RaschPy

Mark Elliott

15 January 2024

Abstract

RaschPy (Elliott, 2023) is a Python package for Rasch analysis which can estimate parameters for a variety of Rasch models, generate a range of model fit statistics and output tables and graphical plots. *RaschPy* also contains simulation functionality (used for the simulations in this work). *RaschPy* is open source and free to download. Specifications are subject to change as the software is developed.

RaschPy is capable of estimating parameters and generating tables of estimates and fit statistics plus assorted plotting and simulation functionaility, for the following Rasch models:

- Simple Logistic Model (SLM), aka dichotomous Rasch model (Rasch, 1960, 1968)
- Partial Credit Model (PCM) (Masters, 1982)
- Rating Scale Model (RSM) (Andrich, 1978)
- Many-Facet Rasch Model (MFRM) (Linacre, 1994), RSM formulation
- Extended Many-Facet Rasch Models (Extended MFRM) (Elliott & Buttery, 2022b), RSM formulation

Parameter estimation uses non-iterative conditoinal pairwise methods: PAIR (Choppin, 1968, 1985) and the Eigenvector Method (EVM) (Garner & Engelhard, 2002, 2009) for SLM and PCM, and the Conditional Pairwise Adjacent Thresholds (CPAT) method (Elliott & Buttery, 2022a) for the RSM and MFRM models.

For each model, methods are grouped under the following sections:

- Preliminaries: Loading data and editing labels.
- Core functions: Functions for category probabilities, expected scores, variance and kurtosis.
- Parameter estimation: Estimating item, person and (where applicable) rater parameters and generating bootstrapped standard errors of estimation.
- Statistical output: Generating tables of estimates and fit statistics for items, persons and (where applicable) raters, plus overall test statistics, residual correlation analysis and category counts.
- Plotting functionality: Plots for a variety of item-level and test-level functions, showing theoretical curves with the option to overlay relevant features such as observed class category responses and threshold lines.

Contents

1 About *RaschPy* **8**

2 Loading data **9**

2.1 SLM . . . 9

2.1.1 loadup_slm . . . 9

2.2 PCM . . . 11

2.2.1 loadup_pcm . . . 11

2.3 RSM . . . 13

2.3.1 loadup_rsm . . . 13

2.4 MFRM . . . 15

2.4.1 loadup_mfrm_single . . . 15

2.4.2 loadup_mfrm_xlsx_tabs . . . 16

2.4.3 loadup_mfrm_multiple . . . 18

3 class SLM **21**

3.1 Preliminaries . . . 21

3.1.1 SLM . . . 21

3.1.2 rename_item . . . 22

3.1.3 rename_items_all . . . 22

3.1.4 rename_person . . . 23

3.1.5 rename_persons_all . . . 23

3.2 Core functions . . . 24

3.2.1 cat_prob . . . 24

3.2.2 exp_score . . . 24

3.2.3 variance . . . 25

3.2.4 kurtosis . . . 26

3.3 Parameter estimation . . . 27

3.3.1 calibrate . . . 27

3.3.2 std_errors . . . 28

3.3.3 abil . . . 29

3.3.4 person_abils . . . 30

3.3.5 score_abil . . . 31

3.3.6 abil_lookup_table . . . 32

3.3.7 csem . . . 34

3.4 Statistical output . . . 35

3.4.1 item_stats_df . . . 35

3.4.2 person_stats_df . . . 37

3.4.3 test_stats_df . . . 39

3.4.4 res_corr_analysis . . . 41

3.4.5 category_counts_df . . . 43

3.5 Plotting functionality . . . 44
3.5.1 Shared plotting arguments . . . 44
3.5.2 icc . . . 44
3.5.3 crcs . . . 46
3.5.4 iic . . . 47
3.5.5 tcc . . . 48
3.5.6 test_info . . . 50
3.5.7 test_csem . . . 51
3.5.8 std_residuals_plot . . . 52

4 class PCM 53
4.1 Preliminaries . . . 53
4.1.1 PCM . . . 53
4.1.2 rename_item . . . 54
4.1.3 rename_items_all . . . 54
4.1.4 rename_person . . . 55
4.1.5 rename_persons_all . . . 55
4.2 Core functions . . . 57
4.2.1 cat_prob_centred . . . 57
4.2.2 cat_prob_uncentred . . . 58
4.2.3 exp_score_centred . . . 59
4.2.4 exp_score_uncentred . . . 59
4.2.5 variance_centred . . . 60
4.2.6 variance_uncentred . . . 61
4.2.7 kurtosis_centred . . . 62
4.2.8 kurtosis_uncentred . . . 62
4.3 Parameter estimation . . . 64
4.3.1 calibrate . . . 64
4.3.2 std_errors . . . 65
4.3.3 abil . . . 67
4.3.4 person_abils . . . 68
4.3.5 score_abil . . . 69
4.3.6 abil_lookup_table . . . 70
4.3.7 csem . . . 71
4.4 Statistical output . . . 72
4.4.1 item_stats_df . . . 72
4.4.2 threshold_stats_df . . . 74
4.4.3 person_stats_df . . . 77
4.4.4 test_stats_df . . . 79
4.4.5 res_corr_analysis . . . 81
4.4.6 category_counts_df . . . 83

4.5 Plotting functionality . . . 84
4.5.1 Shared plotting arguments . . . 84
4.5.2 `icc` . . . 84
4.5.3 `crcs` . . . 86
4.5.4 `threshold_ccs` . . . 87
4.5.5 `iic` . . . 89
4.5.6 `tcc` . . . 90
4.5.7 `test_info` . . . 92
4.5.8 `test_csem` . . . 93
4.5.9 `std_residuals_plot` . . . 94

5 class RSM **95**
5.1 Preliminaries . . . 95
5.1.1 `RSM` . . . 95
5.1.2 `rename_item` . . . 96
5.1.3 `rename_items_all` . . . 96
5.1.4 `rename_person` . . . 97
5.1.5 `rename_persons_all` . . . 97
5.2 Core functions . . . 99
5.2.1 `cat_prob` . . . 99
5.2.2 `exp_score` . . . 100
5.2.3 `variance` . . . 100
5.2.4 `kurtosis` . . . 101
5.3 Parameter estimation . . . 103
5.3.1 `calibrate` . . . 103
5.3.2 `std_errors` . . . 104
5.3.3 `abil` . . . 106
5.3.4 `person_abils` . . . 107
5.3.5 `score_abil` . . . 108
5.3.6 `abil_lookup_table` . . . 109
5.3.7 `csem` . . . 110
5.4 Statistical output . . . 111
5.4.1 `item_stats_df` . . . 111
5.4.2 `threshold_stats_df` . . . 113
5.4.3 `person_stats_df` . . . 116
5.4.4 `test_stats_df` . . . 118
5.4.5 `res_corr_analysis` . . . 120
5.4.6 `category_counts_df` . . . 122

5.5 Plotting functionality . . . 123
5.5.1 Shared plotting arguments . . . 123
5.5.2 icc . . . 123
5.5.3 crcs . . . 125
5.5.4 threshold_ccs . . . 127
5.5.5 iic . . . 128
5.5.6 tcc . . . 130
5.5.7 test_info . . . 131
5.5.8 test_csem . . . 132
5.5.9 std_residuals_plot . . . 133

6 class MFRM **135**
6.1 Preliminaries . . . 135
6.1.1 MFRM . . . 135
6.1.2 rename_item . . . 136
6.1.3 rename_items_all . . . 137
6.1.4 rename_person . . . 137
6.1.5 rename_persons_all . . . 138
6.1.6 rename_rater . . . 138
6.1.7 rename_raters_all . . . 139
6.2 Core functions . . . 140
6.2.1 cat_prob . . . 140
6.2.2 exp_score . . . 143
6.2.3 variance . . . 144
6.2.4 kurtosis . . . 146
6.3 Parameter estimation . . . 149
6.3.1 calibrate . . . 149
6.3.2 calibrate_anchor . . . 151
6.3.3 std_errors . . . 153
6.3.4 abil . . . 160
6.3.5 person_abils . . . 162
6.3.6 score_abil . . . 164
6.3.7 abil_lookup_table . . . 165
6.3.8 csem . . . 167
6.4 Statistical output . . . 169
6.4.1 item_stats_df . . . 169
6.4.2 threshold_stats_df . . . 172
6.4.3 rater_stats_df . . . 175
6.4.4 person_stats_df . . . 178
6.4.5 test_stats_df . . . 181
6.4.6 item_res_corr_analysis . . . 183
6.4.7 rater_res_corr_analysis . . . 186
6.4.8 category_counts_df . . . 188

6.5 Plotting functionality . . . 190
6.5.1 Shared plotting arguments . . . 190
6.5.2 icc . . . 191
6.5.3 crcs . . . 193
6.5.4 threshold_ccs . . . 195
6.5.5 iic . . . 197
6.5.6 tcc . . . 199
6.5.7 test_info . . . 201
6.5.8 test_csem . . . 202
6.5.9 std_residuals_plot . . . 204

7 class SLM_Sim 206
7.1 Generating an SLM_Sim simulation . . . 206
7.1.1 SLM_Sim . . . 206
7.2 Customising an SLM_Sim simulation . . . 209
7.2.1 rename_item . . . 209
7.2.2 rename_items_all . . . 209
7.2.3 rename_person . . . 210
7.2.4 rename_persons_all . . . 210

8 class PCM_Sim 211
8.1 Generating a PCM_Sim simulation . . . 211
8.1.1 PCM_Sim . . . 211
8.2 Customising an PCM_Sim simulation . . . 214
8.2.1 rename_item . . . 214
8.2.2 rename_items_all . . . 214
8.2.3 rename_person . . . 215
8.2.4 rename_persons_all . . . 215

9 class RSM_Sim 216
9.1 Generating an RSM_Sim simulation . . . 216
9.1.1 RSM_Sim . . . 216
9.2 Customising an RSM_Sim simulation . . . 220
9.2.1 rename_item . . . 220
9.2.2 rename_items_all . . . 220
9.2.3 rename_person . . . 221
9.2.4 rename_persons_all . . . 221

10 class MFRM_Sim_Global **222**
10.1 Generating an MFRM_Sim_Global simulation . . . 222
10.1.1 MFRM_Sim_Global . . . 222
10.2 Customising an MFRM_Sim_Global simulation . . . 227
10.2.1 rename_item . . . 227
10.2.2 rename_items_all . . . 227
10.2.3 rename_person . . . 228
10.2.4 rename_persons_all . . . 228
10.2.5 rename_rater . . . 229
10.2.6 rename_raters_all . . . 229

11 class MFRM_Sim_Items **230**
11.1 Generating an MFRM_Sim_Items simulation . . . 230
11.1.1 MFRM_Sim_Items . . . 230
11.2 Customising an MFRM_Sim_Items simulation . . . 235
11.2.1 rename_item . . . 235
11.2.2 rename_items_all . . . 235
11.2.3 rename_person . . . 236
11.2.4 rename_persons_all . . . 236
11.2.5 rename_rater . . . 237
11.2.6 rename_raters_all . . . 237

12 class MFRM_Sim_Thresholds **238**
12.1 Generating an MFRM_Sim_Thresholds simulation . . . 238
12.1.1 MFRM_Sim_Thresholds . . . 238
12.2 Customising an MFRM_Sim_Thresholds simulation . . . 243
12.2.1 rename_item . . . 243
12.2.2 rename_items_all . . . 243
12.2.3 rename_person . . . 244
12.2.4 rename_persons_all . . . 244
12.2.5 rename_rater . . . 245
12.2.6 rename_raters_all . . . 245

13 class MFRM_Sim_Matrix **246**
13.1 Generating an MFRM_Sim_Matrix simulation . . . 246
13.1.1 MFRM_Sim_Matrix . . . 246
13.2 Customising an MFRM_Sim_Matrix simulation . . . 251
13.2.1 rename_item . . . 251
13.2.2 rename_items_all . . . 251
13.2.3 rename_person . . . 252
13.2.4 rename_persons_all . . . 252
13.2.5 rename_rater . . . 253
13.2.6 rename_raters_all . . . 253

References **254**

1 About *RaschPy*

Built-in dependencies: `itertools`, `math`, `statistics`, `string`.

Non-built-in dependencies: `numpy`, `pandas`, `matplotlib`, `scipy`, `sklearn`, `xlsxwriter`.

The only non-built-in dependency which is not part of the core `anaconda` installation is `xlsxwriter`.

Maintainer: Mark Elliott `<markelliott@cantab.net>`

URL: `https://github.com/MarkElliott999/RaschPy`

Issues: `https://github.com/MarkElliott999/RaschPy/issues`

RaschPy is offered free under an Apache 2.0 licence, but please cite when used, using the following format:

Elliott, M. (2024). *RaschPy Rasch analysis in Python.* URL: `https://github.com/MarkElliott999/RaschPy`

2 Loading data

2.1 SLM

2.1.1 `loadup_slm`

Description

Function to load dichotomously scored data for use with class `SLM`.

Usage

```
loadup_slm(filename, item_names=True, person_names=True, long=False)
```

Arguments

`filename`	The full filename, including suffix `.csv` or `.xlsx`.
`item_names`	Boolean. If `True`, the first row of data will be read as the item names. If `False`, item names will be allocated following the format `Item_1` etc. Default value is `True`.
`person_names`	Boolean. If `True`, the first column of data will be read as the person names. If `False`, item names will be allocated following the format `Person_1` etc. Default value is `True`.
`long`	Boolean. If value is `True`, data will be expected in long format; if value is `False`, data will be expected in wide format, as per toy example below. Default value is `False`.

Input file format

	Item_1	Item_2
Person_1	1	0
Person_2	0	1
Person_3	1	

Table 1: Wide format `SLM` data

Person	Item	Score
Person_1	Item_1	1
Person_1	Item_2	0
Person_2	Item_1	0
Person_2	Item_2	1
Person_3	Item_1	1
Person_3	Item_2	

Table 2: Long format `SLM` data

Input files can be in wide format, with `long=False` – the default – or long format, with `long=False`. It is not necessary to specify whether the file is a csv or xlsx file; this will be inferred from the suffix `.csv` or `.xlsx`. Examples of the required formats can be found in Table 1, which shows a toy example with three persons and two items in wide format, and Table 2, which shows the same data in long format. Missing data should be left blank, as shown for Person_3 and Item_2. Person and item names are optional, but if they are omitted, arguments `person_names=False` and/or `item_names=False` must be passed.

Returns

- pandas dataframe of responses in required format for class `SLM`
- pandas dataframe of invalid responses, where a person has not responded to any items; these responses are removed from the response dataframe.

Examples

To load a csv file in wide format with item and person labels:

```
my_slm_df, my_invalid_responses = loadup_slm('my_slm_data.csv')
```

To load a csv file in wide format with no item or person labels:

```
my_slm_df, my_invalid_responses = loadup_slm('my_slm_data.csv', item_names=False,
                                              person_names=False)
```

To load an xlsx file in long format:

```
my_slm_df, my_invalid_responses = loadup_slm('my_slm_data.xlsx', long=True)
```

2.2 PCM

2.2.1 `loadup_pcm`

Description

Function to load polytomously scored data for use with class `PCM`.

Usage

```
loadup_pcm(filename, max_score_vector=None, item_names=True, person_names=True,
          long=False)
```

Arguments

`filename`	The full filename, including suffix `.csv` or `.xlsx`.
`max_score_vector`	A vector of the maximum score for each item, as a list or numpy array. If omitted or `max_score_vector=None`, the maximum scores will be inferred from the data (passing a vector is recommended, however).
`item_names`	Boolean. If `True`, the first row of data will be read as the item names. If `False`, item names will be allocated following the format `Item_1` etc. Default value is `True`.
`person_names`	Boolean. If `True`, the first column of data will be read as the person names. If `False`, item names will be allocated following the format `Person_1` etc. Default value is `True`.
`long`	Boolean. If value is `True`, data will be expected in long format; if value is `False`, data will be expected in wide format, as per toy example below. Default value is `False`.

Input file format

	Item_1	Item_2
Person_1	1	3
Person_2	2	2
Person_3	2	

Table 3: Wide format `PCM` data

Person	Item	Score
Person_1	Item_1	1
Person_1	Item_2	3
Person_2	Item_1	2
Person_2	Item_2	2
Person_3	Item_1	2
Person_3	Item_2	

Table 4: Long format `PCM` data

Input files can be in wide format, with `long=False` – the default – or long format, with `long=False`. Data loaded in long format will be converted to wide format for analysis. It is not necessary to specify whether the file is a csv or xlsx file; this will be inferred from the suffix `.csv` or `.xlsx`. Examples of the required formats can be found in Table 3, which shows a toy example with three persons and two items in wide format, and Table 4, which shows the same data in long format. Missing data should be left blank, as shown for Person_3 and Item_2. Person and item names are optional, but if they are omitted, arguments `person_names=False` and/or `item_names=False` must be passed.

Returns

- pandas dataframe of responses in required format for class `PCM`
- pandas dataframe of invalid responses, where a person has not responded to any items; these responses are removed from the response dataframe.

Examples

To load a csv file in wide format for six items with item and person labels, passing a list of maximum scores:

```
my_pcm_df, my_invalid_responses = loadup_pcm('my_pcm_data.csv',
                                              max_score_vector=[2, 2, 3, 3, 5, 5])
```

To load a csv file in wide format with no item or person labels, passing a the variable name for a pre-existing vectors of maximum scores:

```
my_pcm_df, my_invalid_responses = loadup_pcm('my_pcm_data.csv',
                                              max_score_vector=my_max_score_vector,
                                              item_names=False, person_names=False)
```

To load an .xlsx file in long format, inferring the maximum scores from the data:

```
my_pcm_df, my_invalid_responses = loadup_pcm('my_pcm_data.xlsx', long=True)
```

2.3 RSM

2.3.1 `loadup_rsm`

Description

Function to load polytomously scored data for use with class `RSM`.

Usage

```
loadup_rsm(filename, max_score=None, item_names=True, person_names=True, long=False)
```

Arguments

`filename`	The full filename, including suffix `.csv` or `.xlsx`.
`max_score`	The maximum score available. If omitted or `max_score=None`, the maximum scores will be inferred from the data (passing a maximum score is recommended, however).
`item_names`	Boolean. If `True`, the first row of data will be read as the item names. If `False`, item names will be allocated following the format `Item_1` etc. Default value is `True`.
`person_names`	Boolean. If `True`, the first column of data will be read as the person names. If `False`, item names will be allocated following the format `Person_1` etc. Default value is `True`.
`long`	Boolean. If value is `True`, data will be expected in long format; if value is `False`, data will be expected in wide format. Default value is `False`.

Input file format

	Item_1	Item_2
Person_1	1	3
Person_2	3	2
Person_3	2	

Table 5: Wide format `RSM` data

Person	Item	Score
Person_1	Item_1	1
Person_1	Item_2	3
Person_2	Item_1	3
Person_2	Item_2	2
Person_3	Item_1	2
Person_3	Item_2	

Table 6: Long format `RSM` data

Input files can be in wide format, with `long=False` – the default – or long format, with `long=False`. Data loaded in long format will be converted to wide format for analysis. It is not necessary to specify whether the file is a csv or xlsx file; this will be inferred from the suffix `.csv` or `.xlsx`. Examples of the required formats can be found in Table 5, which shows a toy example with three persons and two items in wide format, and Table 6, which shows the same data in long format. Missing data should be left blank, as shown for Person_3 and Item_2. Person and item names are optional, but if they are omitted, arguments `person_names=False` and/or `item_names=False` must be passed.

Returns

- pandas dataframe of responses in required format for class `RSM`
- pandas dataframe of invalid responses, where a person has not responded to any items; these responses are removed from the response dataframe.

Examples

To load a csv file in wide format with item and person labels, with a maximum score of 5: `my_rsm_df, my_invalid_responses = loadup_rsm('my_rsm_data.csv’, max_score=5)`

To load a csv file in wide format with no item or person labels, passing a maximum scores stored as a variable `my_max_score`:

```
my_rsm_df, my_invalid_responses = loadup_rsm('my_rsm_data.csv',
                                             max_score_vector=my_max_score_vector,
                                             item_names=False, person_names=False)
```

To load an .xlsx file in long format, inferring the maximum score from the data:

```
my_rsm_df, my_invalid_responses = loadup_rsm('my_rsm_data.xlsx', long=True)
```

2.4 MFRM

2.4.1 `loadup_mfrm_single`

Description

Function to load polytomously scored data for use with class `MFRM` from a single csv file or .xlsx tab.

Usage

`loadup_mfrm_single(filename, max_score=None, item_names=True, long=False)`

Arguments

`filename`	The full filename, including suffix `.csv` or `.xlsx`.
`max_score`	The maximum score available. If omitted or `max_score=None`, the maximum scores will be inferred from the data (passing a maximum score is recommended, however).
`item_names`	Boolean. If `True`, the first row of data will be read as the item names. If `False`, item names will be allocated following the format `Item_1` etc. Default value is `True`.
`long`	Boolean. If value is `True`, data will be expected in long format; if value is `False`, data will be expected in wide format. Default value is `False`.

Input file format

		Item_1	Item_2
Rater_1	Person_1	1	3
Rater_1	Person_2	3	2
Rater_2	Person_1	2	2
Rater_2	Person_2	4	

Table 7: Wide format `MFRM` data, single sheet

Rater	Person	Item	Score
Rater_1	Person_1	Item_1	1
Rater_1	Person_1	Item_2	3
Rater_1	Person_2	Item_1	3
Rater_1	Person_2	Item_2	2
Rater_2	Person_1	Item_1	2
Rater_2	Person_1	Item_2	2
Rater_2	Person_2	Item_1	4
Rater_2	Person_2	Item_2	

Table 8: Long format `MFRM` data, single sheet

Input files can be in wide format, with `long=False` – the default – or long format, with `long=False`. Data loaded in long format will be converted to wide format for analysis. It is not necessary to specify whether the file is a csv or xlsx file; this will be inferred from the suffix `.csv` or `.xlsx`. Examples of the required formats can be found in Table 7, which shows a toy example with two persons and two items rated by two raters in wide format, and Table 8, which shows the same data in long format. Missing data should either be left blank, as shown for Person_2 and Item_2, rated by Rater_2, or the row may be omitted entirely if there are no observations (for any individual observation in the case of long form). Item names are optional, but if they are omitted, the argument `item_names=False` must be passed. Unlike for `SLM`, `PCM` and `RSM`, person names are mandatory for `MFRM` data.

Returns

- pandas dataframe of responses in required format for class `MFRM`
- pandas dataframe of invalid responses, where a person has not responded to any items; these responses are removed from the response dataframe.

Examples

To load a csv file in wide format with item labels, with a maximum score of 5:

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_single('my_mfrm_data.csv', max_score=5)
```

To load a csv file in wide format with no item labels, passing a maximum scores stored as a variable `my_max_score` :

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_single('my_mfrm_data.csv',
                                                      max_score=my_max_score,
                                                      item_names=False)
```

To load an .xlsx file in long format, inferring the maximum score from the data:

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_single('my_mfrm_data.xlsx', long=True)
```

2.4.2 `loadup_mfrm_xlsx_tabs`

Description

Function to load polytomously scored data for use with class `MFRM` from an xlsx file with multiple tabs: one tab for each rater.

Usage

```
loadup_mfrm_xlsx_tabs(filename, max_score, item_names=True, missing=None, long=False)
```

Arguments

`filename`	The full filename, including suffix `.xlsx`.
`max_score`	The maximum score available. If omitted or `max_score=None`, the maximum scores will be inferred from the data (passing a maximum score is recommended, however).
`item_names`	Boolean. If `True`, the first row of data will be read as the item names. If `False`, item names will be allocated following the format `Item_1` etc. Default value is `True`.
`long`	Boolean. If value is `True`, data will be expected in long format; if value is `False`, data will be expected in wide format. Default value is `False`.

Input file format

	Item_1	Item_2
Person_1	1	3
Person_3	5	4

Table 9: Wide format `MFRM` data, multiple xlsx tabs: Rater_1

	Item_1	Item_2
Person_2	3	4
Person_3	5	

Table 10: Wide format `MFRM` data, multiple xlsx tabs: Rater_2

Person	Item	Score
Person_1	Item_1	1
Person_1	Item_2	3
Person_3	Item_1	5
Person_3	Item_2	4

Table 11: Long format `MFRM` data, multiple xlsx tabs: Rater_1

Person	Item	Score
Person_2	Item_1	3
Person_2	Item_2	4
Person_3	Item_1	5

Table 12: Long format `MFRM` data, multiple xlsx tabs: Rater_2

Input files can be in wide format, with `long=False` – the default – or long format, with `long=False`. Data loaded in long format will be converted to wide format for analysis. Examples of the required formats can be found in Tables 9 and 10, which show a toy example with three persons and two items rated by two raters in wide format, and Tables 11 and 12, which show the same data in long format. The names of the tabs should match the raters, and will automatically be processed as the rater names (in particular, they should be unique). Missing data should either be left blank, as shown for Person_3 and Item_2, rated by Rater_2, or the row may be omitted entirely if there are no observations (or for any individual observation in the case of long form). Item names are optional, but if they are omitted, the argument `item_names=False` must be passed. Unlike for `SLM`, `PCM` and `RSM`, person names are mandatory for `MFRM` data.

Returns

- pandas dataframe of responses in required format for class `MFRM`
- pandas dataframe of invalid responses, where a person has not responded to any items; these responses are removed from the response dataframe.

Examples

To load an xlsx file in wide format with item labels, with a maximum score of 5:

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_xlsx_tabs('my_mfrm_data.xlsx', max_score=5)
```

To load an xlsx file in wide format with no item labels, passing a maximum scores stored as a variable `my_max_score` :

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_xlsx_tabs('my_mfrm_data.xlsx',
                                                         max_score=my_max_score,
                                                         item_names=False)
```

To load an xlsx file in long format, inferring the maximum score from the data:

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_xlsx_tabs('my_mfrm_data.xlsx', long=True)
```

2.4.3 `loadup_mfrm_multiple`

Description

Function to load polytomously scored data for use with class `MFRM` from multiple csv or xlsx files files with one file for each rater.

Usage

`loadup_mfrm_xlsx_tabs(filenames, max_score, item_names=True, missing=None, long=False)`

Arguments

`filename`	A dictionary in the format `{'Rater_1': filename_1, 'Rater_2': filename_2, ... }` with the full filenames, including suffixes `.csv` or `.xlsx`.
`max_score`	The maximum score available. If omitted or `max_score=None`, the maximum scores will be inferred from the data (passing a maximum score is recommended, however).
`item_names`	Boolean. If `True`, the first row of data will be read as the item names. If `False`, item names will be allocated following the format `Item_1` etc. Default value is `True`.
`long`	Boolean. If value is `True`, data will be expected in long format; if value is `False`, data will be expected in wide format. Default value is `False`.

Input file format

	Item_1	Item_2
Person_1	1	3
Person_3	5	4

Table 13: Wide format MFRM data, multiple files: Rater_1

	Item_1	Item_2
Person_2	3	4
Person_3	5	

Table 14: Wide format MFRM data, multiple files: Rater_2

Person	Item	Score
Person_1	Item_1	1
Person_1	Item_2	3
Person_3	Item_1	5
Person_3	Item_2	4

Table 15: Long format MFRM data, multiple files: Rater_1

Person	Item	Score
Person_2	Item_1	3
Person_2	Item_2	4
Person_3	Item_1	5

Table 16: Long format MFRM data, multiple files: Rater_2

Input files can be in wide format, with `long=False` – the default – or long format, with `long=False`. Data loaded in long format will be converted to wide format for analysis. Examples of the required formats can be found in Tables 13 and 14, which show a toy example with three persons and two items rated by two raters in wide format, and Tables 15 and 16, which show the same data in long format. Missing data should either be left blank, as shown for Person_3 and Item_2, rated by Rater_2, or the row may be omitted entirely if there are no observations (or for any individual observation in the case of long form). Item names are optional, but if they are omitted, the argument `item_names=False` must be passed. Unlike for `SLM`, `PCM` and `RSM`, person names are mandatory for `MFRM` data.

Returns

- pandas dataframe of responses in required format for class `MFRM`
- pandas dataframe of invalid responses, where a person has not responded to any items; these responses are removed from the response dataframe.

Examples

To load data from two csv files in wide format with item labels, with a maximum score of 5:

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_multiple({'Rater_1':'my_mfrm_data_1.csv',
                                                         'Rater_2':'my_mfrm_data_2.csv'},
                                                         max_score=5)
```

To load data from two csv files in wide format with no item labels, passing a maximum score stored as a variable `my_max_score` :

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_multiple({'Rater_1':'my_mfrm_data_1.csv',
                                                          'Rater_2':'my_mfrm_data_2.csv'},
                                                         max_score=my_max_score,
                                                         item_names=False)
```

To load data from a csv file and an xlsx file in long format, inferring the maximum score from the data:

```
my_mfrm_df, my_invalid_responses = loadup_mfrm_multiple({'Rater_1':'my_mfrm_data_1.csv',
                                                          'Rater_2':'my_mfrm_data_2.csv'},
                                                         long=True)
```

3 class SLM

3.1 Preliminaries

3.1.1 SLM

Description
Creates an object of the class SLM from a pandas dataframe of dichotomously scored data for analysis. No analysis can be run until an object is created.

Usage
```
SLM(dataframe, extreme_persons=True, no_of_classes=5)
```

Arguments

dataframe	pandas dataframe with items as columns (item names as column names) and persons as index (person names as row names).
extreme_persons	Boolean: if False, all persons with extreme scores (all responses correct or all responses incorrect) are removed from the response dataframe. Default is extreme_persons=True.
no_of_classes	Integer: the number of classes of persons grouped by ability for overplotting observed responses on theoretical curves. Default is no_of_classes=5

Returns
Object of class SLM. Analyses are run using methods defined on the SLM object, with results stored as attributes of the SLM object.

Several attributes of object SLM are automatically generated on its creation:

self.dataframe	pandas dataframe: Dataframe of valid responses.
self.invalid_responses	pandas dataframe: Dataframe of invalid responses (persons with no responses to any items, i.e. all missing data).
self.no_of_items	Integer: Number of items.
self.items	List: List of item names.
self.no_of_persons	Integer: Number of persons.
self.persons	List: List of person names.

Example
To create an object from a dataframe my_slm_dataframe, with 10 observed classes:

```
my_slm = SLM(my_slm_dataframe, no_of_classes=10)
```

3.1.2 `rename_item`

Description

Method to rename a single item.

Usage

`self.rename_item(old, new)`

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.dataframe` with new name.

Example

To rename an item in object `my_slm` from `Item_1` to `my_new_item_name`:

```
my_slm.rename_item('Item_1', 'my_new_item_name')
```

3.1.3 `rename_items_all`

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.dataframe` with new names.

Example

To rename all items in object `my_slm` with item names in a list stored as a variable `my_new_item_names`:

```
my_slm.rename_items_all(my_new_item_names)
```

3.1.4 `rename_person`

Description

Method to rename a single person.

Usage

`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns

Replaces specified person name in the index of `self.dataframe` with new name.

Example

To rename a person in object `my_slm` from `Person_1` to `my_new_person_name`:

```
my_slm.rename_person('Person_1', 'my_new_person_name')
```

3.1.5 `rename_persons_all`

Description Method to rename all persons.

Usage

`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns

Replaces all person names in the index of `self.dataframe` with new names.

Example

To rename all persons in object `my_slm` with person names in a list stored as a variable `my_new_person_names`:

```
my_slm.rename_persons_all(my_new_person_names)
```

3.2 Core functions

3.2.1 `cat_prob`

Description

Category probability function which calculates the probability $P(X_{ni} = k)$ of scoring k, with $k \in \{0,\ 1\}$ from person ability and item difficulty. For a person n with ability β_n attempting an item i with difficulty δ_i, the probability of obtaining a score of k is given by:

$$P(X_{ni} = k) = \frac{e^{k(\beta_n - \delta_i)}}{1 + e^{\beta_n - \delta_i}}$$

Usage

`self.cat_prob(ability, difficulty, category)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty
`category`	Integer: Response category k, with $k \in \{0,\ 1\}$.

Returns

Float: probability of obtaining score k.

Example

To obtain the probability of a person of ability 0.5 scoring 0 on an item of difficulty 0 and store the result as a variable `my_cat_prob`:

```
my_cat_prob = self.cat_prob(0.5, 0, 0)
```

3.2.2 `exp_score`

Description

Expected score function which calculates the expected score $E(X_{ni})$ from person ability and item difficulty. The expected score is given by:

$$E(X_{ni}) = \sum_{k=0}^{1} kP(X_{ni} = k)$$

where $P(X_{ni} = k)$ is as described in Section 3.2.1.

In the dichotomous case, this is also the probability of obtaining a correct response; for a person n with ability β_n attempting an item i with difficulty δ_i,the equation reduces to the most familiar formulation of the SLM:

$$E(X_{ni}) = \frac{e^{\beta_n - \delta_i}}{1 + e^{\beta_n - \delta_i}}$$

Usage

`self.exp_score(ability, difficulty)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty

Returns

Float: expected score (also the probability of obtaining a correct response).

Example

To obtain the expected score for a person of ability 0.5 attempting an item of difficulty 0 and store the result as a variable `my_exp_score`:

```
my_exp_score = self.exp_score(0.5, 0)
```

3.2.3 `variance`

Description

Variance function which calculates the variance of the score $V(X_{ni})$ from person ability and item difficulty. The variance is given by:

$$V(X_{ni}) = \sum_{k=0}^{1} P(X_{ni} = k)(k - E(X_{ni}))^2$$

where $P(X_{ni} = k)$ and $E(X_{ni})$ are as described in Sections 3.2.1 and 3.2.2 respectively.

In the dichotomous case, since this is a Bernoulli variable, this reduces to $p(1-p)$, where p is the probability of obtaining a correct response; for a person n with ability β_n attempting an item i with difficulty δ_i, the equation is given by:

$$E(X_{ni}) = \frac{e^{\beta_n - \delta_i}}{(1 + e^{\beta_n - \delta_i})^2}$$

The variance is also both the Fisher information for the response and the first partial differential of the expected score function with respect to person ability.

Usage

`self.variance(ability, difficulty)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty

Returns
Float: variance (also the Fisher information provided by the response).

Example
To obtain the variance for a person of ability 0.5 attempting an item of difficulty 0 and store the result as a variable `my_variance`:

```
my_variance = self.variance(0.5, 0)
```

3.2.4 `kurtosis`

Description
Kurtosis function which calculates the kurtosis of the score $\kappa(X_{ni})$ from person ability and item difficulty. The variance is given by:

$$\kappa(X_{ni}) = \sum_{k=0}^{1} P(X_{ni} = k)(k - E(X_{ni}))^4$$

where $P(X_{ni} = k)$ and $E(X_{ni})$ are as described in Sections 3.2.1 and 3.2.2 respectively.

Usage

```
self.kurtosis(ability, difficulty)
```

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty

Returns
Float: kurtosis

Example
To obtain the kurtosis for a person of ability 0.5 attempting an item of difficulty 0 and store the result as a variable `my_kurtosis`:

```
my_kurtosis = self.kurtosis(0.5, 0)
```

3.3 Parameter estimation

3.3.1 `calibrate`

Description

Produces item difficulty estimates using PAIR (Choppin, 1968, 1985), eigenvector method (Garner & Engelhard, 2002) or related conditional pairwise methods.

Usage

`self.calibrate(constant=0.1, method='cos', matrix_power=3, log_lik_tol=0.000001)`

Arguments

`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method. Ignored for other methods.

Returns

Attribute `self.diffs`: a pandas series of item difficulty estimates with the item names as keys and estimates as values.

Examples

To generate a set of estimates using the cosine similarity method, with additive smoothing constant of 0.1:

```
self.calibrate()
```

To generate a set of estimates using the log-likelihood method, with matrix raised to power 7 and a convergence stopping criterion of 0.00000001:

```
self.calibrate(method='log-lik', matrix_power=7, log_lik_tol=0.00000001)
```

3.3.2 `std_errors`

Description

Produces bootstrapped estimates for the standard errors of item difficulty estimates.

