Collate 'RcppExports.R' 'import.R' 'extensions.R' 'item_class.R'
  'calc_prob_functions.r' 'calc_escore_functions.r'
  'calc_location_functions.r' 'calc_fisher_functions.r'
  'calc_loglikelihood_functions.r' 'calc_jacobian_functions.r'
  'calc_hessian_functions.r' 'sim_resp_functions.r'
  'loading_functions.R' 'static_class.R' 'shadow_class.R'
  'item_pool_operators.R' 'item_attrib_operators.R'
  'st_attrib_operators.R' 'constraints_operators.R'
  'static_functions.R' 'shadow_functions.R' 'bayes_functions.R'
  'constraint_functions.R' 'cpp_calc_documents.r'
  'cpp_core_documents.r' 'cpp_theta_documents.r' 'datasets.R'
  'eligibility_functions.R' 'exposure_control_functions.R'
  'solver_functions.R' 'helper_functions.R'
  'item_pool_cluster_operators.R' 'other_functions.R'
  'partitioning_class.r' 'partitioning_functions.r'
  'plot_functions.R' 'summary_class.R' 'print_functions.R'
  'runshiny.R' 'shadowtest_functions.R' 'summary_functions.R'
  'show_functions.R' 'simulation_data_cache_class.r'
  'simulation_data_cache_operators.r' 'theta_functions.R'
  'xdata_functions.R'

NeedsCompilation yes

Author  Seung W. Choi [aut, cre] (<https://orcid.org/0000-0003-4777-5420>),
        Sangdon Lim [aut] (<https://orcid.org/0000-0002-2988-014X>)

Repository  CRAN

Date/Publication  2023-01-30 15:40:02 UTC

R topics documented:

app ................................................................. 4
a_to_alpha ......................................................... 5
buildConstraints .................................................. 5
calcEscore .......................................................... 6
calcFisher ............................................................ 9
calcHessian .......................................................... 11
calcJacobian .......................................................... 14
calcLocation-methods ............................................. 16
calcLogLikelihood ................................................. 18
calcProb-methods ................................................ 19
calc_info .......................................................... 22
calc_info_EB ....................................................... 23
calc_info_FB ....................................................... 24
calc_likelihood .................................................. 24
calc_MI_FB ........................................................ 27
calc_posterior .................................................... 27
calc_posterior_function ....................................... 28
calc_posterior_single ......................................... 28
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkConstraints</td>
<td>29</td>
</tr>
<tr>
<td>config_Shadow-class</td>
<td>29</td>
</tr>
<tr>
<td>config_Static-class</td>
<td>33</td>
</tr>
<tr>
<td>constraint-class</td>
<td>35</td>
</tr>
<tr>
<td>constraints-class</td>
<td>36</td>
</tr>
<tr>
<td>constraints-operators</td>
<td>37</td>
</tr>
<tr>
<td>dataset_bayes</td>
<td>38</td>
</tr>
<tr>
<td>dataset_fatigue</td>
<td>38</td>
</tr>
<tr>
<td>dataset_reading</td>
<td>39</td>
</tr>
<tr>
<td>dataset_science</td>
<td>40</td>
</tr>
<tr>
<td>detectBestSolver</td>
<td>40</td>
</tr>
<tr>
<td>eap</td>
<td>41</td>
</tr>
<tr>
<td>e_item</td>
<td>42</td>
</tr>
<tr>
<td>find_segment</td>
<td>44</td>
</tr>
<tr>
<td>getSolution</td>
<td>45</td>
</tr>
<tr>
<td>getSolution.Attributes</td>
<td>46</td>
</tr>
<tr>
<td>h_item</td>
<td>47</td>
</tr>
<tr>
<td>info_item</td>
<td>49</td>
</tr>
<tr>
<td>iparPosteriorSample</td>
<td>53</td>
</tr>
<tr>
<td>item-classes</td>
<td>53</td>
</tr>
<tr>
<td>item_atirb-class</td>
<td>54</td>
</tr>
<tr>
<td>item_atirb-operators</td>
<td>55</td>
</tr>
<tr>
<td>item_pool-class</td>
<td>56</td>
</tr>
<tr>
<td>item_pool-operators</td>
<td>57</td>
</tr>
<tr>
<td>item_pool_cluster-class</td>
<td>59</td>
</tr>
<tr>
<td>j_item</td>
<td>59</td>
</tr>
<tr>
<td>lnHyperPars</td>
<td>61</td>
</tr>
<tr>
<td>loadConstraints</td>
<td>62</td>
</tr>
<tr>
<td>loadItemPool</td>
<td>63</td>
</tr>
<tr>
<td>logitHyperPars</td>
<td>65</td>
</tr>
<tr>
<td>makeItemPoolCluster</td>
<td>65</td>
</tr>
<tr>
<td>makeSimulationDataCache</td>
<td>66</td>
</tr>
<tr>
<td>makeTest</td>
<td>67</td>
</tr>
<tr>
<td>mle</td>
<td>68</td>
</tr>
<tr>
<td>mlef</td>
<td>68</td>
</tr>
<tr>
<td>output_Shadow-class</td>
<td>73</td>
</tr>
<tr>
<td>output_Shadow_all-class</td>
<td>74</td>
</tr>
<tr>
<td>output_Split-class</td>
<td>75</td>
</tr>
<tr>
<td>output_Static-class</td>
<td>75</td>
</tr>
<tr>
<td>plot</td>
<td>76</td>
</tr>
<tr>
<td>print</td>
<td>80</td>
</tr>
<tr>
<td>p_item</td>
<td>82</td>
</tr>
<tr>
<td>RE</td>
<td>85</td>
</tr>
<tr>
<td>RMSE</td>
<td>85</td>
</tr>
<tr>
<td>runAssembly</td>
<td>86</td>
</tr>
<tr>
<td>Shadow</td>
<td>87</td>
</tr>
<tr>
<td>show</td>
<td>89</td>
</tr>
</tbody>
</table>
app

Open TestDesign app

Description

app and OAT are aliases of TestDesign.

Usage

app()

OAT()

Details

TestDesign is a caller function to open the Shiny interface of TestDesign package.

Examples

```r
## Not run:
if (interactive()) {
  TestDesign()
}
## End(Not run)
```
a_to_alpha

**Description**

*a_to_alpha* is a function for converting an a-parameter vector to an alpha angle vector. The returned values are in the radian metric.

**Usage**

```r
a_to_alpha(a)
```

**Arguments**

- `a` the a-parameter vector.

**Examples**

```r
a_to_alpha(c(1, 1))
```

---

**buildConstraints**

**Build constraints (shortcut to other loading functions)**

**Description**

*buildConstraints* is a data loading function to create a constraints object. *buildConstraints* is a shortcut that calls other data loading functions. The constraints must be in the expected format; see the vignette in vignette("constraints").

**Usage**

```r
buildConstraints(
  object,
  item_pool,
  item_attrib,
  st_attrib = NULL,
  pool = NULL,
  constraints = NULL
)
```
Arguments

object constraint specifications. Can be a data.frame or the file path of a .csv file. See the vignette for the expected format.

item_pool item parameters. Can be a item_pool object, a data.frame or the file path of a .csv file.

item_attrib item attributes. Can be an item_attrib object, a data.frame or the file path of a .csv file.

st_attrib (optional) stimulus attributes. Can be an st_attrib object, a data.frame or the file path of a .csv file.

pool (deprecated) use item_pool argument instead.

constraints (deprecated) use object argument instead.

Value

buildConstraints returns a constraints object. This object is used in Static and Shadow.

Examples

## Read from objects:
constraints_science <- buildConstraints(constraints_science_data, itempool_science, itemattrib_science)
constraints_reading <- buildConstraints(constraints_reading_data, itempool_reading, itemattrib_reading, stimattrib_reading)

## Read from data.frame:
constraints_science <- buildConstraints(constraints_science_data, itempool_science_data, itemattrib_science_data)
constraints_reading <- buildConstraints(constraints_reading_data, itempool_reading_data, itemattrib_reading_data, stimattrib_reading_data)

## Read from file: write to tempdir() for illustration and clean afterwards
f1 <- file.path(tempdir(), "constraints_science.csv")
f2 <- file.path(tempdir(), "itempool_science.csv")
f3 <- file.path(tempdir(), "itemattrib_science.csv")
write.csv(constraints_science_data, f1, row.names = FALSE)
write.csv(itempool_science_data, f2, row.names = FALSE)
write.csv(itemattrib_science_data, f3, row.names = FALSE)
constraints_science <- buildConstraints(f1, f2, f3)
file.remove(f1)
file.remove(f2)
file.remove(f3)

calcEscore Calculate expected scores

Description

calcEscore is a function for calculating expected scores.
Usage

calcEscore(object, theta)

## S4 method for signature 'item_1PL,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_2PL,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_3PL,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_PC,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_GPC,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_GR,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_pool,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_1PL,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_2PL,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_3PL,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_PC,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_GPC,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_GR,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_pool,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_pool_cluster,numeric'
calcEscore(object, theta)
**Arguments**

- **object**: an item or an item_pool object.
- **theta**: theta values to use.

**Value**

- **item object**: calcEscore returns a vector containing expected score of the item at the theta values.
- **item_pool object**: calcEscore returns a vector containing the pool-level expected score at the theta values.

**References**


**Examples**

```r
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5 <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6 <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

ICC_item_1 <- calcEscore(item_1, seq(-3, 3, 1))
ICC_item_2 <- calcEscore(item_2, seq(-3, 3, 1))
ICC_item_3 <- calcEscore(item_3, seq(-3, 3, 1))
```
calcFisher

\[
\text{ICC\_item\_4} \leftarrow \text{calcEscore(item\_4, seq(-3, 3, 1))}
\]
\[
\text{ICC\_item\_5} \leftarrow \text{calcEscore(item\_5, seq(-3, 3, 1))}
\]
\[
\text{ICC\_item\_6} \leftarrow \text{calcEscore(item\_6, seq(-3, 3, 1))}
\]
\[
\text{TCC\_pool} \leftarrow \text{calcEscore(itempool\_science, seq(-3, 3, 1))}
\]

\begin{tabular}{|l|l|}
\hline
\textbf{calcFisher} & \textit{Calculate Fisher information} \\
\hline
\end{tabular}

Description

\texttt{calcFisher} is a function to calculate Fisher information.

Usage

calcFisher(object, theta)

## S4 method for signature 'item\_1PL,numeric'
calcFisher(object, theta)

## S4 method for signature 'item\_2PL,numeric'
calcFisher(object, theta)

## S4 method for signature 'item\_3PL,numeric'
calcFisher(object, theta)

## S4 method for signature 'item\_PC,numeric'
calcFisher(object, theta)

## S4 method for signature 'item\_GPC,numeric'
calcFisher(object, theta)

## S4 method for signature 'item\_GR,numeric'
calcFisher(object, theta)

## S4 method for signature 'item\_pool,numeric'
calcFisher(object, theta)

## S4 method for signature 'item\_1PL,matrix'
calcFisher(object, theta)

## S4 method for signature 'item\_2PL,matrix'
calcFisher(object, theta)

## S4 method for signature 'item\_3PL,matrix'
calcFisher(object, theta)
## S4 method for signature 'item_PC,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_GPC,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_GR,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_pool,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_pool_cluster,numeric'
calcFisher(object, theta)

### Arguments

- **object**: an item or an *item_pool* object.
- **theta**: theta values to use.

### Value

- **item object**: `calcFisher` returns a \((nq, I)\) matrix of information values.
- **item_pool object**: `calcProb` returns a \((nq, ni)\) matrix of information values.

**notations**

- \(nq\) denotes the number of theta values.
- \(ni\) denotes the number of items in the *item_pool* object.

A vector of Fisher information values over theta \((nq values)\) for a single item or a matrix of dimension \((nq, ni)\) for an "item_pool".

### References


### Examples

```r
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5 <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6 <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

info_item_1 <- calcFisher(item_1, seq(-3, 3, 1))
info_item_2 <- calcFisher(item_2, seq(-3, 3, 1))
info_item_3 <- calcFisher(item_3, seq(-3, 3, 1))
info_item_4 <- calcFisher(item_4, seq(-3, 3, 1))
info_item_5 <- calcFisher(item_5, seq(-3, 3, 1))
info_item_6 <- calcFisher(item_6, seq(-3, 3, 1))
info_pool <- calcFisher(itempool_science, seq(-3, 3, 1))
```

### Description

`calcHessian` is a function to calculate the second derivative of the log-likelihood function.

### Usage

```r
calcHessian(object, theta, resp)
```

## S4 method for signature 'item_1PL,numeric,numeric'

calcHessian(object, theta, resp)

## S4 method for signature 'item_2PL,numeric,numeric'

calcHessian(object, theta, resp)

## S4 method for signature 'item_3PL,numeric,numeric'

calcHessian(object, theta, resp)

## S4 method for signature 'item_PC,numeric,numeric'

calcHessian(object, theta, resp)

## S4 method for signature 'item_GPC,numeric,numeric'

calcHessian(object, theta, resp)

## S4 method for signature 'item_GR,numeric,numeric'

calcHessian(object, theta, resp)
calcHessian(object, theta, resp)

## S4 method for signature 'item_GPC,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_GR,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_1PL,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_2PL,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_3PL,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_PC,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_GPC,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_GR,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_pool,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_pool_cluster,numeric,list'
calcHessian(object, theta, resp)

### Arguments

- **object**: an *item* or an *item_pool* object.
- **theta**: theta values to use.
- **resp**: the response data to use. This must be a single value for an *item*, or a length $ni$ vector for an *item_pool*.

### Details

- **notations**
  - $nq$ denotes the number of theta values.
  - $ni$ denotes the number of items in the *item_pool* object.

### Value

- **item object**: *calcHessian* returns a length $nq$ vector containing the second derivative of the log-likelihood function, of observing the response at each theta.
calcHessian

**item_pool object**: `calcHessian` returns a \((nq, ni)\) matrix containing the second derivative of the log-likelihood function, of observing the response at each theta.

References


Examples

```r
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5 <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6 <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

h_item_1 <- calcHessian(item_1, seq(-3, 3, 1), 0)
```

calcJacobian

Calculate first derivative of log-likelihood

Description

calcJacobian is a function for calculating the first derivative of the log-likelihood function.

Usage

calcJacobian(object, theta, resp)

## S4 method for signature 'item_1PL,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_2PL,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_3PL,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_PC,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_GPC,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_GR,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_1PL,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_2PL,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_3PL,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_PC,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_GPC,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_GR,matrix,numeric'
calcJacobian(object, theta, resp)
calcJacobian

## S4 method for signature 'item_pool, numeric, numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_pool_cluster, numeric, list'
calcJacobian(object, theta, resp)

### Arguments

- **object**: an item or an item_pool object.
- **theta**: theta values to use.
- **resp**: the response value to use for each item.

### Value

- **item object**: calcJacobian returns a length $nq$ vector containing the first derivative of the log-likelihood function, of observing the response at each theta.
- **item_pool object**: calcJacobian returns a ($nq$, $ni$) matrix containing the first derivative of the log-likelihood function, of observing the response at each theta.

### Notations

- $nq$ denotes the number of theta values.
- $ni$ denotes the number of items in the item_pool object.

### References


Examples

```r
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5 <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6 <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

j_item_1 <- calcJacobian(item_1, seq(-3, 3, 1), 0)
j_item_2 <- calcJacobian(item_2, seq(-3, 3, 1), 0)
j_item_3 <- calcJacobian(item_3, seq(-3, 3, 1), 0)
j_item_4 <- calcJacobian(item_4, seq(-3, 3, 1), 0)
j_item_5 <- calcJacobian(item_5, seq(-3, 3, 1), 0)
j_item_6 <- calcJacobian(item_6, seq(-3, 3, 1), 0)
j_pool <- calcJacobian(
  itempool_science, seq(-3, 3, 1),
  rep(0, itempool_science@ni)
)
```

calcLocation-methods  

`calcLocation` is a function to calculate the central location (overall difficulty) of items.

Description

`calcLocation` is a function to calculate the central location (overall difficulty) of items.

Usage

```r
calcLocation(object)
## S4 method for signature 'item_1PL'
calcLocation(object)
## S4 method for signature 'item_2PL'
calcLocation(object)
## S4 method for signature 'item_3PL'
calcLocation(object)
## S4 method for signature 'item_PC'
calcLocation(object)
## S4 method for signature 'item_GPC'
calcLocation(object)
## S4 method for signature 'item_GR'
calcLocation(object)
```
## S4 method for signature 'item_pool'

```r
calcLocation(object)
```

### Arguments

- **object**: an *item* or an *item_pool* object.

### Value

- **item object**: `calcLocation` returns a theta value representing the central location.
- **item_pool object**: `calcProb` returns a length `ni` list, each containing the central location of the item.

**notations**

- `ni` denotes the number of items in the *item_pool* object.

### References


### Examples

```r
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5 <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
```
calcLogLikelihood <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

loc_item_1 <- calcLocation(item_1)
loc_item_2 <- calcLocation(item_2)
loc_item_3 <- calcLocation(item_3)
loc_item_4 <- calcLocation(item_4)
loc_item_5 <- calcLocation(item_5)
loc_item_6 <- calcLocation(item_6)
loc_pool <- calcLocation(itempool_science)

---

calcLogLikelihood  
*Calculate log-likelihood*

**Description**

calcLogLikelihood is a function to calculate log-likelihood values.

