluation procedure reads in both the set of p.m.f.s (each simulator will produce its own) and the behavioural data, then evaluates the likelihood of the behavioural data against the p.m.f. for each simulator. The likelihood results are stored in dataframes and saved to csv files; our results are in the results_dfs directory.

Within model_evaluation.py, there are functions for evaluating each case we looked at (some of which did not make the final paper for various reasons). Each of these follows the same structure and each uses the same probability routine (which is reasonably well documented); an exmaple can be found on Lines 93-103 which corresponds to the equations given in the paper.

5 Mimic-data generation

The data generation routines can be found in data_production.py The mimic-data is curated such that the mean vector of the mimic populations is arbitrarily close to the mean vector of the behavioural populations. There are two separate routines, one for the cue combination data, and another for the three-day experiment. Each routine simply reads in the corresponding behavioural data which is then used to choose a suitable population size.