Usage

```
self.std_errors(interval=None, no_of_samples=100, constant=0.1, method='cos',
                matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval. Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.
`no_of_samples`	Integer: Number of bootstrap samples to generate. More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method. Ignored for other methods.

Returns

`self.item_se`	pandas series: tem names as keys and item standard errors as values.
`self.item_bootstrap`	pandas dataframe: Contains the full bootstrap results, with a row for each bootstrap sample and a column for each item estimate.

If `interval` is specified, also returns:

`self.item_low`	Float: The lower bound of the specified interval.
`self.item_high`	Float: The upper bound of the specified interval.

Example

To generate item standard errors with a 95% interval from 200 samples:

```
self.std_errors(interval=0.95, no_of_samples=200)
```

Modifications to the estimation method are discussed in Section 3.3.1.

3.3.3 `abil`

Description

Generates an ability estimate for a person using the Newton-Raphson method to produce a maximum likelihood estimate, with optional Warm bias correction (Warm, 1989).

Usage

```
self.abil(person, items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
          ext_score_adjustment=0.5)
```

Arguments

`person`	String: The person name for the ability being estimated.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Float: person ability estimate.

Examples
To generate a person ability estimate for `Person_1` using the default settings and store the result as a variable, `my_person_ability`:

`my_person_ability = my_person_ability = self.abil('Person_1')`

To generate an MLE person ability estimate without Warm bias correction for `Person_1` based on the first three items and store the result as a variable, `my_person_ability`:

`my_person_ability = self.abil('Person_1', ['Item_1', 'Item_2', 'Item_3'], warm_corr=False)`

3.3.4 `person_abils`

Description
Generates ability estimates for all persons using the Newton-Raphson method to produce maximum likelihood estimates, with optional Warm bias correction (Warm, 1989).

Usage
```
self.person_abils(items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
                  ext_score_adjustment=0.5)
```

Arguments

`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.

Arguments continue on the next page.

Arguments (continued)

`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns

Attribute `self.person_abilities`: pandas series with person names as keys and ability estimates as values.

Example

To generate a set of person ability estimates with Warm bias correction:

```
self.person_abils()
```

To generate a set of person ability estimates without Warm bias correction, on a subset of the first three items only:

```
self.person_abils(items=['Item_1', 'Item_2', 'Item_3'], warm_corr=False)
```

3.3.5 `score_abil`

Description

Generates an ability estimate for a given raw scoreon responses to a given set of items using the Newton-Raphson method to produce a maximum likelihood estimate, with optional Warm bias correction (Warm, 1989).

Usage

```
self.score_abil(score, items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
                ext_score_adjustment=0.5)
```

Arguments

`score`	Integer: The raw score for which ability is being estimated.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.

Arguments continue on the next page.

Arguments (continued)

`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns

pandas series with raw scores as keys and person ability estimates as values.

Examples

To generate an ability estimate for a score of 10 on all items, with Warm bias correction, and store the result as a variable, `my_score_ability`:

```
my_score_ability = self.score_abil(10)
```

To generate an ability estimate for a score of 10 on a subset of items saved as a variable `my_items`, without Warm bias correction, and store the result as a variable, `my_score_ability`:

```
my_score_ability = self.score_abil(10, items=my_items, warm_corr=False)
```

3.3.6 `abil_lookup_table`

Description

Generates a lookup table of ability estimates corresponding to all available raw scores on a set of items with no missing responses, using the Newton-Raphson method to produce maximum likelihood estimates and with optional Warm bias correction (Warm, 1989).

Usage

```
self.abil_lookup_table(items=None, ext_scores=True, warm_corr=True, tolerance=1e-07,
                       max_iters=100, ext_score_adjustment=0.5)
```

Arguments

`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`ext_scores`	Boolean: If `True`, ability estimates for extreme scores (all correct/all incorrect) will be generated using the `ext_score_adjustment` argument. Default is `ext_scores=True`.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Attribute `self.abil_table`: pandas series with raw scores as keys and corresponding ability estimates as values.

Examples
To generate an ability lookup table for all items, including extreme scores, with Warm bias correction:

```
self.abil_lookup_table()
```

To generate an ability lookup table for a subset of items saved as a variable `my_items`), without extreme scores and without Warm bias correction:

```
self.abil_lookup_table(items=my_items, ext_scores=False)
```

3.3.7 `csem`

Description

Calculates conditional standard error of measurement for a person.

Usage

```
self.csem(person, abilities=None, items=None)
```

Arguments

`person`	Person name.
`abilities`	pandas series (or dictionary) with person names as keys and abilities as values. If `None`, uses `self.person_abilities`, automatically generating if necessary. Default is `self.person_abilities=None`.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.

Returns

Float: conditional standard error of measurement for ability estimate.

Examples

To generate the CSEM for `Person_1` on all items and save the result as a variable, `my_csem`:

```
my_csem = self.csem('Person_1')
```

To generate the CSEM for a raw score of 3 on a subset of items saved to a variable `my_items` and save the result as a variable, `my_csem`:

```
my_csem = self.csem(3, abilities=self.abil_table, items=my_items)
```

where `self.abil_table` is the output from running `self.abil_lookup_table(items=my_items)`, as described in Section 3.3.6.

3.4 Statistical output

3.4.1 `item_stats_df`

Description

Produces a table of item statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.item_stats_df(full=False, zstd=False, disc=Flase, point_measure_corr=False, dp=3, warm_corr=True,
                   tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5, method='cos',
                   constant=0.1, no_of_samples=100, interval=None)
```

Arguments

`full`	Boolean: If `True`, a full table with all available statistics is produced. Default is `full=False` for a summary table with the most commonly reported statistics.
`zstd`	Boolean: If `True`, infit and outfit standardised z-scores are reported alongside mean square statistics. Default is `zstd=False`. Automatically `True` if `full=True`.
`disc`	Boolean: If `True`, item discrimination is reported. The discrimination of the empirical item slope relative to the ideal logistic ogive, with 1 perfect, greater than 1 showing overfit and less than 1 showing underfit; discrimination is similar to the 2PL IRT discrimination parameter (Linacre, 2023), but is a descriptive statistic in the SLM rather than an item parameter.
`point_measure_corr`	Boolean: If `True`, point-biserial correlation (Kornbrot, 2014) between person ability and item score is reported, together with the expected value of the point-measure biserial correlation for an ideal item. Default is `point_measure_corr=False`. Automatically `True` if `full=True`.
`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.

Arguments continue on the next page.

Arguments (continued)

`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during item estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for item estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute `self.item_stats`, a pandas dataframe with one row for each item and the following columns:

`Estimate`	Item difficulty estimate.
`SE`	Bootstrapped standard error of item difficulty estimate.
`2.5%`	Lower bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 2.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `5%`.

Returns continue on the next page.

Returns (continued)

`97.5%`	Upper bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 97.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `95%`.
`Count`	Count of responses.
`Facility`	Item facility: proportion of correct responses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score. Only produced if `zstd=True`.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score. Only produced if `zstd=True`.
`Discrim`	Item discrimination. Only produced if `disc=True`.
`PM corr`	Point-biserial correlation. Only produced if `point_measure_corr=True`.
`Exp PM corr`	Expected value of point-measure biserial correlation. Only produced if `point_measure_corr=True`.

Examples

To produce a summary `self.item_stats` table with the most commonly reported statistics:
`self.item_stats_df()`

To produce a full `self.item_stats` table with all statistics:
`self.item_stats_df(full=True)`

To produce an `self.item_stats` table with with the most commonly reported statistics, plus standardised z-scores for infit and outfit:
`self.item_stats_df(zstd=True)`

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

3.4.2 `person_stats_df`

Description

Produces a table of person statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.person_stats_df(full=False, rsem=False, dp=3, warm_corr=True, tolerance=1e-07,
                     max_iters=100, ext_score_adjustment=0.5, method='cos', constant=0.1)
```

Arguments

`full`	Boolean: If `True`, a full table with all available statistics is produced. Default is `full=False` for a summary table with the most commonly reported statistics.
`rsem`	Boolean: If `True`, realistic standard error of measurement (RSEM), which takes into account for item misfit (Wright, 1996), is reported alongside the conditional standard error of measurement (CSEM). Default is `rsem=False`. Automatically `True` if `full=True`.
`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during item estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for item estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.

Arguments continue on the next page.

Arguments (continued)

`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute `self.person_stats`, a pandas dataframe with one row for each person and the following columns:

`Estimate`	Item difficulty estimate.
`CSEM`	Conditional standard error of measurement for person ability estimate.
`RSEM`	Realistic standard error of measurement for person ability estimate. Only produced if `rsem=True`
`Score`	Number of correct responses.
`Max score`	Maximum available score (number of items attempted).
`p`	Proportion of correct repsonses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score.

Examples

To produce a summary `self.person_stats` table with the most commonly reported statistics:
`self.person_stats_df()`

To produce a full `self.person_stats` table with all statistics:
`self.person_stats_df(full=True)`

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

3.4.3 `test_stats_df`

Description

Produces a table of test-level statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.test_stats_df(dp=3, warm_corr=True, tolerance=1e-07, max_iters=100,
                   ext_score_adjustment=0.5, method='cos', constant=0.1)
```

Arguments

`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during item estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for item estimation (see section 3.3.1). Default value is `constant=0.1`.

Returns

Attribute `self.test_stats`, a pandas dataframe with two columns, `Items` and `Persons` and rows for a range of descriptive statistics describing the distributions of the estimates and different statistics related to reliability – these statistics describe the suitability of the data for estimating and differentiating the parameters, rather than properties of the parameters themselves. The statistics are:

`Mean`	The mean of the estimates.
`SD`	The standard deviation of the estimates.
`Separation ratio`	The separation ratio (Wright, 1996; Wright & Masters, 1982), which is the standard deviation of person abilities reported as a ratio of standard error units. For persons: $$G_p = \sigma_p / \sqrt{\sum_n SE_n^2}$$ where σ_p is the variance of the person estimates and SE_n is the RSEM (see Section 3.4.2) for person n. The formula is symmetrical for items, substituting the standard error of estimation for RSEM.
`Strata`	The number of statistically distinct levels of either person ability or item difficulty as strata with centers three measurement errors apart (Wright & Masters, 1982:106). For persons: $$H_p = (4G_p + 1)/3$$ with symmetrical results for items.
`Reliability`	A Rasch-specific reliability statistic (Wright, 1996), derived from PSI and which is a Rasch-specific reliability statistic similar to Cronbach's Alpha (Cronbach, 1951), and which may be interpreted the same way – as the proportion of variance of the estimates which stems from variation in ability or difficulty rather than estimation error. For persons: $$R_p = G_p^2/(1 + G_p^2)$$ with symmetrical results for items.

Example

To produce a `self.test_stats` table:

```
self.test_stats_df()
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

3.4.4 `res_corr_analysis`

Description

Analysis of correlations of standardised residuals to tests for violations of local item interdependence and unidimensionality requirements.

Usage

```
self.res_corr_analysis(warm_corr=True, tolerance=1e-07, max_iters=100,
                       ext_score_adjustment=0.5, constant=0.1, method='cos',
                       matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for generation of item difficulty estimates (see Section 3.3.1). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) for generation of item difficulty estimates (see Section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) is raised for generation of item difficulty estimates (see Section 3.3.1). Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for generation of item difficulty estimates (see Section 3.3.1). Ignored for other methods.

Returns

For tests of violation of the requirement for local item independence (Andrich & Kreiner, 2010; Marais, 2012):

`self.residual_correlations`	A pandas dataframe of pairwise correlations between item standardised residuals.

For tests of violation of the requirement for unidimensionality based on principal component analysis of the standardised residual correlations (Pallant & Tennant, 2007; Smith, 2002):

`self.eigenvectors`	The eigenvectors of the standardised residual correlations matrix.
`self.eigenvalues`	The eigenvalues corresponding to the eigenvectors.
`self.variance_explained`	The variance explained by each principal component.
`self.loadings`	The loading of each item onto each of the principal components, for the the first of which large loadings ('large'typically interpreted as > 0.4 or < -0.4) may be interpreted as representing the presence of significant dimensionality, in analogy to factor analysis (**<empty citation>**).

Example
To produce a residual correlation analysis:
`self.res_corr_analysis()`

Arguments may be used to alter parameters of item difficulty and/or person ability estimation.

3.4.5 `category_counts_df`

Description
Produces a table of counts of scores in each category, plus responses and missing responses, for each item.

Usage
`self.category_counts_df()`

Arguments None

Returns
Attribute `self.category_counts`, a pandas dataframe of category counts with one row per item and one column per response category, plus total responses per item and missing responses per item.

Example
To produce a dataframe of category counts:
`self.category_counts_df()`

3.5 Plotting functionality

3.5.1 Shared plotting arguments

All the plotting methods described in this section share a set of arguments which may be used to customise the appearance of the plot or save the plot to file automatically. These arguments are:

`title`	String: Title for the plot, to appear in the image. Default is `title=None`.
`xmin`	Float: Minimum displayed point on x-axis, in logits. Default is `xmin=-5`.
`xmax`	Float: Maximum displayed point on x-axis, in logits. Default is `xmax=5`.
`plot_style`	String: Plot style to use. Available styles are `'white'`, producing a plot on a white background, and `'dark'`, producing a plot on a grey background. The styles correspond to Seaborn (Waskom, 2021) styles `whitegrid` and `darkgrid`. Default is `plot_style='white'`.
`palette`	String: Controls colours of lines and overplotted class intervals dots. Options are `'dark blue'`, `'light blue'`, `'dark red'`, `'light red'`, `'dark green'`, `'light green'`, `'dark grey'`, `'light grey'`, `'dark multi'` and `'light multi'`. Default is `palette='dark blue'`. Not used for `std_residual_plot`.
`black`	Boolean: If `True`, the plot will be rendered in black and white. Default is `black=False`.
`font`	String: The font to use in the plot. Default is `font='Times'`.
`title_font_size`	Float: The size of the title font in points. Default is `title_font_size=15`.
`axis_font_size`	Float: The size of the axis label font in points. Default is `axis_font_size=15`.
`labelsize`	Float: The size of the axis tick label font in points. Default is `labelsize=15`.
`filename`	String: The filename for the saved plot, with no suffix for format. If `None`, no file will be saved. Default is `filename=None`.
`file_format`	The format of the file: `png`, `jpg` or `svg`. Default is `file_format=png`.
`dpi`	The resolution of the plot in dpi (dots per inch) – higher resolution plots are better quality but have larger file sizes. Default is `dpi=300`.

3.5.2 `icc`

Description

Plots the item characteristic curves (or item response function) for an item: person ability on the x-axis against expected score on the y-axis. Options to plot observed responses, item threshold line and lines showing abilities corresponding to specified expected scores, and to highlight a specified response category.

Usage

```
self.icc(item, obs=False, no_of_classes=5, thresh_line=False, score_lines=None,
        score_labels=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`obs`	Boolean: If `True`, mean observed scores for each of the ordered response categories will be plotted against the mean ability of the corresponding response class. Default is `obs=False`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`thresh_line`	Boolean: If `True`, a vertical line showing the threshold corresponding to the item difficulty (the threshold between the ability regions for which 0 or 1 are the most probable score) will be plotted. Default is `thresh_line=False`.
`score_lines`	List of floats between 0 and 1: Each float in the list represents an expected score, for which a horizontal line from the y-axis to where it intersects the item characteristic curve will be plotted, and from there a vertical line down to the x-axis will be plotted to show the ability corresponding to the expected score. Default is `score_lines=None`.
`score_labels`	Boolean: If `True`, scores and abilities corresponding to arguments passed to `score_lines` will be labelled on the plot. Default is `score_labels=False`.
`cat_highlight`	Integer: Passing 0 or 1 will highlight the range of abilities for which the selected score is the most probable response (all abilities to one side of the item difficulty. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section 3.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot a basic item characteristic curve for `Item_1` and store the output as a variable `my_icc_plot` and save it to file as `my_icc_plot.png`:

```
my_icc_plot = self.icc('Item_1', filename=my_icc_plot)
```

To plot an item characteristic curve for `Item_1` with observed responses for 8 response classes and store the output as a variable `my_icc_plot`:

```
my_icc_plot = self.icc('Item_1', obs=True, no_of_classes=8)
```

To plot an item characteristic curve for `Item_1` with a threshold line and highlighted zero category, and store the output as a variable `my_icc_plot`:

```
my_icc_plot = self.icc('Item_1', thresh_line=True, cat_highlight=0)
```

To plot an item characteristic curve for `Item_1` with lines showing the abilities corresponding to expected scores of 0.3 and 0.7, with the expected score and corresponding ability labelled, and store the output as a variable `my_icc_plot`:

```
my_icc_plot = self.icc('Item_1', score_lines=[0.3, 0.7], score_labels=True)
```

3.5.3 `crcs`

Description

Plots category response curves for an item: person ability on the x-axis against expected the probability of obtaining a score in each category (0 or 1) on the y-axis. Options to plot observed proportions and item threshold line, and to highlight a specified response category.

Usage

`self.crcs(item, obs=None, no_of_classes=5, thresh_line=False, cat_highlight=None)`

Arguments

`item`	String: The name of the item to plot.
`obs`	List: List of integers (0 or 1). For each value, mean observed proportions in each ordered response category scoring in that category are plotted against the mean ability of the corresponding response class. Default is `obs=None`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`thresh_line`	Boolean: If `True`, a vertical line showing the threshold corresponding to the item difficulty (the threshold between the ability regions for which 0 or 1 are the most probable score) will be plotted. Default is `thresh_line=False`.
`cat_highlight`	Integer: Passing 0 or 1 will highlight the range of abilities for which the selected score is the most probable response (all abilities to one side of the item difficulty. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section 3.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot basic category response curves for `Item_1` and store the output as a variable `my_crcs_plot` and save it to file as `my_crcs_plot.png`:

```
my_crcs_plot = self.crcs('Item_1', filename=my_crcs_plot)
```

To plot category response curves for `Item_1` with observed response proportions for category 0 for 8 response classes and store the output as a variable `my_crcs_plot`:

```
my_crcs_plot = self.crcs('Item_1', obs=[0], no_of_classes=8)
```

To plot category response curves for `Item_1` with a threshold line and highlighted zero category, and store the output as a variable `my_crcs_plot`:

```
my_crcs_plot = self.crcs('Item_1', thresh_line=True, cat_highlight=0)
```

3.5.4 `iic`

Description

Plots the item information curve for an item: person ability on the x-axis against Fisher information on the y-axis. Options to plot item threshold line and lines showing Fisher information corresponding to specified abilities, and to highlight a specified response category.

Usage

```
self.iic(item, ymax=None, thresh_line=False, point_info_lines=None, point_info_labels=False,
        cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`ymax`	Float: The maximum value to show on the y-axis. If `None`, will infer, plotting a maximum of 1.1 times the maximum item information. Default is `ymax=None`
`thresh_line`	Boolean: If `True`, a vertical line showing the threshold corresponding to the item difficulty, which the ability for which scores of 0 and 1 are equally probable, will be plotted. Default is `thresh_line=False`.
`point_info_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the item information curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the Fisher information corresponding to the ability. Default is `point_info_lines=None`.

Arguments continue on the next page.

Arguments (continued)

`point_info_labels`	Boolean: If `True`, abilities and Fisher information corresponding to arguments passed to `point_info_lines` will be labelled on the plot. Default is `point_info_labels=False`.
`cat_highlight`	Integer: Passing 0 or 1 will highlight the range of abilities for which the selected score is the most probable response (all abilities to one side of the item difficulty. Default is `cat_highlight=None` (no category highlighted).
`ymax`	The maximum point displayed on the y-axis, in Fisher information.

Additional arguments to customise the appearance of the plot are detailed in Section 3.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot a basic item information curve for `Item_1` and store the output as a variable `my_iic_plot` and save it to file as `my_iic_plot.png`:

```
my_iic_plot = self.iic('Item_1', filename='my_iic_plot')
```

To plot an item information curve for `Item_1` with a threshold line and highlighted zero category, and store the output as a variable `my_iic_plot`:

```
my_iic_plot = self.iic('Item_1', thresh_line=True, cat_highlight=0)
```

To plot an item information curve for `Item_1` with lines showing the Fisher information corresponding to abilities of -0.3 and 0.7, with the ability and corresponding Fisher information labelled, and store the output as a variable `my_iic_plot`:

```
my_iic_plot = self.icc('Item_1', point_info_lines=[-0.3, 0.7], point_info_labels=True)
```

3.5.5 `tcc`

Description

Plots the test characteristic curve (or test response function) for a set of items: person ability on the x-axis against expected score on the y-axis. Options to plot observed responses and lines showing abilities corresponding to specified expected scores.

Usage

```
self.tcc(items=None, obs=False, no_of_classes=5, score_lines=None, score_labels=False)
```

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`obs`	Boolean: If `True`, mean observed scores for each of the ordered response categories will be plotted against the mean ability of the corresponding response class. Default is `obs=False`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`score_lines`	List of floats between 0 and 1: Each float in the list represents an expected score, for which a horizontal line from the y-axis to where it intersects the item characteristic curve will be plotted, and from there a vertical line down to the x-axis will be plotted to show the ability corresponding to the expected score. Default is `score_lines=None`.
`score_labels`	Boolean: If `True`, scores and abilities corresponding to arguments passed to `score_lines` will be labelled on the plot. Default is `score_labels=False`.

Additional arguments to customise the appearance of the plot are detailed in Section 3.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot a basic test characteristic curve for all items and store the output as a variable `my_tcc_plot` and save it to file as `my_tcc_plot.png`:

```
my_tcc_plot = self.tcc(filename=my_tcc_plot)
```

To plot a test characteristic curve for `Item_1` for a subset of items stored as a list **`my_item_list`**, with observed responses for 8 response classes and store the output as a variable `my_tcc_plot`:

```
my_tcc_plot = self.tcc(obs=True, no_of_classes=8)
```

To plot a test characteristic curve for `Item_1` for all items with lines showing the abilities corresponding to expected scores of 13 and 20, with the expected score and corresponding ability labelled, and store the output as a variable `my_tcc_plot`:

```
my_tcc_plot = self.tcc(score_lines=[13, 20], score_labels=True)
```

3.5.6 test_info

Description

Plots the test information curve: person ability on the x-axis against total Fisher information on the y-axis. Option to plot lines showing Fisher information corresponding to specified abilities.

Usage

```
self.test_info(items=None, ymax=None, point_info_lines=None, point_info_labels=False)
```

Arguments

items	List: The names of the items to be used in the plot. If None, the full set of items will be used. Default is items=None.
ymax	Float: The maximum value to show on the y-axis. If None, will infer, plotting a maximum of 1.1 times the maximum test information. Default is ymax=None
midrule point_info_lines	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the item information curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the total Fisher information corresponding to the ability. Default is point_info_lines=None.
point_info_labels	Boolean: If True, abilities and total Fisher information corresponding to arguments passed to point_info_lines will be labelled on the plot. Default is point_info_labels=False.
ymax	The maximum point displayed on the y-axis, in Fisher information.

Additional arguments to customise the appearance of the plot are detailed in Section 3.5.1.

Returns

matplotlib image. If a filename argument is passed, also saves the image to file.

Examples

To plot a basic test information curve and store the output as a variable my_test_info_plot and save it to file as my_test_info_plot.png:

```
my_test_info_plot = self.test_info(filename='my_test_info_plot')
```

To plot a test information curve for a subset of items stored as a list **my_item_list** and store the output as a variable my_test_info_plot:

```
my_test_info_plot = self.test_info(items=my_item_list)
```

To plot a test information curve with lines showing the total Fisher information corresponding to abilities of -0.3 and 0.7, with the ability and corresponding total Fisher information labelled, and store the output as a variable `my_test_info_plot`:

```
my_test_info_plot = self.test_info(point_info_lines=[-0.3, 0.7], point_info_labels=True)
```

3.5.7 `test_csem`

Description

Plots the test conditional standard error of measurement (CSEM) curve: person ability on the x-axis against CSEM (in logits) on the y-axis. Option to plot lines showing CSEM corresponding to specified abilities.

Usage

```
self.test_csem(items=None, ymax=5, point_csem_lines=None, point_csem_labels=False, ymax=5)
```

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`ymax`	Float: The maximum value to show on the y-axis, in logits. Default is `ymax=5`
`csem_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the CSEM curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the CSEM corresponding to the ability. Default is `csem_lines=None`.
`point_csem_labels`	Boolean: If `True`, abilities and CSEM corresponding to arguments passed to `point_csem_lines` will be labelled on the plot. Default is `point_csem_labels=False`.

Additional arguments to customise the appearance of the plot are detailed in Section 3.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot a basic CSEM curve and store the output as a variable `my_test_csem_plot` and save it to file as `my_test_csem_plot.png`:

```
my_test_csem_plot = self.test_csem(filename='my_test_csem_plot')
```

To plot a CSEM curve for a subset of items stored as a list `my_item_list` and store the output as a variable `my_test_csem_plot`:

```
my_test_csem_plot = self.test_csem(items=my_item_list)
```

To plot a CSEM curve with lines showing the CSEM corresponding to abilities of -0.3 and 0.7, with the ability and corresponding CSEM labelled, and store the output as a variable `my_test_csem_plot`:

```
my_test_csem_plot = self.test_csem(point_csem_lines=[-0.3, 0.7], point_csem_labels=True)
```

3.5.8 `std_residuals_plot`

Description
Plots histogram of standardised residuals, with optional overplotting of standard Normal distribution.

Usage
`self.std_residuals_plot(items=None, bin_width=0.5, normal=False)`

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`bin_width`	Float: The width of the histogram bins along the x-axis. Default is `bin_width=0.5`.
`normal`	Boolean: If `True`, plots a standard normal distribution over the standardised residual histogram for comparison. Default is `normal=False`.

Additional arguments to customise the appearance of the plot are detailed in Section 3.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot and display a basic standardised residuals histogram and save it to file as `my_std_residuals_plot.png`:

```
self.std_residuals_plot(filename='my_std_residuals_plot')
```

To plot and display a standardised residuals histogram with bin width 1, with standard normal curve:

```
self.std_residuals_plot(bin_width=1, normal=True)
```

To plot and display a standardised residuals histogram on a subset of items stored as a list in a variable `my_item_list`:

```
self.std_residuals_plot(items=my_item_list)
```

4 class PCM

4.1 Preliminaries

4.1.1 PCM

Description

Creates an object of the class PCM from a pandas dataframe of polytomously scored data of items with individual Rasch-Andrich threshold structures, and potentially different maximum scores, for analysis. No analysis can be run until an object is created.

Usage

```
PCM(dataframe, max_score_vector=None, extreme_persons=True, no_of_classes=5)
```

Arguments

dataframe	pandas dataframe with items as columns (item names as column names) and persons as index (person names as row names).
max_score_vector	List or 1-dimensional numpy array of integers: List with the maximum possible score for each item. If no vector is passed, max_score_vector will be inferred from the data, although passing an argument is recommended. Default is max_score_vector=None.
extreme_persons	Boolean: if False, all persons with extreme scores (all responses correct or all responses incorrect) are removed from the response dataframe. Default is extreme_persons=True.
no_of_classes	Integer: the number of classes of persons grouped by ability for overplotting observed responses on theoretical curves. Default is no_of_classes=5

Returns

Object of class PCM. Analyses are run using methods defined on the PCM object, with results stored as attributes of the PCM object.

Several attributes of object PCM are automatically generated on its creation:

`self.dataframe`	pandas dataframe: Dataframe of valid responses.
`self.invalid_responses`	pandas dataframe: Dataframe of invalid responses (persons with no responses to any items, i.e. all missing data).
`self.max_score_vector`	List or 1-dimensional numpy array of integers: List with the maximum possible score for each item.
`self.no_of_items`	Integer: Number of items.

Returns continue on the next page.

Returns (continued)

`self.items`	List: List of item names.
`self.no_of_persons`	Integer: Number of persons.
`self.persons`	List: List of person names.

Example

To create an object from a dataframe `my_pcm_dataframe`, with four items with maximum scores of 3, 3, 5 and 5, and with 10 observed classes:

```
my_rsm = RSM(my_rsm_dataframe, max_score_vector=[3, 3, 5, 5], no_of_classes=10)
```

4.1.2 `rename_item`

Description

Method to rename a single item.

Usage

```
self.rename_item(old, new)
```

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.dataframe` with new name.

Example

To rename an item in object `my_pcm` from `Item_1` to `my_new_item_name`:

```
my_pcm.rename_item('Item_1', 'my_new_item_name')
```

4.1.3 `rename_items_all`

Description

Method to rename all items.

Usage

```
self.rename_items_all(new_names)
```

Arguments

`new_names`	List of new item names as strings

Returns
Replaces all item names in the columns of `self.dataframe` with new names.

Example
To rename all items in object `my_pcm` with item names in a list stored as a variable `my_new_item_names`:
`my_pcm.rename_items_all(my_new_item_names)`

4.1.4 `rename_person`

Description
Method to rename a single person.

Usage
`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns
Replaces specified person name in the index of `self.dataframe` with new name.

Example
To rename a person in object `my_pcm` from `Person_1` to `my_new_person_name`:
`my_pcm.rename_person('Person_1', 'my_new_person_name')`

4.1.5 `rename_persons_all`

Description Method to rename all persons.

Usage
`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns

Replaces all person names in the index of `self.dataframe` with new names.

Example

To rename all persons in object `my_pcm` with person names in a list stored as a variable `my_new_person_names`:
`my_pcm.rename_persons_all(my_new_person_names)`

4.2 Core functions

4.2.1 `cat_prob_centred`

Description

Category probability function which calculates the probability $P(X_{ni} = k)$ of scoring k, with $k \in \{0,\ m\}$, where m is the maximum score, from person ability, central item difficulty and Rasch-Andrich thresholds (see Section 4.2.2 below for the uncentred thresholds formulation). For a person n with ability β_n attempting an item i with central item difficulty δ_i and Rasch-Andrich thresholds $\{\tau_{i0}, ..., \tau_{im}\}$, the probability of obtaining a score of k is given by:

$$P(X_{ni} = k) = \frac{e^{k(\beta_n - \delta_i) - \sum_{t=0}^{k} \tau_{it}}}{\sum_{k=0}^{m} e^{k(\beta_n - \delta_i) - \sum_{t=0}^{k} \tau_{it}}}$$

In this formulation, an item is defined by a central item difficulty, δ_i and a set of centred Rasch-Andrich thresholds $\{\tau_{ik}\}$, $k \in \{0,\ ...,\ m\}$ which sum to zero: an alternative formulation is to define the item solely by a set of uncentred thresholds, $\{\tau'_{ik}\}$, $k \in \{0,\ ...,\ m\}$, where $\tau'_{ik} = \delta_i + \tau_{ik}$ for $i = 1,\ ...,\ m$ and $\tau'_{ik} = 0$. Separate methods for all core functions are defined using uncentred thresholds.

Usage

```
self.cat_prob(ability, difficulty, category, thresholds)
```

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Central item difficulty
`category`	Integer: Response category k, with $k \in \{0,\ 1\}$.
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns

Float: probability of obtaining score k.

Example

To obtain the probability of a person of ability 0.5 scoring 0 on an item of central difficulty 0 with a set of Rasch-Andrich thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_cat_prob`:

```
my_cat_prob = self.cat_prob_centred(0.5, 0, 0, my_thresholds)
```

4.2.2 `cat_prob_uncentred`

Description

Category probability function which calculates the probability $P(X_{ni} = k)$ of scoring k, with $k \in \{0,\ m\}$, where m is the maximum score, from person ability and uncentred thresholds (see Section 4.2.1 above for the centred thresholds formulation). For a person n with ability β_n attempting an item i with thresholds $\{\tau_{i1}, ..., \tau_{im}\}$, the probability of obtaining a score of k is given by:

$$P(X_{ni} = k) = \frac{e^{k\beta_n - \sum_{t=0}^{k} \tau_{it}}}{\sum_{k=0}^{m} e^{k\beta_n - \sum_{t=0}^{k} \tau_{it}}}$$

In this formulation, an item is defined by a set of thresholds $\{\tau_{ik}\}$, $k \in \{1,\ ...,\ m\}$: an alternative formulation is to define the item a central item difficulty, δ_i and a set of centred Rasch-Andrich thresholds $\{\tau'_k\}$, $k \in \{0,\ ...,\ m\}$ which sum to zero, where $\delta_i = \sum_i = 1^m \tau_{ik}/m$ and $\tau'_{ik} = \tau_{ik} - \delta_i$. Separate methods for all core functions are defined using centred thresholds.

Usage

`self.cat_prob(ability, category, thresholds)`

Arguments

`ability`	Float: Person ability
`category`	Integer: Response category k, with $k \in \{0,\ 1\}$.
`thresholds`	List or numpy array: Set of m uncentred thresholds for scores 1 to m, where m is the maximum score.

Returns

Float: probability of obtaining score k.

Example

To obtain the probability of a person of ability 0.5 scoring 0 on an item with central difficulty 0 and a set of thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_cat_prob`:

```
my_cat_prob = self.cat_prob_centred(0.5, 0, my_thresholds)
```

4.2.3 `exp_score_centred`

Description

Expected score function which calculates the expected score $E(X_{ni})$ from person ability, central item difficulty and set of centred Rasch-Andrich thresholds. The expected score is given by:

$$E(X_{ni}) = \sum_{k=0}^{1} kP(X_{ni} = k)$$

where $P(X_{ni} = k)$ is as described in Section 4.2.1.

Usage

`self.exp_score(ability, difficulty, thresholds)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Central item difficulty
`thresholds`	List or numpy array: Set of $m+1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns

Float: expected score.

Example

To obtain the expected score for a person of ability 0.5 attempting an item of difficulty 0 with a set of Rasch-Andrich thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_exp_score`:

```
my_exp_score = self.exp_score_centred(0.5, 0, my_thresholds)
```

4.2.4 `exp_score_uncentred`

Description

Expected score function which calculates the expected score $E(X_{ni})$ from person ability and a set of uncentred thresholds. The expected score is given by:

$$E(X_{ni}) = \sum_{k=0}^{1} kP(X_{ni} = k)$$

where $P(X_{ni} = k)$ is as described in Section 4.2.2.

Usage

`self.exp_score_uncentred(ability, thresholds)`

Arguments

`ability`	Float: Person ability
`thresholds`	List or numpy array: Set of m thresholds, where m is the maximum score.

Returns
Float: expected score.

Example
To obtain the expected score for a person of ability 0.5 attempting an item of difficulty 0 with a set of thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_exp_score`:

```
my_exp_score = self.exp_score_uncentred(0.5, my_thresholds)
```

4.2.5 `variance_centred`

Description
Variance function which calculates the variance of the score $V(X_{ni})$ from person ability, central item difficulty and set of Rasch-Andrich thresholds. The variance is given by:

$$V(X_{ni}) = \sum_{k=0}^{1} P(X_{ni} = k)(k - E(X_{ni}))^2$$

where $P(X_{ni} = k)$ and $E(X_{ni})$ are as described in Sections 4.2.1 and 4.2.3 respectively.

The variance is also both the Fisher information for the response and the first partial differential of the expected score function with respect to person ability.

Usage

```
self.variance_centred(ability, difficulty, thresholds)
```

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Central item difficulty
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns
Float: variance (also the Fisher information provided by the response).

Example

To obtain the variance for a person of ability 0.5 attempting an item with central difficulty 0 and a set of Rasch-Andrich thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_variance`:

```
my_variance = self.variance_centred(0.5, 0, my_thresholds)
```

4.2.6 `variance_uncentred`

Description

Variance function which calculates the variance of the score $V(X_{ni})$ from person ability and set of uncentred thresholds. The variance is given by:

$$V(X_{ni}) = \sum_{k=0}^{1} P(X_{ni} = k)(k - E(X_{ni}))^2$$

where $P(X_{ni} = k)$ and $E(X_{ni})$ are as described in Sections 4.2.2 and 4.2.4 respectively.

The variance is also both the Fisher information for the response and the first partial differential of the expected score function with respect to person ability.

Usage

```
self.variance_uncentred(ability, thresholds)
```

Arguments

`ability`	Float: Person ability
`thresholds`	List or numpy array: Set of m thresholds, where m is the maximum score.

Returns

Float: variance (also the Fisher information provided by the response).

Example

To obtain the variance for a person of ability 0.5 attempting an item with a set of uncentred thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_variance`:

```
my_variance = self.variance_centred(0.5, my_thresholds)
```

4.2.7 `kurtosis_centred`

Description

Kurtosis function which calculates the kurtosis of the score $\kappa(X_{ni})$ from person ability, central item difficulty and Rasch-Andrich thresholds. The variance is given by:

$$\kappa(X_{ni}) = \sum_{k=0}^{1} P(X_{ni} = k)(k - E(X_{ni}))^4$$

where $P(X_{ni} = k)$ and $E(X_{ni})$ are as described in Sections 5.2.1 and 5.2.2 respectively.

Usage

`self.kurtosis_centred(ability, difficulty, thresholds)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns

Float: kurtosis

Example

To obtain the kurtosis for a person of ability 0.5 attempting an item with central difficulty 0 and a set of Rasch-Andrich thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_kurtosis`:

```
my_kurtosis = self.kurtosis_centred(0.5, 0, my_thresholds)
```

4.2.8 `kurtosis_uncentred`

Description

Kurtosis function which calculates the kurtosis of the score $\kappa(X_{ni})$ from person ability, central item difficulty and Rasch-Andrich thresholds. The variance is given by:

$$\kappa(X_{ni}) = \sum_{k=0}^{1} P(X_{ni} = k)(k - E(X_{ni}))^4$$

where $P(X_{ni} = k)$ and $E(X_{ni})$ are as described in Sections 4.2.2 and 4.2.4 respectively.

Usage

`self.kurtosis_uncentred(ability, thresholds)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns

Float: kurtosis

Example

To obtain the kurtosis for a person of ability 0.5 attempting an item with a set of uncentred thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_kurtosis`:

```
my_kurtosis = self.kurtosis_uncentred(0.5, my_thresholds)
```

4.3 Parameter estimation

4.3.1 `calibrate`

Description

Produces item estimates in two formats: central item difficulty and Rasch-Andrich threshold estimates (See sections 4.2.1 and 4.2.2 for details), plus uncentred thresholds using PAIR (Choppin, 1968, 1985), eigenvector method (Garner & Engelhard, 2002) or related conditional pairwise methods.

Usage

```
self.calibrate(constant=0.1, method='cos', matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of central item difficulty estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) for central item difficulty estimates is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for central item difficulty estimates. Ignored for other methods.

Returns

Returns four attributes:

`self.central_diffs`	pandas series: Central item difficulty estimates with the item names as keys and estimates as values.
`self.thresholds_centred`	Dictionary: Item names as keys, numpy arrays of m centred threshold estimates which sum to zero and the first of which is zero, where m is the maximum score, as values.
`self.thresholds_uncentred`	Dictionary: Item names as keys, numpy arrays of m uncentred threshold estimates, where m is the maximum score, as values.
`self.cat_widths`	Dictionary: Item names as keys, numpy arrays of $m-1$ category width estimates, where m is the maximum score, as values.

Examples

To generate a set of estimates using the cosine similarity method for central item difficulties, with additive smoothing constant of 0.1:

```
self.calibrate()
```

To generate a set of estimates using the log-likelihood method for central item difficulties, with matrix raised to power 7 and a convergence stopping criterion of 0.00000001:

```
self.calibrate(method='log-lik', matrix_power=7, log_lik_tol=0.00000001)
```

4.3.2 `std_errors`

Description

Produces bootstrapped estimates for the standard errors of central item difficulty estimates and threshold estimates. Threshold standard errors are shared between centred and uncentred thresholds, while central item difficulty estimates are calculated separately.

Usage

```
self.std_errors(interval=None, no_of_samples=100, constant=0.1, method='cos',
                matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval. Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.
`no_of_samples`	Integer: Number of bootstrap samples to generate. More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of central item difficulty estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`

Arguments continue on the next page.

Arguments (continued)

`matrix_power`	Integer: power to which conditional category response frequency matrix for central item difficulty estimates (Elliott & Buttery, 2022b:991) is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for central item difficulty estimates. Ignored for other methods.

Returns
Attributes:

`self.central_bootstrap`	pandas dataframe: Full central item difficulty bootstrap results, with a row for each bootstrap sample and a column for each central item difficulty estimate.
`self.central_se`	pandas series: Item names as keys and item standard errors as values.
`self.threshold_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each threshold estimate. `self.threshold_bootstrap` contains uncentred threshold estimates (standard errors would be identical with centred thresholds).
`self.threshold_se`	pandas series: Threshold numbers as keys and threshold standard errors as values.
`self.cat_width_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each category width estimate.
`self.cat_width_se`	pandas series: Category numbers as keys and category width standard errors as values.

If an argument is passed to `interval`, also returns:

`self.item_low`	Lower bound of the specified interval for item estimates.
`self.item_high`	Upper bound of the specified interval for item estimates.
`self.threshold_low`	Lower bound of the specified interval for threshold estimates.
`self.threshold_high`	Upper bound of the specified interval for threshold estimates.
`self.cat_width_low`	Lower bound of the specified interval for category estimates.
`self.cat_width_high`	Upper bound of the specified interval for category estimates.