**Usage**
calcLogLikelihood(object, theta, resp)

```
## S4 method for signature 'item_pool,numeric,numeric'
calcLogLikelihood(object, theta, resp)

## S4 method for signature 'item_pool,numeric,matrix'
calcLogLikelihood(object, theta, resp)

## S4 method for signature 'item_pool,matrix,numeric'
calcLogLikelihood(object, theta, resp)

## S4 method for signature 'item_pool,matrix,matrix'
calcLogLikelihood(object, theta, resp)
```

**Arguments**

- `object`: an `item_pool` object.
- `theta`: theta values to use.
- `resp`: the response data to use.

**Value**

calcLogLikelihood returns values of log-likelihoods.
References


Examples

```r
j_pool <- calcLogLikelihood(itempool_science, seq(-3, 3, 1), 0)
```

---

**Description**

calcProb is a function to calculate item response probabilities.

**Usage**

calcProb(object, theta)

```r
## S4 method for signature 'item_1PL,numeric'
calcProb(object, theta)
```

```r
## S4 method for signature 'item_2PL,numeric'
calcProb(object, theta)
```
## S4 method for signature 'item_3PL,numeric'
calcProb(object, theta)

## S4 method for signature 'item_PC,numeric'
calcProb(object, theta)

## S4 method for signature 'item_GPC,numeric'
calcProb(object, theta)

## S4 method for signature 'item_GR,numeric'
calcProb(object, theta)

## S4 method for signature 'item_pool,numeric'
calcProb(object, theta)

## S4 method for signature 'item_1PL,matrix'
calcProb(object, theta)

## S4 method for signature 'item_2PL,matrix'
calcProb(object, theta)

## S4 method for signature 'item_3PL,matrix'
calcProb(object, theta)

## S4 method for signature 'item_PC,matrix'
calcProb(object, theta)

## S4 method for signature 'item_GPC,matrix'
calcProb(object, theta)

## S4 method for signature 'item_GR,matrix'
calcProb(object, theta)

## S4 method for signature 'item_pool,matrix'
calcProb(object, theta)

## S4 method for signature 'item_pool_cluster,numeric'
calcProb(object, theta)

### Arguments

- **object**: an `item` or an `item_pool` object.
- **theta**: theta values to use.

### Value

- **item object**: `calcProb` returns a $(ng, ncat)$ matrix of probability values.
item_pool object: calcProb returns a length $ni$ list, each containing a matrix of probability values.

notations
- $nq$ denotes the number of theta values.
- $ncat$ denotes the number of response categories.
- $ni$ denotes the number of items in the item_pool object.

References


Examples

```r
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5 <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6 <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

prob_item_1 <- calcProb(item_1, seq(-3, 3, 1))
prob_item_2 <- calcProb(item_2, seq(-3, 3, 1))
prob_item_3 <- calcProb(item_3, seq(-3, 3, 1))
prob_item_4 <- calcProb(item_4, seq(-3, 3, 1))
prob_item_5 <- calcProb(item_5, seq(-3, 3, 1))
prob_item_6 <- calcProb(item_6, seq(-3, 3, 1))
prob_pool <- calcProb(itempool_science, seq(-3, 3, 1))
```
Description

calc_info() and calc_info_matrix() are functions for calculating Fisher information. These functions are designed for multiple items.

Usage

calc_info(x, item_parm, ncat, model)
calc_info_matrix(x, item_parm, ncat, model)

Arguments

x
  the theta value. This must be a column vector in matrix form for calc_info_matrix().

item_parm
  a matrix containing item parameters. Each row should represent an item.

ncat
  a vector containing the number of response categories of each item.

model
  a vector indicating item models of each item, using

  • 1: 1PL model
  • 2: 2PL model
  • 3: 3PL model
  • 4: PC model
  • 5: GPC model
  • 6: GR model

Details

calc_info() accepts a single theta value, and calc_info_matrix() accepts multiple theta values. Currently supports unidimensional models.

References


Examples

```r
# item parameters
item_parm <- matrix(c(
  1, NA, NA, 
  1, 2, NA, 
  1, 2, 0.25, 
  0, 1, NA, 
  2, 0, 1, 
  2, 0, 2),
  nrow = 6,
  byrow = TRUE
)

ncat <- c(2, 2, 2, 3, 3, 3)
model <- c(1, 2, 3, 4, 5, 6)

# single theta example
x <- 0.5
calc_info(x, item_parm, ncat, model)

# multiple thetas example
x <- matrix(seq(0.1, 0.5, 0.1)) # column vector in matrix form
calc_info_matrix(x, item_parm, ncat, model)
```

**Description**

Calculate the Fisher information using empirical Bayes.
Usage

calc_info_EB(x, item_parm, ncat, model)

Arguments

x A numeric vector of MCMC sampled theta values.
item_parm A numeric matrix of item parameters.
ncat a numeric vector specifying the number of response categories in each item.
model a numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).

Usage

calc_info_FB(x, items_list, ncat, model, useEAP = FALSE)

Arguments

x A numeric vector of MCMC sampled theta values.
items_list A list of item parameter matrices.
ncat a numeric vector specifying the number of response categories in each item.
model a numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
useEAP TRUE to use the mean of MCMC theta draws.

Description

calc_info_FB Calculate the Fisher information using full Bayesian

Usage

calc_info_FB(x, items_list, ncat, model, useEAP = FALSE)

Arguments

x A numeric vector of MCMC sampled theta values.
items_list A list of item parameter matrices.
ncat a numeric vector specifying the number of response categories in each item.
model a numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
useEAP TRUE to use the mean of MCMC theta draws.

Description

calc_info_FB() is a function for calculating the Fisher information using full Bayesian.

Usage

calc_info_FB(x, items_list, ncat, model, useEAP = FALSE)

Arguments

x A numeric vector of MCMC sampled theta values.
items_list A list of item parameter matrices.
ncat a numeric vector specifying the number of response categories in each item.
model a numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
useEAP TRUE to use the mean of MCMC theta draws.

Description

calcInfo_FB() is a function for calculating the Fisher information using full Bayesian.
Usage

calc_likelihod(x, item_parm, resp, ncat, model)

calc_likelihod_function(theta_grid, item_parm, resp, ncat, model)

calc_log_likelihod(x, item_parm, resp, ncat, model, prior, prior_parm)

calc_log_likelihod_function(
    theta_grid,
    item_parm,
    resp,
    ncat,
    model,
    prior,
    prior_parm
)

Arguments

x, theta_grid the theta value. This must be a column vector in matrix form for calc_*_function() functions.

item_parm a matrix containing item parameters. Each row should represent an item.

resp a vector containing responses on each item.

ncat a vector containing the number of response categories of each item.

model a vector indicating item models of each item, using

• 1: 1PL model
• 2: 2PL model
• 3: 3PL model
• 4: PC model
• 5: GPC model
• 6: GR model

prior an integer indicating the type of prior distribution, using

• 1: normal distribution
• 2: uniform distribution

prior_parm a vector containing parameters for the prior distribution.

Details

calc_log_likelihod() and calc_log_likelihod_function() are functions for calculating log likelihoods.

These functions are designed for multiple items.

calc_*() functions accept a single theta value, and calc_*_function() functions accept multiple theta values.

Currently supports unidimensional models.
References


Examples

```r
# item parameters
item_parm <- matrix(c(
  1, NA, NA,
  1, 2, NA,
  1, 2, 0.25,
  0, 1, NA,
  2, 0, 1,
  2, 0, 2),
  nrow = 6, byrow = TRUE)

ncat <- c(2, 2, 2, 3, 3, 3)
model <- c(1, 2, 3, 4, 5, 6)
resp <- c(0, 1, 0, 1, 0, 1)

x <- 3
l <- calc_likelihood(x, item_parm, resp, ncat, model)
ll <- calc_log_likelihood(x, item_parm, resp, ncat, model, 2, NA)
log(l) == ll

x <- matrix(seq(-3, 3, .1))
```
calc_MI_FB

1 <- calc_likelihood_function(x, item_parm, resp, ncat, model)
ll <- calc_log_likelihood_function(x, item_parm, resp, ncat, model, 2, NA)
all(log(l) == ll)

calc_MI_FB

Calculate the mutual information using full Bayesian

Description

Calculate the mutual information using full Bayesian.

Usage

calc_MI_FB(x, items_list, ncat, model)

Arguments

x
A numeric vector of MCMC sampled theta values.

items_list
A list of item parameter matrices.

ncat
A numeric vector specifying the number of response categories in each item.

model
A numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).

calc_posterior

Calculate a posterior value of theta

Description

Calculate a posterior value of theta.

Usage

calc_posterior(x, item_parm, resp, ncat, model, prior, prior_parm)

Arguments

x
A length-one numeric vector for a theta value.

item_parm
A numeric matrix of item parameters.

resp
A numeric vector containing item responses.

ncat
A numeric vector specifying the number of response categories in each item.

model
A numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).

prior
The type of prior distribution (1: normal, 2: uniform).

prior_parm
A numeric vector of hyperparameters for the prior distribution, c(mu, sigma) or c(ll, ul).
calc_posterior_function

*Calculate a posterior distribution of theta*

**Description**

Calculate a posterior distribution of theta.

**Usage**

```r
calc_posterior_function(
    theta_grid,  
    item_parm, 
    resp, 
    ncat, 
    model, 
    prior, 
    prior_parm
)
```

**Arguments**

- `theta_grid`: An equi-spaced grid of theta values.
- `item_parm`: A numeric matrix of item parameters.
- `resp`: A numeric vector containing item responses.
- `ncat`: A numeric vector of the number of response categories by item.
- `model`: A numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
- `prior`: The type of prior distribution (1: normal, 2: uniform).
- `prior_parm`: A numeric vector of hyperparameters for the prior distribution, c(mu, sigma) or c(ll, ul).

---

calc_posterior_single

*Calculate a posterior value of theta for a single item*

**Description**

Calculate a posterior value of theta for a single item.

**Usage**

```r
calc_posterior_single(x, item_parm, resp, ncat, model, prior, prior_parm)
```
checkConstraints

**Arguments**

- **x**
  - A length-one numeric vector for a theta value.
- **item_parm**
  - A numeric vector of item parameters (for one item).
- **resp**
  - A length-one numeric vector of item responses.
- **ncat**
  - A length-one numeric vector of the number of response categories by item.
- **model**
  - A length-one numeric vector of the IRT model by item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
- **prior**
  - The type of prior distribution (1: normal, 2: uniform).
- **prior_parm**
  - A numeric vector of hyperparameters for the prior distribution, c(mu, sigma) or c(ll, ul).

---

checkConstraints  
*Check the consistency of constraints and item usage*

**Description**

Check the consistency of constraints and item usage.

**Usage**

checkConstraints(constraints, usage_matrix, true_theta = NULL)

**Arguments**

- **constraints**
  - A constraints object generated by loadConstraints.
- **usage_matrix**
  - A matrix of item usage data from Shadow.
- **true_theta**
  - A vector of true theta values.

---

config_Shadow-class  
*Create a config_Shadow object*

**Description**

createShadowTestConfig is a config function to create a config_Shadow object for Shadow test assembly. Default values are used for any unspecified parameters/slots.
Usage

createShadowTestConfig(
    item_selection = NULL,
    content_balancing = NULL,
    MIP = NULL,
    MCMC = NULL,
    exclude_policy = NULL,
    refresh_policy = NULL,
    exposure_control = NULL,
    stopping_criterion = NULL,
    interim_theta = NULL,
    final_theta = NULL,
    theta_grid = seq(-4, 4, 0.1)
)

Arguments

item_selection  a named list containing item selection criteria.
    • method the type of selection criteria. Accepts MFI, MPWI, FB, EB, GFI. (default = MFI)
    • info_type the type of information. Accepts FISHER. (default = FISHER)
    • initial_theta (optional) initial theta values to use.
    • fixed_theta (optional) fixed theta values to use throughout all item positions.
    • target_value (optional) the target value to use for method = 'GFI'.

content_balancing  a named list containing content balancing options.
    • method the type of balancing method. Accepts NONE, STA. (default = STA)

MIP  a named list containing solver options.
    • solver the type of solver. Accepts Rsymphony, gurobi, lpSolve, Rglpk. (default = LPSOLVE)
    • verbosity verbosity level of the solver. (default = -2)
    • time_limit time limit in seconds. Used in solvers Rsymphony, gurobi, Rglpk. (default = 60)
    • gap_limit search termination criterion. Gap limit in relative scale passed onto the solver. Used in solver gurobi. (default = .05)
    • gap_limit_abs search termination criterion. Gap limit in absolute scale passed onto the solver. Used in solvers Rsymphony. (default = 0.05)
    • obj_tol search termination criterion. The lower bound to use on the min-max deviation variable. Used when item_selection$method is GFI, and ignored otherwise. (default = 0.05)
    • retry number of times to retry running the solver if the solver returns no solution. Some solvers incorrectly return no solution even when a solution exists. This is the number of attempts to verify that the problem is indeed infeasible in such cases. Set to 0 to not retry. (default = 5)
MCMC a named list containing Markov-chain Monte Carlo configurations for obtaining posterior samples.
- burn_in the number of chains from the start to discard. (default = 100)
- post_burn_in the number of chains to use after discarding the first burn_in chains. (default = 500)
- thin thinning interval to apply. 1 represents no thinning. (default = 1)
- jump_factor the jump (scaling) factor for the proposal distribution. 1 represents no jumping. (default = 2.4)

exclude_policy a named list containing the exclude policy for use with the exclude argument in Shadow.
- method the type of policy. Accepts HARD, SOFT. (default = HARD)
- M the Big M penalty to use on item information. Used in the SOFT method.

refresh_policy a named list containing the refresh policy for when to obtain a new shadow test.
- method the type of policy. Accepts ALWAYS, POSITION, INTERVAL, THRESHOLD, INTERVAL-THRESHOLD, STIMULUS, SET, PASSAGE. (default = ALWAYS)
- interval used in methods INTERVAL, INTERVAL-THRESHOLD. Set to 1 to refresh at each position, 2 to refresh at every two positions, and so on. (default = 1)
- threshold used in methods THRESHOLD, INTERVAL-THRESHOLD. The absolute change in between interim theta estimates to trigger the refresh. (default = 0.1)
- position used in methods POSITION. Item positions to trigger the refresh. (default = 1)

exposure_control a named list containing exposure control settings.
- method the type of exposure control method. Accepts NONE, ELIGIBILITY, BIGM, BIGM-BAYESIAN. (default = ELIGIBILITY)
- M used in methods BIGM, BIGM-BAYESIAN. the Big M penalty to use on item information.
- max_exposure_rate target exposure rates for each segment. (default = rep(0.25, 7))
- acceleration_factor the acceleration factor to apply. (default = 1)
- n_segment the number of theta segments to use. (default = 7)
- first_segment (optional) the theta segment assumed at the beginning of test for all participants.
- segment_cut theta segment cuts. (default = c(-Inf, seq(-2.5, 2.5, 1), Inf))
- initial_eligibility_stats (optional) initial eligibility statistics to use.
- fading_factor the fading factor to apply. (default = .999)
- diagnostic_stats set to TRUE to generate segment-wise diagnostic statistics. (default = FALSE)

stopping_criterion a named list containing stopping criterion.
- method the type of stopping criterion. Accepts FIXED. (default = FIXED)
• test_length test length.
• min_ni the maximum number of items to administer.
• max_ni the minimum number of items to administer.
• se_threshold standard error threshold. Item administration is stopped when theta estimate standard error becomes lower than this value.

interim_theta a named list containing interim theta estimation options.

• method the type of estimation. Accepts EAP, MLE, MLEF, EB, FB. (default = EAP)
• shrinkage_correction set TRUE to apply shrinkage correction. Used when method is EAP. (default = FALSE)
• prior_dist the type of prior distribution. Accepts NORMAL, UNIFORM. (default = NORMAL)
• prior_par distribution parameters for prior_dist. (default = c(0, 1))
• bound_ML theta bound in c(lower_bound, upper_bound) format. Used when method is MLE. (default = -4, 4)
• truncate_ML set TRUE to truncate ML estimate within bound_ML. (default = FALSE)
• max_iter maximum number of Newton-Raphson iterations. Used when method is MLE. (default = 50)
• crit convergence criterion. Used when method is MLE. (default = 1e-03)
• max_change maximum change in ML estimates between iterations. Changes exceeding this value is clipped to this value. Used when method is MLE. (default = 1.0)
• use_step_size set TRUE to use step_size. Used when method is MLE or MLEF. (default = FALSE)
• step_size upper bound to impose on the absolute change in initial theta and estimated theta. Absolute changes exceeding this value will be capped to step_size. Used when method is MLE or MLEF. (default = 0.5)
• do_Fisher set TRUE to use Fisher’s method of scoring. Used when method is MLE. (default = TRUE)
• fence_slope slope parameter to use for method = 'MLEF'. This must have two values in total, for the lower and upper bound item respectively. Use one value to use the same value for both bounds. (default = 5)
• fence_difficulty difficulty parameters to use for method = 'MLEF'. This must have two values in total, for the lower and upper bound item respectively. (default = c(-5, 5))
• hand_scored_attribute (optional) the item attribute name for whether each item is hand-scored or not. The attribute should have TRUE (hand-scored) and FALSE (machine-scored) values. If a hand-scored item is administered to an examinee, the previous interim theta (or the starting theta if this occurs for the first item) is reused without updating the estimate.

final_theta a named list containing final theta estimation options.

• method the type of estimation. Accepts EAP, MLE, MLEF, EB, FB. (default = EAP)
• shrinkage_correction set TRUE to apply shrinkage correction. Used when method is EAP. (default = FALSE)
• prior_dist the type of prior distribution. Accepts NORMAL, UNIFORM. (default = NORMAL)
• prior_par distribution parameters for prior_dist. (default = c(0, 1))
• bound_ML theta bound in c(lower_bound, upper_bound) format. Used when method is MLE. (default = -4, 4)
• truncate_ML set TRUE to truncate ML estimate within bound_ML. (default = FALSE)
• max_iter maximum number of Newton-Raphson iterations. Used when method is MLE. (default = 50)
• crit convergence criterion. Used when method is MLE. (default = 1e-03)
• max_change maximum change in ML estimates between iterations. Changes exceeding this value is clipped to this value. Used when method is MLE. (default = 1.0)
• use_step_size set TRUE to use step_size. Used when method is MLE or MLEF. (default = FALSE)
• step_size upper bound to impose on the absolute change in initial theta and estimated theta. Absolute changes exceeding this value will be capped to step_size. Used when method is MLE or MLEF. (default = 0.5)
• do_Fisher set TRUE to use Fisher's method of scoring. Used when method is MLE. (default = TRUE)
• fence_slope slope parameter to use for method = 'MLEF'. This must have two values in total, for the lower and upper bound item respectively. Use one value to use the same value for both bounds. (default = 5)
• fence_difficulty difficulty parameters to use for method = 'MLEF'. This must have two values in total, for the lower and upper bound item respectively. (default = c(-5, 5))

theta_grid the theta grid to use as quadrature points.

Examples

```r
cfg1 <- createShadowTestConfig(refresh_policy = list(
  method = "STIMULUS"
))
cfg2 <- createShadowTestConfig(refresh_policy = list(
  method = "POSITION",
  position = c(1, 5, 9)
))
```

Description

`createStaticTestConfig` is a config function to create a `config_Static` object for Static (fixed-form) test assembly. Default values are used for any unspecified parameters/slots.
Usage

createStaticTestConfig(item_selection = NULL, MIP = NULL)

Arguments

item_selection  a named list containing item selection criteria.

  • method the type of selection criteria. Accepts MAXINFO, TIF, TCC. (default = MAXINFO)
  • info_type the type of information. Accepts FISHER. (default = FISHER)
  • target_location a numeric vector containing the locations of target theta points. (e.g. c(-1, 0, 1)) (default = c(-1.2, 0, 1.2))
  • target_value a numeric vector containing the target values at each theta location. This should have the same length with target_location. Ignored if method is MAXINFO. (default = NULL)
  • target_weight a numeric vector containing the weights for each theta location. This should have the same length with target_location. (default = rep(1, length(target_location))

MIP  a named list containing solver options.

  • solver the type of solver. Accepts Rsymphony, gurobi, lpSolve, Rglpk. (default = LPSOLVE)
  • verbosity verbosity level of the solver. (default = -2)
  • time_limit time limit in seconds. Used in solvers Rsymphony, gurobi, Rglpk. (default = 60)
  • gap_limit search termination criterion. Gap limit in relative scale passed onto the solver. Used in solver gurobi. (default = .05)
  • gap_limit_abs search termination criterion. Gap limit in absolute scale passed onto the solver. Used in solvers Rsymphony. (default = 0.05)
  • obj_tol search termination criterion. The lower bound to use on the min-max deviation variable. Used when item_selection$method is TIF or TCC. (default = 0.05)
  • retry number of times to retry running the solver if the solver returns no solution. Some solvers incorrectly return no solution even when a solution exists. This is the number of attempts to verify that the problem is indeed infeasible in such cases. Set to 0 to not retry. (default = 5)

Value

createStaticTestConfig returns a config_Static object. This object is used in Static.

Examples

```r
cfg1 <- createStaticTestConfig(
  list(  
    method = "MAXINFO",  
    info_type = "FISHER",  
    target_location = c(-1, 0, 1),  
    target_weight = c(1, 1, 1)  
  )
```

```r
```
### constraint-class

**Class 'constraint': a single constraint**

**Description**

`constraint` is an S4 class to represent a single constraint.

**Slots**

- `constraint` the numeric index of the constraint.
- `constraint_id` the character ID of the constraint.
- `nc` the number of MIP-format constraints translated from this constraint.
- `mat, dir, rhs` these represent MIP-format constraints. A single MIP-format constraint is associated with a row in `mat`, a value in `rhs`, and a value in `dir`.
  - the `i`-th row of `mat` represents LHS coefficients to use on decision variables in the `i`-th MIP-format constraint.
  - the `i`-th value of `rhs` represents RHS values to use in the `i`-th MIP-format constraint.
  - the `i`-th value of `dir` represents the imposed constraint between LHS and RHS.
- `suspend` TRUE if the constraint is not to be imposed.

```r

cfg2 <- createStaticTestConfig(
  list(
    method = "TIF",
    info_type = "FISHER",
    target_location = c(-1, 0, 1),
    target_weight = c(1, 1, 1),
    target_value = c(8, 10, 12)
  )
)

cfg3 <- createStaticTestConfig(
  list(
    method = "TCC",
    info_type = "FISHER",
    target_location = c(-1, 0, 1),
    target_weight = c(1, 1, 1),
    target_value = c(10, 15, 20)
  )
)
```
constraints-class  

Class 'constraints': a set of constraints

Description

constraints is an S4 class to represent a set of constraints and its associated objects.

Details

See constraints-operators for object manipulation functions.

Slots

- `constraints` a data.frame containing the constraint specifications.
- `list_constraints` a list containing the constraint object representation of each constraint.
- `pool` the item_pool object associated with the constraints.
- `item_attrib` the item_attrib object associated with the constraints.
- `st_attrib` the st_attrib object associated with the constraints.
- `test_length` the test length specified in the constraints.
- `nv` the number of decision variables. Equals ni + ns.
- `ni` the number of items to search from.
- `ns` the number of stimulus to search from.
- `id` the item/stimulus ID string of each item/stimulus.
- `index,mat,dir,rhs` these represent MIP-format constraints. A single MIP-format constraint is associated with a value in index, a row in mat, a value in rhs, and a value in dir.
  - the i-th value of index represents which constraint specification in the constraints argument it was translated from.
  - the i-th row of mat represents LHS coefficients to use on decision variables in the i-th MIP-format constraint.
  - the i-th value of rhs represents RHS values to use in the i-th MIP-format constraint.
  - the i-th value of dir represents the imposed constraint between LHS and RHS.
- `set_based` TRUE if the constraint is set-based. FALSE otherwise.
- `item_order` the item attribute of each item to use in imposing an item order constraint, if any.
- `item_order_by` the name of the item attribute to use in imposing an item order constraint, if any.
- `stim_order` the stimulus attribute of each stimulus to use in imposing a stimulus order constraint, if any.
- `stim_order_by` the name of the stimulus attribute to use in imposing a stimulus order constraint, if any.
- `item_index_by_stimulus` a list containing item indices of each stimulus.
- `stimulus_index_by_item` the stimulus indices of each item.
Basic operators for constraints objects

Description

Create a subset of a constraints object:

- `constraints[i]`
- `subsetConstraints(constraints, 1:10)`

Combine two constraints objects:

- `c(constraints1, constraints2)`
- `combineConstraints(constraints1, constraints2)`

Usage

```r
subsetConstraints(x, i = NULL)
combineConstraints(x1, x2)
```

Arguments

- `x, x1, x2` a constraints object.
- `i, j` indices to use in subsetting.
- `...` not used, exists for compatibility.
- `drop` not used, exists for compatibility.

Examples

```r
c1 <- constraints_science
c2 <- c1[1:10]
c3 <- c1[c(1, 11:36)] # keep constraint 1 for test length
c4 <- c(c2, c3)
```
dataset_bayes \hspace{1cm} \textit{Bayes dataset}

**Description**

Item-based example item pool with standard errors (320 items).

**Details**

This pool is associated with the following objects:

- itempool_bayes an \texttt{item\_pool} object containing 320 items.
- itemattrib_bayes a \texttt{item\_attrib} object containing 5 item-level attributes.
- constraints_bayes a \texttt{constraints} object containing 14 constraints.

Also, the following objects are intended for illustrating expected data structures.

- itempool_bayes\_data a \texttt{data.frame} containing item parameters.
- itempool\_se\_bayes\_data a \texttt{data.frame} containing item parameter standard errors.
- itemattrib\_bayes\_data a \texttt{data.frame} containing item attributes.
- constraints\_bayes\_data a \texttt{data.frame} containing constraint specifications.

**Examples**

```r
itempool_bayes <- loadItemPool(itempool_bayes_data, itempool_se_bayes_data)
itemattrib_bayes <- loadItemAttrib(itemattrib_bayes_data, itempool_bayes)
constraints_bayes <- loadConstraints(constraints_bayes_data, 
                                     itempool_bayes, itemattrib_bayes)
```

dataset_fatigue \hspace{1cm} \textit{Fatigue dataset}

**Description**

Item-based example pool with item contents (95 items).

**Details**

This pool is associated with the following objects:

- itempool_fatigue an \texttt{item\_pool} object containing 95 items.
- itemattrib_fatigue an \texttt{item\_attrib} object containing 7 item-level attributes.
- constraints_fatigue a \texttt{constraints} object containing 111 constraints.
Also, the following objects are intended for illustrating expected data structures.

- `itempool_fatigue_data` a `data.frame` containing item parameters.
- `itemattrib_fatigue_data` a `data.frame` containing item attributes.
- `itemtext_fatigue_data` a `data.frame` containing item texts.
- `constraints_fatigue_data` a `data.frame` containing constraint specifications.
- `resp_fatigue_data` a `data.frame` containing raw response data.

**Examples**

```r
itempool_fatigue   <- loadItemPool(itempool_fatigue_data)
itemattrib_fatigue <- loadItemAttr(itemattrib_fatigue_data, itempool_fatigue)
constraints_fatigue <- loadConstraints(constraints_fatigue_data,
                                       itempool_fatigue, itemattrib_fatigue)
```

---

**Description**

Stimulus-based example item pool (303 items, 35 stimuli).

**Details**

This pool is associated with the following objects:

- `itempool_reading` an `item_pool` object containing 303 items.
- `itemattrib_reading` an `item_attrib` object containing 12 item-level attributes.
- `stimattrib_reading` a `st_attrib` object containing 4 stimulus-level attributes.
- `constraints_reading` a `constraints` object containing 18 constraints.

Also, the following objects are intended for illustrating expected data structures.

- `itempool_reading_data` a `data.frame` containing item parameters.
- `itemattrib_reading_data` a `data.frame` containing item attributes.
- `stimattrib_reading_data` a `data.frame` containing stimulus attributes.
- `constraints_reading_data` a `data.frame` containing constraint specifications.

**Examples**

```r
itempool_reading   <- loadItemPool(itempool_reading_data)
itemattrib_reading <- loadItemAttr(itemattrib_reading_data, itempool_reading)
stimattrib_reading <- loadStAttrib(stimattrib_reading_data, itemattrib_reading)
constraints_reading <- loadConstraints(constraints_reading_data,
                                        itempool_reading, itemattrib_reading, stimattrib_reading)
```
dataset_science  Science dataset

Description

Item-based example item pool (1000 items).

Details

This pool is associated with the following objects:

- itempool_science an item_pool object containing 1000 items.
- itemattrib_science an item_attrib object containing 9 item-level attributes.
- constraints_science a constraints object containing 36 constraints.

Also, the following objects are intended for illustrating expected data structures.

- itempool_science_data a data.frame containing item parameters.
- itemattrib_science_data a data.frame containing item attributes.
- constraints_science_data a data.frame containing constraint specifications.

Examples

```r
itempool_science <- loadItemPool(itempool_science_data)
itemattrib_science <- loadItemAttrib(itemattrib_science_data, itempool_science)
constraints_science <- loadConstraints(constraints_science_data,
                                        itempool_science, itemattrib_science)
```

detectBestSolver  Detect best solver

Description

Detect best solver

Usage

detectBestSolver()

Value

the package name of the best available solver on the system.
eap

Examples

```r
solver <- detectBestSolver()
cfg <- createStaticTestConfig(MIP = list(solver = solver))
cfg <- createShadowTestConfig(MIP = list(solver = solver))
```

---

**Description**

`eap` is a function to compute expected a posteriori estimates of theta.

**Usage**

```r
eap(
  object,
  select = NULL,
  resp,
  theta_grid = seq(-4, 4, 0.1),
  prior = rep(1/81, 81)
)
```

```r
# S4 method for signature 'item_pool'
eap(
  object,
  select = NULL,
  resp,
  theta_grid = seq(-4, 4, 0.1),
  prior = rep(1/81, 81)
)
```

```r
EAP(object, select = NULL, prior, reset_prior = FALSE)
```

```r
# S4 method for signature 'test'
EAP(object, select = NULL, prior, reset_prior = FALSE)
```

```r
# S4 method for signature 'test_cluster'
EAP(object, select = NULL, prior, reset_prior = FALSE)
```

**Arguments**

- `object`  
  an *item_pool* object.

- `select`  
  (optional) if item indices are supplied, only the specified items are used.

- `resp`  
  item response on all (or selected) items in the `object` argument. Can be a vector, a matrix, or a data frame. `length(resp)` or `nrow(resp)` must be equal to the number of all (or selected) items.
theta_grid  the theta grid to use as quadrature points. (default = seq(-4, 4, .1))
prior       a prior distribution, a numeric vector for a common prior or a matrix for individualized priors. (default = rep(1 / 81, 81))
reset_prior used for test_cluster objects. If TRUE, reset the prior distribution for each test object.

Value
eap returns a list containing estimated values.

- th theta value.
- se standard error.

Examples
eap(itempool_fatigue, resp = resp_fatigue_data[10, ])
eap(itempool_fatigue, select = 1:20, resp = resp_fatigue_data[10, 1:20])

---

e_item  (C++) Calculate expected scores

Description
e_*() and array_e_*() are C++ functions for calculating expected scores.

Usage
e_1pl(x, b)
e_2pl(x, a, b)
e_m_2pl(x, a, d)
e_3pl(x, a, b, c)
e_m_3pl(x, a, d, c)
e_pc(x, b)
e_gpc(x, a, b)
e_m_gpc(x, a, d)
e_gr(x, a, b)
e_m_gr(x, a, d)
array_e_1pl(x, b)
array_e_2pl(x, a, b)
array_e_3pl(x, a, b, c)
array_e_pc(x, b)
array_e_gpc(x, a, b)
array_e_gr(x, a, b)

Arguments

x the theta value. The number of columns should correspond to the number of
dimensions. For array_*() functions, the number of theta values must corre-
spond to the number of rows.
b, d the difficulty parameter. b is used for unidimensional items, and d is used for
multidimensional items.
a the a-parameter.
c the c-parameter.

Details

e_*() functions accept a single theta value, and array_p_*() functions accept multiple theta val-
ues.

Supports unidimensional and multidimensional models.

• e_1pl(), array_e_1pl(): 1PL models
• e_2pl(), array_e_2pl(): 2PL models
• e_3pl(), array_e_3pl(): 3PL models
• e_pc(), array_e_pc(): PC (partial credit) models
• e_gpc(), array_e_gpc(): GPC (generalized partial credit) models
• e_gr(), array_e_gr(): GR (graded response) models
• e_m_2pl(), array_e_m_2pl(): multidimensional 2PL models
• e_m_3pl(), array_e_m_3pl(): multidimensional 3PL models
• e_m_gpc(), array_e_m_gpc(): multidimensional GPC models
• e_m_gr(), array_e_m_gr(): multidimensional GR models

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests*. Copenhagen:
Danish Institute for Educational Research.


Examples

```r
x <- 0.5
e_1pl(x, 1)
e_2pl(x, 1, 2)
e_3pl(x, 1, 2, 0.25)
e_pc(x, c(0, 1))
e_gpc(x, 2, c(0, 1))
e_gr(x, 2, c(0, 2))
```

```r
x <- matrix(seq(-3, 3, 1)) # three theta values, unidimensional
array_e_1pl(x, 1)
array_e_2pl(x, 1, 2)
array_e_3pl(x, 1, 2, 0.25)
array_e_pc(x, c(0, 1))
array_e_gpc(x, 2, c(0, 1))
array_e_gr(x, 2, c(0, 2))
```

---

`find_segment` (C++) Classify theta values into segments using cutpoints

Description

`find_segment()` is a function to classify theta values into segments based on supplied cutpoints.
**Usage**

```r
find_segment(x, segment)
```

**Arguments**

- `x`: the theta value. This can be a vector.
- `segment`: segment cutpoints. Values of \(-\text{Inf}, \text{Inf}\) are not implied and must be explicitly supplied if intended.