Example
To generate item standard errors with a 95% interval from 200 samples: `self.std_errors(interval=0.95, no_of_samples=200)`

Modifications to the estimation method are discussed in Section 4.3.1.

4.3.3 `abil`

Description
Generates an ability estimate for a person using the Newton-Raphson method to produce a maximum likelihood estimate, with optional Warm bias correction (Warm, 1989).

Usage
```
self.abil(person, items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
        ext_score_adjustment=0.5)
```

Arguments

`person`	String: The person name for the ability being estimated.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Float: person ability estimate.

Examples
To generate a person ability estimate for `Person_1` using the default settings and store the result as a variable, `my_person_ability`:

```
my_person_ability = my_person_ability = my_pcm.abil('Person_1')
```

To generate an MLE person ability estimate without Warm bias correction for `Person_1` based on the first three items and store the result as a variable, `my_person_ability`:

```
my_person_ability = my_pcm.abil('Person_1', ['Item_1', 'Item_2', 'Item_3'], warm_corr=False)
```

4.3.4 `person_abils`

Description
Generates ability estimates for all persons using the Newton-Raphson method to produce maximum likelihood estimates, with optional Warm bias correction (Warm, 1989).

Usage
```
self.person_abils(items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
                  ext_score_adjustment=0.5)
```

Arguments

`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Attribute `self.person_abilities`: pandas series with person names as keys and ability estimates as values.

Examples

To generate a set of person ability estimates with Warm bias correction:

```
self.person_abils()
```

To generate a set of person ability estimates without Warm bias correction, on a subset of the first three items only:

```
my_pcm.person_abils(items=['Item_1', 'Item_2', 'Item_3'], warm_corr=False)
```

4.3.5 `score_abil`

Description

Generates an ability estimate for a given raw score on responses to a given set of items using the Newton-Raphson method to produce a maximum likelihood estimate, with optional Warm bias correction (Warm, 1989).

Usage

```
self.score_abil(score, items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
          ext_score_adjustment=0.5)
```

Arguments

`score`	Integer: The raw score for which ability is being estimated.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns

pandas series with raw scores as keys and person ability estimates as values.

Examples

To generate an ability estimate for a score of 10 on all items, with Warm bias correction, and store the result as a variable, `my_score_ability`:

```
my_score_ability = my_pcm.score_abil(10)
```

To generate an ability estimate for a score of 10 on a subset of items saved as a variable `my_items`, without Warm bias correction, and store the result as a variable, `my_score_ability`:

```
my_score_ability = my_pcm.score_abil(10, items=my_items, warm_corr=False)
```

4.3.6 `abil_lookup_table`

Description

Generates a lookup table of ability estimates corresponding to all available raw scores on a set of items with no missing responses, using the Newton-Raphson method to produce maximum likelihood estimates and with optional Warm bias correction (Warm, 1989).

Usage

```
self,abil_lookup_table(items=None, ext_scores=True, warm_corr=True, tolerance=1e-07,
                    max_iters=100, ext_score_adjustment=0.5)
```

Arguments

`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`ext_scores`	Boolean: If `True`, ability estimates for extreme scores (all correct/all incorrect) will be generated using the `ext_score_adjustment` argument. Default is `ext_scores=True`.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Attribute `self.abil_table`: pandas series with raw scores as keys and corresponding ability estimates as values.

Examples
To generate an ability lookup table for all items, including extreme scores, with Warm bias correction:

```
my_pcm.abil_lookup_table()
```

To generate an ability lookup table for a subset of items saved as a variable `my_items`), without extreme scores and without Warm bias correction:

```
my_pcm.abil_lookup_table(items=my_items, ext_scores=False)
```

4.3.7 `csem`

Description
Calculates conditional standard error of measurement for a person.

Usage
`self.csem(person, abilities=None, items=None)`

Arguments

`person`	Person name.
`abilities`	pandas series (or dictionary) with person names as keys and abilities as values. If `None`, uses `self.person_abilities`, automatically generating if necessary. Default is `self.person_abilities=None`.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.

Returns
Float: conditional standard error of measurement for ability estimate.

Examples
To generate the CSEM for `Person_1` on all items and save the result as a variable, `my_csem`:

```
my_csem = my_pcm.csem('Person_1')
```

To generate the CSEM for a raw score of 3 on a subset of items saved to a variable `my_items` and save the result as a variable, `my_csem`:

```
my_csem = my_pcm.csem(3, abilities=self.abil_table, items=my_items)
```

where `self.abil_table` is the output from `self.abil_lookup_table(items=my_items)` (see Section 4.3.6).

4.4 Statistical output

4.4.1 item_stats_df

Description

Produces a table of item statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.item_stats_df(full=False, zstd=False, point_measure_corr=False, dp=3,
                   warm_corr=True, tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5,
                   method='cos', constant=0.1, no_of_samples=100, interval=None)
```

Arguments

full	Boolean: If True, a full table with all available statistics is produced. Default is full=False for a summary table with the most commonly reported statistics.
zstd	Boolean: If True, infit and outfit standardised z-scores are reported alongside mean square statistics. Default is zstd=False. Automatically True if full=True.
point_measure_corr	Boolean: If True, point-polyserial correlation (Kornbrot, 2014) between person ability and item score is reported, together with the expected value of the point-measure polyserial correlation for an ideal item. Default is point_measure_corr=False. Automatically True if full=True.
dp	Integer: Number of decimal places reported in output.
warm_corr	Boolean: If True, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is warm_corr=True.
tolerance	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is tolerance=1e-07.
max_iters	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is max_iters=100.

Arguments continue on the next page.

Arguments (continued)

`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during item estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for item estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute `self.item_stats`, a pandas dataframe with one row for each item and the following columns:

`Estimate`	Central item difficulty estimate.
`SE`	Bootstrapped standard error of central item difficulty estimate.
`2.5%`	Lower bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 2.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `5%`.

Returns continue on the next page.

Returns (continued)

`97.5%`	Upper bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 97.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `95%`.
`Count`	Count of responses.
`Facility`	Item facility: proportion of correct responses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score. Only produced if `zstd=True`.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score. Only produced if `zstd=True`.
`PM corr`	Point-polyserial correlation. Only produced if `point_measure_corr=True`.
`Exp PM corr`	Expected value of point-measure polyserial correlation. Only produced if `point_measure_corr=True`.
`Disordered`	If `True`, threshold estimates indicate the presence of disordered thresholds within the item (Andrich, 2010).

Examples

To produce a summary `item_stats` table with the most commonly reported statistics:

```
my_pcm.item_stats_df()
```

To produce a full `item_stats` table with all statistics:

```
my_pcm.item_stats_df(full=True)
```

To produce an `item_stats` table with with the most commonly reported statistics, plus standardised z-scores for infit and outfit:

```
my_pcm.item_stats_df(zstd=True)
```

Other arguments may be used to alter parameters of central item difficulty and/or person ability estimation.

4.4.2 `threshold_stats_df`

Description

Produces a table of threshold statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.threshold_stats_df(full=False, zstd=False, disc=False, point_measure_corr=False, dp=3,
                        warm_corr=True, tolerance=1e-07, max_iters=100,
                        ext_score_adjustment=0.5, method='cos', constant=0.1,
                        no_of_samples=100, interval=None)
```

Arguments

`full`	Boolean: If `True`, a full table with all available statistics is produced. Default is `full=False` for a summary table with the most commonly reported statistics.
`zstd`	Boolean: If `True`, infit and outfit standardised z-scores are reported alongside mean square statistics. Default is `zstd=False`. Automatically `True` if `full=True`.
`disc`	Boolean: If `True`, item discrimination is reported. The discrimination of the empirical item slope relative to the ideal logistic ogive, with 1 perfect, greater than 1 showing overfit and less than 1 showing underfit; discrimination is similar to the 2PL IRT discrimination parameter (Linacre, 2023), but is a descriptive statistic in the SLM rather than an item parameter.
`point_measure_corr`	Boolean: If `True`, point-biserial correlation (Kornbrot, 2014) between person ability and item score is reported, together with the expected value of the point-measure biserial correlation for an ideal item. Default is `point_measure_corr=False`. Automatically `True` if `full=True`.
`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Arguments continue on the next page.

Arguments (continued)

`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 5.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Two attributes: `self.threshold_stats_centred` and `self.threshold_stats_centred`, pandas dataframes which differ only in their first column, `Estimate`. The dataframes have one row for each individual threshold and the following columns:

`Estimate`	Threshold difficulty estimate. For the `threshold_stats_centred` table, this contains the estimates of the centred Rasch-Andrich thresholds, while for the `threshold_stats_uncentred` table, it contains the estimates of the uncentred thresholds.
`SE`	Bootstrapped standard error of item difficulty estimate.
`2.5%`	Lower bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 2.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `5%`.

Returns continue on the next page.

Returns (continued)

`97.5%`	Upper bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 97.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `95%`.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score. Only produced if `zstd=True`.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score. Only produced if `zstd=True`.
`Discrim`	Item discrimination. Only produced if `disc=True`.
`PM corr`	Point-biserial correlation. Only produced if `point_measure_corr=True`.
`Exp PM corr`	Expected value of point-measure biserial correlation. Only produced if `point_measure_corr=True`.

Examples

To produce summary `threshold_stats` tables with the most commonly reported statistics:

```
self.threshold_stats_df()
```

To produce a full `my_pcm.threshold_stats` table with all statistics:

```
my_pcm.threshold_stats_df(full=True)
```

To produce `threshold_stats` tables with with the most commonly reported statistics, plus standardised z-scores for infit and outfit:

```
my_pcm.threshold_stats_df(zstd=True)
```

Other arguments may be used to alter parameters of central item difficulty, threshold and person ability estimation.

4.4.3 `person_stats_df`

Description

Produces a table of person statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.person_stats_df(full=False, rsem=False, dp=3, warm_corr=True, tolerance=1e-07,
                     max_iters=100, ext_score_adjustment=0.5, method='cos', constant=0.1)
```

Arguments

`full`	Boolean: If `True`, a full table with all available statistics is produced. Default is `full=False` for a summary table with the most commonly reported statistics.
`rsem`	Boolean: If `True`, realistic standard error of measurement (RSEM), which takes into account for item misfit (Wright, 1996), is reported alongside the conditional standard error of measurement (CSEM). Default is `rsem=False`. Automatically `True` if `full=True`.
`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.

Arguments continue on the next page.

Arguments (continued)

`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute `self.person_stats`, a pandas dataframe with one row for each person and the following columns:

`Estimate`	Item difficulty estimate.
`CSEM`	Conditional standard error of measurement for person ability estimate.
`RSEM`	Realistic standard error of measurement for person ability estimate. Only produced if `rsem=True`
`Score`	Number of correct responses.
`Max score`	Maximum available score (number of items attempted).
`p`	Proportion of correct repsonses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score.

Examples

To produce a summary `person_stats` table with the most commonly reported statistics:

```
my_pcm.person_stats_df()
```

To produce a full `self.person_stats` table with all statistics:

```
my_pcm.person_stats_df(full=True)
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

4.4.4 `test_stats_df`

Description

Produces a table of test-level statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.test_stats_df(dp=3, warm_corr=True, tolerance=1e-07, max_iters=100,
                   ext_score_adjustment=0.5, method='cos', constant=0.1)
```

Arguments

`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is `constant=0.1`.

Returns

Attribute `self.test_stats`, a pandas dataframe with three columns, `Items`, `Thresholds` and `Persons` and rows for a range of descriptive statistics describing the distributions of the estimates and different statistics related to reliability – these statistics describe the suitability of the data for estimating and differentiating the parameters, rather than properties of the parameters themselves. The statistics are:

`Mean`	The mean of the estimates.
`SD`	The standard deviation of the estimates.
`Separation ratio`	The separation ratio (Wright, 1996; Wright & Masters, 1982), which is the standard deviation of person abilities reported as a ratio of standard error units. For persons: $$G_p = \sigma_p / \sqrt{\sum_n SE_n^2}$$ where σ_p is the variance of the person estimates and SE_n is the RSEM (see Section 3.4.2) for person n. The formula is symmetrical for items, substituting the standard error of estimation for RSEM.
`Strata`	The number of statistically distinct levels of either person ability or item difficulty as strata with centers three measurement errors apart (Wright & Masters, 1982:106). For persons: $$H_p = (4G_p + 1)/3$$ with symmetrical results for items.
`Reliability`	A Rasch-specific reliability statistic (Wright, 1996), derived from PSI and which is a Rasch-specific reliability statistic similar to Cronbach's Alpha (Cronbach, 1951), and which may be interpreted the same way – as the proportion of variance of the estimates which stems from variation in ability or difficulty rather than estimation error. For persons: $$R_p = G_p^2/(1 + G_p^2)$$ with symmetrical results for items.

Example

To produce a `my_pcm.test_stats` table:

```
my_pcm.test_stats_df()
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

4.4.5 `res_corr_analysis`

Description

Analysis of correlations of standardised residuals to tests for violations of local item interdependence and unidimensionality requirements.

Usage

```
self.res_corr_analysis(warm_corr=True, tolerance=1e-07, max_iters=100,
                       ext_score_adjustment=0.5, constant=0.1, method='cos',
                       matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for generation of item difficulty estimates (see Section 3.3.1). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) for generation of central item difficulty estimates (see Section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) is raised for generation of central item difficulty estimates (see Section 3.3.1). Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for generation of item difficulty estimates (see Section 3.3.1). Ignored for other methods.

Returns

For tests of violation of the requirement for local item independence (Andrich & Kreiner, 2010; Marais, 2012):

`self.residual_correlations`	A pandas dataframe of pairwise correlations between item standardised residuals.

For tests of violation of the requirement for unidimensionality based on principal component analysis of the standardised residual correlations (Pallant & Tennant, 2007; Smith, 2002):

`self.eigenvectors`	The eigenvectors of the standardised residual correlations matrix.
`self.eigenvalues`	The eigenvalues corresponding to the eigenvectors.
`self.variance_explained`	The variance explained by each principal component.
`self.loadings`	The loading of each item onto each of the principal components, for the the first of which large loadings (‘large'typically interpreted as > 0.4 or < -0.4) may be interpreted as representing the presence of significant dimensionality, in analogy to factor analysis (**<empty citation>**).

Example
To produce a residual correlation analysis:

```
my_pcm.res_corr_analysis()
```

Arguments may be used to alter parameters of item difficulty and/or person ability estimation.

4.4.6 `category_counts_df`

Description
Produces a table of counts of scores in each category, plus responses and missing responses, for each item.

Usage
```
self.category_counts_df()
```

Arguments None

Returns
Attribute `self.category_counts`, a pandas dataframe of category counts with one row per item and one column per response category, plus total responses per item and missing responses per item.

Example
To produce a dataframe of category counts:

```
my_pcm.category_counts_df()
```

4.5 Plotting functionality

4.5.1 Shared plotting arguments

All the plotting methods described in this section share a set of arguments which may be used to customise the appearance of the plot or save the plot to file automatically. These arguments are:

`title`	String: Title for the plot, to appear in the image. Default is `title=None`.
`xmin`	Float: Minimum displayed point on x-axis, in logits. Default is `xmin=-5`.
`xmax`	Float: Maximum displayed point on x-axis, in logits. Default is `xmax=5`.
`plot_style`	String: Plot style to use. Available styles are `'white'`, producing a plot on a white background, and `'dark'`, producing a plot on a grey background. The styles correspond to Seaborn (Waskom, 2021) styles `whitegrid` and `darkgrid`. Default is `plot_style='white'`.
`palette`	String: Controls colours of lines and overplotted class intervals dots. Options are `'dark blue'`, `'light blue'`, `'dark red'`, `'light red'`, `'dark green'`, `'light green'`, `'dark grey'`, `'light grey'`, `'dark multi'` and `'light multi'`. Default is `palette='dark blue'`. Not used for `std_residual_plot`.
`black`	Boolean: If `True`, the plot will be rendered in black and white. Default is `black=False`.
`font`	String: The font to use in the plot. Default is `font='Times'`.
`title_font_size`	Float: The size of the title font in points. Default is `title_font_size=15`.
`axis_font_size`	Float: The size of the axis label font in points. Default is `axis_font_size=15`.
`labelsize`	Float: The size of the axis tick label font in points. Default is `labelsize=15`.
`filename`	String: The filename for the saved plot, with no suffix for format. If `None`, no file will be saved. Default is `filename=None`.
`file_format`	The format of the file: `png`, `jpg` or `svg`. Default is `file_format=png`.
`dpi`	The resolution of the plot in dpi (dots per inch) – higher resolution plots are better quality but have larger file sizes. Default is `dpi=300`.

4.5.2 `icc`

Description

Plots the item characteristic curves (or item response function) for an item: person ability on the x-axis against expected score on the y-axis. Options to plot observed responses, item threshold line and lines showing abilities corresponding to specified expected scores, and to highlight a specified response category.

Usage

```
self.icc(item, obs=False, no_of_classes=5, thresh_lines=False, central_diff=False,
         score_lines=None, score_labels=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`obs`	Boolean: If `True`, mean observed scores for each of the ordered response categories will be plotted against the mean ability of the corresponding response class. Default is `obs=False`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`score_lines`	List of floats between 0 and `self.max_score`: Each float in the list represents an expected score, for which a horizontal line from the y-axis to where it intersects the item characteristic curve will be plotted, and from there a vertical line down to the x-axis will be plotted to show the ability corresponding to the expected score.
`score_labels`	Boolean: If `True`, scores and abilities corresponding to arguments passed to `score_lines` will be labelled on the plot. Default is `score_labels=False`.
`cat_highlight`	Integer: Passing a score between 0 and `self.max_score` will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section **??**.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot a basic item characteristic curve for `Item_1` and store the output as a variable `my_icc_plot` and save it to file as `my_icc_plot.png`:

```
my_icc_plot = my_pcm.icc('Item_1', filename=my_icc_plot)
```

To plot an item characteristic curve for `Item_1` with observed responses for 8 response classes and store the output as a variable `my_icc_plot`:

```
my_icc_plot = self.icc('Item_1', obs=True, no_of_classes=8)
```

To plot an item characteristic curve for `Item_1` with a threshold line and highlighted zero category, and store the output as a variable `my_icc_plot`:

```
my_icc_plot = my_pcm.icc('Item_1', thresh_line=True, cat_highlight=0)
```

To plot an item characteristic curve for `Item_1` with lines showing the abilities corresponding to expected scores of 0.7 and 1.6, with the expected score and corresponding ability labelled, and store the output as a variable `my_icc_plot`:

```
my_icc_plot = my_pcm.icc('Item_1', score_lines=[0.7, 1.6], score_labels=True)
```

4.5.3 `crcs`

Description

Plots category response curves for an item: person ability on the x-axis against expected the probability of obtaining a score in each category (0 or 1) on the y-axis. Options to plot observed proportions and item threshold line, and to highlight a specified response category.

Usage

```
self.crcs(item, obs=None, no_of_classes=5, thresh_lines=False, central_diff=False,
          cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`obs`	List: List of integers between 0 and `self.max_score`. For each value, mean observed proportions in each ordered response category scoring in that category are plotted against the mean ability of the corresponding response class.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.

Arguments continue on the next page.

Arguments (continued)

`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`cat_highlight`	Integer: Passing a score between 0 and `self.max_score` will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot basic category response curves for `Item_1` and store the output as a variable `my_crcs_plot` and save it to file as `my_crcs_plot.png`:

```
my_crcs_plot = my_pcm.crcs('Item_1', filename=my_crcs_plot)
```

To plot category response curves for `Item_1` with observed response proportions for category 0 for 8 response classes and store the output as a variable `my_crcs_plot`:

```
my_crcs_plot = my_pcm.crcs('Item_1', obs=[0], no_of_classes=8)
```

To plot category response curves for `Item_1` with a threshold line and highlighted zero category, and store the output as a variable `my_crcs_plot`:

```
my_crcs_plot = my_pcm.crcs('Item_1', thresh_line=True, cat_highlight=0)
```

4.5.4 `threshold_ccs`

Description
Plots the threshold characteristic curves (or threshold response functions)for an item. For threshold τ_k, $k \in$ {1, ..., `self.max_score`}, the threshold characteristic curve is the probability of obtaining a score of k rather than $k-1$, conditional on the score being either $k-1$ rather than k, for a given person ability. Each threshold characteristic curve functions as a dichotomous item characteristic curve under the SLM (see Sections 3.2.2 and 3.5.2).

For threshold τ_k, `threshold_ccs` plots person ability on the x-axis against the probability of obtaining a score of k on the y-axis. Options to plot threshold lines and central item difficulties, and to highlight a specified response category.

Usage

```
self.threshold_ccs(item, obs=None, no_of_classes=5, thresh_lines=False, central_diff=False,)
                   cat_highlight=None,)
```

Arguments

`item`	String: The name of the item to plot.
`obs`	List: List of integers corresponding to thresholds τ_1 to τ_m, where m =`self.max_score`, or `'all'` or `'none'`. If `obs=[k]`, mean proportions of persons obtaining a score of k rather than $k-1$, conditional on the score being either $k-1$ or k, for each of the ordered response categories, will be plotted against the mean ability of the corresponding response class. Multiple thresholds may be passed. Default is `obs=None`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`cat_highlight`	Integer: Passing 0 or 1 will highlight the range of abilities for which the selected score is the most probable response (all abilities to one side of the item difficulty. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot basic threshold characteristics curves for `Item_1`, store the output as variable `my_threshold_ccs_plot`, and save it to file as `my_threshold_ccs_plot.png`:

```
my_threshold_ccs_plot = my_pcm.threshold_ccs(item='Item_1', filename='my_iic_plot')
```

To plot threshold characteristics curves for `Item_1` with threshold lines and category 1 highlighted, and store the output as a variable `my_threshold_ccs_plot`:

```
my_threshold_ccs_plot = my_pcm.threshold_ccs(item='Item_1', thresh_lines=True, cat_highlight=1)
```

To plot threshold characteristics curves for `Item_1` with observed responses plotted for thresholds 2 and 4 and central item difficulty line, and store the output as a variable `my_threshold_ccs_plot`:

```
my_threshold_ccs_plot = my_pcm.threshold_ccs(item='Item_1', obs=[2, 4], central_diff=True)
```

4.5.5 `iic`

Description

Plots the item information curve for an item: person ability on the x-axis against Fisher information on the y-axis. Options to plot item threshold line and lines showing Fisher information corresponding to specified abilities, and to highlight a specified response category.

Usage

```
self.iic(item, ymax=None, thresh_lines=False, central_diff=False, point_info_lines=None,
        point_info_labels=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`ymax`	Float: The maximum value to show on the y-axis. If `None`, will infer, plotting a maximum of 1.1 times the maximum item information. Default is `ymax=None`
`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`point_info_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the item information curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the Fisher information corresponding to the ability. Default is `point_info_lines=None`.

Arguments continue on the next page.

Arguments (continued)

point_info_labels	Boolean: If True, abilities and Fisher information corresponding to arguments passed to point_info_lines will be labelled on the plot. Default is point_info_labels=False.
cat_highlight	Integer: Passing a score between 0 and self.max_score will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is cat_highlight=None (no category highlighted).
ymax	The maximum point displayed on the y-axis, in Fisher information.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns
matplotlib image. If a filename argument is passed, also saves the image to file.

Examples
To plot a basic item information curve for Item_1 and store the output as a variable my_iic_plot and save it to file as my_iic_plot.png:

```
my_iic_plot = my_pcm.iic('Item_1', filename='my_iic_plot')
```

To plot an item information curve for Item_1 with threshold lines, central item difficulty line and category 1 highlighted, and store the output as a variable my_iic_plot:

```
my_iic_plot = my_pcm.iic('Item_1', thresh_line=True, central_diff=True, cat_highlight=1)
```

To plot an item information curve for Item_1 with lines showing the Fisher information corresponding to abilities of -0.3 and 0.7, with the ability and corresponding Fisher information labelled, and store the output as a variable my_iic_plot:

```
my_iic_plot = my_pcm.icc('Item_1', point_info_lines=[-0.3, 0.7], point_info_labels=True)
```

4.5.6 tcc

Description
Plots the test characteristic curve (or test response function) for a set of items: person ability on the x-axis against expected score on the y-axis. Options to plot observed responses and lines showing abilities corresponding to specified expected scores.

Usage

```
self.tcc(items=None, obs=False, no_of_classes=5, score_lines=None, score_labels=False)
```

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`obs`	Boolean: If `True`, mean observed scores for each of the ordered response categories will be plotted against the mean ability of the corresponding response class. Default is `obs=False`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`score_lines`	List of floats between 0 and `self.max_score`: Each float in the list represents an expected score, for which a horizontal line from the y-axis to where it intersects the item characteristic curve will be plotted, and from there a vertical line down to the x-axis will be plotted to show the ability corresponding to the expected score. Default is `score_lines=None`.
`score_labels`	Boolean: If `True`, scores and abilities corresponding to arguments passed to `score_lines` will be labelled on the plot. Default is `score_labels=False`.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot a basic test characteristic curve for all items and store the output as a variable `my_tcc_plot` and save it to file as `my_tcc_plot.png`:

```
my_tcc_plot = my_pcm.tcc(filename=my_tcc_plot)
```

To plot a test characteristic curve for `Item_1` for a subset of items stored as a list **`my_item_list`**, with observed responses for 8 response classes and store the output as a variable `my_tcc_plot`:

```
my_tcc_plot = my_pcm.tcc(obs=True, no_of_classes=8)
```

To plot a test characteristic curve for `Item_1` for all items with lines showing the abilities corresponding to expected scores of 13 and 20, with the expected score and corresponding ability labelled, and store the output as a variable `my_tcc_plot`:

```
my_tcc_plot = my_pcm.tcc(score_lines=[13, 20], score_labels=True)
```

4.5.7 `test_info`

Description

Plots the test information curve: person ability on the x-axis against total Fisher information on the y-axis. Option to plot lines showing Fisher information corresponding to specified abilities.

Usage

`self.test_info(items=None, ymax=None, point_info_lines=None, point_info_labels=False)`

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`point_info_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the item information curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the total Fisher information corresponding to the ability. Default is `point_info_lines=None`.
`point_info_labels`	Boolean: If `True`, abilities and total Fisher information corresponding to arguments passed to `point_info_lines` will be labelled on the plot. Default is `point_info_labels=False`.
`ymax`	The maximum point displayed on the y-axis, in Fisher information.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot a basic test information curve and store the output as a variable `my_test_info_plot` and save it to file as `my_test_info_plot.png`:

```
my_test_info_plot = my_pcm.test_info(filename='my_test_info_plot')
```

To plot a test information curve for a subset of items stored as a list **`my_item_list`** and store the output as a variable `my_test_info_plot`:

```
my_test_info_plot = my_pcm.test_info(items=my_item_list)
```

To plot a test information curve with lines showing the total Fisher information corresponding to abilities of -0.3 and 0.7, with the ability and corresponding total Fisher information labelled, and store the output as a variable `my_test_info_plot`:

```
my_test_info_plot = my_pcm.test_info(point_info_lines=[-0.3, 0.7], point_info_labels=True)
```

4.5.8 `test_csem`

Description
Plots the test conditional standard error of measurement (CSEM) curve: person ability on the x-axis against CSEM (in logits) on the y-axis. Option to plot lines showing CSEM corresponding to specified abilities.

Usage

```
self.test_csem(items=None, point_csem_lines=None, point_csem_labels=False, ymax=5)
```

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`point_csem_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the CSEM curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the CSEM corresponding to the ability. Default is `point_csem_lines=None`.
`point_csem_labels`	Boolean: If `True`, abilities and CSEM corresponding to arguments passed to `point_csem_lines` will be labelled on the plot. Default is `point_csem_labels=False`.
`ymax`	The maximum point displayed on the y-axis, in logits.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot a basic CSEM curve and store the output as a variable `my_test_csem_plot` and save it to file as `my_test_csem_plot.png`:

```
my_test_csem_plot = my_pcm.test_csem(filename='my_test_csem_plot')
```

To plot a CSEM curve for a subset of items stored as a list **`my_item_list`** and store the output as a variable `my_test_csem_plot`:

`my_test_csem_plot = my_pcm.test_csem(items=my_item_list)`

To plot a CSEM curve with lines showing the CSEM corresponding to abilities of -0.3 and 0.7, with the ability and corresponding CSEM labelled, and store the output as a variable `my_test_csem_plot`:

```
my_test_csem_plot = my_pcm.test_csem(point_csem_lines=[-0.3, 0.7], point_csem_labels=True)
```

4.5.9 `std_residuals_plot`

Description
Plots histogram of standardised residuals, with optional overplotting of standard Normal distribution.

Usage
`self.std_residuals_plot(items=None, bin_width=0.5, normal=False)`

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`bin_width`	Float: The width of the histogram bins along the x-axis. Default is `bin_width=0.5`.
`normal`	Boolean: If `True`, plots a standard normal distribution over the standardised residual histogram for comparison. Default is `normal=False`.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot and display a basic standardised residuals histogram and save it to file as `my_std_residuals_plot.png`:
`my_pcm.std_residuals_plot(filename='my_std_residuals_plot')`

To plot and display a standardised residuals histogram with bin width 1, with standard normal curve:
`self.std_residuals_plot(bin_width=1, normal=True)`

To plot and display a standardised residuals histogram on a subset of items stored as a list in a variable `my_item_list`:
`my_pcm.std_residuals_plot(items=my_item_list)`

5 class RSM

5.1 Preliminaries

5.1.1 RSM

Description

Creates an object of the class RSM from a pandas dataframe of polytomously scored data of items which share the same maximum score and Rasch-Andrich threshold structure for analysis. No analysis can be run until an object is created.

Usage

`RSM(dataframe, max_score=None, extreme_persons=True, no_of_classes=5)`

Arguments

dataframe	pandas dataframe with items as columns (item names as column names) and persons as index (person names as row names).
max_score	Integer: The maximum possible score, shared across all items. If no score is passed, max_score will be inferred from the data, although passing an argument is recommended. Default is max_score=None.
extreme_persons	Boolean: if False, all persons with extreme scores (all responses correct or all responses incorrect) are removed from the response dataframe. Default is extreme_persons=True.
no_of_classes	Integer: the number of classes of persons grouped by ability for overplotting observed responses on theoretical curves. Default is no_of_classes=5

Returns

Object of class RSM. Analyses are run using methods defined on the RSM object, with results stored as attributes of the RSM object.

Several attributes of object RSM are automatically generated on its creation:

`self.dataframe`	pandas dataframe: Dataframe of valid responses.
`self.invalid_responses`	pandas dataframe: Dataframe of invalid responses (persons with no responses to any items, i.e. all missing data).
`self.max_score`	Integer: The maximum possible score, shared across all items.
`self.no_of_items`	Integer: Number of items.
`self.items`	List: List of item names.
`self.no_of_persons`	Integer: Number of persons.
`self.persons`	List: List of person names.

Example

To create an object from a dataframe `my_rsm_dataframe`, with a maximum score of 5 and 10 observed classes:

```
my_rsm = RSM(my_rsm_dataframe, max_score=5, no_of_classes=10)
```

5.1.2 `rename_item`

Description

Method to rename a single item.

Usage

```
self.rename_item(old, new)
```

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.dataframe` with new name.

Example

To rename an item in object `my_rsm` from `Item_1` to `my_new_item_name`:

```
my_rsm.rename_item('Item_1', 'my_new_item_name')
```

5.1.3 `rename_items_all`

Description

Method to rename all items.

Usage

```
self.rename_items_all(new_names)
```

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.dataframe` with new names.

Example
To rename all items in object `my_rsm` with item names in a list stored as a variable `my_new_item_names`:

```
my_rsm.rename_items_all(my_new_item_names)
```

5.1.4 `rename_person`

Description
Method to rename a single person.

Usage
```
self.rename_person(old, new)
```

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns
Replaces specified person name in the index of `self.dataframe` with new name.

Example
To rename a person in object `my_rsm` from `Person_1` to `my_new_person_name`:

```
my_rsm.rename_person('Person_1', 'my_new_person_name')
```

5.1.5 `rename_persons_all`

Description Method to rename all persons.

Usage
```
self.rename_persons_all(new_names)
```

Arguments

`new_names`	List of new person names as strings

Returns
Replaces all person names in the index of `self.dataframe` with new names.

Example

To rename all persons in object `my_rsm` with person names in a list stored as a variable `my_new_person_names`:

```
my_rsm.rename_persons_all(my_new_person_names)
```

5.2 Core functions

5.2.1 `cat_prob`

Description

Category probability function which calculates the probability $P(X_{ni} = k)$ of scoring k, with $k \in \{0,\ m\}$, where m is the maximum score, from person ability, central item difficulty and Rasch-Andrich thresholds. For a person n with ability β_n attempting an item i with central item difficulty δ_i and Rasch-Andrich thresholds $\{\tau_0, ..., \tau_m\}$, the probability of obtaining a score of k is given by:

$$P(X_{ni} = k) = \frac{e^{k(\beta_n - \delta_i) - \sum_{t=0}^{k} \tau_t}}{\sum_{k=0}^{m} e^{k(\beta_n - \delta_i) - \sum_{t=0}^{k} \tau_t}}$$

In this formulation, an item is defined by a central item difficulty, δ_i and a set of centred Rasch-Andrich thresholds $\{\tau_k\}$, $k \in \{0,\ ...,\ m\}$ which sum to zero: an alternative formulation would be to define the item solely by m uncentred thresholds, $\{\tau'_{ik}\}$, $k \in \{1,\ ...,\ m\}$, where $\tau'_{ik} = \delta_i + \tau_k$, in analogy with the partial credit model formulation described in Section **??**, but we will use the centred thresholds formulation throughout here, apart from in item plots where absolute threshold location is salient.

Usage

`self.cat_prob(ability, difficulty, category, thresholds)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty
`category`	Integer: Response category k, with $k \in \{0,\ 1\}$.
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns

Float: probability of obtaining score k.

Example

To obtain the probability of a person of ability 0.5 scoring 0 on an item with central difficulty 0 and a set of Rasch-Andrich thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_cat_prob`:

```
my_cat_prob = self.cat_prob(0.5, 0, 0, my_thresholds)
```

5.2.2 `exp_score`

Description

Expected score function which calculates the expected score $E(X_{ni})$ from person ability, central item difficulty and set of Rasch-Andrich thresholds. The expected score is given by:

$$E(X_{ni}) = \sum_{k=0}^{1} kP(X_{ni} = k)$$

where $P(X_{ni} = k)$ is as described in Section 5.2.1.

Usage

`self.exp_score(ability, difficulty, thresholds)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns

Float: expected score.

Example

To obtain the expected score for a person of ability 0.5 attempting an item of difficulty 0 with a set of Rasch-Andrich thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_exp_score`:

```
my_exp_score = self.exp_score(0.5, 0, my_thresholds)
```

5.2.3 `variance`

Description

Variance function which calculates the variance of the score $V(X_{ni})$ from person ability, central item difficulty and set of Rasch-Andrich thresholds. The variance is given by:

$$V(X_{ni}) = \sum_{k=0}^{1} P(X_{ni} = k)(k - E(X_{ni}))^2$$

where $P(X_{ni} = k)$ and $E(X_{ni})$ are as described in Sections 5.2.1 and 5.2.2 respectively.

The variance is also both the Fisher information for the response and the first partial differential of the expected score function with respect to person ability.

Usage

`self.variance(ability, difficulty, thresholds)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns

Float: variance (also the Fisher information provided by the response).

Example

To obtain the variance for a person of ability 0.5 attempting an item with central difficulty 0 and a set of Rasch-Andrich thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_variance`:

```
my_variance = self.variance(0.5, 0, my_thresholds)
```

5.2.4 `kurtosis`

Description

Kurtosis function which calculates the kurtosis of the score $\kappa(X_{ni})$ from person ability, central item difficulty and Rasch-Andrich thresholds. The variance is given by:

$$\kappa(X_{ni}) = \sum_{k=0}^{1} P(X_{ni} = k)(k - E(X_{ni}))^4$$

where $P(X_{ni} = k)$ and $E(X_{ni})$ are as described in Sections 5.2.1 and 5.2.2 respectively.

Usage

`self.kurtosis(ability, difficulty, thresholds)`

Arguments

`ability`	Float: Person ability
`difficulty`	Float: Item difficulty
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Returns
Float: kurtosis

Example
To obtain the kurtosis for a person of ability 0.5 attempting an item of difficulty 0 with a set of Rasch-Andrich thresholds stored as a variable `my_thresholds`, and store the result as a variable `my_kurtosis`:

```
my_kurtosis = self.kurtosis(0.5, 0, my_thresholds)
```

5.3 Parameter estimation

5.3.1 `calibrate`

Description

Produces central item difficulty and Rasch-Andrich threshold estimates using the conditional pairwise estimation (CPAT) algorithm (Elliott & Buttery, 2022b).

Usage

`self.calibrate(constant=0.1, method='cos', matrix_power=3, log_lik_tol=0.000001)`

Arguments

`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of central item difficulty estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) for central item difficulty estimates is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for central item difficulty estimates. Ignored for other methods.

Returns

Returns three attributes:

`self.diffs`	pandas series: Item difficulty estimates with the item names as keys and estimates as values.
`self.thresholds`	numpy array: Rasch-Andrich threshold estimates, an array of $m+1$ estimates, where m is the maximum score, which sum to zero and the first of which is zero.
`self.cat_widths`	numpy array: Array of $m-1$ category width estimates, where m is the maximum score.

Examples

To generate a set of estimates using the cosine similarity method for central item difficulties, with additive smoothing constant of 0.1:

```
self.calibrate()
```

To generate a set of estimates using the log-likelihood method for central item difficulties, with matrix raised to power 7 and a convergence stopping criterion of 0.00000001:

```
self.calibrate(method='log-lik', matrix_power=7, log_lik_tol=0.00000001)
```

5.3.2 `std_errors`

Description

Produces bootstrapped estimates for the standard errors of central item difficulty estimates, threshold estimates and category width estimates for bounded (non-extreme) categories.