**Examples**

```r
cuts <- c(-\text{Inf}, -2, 0, 2, \text{Inf})
find_segment(-3, cuts)
find_segment(-1, cuts)
find_segment(1, cuts)
find_segment(3, cuts)
find_segment(seq(-3, 3, 2), cuts)
```

---

**getSolution**

**Print solution items**

**Description**

Print solution items

**Usage**

```r
getSolution(object, examinee = NA, position = NA, index_only = TRUE)
```

```r
## S4 method for signature 'list'
getSolution(object, examinee = NA, position = NA, index_only = TRUE)
```

```r
## S4 method for signature 'output_Static'
getSolution(object, examinee = NA, position = NA, index_only = TRUE)
```

**Arguments**

- `object`: an `output_Static` object or an `output_Shadow` object.
- `examinee`: (optional) the examinee index to display the solution. Used when the 'object' argument is an `output_Shadow` object.
- `position`: (optional) if supplied, display the item attributes of the assembled test at that item position. If not supplied, display the item attributes of the administered items. Used when the 'object' argument is an `output_Shadow` object.
- `index_only`: if TRUE, only print item indices. if FALSE, print all item attributes. (default = TRUE)
getSolutionAttributes

Retrieve constraints-related attributes from solution

Description

getSolutionAttributes is a helper function to retrieve constraints-related attributes from a solution.

Usage

getSolutionAttributes(constraints, item_idx, all_values = FALSE)

Arguments

- **constraints**: a constraints object.
- **item_idx**: item indices from a solution.
- **all_values**: if TRUE, return all values as-is without taking the mean when there are multiple values. If FALSE, return the mean when there are multiple values. This has an effect when there is a constraint on items per stimulus, where there are multiple values of number of items per stimulus. In this case, if TRUE, the number of items for every stimuli are returned as-is. If FALSE, the average number of items across stimuli is returned. (default = FALSE)

Value

- If all_values == FALSE, getSolutionAttributes returns a data.frame containing constraints data and their associated attributes.
- If all_values == TRUE, getSolutionAttributes returns a list containing attributes associated to each constraint.

Examples

```r
item_idx <-
c( 29, 33, 26, 36, 34,
  295, 289, 296, 291, 126,
  133, 124, 134, 129, 38,
  47, 39, 41, 46, 45,
  167, 166, 170, 168, 113,
  116, 119, 117, 118, 114)

getSolutionAttributes(constraints_reading, item_idx, FALSE)
getSolutionAttributes(constraints_reading, item_idx, TRUE)
```
Description

$h_1()$ and $array_h_1()$ are C++ functions for calculating the second derivative of the log-likelihood function.

Usage

\begin{align*}
  h_1pl(x, b, u) \\
  h_2pl(x, a, b, u) \\
  h_m_2pl(x, a, d, u) \\
  h_3pl(x, a, b, c, u) \\
  h_m_3pl(x, a, d, c, u) \\
  h_pc(x, b, u) \\
  h_gpc(x, a, b, u) \\
  h_m_gpc(x, a, d, u) \\
  h_gr(x, a, b, u) \\
  h_m_gr(x, a, d, u) \\
  array_h_1pl(x, b, u) \\
  array_h_2pl(x, a, b, u) \\
  array_h_3pl(x, a, b, c, u) \\
  array_h_pc(x, b, u) \\
  array_h_gpc(x, a, b, u) \\
  array_h_gr(x, a, b, u)
\end{align*}

Arguments

- $x$ the theta value. The number of columns should correspond to the number of dimensions. For $array_*()$ functions, the number of theta values must correspond to the number of rows.
\( b, d \) the difficulty parameter. \( b \) is used for unidimensional items, and \( d \) is used for multidimensional items.

\( u \) the response value.

\( a \) the \( a \)-parameter.

\( c \) the \( c \)-parameter.

**Details**

\( h_\cdot() \) functions accept a single theta value, and \( \text{array}_h_\cdot() \) functions accept multiple theta values.

Supports unidimensional and multidimensional models.

- \( h_1p1(), \text{array}_h_1p1() \): 1PL models
- \( h_2p1(), \text{array}_h_2p1() \): 2PL models
- \( h_3p1(), \text{array}_h_3p1() \): 3PL models
- \( h_{pc}(), \text{array}_h_{pc}() \): PC (partial credit) models
- \( h_{gpc}(), \text{array}_h_{gpc}() \): GPC (generalized partial credit) models
- \( h_{gr}(), \text{array}_h_{gr}() \): GR (graded response) models
- \( h_{m\_2p1}(), \text{array}_h_{m\_2p1}() \): multidimensional 2PL models
- \( h_{m\_3p1}(), \text{array}_h_{m\_3p1}() \): multidimensional 3PL models
- \( h_{m\_gpc}(), \text{array}_h_{m\_gpc}() \): multidimensional GPC models
- \( h_{m\_gr}(), \text{array}_h_{m\_gr}() \): multidimensional GR models

**References**


Examples

```r
u <- 1
dx <- 0.5
h_1pl(x, 1, u)
h_2pl(x, 1, 2, u)
h_3pl(x, 1, 2, 0.25, u)
h_pc(x, c(0, 1), u)
h_gpc(x, 2, c(0, 1), u)
h_gr(x, 2, c(0, 2), u)
```

```r
x <- matrix(seq(-3, 3, 1)) # three theta values, unidimensional
array_h_1pl(x, 1, u)
array_h_2pl(x, 1, 2, u)
array_h_3pl(x, 1, 2, 0.25, u)
array_h_pc(x, c(0, 1), u)
array_h_gpc(x, 2, c(0, 1), u)
array_h_gr(x, 2, c(0, 2), u)
```

### Description

`info_*()` and `array_info_*()` are functions for calculating Fisher information.

### Usage

- `info_1pl(x, b)`
- `info_2pl(x, a, b)`
- `info_m_2pl(x, a, d)`
- `dirinfo_m_2pl(x, a, d)`
- `thisdirinfo_m_2pl(x, alpha_vec, a, d)`
- `info_3pl(x, a, b, c)`
- `info_m_3pl(x, a, d, c)`
- `array_info_1pl(x, 1, u)`
- `array_info_2pl(x, 1, 2, u)`
- `array_info_3pl(x, 1, 2, 0.25, u)`
- `array_info_pc(x, c(0, 1), u)`
- `array_info_gpc(x, 2, c(0, 1), u)`
- `array_info_gr(x, 2, c(0, 2), u)`
dirinfo_m_3pl(x, a, d, c)
thisdirinfo_m_3pl(x, alpha_vec, a, d, c)
info_pc(x, b)
info_gpc(x, a, b)
info_m_gpc(x, a, d)
dirinfo_m_gpc(x, a, d)
thisdirinfo_m_gpc(x, alpha_vec, a, d)
info_gr(x, a, b)
info_m_gr(x, a, d)
dirinfo_m_gr(x, a, d)
thisdirinfo_m_gr(x, alpha_vec, a, d)
array_info_1pl(x, b)
array_info_2pl(x, a, b)
array_info_m_2pl(x, a, d)
array_dirinfo_m_2pl(x, a, d)
array_thisdirinfo_m_2pl(x, alpha_vec, a, d)
array_info_3pl(x, a, b, c)
array_info_m_3pl(x, a, d, c)
array_dirinfo_m_3pl(x, a, d, c)
array_thisdirinfo_m_3pl(x, alpha_vec, a, d, c)
array_info_pc(x, b)
array_info_gpc(x, a, b)
array_info_m_gpc(x, a, d)
array_dirinfo_m_gpc(x, a, d)
array_thisdirinfo_m_gpc(x, alpha_vec, a, d)
array_info_gr(x, a, b)
array_info_m_gr(x, a, d)
array_dirinfo_m_gr(x, a, d)
array_thisdirinfo_m_gr(x, alpha_vec, a, d)

Arguments

- **x**
  - the theta value. The number of columns should correspond to the number of dimensions. For `array_*()` functions, the number of theta values must correspond to the number of rows.
- **b**, **d**
  - the difficulty parameter. `b` is used for unidimensional items, and `d` is used for multidimensional items.
- **a**
  - the `a`-parameter.
- **alpha_vec**
  - the alpha angle vector. Used for directional information in `thisdirinfo_*()` and `array_thisdirinfo_*()`.
- **c**
  - the `c`-parameter.

Details

`info_*()` functions accept a single theta value, and `array_info_*` functions accept multiple theta values.

Supports unidimensional and multidimensional models.

- `info_1pl()`, `array_info_1pl()`: 1PL models
- `info_2pl()`, `array_info_2pl()`: 2PL models
- `info_3pl()`, `array_info_3pl()`: 3PL models
- `info_pc()`, `array_info_pc()`: PC (partial credit) models
- `info_gpc()`, `array_info_gpc()`: GPC (generalized partial credit) models
- `info_gr()`, `array_info_gr()`: GR (graded response) models
- `info_m_2pl()`, `array_info_m_2pl()`: multidimensional 2PL models
- `info_m_3pl()`, `array_info_m_3pl()`: multidimensional 3PL models
- `info_m_gpc()`, `array_info_m_gpc()`: multidimensional GPC models
- `info_m_gr()`, `array_info_m_gr()`: multidimensional GR models

- Directional information for a specific angle
  - `thisdirinfo_m_2pl()`, `array_thisdirinfo_m_2pl()`: multidimensional 2PL models
  - `thisdirinfo_m_3pl()`, `array_thisdirinfo_m_3pl()`: multidimensional 3PL models
  - `thisdirinfo_m_gpc()`, `array_thisdirinfo_m_gpc()`: multidimensional GPC models
  - `thisdirinfo_m_gr()`, `array_thisdirinfo_m_gr()`: multidimensional GR models
References


Examples

```r
x <- 0.5
info_1pl(x, 1)
info_2pl(x, 1, 2)
info_3pl(x, 1, 2, 0.25)
info_pc(x, c(0, 1))
info_gpc(x, 2, c(0, 1))
info_gr(x, 2, c(0, 2))

x <- matrix(seq(0.1, 0.5, 0.1)) # three theta values, unidimensional
array_info_1pl(x, 1)
array_info_2pl(x, 1, 2)
array_info_3pl(x, 1, 2, 0.25)
array_info_pc(x, c(0, 1))
array_info_gpc(x, 2, c(0, 1))
array_info_gr(x, 2, c(0, 2))
```
**iparPosteriorSample**  
*Generate item parameter samples using standard errors*

**Description**

`iparPosteriorSample` is a function for generating item parameter samples.

**Usage**

`iparPosteriorSample(pool, n_sample = 500)`

**Arguments**

- `pool`: an `item_pool` object.
- `n_sample`: the number of samples to draw.

**Value**

`iparPosteriorSample` returns a length-`ni` list of item parameter matrices, with each matrix having `n_sample` rows.

**Examples**

```r
ipar <- iparPosteriorSample(itempool_bayes, 5)
```

---

**item-classes**  
*Item classes*

**Description**

- `item_1PL` class represents a 1PL item.
- `item_2PL` class represents a 2PL item.
- `item_3PL` class represents a 3PL item.
- `item_PC` class represents a partial credit item.
- `item_GPC` class represents a generalized partial credit item.
- `item_GR` class represents a graded response item.

**Slots**

- `slope`: a slope parameter value
- `difficulty`: a difficulty parameter value
- `guessing`: a guessing parameter value
- `threshold`: a vector of threshold parameter values
- `category`: a vector of category boundary values
- `ncat`: the number of response categories
References


Examples

```r
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-0.5, 0.5), ncat = 3)
item_5 <- new("item_GPC", slope = 1.0, threshold = c(-0.5, 0.0, 0.5), ncat = 4)
item_6 <- new("item_GR", slope = 1.0, category = c(-2.0, -1.0, 0, 1.0, 2.0), ncat = 6)
```

---

**item_attrib-class**

*Load item attributes*

**Description**

`loadItemAttrib` is a data loading function to create an `item_attrib` object. `loadItemAttrib` can read item attributes a `data.frame` or a `.csv` file.

**Usage**

`loadItemAttrib(object, pool, file = NULL)`
Arguments

object  item attributes. Can be a data.frame or the file path of a .csv file. The content should at least include column `ID` that matches with the item_pool object.

pool  an item_pool object. Use loadItemPool for this.

file  (deprecated) use object argument instead.

Value

loadItemAttrib returns an item_attrib object.

- data a data.frame containing item attributes.

See Also

dataset_science, dataset_reading, dataset_fatigue, dataset_bayes for examples.

Examples

```r
## Read from data.frame:
itempool_science <- loadItemPool(itempool_science_data)
itemattrib_science <- loadItemAttrib(itemattrib_science_data, itempool_science)

## Read from file: write to tempdir() for illustration and clean afterwards
f <- file.path(tempdir(), "itemattrib_science.csv")
write.csv(itemattrib_science_data, f, row.names = FALSE)
itemattrib_science <- loadItemAttrib(f, itempool_science)
file.remove(f)

## TestDesign 1.1.0 - Deprecated arguments
## Not run:
loadItemAttrib(object = "iatt.csv", pool) # is equivalent to
loadItemAttrib(file = "iatt.csv", pool) # pre 1.1.0

## End(Not run)
```
Usage

```r
## S4 method for signature 'item_attrib, numeric'
x[i, j, ..., drop = TRUE]
```

```r
## S4 method for signature 'item_attrib'
dim(x)
```

```r
## S4 method for signature 'item_attrib'
colnames(x)
```

```r
## S4 method for signature 'item_attrib'
rownames(x)
```

```r
## S4 method for signature 'item_attrib'
names(x)
```

```r
## S4 method for signature 'item_attrib'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

- `x`: an `item_attrib` object.
- `i, j`: indices to use in subsetting.
- `...`: not used, exists for compatibility.
- `drop`: not used, exists for compatibility.
- `row.names`: not used, exists for compatibility.
- `optional`: not used, exists for compatibility.

Examples

```r
x <- itemattrib_science
x[1:10]
dim(x)
ncol(x)
nrow(x)
colnames(x)
rownames(x)
names(x)
as.data.frame(x)
```

---

**item_pool-class**

Class 'item_pool': an item pool

Description

`item_pool` is an S4 class to represent an item pool.
Details

See `item_pool-operators` for object manipulation functions.

Slots

- `ni` the number of items in the pool.
- `max_cat` the maximum number of response categories across the pool.
- `index` the numeric index of each item.
- `id` the ID string of each item.
- `model` the item class name of each item. See `item-classes`.
- `NCAT` the number of response categories of each item.
- `parms` a list containing item class objects. See `item-classes`.
- `ipar` a matrix containing item parameters.
- `se` a matrix containing item parameter standard errors.
- `raw` the raw input `data.frame` used in `loadItemPool` to create this object.
- `raw_se` the raw input `data.frame` used in `loadItemPool` to create this object.
- `unique` whether item IDs must be unique for this object to be a valid object.

---

**item_pool-operators**  
*Basic operators for item pool objects*

Description

Create a subset of an `item_pool` object:

- `pool[i]`
- `subsetItemPool(pool, i)`

Combine two `item_pool` objects:

- `c(pool1, pool2)`
- `combineItemPool(pool1, pool2)`
- `pool1 + pool2`

`pool1 - pool2` excludes items in `pool2` from `pool1`.
`pool1 == pool2` tests whether two `item_pool` objects are identical.
Usage

subsetItemPool(x, i = NULL)

combineItemPool(x1, x2, unique = TRUE, verbose = TRUE)

## S4 method for signature 'item_pool,numeric'
## S4 method for signature 'item_pool'
## S4 method for signature 'item_pool'

Arguments

x, x1, x2 an item_pool object.
i item indices to use in subsetting.
unique if TRUE, remove items with duplicate IDs after combining. (default = TRUE)
verbose if TRUE, raise a warning if duplicate IDs are found after combining. (default = TRUE)
j, drop, ... not used, exists for compatibility.

Examples

p1 <- itempool_science[1:100]
p2 <- c(itempool_science, itempool_reading)
p3 <- p2 - p1

p1 <- itempool_science[1:500]
p2 <- itempool_science - p1
p3 <- itempool_science[501:1000]
identical(p2, p3) ## TRUE

p <- p1 + p3
p == itempool_science ## TRUE
item_pool_cluster-class

Class 'item_pool_cluster': an item pool

Description

item_pool_cluster is an S4 class to represent a group of item pools.

Slots

np the number of item pools.
pools a list of item_pool objects.
names a vector containing item pool names.

j_item (C++) Calculate first derivative of log-likelihood

Description

j_*() and array_j_*() are C++ functions for calculating the first derivative of the log-likelihood function.

Usage

j_1pl(x, b, u)
j_2pl(x, a, b, u)
j_m_2pl(x, a, d, u)
j_3pl(x, a, b, c, u)
j_m_3pl(x, a, d, c, u)
j_pc(x, b, u)
j_gpc(x, a, b, u)
j_m_gpc(x, a, d, u)
j_gr(x, a, b, u)
j_m_gr(x, a, d, u)
array_j_1pl(x, b, u)
array_j_2pl(x, a, b, u)
array_j_3pl(x, a, b, c, u)
array_j_pc(x, b, u)
array_j_gpc(x, a, b, u)
array_j_gr(x, a, b, u)

Arguments

x the theta value. The number of columns should correspond to the number of dimensions. For array_*() functions, the number of theta values must correspond to the number of rows.

b, d the difficulty parameter. b is used for unidimensional items, and d is used for multidimensional items.
u the response value.
a the a-parameter.
c the c-parameter.

details

j_*() functions accept a single theta value, and array_j_*() functions accept multiple theta values.

Supports unidimensional and multidimensional models.

- j_1pl(), array_j_1pl(): 1PL models
- j_2pl(), array_j_2pl(): 2PL models
- j_3pl(), array_j_3pl(): 3PL models
- j_pc(), array_j_pc(): PC (partial credit) models
- j_gpc(), array_j_gpc(): GPC (generalized partial credit) models
- j_gr(), array_j_gr(): GR (graded response) models
- j_m_2pl(), array_j_m_2pl(): multidimensional 2PL models
- j_m_3pl(), array_j_m_3pl(): multidimensional 3PL models
- j_m_gpc(), array_j_m_gpc(): multidimensional GPC models
- j_m_gr(), array_j_m_gr(): multidimensional GR models

References


Examples

\[
\begin{align*}
  u & \leftarrow 1 \\
  x & \leftarrow 0.5 \\
  j_1pl(x, 1, u) \\
  j_2pl(x, 1, 2, u) \\
  j_3pl(x, 1, 2, 0.25, u) \\
  j_{pc}(x, c(0, 1), u) \\
  j_{gpc}(x, 2, c(0, 1), u) \\
  j_{gr}(x, 2, c(0, 2), u) \\
\end{align*}
\]

\[
\begin{align*}
  x & \leftarrow \text{matrix(seq(-3, 3, 1))} \quad \text{# three theta values, unidimensional} \\
  \text{array}_j_{pl}(x, 1, u) \\
  \text{array}_j_{2pl}(x, 1, 2, u) \\
  \text{array}_j_{3pl}(x, 1, 2, 0.25, u) \\
  \text{array}_j_{pc}(x, c(0, 1), u) \\
  \text{array}_j_{gpc}(x, 2, c(0, 1), u) \\
  \text{array}_j_{gr}(x, 2, c(0, 2), u) \\
\end{align*}
\]

\begin{tabular}{ll}
\hline
\textbf{lnHyperPars} & Convert mean and standard deviation into log-normal distribution parameters \\
\hline
\end{tabular}
Description

\texttt{lnHyperPars} is a function for calculating parameters for a log-normal distribution, such that the distribution yields desired mean and standard deviation. Used for sampling the a-parameter.

Usage

\texttt{lnHyperPars(mean, sd)}

Arguments

- \texttt{mean} the desired mean.
- \texttt{sd} the desired standard deviation.

Value

\texttt{lnHyperPars} returns two values. These can be directly supplied to \texttt{rlnorm}.

Examples

\begin{verbatim}
pars <- lnHyperPars(2, 4)
x <- rlnorm(1000000, pars[1], pars[2])
mean(x) # close to 2
sd(x)  # close to 4
\end{verbatim}

---

loadConstraints \hspace{1cm} \textit{Load constraints}

Description

\texttt{loadConstraints} is a data loading function to create a \texttt{constraints} object. \texttt{loadConstraints} can read constraints from a data.frame or a .csv file. The contents must be in the expected format; see the vignette in \texttt{vignette(“constraints”)}.

Usage

\texttt{loadConstraints(object, pool, item_attrib, st_attrib = NULL, file = NULL)}

Arguments

- \texttt{object} constraint specifications. Can be a \texttt{data.frame} or the file path of a .csv file. See the vignette for the expected format.
- \texttt{pool} an \texttt{item_pool} object. Use \texttt{loadItemPool} for this.
- \texttt{item_attrib} an \texttt{item_attrib} object. Use \texttt{loadItemAttrib} for this.
- \texttt{st_attrib} (optional) an \texttt{st_attrib} object. Use \texttt{loadStAttrib} for this.
- \texttt{file} (deprecated) use object argument instead.
Value

`loadConstraints` returns a `constraints` object. This object is used in `Static` and `Shadow`.

See Also

dataset_science, dataset_reading, dataset_fatigue, dataset_bayes for examples.

Examples