Usage

```
self.std_errors(interval=None, no_of_samples=100, constant=0.1, method='cos',
                matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval. Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.
`no_of_samples`	Integer: Number of bootstrap samples to generate. More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of central item difficulty estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`

Arguments continue on the next page.

Arguments (continued)

`matrix_power`	Integer: power to which conditional category response frequency matrix for central item difficulty estimates (Elliott & Buttery, 2022b:991) is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for central item difficulty estimates. Ignored for other methods.

Returns

Attributes:

`self.item_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each item estimate.
`self.item_se`	pandas series: Item names as keys and item standard errors as values.
`self.threshold_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each threshold estimate.
`self.threshold_se`	pandas series: Threshold numbers as keys and threshold standard errors as values.
`self.cat_width_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each category width estimate.
`self.cat_width_se`	pandas series: Category numbers as keys and category width standard errors as values.

If an argument is passed to `interval`, also returns:

`self.item_low`	Lower bound of the specified interval for item estimates.
`self.item_high`	Upper bound of the specified interval for item estimates.
`self.threshold_low`	Lower bound of the specified interval for threshold estimates.
`self.threshold_high`	Upper bound of the specified interval for threshold estimates.
`self.cat_width_low`	Lower bound of the specified interval for category estimates.
`self.cat_width_high`	Upper bound of the specified interval for category estimates.

Example

To generate item standard errors with a 95% interval from 200 samples:

```
self.std_errors(interval=0.95, no_of_samples=200)
```

Modifications to the estimation method are discussed in Section 5.3.1.

5.3.3 abil

Description

Generates an ability estimate for a person using the Newton-Raphson method to produce a maximum likelihood estimate, with optional Warm bias correction (Warm, 1989).

Usage

```
self.abil(person, items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
        ext_score_adjustment=0.5)
```

Arguments

person	String: The person name for the ability being estimated.
items	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
warm_corr	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
tolerance	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
max_iters	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
ext_score_adjustment	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns

Float: person ability estimate.

Examples

To generate a person ability estimate for `Person_1` using the default settings and store the result as a variable, `my_person_ability`:

```
my_person_ability = my_person_ability = self.abil('Person_1')
```

To generate an MLE person ability estimate without Warm bias correction for `Person_1` based on the first three items and store the result as a variable, `my_person_ability`:

```
my_person_ability = self.abil('Person_1', ['Item_1', 'Item_2', 'Item_3'], warm_corr=False)
```

5.3.4 `person_abils`

Description
Generates ability estimates for all persons using the Newton-Raphson method to produce maximum likelihood estimates, with optional Warm bias correction (Warm, 1989).

Usage
```
self.person_abils(items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
                  ext_score_adjustment=0.5)
```

Arguments

`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Attribute `self.person_abilities`: pandas series with person names as keys and ability estimates as values.

Examples
To generate a set of person ability estimates with Warm bias correction:

```
self.person_abils()
```

To generate a set of person ability estimates without Warm bias correction, on a subset of the first three items only:

```
self.person_abils(items=['Item_1', 'Item_2', 'Item_3'], warm_corr=False)
```

5.3.5 `score_abil`

Description
Generates an ability estimate for a given raw score on responses to a given set of items using the Newton-Raphson method to produce a maximum likelihood estimate, with optional Warm bias correction (Warm, 1989).

Usage

```
self.score_abil(score, items=None, warm_corr=True, tolerance=1e-07, max_iters=100,
          ext_score_adjustment=0.5)
```

Arguments

`score`	Integer: The raw score for which ability is being estimated.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
pandas series with raw scores as keys and person ability estimates as values.

Examples

To generate an ability estimate for a score of 10 on all items, with Warm bias correction, and store the result as a variable, `my_score_ability`:

```
my_score_ability = self.score_abil(10)
```

To generate an ability estimate for a score of 10 on a subset of items saved as a variable `my_items`, without Warm bias correction, and store the result as a variable, `my_score_ability`: `my_score_ability = self.score_abil(10, items=my_items, warm_corr=False)`

5.3.6 `abil_lookup_table`

Description

Generates a lookup table of ability estimates corresponding to all available raw scores on a set of items with no missing responses, using the Newton-Raphson method to produce maximum likelihood estimates and with optional Warm bias correction (Warm, 1989).

Usage

```
self,abil_lookup_table(items=None, ext_scores=True, warm_corr=True, tolerance=1e-07,
                   max_iters=100, ext_score_adjustment=0.5)
```

Arguments

`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`ext_scores`	Boolean: If `True`, ability estimates for extreme scores (all correct/all incorrect) will be generated using the `ext_score_adjustment` argument. Default is `ext_scores=True`.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Attribute `self.abil_table`: pandas series with raw scores as keys and corresponding ability estimates as values.

Examples
To generate an ability lookup table for all items, including extreme scores, with Warm bias correction:

```
self.abil_lookup_table()
```

To generate an ability lookup table for a subset of items saved as a variable `my_items`), without extreme scores and without Warm bias correction:

```
self.abil_lookup_table(items=my_items, ext_scores=False)
```

5.3.7 `csem`

Description
Calculates conditional standard error of measurement for a person.

Usage
```
self.csem(person, abilities=None, items=None)
```

Arguments

`person`	Person name.
`abilities`	pandas series (or dictionary) with person names as keys and abilities as values. If `None`, uses `self.person_abilities`, automatically generating if necessary. Default is `self.person_abilities=None`.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.

Returns
Float: conditional standard error of measurement for ability estimate.

Examples
To generate the CSEM for `Person_1` on all items and save the result as a variable, `my_csem`:

```
my_csem = self.csem('Person_1')
```

To generate the CSEM for a raw score of 3 on a subset of items saved to a variable `my_items` and save the result as a variable, `my_csem`:

```
my_csem = self.csem(3, abilities=self.abil_table, items=my_items)
```
where `self.abil_table` is the output from `self.abil_lookup_table(items=my_items)` (see Section 6.3.7).

5.4 Statistical output

5.4.1 item_stats_df

Description

Produces a table of item statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.item_stats_df(full=False, zstd=False, point_measure_corr=False, dp=3, warm_corr=True,
                   tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5, method='cos',
                   constant=0.1, no_of_samples=100, interval=None)
```

Arguments

full	Boolean: If True, a full table with all available statistics is produced. Default is full=False for a summary table with the most commonly reported statistics.
zstd	Boolean: If True, infit and outfit standardised z-scores are reported alongside mean square statistics. Default is zstd=False. Automatically True if full=True.
point_measure_corr	Boolean: If True, point-polyserial correlation (Kornbrot, 2014) between person ability and item score is reported, together with the expected value of the point-measure polyserial correlation for an ideal item. Default is point_measure_corr=False. Automatically True if full=True.
dp	Integer: Number of decimal places reported in output.
warm_corr	Boolean: If True, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is warm_corr=True.
tolerance	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is tolerance=1e-07.
max_iters	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is max_iters=100.

Arguments continue on the next page.

Arguments (continued)

`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during item estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for item estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute `self.item_stats`, a pandas dataframe with one row for each item and the following columns:

`Estimate`	Central item difficulty estimate.
`SE`	Bootstrapped standard error of central item difficulty estimate.
`2.5%`	Lower bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 2.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `5%`.

Returns continue on the next page.

Returns (continued)

`97.5%`	Upper bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 97.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `95%`.
`Count`	Count of responses.
`Facility`	Item facility: proportion of correct responses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score. Only produced if `zstd=True`.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score. Only produced if `zstd=True`.
`PM corr`	Point-polyserial correlation. Only produced if `point_measure_corr=True`.
`Exp PM corr`	Expected value of point-measure polyserial correlation. Only produced if `point_measure_corr=True`.
`Disordered`	If `True`, threshold estimates indicate the presence of disordered thresholds within the item (Andrich, 2010).

Examples

To produce a summary `item_stats` table with the most commonly reported statistics:

```
my_rsm.item_stats_df()
```

To produce a full `item_stats` table with all statistics:

```
my_rsm.item_stats_df(full=True)
```

To produce an `item_stats` table with with the most commonly reported statistics, plus standardised z-scores for infit and outfit:

```
my_rsm.item_stats_df(zstd=True)
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

5.4.2 `threshold_stats_df`

Description

Produces a table of threshold statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.threshold_stats_df(full=False, zstd=False, disc=False, point_measure_corr=False, dp=3,
                        warm_corr=True, tolerance=1e-07, max_iters=100,
                        ext_score_adjustment=0.5, method='cos', constant=0.1,
                        no_of_samples=100, interval=None)
```

Arguments

`full`	Boolean: If `True`, a full table with all available statistics is produced. Default is `full=False` for a summary table with the most commonly reported statistics.
`zstd`	Boolean: If `True`, infit and outfit standardised z-scores are reported alongside mean square statistics. Default is `zstd=False`. Automatically `True` if `full=True`.
`disc`	Boolean: If `True`, item discrimination is reported. The discrimination of the empirical item slope relative to the ideal logistic ogive, with 1 perfect, greater than 1 showing overfit and less than 1 showing underfit; discrimination is similar to the 2PL IRT discrimination parameter (Linacre, 2023), but is a descriptive statistic in the SLM rather than an item parameter.
`point_measure_corr`	Boolean: If `True`, point-biserial correlation (Kornbrot, 2014) between person ability and item score is reported, together with the expected value of the point-measure biserial correlation for an ideal item. Default is `point_measure_corr=False`. Automatically `True` if `full=True`.
`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.

Arguments continue on the next page.

Arguments (continued)

`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 5.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute `self.threshold_stats`, a pandas dataframe with one row for each threshold and the following columns:

`Estimate`	Rasch-Andrich threshold estimate.
`SE`	Bootstrapped standard error of threshold estimate.
`2.5%`	Lower bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 2.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `5%`.

Returns continue on the next page.

Returns (continued)

`97.5%`	Upper bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 97.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `95%`.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score. Only produced if `zstd=True`.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score. Only produced if `zstd=True`.
`Discrim`	Item discrimination. Only produced if `disc=True`.
`PM corr`	Point-biserial correlation. Only produced if `point_measure_corr=True`.
`Exp PM corr`	Expected value of point-measure biserial correlation. Only produced if `point_measure_corr=True`.

Examples

To produce a summary `threshold_stats` table with the most commonly reported statistics:

```
my_rsm.threshold_stats_df()
```

To produce a full `threshold_stats` table with all statistics:

```
my_rsm.threshold_stats_df(full=True)
```

To produce an `threshold_stats` table with with the most commonly reported statistics, plus standardised z-scores for infit and outfit:

```
my_rsm.threshold_stats_df(zstd=True)
```

Other arguments may be used to alter parameters of central item difficulty, threshold and person ability estimation.

5.4.3 `person_stats_df`

Description

Produces a table of person statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.person_stats_df(full=False, rsem=False, dp=3, warm_corr=True, tolerance=1e-07,
                     max_iters=100, ext_score_adjustment=0.5, method='cos', constant=0.1)
```

Arguments

full	Boolean: If True, a full table with all available statistics is produced. Default is full=False for a summary table with the most commonly reported statistics.
rsem	Boolean: If True, realistic standard error of measurement (RSEM), which takes into account for item misfit (Wright, 1996), is reported alongside the conditional standard error of measurement (CSEM). Default is rsem=False. Automatically True if full=True.
dp	Integer: Number of decimal places reported in output.
warm_corr	Boolean: If True, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is warm_corr=True.
tolerance	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is tolerance=1e-07.
max_iters	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is max_iters=100.
ext_score_adjustment	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to ext_score_adjustment (if zero) or the maximum possible score minus ext_score_adjustment (if maximum score) before estimating ability.
method	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 3.3.1). Options are 'ls' for least squares (Choppin, 1968, 1985), 'evm' for the eigenvector method(Garner & Engelhard, 2002), 'cos' for cosine similarity (Kou & Lin, 2014) or 'log-lik' for (iterative) log-likelihood (Bradley & Terry, 1952). Default is method='cos'
constant	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is constant=0.1.
no_of_samples	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is no_of_samples=100.

Arguments continue on the next page.

Arguments (continued)

`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute `self.person_stats`, a pandas dataframe with one row for each person and the following columns:

`Estimate`	Item difficulty estimate.
`CSEM`	Conditional standard error of measurement for person ability estimate.
`RSEM`	Realistic standard error of measurement for person ability estimate. Only produced if `rsem=True`
`Score`	Number of correct responses.
`Max score`	Maximum available score (number of items attempted).
`p`	Proportion of correct repsonses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score.

Examples

To produce a summary `person_stats` table with the most commonly reported statistics:

```
my_rsm.person_stats_df()
```

To produce a full `person_stats` table with all statistics:

```
my_rsm.person_stats_df(full=True)
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

5.4.4 `test_stats_df`

Description

Produces a table of test-level statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

```
self.test_stats_df(dp=3, warm_corr=True, tolerance=1e-07, max_iters=100,
                   ext_score_adjustment=0.5, method='cos', constant=0.1)
```

Arguments

`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is `constant=0.1`.

Returns

Attribute `self.test_stats`, a pandas dataframe with two columns, `Items` and `Persons` and rows for a range of descriptive statistics describing the distributions of the estimates and different statistics related to reliability – these statistics describe the suitability of the data for estimating and differentiating the parameters, rather than properties of the parameters themselves. The statistics are:

`Mean`	The mean of the estimates.
`SD`	The standard deviation of the estimates.
`Separation ratio`	The separation ratio (Wright, 1996; Wright & Masters, 1982), which is the standard deviation of person abilities reported as a ratio of standard error units. For persons: $$G_p = \sigma_p / \sqrt{\sum_n SE_n^2}$$ where σ_p is the variance of the person estimates and SE_n is the RSEM (see Section 3.4.2) for person n. The formula is symmetrical for items, substituting the standard error of estimation for RSEM.
`Strata`	The number of statistically distinct levels of either person ability or item difficulty as strata with centers three measurement errors apart (Wright & Masters, 1982:106). For persons: $$H_p = (4G_p + 1)/3$$ with symmetrical results for items.
`Reliability`	A Rasch-specific reliability statistic (Wright, 1996), derived from PSI and which is a Rasch-specific reliability statistic similar to Cronbach's Alpha (Cronbach, 1951), and which may be interpreted the same way – as the proportion of variance of the estimates which stems from variation in ability or difficulty rather than estimation error. For persons: $$R_p = G_p^2/(1 + G_p^2)$$ with symmetrical results for items.

Example

To produce a `self.test_stats` table:

```
my_rsm.test_stats_df()
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

5.4.5 `res_corr_analysis`

Description

Analysis of correlations of standardised residuals to tests for violations of local item interdependence and unidimensionality requirements.

Usage

```
self.res_corr_analysis(warm_corr=True, tolerance=1e-07, max_iters=100,
                       ext_score_adjustment=0.5, constant=0.1, method='cos',
                       matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for generation of item difficulty estimates (see Section 3.3.1). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) for generation of central item difficulty estimates (see Section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) is raised for generation of central item difficulty estimates (see Section 3.3.1). Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for generation of item difficulty estimates (see Section 3.3.1). Ignored for other methods.

Returns

For tests of violation of the requirement for local item independence (Andrich & Kreiner, 2010; Marais, 2012):

`self.residual_correlations`	A pandas dataframe of pairwise correlations between item standardised residuals.

For tests of violation of the requirement for unidimensionality based on principal component analysis of the standardised residual correlations (Pallant & Tennant, 2007; Smith, 2002):

`self.eigenvectors`	The eigenvectors of the standardised residual correlations matrix.
`self.eigenvalues`	The eigenvalues corresponding to the eigenvectors.
`self.variance_explained`	The variance explained by each principal component.
`self.loadings`	The loading of each item onto each of the principal components, for the the first of which large loadings ('large'typically interpreted as > 0.4 or < -0.4) may be interpreted as representing the presence of significant dimensionality, in analogy to factor analysis (**<empty citation>**).

Example
To produce a residual correlation analysis:

`my_rsm.res_corr_analysis()`

Arguments may be used to alter parameters of item difficulty and/or person ability estimation.

5.4.6 `category_counts_df`

Description
Produces a table of counts of scores in each category, plus responses and missing responses, for each item.

Usage
`self.category_counts_df()`

Arguments None

Returns
Attribute `self.category_counts`, a pandas dataframe of category counts with one row per item and one column per response category, plus total responses per item and missing responses per item.

Example
To produce a dataframe of category counts:

`my_rsm.category_counts_df()`

5.5 Plotting functionality

5.5.1 Shared plotting arguments

All the plotting methods described in this section share a set of arguments which may be used to customise the appearance of the plot or save the plot to file automatically. These arguments are:

`title`	String: Title for the plot, to appear in the image. Default is `title=None`.
`xmin`	Float: Minimum displayed point on x-axis, in logits. Default is `xmin=-5`.
`xmax`	Float: Maximum displayed point on x-axis, in logits. Default is `xmax=5`.
`plot_style`	String: Plot style to use. Available styles are `'white'`, producing a plot on a white background, and `'dark'`, producing a plot on a grey background. The styles correspond to Seaborn (Waskom, 2021) styles `whitegrid` and `darkgrid`. Default is `plot_style='white'`.
`palette`	String: Controls colours of lines and overplotted class intervals dots. Options are `'dark blue'`, `'light blue'`, `'dark red'`, `'light red'`, `'dark green'`, `'light green'`, `'dark grey'`, `'light grey'`, `'dark multi'` and `'light multi'`. Default is `palette='dark blue'`. Not used for `std_residual_plot`.
`black`	Boolean: If `True`, the plot will be rendered in black and white. Default is `black=False`.
`font`	String: The font to use in the plot. Default is `font='Times'`.
`title_font_size`	Float: The size of the title font in points. Default is `title_font_size=15`.
`axis_font_size`	Float: The size of the axis label font in points. Default is `axis_font_size=15`.
`labelsize`	Float: The size of the axis tick label font in points. Default is `labelsize=15`.
`filename`	String: The filename for the saved plot, with no suffix for format. If `None`, no file will be saved. Default is `filename=None`.
`file_format`	The format of the file: `png`, `jpg` or `svg`. Default is `file_format=png`.
`dpi`	The resolution of the plot in dpi (dots per inch) – higher resolution plots are better quality but have larger file sizes. Default is `dpi=300`.

5.5.2 `icc`

Description

Plots the item characteristic curves (or item response function) for an item: person ability on the x-axis against expected score on the y-axis. Options to plot observed responses, item threshold line and lines showing abilities corresponding to specified expected scores, and to highlight a specified response category.

Usage

```
self.icc(item, obs=False, no_of_classes=5, thresh_lines=False, central_diff=False,
         score_lines=None, score_labels=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`obs`	Boolean: If `True`, mean observed scores for each of the ordered response categories will be plotted against the mean ability of the corresponding response class. Default is `obs=False`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`score_lines`	List of floats between 0 and `self.max_score`: Each float in the list represents an expected score, for which a horizontal line from the y-axis to where it intersects the item characteristic curve will be plotted, and from there a vertical line down to the x-axis will be plotted to show the ability corresponding to the expected score.
`score_labels`	Boolean: If `True`, scores and abilities corresponding to arguments passed to `score_lines` will be labelled on the plot. Default is `score_labels=False`.
`cat_highlight`	Integer: Passing a score between 0 and `self.max_score` will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot a basic item characteristic curve for `Item_1` and store the output as a variable `my_icc_plot` and save it to file as `my_icc_plot.png`:

```
my_icc_plot = self.icc('Item_1', filename=my_icc_plot)
```

To plot an item characteristic curve for `Item_1` with observed responses for 8 response classes and store the output as a variable `my_icc_plot`:

```
my_icc_plot = my_rsm.icc('Item_1', obs=True, no_of_classes=8)
```

To plot an item characteristic curve for `Item_1` with a threshold line and highlighted zero category, and store the output as a variable `my_icc_plot`:

```
my_icc_plot = my_rsm.icc('Item_1', thresh_line=True, cat_highlight=0)
```

To plot an item characteristic curve for `Item_1` with lines showing the abilities corresponding to expected scores of 0.7 and 1.6, with the expected score and corresponding ability labelled, and store the output as a variable `my_icc_plot`:

```
my_icc_plot = my_rsm.icc('Item_1', score_lines=[0.7, 1.6], score_labels=True)
```

5.5.3 `crcs`

Description

Plots category response curves for an item: person ability on the x-axis against expected the probability of obtaining a score in each category (0 or 1) on the y-axis. Options to plot observed proportions and item threshold line, and to highlight a specified response category.

Usage

```
self.crcs(item=None, obs=None, no_of_classes=5, thresh_lines=False, central_diff=False,
          cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot. If `item=None`, the shared set of Rasch-Andrich thresholds with mean zero will be plotted, and any observed proportions overplotted using `obs` will use data from all items. Default is `item=None`.
`obs`	List: List of integers between 0 and `self.max_score`. For each value, mean observed proportions in each ordered response category scoring in that category are plotted against the mean ability of the corresponding response class.

Arguments continue on the next page.

Arguments (continued)

`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`cat_highlight`	Integer: Passing a score between 0 and `self.max_score` will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot basic category response curves and store the output as a variable `my_crcs_plot` and save it to file as `my_crcs_plot.png`:

```
my_crcs_plot = my_rsm.crcs(filename=my_crcs_plot)
```

To plot category response curves for `Item_1` with observed response proportions for category 0 for 8 response classes and store the output as a variable `my_crcs_plot`:

```
my_crcs_plot = my_rsm.crcs(item='Item_1', obs=[0], no_of_classes=8)
```

To plot category response curves for `Item_1` with a threshold line and highlighted zero category, and store the output as a variable `my_crcs_plot`:

```
my_crcs_plot = my_rsm.crcs(item='Item_1', thresh_line=True, cat_highlight=0)
```

5.5.4 `threshold_ccs`

Description

Plots the threshold characteristic curves (or threshold response functions)for an item. For threshold τ_k, $k \in \{1, ..., \texttt{self.max\_score}\}$, the threshold characteristic curve is the probability of obtaining a score of k rather than $k-1$, conditional on the score being either $k-1$ rather than k, for a given person ability. Each threshold characteristic curve functions as a dichotomous item characteristic curve under the SLM (see Sections 3.2.2 and 3.5.2).

For threshold τ_k, `threshold_ccs` plots person ability on the x-axis against the probability of obtaining a score of k on the y-axis. Options to plot threshold lines and central item difficulties, and to highlight a specified response category.

Usage

```
self.threshold_ccs(item=None, obs=None, no_of_classes=5, thresh_lines=False,
                   central_diff=False, cat_highlight=None,)
```

Arguments

`item`	String: The name of the item to plot. If `item=None`, the shared set of Rasch-Andrich thresholds with mean zero will be plotted, and any observed proportions overplotted using `obs` will use data from all items. Default is `item=None`.
`obs`	List: List of integers corresponding to thresholds τ_1 to τ_m, where m =`self.max_score`, or `'all'` or `'none'`. If `obs=[k]`, mean proportions of persons obtaining a score of k rather than $k-1$, conditional on the score being either $k-1$ or k, for each of the ordered response categories, will be plotted against the mean ability of the corresponding response class. Multiple thresholds may be passed. Default is `obs=None`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`cat_highlight`	Integer: Passing 0 or 1 will highlight the range of abilities for which the selected score is the most probable response (all abilities to one side of the item difficulty. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot basic threshold characteristics curves, store the output as variable `my_threshold_ccs_plot`, and save it to file as `my_threshold_ccs_plot.png`:

```
my_threshold_ccs_plot = my_rsm.threshold_ccs(filename='my_iic_plot')
```

To plot threshold characteristics curves for `Item_1` with threshold lines and category 1 highlighted, and store the output as a variable `my_threshold_ccs_plot`:

```
my_threshold_ccs_plot = my_rsm.threshold_ccs(item='Item_1', thresh_lines=True,
                                             cat_highlight=1)
```

To plot threshold characteristics curves for `Item_1` with observed responses plotted for thresholds 2 and 4 and central item difficulty line, and store the output as a variable `my_threshold_ccs_plot`:

```
my_threshold_ccs_plot = my_rsm.threshold_ccs(item='Item_1', obs=[2, 4], central_diff=True)
```

5.5.5 `iic`

Description

Plots the item information curve for an item: person ability on the x-axis against Fisher information on the y-axis. Options to plot item threshold line and lines showing Fisher information corresponding to specified abilities, and to highlight a specified response category.

Usage

```
self.iic(item, ymax=None, thresh_lines=False, central_diff=False, point_info_lines=None,
         point_info_labels=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`ymax`	Float: The maximum value to show on the y-axis. If `None`, will infer, plotting a maximum of 1.1 times the maximum item information. Default is `ymax=None`

Arguments continue on the next page.

Arguments (continued)

`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`point_info_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the item information curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the Fisher information corresponding to the ability. Default is `point_info_lines=None`.
`point_info_labels`	Boolean: If `True`, abilities and Fisher information corresponding to arguments passed to `point_info_lines` will be labelled on the plot. Default is `point_info_labels=False`.
`cat_highlight`	Integer: Passing a score between 0 and `self.max_score` will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is `cat_highlight=None` (no category highlighted).
`ymax`	The maximum point displayed on the y-axis, in Fisher information.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot a basic item information curve for `Item_1` and store the output as a variable `my_iic_plot` and save it to file as `my_iic_plot.png`:

```
my_iic_plot = my_rsm.iic('Item_1', filename='my_iic_plot')
```

To plot an item information curve for `Item_1` with threshold lines, central item difficulty line and category 1 highlighted, and store the output as a variable `my_iic_plot`:

```
my_iic_plot = my_rsm.iic('Item_1', thresh_line=True, central_diff=True, cat_highlight=1)
```

To plot an item information curve for `Item_1` with lines showing the Fisher information corresponding to abilities of -0.3 and 0.7, with the ability and corresponding Fisher information labelled, and store the output as a variable `my_iic_plot`:

```
my_iic_plot = my_rsm.icc('Item_1', point_info_lines=[-0.3, 0.7], point_info_labels=True)
```

5.5.6 `tcc`

Description

Plots the test characteristic curve (or test response function) for a set of items: person ability on the x-axis against expected score on the y-axis. Options to plot observed responses and lines showing abilities corresponding to specified expected scores.

Usage

```
self.tcc(items=None, obs=False, no_of_classes=5, score_lines=None, score_labels=False)
```

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`obs`	Boolean: If `True`, mean observed scores for each of the ordered response categories will be plotted against the mean ability of the corresponding response class. Default is `obs=False`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`score_lines`	List of floats between 0 and `self.max_score`: Each float in the list represents an expected score, for which a horizontal line from the y-axis to where it intersects the item characteristic curve will be plotted, and from there a vertical line down to the x-axis will be plotted to show the ability corresponding to the expected score. Default is `score_lines=None`.
`score_labels`	Boolean: If `True`, scores and abilities corresponding to arguments passed to `score_lines` will be labelled on the plot. Default is `score_labels=False`.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot a basic test characteristic curve for all items and store the output as a variable `my_tcc_plot` and save it to file as `my_tcc_plot.png`:

```
my_tcc_plot = my_rsm.tcc(filename=my_tcc_plot)
```

To plot a test characteristic curve for `Item_1` for a subset of items stored as a list **`my_item_list`**, with observed responses for 8 response classes and store the output as a variable `my_tcc_plot`:

```
my_tcc_plot = my_rsm.tcc(obs=True, no_of_classes=8)
```

To plot a test characteristic curve for `Item_1` for all items with lines showing the abilities corresponding to expected scores of 13 and 20, with the expected score and corresponding ability labelled, and store the output as a variable `my_tcc_plot`:

```
my_tcc_plot = my_rsm.tcc(score_lines=[13, 20], score_labels=True)
```

5.5.7 `test_info`

Description

Plots the test information curve: person ability on the x-axis against total Fisher information on the y-axis. Option to plot lines showing Fisher information corresponding to specified abilities.

Usage

```
self.test_info(items=None, ymax=None, point_info_lines=None, point_info_labels=False)
```

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`point_info_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the item information curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the total Fisher information corresponding to the ability. Default is `point_info_lines=None`.
`point_info_labels`	Boolean: If `True`, abilities and total Fisher information corresponding to arguments passed to `point_info_lines` will be labelled on the plot. Default is `point_info_labels=False`.
`ymax`	The maximum point displayed on the y-axis, in Fisher information.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns

matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot a basic test information curve and store the output as a variable `my_test_info_plot` and save it to file as `my_test_info_plot.png`:

```
my_test_info_plot = my_rsm.test_info(filename='my_test_info_plot')
```

To plot a test information curve for a subset of items stored as a list **`my_item_list`** and store the output as a variable `my_test_info_plot`:

```
my_test_info_plot = my_rsm.test_info(items=my_item_list)
```

To plot a test information curve with lines showing the total Fisher information corresponding to abilities of -0.3 and 0.7, with the ability and corresponding total Fisher information labelled, and store the output as a variable `my_test_info_plot`:

```
my_test_info_plot = my_rsm.test_info(point_info_lines=[-0.3, 0.7], point_info_labels=True)
```

5.5.8 `test_csem`

Description
Plots the test conditional standard error of measurement (CSEM) curve: person ability on the x-axis against CSEM (in logits) on the y-axis. Option to plot lines showing CSEM corresponding to specified abilities.

Usage
```
self.test_csem(items=None, point_csem_lines=None, point_csem_labels=False, ymax=5)
```

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`point_csem_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the CSEM curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the CSEM corresponding to the ability. Default is `point_csem_lines=None`.
`point_csem_labels`	Boolean: If `True`, abilities and CSEM corresponding to arguments passed to `point_csem_lines` will be labelled on the plot. Default is `point_csem_labels=False`.
`ymax`	The maximum point displayed on the y-axis, in logits.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot a basic CSEM curve and store the output as a variable `my_test_csem_plot` and save it to file as `my_test_csem_plot.png`:

```
my_test_csem_plot = my_rsm.test_csem(filename='my_test_csem_plot')
```

To plot a CSEM curve for a subset of items stored as a list `my_item_list` and store the output as a variable `my_test_csem_plot`:

```
my_test_csem_plot = my_rsm.test_csem(items=my_item_list)
```

To plot a CSEM curve with lines showing the CSEM corresponding to abilities of -0.3 and 0.7, with the ability and corresponding CSEM labelled, and store the output as a variable `my_test_csem_plot`:

```
my_test_csem_plot = my_rsm.test_csem(point_csem_lines=[-0.3, 0.7], point_csem_labels=True)
```

5.5.9 `std_residuals_plot`

Description
Plots histogram of standardised residuals, with optional overplotting of standard Normal distribution.

Usage
`self.std_residuals_plot(items=None, bin_width=0.5, normal=False)`

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`bin_width`	Float: The width of the histogram bins along the x-axis. Default is `bin_width=0.5`.
`normal`	Boolean: If `True`, plots a standard normal distribution over the standardised residual histogram for comparison. Default is `normal=False`.

Additional arguments to customise the appearance of the plot are detailed in Section 5.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples

To plot and display a basic standardised residuals histogram and save it to file as `my_std_residuals_plot.png`:

`my_rsm.std_residuals_plot(filename='my_std_residuals_plot')`

To plot and display a standardised residuals histogram with bin width 1, with standard normal curve:

`self.std_residuals_plot(bin_width=1, normal=True)`

To plot and display a standardised residuals histogram on a subset of items stored as a list in a variable `my_item_list`:

`my_rsm.std_residuals_plot(items=my_item_list)`

6 class MFRM

6.1 Preliminaries

6.1.1 MFRM

Description

Creates an object of the class MFRM from a pandas multiindex dataframe of polytomously scored data of items, rated by multiple raters which share the same maximum score for analysis. No analysis can be run until an object is created.

Usage

```
MFRM(dataframe, max_score=None, extreme_persons=True, no_of_classes=5)
```

Arguments

dataframe	pandas multiindex dataframe with items as columns (item names as column names) and raters and persons as the two levels of the multiindex (rater names as index level 0 names and person names as index level 1 names).
max_score	Integer: The maximum possible score, shared across all items. If no score is passed, max_score will be inferred from the data, although passing an argument is recommended. Default is max_score=None.
extreme_persons	Boolean: if False, all persons with extreme scores (all responses correct or all responses incorrect across all raters) are removed from the response dataframe. Default is extreme_persons=True.
no_of_classes	Integer: the number of classes of persons grouped by ability for overplotting observed responses on theoretical curves. Default is no_of_classes=5

Returns

Object of class MFRM. Analyses are run using methods defined on the MFRM object, with results stored as attributes of the MFRM object.

Several attributes of object MFRM are automatically generated on its creation:

self.dataframe	pandas muliindex dataframe: Dataframe of valid responses.
self.invalid_responses	pandas multiindex dataframe: Dataframe of invalid responses (persons with no responses to any items, i.e. all missing data).
self.max_score	Integer: The maximum possible score, shared across all items.
self.no_of_items	Integer: Number of items.

Returns continue on the next page.

Returns (continued)

`self.items`	List: List of item names.
`self.no_of_persons`	Integer: Number of persons.
`self.persons`	List: List of unique person names.
`self.no_of_raters`	Integer: Number of raters.
`self.raters`	List: List of rater names.
`self.no_of_classes`	Integer: Number of response classes, defined by the argument no_of_classes.

Example

To create an object from a dataframe `my_mfrm_dataframe`, with a maximum score of 5 and 10 observed classes:

```
my_mfrm = MFRM(my_mfrm_dataframe, max_score=5, no_of_classes=10)
```

6.1.2 `rename_item`

Description

Method to rename a single item.

Usage

```
self.rename_item(old, new)
```

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.dataframe` with new name.

Example

To rename an item in object `my_rsm` from `Item_1` to `my_new_item_name`:

```
my_rsm.rename_item('Item_1', 'my_new_item_name')
```

6.1.3 `rename_items_all`

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.dataframe` with new names.

Example

To rename all items in object `my_rsm` with item names in a list stored as a variable `my_new_item_names`:

```
my_rsm.rename_items_all(my_new_item_names)
```

6.1.4 `rename_person`

Description

Method to rename a single person.

Usage

`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns

Replaces specified person name in the second level of the multiindex of `self.dataframe` with new name.

Example

To rename a person in object `my_mfrm` from `Person_1` to `my_new_person_name`:

```
my_mfrm.rename_person('Person_1', 'my_new_person_name')
```

6.1.5 `rename_persons_all`

Description Method to rename all persons.

Usage

`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns

Replaces all person names in the second level of the multiindex of `self.dataframe` with new names.

Example

To rename all persons in object `my_mfrm` with person names in a list stored as a variable `my_new_person_names`:

```
my_mfrm.rename_persons_all(my_new_person_names)
```

6.1.6 `rename_rater`

Description

Method to rename a single rater.

Usage

`self.rename_item(old, new)`

Arguments

`old`	String: the old name for the rater
`new`	String: the new name for the rater

Returns

Replaces specified rater name in the first level of the multiindex of `self.dataframe` with new name.

Example

To rename an item in object `my_mfrm` from `Rater_1` to `my_new_rater_name`:

```
my_mfrm.rename_rater('Item_1', 'my_new_rater_name')
```

6.1.7 `rename_raters_all`

Description

Method to rename all raters.

Usage

```
self.rename_raters_all(new_names)
```

Arguments

`new_names`	List of new rater names as strings

Returns

Replaces all rater names in the first level of the multiindex of `self.dataframe` with new names.

Example

To rename all raters in object `my_mfrm` with item names in a list stored as a variable `my_new_rater_names`:

```
my_mfrm.rename_raters_all(my_new_rater_names)
```

6.2 Core functions

6.2.1 `cat_prob`

Description

Category probability function which calculates the probability $P(X_{nir} = k)$ of scoring k, with $k \in \{0,\ m\}$, where m is the maximum score, rated by rater r, from person ability, central item difficulty, Rasch-Andrich thresholds and rater severity representation. The precise formulation depends on the rater representation, which may be:

- `global`: a single global scalar representation which assumes uniform rater behaviour across items and thresholds.
- `items`: an extended vector representation which applies a different severity for each item but assumes uniform rater behaviour across thresholds.
- `thresholds`: an extended vector representation which applies a different severity for each threshold but assumes uniform rater behaviour across items.
- `matrix`: an extended matrix representation which which applies a different severity for every possible item/threshold combination.

See Elliott and Buttery (2022a) for a detailed discussion of extended rater representations and their uses and implications.

`global`

For a person n with ability β_n attempting an item i with central item difficulty δ_i, Rasch-Andrich thresholds $\{\tau_0, ..., \tau_m\}$ and rater r with severity λ_r, the probability of obtaining a score of k is given by:

$$P(X_{nir} = k) = \frac{e^{k(\beta_n - \delta_i - \lambda_r) - \sum_{t=0}^{k} \tau_t}}{\sum_{k=0}^{m} e^{k(\beta_n - \delta_i - \lambda_r) - \sum_{t=0}^{k} \tau_t}}$$

In this formulation, an item is defined by a central item difficulty, δ_i and a set of centred Rasch-Andrich thresholds $\{\tau_k\}$, $k \in \{0,\ ...,\ m\}$ which sum to zero: an alternative formulation would be to define the item solely by m uncentred thresholds, $\{\tau'_{ik}\}$, $k \in \{1,\ ...,\ m\}$, where $\tau'_{ik} = \delta_i + \tau_k$, in analogy with the partial credit model formulation described in Section **??**, but we will use the centred thresholds formulation throughout here, apart from in item plots where absolute threshold location is salient.

`items`

For a person n with ability β_n attempting an item i with central item difficulty δ_i, Rasch-Andrich thresholds $\{\tau_0, ..., \tau_m\}$ and rater r with severity $\{\lambda_{ri}\}$ for item i, the probability of obtaining a score of k is given by:

$$P(X_{nir} = k) = \frac{e^{k(\beta_n - \delta_i - \lambda_{ri}) - \sum_{t=0}^{k} \tau_t}}{\sum_{k=0}^{m} e^{k(\beta_n - \delta_i - \lambda_{ri}) - \sum_{t=0}^{k} \tau_t}}$$

`thresholds`

For a person n with ability β_n attempting an item i with central item difficulty δ_i, Rasch-Andrich thresholds $\{\tau_0, ..., \tau_m\}$ and rater r with severity vector $\{\lambda_{rj}\}$, $j \in \{0, ..., m\}$, the probability of obtaining a score of k is given by:

$$P(X_{nir} = k) = \frac{e^{k(\beta_n - \delta_i) - \sum_{t=0}^{k}(\tau_t + \lambda_{rt})}}{\sum_{k=0}^{m} e^{k(\beta_n - \delta_i) - \sum_{t=0}^{k}(\tau_t + \lambda_{rt})}}$$

`matrix`

For a person n with ability β_n attempting an item i with central item difficulty δ_i, Rasch-Andrich thresholds $\{\tau_0, ..., \tau_m\}$ and rater r with severity vector $\{\lambda_{rij}\}$, $j \in \{0, ..., m\}$ for item i, the probability of obtaining a score of k is given by:

$$P(X_{nir} = k) = \frac{e^{k(\beta_n - \delta_i) - \sum_{t=0}^{k}(\tau_t + \lambda_{rit})}}{\sum_{k=0}^{m} e^{k(\beta_n - \delta_i) - \sum_{t=0}^{k}(\tau_t + \lambda_{rit})}}$$

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.cat_prob_global(ability, item, difficulties, rater, severities, category, thresholds)

self.cat_prob_items(ability, item, difficulties, rater, severities, category, thresholds)

self.cat_prob_thresholds(ability, item, difficulties, rater, severities, category,
                         thresholds)

self.cat_prob_matrix(ability, item, difficulties, rater, severities, category, thresholds)
```

Arguments

`ability`	Float: Person ability
`item`	Name of item.
`difficulties`	pandas series: Item names as keys and central item difficulties as values.
`rater`	Name of rater.
`severities`	pandas series or dictionary: Rater names as keys and the appropriate rater representations for `global`, `items`, `thresholds` or `matrix` as values. Details of the formats for severities for the different representations are given below.
`category`	Integer: Response category k, with $k \in \{0, 1\}$.
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Rater representations for `severities` argument

Toy examples of each rater representation format, with two raters and two items with three categories:

Rater representation `global`: pandas series with floats (severities) for values.

```
severities = pd.Series({'Rater_1':  -0.5,
                        'Rater_2':   0.5}
```

Rater representation `items`: Dictionary with pandas series for values. The pandas series have item names for keys and floats (item severities) for values.

```
severities = {'Rater_1':  pd.Series({'Item_1':  -1, 'Item_2':  0}),
              'Rater_2':  pd.Series({'Item_1':   0, 'Item_2':  1})}
```

Rater representation `thresholds`: Dictionaries with numpy arrays for values. The numpy arrays contain the severities by threshold, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
severities = {'Rater_1':  np.array([0,  -1, -0.5,    0]),
              'Rater_2':  np.array([0, 1.5,  0.5, -0.5])}
```

Rater representation `matrix`: `matrix`: Nested dictionary. The outer dictionary has item names for values, which are also the keys for the inner dictionary. Values for the inner dictionary are numpy arrays of severities by threshold for the item key, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
severities = {'Rater_1':  {'Item_1':  np.array([0, -1.5, -1, 0.5]),
                           'Item_2':  np.array([0,    1,  0,  -1])},
              'Rater_2':  {'Item_1':  np.array([0, -0.5,  0, 0.5]),
                           'Item_2':  np.array([0,    2,  1,   0])}}
```

Returns

Float: probability of obtaining score k.

Example

To obtain the probability of a person of ability 0.5 scoring 1 on 'Item_1' from central item difficulties set my_item_diffs with a set of Rasch-Andrich thresholds my_thresholds, rated by rater 'Rater_1' from rater severities set my_severities, using the matrix rater representation, and store the result as a variable `my_cat_prob`:

```
my_cat_prob = self.cat_prob_matrix(0.5, 'Item_1', my_item_diffs, 'Rater_1', my_severities, 1,
                                   my_thresholds)
```

The same format is followed for other rater representations with no changes apart from the method name and the format of the `severities` argument.

6.2.2 exp_score

Description

Expected score function which calculates the expected score $E(X_{nir})$, rated by rater r from person ability, central item difficulty, Rasch-Andrich thresholds and rater severity representation. The expected score is given by:

$$E(X_{nir}) = \sum_{k=0}^{1} kP(X_{nir} = k)$$

where $P(X_{nir} = k)$ is the relevant category probability equation for the rater representation, described in Section 6.2.1.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.exp_score_global(ability, item, difficulties, rater, severities, thresholds)
```

```
self.exp_score_items(ability, item, difficulties, rater, severities, thresholds)
```

```
self.exp_score_thresholds(ability, item, difficulties, rater, severities, thresholds)
```

```
self.exp_score_matrix(ability, item, difficulties, rater, severities, thresholds)
```

Arguments

`ability`	Float: Person ability
`item`	Name of item.
`difficulties`	pandas series: Item names as keys and central item difficulties as values.
`rater`	Name of rater.
`severities`	pandas series or dictionary: Rater names as keys and the appropriate rater representations for `global`, `items`, `thresholds` or `matrix` as values. Details of the formats for severities for the different representations are given below.
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Rater representations for `severities` argument

Toy examples of each rater representation format, with two raters and two items with three categories:

Rater representation `global`: pandas series with floats (severities) for values.

```
severities = pd.Series({'Rater_1':  -0.5,
                        'Rater_2':   0.5}
```

Rater representation `items`: Dictionary with pandas series for values. The pandas series have item names for keys and floats (item severities) for values.

```
severities = {'Rater_1':  pd.Series({'Item_1':  -1, 'Item_2':  0}),
              'Rater_2':  pd.Series({'Item_1':   0, 'Item_2':  1})}
```

Rater representation `thresholds`: Dictionaries with numpy arrays for values. The numpy arrays contain the severities by threshold, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
severities = {'Rater_1':  np.array([0,  -1, -0.5,    0]),
              'Rater_2':  np.array([0, 1.5,  0.5, -0.5])}
```

Rater representation `matrix`: `matrix`: Nested dictionary. The outer dictionary has item names for values, which are also the keys for the inner dictionary. Values for the inner dictionary are numpy arrays of severities by threshold for the item key, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
severities = {'Rater_1':  {'Item_1':  np.array([0, -1.5, -1, 0.5]),
                           'Item_2':  np.array([0,    1,  0,  -1])},
              'Rater_2':  {'Item_1':  np.array([0, -0.5,  0, 0.5]),
                           'Item_2':  np.array([0,    2,  1,   0])}}
```

Returns

Float: expected score.

Example

To obtain the expected score for a person of ability 0.5 on 'Item_1' from central item difficulties set my_item_diffs with a set of Rasch-Andrich thresholds my_thresholds, rated by rater 'Rater_1' from rater severities set my_severities, using the matrix rater representation, and store the result as a variable `my_exp_score`:

```
my_exp_score = self.exp_score_matrix(0.5, 'Item_1', my_item_diffs, 'Rater_1', my_severities,
                                     my_thresholds)
```

6.2.3 `variance`

Description

Variance function which calculates the variance of the score $V(X_{ni})$ from person ability, item difficulty and Rasch-Andrich thresholds. The variance is given by:

$$V(X_{nir}) = \sum_{k=0}^{1} P(X_{nir} = k)(k - E(X_{nir}))^2$$

where $P(X_{nir} = k)$ and $E(X_{nir})$ are as described in Sections 6.2.1 and 6.2.2 respectively.

The variance is also both the Fisher information for the response and the first partial differential of the expected score function with respect to person ability.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.variance_global(ability, item, difficulties, rater, severities, thresholds)

self.variance_items(ability, item, difficulties, rater, severities, thresholds)

self.variance_thresholds(ability, item, difficulties, rater, severities, thresholds)

self.variance_matrix(ability, item, difficulties, rater, severities, thresholds)
```

Arguments

`ability`	Float: Person ability
`item`	Name of item.
`difficulties`	pandas series: Item names as keys and central item difficulties as values.
`rater`	Name of rater.
`severities`	pandas series or dictionary: Rater names as keys and the appropriate rater representations for `global`, `items`, `thresholds` or `matrix` as values. Details of the formats for severities for the different representations are given below.
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Rater representations for `severities` argument

Toy examples of each rater representation format, with two raters and two items with three categories:

Rater representation `global`: pandas series with floats (severities) for values.

```
severities = pd.Series({'Rater_1':  -0.5,
                        'Rater_2':   0.5}
```

Rater representation `items`: Dictionary with pandas series for values. The pandas series have item names for keys and floats (item severities) for values.

```
severities = {'Rater_1':  pd.Series({'Item_1':  -1, 'Item_2':  0}),
              'Rater_2':  pd.Series({'Item_1':   0, 'Item_2':  1})}
```

Rater representation `thresholds`: Dictionaries with numpy arrays for values. The numpy arrays contain the severities by threshold, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
severities = {'Rater_1':  np.array([0,  -1, -0.5,    0]),
              'Rater_2':  np.array([0, 1.5,  0.5, -0.5])}
```

Rater representation `matrix`: `matrix`: Nested dictionary. The outer dictionary has item names for values, which are also the keys for the inner dictionary. Values for the inner dictionary are numpy arrays of severities by threshold for the item key, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
severities = {'Rater_1':  {'Item_1':  np.array([0, -1.5, -1, 0.5]),
                           'Item_2':  np.array([0,    1,  0,  -1])},
              'Rater_2':  {'Item_1':  np.array([0, -0.5,  0, 0.5]),
                           'Item_2':  np.array([0,    2,  1,   0])}}
```

Returns
Float: expected score.

Example
To obtain the variance for a person of ability 0.5 on 'Item_1' from central item difficulties set my_item_diffs with a set of Rasch-Andrich thresholds my_thresholds, rated by rater 'Rater_1' from rater severities set my_severities, using the matrix rater representation, and store the result as a variable `my_variance`:

```
my_variance = self.variance_matrix(0.5, 'Item_1', my_item_diffs, 'Rater_1', my_severities,
                                   my_thresholds)
```

6.2.4 `kurtosis`

Description
Kurtosis function which calculates the kurtosis of the score $\kappa(X_{ni})$ from person ability, central item difficulty and Rasch-Andrich thresholds. The variance is given by:

$$\kappa(X_{nir}) = \sum_{k=0}^{1} P(X_{nir} = k)(k - E(X_{nir}))^4$$

where $P(X_{nir} = k)$ and $E(X_{nir})$ are as described in Sections 6.2.1 and 6.2.2 respectively.

Usage
Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.kurtosis_global(ability, item, difficulties, rater, severities, thresholds)

self.kurtosis_items(ability, item, difficulties, rater, severities, thresholds)

self.kurtosis_thresholds(ability, item, difficulties, rater, severities, thresholds)

self.kurtosis_matrix(ability, item, difficulties, rater, severities, thresholds)
```

Arguments

`ability`	Float: Person ability
`item`	Name of item.
`difficulties`	pandas series: Item names as keys and central item difficulties as values.
`rater`	Name of rater.
`severities`	pandas series or dictionary: Rater names as keys and the appropriate rater representations for `global`, `items`, `thresholds` or `matrix` as values. Details of the formats for severities for the different representations are given below.
`thresholds`	List or numpy array: Set of $m + 1$ Rasch-Andrich thresholds, where m is the maximum score, which sum to zero and the first of which is zero.

Rater representations for `severities` argument

Toy examples of each rater representation format, with two raters and two items with three categories:

Rater representation `global`: pandas series with floats (severities) for values.

```
severities = pd.Series({'Rater_1':  -0.5,
                        'Rater_2':   0.5}
```

Rater representation `items`: Dictionary with pandas series for values. The pandas series have item names for keys and floats (item severities) for values.

```
severities = {'Rater_1':  pd.Series({'Item_1':  -1, 'Item_2':  0}),
              'Rater_2':  pd.Series({'Item_1':   0, 'Item_2':  1})}
```

Rater representation `thresholds`: Dictionaries with numpy arrays for values. The numpy arrays contain the severities by threshold, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
severities = {'Rater_1':  np.array([0,  -1, -0.5,    0]),
              'Rater_2':  np.array([0, 1.5,  0.5, -0.5])}
```

Rater representation `matrix`: `matrix`: Nested dictionary. The outer dictionary has item names for values, which are also the keys for the inner dictionary. Values for the inner dictionary are numpy arrays of severities by threshold for the item key, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
severities = {'Rater_1':  {'Item_1':  np.array([0, -1.5, -1, 0.5]),
                           'Item_2':  np.array([0,    1,  0,  -1])},
              'Rater_2':  {'Item_1':  np.array([0, -0.5,  0, 0.5]),
                           'Item_2':  np.array([0,    2,  1,   0])}}
```

Returns

Float: expected score.

Example

To obtain the kurtosis for a person of ability 0.5 on 'Item_1' from central item difficulties set my_item_diffs with a set of Rasch-Andrich thresholds my_thresholds, rated by rater 'Rater_1' from rater severities set my_severities, using the matrix rater representation, and store the result as a variable `my_kurtosis`:

```
my_kurtosis = self.kurtosis_matrix(0.5, 'Item_1', my_item_diffs, 'Rater_1', my_severities,
                                   my_thresholds)
```

6.3 Parameter estimation

6.3.1 `calibrate`

Description

Produces central item difficulty, Rasch-Andrich threshold and rater severity estimates using the conditional pairwise estimation (CPAT) algorithm (Elliott & Buttery, 2022b).

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.calibrate_global(constant=0.1, method='cos', matrix_power=3, log_lik_tol=0.000001)

self.calibrate_items(constant=0.1, method='cos', matrix_power=3, log_lik_tol=0.000001)

self.calibrate_thresholds(constant=0.1, method='cos', matrix_power=3, log_lik_tol=0.000001)

self.calibrate_matrix(constant=0.1, method='cos', matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: method for derivation of central item difficulty and rater severity estimates from pairwise reciprocal matrices (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) for central item difficulty estimates is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for central item difficulty estimates. Ignored for other methods.

Returns

Returns four attributes:

`self.diffs`	pandas series: Item difficulty estimates with the item names as keys and estimates as values.

Returns continue on the next page.

Returns (continued)

`self.thresholds`	numpy array: Rasch-Andrich threshold estimates, an array of $m+1$ estimates, where m is the maximum score, which sum to zero and the first of which is zero.
`self.severities`	pandas series or dictionary: Rater names as keys and the appropriate rater representations for `global`, `items`, `thresholds` or `matrix` as values. Details of the formats for severities for the different representations are given below.
`self.cat_widths`	numpy array: Array of $m-1$ category width estimates, where m is the maximum score.

Example `self.severities` outputs

Toy examples of each rater representation format, with two raters and two items with three categories:

Rater representation `global`: pandas series with floats (severities) for values.

```
self.severities_global = pd.Series({'Rater_1':  -0.5,
                                    'Rater_2':   0.5}
```

Rater representation `items`: Nested dictionary. The outer dictionary has item names for values, which are also the keys for the inner dictionary. The inner dictionaries item names for keys and floats (item severities) for values.

```
self.severities_items = {'Rater_1':  {'Item_1':  -1, 'Item_2':  0},
                         'Rater_2':  'Item_1':   0, 'Item_2':  1}}
```

Rater representation `thresholds`: Dictionaries with numpy arrays for values. The numpy arrays contain the severities by threshold, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
self.severities_thresholds = {'Rater_1':  np.array([0,  -1, -0.5,    0]),
                              'Rater_2':  np.array([0, 1.5,  0.5, -0.5])}
```

Rater representation `matrix`: `matrix`: Nested dictionary. The outer dictionary has item names for values, which are also the keys for the inner dictionary. Values for the inner dictionary are numpy arrays of severities by threshold for the item key, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
self.severities_matrix = {'Rater_1':  {'Item_1':  np.array([0, -1.5, -1, 0.5]),
                                       'Item_2':  np.array([0,    1,  0,  -1])},
                          'Rater_2':  {’Item_1’:  np.array([0, -0.5,  0, 0.5]),
                                       'Item_2':  np.array([0,    2,  1,   0])}}
```

Examples

To generate a set of estimates for the global rater representation using the cosine similarity method for central item difficulties and rater severities, with additive smoothing constant of 0.1:

```
self.calibrate_global()
```

To generate a set of estimates for the matrix rater representation using the log-likelihood method for central item difficulties and rater severities, with matrix raised to power 7 and a convergence stopping criterion of 0.00000001:

```
self.calibrate_matrix(method='log-lik', matrix_power=7, log_lik_tol=0.00000001)
```

6.3.2 `calibrate_anchor`

Description

Anchors a calibration to a defined set of 'gold standard' anchor raters, who provide a frame of reference and for whom mean severity is defined as zero; other raters' severities shifted accordingly. For extended rater representations, this process has a knock-on effect on the central item difficulty and/or Rasch-Andrich threshold estimates. Full details of the anchoring procedure can be found in Elliott and Buttery (2022a).

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.calibrate_global_anchor(anchor_raters, calibrate=False, constant=0.1, method='cos',
                             matrix_power=3, log_lik_tol=0.000001)

self.calibrate_items_anchor(anchor_raters, calibrate=False, constant=0.1, method='cos',
                            matrix_power=3, log_lik_tol=0.000001)

self.calibrate_thresholds_anchor(anchor_raters, calibrate=False, constant=0.1,
                                 method='cos', matrix_power=3, log_lik_tol=0.000001)

self.calibrate_matrix_anchor(anchor_raters, calibrate=False, constant=0.1, method='cos',
                             matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`anchor_raters`	List: List of names of 'gold standard' anchor raters.
`calibrate`	Boolean: Only needed when a standard unanchored calibration has not been run, so unanchored parameters have not yet been generated. If `True`, a standard unanchored calibration will first be run. Default value is `calibrate=False`.
`constant`	Float: Only relevant when `calibrate=True`. Additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: Only relevant when `calibrate=True`. Method for derivation of central item difficulty and rater severity estimates from pairwise reciprocal matrices (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`

Arguments continue on the next page.

Arguments (continued)

`matrix_power`	Integer: Only relevant when `calibrate=True`. Power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) for central item difficulty estimates is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: Only relevant when `calibrate=True`. Convergence stopping criterion for log-likelihood method for central item difficulty estimates. Ignored for other methods.
`self.anchor_diffs`	pandas series: Item difficulty estimates with the item names as keys and estimates as values. Depending on the rater representation, the attribute will be stored as: `self.anchor_diffs_global`, `self.anchor_diffs_items`, `self.anchor_diffs_thresholds` or `self.anchor_diffs_matrix`.
`self.anchor_thresholds`	numpy array: Rasch-Andrich threshold estimates, an array of $m+1$ estimates, where m is the maximum score, which sum to zero and the first of which is zero. Depending on the rater representation, the attribute will be stored as: `self.anchor_thresholds_global`, `self.anchor_thresholds_items`, `self.anchor_thresholds_thresholds` or `self.anchor_thresholds_matrix`.
`self.anchor_severities`	pandas series or dictionary: Rater names as keys and the appropriate rater representations for `global`, `items`, `thresholds` or `matrix` as values. Details of the formats for severities for the different representations are given below. Depending on the rater representation, the attribute will be stored as: `self.anchor_severities_global`, `self.anchor_severities_items`, `self.anchor_severities_thresholds` or `self.anchor_severities_matrix`.
`self.anchor_cat_widths`	numpy array Array of $m-1$ category width estimates, where m is the maximum score, as values. Depending on the rater representation, the attribute will be stored as: `self.anchor_cat_widths_global`, `self.anchor_cat_widths_items`, `self.anchor_cat_widths_thresholds` or `self.anchor_cat_widths_matrix`.

Example `self.severities` **outputs**
Toy examples of each rater representation format, with two raters and two items with three categories:

Rater representation `global`: pandas series with floats (severities) for values.

```
self.severities_global = pd.Series({'Rater_1':  -0.5,
                                    'Rater_2':   0.5}
```

Rater representation `items`: Dictionary with pandas series for values. The pandas series have item names for keys and floats (item severities) for values.

```
self.severities_items = {'Rater_1':  pd.Series({'Item_1':  -1, 'Item_2':  0}),
                         'Rater_2':  pd.Series({'Item_1':   0, 'Item_2':  1})}
```

Rater representation `thresholds`: Dictionaries with numpy arrays for values. The numpy arrays contain the severities by threshold, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
self.severities_thresholds = {'Rater_1':  np.array([0,  -1, -0.5,    0]),
                              'Rater_2':  np.array([0, 1.5,  0.5, -0.5])}
```

Rater representation `matrix`: `matrix`: Nested dictionary. The outer dictionary has item names for values, which are also the keys for the inner dictionary. Values for the inner dictionary are numpy arrays of severities by threshold for the item key, with the integer of the array index corresponding to each threshold (0 to `self.max_score`).

```
self.severities_matrix = {'Rater_1':  {'Item_1':  np.array([0, -1.5, -1, 0.5]),
                                       'Item_2':  np.array([0,    1,  0,  -1])},
                          'Rater_2':  {’Item_1’:  np.array([0, -0.5,  0, 0.5]),
                                       'Item_2':  np.array([0,    2,  1,   0])}}
```

Examples
To generate a set of anchored estimates for the global rater representation from an existing unanchored calibration, anchored to raters 'Rater_1' and `'Rater_1'`:

```
self.calibrate_global_anchor(['Rater_1', 'Rater_2'])
```

To generate a set of estimates for the matrix rater representation anchored to raters 'Rater_1' and `'Rater_1'`, first running and unanchored calibration, using the log-likelihood method for central item difficulties and rater severities, with matrix raised to power 7 and a convergence stopping criterion of 0.00000001:

```
self.calibrate_matrix(['Rater_1', 'Rater_2'], calibrate=True, method='log-lik',
               method='log-lik', matrix_power=7, log_lik_tol=0.00000001)
```

6.3.3 `std_errors`

Description
Produces bootstrapped estimates for the standard errors of central item difficulty estimates, threshold estimates and category width estimates for bounded (non-extreme) categories.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.std_errors_global(anchor_raters=None, interval=None, no_of_samples=100, constant=0.1,
                       method='cos', matrix_power=3, log_lik_tol=0.000001)
```

```
self.std_errors_items(anchor_raters=None, interval=None, no_of_samples=100, constant=0.1,
                      method='cos', matrix_power=3, log_lik_tol=0.000001)
```

```
self.std_errors_thresholds(anchor_raters=None, interval=None, no_of_samples=100,
                           constant=0.1, method='cos', matrix_power=3, log_lik_tol=0.000001)
```

```
self.std_errors_matrix(anchor_raters=None, interval=None, no_of_samples=100, constant=0.1,
                       method='cos', matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`anchor_raters`	List: List of names of 'gold standard' anchor raters. Only used if anchored standard errors are being produced. Default is `anchor_raters=None`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval. Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.
`no_of_samples`	Integer: Number of bootstrap samples to generate. More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of central item difficulty estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix for central item difficulty estimates (Elliott & Buttery, 2022b:991) is raised. Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for central item difficulty estimates. Ignored for other methods.

Returns
Creates a set of attributes, which depend on whether an argument is passed to `anchor_raters`.

If `anchor_raters=None`:

`self.item_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each item estimate: `self.item_bootstrap_global`, `self.item_bootstrap_items`, `self.item_bootstrap_thresholds` or `self.item_bootstrap_matrix`,
`self.item_se`	pandas series: Item names as keys and item standard errors as values.: `self.item_se_global`, `self.item_se_items`, `self.item_se_thresholds` or `self.item_se_matrix`,
`self.threshold_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each threshold estimate: `self.threshold_bootstrap_global`, `self.threshold_bootstrap_items`, `self.threshold_bootstrap_thresholds` or `self.threshold_bootstrap_matrix`,
`self.threshold_se`	pandas series: Threshold numbers as keys and item standard errors as values: `self.threshold_se_global`, `self.threshold_se_items`, `self.threshold_se_thresholds` or `self.threshold_se_matrix`,
`self.cat_width_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each category width estimate: `self.cat_width_bootstrap_global`, `self.cat_width_bootstrap_items`, `self.cat_width_bootstrap_thresholds` or `self.cat_width_bootstrap_matrix`,

Returns continue on the next page.

Returns (continued)

`self.cat_width_se`	pandas series: Category numbers as keys and item standard errors as values: `self.cat_width_se_global`, `self.cat_width_se_items`, `self.cat_width_se_thresholds` or `self.cat_width_se_matrix`,
`self.rater_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each item estimate: `self.rater_bootstrap_global`, `self.rater_bootstrap_items`, `self.rater_bootstrap_thresholds` or `self.rater_bootstrap_matrix`,
`self.rater_se`	pandas series: Item names as keys and item standard errors as values: `self.rater_se_global`, `self.rater_se_items`, `self.rater_se_thresholds` or `self.rater_se_matrix`,

If an argument is passed to `interval`, also returns:

`self.item_low`	Lower bound of the specified interval for item estimates: `self.item_low_global`, `self.item_low_items`, `self.item_low_thresholds` or `self.item_low_matrix`,
`self.item_high`	Upper bound of the specified interval for item estimates: `self.item_high_global`, `self.item_high_items`, `self.item_high_thresholds` or `self.item_high_matrix`,
`self.threshold_low`	Lower bound of the specified interval for threshold estimates: `self.threshold_low_global`, `self.threshold_low_items`, `self.threshold_low_thresholds` or `self.threshold_low_matrix`,

Returns continue on the next page.

Returns (continued)

self.threshold_high	Upper bound of the specified interval for threshold estimates: self.threshold_high_global, self.threshold_high_items, self.threshold_high_thresholds or self.threshold_high_matrix,
self.cat_width_low	Lower bound of the specified interval for category estimates: self.cat_width_low_global, self.cat_width_low_items, self.cat_width_low_thresholds or self.cat_width_low_matrix,
self.cat_width_high	Upper bound of the specified interval for category estimates: self.cat_width_high_global, self.cat_width_high_items, self.cat_width_high_thresholds or self.cat_width_high_matrix,
self.rater_low	Lower bound of the specified interval for item estimates: self.self.rater_low_global, self.self.rater_low_items, self.self.rater_low_thresholds or self.self.rater_low_matrix,
self.rater_high	Upper bound of the specified interval for item estimates: self.rater_high_global, self.rater_high_items, self.rater_high_thresholds or self.rater_high_matrix,

If anchor_raters is not None:

self.anchor_item_bootstrap	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each item estimate: self.anchor_item_bootstrap_global, self.anchor_item_bootstrap_items, self.anchor_item_bootstrap_thresholds or self.anchor_item_bootstrap_matrix,

Returns continue on the next page.

Returns (continued)

`self.anchor_item_se`	pandas series: Item names as keys and item standard errors as values.: `self.anchor_item_se_global`, `self.anchor_item_se_items`, `self.anchor_item_se_thresholds` or `self.anchor_item_se_matrix`,
`self.anchor_threshold_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each threshold estimate: `self.anchor_threshold_bootstrap_global`, `self.anchor_threshold_bootstrap_items`, `self.anchor_threshold_bootstrap_thresholds` or `self.anchor_threshold_bootstrap_matrix`,
`self.anchor_threshold_se`	pandas series: Threshold numbers as keys and item standard errors as values: `self.anchor_threshold_se_global`, `self.anchor_threshold_se_items`, `self.anchor_threshold_se_thresholds` or `self.anchor_threshold_se_matrix`,
`self.anchor_cat_width_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each category width estimate: `self.anchor_cat_width_bootstrap_global`, `self.anchor_cat_width_bootstrap_items`, `self.anchor_cat_width_bootstrap_thresholds` or `self.anchor_cat_width_bootstrap_matrix`,
`self.cat_width_se`	pandas series: Category numbers as keys and item standard errors as values: `self.anchor_cat_width_se_global`, `self.anchor_cat_width_se_items`, `self.anchor_cat_width_se_thresholds` or `self.anchor_cat_width_se_matrix`,

Returns continue on the next page.

Returns (continued)

`self.anchor_rater_bootstrap`	pandas dataframe: Full bootstrap results, with a row for each bootstrap sample and a column for each item estimate: `self.anchor_rater_bootstrap_global`, `self.anchor_rater_bootstrap_items`, `self.anchor_rater_bootstrap_thresholds` or `self.anchor_rater_bootstrap_matrix`,
`self.anchor_rater_se`	pandas series: Item names as keys and item standard errors as values: `self.anchor_rater_se_global`, `self.anchor_rater_se_items`, `self.anchor_rater_se_thresholds` or `self.anchor_rater_se_matrix`,

If an argument is passed to `interval`, also returns:

`self.anchor_item_low`	Lower bound of the specified interval for item estimates: `self.anchor_item_low_global`, `self.anchor_item_low_items`, `self.anchor_item_low_thresholds` or `self.anchor_item_low_matrix`,
`self.anchor_item_high`	Upper bound of the specified interval for item estimates: `self.anchor_item_high_global`, `self.anchor_item_high_items`, `self.anchor_item_high_thresholds` or `self.anchor_item_high_matrix`,
`self.anchor_threshold_low`	Lower bound of the specified interval for threshold estimates: `self.anchor_threshold_low_global`, `self.anchor_threshold_low_items`, `self.anchor_threshold_low_thresholds` or `self.anchor_threshold_low_matrix`,
`self.anchor_threshold_high`	Upper bound of the specified interval for threshold estimates: `self.anchor_threshold_high_global`, `self.anchor_threshold_high_items`, `self.anchor_threshold_high_thresholds` or `self.anchor_threshold_high_matrix`,

Returns continue on the next page.

Returns (continued)

`self.anchor_cat_width_low`	Lower bound of the specified interval for category estimates: `self.anchor_cat_width_low_global`, `self.anchor_cat_width_low_items`, `self.anchor_cat_width_low_thresholds` or `self.anchor_cat_width_low_matrix`,
`self.anchor_cat_width_high`	Upper bound of the specified interval for category estimates: `self.anchor_cat_width_high_global`, `self.anchor_cat_width_high_items`, `self.anchor_cat_width_high_thresholds` or `self.anchor_cat_width_high_matrix`,
`self.anchor_rater_low`	Lower bound of the specified interval for item estimates: `self.anchor_self.rater_low_global`, `self.anchor_self.rater_low_items`, `self.anchor_self.rater_low_thresholds` or `self.anchor_self.rater_low_matrix`,
`self.anchor_rater_high`	Upper bound of the specified interval for item estimates: `self.anchor_rater_high_global`, `self.anchor_rater_high_items`, `self.anchor_rater_high_thresholds` or `self.anchor_rater_high_matrix`,

Examples

To generate unanchored item standard errors for the global representation with a 95% interval from 200 samples:

```
self.std_errors_global(interval=0.95, no_of_samples=200)
```

To generate anchored item standard errors for the matrix representation with a 90% interval from 200 samples, anchored to raters `'Rater_1'` and `'Rater_2'`:

```
self.anchor_std_errors_matrix(anchor_raters=['Rater_1', 'Rater_2'],
                              no_of_samples=200)
```

Modifications to the estimation method are discussed in Section 5.3.1.

6.3.4 `abil`

Description

Generates an ability estimate for a person using the Newton-Raphson method to produce a maximum likelihood estimate, with optional Warm bias correction (Warm, 1989).

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.abil_global(person, anchor=False, items=None, raters=None, warm_corr=True,
                 tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)

self.abil_items(person, anchor=False, items=None, raters=None, warm_corr=True,
                tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)

self.abil_thresholds(person, anchor=False, items=None, raters=None, warm_corr=True,
                     tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)

self.abil_matrix(person, anchor=False, items=None, raters=None, warm_corr=True,
                 tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

Arguments

`person`	String: The person name for the ability being estimated.
`anchor`	Boolean:If `True`, parameters from an anchored calibration will be used. Default is `anchor=False`.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`raters`	List: List of names of a subset of rater, based on which to generate the ability estimate. Default is `raters=None`, which generates an ability based on the full set of raters. Only use when an estimate based on a subset of raters or an individual rater is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Float: person ability estimate.

Example
To generate an unanchored person ability estimate for `Person_1` using a global rater representation using the default settings and store the result as a variable, `my_person_ability`:

```
my_person_ability = my_person_ability = self.abil_global('Person_1')
```

To generate an anchored MLE person ability estimate under the matrix rater representation without Warm bias correction for `Person_1` based on the first three items rated by `Rater_1` and store the result as a variable, `my_person_ability`:

```
my_person_ability = self.abil_matrix('Person_1', items=['Item_1', 'Item_2', 'Item_3'],
                                     raters=['Rater_1'], warm_corr=False)
```

6.3.5 `person_abils`

Description
Generates ability estimates for all persons using the Newton-Raphson method to produce maximum likelihood estimates, with optional Warm bias correction (Warm, 1989).

Usage
Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.person_abil_global(anchor=False, items=None, raters=None, warm_corr=True,
                        tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

```
self.person_abil_items(anchor=False, items=None, raters=None, warm_corr=True,
                       tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

```
self.person_abil_thresholds(anchor=False, items=None, raters=None, warm_corr=True,
                            tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

```
self.person_abil_matrix(anchor=False, items=None, raters=None, warm_corr=True,
                        tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

Arguments

`anchor`	Boolean:If `True`, parameters from an anchored calibration will be used. Default is `anchor=False`.

Arguments continue on the next page.

Arguments (continued)

`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`raters`	List: List of names of a subset of rater, based on which to generate the ability estimate. Default is `raters=None`, which generates an ability based on the full set of raters. Only use when an estimate based on a subset of raters or an individual rater is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns

- If `anchor=False`, attribute `self.abils_global`, `self.abils_items`, `self.abils_thresholds` or `self.abils_matrix`: pandas series with person names as keys and ability estimates as values.
- If `anchor=True`, attribute `self.anchor_abils_global`, `self.anchor_abils_items`, `self.anchor_abils_thresholds` or `self.anchor_abils_matrix`: pandas series with person names as keys and ability estimates as values.

Examples

To generate a set of unanchored person ability estimates for the global rater representation from all items and raters with Warm bias correction:

```
self.person_abils_global()
```

To generate a set of anchored person ability estimates for the matrix rater representation without Warm bias correction, on a subset of the first three items only, rated by `Rater_1`:

```
self.person_abils_matrix(anchor=True, items=['Item_1', 'Item_2', 'Item_3'], raters=['Rater_1'],
                         warm_corr=False)
```

6.3.6 `score_abil`

Description

Generates an ability estimate for a given raw score on responses to a given set of items using the Newton-Raphson method to produce a maximum likelihood estimate, with optional Warm bias correction (Warm, 1989).

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.score_abil_global(score, anchor=False, items=None, raters=None, warm_corr=True,)
                       tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

```
self.score_abil_items(score, anchor=False, items=None, raters=None, warm_corr=True,)
                      tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

```
self.score_abil_thresholds(score, anchor=False, items=None, raters=None, warm_corr=True,)
                           tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

```
self.score_abil_matrix(score, anchor=False, items=None, raters=None, warm_corr=True,)
                       tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

Arguments

`score`	Integer: The raw score for which ability is being estimated.
`anchor`	Boolean:If `True`, parameters from an anchored calibration will be used. Default is `anchor=False`.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`raters`	List: List of names of a subset of rater, based on which to generate the ability estimate. Default is `raters=None`, which generates an ability based on the full set of raters. Only use when an estimate based on a subset of raters or an individual rater is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.

Arguments continue on the next page.

Arguments (continued)

`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Float: Ability estimate corresponding to raw score.

Examples
To generate an unanchored ability estimate under the global rater representation for a score of 10 on all items rated by all raters, with Warm bias correction, and store the result as a variable, `my_score_ability`:

```
my_score_ability = self.score_abil_global(10)
```

To generate an anchored ability estimate under the matrix rater representation for a score of 10 on a subset of items saved as a variable `my_items`, rated by `Rater_1` without Warm bias correction, and store the result as a variable, `my_score_ability`:

```
my_score_ability = self.score_abil_matrix(10, anchor=True, items=my_items,
                                          raters=['Rater_1'], warm_corr=False)
```

6.3.7 `abil_lookup_table`

Description
Generates a lookup table of ability estimates corresponding to all available raw scores on a set of items with no missing responses, using the Newton-Raphson method to produce maximum likelihood estimates and with optional Warm bias correction (Warm, 1989).

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.abil_lookup_table_global(anchor=False, items=None, raters=None, warm_corr=True,
                              tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)

self.abil_lookup_table_items(anchor=False, items=None, raters=None, warm_corr=True,
                             tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)

self.abil_lookup_table_thresholds(anchor=False, items=None, raters=None, warm_corr=True,
                                  tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)

self.abil_lookup_table_matrix(anchor=False, items=None, raters=None, warm_corr=True,
                              tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5)
```

Arguments

`anchor`	Boolean:If `True`, parameters from an anchored calibration will be used. Default is `anchor=False`.
`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`raters`	List: List of names of a subset of rater, based on which to generate the ability estimate. Default is `raters=None`, which generates an ability based on the full set of raters. Only use when an estimate based on a subset of raters or an individual rater is required.
`ext_scores`	Boolean: If `True`, ability estimates for extreme scores (all correct/all incorrect) will be generated using the `ext_score_adjustment` argument. Default is `ext_scores=True`.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns
Attribute `self.abil_table`: pandas series with raw scores as keys and corresponding ability estimates as values.

Examples
To generate an unanchored ability lookup table under the global rater representation for all items rated by all raters, including extreme scores, with Warm bias correction:

```
self.abil_lookup_table()
```

To generate an anchored ability lookup table under the matrix rater representation for a subset of items saved as a variable `my_items`, rated by `Rater_1`,without extreme scores and without Warm bias correction:

```
self.abil_lookup_table(anchor=True, items=my_items, raters=['Rater_1'], ext_scores=False)
```

6.3.8 `csem`

Description
Calculates conditional standard error of measurement for a person ability estimate from a set of items and raters.

Usage
Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.csem_global(person, anchor=False, items=None, raters=None, warm_corr=True,
                 tolerance=0.0000001, max_iters=100, ext_score_adjustment=0.5)

self.csem_items(person, anchor=False, items=None, raters=None, warm_corr=True,
                tolerance=0.0000001, max_iters=100, ext_score_adjustment=0.5)

self.csem_thresholds(person, anchor=False, items=None, raters=None, warm_corr=True,
                     tolerance=0.0000001, max_iters=100, ext_score_adjustment=0.5)

self.csem_matrix(person, anchor=False, items=None, raters=None, warm_corr=True,
                 tolerance=0.0000001, max_iters=100, ext_score_adjustment=0.5)
```

Arguments

`person`	Person name.
`anchor`	Boolean:If `True`, parameters from an anchored calibration will be used. Default is `anchor=False`.

Arguments continue on the next page.

Arguments (continued)

`items`	List: List of names of a subset of items, based on which to generate the ability estimate. Default is `items=None`, which generates an ability based on the full set of items. Only use when an estimate based on a subset of items is required.
`raters`	List: List of names of a subset of rater, based on which to generate the ability estimate. Default is `raters=None`, which generates an ability based on the full set of raters. Only use when an estimate based on a subset of raters or an individual rater is required.
`warm_corr`	Boolean: if `True`, Warm's bias correction (Warm, 1989) is applied to the estimate. Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations. Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations. Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned if the score is extreme (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Returns

Float: conditional standard error of measurement for ability estimate.