```r
## Read from data.frame:
itempool_science <- loadItemPool(itempool_science_data)
itemattrib_science <- loadItemAttrib(itemattrib_science_data, itempool_science)
constraints_science <- loadConstraints(constraints_science_data,
                                       itempool_science, itemattrib_science)

## Read from file: write to tempdir() for illustration and clean afterwards
f <- file.path(tempdir(), "constraints_science.csv")
write.csv(constraints_science_data, f, row.names = FALSE)
constraints_science <- loadConstraints(f,
                                       itempool_science, itemattrib_science)
file.remove(f)

## TestDesign 1.1.0 - Deprecated arguments
## Not run:
loadConstraints(object = "consts.csv", pool, item_attrib) # is equivalent to
loadConstraints(file = "consts.csv", pool, item_attrib) # pre 1.1.0

## End(Not run)
```

### loadItemPool

**Load item pool**

**Description**

`loadItemPool` is a data loading function to create an `item_pool` object. `loadItemPool` can read item parameters and standard errors from a `data.frame` or a `.csv` file.

**Usage**

`loadItemPool(ipar, ipar_se = NULL, file = NULL, se_file = NULL, unique = FALSE)`

**Arguments**

- `ipar`: item parameters. Can be a `data.frame` or the file path of a `.csv` file. The content should at least include columns `ID` and `MODEL`.
- `ipar_se`: (optional) standard errors. Can be a `data.frame` or the file path of a `.csv` file.
- `file`: (deprecated) use `ipar` argument instead.
se_file       (deprecated) use ipar_se argument instead.
unique        if TRUE, item IDs must be unique to create a valid item_pool object. (default = FALSE)

Value

loadItemPool returns an item_pool object.

• ni the number of items in the pool.
• max_cat the maximum number of response categories across all items in the pool.
• index the numeric item index of each item.
• id the item ID string of each item.
• model the object class names of each item representing an item model type. Can be item_1PL, item_2PL, item_3PL, item_PC, item_GPC, or item_GR.
• NCAT the number of response categories of each item.
• parms a list containing the item object of each item.
• ipar a matrix containing all item parameters.
• se a matrix containing all item parameter standard errors. The values will be 0 if the argument ipar_se was not supplied.
• raw the original input data.frame used to create this object.

See Also

dataset_science, dataset_reading, dataset_fatigue, dataset_bayes for examples.

Examples

## Read from data.frame:
itempool_science <- loadItemPool(itempool_science_data)

## Read from file: write to tempdir() for illustration and clean afterwards
f <- file.path(tempdir(), "itempool_science.csv")
write.csv(itempool_science_data, f, row.names = FALSE)
itempool_science <- loadItemPool(f)
file.remove(f)

## TestDesign 1.1.0 - Deprecated arguments
## Not run:
loadItemPool(ipar = "ipar.csv", ipar_se = "se.csv") # is equivalent to
loadItemPool(file = "ipar.csv", se_file = "se.csv") # pre 1.1.0

## End(Not run)
logitHyperPars

Convert mean and standard deviation into logit-normal distribution parameters

Description

`logitHyperPars` is a function for calculating parameters for a logit-normal distribution, such that the distribution yields desired mean and standard deviation. Used for sampling the c-parameter.

Usage

```r
logitHyperPars(mean, sd)
```

Arguments

- `mean` the desired mean.
- `sd` the desired standard deviation.

Value

`logitHyperPars` returns two values. These can be directly supplied to `rlogitnorm`.

Examples

```r
pars <- logitHyperPars(0.2, 0.1)
x <- logitnorm::rlogitnorm(1000000, pars[1], pars[2])
mean(x) # close to 0.2
sd(x)  # close to 0.1
```

makeItemPoolCluster

Create an item pool cluster object

Description

Create a `item_pool_cluster` object.

`item_pool_cluster1 == item_pool_cluster2` tests equality of two `item_pool_cluster` objects.

Usage

```r
makeItemPoolCluster(x, ..., names = NULL)
```

## S4 method for signature 'item_pool'
```r
makeItemPoolCluster(x, ..., names = NULL)
```

## S3 method for class 'item_pool_cluster'
```r
item_pool_cluster1 == item_pool_cluster2
```
Arguments

- `x, ...`: `item_pool` objects.
- `names` (optional): names to use for `item_pool`.
- `item_pool_cluster1`: an `item_pool_cluster` object.
- `item_pool_cluster2`: an `item_pool_cluster` object.

Examples

```r
cluster <- makeItemPoolCluster(itempool_science, itempool_reading)
cluster1 <- makeItemPoolCluster(itempool_science, itempool_reading)
cluster2 <- makeItemPoolCluster(cluster1@pools[[1]], cluster1@pools[[2]])
cluster1 == cluster2  # TRUE
```

---

`makeSimulationDataCache` is a function for creating a `simulation_data_cache` object. This is used in Shadow to make all necessary data (e.g., item information, response data) prior to the main simulation.

Usage

```r
makeSimulationDataCache(
  item_pool,
  info_type = "FISHER",
  theta_grid = seq(-4, 4, 0.1),
  seed = NULL,
  true_theta = NULL,
  response_data = NULL
)
```

```r
## S4 method for signature 'item_pool'
makeSimulationDataCache(
  item_pool,
  info_type = "FISHER",
  theta_grid = seq(-4, 4, 0.1),
  seed = NULL,
  true_theta = NULL,
  response_data = NULL
)
```
Arguments

- **item_pool**
  - an *item_pool* object.

- **info_type**
  - the type of information.

- **theta_grid**
  - a grid of theta values.

- **seed**
  - (optional) seed to use for generating response data if needed.

- **true_theta**
  - (optional) true theta values of all simulees.

- **response_data**
  - (optional) response data on all items for all simulees.

**makeTest**

*Create a test object*

Description

*makeTest* is a function for creating a test object. This is used to make all necessary data (e.g., item information, response data) prior to the main simulation. This function is only kept for backwards compatibility. The functionality of this function is superseded by *makeSimulationDataCache*.

Usage

```r
makeTest(
  object,
  theta = seq(-4, 4, 0.1),
  info_type = "FISHER",
  true_theta = NULL
)
```

Arguments

- **object**
  - an *item_pool* object.

- **theta**
  - a grid of theta values.

- **info_type**
  - the type of information.

- **true_theta**
  - (optional) true theta values to simulate response data.

Examples

```r
test <- makeTest(itempool_science, seq(-3, 3, 1))
```
**makeTestCluster**

*Create a test cluster object*

**Description**

`makeTestCluster` is a function for creating a `test_cluster` object. This is used to make all necessary data (e.g., item information, response data) prior to the main simulation. This function is only kept for backwards compatibility.

**Usage**

```r
makeTestCluster(object, theta, true_theta)
```

## S4 method for signature 'item_pool_cluster,numeric,numeric'

```r
makeTestCluster(object, theta, true_theta)
```

## S4 method for signature 'item_pool_cluster,numeric,list'

```r
makeTestCluster(object, theta, true_theta)
```

**Arguments**

- `object`: an `item_pool_cluster` object.
- `theta`: a grid of theta values.
- `true_theta`: an optional vector of true theta values to simulate response data.

---

**mle**

*Compute maximum likelihood estimates of theta*

**Description**

`mle` is a function to compute maximum likelihood estimates of theta.

**Usage**

```r
mle(
  object,
  select = NULL,
  resp,
  start_theta = NULL,
  max_iter = 100,
  crit = 0.001,
  truncate = FALSE,
  theta_range = c(-4, 4),
  max_change = 1,
  use_step_size = FALSE,
```
step_size = 0.5,
do_Fisher = TRUE
)

## S4 method for signature 'item_pool'
mle(
  object,
  select = NULL,
  resp,
  start_theta = NULL,
  max_iter = 50,
  crit = 0.005,
  truncate = FALSE,
  theta_range = c(-4, 4),
  max_change = 1,
  use_step_size = FALSE,
  step_size = 0.5,
  do_Fisher = TRUE
)

MLE(
  object,
  select = NULL,
  start_theta = NULL,
  max_iter = 100,
  crit = 0.001,
  theta_range = c(-4, 4),
  truncate = FALSE,
  max_change = 1,
  do_Fisher = TRUE
)

## S4 method for signature 'test'
MLE(
  object,
  select = NULL,
  start_theta = NULL,
  max_iter = 100,
  crit = 0.001,
  theta_range = c(-4, 4),
  truncate = FALSE,
  max_change = 1,
  do_Fisher = TRUE
)

## S4 method for signature 'test_cluster'
MLE(object, select = NULL, start_theta = NULL, max_iter = 100, crit = 0.001)
Arguments

object  an item_pool object.
select (optional) if item indices are supplied, only the specified items are used.
resp  item response on all (or selected) items in the object argument. Can be a vector, a matrix, or a data frame. length(resp) or ncol(resp) must be equal to the number of all (or selected) items.
start_theta (optional) initial theta values. If not supplied, EAP estimates using uniform priors are used as initial values. Uniform priors are computed using the theta_range argument below, with increments of .1.
max_iter maximum number of iterations. (default = 100)
crit convergence criterion to use. (default = 0.001)
truncate set TRUE to impose a bound using theta_range on the estimate. (default = FALSE)
theta_range a range of theta values to bound the estimate. Only effective when truncate is TRUE. (default = c(-4, 4))
max_change upper bound to impose on the absolute change in theta between iterations. Absolute changes exceeding this value will be capped to max_change. (default = 1.0)
use_step_size set TRUE to use step_size. (default = FALSE)
step_size upper bound to impose on the absolute change in initial theta and estimated theta. Absolute changes exceeding this value will be capped to step_size. (default = 0.5)
do_Fisher set TRUE to use Fisher scoring instead of Newton-Raphson method. (default = TRUE)

Value

mle returns a list containing estimated values.

• th theta value.
• se standard error.
• conv TRUE if estimation converged.
• trunc TRUE if truncation was applied on th.

Examples

mle(itempool_fatigue, resp = resp_fatigue_data[10, ])
mle(itempool_fatigue, select = 1:20, resp = resp_fatigue_data[10, 1:20])
**mlef**  
Compute maximum likelihood estimates of theta using fence items

**Description**

*mlef* is a function to compute maximum likelihood estimates of theta using fence items.

**Usage**

```r
mlef(
  object,
  select = NULL,
  resp,
  fence_slope = 5,
  fence_difficulty = c(-5, 5),
  start_theta = NULL,
  max_iter = 100,
  crit = 0.001,
  truncate = FALSE,
  theta_range = c(-4, 4),
  max_change = 1,
  use_step_size = FALSE,
  step_size = 0.5,
  do_Fisher = TRUE
)
```

## S4 method for signature 'item_pool'