Examples

To generate the unanchored CSEM for `Person_1` under the global rater representation on all items and raters and save the result as a variable, `my_csem`:

```
my_csem = self.csem_global('Person_1')
```

To generate the anchored CSEM for `Person_1` under the matrix rater representation on a subset of items `['Item_1', 'Item_2']`, and a subset of raters `['Rater_1', 'Rater_2']`, and save the result as a variable, `my_csem`:

```
my_csem = self.csem_matrix(Person_1, anchor=True, items=['Item_1', 'Item_2'],
                           raters=['Rater_1', 'Rater_2'])
```

6.4 Statistical output

6.4.1 `item_stats_df`

Description

Produces a table of item statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.item_stats_df_global(anchor_raters=None, full=False, zstd=False,
                          point_measure_corr=False, dp=3, warm_corr=True, tolerance=1e-07,
                          max_iters=100, ext_score_adjustment=0.5, method='cos',
                          constant=0.1, no_of_samples=100, interval=None)
```

```
self.item_stats_df_items(anchor_raters=None, full=False, zstd=False,
                         point_measure_corr=False, dp=3, warm_corr=True, tolerance=1e-07,
                         max_iters=100, ext_score_adjustment=0.5, method='cos',
                         constant=0.1, no_of_samples=100, interval=None)
```

```
self.item_stats_df_thresholds(anchor_raters=None, full=False, zstd=False,
                              point_measure_corr=False, dp=3, warm_corr=True,
                              tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5,
                              method='cos', constant=0.1, no_of_samples=100, interval=None)
```

```
self.item_stats_df_matrix(anchor_raters=None, full=False, zstd=False,
                          point_measure_corr=False, dp=3, warm_corr=True, tolerance=1e-07,
                          max_iters=100, ext_score_adjustment=0.5, method='cos',
                          constant=0.1, no_of_samples=100, interval=None)
```

Arguments

`anchor_raters`	List: If `None`, unanchored central item difficulty estimates and standard errors will be used. If a list of rater names is passed, anchored estimates and standard errors, anchored to the raters passed, will be used. Other statistics are unaffected by `anchor_raters`. Default is `anchor_raters=None`.
`full`	Boolean: If `True`, a full table with all available statistics is produced. Default is `full=False` for a summary table with the most commonly reported statistics.
`zstd`	Boolean: If `True`, infit and outfit standardised z-scores are reported alongside mean square statistics. Default is `zstd=False`. Automatically `True` if `full=True`.

Arguments continue on the next page.

Arguments (continued)

`point_measure_corr`	Boolean: If `True`, point-polyserial correlation (Kornbrot, 2014) between person ability and item score is reported, together with the expected value of the point-measure polyserial correlation for an ideal item. Default is `point_measure_corr=False`. Automatically `True` if `full=True`.
`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during item estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for item estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute: one of `self.item_stats_global`, `self.item_stats_items`, `self.item_stats_thresholds` or `self.item_stats_matrix`, a pandas dataframe with one row for each item and the following columns:

`Estimate`	Central item difficulty estimate.
`SE`	Bootstrapped standard error of central item difficulty estimate.
`2.5%`	Lower bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 2.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `5%`.
`97.5%`	Upper bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 97.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `95%`.
`Count`	Count of responses.
`Facility`	Item facility: proportion of correct responses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score. Only produced if `zstd=True`.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score. Only produced if `zstd=True`.
`PM corr`	Point-biserial correlation. Only produced if `point_measure_corr=True`.
`Exp PM corr`	Expected value of point-measure biserial correlation. Only produced if `point_measure_corr=True`.
`Disordered`	If `True`, threshold estimates indicate the presence of disordered thresholds within the item (Andrich, 2010).

Examples

To produce a summary unanchored `self.item_stats_global` table with the most commonly reported statistics:

```
self.item_stats_df_global()
```

To produce a full `self.item_stats_items` table with all statistics, anchored to raters `Rater_1` and `Rater_2`:

```
self.item_stats_df_items(anchor_raters=['Rater_1', 'Rater_2'], full=True)
```

To produce an unanchored `self.item_stats_matrix` table with with the most commonly reported statistics, plus standardised z-scores for infit and outfit:

```
self.item_stats_df_matrix(zstd=True)
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

6.4.2 `threshold_stats_df`

Description

Produces a table of threshold statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.threshold_stats_df_global(anchor_raters=None, full=False, zstd=False, disc=False,
                               point_measure_corr=False, dp=3, warm_corr=True,
                               tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5,
                               method='cos', constant=0.1, no_of_samples=100,
                               interval=None)
```

```
self.threshold_stats_df_items(anchor_raters=None, full=False, zstd=False, disc=False,
                              point_measure_corr=False, dp=3, warm_corr=True,
                              tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5,
                              method='cos', constant=0.1, no_of_samples=100,
                              interval=None)
```

```
self.threshold_stats_df_thresholds(anchor_raters=None, full=False, zstd=False, disc=False,
                                   point_measure_corr=False, dp=3, warm_corr=True,
                                   tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5,
                                   method='cos', constant=0.1, no_of_samples=100,
                                   interval=None)
```

```
self.threshold_stats_df_matrix(anchor_raters=None, full=False, zstd=False, disc=False,
                               point_measure_corr=False, dp=3, warm_corr=True,
                               tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5,
                               method='cos', constant=0.1, no_of_samples=100,
                               interval=None)
```

Arguments

`anchor_raters`	List: If `None`, unanchored central item difficulty estimates and standard errors will be used. If a list of rater names is passed, anchored estimates and standard errors, anchored to the raters passed, will be used. Other statistics are unaffected by `anchor_raters`. Default is `anchor_raters=None`.
`full`	Boolean: If `True`, a full table with all available statistics is produced. Default is `full=False` for a summary table with the most commonly reported statistics.

Arguments continue on the next page.

Arguments (continued)

`zstd`	Boolean: If `True`, infit and outfit standardised z-scores are reported alongside mean square statistics. Default is `zstd=False`. Automatically `True` if `full=True`.
`disc`	Boolean: If `True`, item discrimination is reported. The discrimination of the empirical item slope relative to the ideal logistic ogive, with 1 perfect, greater than 1 showing overfit and less than 1 showing underfit; discrimination is similar to the 2PL IRT discrimination parameter (Linacre, 2023), but is a descriptive statistic in the SLM rather than an item parameter.
`point_measure_corr`	Boolean: If `True`, point-polyserial correlation (Kornbrot, 2014) between person ability and item score is reported, together with the expected value of the point-measure polyserial correlation for an ideal item. Default is `point_measure_corr=False`. Automatically `True` if `full=True`.
`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 5.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`

Arguments continue on the next page.

Arguments (continued)

`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns
Attribute `self.threshold_stats_global`, `self.threshold_stats_items`, `self.threshold_stats_thresholds` or `self.threshold_stats_matrix`, a pandas dataframe with one row for each threshold and the following columns:

`Estimate`	Rasch-Andrich threshold difficulty estimate.
`SE`	Bootstrapped standard error of threshold estimate.
`2.5%`	Lower bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 2.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `5%`.
`97.5%`	Upper bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 97.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `95%`.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score. Only produced if `zstd=True`.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score. Only produced if `zstd=True`.
`Discrim`	Item discrimination. Only produced if `disc=True`.
`PM corr`	Point-biserial correlation. Only produced if `point_measure_corr=True`.
`Exp PM corr`	Expected value of point-measure biserial correlation. Only produced if `point_measure_corr=True`.

Examples

To produce an unanchored summary `self.threshold_stats` table under the global representation with the most commonly reported statistics:

```
self.threshold_stats_df_global()
```

To produce a full `self.threshold_stats` table under the item representation with all statistics, anchored to raters `Rater_1` and `Rater_2`:

```
self.threshold_stats_df(anchor_raters=['Rater_1', 'Rater_2'], full=True)
```

To produce an unanchored `self.threshold_stats` table under the matrix representation with with the most commonly reported statistics, plus standardised z-scores for infit and outfit:

```
self.threshold_stats_df_matrix(zstd=True)
```

Other arguments may be used to alter parameters of central item difficulty, threshold and person ability estimation.

6.4.3 `rater_stats_df`

Description

Produces a table of rater statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.rater_stats_df_global(anchor_raters=None, full=False, zstd=False,
                           dp=3, warm_corr=True, tolerance=1e-07,
                           max_iters=100, ext_score_adjustment=0.5, method='cos',
                           constant=0.1, no_of_samples=100, interval=None)
```

```
self.rater_stats_df_items(anchor_raters=None, full=False, zstd=False,
                           dp=3, warm_corr=True, tolerance=1e-07,
                           max_iters=100, ext_score_adjustment=0.5, method='cos',
                           constant=0.1, no_of_samples=100, interval=None)
```

```
self.rater_stats_df_thresholds(anchor_raters=None, full=False, zstd=False,
                           dp=3, warm_corr=True,
                           tolerance=1e-07, max_iters=100, ext_score_adjustment=0.5,
                           method='cos', constant=0.1, no_of_samples=100, interval=None)
```

```
self.rater_stats_df_matrix(anchor_raters=None, full=False, zstd=False,
                           dp=3, warm_corr=True, tolerance=1e-07,
                           max_iters=100, ext_score_adjustment=0.5, method='cos',
                           constant=0.1, no_of_samples=100, interval=None)
```

Arguments

anchor_raters	List: If None, unanchored central item difficulty estimates and standard errors will be used. If a list of rater names is passed, anchored estimates and standard errors, anchored to the raters passed, will be used. Other statistics are unaffected by anchor_raters. Default is anchor_raters=None.
full	Boolean: If True, a full table with all available statistics is produced. Default is full=False for a summary table with the most commonly reported statistics.
zstd	Boolean: If True, infit and outfit standardised z-scores are reported alongside mean square statistics. Default is zstd=False. Automatically True if full=True.
dp	Integer: Number of decimal places reported in output.
warm_corr	Boolean: If True, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is warm_corr=True.
tolerance	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is tolerance=1e-07.
max_iters	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is max_iters=100.
ext_score_adjustment	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to ext_score_adjustment (if zero) or the maximum possible score minus ext_score_adjustment (if maximum score) before estimating ability.
method	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during item estimation (see section 3.3.1). Options are 'ls' for least squares (Choppin, 1968, 1985), 'evm' for the eigenvector method(Garner & Engelhard, 2002), 'cos' for cosine similarity (Kou & Lin, 2014) or 'log-lik' for (iterative) log-likelihood (Bradley & Terry, 1952). Default is method='cos'
constant	Float: additive smoothing constant (Elliott & Buttery, 2022b) for item estimation (see section 3.3.1). Default value is constant=0.1.

Arguments continue on the next page.

Arguments (continued)

`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute: one of `self.rater_stats_global`, `self.rater_stats_items`, `self.rater_stats_thresholds` or `self.rater_stats_matrix`, a pandas dataframe with one row for each item and the following columns:

`Estimate`	Central item difficulty estimate.
`SE`	Bootstrapped standard error of central item difficulty estimate.
`2.5%`	Lower bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 2.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `5%`.
`97.5%`	Upper bound of empirical bootstrap estimate interval. Column name will change to fit `interval` value: 97.5% represents the lower bound for `interval=0.95` but, for example, `interval=0.9` will produce a column labelled `95%`.
`Count`	Count of responses.
`Facility`	Item facility: proportion of correct responses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score. Only produced if `zstd=True`.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score. Only produced if `zstd=True`.
`PM corr`	Point-biserial correlation. Only produced if `point_measure_corr=True`.
`Exp PM corr`	Expected value of point-measure biserial correlation. Only produced if `point_measure_corr=True`.

Examples
To produce a summary unanchored `self.rater_stats_global` table with the most commonly reported statistics:

```
self.rater_stats_df_global()
```

To produce a full `self.rater_stats_items` table with all statistics, anchored to raters `Rater_1` and `Rater_2`:

```
self.rater_stats_df_items(anchor_raters=['Rater_1', 'Rater_2'], full=True)
```

To produce an unanchored `self.item_stats_matrix` table with with the most commonly reported statistics, plus standardised z-scores for infit and outfit:

```
self.item_stats_df_matrix(zstd=True)
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

6.4.4 `person_stats_df`

Description
Produces a table of person statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage
Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.person_stats_df_global(anchor_raters=None, full=False, rsem=False, dp=3,
                            warm_corr=True, tolerance=1e-07, max_iters=100,
                            ext_score_adjustment=0.5, method='cos', constant=0.1)

self.person_stats_df_items(anchor_raters=None, full=False, rsem=False, dp=3,
                           warm_corr=True, tolerance=1e-07, max_iters=100,
                           ext_score_adjustment=0.5, method='cos', constant=0.1)

self.person_stats_df_thresholds(anchor_raters=None, full=False, rsem=False, dp=3,
                                warm_corr=True, tolerance=1e-07, max_iters=100,
                                ext_score_adjustment=0.5, method='cos', constant=0.1)

self.person_stats_df_matrix(anchor_raters=None, full=False, rsem=False, dp=3,
                            warm_corr=True, tolerance=1e-07, max_iters=100,
                            ext_score_adjustment=0.5, method='cos', constant=0.1)
```

Arguments

`anchor_raters`	List: If `None`, unanchored central item difficulty estimates will be used to generate person ability estimates. If a list of rater names is passed, anchored estimates and standard errors, anchored to the raters passed, will be used. Other statistics are unaffected by `anchor_raters`. Default is `anchor_raters=None`.
`full`	Boolean: If `True`, a full table with all available statistics is produced. Default is `full=False` for a summary table with the most commonly reported statistics.
`rsem`	Boolean: If `True`, realistic standard error of measurement (RSEM), which takes into account for item misfit (Wright, 1996), is reported alongside the conditional standard error of measurement (CSEM). Default is `rsem=False`. Automatically `True` if `full=True`.
`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`

Arguments continue on the next page.

Arguments (continued)

`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is `constant=0.1`.
`no_of_samples`	Integer: Number of bootstrap samples to generate for standard error estimates (see Section 3.3.2). More samples lead to more accurate standard error estimates, but take correspondingly longer to compute. Default is `no_of_samples=100`.
`interval`	Float. Empirical interval to define quantiles of estimates from bootstrap samples, as an alternative to a confidence interval (see Section 3.3.2). Defines a central interval of proportion p to determine upper and lower bounds of $1-(1-p)/2$ and $(1-p)/2$, e.g. `interval=0.9` defines quantiles at 2.5% and 97.5%. More stable with larger numbers of bootstrap samples. Default is `interval=None`.

Returns

Attribute `self.person_stats_global`, `self.person_stats_items`, `self.person_stats_thresholds`, `self.person_stats_matrix`, a pandas dataframe with one row for each person and the following columns:

`Estimate`	Item difficulty estimate.
`CSEM`	Conditional standard error of measurement for person ability estimate.
`RSEM`	Realistic standard error of measurement for person ability estimate. Only produced if `rsem=True`
`Score`	Number of correct responses.
`Max score`	Maximum available score (number of items attempted).
`p`	Proportion of correct repsonses.
`Infit MS`	Infit mean square.
`Infit Z`	Standardised infit z-score.
`Outfit MS`	Outfit mean square.
`Outfit Z`	Standardised outfit z-score.

Examples

To produce an unanchored summary `self.person_stats` table under the global rater representation with the most commonly reported statistics:

```
self.person_stats_df_global()
```

To produce a full `self.person_stats` table under the matrix rater representation with all statistics, anchored to raters `Rater_1` and `Rater_2`:

```
self.person_stats_df(anchor_raters=['Rater_1', 'Rater_2'], full=True)
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

6.4.5 `test_stats_df`

Description

Produces a table of test-level statistics in the form of a pandas dataframe, which may be saved to formats such as csv, xslx or LaTeX.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.test_stats_df_global(dp=3, warm_corr=True, tolerance=1e-07, max_iters=100,
                          ext_score_adjustment=0.5, method='cos', constant=0.1)


self.test_stats_df_items(dp=3, warm_corr=True, tolerance=1e-07, max_iters=100,
                         ext_score_adjustment=0.5, method='cos', constant=0.1)


self.test_stats_df_thresholds(dp=3, warm_corr=True, tolerance=1e-07, max_iters=100,
                              ext_score_adjustment=0.5, method='cos', constant=0.1)


self.test_stats_df_matrix(dp=3, warm_corr=True, tolerance=1e-07, max_iters=100,
                          ext_score_adjustment=0.5, method='cos', constant=0.1)
```

Arguments

`dp`	Integer: Number of decimal places reported in output.
`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.

Arguments continue on the next page.

Arguments (continued)

`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) during central item difficulty estimation (see section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for central item difficulty estimation (see section 3.3.1). Default value is `constant=0.1`.

Returns

Attribute `self.test_stats`, a pandas dataframe with two columns, `Items` and `Persons` and rows for a range of descriptive statistics describing the distributions of the estimates and different statistics related to reliability – these statistics describe the suitability of the data for estimating and differentiating the parameters, rather than properties of the parameters themselves. The statistics are:

`Mean`	The mean of the estimates.
`SD`	The standard deviation of the estimates.
`Separation ratio`	The separation ratio (Wright, 1996; Wright & Masters, 1982), which is the standard deviation of person abilities reported as a ratio of standard error units. For persons: $$G_p = \sigma_p / \sqrt{\sum_n SE_n^2}$$ where σ_p is the variance of the person estimates and SE_n is the RSEM (see Section 3.4.2) for person n. The formula is symmetrical for items, substituting the standard error of estimation for RSEM.
`Strata`	The number of statistically distinct levels of either person ability or item difficulty as strata with centers three measurement errors apart (Wright & Masters, 1982:106). For persons: $$H_p = (4G_p + 1)/3$$ with symmetrical results for items.

Returns continue on the next page.

Returns (continued)

`Reliability`	A Rasch-specific reliability statistic (Wright, 1996), derived from PSI and which is a Rasch-specific reliability statistic similar to Cronbach's Alpha (Cronbach, 1951), and which may be interpreted the same way – as the proportion of variance of the estimates which stems from variation in ability or difficulty rather than estimation error. For persons: $$R_p = G_p^2/(1+G_p^2)$$ with symmetrical results for items.

Example

To produce a `self.test_stats` table under the global rater representation:

```
self.test_stats_df_global()
```

Other arguments may be used to alter parameters of item difficulty and/or person ability estimation.

6.4.6 `item_res_corr_analysis`

Description

Analysis of correlations of standardised residuals by item to tests for violations of local item interdependence and unidimensionality requirements.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.item_res_corr_analysis_global(warm_corr=True, tolerance=1e-07, max_iters=100,
                                   ext_score_adjustment=0.5, constant=0.1, method='cos',
                                   matrix_power=3, log_lik_tol=0.000001)

self.item_res_corr_analysis_items(warm_corr=True, tolerance=1e-07, max_iters=100,
                                  ext_score_adjustment=0.5, constant=0.1, method='cos',
                                  matrix_power=3, log_lik_tol=0.000001)

self.item_res_corr_analysis_thresholds(warm_corr=True, tolerance=1e-07, max_iters=100,
                                       ext_score_adjustment=0.5, constant=0.1,
                                        method='cos', matrix_power=3, log_lik_tol=0.000001)

self.item_res_corr_analysis_matrix(warm_corr=True, tolerance=1e-07, max_iters=100,
                                   ext_score_adjustment=0.5, constant=0.1, method='cos',
                                   matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.
`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for generation of item difficulty estimates (see Section 3.3.1). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) for generation of central item difficulty estimates (see Section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) is raised for generation of central item difficulty estimates (see Section 3.3.1). Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for generation of item difficulty estimates (see Section 3.3.1). Ignored for other methods.

Returns

For tests of violation of the requirement for local item independence (Andrich & Kreiner, 2010; Marais, 2012):

`self.item_residual_correlations_global`, `self.item_residual_correlations_items`, `self.item_residual_correlations_thresholds` or `self.item_residual_correlations_matrix`	A pandas dataframe of pairwise correlations between item standardised residuals.

For tests of violation of the requirement for unidimensionality based on principal component analysis of the standardised residual correlations (Pallant & Tennant, 2007; Smith, 2002):

`self.item_eigenvectors_global`, `self.item_eigenvectors_items`, `self.item_eigenvectors_thresholds` or `self.item_eigenvectors_matrix`	The eigenvectors of the standardised residual correlations matrix.
`self.item_eigenvalues_global`, `self.item_eigenvalues_items`, `self.item_eigenvalues_thresholds` or `self.item_eigenvalues_matrix`	The eigenvalues corresponding to the eigenvectors.
`self.item_variance_explained_global` `self.item_variance_explained_items`, `self.item_variance_explained_thresholds` or `self.item_variance_explained_matrix`	The variance explained by each principal component.
`self.item_loadings_global`, `self.item_loadings_items`, `self.item_loadings_thresholds` or `self.item_loadings_matrix`	The loading of each item onto each of the principal components, for the the first of which large loadings (‘large'typically interpreted as > 0.4 or < -0.4) may be interpreted as representing the presence of significant dimensionality, in analogy to factor analysis (**<empty citation>**).

Example

To produce an item residual correlation analysis under the global rater representation:

```
self.item_res_corr_analysis_global()
```

Arguments may be used to alter parameters of item difficulty and/or person ability estimation.

6.4.7 `rater_res_corr_analysis`

Description

Analysis of correlations of standardised residuals by item to tests for violations of local item interdependence and unidimensionality requirements.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.rater_res_corr_analysis_global(warm_corr=True, tolerance=1e-07, max_iters=100,
                                    ext_score_adjustment=0.5, constant=0.1, method='cos',
                                    matrix_power=3, log_lik_tol=0.000001)
```

```
self.rater_res_corr_analysis_items(warm_corr=True, tolerance=1e-07, max_iters=100,
                                   ext_score_adjustment=0.5, constant=0.1, method='cos',
                                   matrix_power=3, log_lik_tol=0.000001)
```

```
self.rater_res_corr_analysis_thresholds(warm_corr=True, tolerance=1e-07, max_iters=100,
                                        ext_score_adjustment=0.5, constant=0.1,
                                        method='cos', matrix_power=3, log_lik_tol=0.000001)
```

```
self.rater_res_corr_analysis_matrix(warm_corr=True, tolerance=1e-07, max_iters=100,
                                    ext_score_adjustment=0.5, constant=0.1, method='cos',
                                    matrix_power=3, log_lik_tol=0.000001)
```

Arguments

`warm_corr`	Boolean: If `True`, Warm's bias correction (Warm, 1989) is applied during generation of person ability estimates (see Section 6.3.4). Default is `warm_corr=True`.
`tolerance`	Float: convergence stopping criterion for Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `tolerance=1e-07`.
`max_iters`	Integer: maximum number of Newton-Raphson iterations during generation of person ability estimates (see Section 6.3.4). Default is `max_iters=100`.
`ext_score_adjustment`	Float: Value in range $(0, 1)$ to ensure a estimate is returned during generation of person ability estimates (see Section 6.3.4) if a person has an extreme score (all items responded to are correct or incorrect). Since there is no finite ML ability estimate for extreme scores, this adjusts the person's score to `ext_score_adjustment` (if zero) or the maximum possible score minus `ext_score_adjustment` (if maximum score) before estimating ability.

Arguments continue on the next page.

Arguments (continued)

`constant`	Float: additive smoothing constant (Elliott & Buttery, 2022b) for generation of item difficulty estimates (see Section 3.3.1). Default value is `constant=0.1`.
`method`	String: method for derivation of vector of estimates from pairwise reciprocal matrix (Elliott & Buttery, 2022b:991–992) for generation of central item difficulty estimates (see Section 3.3.1). Options are `'ls'` for least squares (Choppin, 1968, 1985), `'evm'` for the eigenvector method(Garner & Engelhard, 2002), `'cos'` for cosine similarity (Kou & Lin, 2014) or `'log-lik'` for (iterative) log-likelihood (Bradley & Terry, 1952). Default is `method='cos'`
`matrix_power`	Integer: power to which conditional category response frequency matrix (Elliott & Buttery, 2022b:991) is raised for generation of central item difficulty estimates (see Section 3.3.1). Each additional power increases the number of indirect pairwise comparisons (Choppin, 1985; Elliott & Buttery, 2022b).
`log_lik_tol`	Float: convergence stopping criterion for log-likelihood method for generation of item difficulty estimates (see Section 3.3.1). Ignored for other methods.

Returns
For tests of violation of the requirement for local rater independence (Andrich & Kreiner, 2010; Marais, 2012):

`self.rater_residual_correlations_global`, `self.rater_residual_correlations_items`, `self.rater_residual_correlations_thresholds` or `self.rater_residual_correlations_matrix`	A pandas dataframe of pairwise correlations between item standardised residuals.

For tests of violation of the requirement for unidimensionality based on principal component analysis of the standardised residual correlations (Pallant & Tennant, 2007; Smith, 2002):

`self.rater_eigenvectors_global`, `self.rater_eigenvectors_items`, `self.rater_eigenvectors_thresholds` or `self.rater_eigenvectors_matrix`	The eigenvectors of the standardised residual correlations matrix.

Returns continue on the next page.

Returns (continued)

`self.rater_eigenvalues_global`, `self.rater_eigenvalues_items`, `self.rater_eigenvalues_thresholds` or `self.rater_eigenvalues_matrix`	The eigenvalues corresponding to the eigenvectors.
`self.rater_variance_explained_global` `self.rater_variance_explained_items`, `self.rater_variance_explained_thresholds` or `self.rater_variance_explained_matrix`	The variance explained by each principal component.
`self.rater_loadings_global`, `self.rater_loadings_items`, `self.rater_loadings_thresholds` or `self.rater_loadings_matrix`	The loading of each rater onto each of the principal components, for the the first of which large loadings ('large'typically interpreted as > 0.4 or < -0.4) may be interpreted as representing the presence of significant dimensionality, in analogy to factor analysis (**<empty citation>**).

Example
To produce a rater residual correlation analysis under the global rater representation:

`self.rater_res_corr_analysis_global()`

Arguments may be used to alter parameters of item difficulty and/or person ability estimation.

6.4.8 `category_counts_df`

Description
Produces a table of counts of scores in each category, plus responses and missing responses, for each item.

Usage
`self.category_counts_df()`

Arguments None

Returns
Attributes:

`self.category_counts`	pandas dataframe: pandas dataframe of category counts with one row per item and one column per response category, plus total responses per item and missing responses per item.

Returns continue on the next page.

Returns (continued)

`self.category_counts_raters`	pandas multiindex dataframe: For each rater, pandas dataframe of category counts rated by the rater, with one row per item and one column per response category, plus total responses per item and missing responses per item. Formatted as a pandas multiindex dataframe with raters as the outer index level and items as the inner index level.

Example

To produce a dataframe of category counts:

```
self.category_counts_df()
```

6.5 Plotting functionality

6.5.1 Shared plotting arguments

All the plotting methods described in this section share a set of arguments which may be used to customise the appearance of the plot or save the plot to file automatically. These arguments are:

`title`	String: Title for the plot, to appear in the image. Default is `title=None`.
`xmin`	Float: Minimum displayed point on x-axis, in logits. Default is `xmin=-5`.
`xmax`	Float: Maximum displayed point on x-axis, in logits. Default is `xmax=5`.
`plot_style`	String: Plot style to use. Available styles are `'white'`, producing a plot on a white background, and `'dark'`, producing a plot on a grey background. The styles correspond to Seaborn (Waskom, 2021) styles `whitegrid` and `darkgrid`. Default is `plot_style='white'`.
`palette`	String: Controls colours of lines and overplotted class intervals dots. Options are `'dark blue'`, `'light blue'`, `'dark red'`, `'light red'`, `'dark green'`, `'light green'`, `'dark grey'`, `'light grey'`, `'dark multi'` and `'light multi'`. Default is `palette='dark blue'`. Not used for `std_residual_plot`.
`black`	Boolean: If `True`, the plot will be rendered in black and white. Default is `black=False`.
`font`	String: The font to use in the plot. Default is `font='Times'`.
`title_font_size`	Float: The size of the title font in points. Default is `title_font_size=15`.
`axis_font_size`	Float: The size of the axis label font in points. Default is `axis_font_size=15`.
`labelsize`	Float: The size of the axis tick label font in points. Default is `labelsize=15`.
`filename`	String: The filename for the saved plot, with no suffix for format. If `None`, no file will be saved. Default is `filename=None`.
`file_format`	The format of the file: `png`, `jpg` or `svg`. Default is `file_format=png`.
`dpi`	The resolution of the plot in dpi (dots per inch) – higher resolution plots are better quality but have larger file sizes. Default is `dpi=300`.

6.5.2 `icc`

Description

Plots the item characteristic curve (or item response function) for an item: person ability on the x-axis against expected score on the y-axis. Options to plot observed responses, item threshold line and lines showing abilities corresponding to specified expected scores, and to highlight a specified response category.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.icc_global(item, anchor=False, rater=None, obs=False, no_of_classes=5,
                thresh_lines=False, central_diff=False,
                score_lines=None, score_labels=False, cat_highlight=None)
```

```
self.icc_items(item, anchor=False, rater=None, obs=False, no_of_classes=5,
               thresh_lines=False, central_diff=False,
               score_lines=None, score_labels=False, cat_highlight=None)
```

```
self.icc_thresholds(item, anchor=False, rater=None, obs=False, no_of_classes=5,
                    thresh_lines=False, central_diff=False,
                    score_lines=None, score_labels=False, cat_highlight=None)
```

```
self.icc_matrix(item, anchor=False, rater=None, obs=False, no_of_classes=5,
                thresh_lines=False, central_diff=False,
                score_lines=None, score_labels=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`anchor`	Boolean: If `True`, anchored parameters are used; if `False`, unanchored parameters are used. Default is `anchor=False`.
`rater`	String: If a `rater` argument is passed, plots the item characteristic curve for the item as rated by the given rater. If `rater=None`, plots a neutral item characteristic curve for a rater with zero severity. Default is `rater=None`.
`obs`	Boolean: If `True`, mean observed scores for each of the ordered response categories will be plotted against the mean ability of the corresponding response class. Default is `obs=False`.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.

Arguments continue on the next page.

Arguments (continued)

thresh_lines	Boolean: If True, vertical lines showing the thresholds between each response category will be plotted: uncentred thresholds when a rater argument is passed, centred thresholds when rater=None. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is thresh_line=False.
central_diff	Boolean: If True, a vertical line is plotted. If a uncentred thresholds when a rater argument is passed, the line shows the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1); if rater=None, the line will be at zero severity (the mean of the centred thresholds by definition). Default is central_diff=False.
score_lines	List of floats between 0 and self.max_score: Each float in the list represents an expected score, for which a horizontal line from the y-axis to where it intersects the item characteristic curve will be plotted, and from there a vertical line down to the x-axis will be plotted to show the ability corresponding to the expected score.
score_labels	Boolean: If True, scores and abilities corresponding to arguments passed to score_lines will be labelled on the plot. Default is score_labels=False.
cat_highlight	Integer: Passing a score between 0 and self.max_score will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is cat_highlight=None (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section **??**.

Returns

matplotlib image. If a filename argument is passed, also saves the image to file.

Examples

To plot a basic unanchored item characteristic curve for Item_1 under the global rater representation and store the output as a variable my_icc_plot and save it to file as my_icc_plot.png:

```
my_icc_plot = self.icc_global('Item_1', filename=my_icc_plot)
```

To plot an anchored item characteristic curve for Item_1 under the item rater representation, rated by Rater_1, with observed responses for 8 response classes and store the output as a variable my_icc_plot:

```
my_icc_plot = self.icc_items('Item_1', anchor=True, rater='Rater_1', obs=True, no_of_classes=8)
```

To plot an item characteristic curve for Item_1 under the threshold rater representation with a threshold line and highlighted zero category, and store the output as a variable my_icc_plot:

```
my_icc_plot = self.icc_thresholds('Item_1', thresh_line=True, cat_highlight=0)
```

To plot an item characteristic curve for `Item_1` under the matrix rater representation with lines showing the abilities corresponding to expected scores of 0.7 and 1.6, with the expected score and corresponding ability labelled, and store the output as a variable `my_icc_plot`:

```
my_icc_plot = self.icc_matrix('Item_1', score_lines=[0.7, 1.6], score_labels=True)
```

6.5.3 `crcs`

Description

Plots category response curves for an item: person ability on the x-axis against expected the probability of obtaining a score in each category (0 or 1) on the y-axis. Options to plot observed proportions and item threshold line, and to highlight a specified response category.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.crcs_global(item=None, anchor=False, rater=None, obs=None, no_of_classes=5,
                 thresh_lines=False, central_diff=False, cat_highlight=None)

self.crcs_items(item=None, anchor=False, rater=None, obs=None, no_of_classes=5,
                thresh_lines=False, central_diff=False, cat_highlight=None)

self.crcs_thresholds(item=None, anchor=False, rater=None, obs=None, no_of_classes=5,
                     thresh_lines=False, central_diff=False, cat_highlight=None)

self.crcs_matrix(item=None, anchor=False, rater=None, obs=None, no_of_classes=5,
                 thresh_lines=False, central_diff=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot. If `item=None`, the shared set of Rasch-Andrich thresholds with mean zero will be plotted, and any observed proportions overplotted using `obs` will use data from all items. Default is `item=None`.
`anchor`	Boolean: If `True`, anchored parameters are used; if `False`, unanchored parameters are used. Default is `anchor=False`.
`rater`	String: If a `rater` argument is passed, plots the category response curves as rated by the given rater, adjusted by rater severity. If `rater=None`, plots a neutral category response curves for a rater with zero severity. Default is `rater=None`.

Arguments continue on the next page.

Arguments (continued)

`obs`	List: List of integers between 0 and `self.max_score`. For each value, mean observed proportions in each ordered response category scoring in that category are plotted against the mean ability of the corresponding response class.
`no_of_classes`	Integer: The number of observed response categories. Default is `no_of_classes=5`.
`thresh_lines`	Boolean: If `True`, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is `thresh_line=False`.
`central_diff`	Boolean: If `True`, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is `central_diff=False`.
`cat_highlight`	Integer: Passing a score between 0 and `self.max_score` will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is `cat_highlight=None` (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section 6.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot basic unanchored category response curves under the global rater representation, rated by rater `Rater_1`, and store the output as a variable `my_crcs_plot` and save it to file as `my_crcs_plot.png`:

```
my_crcs_plot = self.crcs_global(rater='Rater_1', filename=my_crcs_plot)
```

To plot anchored category response curves for `Item_1` under the threshold rater representation, rated by rater `Rater_1`, with observed response proportions for category 1 for 8 response classes and store the output as a variable `my_crcs_plot`:

```
my_crcs_plot = self.crcs_thresholds(item='Item_1', anchor=True, rater='Rater_1', obs=[1],
                                    no_of_classes=8)
```

To plot unanchored category response curves for `Item_1` under the matrix rater representation, rated by a neutral ‘zero severity’ rater, with a threshold line and highlighted zero category, and store the output as a variable `my_crcs_plot`:

```
my_crcs_plot = self.crcs_matrix(item='Item_1', thresh_line=True, cat_highlight=0)
```

6.5.4 `threshold_ccs`

Description

Plots the threshold characteristic curves (or threshold response functions)for an item. For threshold τ_k, $k \in \{1, ..., \texttt{self.max\_score}\}$, the threshold characteristic curve is the probability of obtaining a score of k rather than $k-1$, conditional on the score being either $k-1$ rather than k, for a given person ability. Each threshold characteristic curve functions as a dichotomous item characteristic curve under the SLM (see Sections 3.2.2 and 3.5.2).

For threshold τ_k, `threshold_ccs` plots person ability on the x-axis against the probability of obtaining a score of k on the y-axis. Options to plot threshold lines and central item difficulties, and to highlight a specified response category.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.threshold_ccs_global(item=None, anchor=False, rater=None, obs=None, no_of_classes=5,)
                          thresh_lines=False, central_diff=False, cat_highlight=None)
```

```
self.threshold_ccs_items(item=None, anchor=False, rater=None, obs=None, no_of_classes=5,)
                         thresh_lines=False, central_diff=False, cat_highlight=None)
```

```
self.threshold_ccs_thresholds(item=None, anchor=False, rater=None, obs=None, no_of_classes=5,)
                              thresh_lines=False, central_diff=False, cat_highlight=None)
```

```
self.threshold_ccs_matrix(item=None, anchor=False, rater=None, obs=None, no_of_classes=5,)
                          thresh_lines=False, central_diff=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot. If `item=None`, the shared set of Rasch-Andrich thresholds with mean zero will be plotted, and any observed proportions overplotted using `obs` will use data from all items. Default is `item=None`.
`anchor`	Boolean: If `True`, anchored parameters are used; if `False`, unanchored parameters are used. Default is `anchor=False`.
`rater`	String: If a `rater` argument is passed, plots the category response curves for the item as rated by the given rater. If `rater=None`, plots neutral category response curves for a rater with zero severity. Default is `rater=None`.

Arguments continue on the next page.

Arguments (continued)

obs	List: List of integers corresponding to thresholds τ_1 to τ_m, where m =self.max_score, or 'all' or 'none'. If obs=[k], mean proportions of persons obtaining a score of k rather than $k-1$, conditional on the score being either $k-1$ or k, for each of the ordered response categories, will be plotted against the mean ability of the corresponding response class. Multiple thresholds may be passed. Default is obs=None.
no_of_classes	Integer: The number of observed response categories. Default is no_of_classes=5.
thresh_lines	Boolean: If True, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is thresh_line=False.
central_diff	Boolean: If True, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is central_diff=False.
cat_highlight	Integer: Passing 0 or 1 will highlight the range of abilities for which the selected score is the most probable response (all abilities to one side of the item difficulty. Default is cat_highlight=None (no category highlighted).

Additional arguments to customise the appearance of the plot are detailed in Section **??**.

Returns

matplotlib image. If a filename argument is passed, also saves the image to file.

Examples

To plot basic unanchored threshold characteristics curves under the global rater representation, rated by Rater_1, and store the output as a variable my_threshold_ccs_plot, and save it to file as my_threshold_ccs_plot.png:

```
my_threshold_ccs_plot = self.threshold_ccs_global(rater='Rater_1', filename='my_iic_plot')
```

To plot anchored threshold characteristics curves for Item_1 under the item rater representation, for a neutral 'zero' rater with threshold lines and category 1 highlighted, and store the output as a variable my_threshold_ccs_plot:

```
my_threshold_ccs_plot = self.threshold_ccs_items(item='Item_1', anchor=True,
                                                  thresh_lines=True, cat_highlight=1)
```

To plot unanchored threshold characteristics curves for `Item_1` under the matrix rater representation, rated by `Rater_1`, with observed responses plotted for thresholds 2 and 4 and central item difficulty line, and store the output as a variable `my_threshold_ccs_plot`:

```
my_threshold_ccs_plot = self.threshold_ccs_matrix(item='Item_1', rater='Rater_1',
                                                  obs=[2, 4], central_diff=True)
```

6.5.5 `iic`

Description

Plots the item information curve for an item: person ability on the x-axis against Fisher information on the y-axis. Options to plot item threshold line and lines showing Fisher information corresponding to specified abilities, and to highlight a specified response category.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.iic_global(item, anchor=False, rater=None, ymax=None, thresh_lines=False,
                thresh_lines=False, central_diff=False, point_info_lines=None,
                point_info_labels=False, cat_highlight=None)

self.iic_items(item, anchor=False, rater=None, ymax=None, thresh_lines=False,
               thresh_lines=False, central_diff=False, point_info_lines=None,
               point_info_labels=False, cat_highlight=None)

self.iic_thresholds(item, anchor=False, rater=None, ymax=None, thresh_lines=False,
                    thresh_lines=False, central_diff=False, point_info_lines=None,
                    point_info_labels=False, cat_highlight=None)

self.iic_matrix(item, anchor=False, rater=None, ymax=None, thresh_lines=False,
                thresh_lines=False, central_diff=False, point_info_lines=None,
                point_info_labels=False, cat_highlight=None)
```

Arguments

`item`	String: The name of the item to plot.
`anchor`	Boolean: If `True`, anchored parameters are used; if `False`, unanchored parameters are used. Default is `anchor=False`.
`rater`	String: If a `rater` argument is passed, plots the item information curve for the item as rated by the given rater. If `rater=None`, plots a neutral item information curve for a rater with zero severity. Default is `rater=None`.