```r
mlef(
  object,
  select = NULL,
  resp,
  fence_slope = 5,
  fence_difficulty = c(-5, 5),
  start_theta = NULL,
  max_iter = 50,
  crit = 0.005,
  truncate = FALSE,
  theta_range = c(-4, 4),
  max_change = 1,
  use_step_size = FALSE,
  step_size = 0.5,
  do_Fisher = TRUE
)
```

**Arguments**

- **object**  
an *item_pool* object.
select (optional) if item indices are supplied, only the specified items are used.

resp item response on all (or selected) items in the object argument. Can be a vector, a matrix, or a data frame. length(resp) or ncol(resp) must be equal to the number of all (or selected) items.

fence_slope the slope parameter to use on fence items. Can be one value, or two values for the lower and the upper fence respectively. (default = 5)

fence_difficulty the difficulty parameter to use on fence items. Must have two values for the lower and the upper fence respectively. (default = c(-5, 5))

start_theta (optional) initial theta values. If not supplied, EAP estimates using uniform priors are used as initial values. Uniform priors are computed using the theta_range argument below, with increments of .1.

max_iter maximum number of iterations. (default = 100)

crit convergence criterion to use. (default = 0.001)

truncate set TRUE to impose a bound using theta_range on the estimate. (default = FALSE)

theta_range a range of theta values to bound the estimate. Only effective when truncate is TRUE. (default = c(-4, 4))

max_change upper bound to impose on the absolute change in theta between iterations. Absolute changes exceeding this value will be capped to max_change. (default = 1.0)

use_step_size set TRUE to use step_size. (default = FALSE)

step_size upper bound to impose on the absolute change in initial theta and estimated theta. Absolute changes exceeding this value will be capped to step_size. (default = 0.5)

do_Fisher set TRUE to use Fisher scoring instead of Newton-Raphson method. (default = TRUE)

Value

mlef returns a list containing estimated values.

- th theta value.
- se standard error.
- conv TRUE if estimation converged.
- trunc TRUE if truncation was applied on th.

References


Examples

mlef(itempool_fatigue, resp = resp_fatigue_data[10, ])

mlef(itempool_fatigue, select = 1:20, resp = resp_fatigue_data[10, 1:20])
Class `output_Shadow`: adaptive assembly solution for one simulee

Description

`output_Shadow` is an S4 class to represent the adaptive assembly solution for one simulee.

Slots

- `simulee_id` the numeric ID of the simulee.
- `true_theta` the true theta of the simulee, if was specified.
- `true_theta_segment` the segment number of the true theta.
- `final_theta_est` final theta estimate.
- `final_se_est` the standard error of `final_theta_est`.
- `administered_item_index` item IDs administered at each position.
- `administered_itemResp` item responses from the simulee at each position.
- `administered_item_ncat` the number of categories of each administered item.
- `administered_stimulus_index` stimulus IDs administered at each position.
- `shadow_test_refreshed` TRUE indicates the shadow test was refreshed for the position.
- `shadow_test_feasible` TRUE indicates the MIP was feasible with all constraints.
- `solve_time` elapsed time in running the solver at each position.
- `initial_theta_est` initial theta estimate.
- `interim_theta_est` interim theta estimates at each position.
- `interim_se_est` the standard error of the interim estimate at each position.
- `theta_segment_index` segment numbers of interim theta estimates.
- `prior` prior distribution, if was specified.
- `prior_par` prior parameters, if were specified.
- `posterior` the posterior distribution after completing test.
- `posterior_sample` posterior samples of interim theta before the estimation of final theta. \( \text{mean}(\text{posterior}\_\text{sample}) = \text{interim}\_\text{theta}\_\text{est}[\text{test}\_\text{length}] \) holds.
- `likelihood` the likelihood distribution after completing test.
- `shadow_test` the list containing the item IDs within the shadow test used in each position.
- `max_cat_pool` the maximum number of response categories the item pool had.
- `ni_pool` the total number of items the item pool had.
- `ns_pool` the total number of stimuli the item pool had.
- `test_length_constraints` the test length constraint used in assembly.
- `set_based` whether the item pool was set-based.
- `item_index_by_stimulus` the list of items by each stimulus the item pool had.
output_Shadow_all-class

Class ‘output_Shadow_all’: a set of adaptive assembly solutions

Description

description_Shadow_all is an S4 class to represent a set of adaptive assembly solutions.

details

notations

• \(n_i\) denotes the number of items in the item_pool object.
• \(n_s\) denotes the number of stimuli.
• \(n_j\) denotes the number of participants.

slots

output a length-\(n_j\) list of output_Shadow objects, containing the assembly results for each participant.
final_theta_est a length-\(n_j\) vector containing final theta estimates for each participant.
final_se_est a length-\(n_j\) vector standard errors of the final theta estimates for each participant.
exposure_rate a matrix containing item-level exposure rates of all items in the pool. Also contains stimulus-level exposure rates if the assembly was set-based.
usage_matrix a \(n_j\) by \((n_i + n_s)\) matrix representing whether the item/stimulus was administered to each participant. Stimuli representations are appended to the right side of the matrix.
true_segment_count a length-\(n_j\) vector containing the how many examinees are now in their segment based on the true theta. This will tend to increase. This can be reproduced with true theta values alone.
est_segment_count a length-\(n_j\) vector containing the how many examinees are now in their segment based on the estimated theta. This will tend to increase. This can be reproduced with estimated theta values alone.
eligibility_stats exposure record for diagnostics.
check_eligibility_stats detailed segment-wise exposure record for diagnostics. available when config_Shadow@exposure_control$diagnostic_stats is TRUE.
no_fading_eligibility_stats detailed segment-wise exposure record without fading for diagnostics. available when config_Shadow@exposure_control$diagnostic_stats is TRUE.
freq_infeasible a table representing the number of times the assembly was initially infeasible.
pool the item_pool used in the assembly.
config the config_Shadow used in the assembly.
constraints the constraints used in the assembly.
true_theta the true_theta argument used in the assembly.
data the data argument used in the assembly.
prior the prior argument used in the assembly.
prior_par the prior_par argument used in the assembly.
output_Split-class  

Class ‘output_Split’: partitioning solution

Description

output_Split is an S4 class to represent the partitioning solution of an item pool.

Slots

- output: a list containing item/set indices of each partition.
- feasible: for partitioning into sub-pools, TRUE indicates the complete assignment problem was feasible.
- solve_time: elapsed time in running the solver.
- set_based: whether the item pool is set-based.
- config: the config_Static used in the assembly.
- constraints: the constraints used in the assembly.
- partition_size_range: the partition size range for splitting into sub-pools.
- partition_type: the partition type. Can be a test or a pool.

output_Static-class  

Class ‘output_Static’: fixed-form assembly solution

Description

output_Static is an S4 class to represent a fixed-form assembly solution.

Slots

- MIP: a list containing the result from MIP solver.
- selected: a data.frame containing the selected items and their attributes.
- obj_value: the objective value of the solution.
- solve_time: the elapsed time in running the solver.
- achieved: a data.frame containing attributes of the assembled test, by each constraint.
- pool: the item_pool used in the assembly.
- config: the config_Static used in the assembly.
- constraints: the constraints used in the assembly.
Description

Extension of plot() for objects in TestDesign package

Usage

```r
## S4 method for signature 'item_pool'
plot(
x,  
y,  
type = "info",  
theta = seq(-3, 3, 0.1),  
info_type = "FISHER",  
plot_sum = TRUE,  
select = NULL,  
examinee_id = 1,  
position = NULL,  
theta_range = c(-5, 5),  
ylim = NULL,  
color = "blue",  
z_ci = 1.96,  
simple = TRUE,  
theta_type = "Estimated",  
color_final = "blue",  
segment = NULL,  
rmse = FALSE,  
use_segment_label = TRUE,  
use_par = TRUE,  
...)

## S4 method for signature 'output_Static'
plot(
x,  
y,  
type = NULL,  
theta = seq(-3, 3, 0.1),  
info_type = "FISHER",  
plot_sum = TRUE,  
select = NULL,  
examinee_id = 1,  
position = NULL,  
theta_range = c(-5, 5),  
ylim = NULL,
```

plot

    color = "blue",
    z_ci = 1.96,
    simple = TRUE,
    use_par = TRUE,
    ...
)

## S4 method for signature 'constraints'
plot(
x,
y,
type = "info",
theta = seq(-3, 3, 0.1),
info_type = "FISHER",
plot_sum = TRUE,
select = NULL,
examinee_id = 1,
position = NULL,
theta_range = c(-5, 5),
ylim = NULL,
color = "blue",
z_ci = 1.96,
simple = TRUE,
use_par = TRUE,
...
)

## S4 method for signature 'output_Shadow'
plot(
x,
y,
type = "audit",
theta = seq(-3, 3, 0.1),
info_type = "FISHER",
plot_sum = TRUE,
select = NULL,
examinee_id = 1,
theta_range = c(-5, 5),
ylim = NULL,
color = "blue",
z_ci = 1.96,
simple = FALSE,
theta_type = "Estimated",
use_par = TRUE,
...
)

## S4 method for signature 'output_Shadow_all'
plot(
  x,
  y,
  type = "audit",
  theta = seq(-3, 3, 0.1),
  info_type = "FISHER",
  plot_sum = TRUE,
  select = NULL,
  examinee_id = 1,
  position = NULL,
  theta_range = c(-5, 5),
  ylim = NULL,
  color = "blue",
  z_ci = 1.96,
  simple = FALSE,
  theta_type = "Estimated",
  color_final = "blue",
  segment = NULL,
  rmse = FALSE,
  use_segment_label = TRUE,
  use_par = TRUE,
  theta_segment = NULL,
  ...
)

## S4 method for signature 'output_Split'
plot(
  x,
  y,
  type = NULL,
  theta = seq(-3, 3, 0.1),
  info_type = "FISHER",
  plot_sum = TRUE,
  select = NULL,
  examinee_id = 1,
  position = NULL,
  theta_range = c(-5, 5),
  ylim = NULL,
  color = "blue",
  z_ci = 1.96,
  simple = TRUE,
  use_par = TRUE,
  ...
)

Arguments

- **x** accepts the following signatures:
item_pool: plot information and expected scores.

constraints: plot information range based on the test length constraint.

output_Static: plot information and expected scores based on the fixed assembly solution.

output_Shadow_all: plot audit trail, shadow test chart, and exposure rates from the adaptive assembly solution.

output_Shadow: plot audit trail and shadow test chart from the adaptive assembly solution.

y not used, exists for compatibility with plot in the base R package.

type

the type of plot.

• info plots information from item_pool, output_Static, and output_Shadow_all.
• score plots expected scores from item_pool and output_Static.
• audit plots audit trail from output_Shadow_all and output_Shadow.
• shadow plots shadow test chart from output_Shadow_all and output_Shadow.
• exposure plots exposure rates from output_Shadow_all.

theta

the theta grid to use in plotting. (default = seq(-3, 3, .1))

info_type

the type of information. Currently accepts FISHER. (default = FISHER)

plot_sum

used in item_pool objects.

• if TRUE then plot pool-level values.
• if FALSE then plot item-level values, and repeat for all items in the pool.
• (default = TRUE)

select

used in item_pool objects. Item indices to subset.

examinee_id

used in output_Shadow and output_Shadow_all with type = 'audit' and type = 'shadow'. The examinee numeric ID to draw the plot.

position

used in output_Shadow_all with type = 'info'. The item position to draw the plot.

theta_range

used in output_Shadow and output_Shadow_all with type = 'audit'. The theta range to plot. (default = c(-5, 5))

ylim

(optional) the y-axis plot range. Used in most plot types.

color

the color of the curve.

z_ci

used in output_Shadow and output_Shadow_all with type = 'audit'. The range to use for confidence intervals. (default = 1.96)

simple

used in output_Shadow and output_Shadow_all with type = 'shadow'. If TRUE, simplify the chart by hiding unused items.

theta_type

used in output_Shadow_all with type = 'exposure'. The type of theta to determine exposure segments. Accepts Estimated or True. (default = Estimated)

color_final

used in output_Shadow_all with type = 'exposure'. The color of item-wise exposure rates, only counting the items administered in the final theta segment as exposed.

segment

used in output_Shadow_all with type = 'exposure'. (optional) The segment index to draw the plot. Leave empty to use all segments.
rmse used in output_Shadow_all with type = 'exposure'. If TRUE, display the RMSE value for each segment. (default = FALSE)

use_segment_label used in output_Shadow_all with type = 'exposure'. If TRUE, display the segment label for each segment. (default = TRUE)

use_par if FALSE, graphical parameters are not overridden inside the function. (default = TRUE)

... arguments to pass onto plot.

theta_segment (deprecated) use theta_type argument instead.

Examples

subitempool <- itempool_science[1:8]

## Plot item information of a pool
plot(subitempool)
plot(itempool_science, select = 1:8)

## Plot expected score of a pool
plot(subitempool, type = "score")
plot(itempool_science, type = "score", select = 1:8)

## Plot assembly results from Static()
cfg <- createStaticTestConfig()
solution <- Static(cfg, constraints_science)
plot(solution) # defaults to the objective type
plot(solution, type = "score") # plot expected scores

## Plot attainable information range from constraints
plot(constraints_science)

## Plot assembly results from Shadow()
cfg <- createShadowTestConfig()
set.seed(1)
solution <- Shadow(cfg, constraints_science, true_theta = rnorm(1))
plot(solution, type = 'audit', examinee_id = 1)
plot(solution, type = 'shadow', examinee_id = 1, simple = TRUE)

## plot(solution, type = 'exposure')

print Extension of print() for objects in TestDesign package

Description

Extension of print() for objects in TestDesign package
Usage

```r
## S4 method for signature 'item_1PL'
print(x)

## S4 method for signature 'item_2PL'
print(x)

## S4 method for signature 'item_3PL'
print(x)

## S4 method for signature 'item_PC'
print(x)

## S4 method for signature 'item_GPC'
print(x)

## S4 method for signature 'item_GR'
print(x)

## S4 method for signature 'item_pool'
print(x)

## S4 method for signature 'item_attrib'
print(x)

## S4 method for signature 'st_attrib'
print(x)

## S4 method for signature 'summary_item_attrib'
print(x)

## S4 method for signature 'summary_st_attrib'
print(x)

## S4 method for signature 'constraints'
print(x)

## S4 method for signature 'config_Static'
print(x)

## S4 method for signature 'config_Shadow'
print(x)

## S4 method for signature 'output_Static'
print(x, index_only = TRUE)

## S4 method for signature 'output_Shadow'
print(x)
```
## S4 method for signature 'output_Shadow_all'
print(x)

## S4 method for signature 'exposure_rate_plot'
print(x)

## S4 method for signature 'summary_item_pool'
print(x)

## S4 method for signature 'summary_constraints'
print(x)

## S4 method for signature 'summary_output_Static'
print(x, digits = 3)

## S4 method for signature 'summary_output_Shadow_all'
print(x, digits = 3)

### Arguments

- **x**: an object to print.
- **index_only**: if TRUE then only print item indices. If FALSE then print all item attributes. (default = TRUE)
- **digits**: minimal number of *significant* digits. See `print.default`.

### Description

`p_*()` and `array_p_*()` are C++ functions for calculating item response probability.

### Usage

- `p_1pl(x, b)`
- `p_2pl(x, a, b)`
- `p_m_2pl(x, a, d)`
- `p_3pl(x, a, b, c)`
- `p_m_3pl(x, a, d, c)`
- `p_pc(x, b)`

---

**p_item**  
(C++) *Calculate item response probability*
p_item

\begin{align*}
& p_{\text{gpc}}(x, a, b) \\
& p_{\text{m-gpc}}(x, a, d) \\
& p_{\text{gr}}(x, a, b) \\
& p_{\text{m-gr}}(x, a, d) \\
& \text{array}_{p}\_\text{1pl}(x, b) \\
& \text{array}_{p}\_\text{2pl}(x, a, b) \\
& \text{array}_{p}\_\text{m-2pl}(x, a, d) \\
& \text{array}_{p}\_\text{3pl}(x, a, b, c) \\
& \text{array}_{p}\_\text{m-3pl}(x, a, d, c) \\
& \text{array}_{p}\_\text{pc}(x, b) \\
& \text{array}_{p}\_\text{gpc}(x, a, b) \\
& \text{array}_{p}\_\text{m-gpc}(x, a, d) \\
& \text{array}_{p}\_\text{gr}(x, a, b) \\
& \text{array}_{p}\_\text{m-gr}(x, a, d)
\end{align*}

**Arguments**

- **x**
  - the theta value. The number of columns should correspond to the number of dimensions. For `array_\text{*}` functions, the number of theta values must correspond to the number of rows.
- **b, d**
  - the difficulty parameter. b is used for unidimensional items, and d is used for multidimensional items.
- **a**
  - the a-parameter.
- **c**
  - the c-parameter.

**Details**

- `p_\text{*}` functions accept a single theta value, and `array_p_\text{*}` functions accept multiple theta values.
- Supports unidimensional and multidimensional models.

- `p_\text{1pl}`, `array_p_\text{1pl}`: 1PL models
- `p_\text{2pl}`, `array_p_\text{2pl}`: 2PL models
• p_3pl(), array_p_3pl(): 3PL models
• p_pc(), array_p_pc(): PC (partial credit) models
• p_gpc(), array_p_gpc(): GPC (generalized partial credit) models
• p_gr(), array_p_gr(): GR (graded response) models
• p_m_2pl(), array_p_m_2pl(): multidimensional 2PL models
• p_m_3pl(), array_p_m_3pl(): multidimensional 3PL models
• p_m_gpc(), array_p_m_gpc(): multidimensional GPC models
• p_m_gr(), array_p_m_gr(): multidimensional GR models

Examples

```
x <- 0.5
p_1pl(x, 1)
p_2pl(x, 1, 2)
p_3pl(x, 1, 2, 0.25)
p_pc(x, c(0, 1))
p_gpc(x, 2, c(0, 1))
p_gr(x, 2, c(0, 2))
```

References


array_p_1pl(x, 1)
array_p_2pl(x, 1, 2)
array_p_3pl(x, 1, 2, 0.25)
array_p_pc(x, c(0, 1))
array_p_gpc(x, 2, c(0, 1))
array_p_gr(x, 2, c(0, 2))

RE  

**Calculate Relative Errors**

**Description**

Calculate Relative Errors.

**Usage**

RE(RMSE_foc, RMSE_ref)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE_foc</td>
<td>A vector of RMSE values for the focal group.</td>
</tr>
<tr>
<td>RMSE_ref</td>
<td>A vector of RMSE values for the reference group.</td>
</tr>
</tbody>
</table>

RMSE  

**Calculate Root Mean Squared Error**

**Description**

Calculate Root Mean Squared Error.

**Usage**

RMSE(x, y, conditional = TRUE)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A vector of values.</td>
</tr>
<tr>
<td>y</td>
<td>A vector of values.</td>
</tr>
<tr>
<td>conditional</td>
<td>If TRUE, calculate RMSE conditional on x.</td>
</tr>
</tbody>
</table>
**runAssembly**

*Run Test Assembly*

**Description**

*runAssembly* is a function to perform test assembly. This function is used internally in Static and Shadow.

**Usage**

```r
callRunAssembly(config, constraints, xdata = NULL, objective = NULL)
```

**Arguments**

- `config`: a `config_Static` or a `config_Shadow` object containing configuration options. Use `createStaticTestConfig` and `createShadowTestConfig` for this.
- `constraints`: a `constraints` object. Use `loadConstraints` for this.
- `xdata`: a list containing extra constraints in MIP form, to force-include previously administered items.
- `objective`: the information value for each item in the pool.

**Value**

A list containing the following entries:

- `MIP`: a list containing the result from MIP solver.
- `status`: the MIP status value, indicating whether an optimal solution was found.
- `shadow_test`: the attributes of the selected items.
- `obj_value`: the objective value of the solution.
- `solve_time`: the elapsed time in running the solver.

**References**

Shadow

**Run adaptive test assembly**

**Description**

*Shadow* is a test assembly function to perform adaptive test assembly based on the generalized shadow-test framework.

**Usage**

```r
Shadow(
  config,
  constraints = NULL,
  true_theta = NULL,
  data = NULL,
  prior = NULL,
  prior_par = NULL,
  exclude = NULL,
  include_items_for_estimation = NULL,
  force_solver = FALSE,
  session = NULL,
  seed = NULL
)
```

## S4 method for signature 'config_Shadow'

```r
Shadow(
  config,
  constraints = NULL,
  true_theta = NULL,
  data = NULL,
  prior = NULL,
  prior_par = NULL,
  exclude = NULL,
  include_items_for_estimation = NULL,
  force_solver = FALSE,
  session = NULL,
  seed = NULL
)
```

**Arguments**

- `config` a `config_Shadow` object. Use `createShadowTestConfig` for this.
- `constraints` a `constraints` object representing test specifications. Use `loadConstraints` for this.
- `true_theta` (optional) true theta values to use in simulation. Either `true_theta` or `data` must be supplied.
data (optional) a matrix containing item response data to use in simulation. Either true_theta or data must be supplied.

prior (optional) density at each config@theta_grid to use as prior. Must be a length-nq vector or a nj * nq matrix. This overrides prior_dist and prior_par in the config. prior and prior_par cannot be used simultaneously.

prior_par (optional) normal distribution parameters c(mean, sd) to use as prior. Must be a length-nq vector or a nj * nq matrix. This overrides prior_dist and prior_par in the config. prior and prior_par cannot be used simultaneously.

exclude (optional) a list containing item names in $i and set names in $s to exclude from selection for each participant. The length of the list must be equal to the number of participants.

include_items_for_estimation
  (optional) an examinee-wise list containing:
  - administered_item_pool items to include in theta estimation as item_pool object.
  - administered_item Resp item responses to include in theta estimation.

force_solver if TRUE, do not check whether the solver is one of recommended solvers for complex problems (set-based assembly, partitioning). (default = FALSE)

session (optional) used to communicate with Shiny app TestDesign.

seed (optional) used to perform data generation internally.

Value

Shadow returns an output_Shadow_all object containing assembly results.

References


Examples

cfg <- createShadowTestConfig()
true_theta <- rnorm(1)
solution <- Shadow(cfg, constraints_science, true_theta)
solution@output
Extension of show() for objects in TestDesign package

Description

Extension of show() for objects in TestDesign package

Usage

## S4 method for signature 'item_1PL'
show(object)

## S4 method for signature 'item_2PL'
show(object)

## S4 method for signature 'item_3PL'
show(object)

## S4 method for signature 'item_PC'
show(object)

## S4 method for signature 'item_GPC'
show(object)

## S4 method for signature 'item_GR'
show(object)

## S4 method for signature 'item_pool'
show(object)

## S4 method for signature 'item_pool_cluster'
show(object)

## S4 method for signature 'item_attrib'
show(object)

## S4 method for signature 'st_attrib'
show(object)

## S4 method for signature 'constraints'
show(object)

## S4 method for signature 'summary_item_pool'
show(object)

## S4 method for signature 'summary_item_attrib'
show(object)
## S4 method for signature 'summary_st_attrib'
show(object)

## S4 method for signature 'summary_constraints'
show(object)

## S4 method for signature 'config_Static'
show(object)

## S4 method for signature 'config_Shadow'
show(object)

## S4 method for signature 'output_Static'
show(object)

## S4 method for signature 'output_Shadow'
show(object)

## S4 method for signature 'output_Shadow_all'
show(object)

## S4 method for signature 'summary_output_Static'
show(object)

## S4 method for signature 'summary_output_Shadow_all'
show(object)

## S4 method for signature 'exposure_rate_plot'
show(object)

### Arguments

- **object**: an object to display.

### Description

The `simResp` function is used to simulate item response data.

### Usage

```r
simResp(object, theta)
```

```r
## S4 method for signature 'item_1PL,numeric'
```
simResp(object, theta)

## S4 method for signature 'item_1PL,matrix'
simResp(object, theta)

## S4 method for signature 'item_2PL,numeric'
simResp(object, theta)

## S4 method for signature 'item_2PL,matrix'
simResp(object, theta)

## S4 method for signature 'item_3PL,numeric'
simResp(object, theta)

## S4 method for signature 'item_3PL,matrix'
simResp(object, theta)

## S4 method for signature 'item_PC,numeric'
simResp(object, theta)

## S4 method for signature 'item_PC,matrix'
simResp(object, theta)

## S4 method for signature 'item_GPC,numeric'
simResp(object, theta)

## S4 method for signature 'item_GPC,matrix'
simResp(object, theta)

## S4 method for signature 'item_GR,numeric'
simResp(object, theta)

## S4 method for signature 'item_GR,matrix'
simResp(object, theta)

## S4 method for signature 'item_pool,numeric'
simResp(object, theta)

## S4 method for signature 'item_pool,matrix'
simResp(object, theta)

## S4 method for signature 'item_pool_cluster,numeric'
simResp(object, theta)

## S4 method for signature 'item_pool_cluster,list'
simResp(object, theta)
Arguments

object: an item or an item_pool object.
theta: theta values to use.

Details

notations
- \( n_q \) denotes the number of theta values.
- \( n_i \) denotes the number of items in the item_pool object.

Value

item object: simResp returns a length \( n_q \) vector containing simulated item response data.
item_pool object: simResp returns a \((n_q, n_i)\) matrix containing simulated item response data.

References


Examples

```r
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5 <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
```
item_6 <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

sim_item_1 <- simResp(item_1, seq(-3, 3, 1))
sim_item_2 <- simResp(item_2, seq(-3, 3, 1))
sim_item_3 <- simResp(item_3, seq(-3, 3, 1))
sim_item_4 <- simResp(item_4, seq(-3, 3, 1))
sim_item_5 <- simResp(item_5, seq(-3, 3, 1))
sim_item_6 <- simResp(item_6, seq(-3, 3, 1))
sim_pool <- simResp(itempool_science, seq(-3, 3, 1))

---

Class `simulation_data_cache`: data cache for Shadow()

### Description

`simulation_data_cache` is an S4 class to represent data cache for Shadow().

### Slots

- **item_pool**: the `item_pool` object.
- **theta_grid**: the theta grid to use as quadrature points.
- **prob_grid**: the list containing item response probabilities at theta quadratures.
- **info_grid**: the matrix containing item information values at theta quadratures.
- **max_info**: the maximum value of `info_grid`.
- **true_theta** (optional): the true theta values.
- **response_data** (optional): the matrix containing item responses.

---

**Split**

Split an item pool into partitions

### Description

`Split` is a function to split a pool into multiple parallel tests or pools. When constructing parallel tests, each test is constructed to satisfy all constraints. When constructing parallel pools, each pool is constructed so that it contains a test that satisfies all constraints.
Usage

```r
Split(
  config,
  constraints,
  n_partition,
  partition_type,
  partition_size_range = NULL,
  force_solver = FALSE
)
```

## S4 method for signature 'config_Static'

```r
Split(
  config,
  constraints,
  n_partition,
  partition_type,
  partition_size_range = NULL,
  force_solver = FALSE
)
```

Arguments

- `config` a `config_Static` object. Use `createStaticTestConfig` for this.
- `constraints` a `constraints` object representing test specifications. Use `loadConstraints` for this.
- `n_partition` the number of partitions to create.
- `partition_type` test to create tests, or pool to create pools.
- `partition_size_range` (optional) two integer values for the desired range for the size of a partition. Has no effect when `partition_type` is test. For discrete item pools, the default partition size is (pool size / number of partitions). For set-based item pools, the default partition size is (pool size / number of partitions) +/- smallest set size.
- `force_solver` if TRUE, do not check whether the solver is one of recommended solvers for complex problems (set-based assembly, partitioning). (default = FALSE)

Value

- `partition` returns an `output_Split` object containing item/set indices of created tests/pools.

Examples

```r
## Not run:
config <- createStaticTestConfig(MIP = list(solver = "RSYMPHONY"))
constraints <- constraints_science[1:10]

solution <- Split(config, constraints, n_partition = 4, partition_type = "test")
plot(solution)
solution <- Split(config, constraints, n_partition = 4, partition_type = "pool")
```
plot(solution)
## End(Not run)

### Static Run fixed-form test assembly

**Description**

*Static* is a test assembly function to perform fixed-form test assembly based on the generalized shadow-test framework.

**Usage**

```r
Static(config, constraints, force_solver = FALSE)
```

```r
## S4 method for signature 'config_Static'
Static(config, constraints, force_solver = FALSE)
```

**Arguments**

- `config`: a `config_Static` object. Use `createStaticTestConfig` for this.
- `constraints`: a `constraints` object representing test specifications. Use `loadConstraints` for this.
- `force_solver`: if TRUE, do not check whether the solver is one of recommended solvers for complex problems (set-based assembly, partitioning). (default = FALSE)

**Value**

*Static* returns a `output_Static` object containing the selected items.

**References**


**Examples**

```r
config_science <- createStaticTestConfig(
  list(
    method = "MAXINFO",
    target_location = c(-1, 1)
  )
)
solution <- Static(config_science, constraints_science)
```
**st_attrib-class**

*Load set/stimulus/passage attributes*

**Description**

`loadStAttrib` is a data loading function to create an `st_attrib` object. `loadStAttrib` can read stimulus attributes a `data.frame` or a `.csv` file.

**Usage**

`loadStAttrib(object, item_attrib, file = NULL)`

**Arguments**

- **object**
  - set attributes. Can be a `data.frame` or the file path of a `.csv` file. The content should at least include the column ‘STID’ referring to the column ‘STID’ in the data slot of the `item_attrib` object.

- **item_attrib**
  - an `item_attrib` object. Use `loadItemAttrib` for this.

- **file**
  - (deprecated) use `object` argument instead.

**Value**

`loadStAttrib` returns a `st_attrib` object.

- `data` a `data.frame` containing stimulus attributes.

**See Also**

`dataset_reading` for examples.

**Examples**

```r
## Read from data.frame:
itempool_reading <- loadItemPool(itempool_reading_data)
itemattrib_reading <- loadItemAttrib(itemattrib_reading_data, itempool_reading)
stimattrib_reading <- loadStAttrib(stimattrib_reading_data, itemattrib_reading)

## Read from file: write to tempdir() for illustration and clean afterwards
f <- file.path(tempdir(), "stimattrib_reading.csv")
write.csv(stimattrib_reading_data, f, row.names = FALSE)
stimattrib_reading <- loadStAttrib(f, itemattrib_reading)
file.remove(f)

## TestDesign 1.1.0 - Deprecated arguments
## Not run:
loadStAttrib(object = "satt.csv", item_attrib) # is equivalent to
loadStAttrib(file = "satt.csv", item_attrib) # pre 1.1.0

## End(Not run)
```
Basic functions for stimulus attribute objects

Usage

## S4 method for signature `'st_attrib,numeric'`
x[i, j, ..., drop = TRUE]

## S4 method for signature `'st_attrib'`
dim(x)

## S4 method for signature `'st_attrib'`
colnames(x)

## S4 method for signature `'st_attrib'`
rownames(x)

## S4 method for signature `'st_attrib'`
names(x)

## S4 method for signature `'st_attrib'`
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

Arguments

x a `st_attrib` object.
i, j indices to use in subsetting.
... not used, exists for compatibility.
drop not used, exists for compatibility.
row.names not used, exists for compatibility.
optional not used, exists for compatibility.

Examples

x <- stimattrib_reading
x[1:10]
dim(x)
ncol(x)
nrow(x)
colnames(x)
rownames(x)
names(x)
as.data.frame(x)

### Summary

**Description**

Extension of summary() for objects in TestDesign package

**Usage**

```r
## S4 method for signature 'item_pool'
summary(object)

## S4 method for signature 'item_attrib'
summary(object)

## S4 method for signature 'st_attrib'
summary(object)

## S4 method for signature 'constraints'
summary(object)

## S4 method for signature 'output_Static'
summary(object, simple = FALSE)

## S4 method for signature 'output_Shadow_all'
summary(object, simple = FALSE)
```

**Arguments**

- **object**: an object to summarize.
- **simple**: if TRUE, do not print constraints. (default = FALSE)

**Examples**

```r
summary(itempool_science)
solution <- Static(cfg, constraints_science)
summary(solution)
solution <- Static(cfg, constraints_science)
solution <- Static(cfg, constraints_science)
summary(solution)
```
summarize-classes

summarize(solution, simple = TRUE)

summary-classes

Summary classes

Description

Summary classes

test-class

Class 'test': data cache for simulations

Description

TestDesign

Open TestDesign app

Description

TestDesign is a caller function to open the Shiny interface of TestDesign package.

Usage

TestDesign()
Examples

```r
## Not run:
if (interactive()) {
  TestDesign()
}
## End(Not run)
```

---

**testSolver**  
*Test solver*

**Description**

Test solver

**Usage**

`testSolver(solver)`

**Arguments**

- `solver` a solver package name. Accepts `lpSolve`, `Rsymphony`, `gurobi`, `Rglpk`.

**Value**

empty string `""` if solver works. A string containing error messages otherwise.

---

**test_cluster-class**  
*Class `test_cluster`*: data cache for simulations

**Description**

`test_cluster` is an S4 class to represent data cache for running simulations. Despite the name, this class does not represent a series of tests and is not related to a series of tests. That is, test length is not stored in this class. This class is only kept for backwards compatibility.

**Slots**

- `nt` the number of `test` objects in this cluster.
- `tests` the list containing `test` objects.
- `names` test ID strings for each `test` object.
**test_operators**

---

**Basic operators for test objects**

**Description**

Create a subset of a `test` object.

**Usage**

```r
subsetTest(x, i = NULL)
```

```r
# S4 method for signature 'test,ANY'
theta_EAP()
```

**Arguments**

- `x` a `test` object.
- `i` item indices to use in subsetting.
- `j, drop, ...` not used, exists for compatibility.

---

**theta_EAP**

(C++) Calculate a theta estimate using EAP (expected a posteriori) method

**Description**

`theta_EAP()` and `theta_EAP_matrix()` are functions for calculating a theta estimate using EAP (expected a posteriori) method.

**Usage**

```r
theta_EAP(theta_grid, item_parm, resp, ncat, model, prior, prior_parm)
```

```r
theta_EAP_matrix(theta_grid, item_parm, resp, ncat, model, prior, prior_parm)
```

**Arguments**

- `theta_grid` theta quadrature points.
- `item_parm` a matrix containing item parameters.
- `resp` responses on each item. Must be a vector for `theta_EAP()`, and a matrix for `theta_EAP_matrix()`. Each row should represent an examinee.
- `ncat` a vector containing the number of response categories of each item.
- `model` a vector indicating item models of each item, using
  - `1`: 1PL model
• 2: 2PL model
• 3: 3PL model
• 4: PC model
• 5: GPC model
• 6: GR model

prior an integer indicating the type of prior distribution, using
• 1: normal distribution
• 2: uniform distribution

prior_parm a vector containing parameters for the prior distribution.

Details

theta_EAP() and theta_EAP_matrix() are designed for multiple items.

theta_EAP() is designed for one examinee, and theta_EAP_matrix() is designed for multiple examinees.

Currently supports unidimensional models.

Examples

```r
# item parameters
item_parm <- matrix(c(
  1, NA, NA, 
  1, 2, NA, 
  1, 2, 0.25, 
  0, 1, NA, 
  2, 0, 1, 
  2, 0, 2),
  nrow = 6,
  byrow = TRUE
)
ncat <- c(2, 2, 2, 3, 3, 3)
model <- c(1, 2, 3, 4, 5, 6)