Arguments continue on the next page.

Arguments (continued)

ymax	Float: The maximum value to show on the y-axis. If None, will infer, plotting a maximum of 1.1 times the maximum item information. Default is ymax=None
thresh_lines	Boolean: If True, vertical lines showing the uncentred thresholds between each response category will be plotted. Threshold τ_k is the person ability for which the scores $k-1$ and k are equally probable. Default is thresh_line=False.
central_diff	Boolean: If True, a vertical line showing the threshold corresponding to the central item difficulty (the mean of the uncentred thresholds – see Section 5.2.1). Default is central_diff=False.
point_info_lines	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the item information curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the Fisher information corresponding to the ability.
point_info_labels	Boolean: If True, abilities and Fisher information corresponding to arguments passed to point_info_lines will be labelled on the plot. Default is point_info_labels=False.
cat_highlight	Integer: Passing a score between 0 and self.max_score will highlight the range of abilities for which the selected score is the most probable response. If the data has disordered thresholds (Andrich, 2010; Pallant & Tennant, 2007) and response category selected is never the most probable, no area will be highlighted. Default is cat_highlight=None (no category highlighted).
ymax	The maximum point displayed on the y-axis, in Fisher information.

Additional arguments to customise the appearance of the plot are detailed in Section 6.5.1.

Returns
matplotlib image. If a filename argument is passed, also saves the image to file.

Examples
To plot a basic unanchored item information curve for Item_1 under the global rater representation, rated by Rater_1, and store the output as a variable my_iic_plot and save it to file as my_iic_plot.png:

```
my_iic_plot = self.iic_global('Item_1', rater='Rater_1', filename='my_iic_plot')
```

To plot an anchored item information curve for `Item_1` under the item rater representation, for a neutral 'zero' rater with threshold lines, central item difficulty line and category 1 highlighted, and store the output as a variable `my_iic_plot`:

```
my_iic_plot = self.iic_items('Item_1', anchor=True, thresh_line=True, central_diff=True,
                              cat_highlight=1)
```

To plot an unanchored item information curve for `Item_1` under the matrix rater representation, rated by `Rater_1`, with lines showing the Fisher information corresponding to abilities of -0.3 and 0.7, with the ability and corresponding Fisher information labelled, and store the output as a variable `my_iic_plot`:

```
my_iic_plot = self.icc_matrix('Item_1', point_info_lines=[-0.3, 0.7],
                               point_info_labels=True)
```

6.5.6 `tcc`

Description
Plots the test characteristic curve (or test response function) for a set of items: person ability on the x-axis against expected score on the y-axis. Options to plot observed responses and lines showing abilities corresponding to specified expected scores.

Usage
Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.tcc_global(anchor=False, items=None, raters='zero', obs=False, no_of_classes=5,
                score_lines=None, score_labels=False)
```

```
self.tcc_items(anchor=False, items=None, raters='zero', obs=False, no_of_classes=5,
               score_lines=None, score_labels=False)
```

```
self.tcc_thresholds(anchor=False, items=None, raters='zero', obs=False, no_of_classes=5,
                    score_lines=None, score_labels=False)
```

```
self.tcc_matrix(anchor=False, items=None, raters='zero', obs=False, no_of_classes=5,
                score_lines=None, score_labels=False)
```

Arguments

`anchor`	Boolean: If `True`, anchored parameters are used; if `False`, unanchored parameters are used. Default is `anchor=False`.
`items`	List: The names of the items to be used in the plot. If `'all'`, the full set of items will be used. Default is `items='all'`.

Arguments continue on the next page.

Arguments (continued)

raters	List: The names of the raters to be used in the plot. raters='all' uses all raters. Default is raters='zero', which generates an ability based on a single neutral 'zero' rater. If multiple raters are selected, the maximum score will be the sum of the maximum scores across raters.
obs	Boolean: If True, mean observed scores for each of the ordered response categories will be plotted against the mean ability of the corresponding response class. Default is obs=False.
no_of_classes	Integer: The number of observed response categories. Default is no_of_classes=5.
score_lines	List of floats between 0 and self.max_score: Each float in the list represents an expected score, for which a horizontal line from the y-axis to where it intersects the item characteristic curve will be plotted, and from there a vertical line down to the x-axis will be plotted to show the ability corresponding to the expected score. Default is score_lines=None.
score_labels	Boolean: If True, scores and abilities corresponding to arguments passed to score_lines will be labelled on the plot. Default is score_labels=False.

Additional arguments to customise the appearance of the plot are detailed in Section 6.5.1.

Returns
matplotlib image. If a filename argument is passed, also saves the image to file.

Examples
To plot a basic unanchored test characteristic curve for all items under the global rater representation for a neutral 'zero' rater and store the output as a variable my_tcc_plot, and save it to file as my_tcc_plot.png:

```
my_tcc_plot = self.tcc_global(filename=my_tcc_plot)
```

To plot a test characteristic curve for Item_1 for a subset of items stored as a list my_item_list under the threshold rater representation for rater Rater_1, with observed responses for 8 response classes and store the output as a variable my_tcc_plot:

```
my_tcc_plot = self.tcc_thresholds(obs=True, no_of_classes=8)
```

To plot a test characteristic curve for Item_1 for all items under the matrix rater representation for all raters, with lines showing the abilities corresponding to expected scores of 13 and 20, with the expected score and corresponding ability labelled, and store the output as a variable my_tcc_plot:

```
my_tcc_plot = self.tcc_matrix(raters='all', score_lines=[13, 20], score_labels=True)
```

6.5.7 `test_info`

Description

Plots the test information curve: person ability on the x-axis against total Fisher information on the y-axis. Option to plot lines showing Fisher information corresponding to specified abilities.

Usage

Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.test_info_global(anchor=False, items=None, raters=None, ymax=None,
                      point_info_lines=None, point_info_labels=False)

self.test_info_items(anchor=False, items=None, raters=None, ymax=None,
                     point_info_lines=None, point_info_labels=False)

self.test_info_thresholds(anchor=False, items=None, raters=None, ymax=None,
                          point_info_lines=None, point_info_labels=False)

self.test_info_matrix(anchor=False, items=None, raters=None, ymax=None,
                      point_info_lines=None, point_info_labels=False)
```

Arguments

`anchor`	Boolean: If `True`, anchored parameters are used; if `False`, unanchored parameters are used. Default is `anchor=False`.
`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`raters`	List: The names of the raters to be used in the plot. `raters='all'` uses all raters. Default is `raters=None`, which generates an ability based on a single neutral 'zero' rater. If multiple raters are selected, the test information will be the sum of the information across raters.
`point_info_lines`	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the item information curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the total Fisher information corresponding to the ability. Default is `point_info_lines=None`.
`point_info_labels`	Boolean: If `True`, abilities and total Fisher information corresponding to arguments passed to `point_info_lines` will be labelled on the plot. Default is `point_info_labels=False`.
`ymax`	The maximum point displayed on the y-axis, in Fisher information.

Additional arguments to customise the appearance of the plot are detailed in Section 6.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot a basic anchored test information curve under the global rater representation using all items and a neutral 'zero' rater, storing the output as a variable `my_test_info_plot` and saving it to file as `my_test_info_plot.png`:

```
my_test_info_plot = self.test_info_global(anchor=True, filename='my_test_info_plot')
```

To plot an unanchored test information curve under the item rater representation using a subset of items stored as a list **`my_item_list`** and `Rater_1`, storing the output as a variable `my_test_info_plot`:

```
my_test_info_plot = self.test_info_items(items=my_item_list, raters='Rater_1')
```

To plot an anchored test information curve under the matrix rater representation with lines showing the total Fisher information corresponding to abilities of -0.3 and 0.7, with the ability and corresponding total Fisher information labelled, and store the output as a variable `my_test_info_plot`:

```
my_test_info_plot = self.test_info_matrix(anchor=True, point_info_lines=[-0.3, 0.7],
                                          point_info_labels=True)
```

6.5.8 `test_csem`

Description
Plots the test conditional standard error of measurement (CSEM) curve: person ability on the x-axis against CSEM (in logits) on the y-axis. Option to plot lines showing CSEM corresponding to specified abilities.

Usage
Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.test_csem_global (anchor=False, items=None, raters=None, point_csem_lines=None,
                       point_csem_labels=False, ymax=5)

self.test_csem_items (anchor=False, items=None, raters=None, point_csem_lines=None,
                      point_csem_labels=False, ymax=5)

self.test_csem_thresholds (anchor=False, items=None, raters=None, point_csem_lines=None,
                           point_csem_labels=False, ymax=5)

self.test_csem_matrix (anchor=False, items=None, raters=None, point_csem_lines=None,
                       point_csem_labels=False, ymax=5)
```

Arguments

anchor	Boolean: If True, anchored parameters are used; if False, unanchored parameters are used. Default is anchor=False.
items	List: The names of the items to be used in the plot. If None, the full set of items will be used. Default is items=None.
raters	List: The names of the raters to be used in the plot. Default is raters=None, which generates an ability based on a single neutral 'zero' rater. If multiple raters are selected, the CSEM will be based on the sum of the information across raters.
point_csem_lines	List of floats: Each float in the list represents an ability, for which a vertical line from the x-axis to where it intersects the CSEM curve will be plotted, and from there a horizontal line across to the y-axis will be plotted to show the CSEM corresponding to the ability. Default is point_csem_lines=None.
point_csem_labels	Boolean: If True, abilities and CSEM corresponding to arguments passed to point_csem_lines will be labelled on the plot. Default is point_csem_labels=False.
ymax	The maximum point displayed on the y-axis, in logits.

Additional arguments to customise the appearance of the plot are detailed in Section 6.5.1.

Returns
matplotlib image. If a filename argument is passed, also saves the image to file.

Examples
To plot a basic anchored CSEM curve for all items under the global rater representation for a neutral 'zero' rater, storing the output as a variable my_test_csem_plot and saving it to file as my_test_csem_plot.png:

```
my_test_csem_plot = self.test_csem_global(anchor=True, filename='my_test_csem_plot')
```

To plot an unanchored CSEM curve under the item rater representation using a subset of items stored as a list **my_item_list** and Rater_1, storing the output as a variable my_test_csem_plot:

```
my_test_csem_plot = self.test_csem_items(items=my_item_list, raters='Rater_1')
```

To plot an anchored CSEM curve under the matrix rater representation with lines showing the CSEM corresponding to abilities of -0.3 and 0.7, with the ability and corresponding CSEM labelled, and store the output as a variable my_test_csem_plot:

```
my_test_csem_plot = self.test_csem_matrix(anchor=True, point_info_lines=[-0.3, 0.7],
                                           point_info_labels=True)
```

6.5.9 `std_residuals_plot`

Description
Plots histogram of standardised residuals, with optional overplotting of standard Normal distribution.

Usage
Four methods are defined, one for each rater representation; all methods share the same argument format:

```
self.std_residuals_plot_global(items=None, raters=None, bin_width=0.5, normal=False)

self.std_residuals_plot_items(items=None, raters=None, bin_width=0.5, normal=False)

self.std_residuals_plot_thresholds(items=None, raters=None, bin_width=0.5, normal=False)

self.std_residuals_plot_matrix(items=None, raters=None, bin_width=0.5, normal=False)
```

Arguments

`items`	List: The names of the items to be used in the plot. If `None`, the full set of items will be used. Default is `items=None`.
`raters`	List: The names of the raters to be used in the plot.If `None`, the full set of raters will be used. Default is `raters=None`.
`bin_width`	Float: The width of the histogram bins along the x-axis. Default is `bin_width=0.5`.
`normal`	Boolean: If `True`, plots a standard normal distribution over the standardised residual histogram for comparison. Default is `normal=False`.

SInce the anchoring process does not affect residuals, there is no `anchor` argument for `std_residuals_plot`. Additional arguments to customise the appearance of the plot are detailed in Section 6.5.1.

Returns
matplotlib image. If a `filename` argument is passed, also saves the image to file.

Examples
To plot and display a basic standardised residuals histogram under the global rater representation for all items and raters, saving it to file as `my_std_residuals_plot.png`:

```
self.std_residuals_plot_global(filename='my_std_residuals_plot')
```

To plot and display a standardised residuals histogram under the threshold rater representation for all items and rater `'Rater_1'`, with bin width 1 and a standard normal curve:

```
self.std_residuals_plot_thresholds(rater='Rater_1', bin_width=1, normal=True)
```

To plot and display a standardised residuals histogram under the matrix rater representation on a subset of items stored as a list in a variable `my_item_list`, and all raters:

```
self.std_residuals_plot_matrix(items=my_item_list)
```

7 class SLM_Sim

7.1 Generating an `SLM_Sim` simulation

7.1.1 `SLM_Sim`

Description

Creates an object of the class `SLM_Sim`. Simulates response data fitting the simple logistic model (SLM) from generating parameters, which may be explicitly specified or randomly generated according to specified generating parameters.

Usage

```
SLM_Sim(no_of_items, no_of_persons, item_range=3, person_sd=1.5, offset=0, missing=0,
        manual_abilities=None, manual_diffs=None, manual_person_names=None,
        manual_item_names=None)
```

Arguments

`no_of_items`	Integer: The number of items in the simulation.
`no_of_persons`	Integer: The number of persons in the simulation.
`item_range`	Float: The range (maximum - minimum) of item difficulties for randomly generated item difficulties, which are drawn from a uniform distribution. Default is `item_range=3`. Ignored if specified item difficulties are passed using the `manual_diffs` argument.
`person_sd`	Float: The range standard deviation of person abilities for randomly generated item difficulties, which are drawn from a normal distribution. Default is `person_sd=1.5`. Ignored if specified person abilities are passed using the `manual_abilities` argument.
`offset`	Float: The difference between the means of the person abilities and item difficulties for randomly generated data. If positive, mean person ability is higher than mean item difficulty; if negative, mean person ability is lower than mean item difficulty. Default is `offset=0`.
`missing`	Float between 0 and 1: The proportion of missing data in the simulation. If `missing` is not zero, responses are removed according to a missing completely at random (MCAR) pattern, with each response having a probability of `missing` of being deleted. Since this process is randomly determined on a response-by-response basis, the proportion of missing data may not be exactly `missing`. Default is `missing=0`.

Arguments continue on the next page.

Arguments (continued)

manual_abilities	List or 1-dimensional numpy array: Optional list of specified person abilities. Allows the user to control the parameters of the simulation. manual_abilities=None, random person abilities are generated. Default is manual_abilities=None.
manual_diffs	List or 1-dimensional numpy array: Optional list of specified item difficulties. Allows the user to control the parameters of the simulation. manual_diffs=None, random item difficulties are generated. Default is manual_diffs=None.
manual_person_names	List: Optional list of person names specified by user. If manual_person_names=None, person names are generated automatically. Default is manual_person_names=None.
manual_person_names	List: Optional list of item names specified by user. If manual_item_names=None, item names are generated automatically. Default is manual_item_names=None.

Returns

Object of class SLM_Sim. Several attributes of object SLM_Sim are automatically generated on its creation:

self.scores	pandas dataframe: The dataframe of responses, which may be saved to file or used to create an SLM object for analysis.
self.no_of_items	Integer: The number of items, specified by argument no_of_items.
self.no_of_persons	Integer: The number of persons, specified by argument no_of_persons.
self.item_range	Float: The generating range of item difficulties, specified by argument item_range.
self.person_sd	The generating standard deviation of person abilities, specified by argument person_sd.
self.offset	The generating offset between the means of person ability and item difficulty, specified by argument offset.
self.missing	The generating proportion of missing data, specified by argument missing
self.abilities	pandas series: keys are person names and values are person abilities.
self.diffs	pandas series: keys are item names and values are item difficulties.
self.persons	List: Person names. If no person names are passed using the manual_person_names argument, default person names are generated in the format Person_1 etc.

Returns continue on the next page.

Returns (continued)

`self.items`	List: Item names. If no item names are passed using the `manual_item_names` argument, default item names are generated in the format `Item_1` etc.

Examples

To create an `SLM_Sim` object called `my_slm_sim` with randomly generated person abilities for 10000 persons and randomly generated item difficulties for 50 items, specifying an item range of 4 logits, a person SD of 2 logits, an offset of 1 logit and a proportion of 0.3 of missing data:

```
my_slm_sim = SLM_Sim(50, 10000, item_range=4, person_sd=2, offset=1, missing=0.3)
```

To create an `SLM_Sim` object called `my_slm_sim` with specified person abilities saved as a variable named `my_person_abils`, 50 items with specified item difficulties saved as a variable named `my_item_diffs`, 100 persons with specified person names saved as a variable named `my_person_names` and specified item names saved as a variable named `my_item_names`, with no missing data:

```
my_slm_sim = SLM_Sim(50, 100, manual_abilities=my_person_abils, manual_diffs=my_item_diffs,
                     manual_person_names=my_person_names, manual_item_names=my_item_names)
```

7.2 Customising an `SLM_Sim` simulation

7.2.1 `rename_item`

Description

Method to rename a single item.

Usage

`self.rename_item(old, new)`

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.scores` with new name.

Example

To rename an item in object `my_slm` from `Item_1` to `my_new_item_name`:

```
my_slm.rename_item('Item_1', 'my_new_item_name')
```

7.2.2 `rename_items_all`

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.scores` with new names.

Example

To rename all items in object `my_slm` with item names in a list stored as a variable `my_new_item_names`:

```
my_slm.rename_items_all(my_new_item_names)
```

7.2.3 `rename_person`

Description

Method to rename a single person.

Usage

`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns

Replaces specified person name in the index of `self.scores` with new name.

Example

To rename a person in object `my_slm` from `Person_1` to `my_new_person_name`:

```
my_slm.rename_person('Person_1', 'my_new_person_name')
```

7.2.4 `rename_persons_all`

Description Method to rename all persons.

Usage

`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns

Replaces all person names in the index of `self.scores` with new names.

Example

To rename all persons in object `my_slm` with person names in a list stored as a variable `my_new_person_names`:

```
my_slm.rename_persons_all(my_new_person_names)
```

8 class PCM_Sim

8.1 Generating a PCM_Sim simulation

8.1.1 PCM_Sim

Description

Creates an object of the class PCM_Sim. Simulates response data fitting the partial credit model (PCM) (Masters, 1982) from generating parameters, which may be explicitly specified or randomly generated according to specified generating parameters.

Usage

```
PCM_Sim(no_of_items, no_of_persons, item_range=3, person_sd=1.5, offset=0, missing=0,
        manual_abilities=None, manual_diffs=None, manual_person_names=None,
        manual_item_names=None)
```

Arguments

no_of_items	Integer: The number of items in the simulation.
no_of_persons	Integer: The number of persons in the simulation.
max_score_vector	List or 1-dimensional numpy array of integers: List with the maximum possible score for each item.
item_range	Float: The range (maximum - minimum) of item difficulties for randomly generated item difficulties, which are drawn from a uniform distribution. Default is item_range=3. Ignored if specified item difficulties are passed using the manual_diffs argument.
person_sd	Float: The range standard deviation of person abilities for randomly generated item difficulties, which are drawn from a normal distribution. Default is person_sd=1.5. Ignored if specified person abilities are passed using the manual_abilities argument.
max_disorder	Float: If a negative value is passed, disordered Rasch-Andrich thresholds (Andrich, 2010; Pallant & Tennant, 2007) may be generated. A positive value may be passed, which controls the minimum category width and limits the variation in category widths around category_base. Default is max_disorder=0, which does not permit disordered thresholds.
offset	Float: The difference between the means of the person abilities and item difficulties for randomly generated data. If positive, mean person ability is higher than mean item difficulty; if negative, mean person ability is lower than mean item difficulty. Default is offset=0.

Arguments continue on the next page.

Arguments (continued)

missing	Float between 0 and 1: The proportion of missing data in the simulation. If missing is not zero, responses are removed according to a missing completely at random (MCAR) pattern, with each response having a probability of missing of being deleted. Since this process is randomly determined on a response-by-response basis, the proportion of missing data may not be exactly missing. Default is missing=0.
manual_abilities	List or 1-dimensional numpy array: Optional list of specified person abilities. Allows the user to control the parameters of the simulation. manual_abilities=None, random person abilities are generated. Default is manual_abilities=None.
manual_diffs	List or 1-dimensional numpy array: Optional list of specified item difficulties. Allows the user to control the parameters of the simulation. manual_diffs=None, random item difficulties are generated. Default is manual_diffs=None.
manual_person_names	List: Optional list of person names specified by user. If manual_person_names=None, person names are generated automatically. Default is manual_person_names=None.
manual_person_names	List: Optional list of item names specified by user. If manual_item_names=None, item names are generated automatically. Default is manual_item_names=None.

Returns

Object of class PCM_Sim. Several attributes of object PCM_Sim are automatically generated on its creation:

self.scores	pandas dataframe: The dataframe of responses, which may be saved to file or used to create an PCM object for analysis.
self.no_of_items	Integer: The number of items, specified by argument no_of_items.
self.no_of_persons	Integer: The number of persons, specified by argument no_of_persons.
self.item_range	Float: The generating range of item difficulties, specified by argument item_range.
self.max_score_vector	List or 1-dimensional numpy array of integers: List with the maximum possible score for each item.
self.person_sd	The generating standard deviation of person abilities, specified by argument person_sd.

Returns continue on the next page.

Returns (continued)

`self.offset`	The generating offset between the means of person ability and item difficulty, specified by argument `offset`.
`self.missing`	The generating proportion of missing data, specified by argument `missing`
`self.abilities`	pandas series: keys are person names and values are person abilities.
`self.diffs`	pandas series: keys are item names and values are item difficulties.
`self.persons`	List: Person names. If no person names are passed using the `manual_person_names` argument, default person names are generated in the format `Person_1` etc.
`self.items`	List: Item names. If no item names are passed using the `manual_item_names` argument, default item names are generated in the format `Item_1` etc.

Example

To create a `PCM_Sim` object called `my_pcm_sim` with randomly generated person abilities and item difficulties, with 4 items, specifying an item range of 4 logits, maximum scores of 3, 3, 5 and 5, a person SD of 2 logits, an offset of 1 logit and a proportion of 0.3 of missing data:

```
my_pcm_sim = PCM_Sim(item_range=4, max_score_vector = [3, 3, 5, 5], person_sd=2, offset=1,
                     missing=0.3)
```

To create an `PCM_Sim` object called `my_pcm_sim` with 1,000 persons with specified person abilities saved as a variable named `my_person_abils`, 30 items with specified item difficulties saved as a variable named `my_item_diffs`, specified maximum scores saved as a variable named `my_max_score_vector`,specified person names saved as a variable named `my_person_names` and specified item names saved as a variable named `my_item_names`, with no missing data:

```
my_pcm_sim = PCM_Sim(30, 1000, max_score_vector=my_max_score_vector,
                     manual_abilities=my_person_abils, manual_diffs=my_item_diffs,
                     manual_person_names=my_person_names, manual_item_names=my_item_names)
```

8.2 Customising an `PCM_Sim` simulation

8.2.1 `rename_item`

Description

Method to rename a single item.

Usage

`self.rename_item(old, new)`

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.scores` with new name.

Example

To rename an item in object `my_slm` from `Item_1` to `my_new_item_name`:

```
my_slm.rename_item('Item_1', 'my_new_item_name')
```

8.2.2 `rename_items_all`

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.scores` with new names.

Example

To rename all items in object `my_slm` with item names in a list stored as a variable `my_new_item_names`:

```
my_slm.rename_items_all(my_new_item_names)
```

8.2.3 `rename_person`

Description

Method to rename a single person.

Usage

`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns

Replaces specified person name in the index of `self.scores` with new name.

Example

To rename a person in object `my_slm` from `Person_1` to `my_new_person_name`:

```
my_slm.rename_person('Person_1', 'my_new_person_name')
```

8.2.4 `rename_persons_all`

Description Method to rename all persons.

Usage

`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns

Replaces all person names in the index of `self.scores` with new names.

Example

To rename all persons in object `my_slm` with person names in a list stored as a variable `my_new_person_names`:

```
my_slm.rename_persons_all(my_new_person_names)
```

9 class RSM_Sim

9.1 Generating an RSM_Sim simulation

9.1.1 RSM_Sim

Description

Creates an object of the class RSM_Sim. Simulates response data fitting the rating scale model (RSM) (Andrich, 1978) from generating parameters, which may be explicitly specified or randomly generated according to specified generating parameters.

Usage

```
RSM_Sim(no_of_items, no_of_persons, max_score, item_range=3, category_mean=1, person_sd=1.5,
        max_disorder=0, offset=0, missing=0, manual_abilities=None, manual_diffs=None,
        manual_thresholds=None, manual_person_names=None, manual_item_names=None)
```

Arguments

no_of_items	Integer: The number of items in the simulation.
no_of_persons	Integer: The number of persons in the simulation.
item_range	Float: The range (maximum - minimum) of item difficulties for randomly generated item difficulties, which are drawn from a uniform distribution. Default is item_range=3. Ignored if specified item difficulties are passed using the manual_diffs argument.
category_base	Float: The base width of response categories for randomly generated Rasch-Andrich thresholds, which are drawn from a uniform distribution between max_disorder (see below) and twice category_base minus max_disorder. Default is category_base=1.
person_sd	Float: The range standard deviation of person abilities for randomly generated item difficulties, which are drawn from a normal distribution. Default is person_sd=1.5. Ignored if specified person abilities are passed using the manual_abilities argument.
max_disorder	Float: If a negative value is passed, disordered Rasch-Andrich thresholds (Andrich, 2010; Pallant & Tennant, 2007) may be generated. A positive value may be passed, which controls the minimum category width and limits the variation in category widths around category_base. Default is max_disorder=0, which does not permit disordered thresholds.

Arguments continue on the next page.

Arguments (continued)

`offset`	Float: The difference between the means of the person abilities and item difficulties for randomly generated data. If positive, mean person ability is higher than mean item difficulty; if negative, mean person ability is lower than mean item difficulty. Default is `offset=0`.
`missing`	Float between 0 and 1: The proportion of missing data in the simulation. If `missing` is not zero, responses are removed according to a missing completely at random (MCAR) pattern, with each response having a probability of `missing` of being deleted. Since this process is randomly determined on a response-by-response basis, the proportion of missing data may not be exactly `missing`. Default is `missing=0`.
`manual_abilities`	List or 1-dimensional numpy array: Optional list of specified person abilities. Allows the user to control the parameters of the simulation. `manual_abilities=None`, random person abilities are generated. Default is `manual_abilities=None`.
`manual_diffs`	List or 1-dimensional numpy array: Optional list of specified central item difficulties. Allows the user to control the parameters of the simulation. `manual_diffs=None`, random central item difficulties are generated. Default is `manual_diffs=None`.
`manual_thresholds`	List or 1-dimensional numpy array: Optional list of specified Rasch-Andrich thresholds. Allows the user to control the parameters of the simulation. `manual_diffs=None`, random Rasch-Andrich thresholds are generated. Default is `manual_thresholds=None`.
`manual_person_names`	List: Optional list of person names specified by user. If `manual_person_names=None`, person names are generated automatically. Default is `manual_person_names=None`.
`manual_person_names`	List: Optional list of item names specified by user. If `manual_item_names=None`, item names are generated automatically. Default is `manual_item_names=None`.

Returns
Object of class `RSM_Sim`. Several attributes of object `RSM_Sim` are automatically generated on its creation:

`self.scores`	pandas dataframe: The dataframe of responses, which may be saved to file or used to create an `RSM` object for analysis.
`self.no_of_items`	Integer: The number of items, specified by argument `no_of_items`.
`self.no_of_persons`	Integer: The number of persons, specified by argument `no_of_persons`.
`self.item_range`	Float: The generating range of item difficulties, specified by argument `item_range`.
`self.max_score`	Integer: The shared maximum possible score on each item, specified by argument `max_score`.
`self.category_base`	The generating base category width for generating random Rasch-Andrich thresholds, specified by argument `category_base`.
`self.person_sd`	The generating standard deviation of person abilities, specified by argument `person_sd`.
`self.offset`	The generating offset between the means of person ability and item difficulty, specified by argument `offset`.
`self.missing`	The generating proportion of missing data, specified by argument `missing`
`self.abilities`	pandas series: keys are person names and values are person abilities.
`self.diffs`	pandas series: keys are item names and values are item difficulties.
`self.thresholds`	1-dimensional numpy array: set of `self.max_score + 1`Rasch-Andrich thresholds, with the index corresponding to each threshold, and threshold 0 set to 0 by convention.
`self.persons`	List: Person names. If no person names are passed using the `manual_person_names` argument, default person names are generated in the format `Person_1` etc.
`self.items`	List: Item names. If no item names are passed using the `manual_item_names` argument, default item names are generated in the format `Item_1` etc.

Examples
To create an `RSM_Sim` object called `my_rsm_sim` with a maximum score of 5, randomly generated person abilities for 5000 persons and randomly generated item difficulties for 8 items, specifying an item range of 4 logits, a base category width of 1.5 logits, a person SD of 2 logits, a maximum permitted threshold disorder of 0.5 logits, an offset of 1 logit and a proportion of 0.3 of missing data:

```
my_rsm_sim = RSM_Sim(8, 5000, item_range=4, category_base=1.5, person_sd=2, max_disorder=0.5,
                     offset=1, missing=0.3)
```

To create an `RSM_Sim` object called `my_rsm_sim` with a maximum score of 5, specified person abilities for 100 persons saved as a variable named `my_person_abils`, specified item difficulties for 12 items saved as a variable named `my_item_diffs`, a specified set of Rasch-Andrich thresholds saved as a variable named `my_thresholds`, specified person names saved as a variable named `my_person_names` and specified item names saved as a variable named `my_item_names`, with no missing data:

```
my_rsm_sim = RSM_Sim(12, 100, manual_abilities=my_person_abils, manual_diffs=my_item_diffs,
                     manual_thresholds=my_thresholds, manual_person_names=my_person_names,
                     manual_item_names=my_item_names)
```

9.2 Customising an `RSM_Sim` simulation

9.2.1 `rename_item`

Description

Method to rename a single item.

Usage

`self.rename_item(old, new)`

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.scores` with new name.

Example

To rename an item in object `my_rsm` from `Item_1` to `my_new_item_name`:

```
my_rsm.rename_item('Item_1', 'my_new_item_name')
```

9.2.2 `rename_items_all`

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.scores` with new names.

Example

To rename all items in object `my_rsm` with item names in a list stored as a variable `my_new_item_names`:

```
my_rsm.rename_items_all(my_new_item_names)
```

9.2.3 `rename_person`

Description
Method to rename a single person.

Usage
`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns
Replaces specified person name in the index of `self.scores` with new name.

Example
To rename a person in object `my_rsm` from `Person_1` to `my_new_person_name`:

```
my_rsm.rename_person('Person_1', 'my_new_person_name')
```

9.2.4 `rename_persons_all`

Description Method to rename all persons.

Usage
`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns
Replaces all person names in the index of `self.scores` with new names.

Example
To rename all persons in object `my_rsm` with person names in a list stored as a variable `my_new_person_names`:

```
my_rsm.rename_persons_all(my_new_person_names)
```

10 class MFRM_Sim_Global

10.1 Generating an MFRM_Sim_Global simulation

10.1.1 MFRM_Sim_Global

Description

Creates an object of the class MFRM_Sim_Global. Simulates response data fitting the rating scale model formulation of the many-facet Rasch model (MFRM) under the global rater representation (Linacre, 1994) from generating parameters, which may be explicitly specified or randomly generated according to specified generating parameters.

Usage

```
MFRM_Sim_Global(no_of_items, no_of_persons, no_of_raters, max_score, item_range=2,
                rater_range=2, category_base=1, person_sd=1.5, max_disorder=0, offset=0,
                missing=0, manual_abilities=None, manual_diffs=None,
                manual_thresholds=None, manual_severities=None, manual_person_names=None,
                manual_item_names=None, manual_rater_names=None)
```

Arguments

no_of_items	Integer: The number of items in the simulation.
no_of_persons	Integer: The number of persons in the simulation.
no_of_raters	Integer: The number of raters in the simulation.
item_range	Float: The range (maximum - minimum) of item difficulties for randomly generated item difficulties, which are drawn from a uniform distribution. Default is item_range=3. Ignored if specified item difficulties are passed using the manual_diffs argument.
rater_range	Float: The range (maximum - minimum) of rater severities for randomly generated rater severities, which are drawn from a uniform distribution. Default is rater_range=3. Ignored if specified rater severities are passed using the rater_diffs argument.
category_base	Float: The base width of response categories for randomly generated Rasch-Andrich thresholds, which are drawn from a uniform distribution between max_disorder (see below) and twice category_base minus max_disorder. Default is category_base=1.

Arguments continue on the next page.

Arguments (continued)

`person_sd`	Float: The range standard deviation of person abilities for randomly generated item difficulties, which are drawn from a normal distribution. Default is `person_sd=1.5`. Ignored if specified person abilities are passed using the `manual_abilities` argument.
`max_disorder`	Float: If a negative value is passed, disordered Rasch-Andrich thresholds (Andrich, 2010; Pallant & Tennant, 2007) may be generated. A positive value may be passed, which controls the minimum category width and limits the variation in category widths around `category_base`. Default is `max_disorder=0`, which does not permit disordered thresholds.
`offset`	Float: The difference between the means of the person abilities and item difficulties for randomly generated data. If positive, mean person ability is higher than mean item difficulty; if negative, mean person ability is lower than mean item difficulty. Default is `offset=0`.
`missing`	Float between 0 and 1: The proportion of missing data in the simulation. If `missing` is not zero, responses are removed according to a missing completely at random (MCAR) pattern, with each response having a probability of `missing` of being deleted. Since this process is randomly determined on a response-by-response basis, the proportion of missing data may not be exactly `missing`. Default is `missing=0`.
`manual_abilities`	List or 1-dimensional numpy array: Optional list of specified person abilities. Allows the user to control the parameters of the simulation. `manual_abilities=None`, random person abilities are generated. Default is `manual_abilities=None`.
`manual_diffs`	List or 1-dimensional numpy array: Optional list of specified central item difficulties. Allows the user to control the parameters of the simulation. `manual_diffs=None`, random central item difficulties are generated. Default is `manual_diffs=None`.
`manual_thresholds`	List or 1-dimensional numpy array: Optional list of specified Rasch-Andrich thresholds. Allows the user to control the parameters of the simulation. `manual_diffs=None`, random Rasch-Andrich thresholds are generated. Default is `manual_thresholds=None`.
`manual_severities`	List or 1-dimensional numpy array: Optional list of specified rater severities. Allows the user to control the parameters of the simulation. `manual_severities=None`, random rater severities are generated. Default is `manual_severities=None`.

Arguments continue on the next page.

Arguments (continued)

`manual_person_names`	List: Optional list of person names specified by user. If `manual_person_names=None`, person names are generated automatically. Default is `manual_person_names=None`.
`manual_item_names`	List: Optional list of item names specified by user. If `manual_item_names=None`, item names are generated automatically. Default is `manual_item_names=None`.
`manual_rater_names`	List: Optional list of item names specified by user. If `manual_rater_names=None`, item names are generated automatically. Default is `manual_rater_names=None`.

Format for `manual_severities` input

Toy example of the input for `manual_severities`, with two items and two persons. Pandas series with rater names as keys and severities as values.

```
manual_severities = pd.Series({'Rater_1':  -0.5,
                               'Rater_2':   0.5})
```

Returns

Object of class `MFRM_Sim_Global`. Several attributes of object `MFRM_Sim_Global` are automatically generated on its creation:

`self.scores`	pandas dataframe: The dataframe of responses, which may be saved to file or used to create an `RSM` object for analysis.
`self.no_of_items`	Integer: The number of items, specified by argument `no_of_items`.
`self.no_of_persons`	Integer: The number of persons, specified by argument `no_of_persons`.
`self.no_of_raters`	Integer: The number of raters, specified by argument `no_of_raters`.
`self.item_range`	Float: The generating range of item difficulties, specified by argument `item_range`.
`self.rater_range`	Float: The generating range of rater severities, specified by argument `rater_range`.
`self.max_score`	Integer: The shared maximum possible score on each item, specified by argument `max_score`.
`self.category_base`	The generating base category width for generating random Rasch-Andrich thresholds, specified by argument `category_base`.

Returns continue on the next page.

Returns (continued)

`self.person_sd`	The generating standard deviation of person abilities, specified by argument `person_sd`.
`self.offset`	The generating offset between the means of person ability and item difficulty, specified by argument `offset`.
`self.missing`	The generating proportion of missing data, specified by argument `missing`
`self.abilities`	pandas series: keys are person names and values are person abilities.
`self.diffs`	pandas series: keys are item names and values are item difficulties.
`self.severities`	pandas series: keys are rater names and values are rater severities.
`self.thresholds`	1-dimensional numpy array: set of `self.max_score + 1`Rasch-Andrich thresholds, with the index corresponding to each threshold, and threshold 0 set to 0 by convention.
`self.persons`	List: Person names. If no person names are passed using the `manual_person_names` argument, default person names are generated in the format `Person_1` etc.
`self.items`	List: Item names. If no item names are passed using the `manual_item_names` argument, default item names are generated in the format `Item_1` etc.
`self.raters`	List: Rater names. If no item names are passed using the `manual_rater_names` argument, default item names are generated in the format `Rater_1` etc.

Example

To create an `MFRM_Sim_Global` object called `my_mfrm_sim` with a maximum score of 5, randomly generated person abilities for 5,000 persons, randomly generated item difficulties for 10 items and randomly generated rater severities for 12 raters, specifying an item range of 4 logits, a rater range of 3 logits, a base category width of 1.5 logits, a person SD of 2 logits, a maximum permitted threshold disorder of 0.5 logits, an offset of 1 logit and a proportion of 0.3 of missing data:

```
my_mfrm_sim = MFRM_Sim_Global(10, 5000, 12, 5, item_range=4, rater_range=3,
                              category_base=1.5, person_sd=2,
                              max_disorder=0.5, offset=1, missing=0.3)
```

To create an `MFRM_Sim_Global` object called `my_mfrm_sim` with a maximum score of 5, specified person abilities for 100 persons saved as a variable named `my_person_abils`, specified item difficulties for 12 items saved as a variable named `my_item_diffs`, specified rater severities for 6 raters saved as a variable named `my_severities`, a specified set of Rasch-Andrich thresholds saved as a variable named `my_thresholds`, specified person names saved as a variable named `my_person_names`, specified item names saved as a variable named `my_item_names` and specified rater names saved as a variable named `my_rater_names`, with no missing data:

```
my_mfrm_sim = MFRM_Sim_Global(12, 100, 6, 5, manual_abilities=my_person_abils,
                              manual_diffs=my_item_diffs,
                              manual_severities=my_severities,
                              manual_thresholds=my_thresholds,
                              manual_person_names=my_person_names,
                              manual_item_names=my_item_names,
                              manual_rater_names=my_rater_names)
```

10.2 Customising an `MFRM_Sim_Global` simulation

10.2.1 `rename_item`

Description

Method to rename a single item.

Usage

`self.rename_item(old, new)`

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.scores` with new name.

Example

To rename an item in object `my_mfrm` from `Item_1` to `my_new_item_name`:

```
my_mfrm.rename_item('Item_1', 'my_new_item_name')
```

10.2.2 `rename_items_all`

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.scores` with new names.

Example

To rename all items in object `my_mfrm` with item names in a list stored as a variable `my_new_item_names`:

```
my_mfrm.rename_items_all(my_new_item_names)
```

10.2.3 `rename_person`

Description
Method to rename a single person.

Usage
`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns
Replaces specified person name in the second multiindex level of `self.scores` with new name.

Example
To rename a person in object `my_mfrm` from `Person_1` to `my_new_person_name`:

```
my_mfrm.rename_person('Person_1', 'my_new_person_name')
```

10.2.4 `rename_persons_all`

Description Method to rename all persons.

Usage
`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns
Replaces all person names in the second multiindex level of `self.scores` with new names.

Example
To rename all persons in object `my_mfrm` with person names in a list stored as a variable `my_new_person_names`:

```
my_mfrm.rename_persons_all(my_new_person_names)
```

10.2.5 `rename_rater`

Description

Method to rename a single rater.

Usage

`self.rename_rater(old, new)`

Arguments

`old`	String: the old name for the rater
`new`	String: the new name for the rater

Returns

Replaces specified rater name in the first multiindex level of `self.scores` with new name.

Example

To rename a rater in object `my_mfrm` from `Rater_1` to `my_new_rater_name`:

```
my_mfrm.rename_rater('Rater_1', 'my_new_rater_name')
```

10.2.6 `rename_raters_all`

Description Method to rename all raters.

Usage

`self.rename_raters_all(new_names)`

Arguments

`new_names`	List of new rater names as strings

Returns

Replaces all rater names in the first multiindex level of `self.scores` with new names.

Example

To rename all raters in object `my_mfrm` with rater names in a list stored as a variable `my_new_rater_names`:

```
my_mfrm.rename_raters_all(my_new_rater_names)
```

11 class MFRM_Sim_Items

11.1 Generating an MFRM_Sim_Items simulation

11.1.1 MFRM_Sim_Items

Description

Creates an object of the class MFRM_Sim_Items. Simulates response data fitting the rating scale model formulation of the many-facet Rasch model (MFRM) under the vector-by-item rater representation (Elliott & Buttery, 2022a) from generating parameters, which may be explicitly specified or randomly generated according to specified generating parameters.

Usage

```
MFRM_Sim_Items(no_of_items, no_of_persons, no_of_raters, max_score, item_range=2,
            rater_range=2, category_mean=1, person_sd=1.5, max_disorder=0, offset=0,
            missing=0, manual_abilities=None, manual_diffs=None, manual_thresholds=None,
            manual_severities=None, manual_person_names=None, manual_item_names=None,
            manual_rater_names=None)
```

Arguments

no_of_items	Integer: The number of items in the simulation.
no_of_persons	Integer: The number of persons in the simulation.
no_of_raters	Integer: The number of raters in the simulation.
item_range	Float: The range (maximum - minimum) of item difficulties for randomly generated item difficulties, which are drawn from a uniform distribution. Default is item_range=3. Ignored if specified item difficulties are passed using the manual_diffs argument.
rater_range	Float: The range (maximum - minimum) of rater severities for randomly generated rater severities, which are drawn from a uniform distribution. Default is rater_range=3. Ignored if specified rater severities are passed using the rater_diffs argument.
category_base	Float: The base width of response categories for randomly generated Rasch-Andrich thresholds, which are drawn from a uniform distribution between max_disorder (see below) and twice category_base minus max_disorder. Default is category_base=1.

Arguments continue on the next page.

Arguments (continued)

person_sd	Float: The range standard deviation of person abilities for randomly generated item difficulties, which are drawn from a normal distribution. Default is person_sd=1.5. Ignored if specified person abilities are passed using the manual_abilities argument.
max_disorder	Float: If a negative value is passed, disordered Rasch-Andrich thresholds (Andrich, 2010; Pallant & Tennant, 2007) may be generated. A positive value may be passed, which controls the minimum category width and limits the variation in category widths around category_base. Default is max_disorder=0, which does not permit disordered thresholds.
offset	Float: The difference between the means of the person abilities and item difficulties for randomly generated data. If positive, mean person ability is higher than mean item difficulty; if negative, mean person ability is lower than mean item difficulty. Default is offset=0.
missing	Float between 0 and 1: The proportion of missing data in the simulation. If missing is not zero, responses are removed according to a missing completely at random (MCAR) pattern, with each response having a probability of missing of being deleted. Since this process is randomly determined on a response-by-response basis, the proportion of missing data may not be exactly missing. Default is missing=0.
manual_abilities	List or 1-dimensional numpy array: Optional list of specified person abilities. Allows the user to control the parameters of the simulation. manual_abilities=None, random person abilities are generated. Default is manual_abilities=None.
manual_diffs	List or 1-dimensional numpy array: Optional list of specified central item difficulties. Allows the user to control the parameters of the simulation. manual_diffs=None, random central item difficulties are generated. Default is manual_diffs=None.
manual_thresholds	List or 1-dimensional numpy array: Optional list of specified Rasch-Andrich thresholds. Allows the user to control the parameters of the simulation. manual_diffs=None, random Rasch-Andrich thresholds are generated. Default is manual_thresholds=None.
manual_severities	Nested dictionary: Optional list of specified rater severities. Allows the user to control the parameters of the simulation. manual_severities=None, random rater severities are generated. Default is manual_severities=None. Details of the format for manual_severities are given below.

Arguments continue on the next page.

Arguments (continued)

manual_person_names	List: Optional list of person names specified by user. If manual_person_names=None, person names are generated automatically. Default is manual_person_names=None.
manual_item_names	List: Optional list of item names specified by user. If manual_item_names=None, item names are generated automatically. Default is manual_item_names=None.
manual_rater_names	List: Optional list of item names specified by user. If manual_rater_names=None, item names are generated automatically. Default is manual_rater_names=None.

Format for manual_severities input

Toy example of the nested dictionary input for manual_severities, with two items and two persons. The outer dictionary has item names for values, which are also the keys for the inner dictionary. The inner dictionaries item names for keys and floats (item severities) for values.

```
manual_severities = {'Rater_1': {'Item_1': -1, 'Item_2': 0},
                     'Rater_2': {'Item_1':  0, 'Item_2': 1}}
```

Returns

Object of class MFRM_Sim_Global. Several attributes of object MFRM_Sim_Global are automatically generated on its creation:

self.scores	pandas dataframe: The dataframe of responses, which may be saved to file or used to create an RSM object for analysis.
self.no_of_items	Integer: The number of items, specified by argument no_of_items.
self.no_of_persons	Integer: The number of persons, specified by argument no_of_persons.
self.no_of_raters	Integer: The number of raters, specified by argument no_of_raters.
self.item_range	Float: The generating range of item difficulties, specified by argument item_range.
self.rater_range	Float: The generating range of rater severities, specified by argument rater_range.
self.max_score	Integer: The shared maximum possible score on each item, specified by argument max_score.
self.category_base	The generating base category width for generating random Rasch-Andrich thresholds, specified by argument category_base.

Returns continue on the next page.

Returns (continued)

`self.person_sd`	The generating standard deviation of person abilities, specified by argument `person_sd`.
`self.offset`	The generating offset between the means of person ability and item difficulty, specified by argument `offset`.
`self.missing`	The generating proportion of missing data, specified by argument `missing`
`self.abilities`	pandas series: keys are person names and values are person abilities.
`self.diffs`	pandas series: keys are item names and values are item difficulties.
`self.severities`	Nested dictionary: Outer keys are rater names, inner keys are item names, as in the `manual_severities` input example given above.
`self.thresholds`	1-dimensional numpy array: set of `self.max_score + 1`Rasch-Andrich thresholds, with the index corresponding to each threshold, and threshold 0 set to 0 by convention.
`self.persons`	List: Person names. If no person names are passed using the `manual_person_names` argument, default person names are generated in the format `Person_1` etc.
`self.items`	List: Item names. If no item names are passed using the `manual_item_names` argument, default item names are generated in the format `Item_1` etc.
`self.raters`	List: Rater names. If no item names are passed using the `manual_rater_names` argument, default item names are generated in the format `Rater_1` etc.

Examples

To create an `MFRM_Sim_Items` object called `my_mfrm_sim` with a maximum score of 5, randomly generated person abilities for 5,000 persons, randomly generated item difficulties for 10 items and randomly generated rater severities for 12 raters, specifying an item range of 4 logits, a rater range of 3 logits, a base category width of 1.5 logits, a person SD of 2 logits, a maximum permitted threshold disorder of 0.5 logits, an offset of 1 logit and a proportion of 0.3 of missing data:

```
my_mfrm_sim = MFRM_Sim_Items(10, 5000, 12, 5, item_range=4, rater_range=3,
                             category_base=1.5, person_sd=2, max_disorder=0.5,
                             offset=1, missing=0.3)
```

To create an `MFRM_Sim_Items` object called `my_mfrm_sim` with a maximum score of 5, specified person abilities for 100 persons saved as a variable named `my_person_abils`, specified item difficulties for 12 items saved as a variable named `my_item_diffs`, specified rater severities for 6 raters saved as a variable named `my_severities`, a specified set of Rasch-Andrich thresholds saved as a variable named `my_thresholds`, specified person names saved as a variable named `my_person_names`, specified item names saved as a variable named `my_item_names` and specified rater names saved as a variable named `my_rater_names`, with no missing data:

```
my_mfrm_sim = MFRM_Sim_Items(12, 100, 6, 5, manual_abilities=my_person_abils,
                             manual_diffs=my_item_diffs,
                             manual_severities=my_severities,
                             manual_thresholds=my_thresholds,
                             manual_person_names=my_person_names,
                             manual_item_names=my_item_names,
                             manual_rater_names=my_rater_names)
```

11.2 Customising an MFRM_Sim_Items simulation

11.2.1 rename_item

Description

Method to rename a single item.

Usage

`self.rename_item(old, new)`

Arguments

old	String: the old name for the item
new	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.scores` with new name.

Example

To rename an item in object `my_mfrm` from `Item_1` to `my_new_item_name`:

```
my_mfrm.rename_item('Item_1', 'my_new_item_name')
```

11.2.2 rename_items_all

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

new_names	List of new item names as strings

Returns

Replaces all item names in the columns of `self.scores` with new names.

Example

To rename all items in object `my_mfrm` with item names in a list stored as a variable `my_new_item_names`:

```
my_mfrm.rename_items_all(my_new_item_names)
```

11.2.3 `rename_person`

Description

Method to rename a single person.

Usage

`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns

Replaces specified person name in the second multiindex level of `self.scores` with new name.

Example

To rename a person in object `my_mfrm` from `Person_1` to `my_new_person_name`:

```
my_mfrm.rename_person('Person_1', 'my_new_person_name')
```

11.2.4 `rename_persons_all`

Description Method to rename all persons.

Usage

`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns

Replaces all person names in the second multiindex level of `self.scores` with new names.

Example

To rename all persons in object `my_mfrm` with person names in a list stored as a variable `my_new_person_names`:

```
my_mfrm.rename_persons_all(my_new_person_names)
```

11.2.5 `rename_rater`

Description

Method to rename a single rater.

Usage

`self.rename_rater(old, new)`

Arguments

`old`	String: the old name for the rater
`new`	String: the new name for the rater

Returns

Replaces specified rater name in the first multiindex level of `self.scores` with new name.

Example

To rename a rater in object `my_mfrm` from `Rater_1` to `my_new_rater_name`:

```
my_mfrm.rename_rater('Rater_1', 'my_new_rater_name')
```

11.2.6 `rename_raters_all`

Description Method to rename all raters.

Usage

`self.rename_raters_all(new_names)`

Arguments

`new_names`	List of new rater names as strings

Returns

Replaces all rater names in the first multiindex level of `self.scores` with new names.

Example

To rename all raters in object `my_mfrm` with rater names in a list stored as a variable `my_new_rater_names`:

```
my_mfrm.rename_raters_all(my_new_rater_names)
```

12 class MFRM_Sim_Thresholds

12.1 Generating an `MFRM_Sim_Thresholds` simulation

12.1.1 `MFRM_Sim_Thresholds`

Description

Creates an object of the class `MFRM_Sim_Thresholds`. Simulates response data fitting the rating scale model formulation of the many-facet Rasch model (MFRM) under the vector-by-threshold rater representation (Elliott & Buttery, 2022a) from generating parameters, which may be explicitly specified or randomly generated according to specified generating parameters.

Usage

```
MFRM_Sim_Thresholds(no_of_items, no_of_persons, no_of_raters, max_score, item_range=2,
                    rater_range=2, category_base=1, person_sd=1.5, max_disorder=0,
                    offset=0, missing=0, manual_abilities=None, manual_diffs=None,
                    manual_thresholds=None, manual_severities=None,
                    manual_person_names=None, manual_item_names=None,
                    manual_rater_names=None)
```

Arguments

`no_of_items`	Integer: The number of items in the simulation.
`no_of_persons`	Integer: The number of persons in the simulation.
`no_of_raters`	Integer: The number of raters in the simulation.
`item_range`	Float: The range (maximum - minimum) of item difficulties for randomly generated item difficulties, which are drawn from a uniform distribution. Default is `item_range=3`. Ignored if specified item difficulties are passed using the `manual_diffs` argument.
`rater_range`	Float: The range (maximum - minimum) of rater severities for randomly generated rater severities, which are drawn from a uniform distribution. Default is `rater_range=3`. Ignored if specified rater severities are passed using the `rater_diffs` argument.
`category_base`	Float: The base width of response categories for randomly generated Rasch-Andrich thresholds, which are drawn from a uniform distribution between `max_disorder` (see below) and twice `category_base` minus `max_disorder`. Default is `category_base=1`.

Arguments continue on the next page.

Arguments (continued)

`person_sd`	Float: The range standard deviation of person abilities for randomly generated item difficulties, which are drawn from a normal distribution. Default is `person_sd=1.5`. Ignored if specified person abilities are passed using the `manual_abilities` argument.
`max_disorder`	Float: If a negative value is passed, disordered Rasch-Andrich thresholds (Andrich, 2010; Pallant & Tennant, 2007) may be generated. A positive value may be passed, which controls the minimum category width and limits the variation in category widths around `category_base`. Default is `max_disorder=0`, which does not permit disordered thresholds.
`offset`	Float: The difference between the means of the person abilities and item difficulties for randomly generated data. If positive, mean person ability is higher than mean item difficulty; if negative, mean person ability is lower than mean item difficulty. Default is `offset=0`.
`missing`	Float between 0 and 1: The proportion of missing data in the simulation. If `missing` is not zero, responses are removed according to a missing completely at random (MCAR) pattern, with each response having a probability of `missing` of being deleted. Since this process is randomly determined on a response-by-response basis, the proportion of missing data may not be exactly `missing`. Default is `missing=0`.
`manual_abilities`	List or 1-dimensional numpy array: Optional list of specified person abilities. Allows the user to control the parameters of the simulation. `manual_abilities=None`, random person abilities are generated. Default is `manual_abilities=None`.
`manual_diffs`	List or 1-dimensional numpy array: Optional list of specified central item difficulties. Allows the user to control the parameters of the simulation. `manual_diffs=None`, random central item difficulties are generated. Default is `manual_diffs=None`.
`manual_thresholds`	List or 1-dimensional numpy array: Optional list of specified Rasch-Andrich thresholds. Allows the user to control the parameters of the simulation. `manual_diffs=None`, random Rasch-Andrich thresholds are generated. Default is `manual_thresholds=None`.
`manual_severities`	Dictionary of 1-dimensional numpy arrays: Optional list of specified rater severities. Allows the user to control the parameters of the simulation. `manual_severities=None`, random rater severities are generated. Default is `manual_severities=None`. Details of the format for `manual_severities` are given below.

Arguments continue on the next page.

Arguments (continued)

`manual_person_names`	List: Optional list of person names specified by user. If `manual_person_names=None`, person names are generated automatically. Default is `manual_person_names=None`.
`manual_item_names`	List: Optional list of item names specified by user. If `manual_item_names=None`, item names are generated automatically. Default is `manual_item_names=None`.
`manual_rater_names`	List: Optional list of item names specified by user. If `manual_rater_names=None`, item names are generated automatically. Default is `manual_rater_names=None`.

Format for `manual_severities` input

Toy example of the dictionary input for `manual_severities`, with two items, two persons and four categories. Dictionary with item names as keys and numpy arrays as values, of length `self.max_score + 1` of item severities for values with the indices of each array corresponding to the threshold, and threshold zero set to zero by convention.

```
manual_severities = {'Rater_1':  np.array([0,  -1, -0.5,    0]),
                     'Rater_2':  np.array([0, 1.5,  0.5, -0.5])}
```

Returns

Object of class `MFRM_Sim_Thresholds`. Several attributes of object `MFRM_Sim_Thresholds` are automatically generated on its creation:

`self.scores`	pandas dataframe: The dataframe of responses, which may be saved to file or used to create an `RSM` object for analysis.
`self.no_of_items`	Integer: The number of items, specified by argument `no_of_items`.
`self.no_of_persons`	Integer: The number of persons, specified by argument `no_of_persons`.
`self.no_of_raters`	Integer: The number of raters, specified by argument `no_of_raters`.
`self.item_range`	Float: The generating range of item difficulties, specified by argument `item_range`.
`self.rater_range`	Float: The generating range of rater severities, specified by argument `rater_range`.
`self.max_score`	Integer: The shared maximum possible score on each item, specified by argument `max_score`.
`self.category_base`	The generating base category width for generating random Rasch-Andrich thresholds, specified by argument `category_base`.

Returns continue on the next page.

Returns (continued)

`self.person_sd`	The generating standard deviation of person abilities, specified by argument `person_sd`.
`self.offset`	The generating offset between the means of person ability and item difficulty, specified by argument `offset`.
`self.missing`	The generating proportion of missing data, specified by argument `missing`
`self.abilities`	pandas series: keys are person names and values are person abilities.
`self.diffs`	pandas series: keys are item names and values are item difficulties.
`self.severities`	Dictionary: Keys are rater names, values are numpy arrays, as in the `manual_severities` input example given above.
`self.thresholds`	1-dimensional numpy array: set of `self.max_score + 1`Rasch-Andrich thresholds, with the index corresponding to each threshold, and threshold 0 set to 0 by convention.
`self.persons`	List: Person names. If no person names are passed using the `manual_person_names` argument, default person names are generated in the format `Person_1` etc.
`self.items`	List: Item names. If no item names are passed using the `manual_item_names` argument, default item names are generated in the format `Item_1` etc.
`self.raters`	List: Rater names. If no item names are passed using the `manual_rater_names` argument, default item names are generated in the format `Rater_1` etc.

Example

To create an `MFRM_Sim_Thresholds` object called `my_mfrm_sim` with a maximum score of 5, randomly generated person abilities for 5,000 persons, randomly generated item difficulties for 10 items and randomly generated rater severities for 12 raters, specifying an item range of 4 logits, a rater range of 3 logits, a base category width of 1.5 logits, a person SD of 2 logits, a maximum permitted threshold disorder of 0.5 logits, an offset of 1 logit and a proportion of 0.3 of missing data:

```
my_mfrm_sim = MFRM_Sim_Thresholds(10, 5000, 12, 5, item_range=4,
                                  category_base=1.5, person_sd=2,
                                  max_disorder=0.5, offset=1, missing=0.3)
```

To create an MFRM_Sim_Thresholds object called my_mfrm_sim with a maximum score of 5, specified person abilities for 100 persons saved as a variable named my_person_abils, specified item difficulties for 12 items saved as a variable named my_item_diffs, specified rater severities for 6 raters saved as a variable named my_severities, a specified set of Rasch-Andrich thresholds saved as a variable named my_thresholds, specified person names saved as a variable named my_person_names, specified item names saved as a variable named my_item_names and specified rater names saved as a variable named my_rater_names, with no missing data:

```
my_mfrm_sim = MFRM_Sim_Thresholds(12, 100, 6, 5, manual_abilities=my_person_abils,
                                  manual_diffs=my_item_diffs,
                                  manual_severities=my_severities,
                                  manual_thresholds=my_thresholds,
                                  manual_person_names=my_person_names,
                                  manual_item_names=my_item_names,
                                  manual_rater_names=my_rater_names)
```

12.2 Customising an `MFRM_Sim_Thresholds` simulation

12.2.1 `rename_item`

Description

Method to rename a single item.

Usage

`self.rename_item(old, new)`

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.scores` with new name.

Example

To rename an item in object `my_mfrm` from `Item_1` to `my_new_item_name`:

```
my_mfrm.rename_item('Item_1', 'my_new_item_name')
```

12.2.2 `rename_items_all`

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.scores` with new names.

Example

To rename all items in object `my_mfrm` with item names in a list stored as a variable `my_new_item_names`:

```
my_mfrm.rename_items_all(my_new_item_names)
```

12.2.3 `rename_person`

Description

Method to rename a single person.

Usage

`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns

Replaces specified person name in the second multiindex level of `self.scores` with new name.

Example

To rename a person in object `my_mfrm` from `Person_1` to `my_new_person_name`:

```
my_mfrm.rename_person('Person_1', 'my_new_person_name')
```

12.2.4 `rename_persons_all`

Description Method to rename all persons.

Usage

`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns

Replaces all person names in the second multiindex level of `self.scores` with new names.

Example

To rename all persons in object `my_mfrm` with person names in a list stored as a variable `my_new_person_names`:

```
my_mfrm.rename_persons_all(my_new_person_names)
```

12.2.5 `rename_rater`

Description

Method to rename a single rater.

Usage

`self.rename_rater(old, new)`

Arguments

`old`	String: the old name for the rater
`new`	String: the new name for the rater

Returns

Replaces specified rater name in the first multiindex level of `self.scores` with new name.

Example

To rename a rater in object `my_mfrm` from `Rater_1` to `my_new_rater_name`:

```
my_mfrm.rename_rater('Rater_1', 'my_new_rater_name')
```

12.2.6 `rename_raters_all`

Description Method to rename all raters.

Usage

`self.rename_raters_all(new_names)`

Arguments

`new_names`	List of new rater names as strings

Returns

Replaces all rater names in the first multiindex level of `self.scores` with new names.

Example

To rename all raters in object `my_mfrm` with rater names in a list stored as a variable `my_new_rater_names`:

```
my_rsm.rename_raters_all(my_new_rater_names)
```

13 class MFRM_Sim_Matrix

13.1 Generating an `MFRM_Sim_Matrix` simulation

13.1.1 `MFRM_Sim_Matrix`

Description

Creates an object of the class `MFRM_Sim_Matrix`. Simulates response data fitting the rating scale model formulation of the many-facet Rasch model (MFRM) under the matrix representation (Elliott & Buttery, 2022a) from generating parameters, which may be explicitly specified or randomly generated according to specified generating parameters.

Usage

```
MFRM_Sim_Matrix(no_of_items, no_of_persons, no_of_raters, max_score, item_range=2
                rater_range=2, category_mean=1, person_sd=1.5, max_disorder=0,
                offset=0, missing=0, manual_abilities=None, manual_diffs=None,
                manual_thresholds=None, manual_severities=None)
```

Arguments

`no_of_items`	Integer: The number of items in the simulation.
`no_of_persons`	Integer: The number of persons in the simulation.
`no_of_raters`	Integer: The number of raters in the simulation.
`item_range`	Float: The range (maximum - minimum) of item difficulties for randomly generated item difficulties, which are drawn from a uniform distribution. Default is `item_range=3`. Ignored if specified item difficulties are passed using the `manual_diffs` argument.
`rater_range`	Float: The range (maximum - minimum) of rater severities for randomly generated rater severities, which are drawn from a uniform distribution. Default is `rater_range=3`. Ignored if specified rater severities are passed using the `rater_diffs` argument.
`category_base`	Float: The base width of response categories for randomly generated Rasch-Andrich thresholds, which are drawn from a uniform distribution between `max_disorder` (see below) and twice `category_base` minus `max_disorder`. Default is `category_base=1`.

Arguments continue on the next page.

Arguments (continued)

person_sd	Float: The range standard deviation of person abilities for randomly generated item difficulties, which are drawn from a normal distribution. Default is person_sd=1.5. Ignored if specified person abilities are passed using the manual_abilities argument.
max_disorder	Float: If a negative value is passed, disordered Rasch-Andrich thresholds (Andrich, 2010; Pallant & Tennant, 2007) may be generated. A positive value may be passed, which controls the minimum category width and limits the variation in category widths around category_base. Default is max_disorder=0, which does not permit disordered thresholds.
offset	Float: The difference between the means of the person abilities and item difficulties for randomly generated data. If positive, mean person ability is higher than mean item difficulty; if negative, mean person ability is lower than mean item difficulty. Default is offset=0.
missing	Float between 0 and 1: The proportion of missing data in the simulation. If missing is not zero, responses are removed according to a missing completely at random (MCAR) pattern, with each response having a probability of missing of being deleted. Since this process is randomly determined on a response-by-response basis, the proportion of missing data may not be exactly missing. Default is missing=0.
manual_abilities	List or 1-dimensional numpy array: Optional list of specified person abilities. Allows the user to control the parameters of the simulation. manual_abilities=None, random person abilities are generated. Default is manual_abilities=None.
manual_diffs	List or 1-dimensional numpy array: Optional list of specified central item difficulties. Allows the user to control the parameters of the simulation. manual_diffs=None, random central item difficulties are generated. Default is manual_diffs=None.
manual_thresholds	List or 1-dimensional numpy array: Optional list of specified Rasch-Andrich thresholds. Allows the user to control the parameters of the simulation. manual_diffs=None, random Rasch-Andrich thresholds are generated. Default is manual_thresholds=None.
manual_severities	Nested dictionary: Optional list of specified rater severities. Allows the user to control the parameters of the simulation. manual_severities=None, random rater severities are generated. Default is manual_severities=None. Details of the format for manual_severities are given below.

Arguments continue on the next page.

Arguments (continued)

`manual_person_names`	List: Optional list of person names specified by user. If `manual_person_names=None`, person names are generated automatically. Default is `manual_person_names=None`.
`manual_item_names`	List: Optional list of item names specified by user. If `manual_item_names=None`, item names are generated automatically. Default is `manual_item_names=None`.
`manual_rater_names`	List: Optional list of item names specified by user. If `manual_rater_names=None`, item names are generated automatically. Default is `manual_rater_names=None`.

Format for `manual_severities` input

Toy example of the nested dictionary input for `manual_severities`, with two items, two persons and four categories. The outer dictionary has item names for values, which are also the keys for the inner dictionary. Values for the inner dictionary are numpy arrays of severities by threshold for the item key, with the index of the array index corresponding to each threshold (0 to `self.max_score`).

```
manual_severities = {'Rater_1':  {'Item_1':  np.array([0, -1.5, -1, 0.5]),
                                  'Item_2':  np.array([0,    1,  0,  -1])},
                     'Rater_2':  {'Item_1':  np.array([0, -0.5,  0, 0.5]),
                                  'Item_2':  np.array([0,    2,  1,   0])}}
```

Returns

Object of class `MFRM_Sim_Global`. Several attributes of object `MFRM_Sim_Global` are automatically generated on its creation:

`self.scores`	pandas dataframe: The dataframe of responses, which may be saved to file or used to create an `RSM` object for analysis.
`self.no_of_items`	Integer: The number of items, specified by argument `no_of_items`.
`self.no_of_persons`	Integer: The number of persons, specified by argument `no_of_persons`.
`self.no_of_raters`	Integer: The number of raters, specified by argument `no_of_raters`.
`self.item_range`	Float: The generating range of item difficulties, specified by argument `item_range`.
`self.rater_range`	Float: The generating range of rater severities, specified by argument `rater_range`.
`self.max_score`	Integer: The shared maximum possible score on each item, specified by argument `max_score`.

Returns continue on the next page.

Returns (continued)

`self.category_base`	The generating base category width for generating random Rasch-Andrich thresholds, specified by argument `category_base`.
`self.person_sd`	The generating standard deviation of person abilities, specified by argument `person_sd`.
`self.offset`	The generating offset between the means of person ability and item difficulty, specified by argument `offset`.
`self.missing`	The generating proportion of missing data, specified by argument `missing`
`self.abilities`	pandas series: keys are person names and values are person abilities.
`self.diffs`	pandas series: keys are item names and values are item difficulties.
`self.severities`	Nested dictionary: Outer keys are rater names, inner keys are item names, as in the `manual_severities` input example given above.
`self.thresholds`	1-dimensional numpy array: set of `self.max_score + 1`Rasch-Andrich thresholds, with the index corresponding to each threshold, and threshold 0 set to 0 by convention.
`self.persons`	List: Person names. If no person names are passed using the `manual_person_names` argument, default person names are generated in the format `Person_1` etc.
`self.items`	List: Item names. If no item names are passed using the `manual_item_names` argument, default item names are generated in the format `Item_1` etc.
`self.raters`	List: Rater names. If no item names are passed using the `manual_rater_names` argument, default item names are generated in the format `Rater_1` etc.

Example

To create an `MFRM_Sim_Matrix` object called `my_mfrm_sim` with a maximum score of 5, randomly generated person abilities for 5,000 persons, randomly generated item difficulties for 10 items and randomly generated rater severities for 12 raters, specifying an item range of 4 logits, a rater range of 3 logits, a base category width of 1.5 logits, a person SD of 2 logits, a maximum permitted threshold disorder of 0.5 logits, an offset of 1 logit and a proportion of 0.3 of missing data:

```
my_mfrm_sim = MFRM_Sim_Matrix(10, 5000, 12, 5, item_range=4, rater_range=3,
                              category_base=1.5, person_sd=2,
                              max_disorder=0.5, offset=1, missing=0.3)
```

To create an MFRM_Sim_Matrix object called my_mfrm_sim with a maximum score of 5, specified person abilities for 100 persons saved as a variable named my_person_abils, specified item difficulties for 12 items saved as a variable named my_item_diffs, specified rater severities for 6 raters saved as a variable named my_severities, a specified set of Rasch-Andrich thresholds saved as a variable named my_thresholds, specified person names saved as a variable named my_person_names, specified item names saved as a variable named my_item_names and specified rater names saved as a variable named my_rater_names, with no missing data:

```
my_mfrm_sim = MFRM_Sim_Matrix(12, 100, 6, 5, manual_abilities=my_person_abils,
                              manual_diffs=my_item_diffs,
                              manual_severities=my_severities,
                              manual_thresholds=my_thresholds,
                              manual_person_names=my_person_names,
                              manual_item_names=my_item_names,
                              manual_rater_names=my_rater_names)
```

13.2 Customising an `MFRM_Sim_Matrix` simulation

13.2.1 `rename_item`

Description

Method to rename a single item.

Usage

`self.rename_item(old, new)`

Arguments

`old`	String: the old name for the item
`new`	String: the new name for the item

Returns

Replaces specified item name in the relevant column of `self.scores` with new name.

Example

To rename an item in object `my_mfrm` from `Item_1` to `my_new_item_name`:

```
my_mfrm.rename_item('Item_1', 'my_new_item_name')
```

13.2.2 `rename_items_all`

Description

Method to rename all items.

Usage

`self.rename_items_all(new_names)`

Arguments

`new_names`	List of new item names as strings

Returns

Replaces all item names in the columns of `self.scores` with new names.

Example

To rename all items in object `my_mfrm` with item names in a list stored as a variable `my_new_item_names`:

```
my_mfrm.rename_items_all(my_new_item_names)
```

13.2.3 `rename_person`

Description
Method to rename a single person.

Usage
`self.rename_person(old, new)`

Arguments

`old`	String: the old name for the person
`new`	String: the new name for the person

Returns
Replaces specified person name in the second multiindex level of `self.scores` with new name.

Example
To rename a person in object `my_mfrm` from `Person_1` to `my_new_person_name`:

```
my_mfrm.rename_person('Person_1', 'my_new_person_name')
```

13.2.4 `rename_persons_all`

Description Method to rename all persons.

Usage
`self.rename_persons_all(new_names)`

Arguments

`new_names`	List of new person names as strings

Returns
Replaces all person names in the second multiindex level of `self.scores` with new names.

Example
To rename all persons in object `my_mfrm` with person names in a list stored as a variable `my_new_person_names`:

```
my_mfrm.rename_persons_all(my_new_person_names)
```

13.2.5 `rename_rater`

Description

Method to rename a single rater.

Usage

`self.rename_rater(old, new)`

Arguments

`old`	String: the old name for the rater
`new`	String: the new name for the rater

Returns

Replaces specified rater name in the first multiindex level of `self.scores` with new name.

Example

To rename a rater in object `my_mfrm` from `Rater_1` to `my_new_rater_name`:

```
my_mfrm.rename_rater('Rater_1', 'my_new_rater_name')
```

13.2.6 `rename_raters_all`

Description Method to rename all raters.

Usage

`self.rename_raters_all(new_names)`

Arguments

`new_names`	List of new rater names as strings

Returns

Replaces all rater names in the first multiindex level of `self.scores` with new names.

Example

To rename all raters in object `my_mfrm` with rater names in a list stored as a variable `my_new_rater_names`:

```
my_mfrm.rename_raters_all(my_new_rater_names)
```

References

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika*, *43*(4), 561–573.

Andrich, D. (2010). Understanding the Response Structure and Process in the Polytomous Rasch Model. In *Handbook of polytomous item response theory models* (123–152).

Andrich, D., & Kreiner, S. (2010). Quantifying response dependence between two dichotomous items using the Rasch model. *Applied Psychological Measurement*, *34*(3), 181–192.

Bradley, R. A., & Terry, M. E. (1952). Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*, *39*(3/4), 324.

Choppin, B. (1968). Item bank using sample-free calibration. *Nature*, *219*(5156), 870–872.

Choppin, B. (1985). A fully conditional estimation procedure for Rasch model parameters. *Evaluation in Education*, *9*(1), 29–42.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, *16*(3), 297–334.

Elliott, M. (2023). RaschPy.

Elliott, M., & Buttery, P. J. (2022a). Extended rater representations in the many-facet Rasch model. *Journal of Applied Measurement*, *22*(1), 133–160.

Elliott, M., & Buttery, P. J. (2022b). Non-iterative conditional pairwise estimation for the rating scale model. *Educational and Psychological Measurement*, *82*(5), 989–1019.

Garner, M., & Engelhard, G. (2002). An eigenvector method for estimating item parameters of the dichotomous and polytomous Rasch models. *Journal of Applied Measurement*, *3*(2), 107–128.

Garner, M., & Engelhard, G. (2009). Using paired comparison matrices to estimate parameters of the partial credit Rasch measurement model for rater-mediated assessments. *Journal of Applied Measurement*, *10*(1), 30–41.

Kornbrot, D. (2014). Point Biserial Correlation. In N. Balakrishnan, T. Colton, B. Everitt, W. Piegorsch, F. Ruggeri, & J. Teugels (Eds.), *Wiley statsref: Statistics reference online*. Wiley.

Kou, G., & Lin, C. (2014). A cosine maximization method for the priority vector derivation in AHP. *European Journal of Operational Research*, *235*(1), 225–232.

Linacre, J. M. (1994). *Many-Facet Rasch Measurement*. MESA Press.

Linacre, J. M. (2023). *Winsteps® Rasch measurement computer program User's Guide. Version 5.6.0*. Winsteps.com.

Marais, I. (2012). Local independence. In K. B. Christensen, S. Kreiner, & M. Mesbah (Eds.), *Rasch models in health* (111–130). John Wiley & Sons.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika*, *47*(2), 149–174.

Pallant, J. F., & Tennant, A. (2007). An introduction to the Rasch measurement model: An example using the Hospital Anxiety and Depression Scale (HADS). *British Journal of Clinical Psychology*, *46*(1), 1–18.

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests*. Danmarks Pædagogiske Institut.

Rasch, G. (1968). A Mathematical Theory of Objectivity and its Consequences for Model Construction. *Paper presented at the European Meeting on Statistics, Econometrics and Management Science, Amsterdam, September 2–7, 1968*.

Smith, E. V. (2002). Detecting and Evaluating the Impact of Multidimensionality Using Item Fit Statistics and Principal Component Analysis of Residuals. *Journal of Applied Measurement*, *3*(2), 205–231.

Warm, T. A. (1989). Weighted likelihood estimation of ability in item response theory. *Psychometrika*, *54*(3), 427–450.

Waskom, M. (2021). Seaborn: Statistical Data Visualization. *Journal of Open Source Software*, *6*(60), 3021.

Wright, B. D. (1996). Comparing Rasch measurement and factor analysis. *Structural Equation Modeling*, *3*(1), 3–24.

Wright, B. D., & Masters, G. N. (1982). *Rating Scale Analysis*. MESA Press.