# simulate response
item_parm <- as.data.frame(item_parm)
item_parm <- cbind(101:106, 1:6, item_parm)
pool <- loadItemPool(item_parm)
true_theta <- seq(-3, 3, 1)
resp <- simResp(pool, true_theta)
theta_grid <- matrix(seq(-3, 3, .1), , 1)
theta_EAP(theta_grid, pool@ipar, resp[1, ], ncat, model, 1, c(1, 2))
theta_EAP_matrix(theta_grid, pool@ipar, resp, ncat, model, 1, c(1, 2))
```
theta_EB

(C++) Calculate a theta estimate using EB (Empirical Bayes) method

Description

theta_EB_single() and theta_EB() are functions to calculate a theta estimate using EB (Empirical Bayes) method.

Usage

theta_EB(
  nx,
  theta_init,
  theta_prop,
  item_parm,
  resp,
  ncat,
  model,
  prior,
  prior_parm
)

theta_EB_single(
  nx,
  theta_init,
  theta_prop,
  item_parm,
  resp,
  ncat,
  model,
  prior,
  prior_parm
)

Arguments

nx                     the number of MCMC draws.
theta_init             the initial estimate to use.
theta_prop             the SD of the proposal distribution.
item_parm              a matrix containing item parameters. Each row should represent an item.
resp                   a vector containing responses on each item.
ncat                   a vector containing the number of response categories of each item.
model                  a vector indicating item models of each item, using
  • 1: 1PL model
  • 2: 2PL model
theta_FB

(C++) Calculate a theta estimate using FB (Full Bayes) method

Description

theta_FB_single() and theta_FB() are functions to calculate a theta estimate using FB (Full Bayes) method.
theta_FB

Usage

theta_FB(
  nx, theta_init, theta_prop, items_list, item_init, resp, ncat, model, prior, prior_parm
)

theta_FB_single(
  nx, theta_init, theta_prop, item_mcmc, item_init, resp, ncat, model, prior, prior_parm
)

Arguments

nx the number of MCMC draws.
theta_init the initial estimate to use.
theta_prop the SD of the proposal distribution.
item_init item parameter estimates. Must be a vector for theta_FB_single(), and a matrix for theta_FB().
resp a vector containing responses on each item.
ncat a vector containing the number of response categories of each item.
model a vector indicating item models of each item, using
  • 1: 1PL model
  • 2: 2PL model
  • 3: 3PL model
  • 4: PC model
  • 5: GPC model
  • 6: GR model
prior an integer indicating the type of prior distribution, using
  • 1: normal distribution
• 2: uniform distribution

prior_parm: a vector containing parameters for the prior distribution.
item_mcmc, items_list: sampled item parameters. Must be a matrix for theta_FB_single(), and a list of matrices for theta_FB().

Details

theta_FB_single() is designed for one item, and theta_FB() is designed for multiple items. Currently supports unidimensional models.

description

toggleConstraints is a function to toggle individual constraints in a constraints object.

Usage

toggleConstraints(object, on = NULL, off = NULL)

Arguments

object: a constraints object from loadConstraints.

on: constraint indices to mark as active. Also accepts character IDs.

off: constraint indices to mark as inactive. Also accepts character IDs.

Value

toggleConstraints returns the updated constraints object.

Examples

constraints_science2 <- toggleConstraints(constraints_science, off = 32:36)
constraints_science3 <- toggleConstraints(constraints_science2, on = 32:36)
constraints_science4 <- toggleConstraints(constraints_science, off = "C32")
Index

* datasets
  dataset_bayes, 38
  dataset_fatigue, 38
  dataset_reading, 39
  dataset_science, 40
  +.item_pool (item_pool-operators), 57
  -.item_pool (item_pool-operators), 57
  ==.item_pool (item_pool-operators), 57
  ==.item_pool_cluster (makeItemPoolCluster), 65
  [],constraints,numeric,ANY,ANY-method (constraints-operators), 37
  [],constraints,numeric-method (constraints-operators), 37
  [],item_attrib,numeric,ANY,ANY-method (item_attrib-operators), 55
  [],item_attrib,numeric-method (item_attrib-operators), 55
  [],item_pool,numeric,ANY,ANY-method (item_pool-operators), 57
  [],item_pool,numeric-method (item_pool-operators), 57
  [],st_attrib,numeric,ANY,ANY-method (st_attrib-operators), 97
  [],st_attrib,numeric-method (st_attrib-operators), 97
  [],test,ANY-method (test_operators), 101
  [],test,numeric,ANY,ANY-method (test_operators), 101

  a_to_alpha, 5
  app, 4, 4
  array_dirinfo_m_2pl (info_item), 49
  array_dirinfo_m_3pl (info_item), 49
  array_dirinfo_m_gpc (info_item), 49
  array_dirinfo_m_gr (info_item), 49
  array_e_1pl (e_item), 42
  array_e_2pl (e_item), 42
  array_e_3pl (e_item), 42
  array_e_gpc (e_item), 42
  array_e_gr (e_item), 42
  array_e_pc (e_item), 42
  array_h_1pl (h_item), 47
  array_h_2pl (h_item), 47
  array_h_3pl (h_item), 47
  array_h_gpc (h_item), 47
  array_h_gr (h_item), 47
  array_h_pc (h_item), 47
  array_info_1pl (info_item), 49
  array_info_2pl (info_item), 49
  array_info_3pl (info_item), 49
  array_info_gpc (info_item), 49
  array_info_gr (info_item), 49
  array_info_m_2pl (info_item), 49
  array_info_m_3pl (info_item), 49
  array_info_m_gpc (info_item), 49
  array_info_m_gr (info_item), 49
  array_info_pc (info_item), 49
  array_j_1pl (j_item), 59
  array_j_2pl (j_item), 59
  array_j_3pl (j_item), 59
  array_j_gpc (j_item), 59
  array_j_gr (j_item), 59
  array_j_pc (j_item), 59
  array_p_1pl (p_item), 82
  array_p_2pl (p_item), 82
  array_p_3pl (p_item), 82
  array_p_gpc (p_item), 82
  array_p_gr (p_item), 82
  array_p_m_2pl (p_item), 82
  array_p_m_3pl (p_item), 82
  array_p_m_gpc (p_item), 82
  array_p_m_gr (p_item), 82
  array_p_m_pc (p_item), 82
  array_thisdirinfo_m_2pl (info_item), 49
  array_thisdirinfo_m_3pl (info_item), 49
  array_thisdirinfo_m_gpc (info_item), 49
  array_thisdirinfo_m_gr (info_item), 49
  as.data.frame,item_attrib-method
INDEX

(calcJacobian), 14
(calcProb-methods), 19
calcJacobian, item_pool, numeric, numeric-method
calcProb, item_GPC, matrix-method
(calcJacobian), 14
(calcProb-methods), 19
calcJacobian, item_pool, numeric-method
calcProb, item_GPC, numeric-method
(calcJacobian), 14
(calcProb-methods), 19
calcJacobian, item_pool_cluster, numeric, list-method
calcProb, item_GR, matrix-method
(calcJacobian), 14
(calcProb-methods), 19
calcJacobian, item_pool_cluster, numeric-method
calcProb, item_GR, numeric-method
(calcJacobian), 14
(calcProb-methods), 19
calcLocation, 16, 17
calcProb, item_PC, matrix-method
(calcProb-methods), 19
calcLogLikelihood, 18, 18
calcProb, item_GPC, numeric-method
(calcProb-methods), 19
calcLocation, item_1PL-method
calcProb, item_PC, numeric-method
(calcProb-methods), 19
calcLocation, item_2PL-method
calcProb, item_pool, matrix-method
(calcProb-methods), 19
calcLocation, item_3PL-method
calcProb, item_pool, numeric-method
(calcProb-methods), 19
calcLocation, item_GPC-method
calcProb, item_pool_cluster, numeric-method
(calcProb-methods), 19
calcLocation, item_GR-method
calcProb-methods, 19
calcLocation, item_PC-method
checkConstraints, 29
calcLocation, item_pool-method
colnames, item_attrib-method
(calcProb-methods), 19
colnames, st_attrib-method
(calcProb-methods), 19
combineConstraints
constraints-operators, 37
calcLogLikelihood, item_pool, matrix-method
combineItemPool (item_pool-operators), 57
calcLogLikelihood, item_pool, matrix, matrix-method
calcLogLikelihood, item_pool-operators, 57
calcLogLikelihood, item_pool, matrix, numeric-method
config_Shadow, 29, 74, 86, 87
(calcLogLikelihood), 18
calcLogLikelihood-class, 29
calcLogLikelihood, item_pool, matrix-method
config_Shadow-class, 29
calcLogLikelihood, item_pool, numeric-method
config_Static, 33, 34, 75, 86, 94, 95
(calcLogLikelihood), 18
calcLogLikelihood, item_pool-class, 33
config_Static-class, 33
calcLogLikelihood, item_pool, numeric-method
constraint-class, 35
(calcLogLikelihood), 18
constraint-class, 36
calcProb, 10, 17, 19-21
constraints-operators, 37
calcProb, item_1PL-method
calcProb-methods, 19
(calcProb-methods), 19
calcProb, item_1PL, numeric-method
calcProb-methods, 19
(calcProb-methods), 19
calcProb, item_2PL-method
calcProb-methods, 19
(calcProb-methods), 19
calcProb, item_2PL, numeric-method
calcProb-methods, 19
(calcProb-methods), 19
calcProb, item_3PL-method
calcProb-methods, 19
(calcProb-methods), 19
calcProb, item_3PL, numeric-method
calcProb-methods, 19
(calcProb-methods), 19
calcProb, item_3PL, numeric-method
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
(calcProb-methods), 19
createShadowTestConfig
(constraints_science_data)
createStaticTestConfig
data.frame
(createStaticTestConfig)
detectBestSolver
e_1pl
dirinfo_m_gr
dirinfo_m_3pl
dirinfo_m_2pl
dim,item_attrib-method
dim,st_attrib-method
dirinfo_m_2pl
(dirinfo_m_3pl)
dirinfo_m_gpc
(dirinfo_m_3pl)
dirinfo_m_gr
(e_1pl)
e_2pl
e_3pl
e_gpc
e_gr
e_item
e_item
EAP
eap
EAP,item_pool-method
EAP,test-method
EAP,test_cluster-method
find_segment
getSolution
(getSolution)
getSolution,config_Shadow-class
getSolution,config_Static-class
getSolution,dataset_science
dataset_science
dataset_reading
dataset_fatigue
dataset_bayes
dataset_parc
dataset_gpc
dataset_gr
dataset_3pl
dataset_2pl
dataset_1pl
dataset_1pl
info_item
info_m_gr
info_m_gpc
info_m_3pl
info_m_2pl
info_m_1pl
info_item
info_gr
info_gpc
info_h
e_item
h_1pl
h_2pl
h_3pl
h_gr
e_item
h_item
h_m_2pl
h_m_3pl
h_m_gpc
h_m_gr
e_item
h_pc
info_item
iparPosteriorSample
item
iitem
item_1PL
item_2PL
item_3PL
item.attrib
item.attrib-method
item.attrib-operators
item_class
item_PC
item_GR
item_GPC
item_1PL
item_2PL
item_3PL
item_attrib
item_attrib-method
item_attrib-class
item_attrib-operators
item_1PL-class
item_2PL-class
item_3PL-class
item.attrib-class
item.attrib-operators
item_pool, 6, 8, 10, 12, 13, 15, 17, 18, 20, 21, 36, 38–41, 53, 55–59, 62–64, 66, 67, 70, 71, 74, 75, 79, 88, 92, 93, 99
item_pool-class, 56
item_pool-operators, 57
item_pool_cluster, 59, 65, 66, 68
item_pool_cluster-class, 59
itemattrib_bayes (item_pool_cluster-class), 38
itemattrib_bayes_data (item_pool_cluster-class), 38
itemattrib_bayes (item_pool_cluster), 38
itemattrib_fatigue (item_pool_cluster), 38
itemattrib_fatigue_data (item_pool_cluster), 38
itemattrib_science (item_pool_cluster), 38
itemattrib_science_data (item_pool_cluster), 38
itemattrib_reading (item_pool_cluster), 38
itemattrib_reading_data (item_pool_cluster), 38
itemattrib_fatigue (item_pool_cluster-class), 38
itemattrib_fatigue_data (item_pool_cluster-class), 38
itemattrib_science (item_pool_cluster-class), 38
itemattrib_science_data (item_pool_cluster-class), 38
itemattrib_reading (item_pool_cluster-class), 38
itemattrib_reading_data (item_pool_cluster-class), 38
loadConstraints, 29, 62, 63, 86, 87, 94, 95, 106
loadItemAttrib, 54, 55, 62, 96
loadItemAttrib (item_attrib-class), 54
loadItemPool, 55, 57, 62, 63, 64
loadStAttrib, 62, 96
loadStAttrib (st_attrib-class), 96
logitHyperPars, 65, 65
makeItemPoolCluster, 65
makeItemPoolCluster (item_pool-method)
makeItemPoolCluster (item_pool-method), 65
makeSimulationDataCache, 66, 66, 67
makeSimulationDataCache (item_pool-method)
makeSimulationDataCache (item_pool-method), 66
makeTest, 67, 67
makeTest (item_pool-method)
makeTest (item_pool-method), 67
makeTestCluster, 68, 68
makeTestCluster (item_pool_cluster-class)
makeTestCluster (item_pool_cluster-class), 68
makeTestCluster (item_pool_cluster-class, numeric, list-method)
makeTestCluster (item_pool_cluster-class, numeric, list-method)
makeTestCluster (item_pool_cluster-class, numeric, numeric-method)
makeTestCluster (item_pool_cluster-class, numeric, numeric-method)
MLE (mle), 68
mle, 68, 68, 70
mle (item_pool-method) (mle), 68
MLE, test-method (mle), 68
MLE, test_cluster-method (mle), 68
mlef, 71, 71, 72
mlef, item_pool-method (mlef), 71
names, item_attrib-method
names, item_attrib-method
names, st_attrib-method
(names, item_attrib-method), 55
(names, st_attrib-method), 97
OAT, 4
OAT (app), 4
output_Shadow, 45, 73, 74, 79
output_Shadow-class, 73
output_Shadow-all, 74, 79, 80, 88
output_Shadow-all-class, 74
output_Split, 75, 94
output_Split-class, 75
output_Static, 45, 75, 79, 95
output_Static-class, 75
p_1pl (p_item), 82
p_2pl (p_item), 82
p_3pl (p_item), 82
INDEX

p_gpc(p_item), 82
p_gr(p_item), 82
p_item, 82
p_m_2pl(p_item), 82
p_m_3pl(p_item), 82
p_m_gpc(p_item), 82
p_m_gr(p_item), 82
p_pc(p_item), 82
partition, 94
plot, constraints-method (plot), 76
plot, item_pool-method (plot), 76
plot, output_Shadow-method (plot), 76
plot, output_Split-method (plot), 76
plot, output_Static-method (plot), 76
print, 80
print, config_Shadow-method (print), 80
print, config_Static-method (print), 80
print, constraints-method (print), 80
print, exposure_rate_plot-method (print), 80
print, item_1PL-method (print), 80
print, item_2PL-method (print), 80
print, item_3PL-method (print), 80
print, item_attrib-method (print), 80
print, item_GPC-method (print), 80
print, item_GR-method (print), 80
print, item_PC-method (print), 80
print, item_pool-method (print), 80
print, output_Shadow-method (print), 80
print, output_Shadow_all-method (print), 80
print, output_Static-method (print), 80
print, st_attrib-method (print), 80
print, summary_constraints-method (print), 80
print, summary_item_attrib-method (print), 80
print, summary_item_pool-method (print), 80
print, summary_output_Shadow_all-method (print), 80
print, summary_output_Static-method (print), 80
print, summary_st_attrib-method (print), 80
print.default, 82
RE, 85
resp_fatigue_data(dataset_fatigue), 38
rlnorm, 62
rlogitnorm, 65
RMSE, 85
rownames, item_attrib-method
(item_attrib-operators), 55
rownames, st_attrib-method
(st_attrib-operators), 97
runAssembly, 86, 86
Shadow, 6, 29, 31, 63, 66, 86, 87, 87, 88
Shadow, config_Shadow-method (Shadow), 87
show, 89
show, config_Shadow-method (show), 89
show, config_Static-method (show), 89
show, constraints-method (show), 89
show, exposure_rate_plot-method (show), 89
show, item_1PL-method (show), 89
show, item_2PL-method (show), 89
show, item_3PL-method (show), 89
show, item_attrib-method (show), 89
show, item_GPC-method (show), 89
show, item_GR-method (show), 89
show, item_PC-method (show), 89
show, item_pool-method (show), 89
show, item_pool_cluster-method (show), 89
show, output_Shadow-method (show), 89
show, output_Shadow_all-method (show), 89
show, output_Static-method (show), 89
show, pool_cluster-method (show), 89
show, st_attrib-method (show), 89
show, summary_constraints-method (show), 89
show, summary_item_attrib-method (show), 89
show, summary_item_pool-method (show), 89
show, summary_output_Shadow_all-method (show), 89
show, summary_output_Static-method (show), 89
simResp, 90, 90, 92
simResp, item_1PL_matrix-method (simResp), 90
simResp, item_1PL_numeric-method (simResp), 90
simResp,item_2PL,matrix-method (simResp), 90
simResp,item_2PL,numeric-method (simResp), 90
simResp,item_3PL,matrix-method (simResp), 90
simResp,item_3PL,numeric-method (simResp), 90
simResp,item_GPC,matrix-method (simResp), 90
simResp,item_GPC,numeric-method (simResp), 90
simResp,item_GR,matrix-method (simResp), 90
simResp,item_GR,numeric-method (simResp), 90
simResp,item_PC,matrix-method (simResp), 90
simResp,item_PC,numeric-method (simResp), 90
simResp,item_pool,matrix-method (simResp), 90
simResp,item_pool,numeric-method (simResp), 90
simResp,item_pool_cluster,list-method (simResp), 90
simResp,item_pool_cluster,numeric-method (simResp), 90
simulation_data_cache, 66, 93, 99
simulation_data_cache-class, 93
Split, 93, 93
Split,config_Static-method (Split), 93
st_attrib, 6, 36, 39, 62, 96, 97
st_attrib-class, 96
st_attrib-operators, 97
Static, 6, 34, 63, 86, 95, 95
Static,config_Static-method (Static), 95
stimattrib_reading (dataset_reading), 39
stimattrib_reading_data (dataset_reading), 39
subsetConstraints (constraints-operators), 37
subsetItemPool (item_pool-operators), 57
subsetTest (test_operators), 101
summary, 98
summary,constraints-method (summary), 98
summary,item_attrib-method (summary), 98
summary,item_pool-method (summary), 98
summary,output_Shadow_all-method (summary), 98
summary,output_Static-method (summary), 98
summary,st_attrib-method (summary), 98
summary-classes, 99
summary_constraints-class (summary-classes), 99
summary_item_attrib-class (summary-classes), 99
summary_item_pool-class (summary-classes), 99
summary_output_Shadow_all-class (summary-classes), 99
summary_output_Static-class (summary-classes), 99
summary_st_attrib-class (summary-classes), 99
test, 42, 67, 99–101
test-class, 99
test_cluster, 42, 68, 100
test_cluster-class, 100
test_operators, 101
testSolver, 100
theta_EAP, 101
theta_EAP_matrix (theta_EAP), 101
theta_EB, 103
theta_EB_single (theta_EB), 103
theta_FB, 104
theta_FB_single (theta_FB), 104
thisdirinfo_m_2pl (info_item), 49
thisdirinfo_m_3pl (info_item), 49
thisdirinfo_m_gpc (info_item), 49
toggleConstraints, 106, 106