Multidimensional Computerized Adaptive Testing with mirtCAT

Phil Chalmers

March 29, 2017

Unidimensional and Multidimensional CATs

Computerized Adaptive Testing

Administer tests by a computer using some graphical user interface (GUI). Items are dynamically selected in an adaptive way given the current latent trait estimate(s), $\hat{\theta}$.

Why Computerized Adaptive Testing?

- Uninformative items are progressively avoided (e.g., high-ability participants should not receive easy items)
- More precise estimates in shorter time/number of items
- Helps to avoid fatigue effects
- Better item security and control (e.g., test items can be different yet have similar measurement properties)
- Better/more modern presentation quality (audio, video, interactive content, etc)

Multidimensional CATs

CATs have mainly been applied using unidimensional item banks. However, previous benefits are also relevant to multidimensional CATs, though some other complications/benefits naturally arise.

- Pro: Correlated traits measured simultaneously; leads to better efficiency
- Pro: Items can be selected which focus on improving precision of some traits more than others
- Pro: Item interdependencies are controlled (e.g., testlets)
- Con: Item selection algorithms not as well developed as unidimensional IRT, and can become situation specific
- Con: More demanding computationally

mirtCAT software in R

mirtCAT: Computerized Adaptive Testing with Multidimensional Item Response Theory

Provides tools to generate an HTML interface for creating adaptive and non-adaptive educational and psychological tests using the shiny package. Suitable for applying unidimensional and multidimensional computerized adaptive tests (CAT) using item response theory methodology and for creating simple questionnaires forms to collect response data directly in R. Additionally, optimal test designs (e.g., "shadow testing") are supported for tests which contain a large number of item selection constraints. Finally, package contains tools useful for performing Monte Carlo simulations for studying the behavior of computerized adaptive test banks.

Why Spend/Waste Time Building **mirtCAT**?

- R solution to building CAT GUIs was non-existent
- CAT packages in R were only for simulation designs (e.g., **catR**, **catIrt**, **MAT**) with mixed flexibility and efficiency
- Multidimensional models in **MAT** limited to M3PL model
- No package could handle mixed item types (e.g., 3PL and GPCM simultaneously)

Why Spend/Waste Time Building **mirtCAT**?

- R solution to building CAT GUIs was non-existent
- CAT packages in R were only for simulation designs (e.g., **catR**, **catIrt**, **MAT**) with mixed flexibility and efficiency
- Multidimensional models in **MAT** limited to M3PL model
- No package could handle mixed item types (e.g., 3PL and GPCM simultaneously)

At the end of the day, I wanted an interface to collect survey data that was adaptive AND non-adaptive, integrated directly with R, and could be run locally/hosted on servers for remote data collection.

mirtCAT Package

Presentation is largely based on what is found in:

- Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*, 1–38.

However, we'll close off with some new features that are not present in that article.

mirtCAT Package

Really only three main functions in the **mirtCAT** package:

- `mirtCAT()` – execute CAT designs off-line or generate GUIs for real-time applications. Objects returned from this function are of class `'mirtCAT'`
    - Associated S3 methods `plot()`, `summary()`, and `print()`

mirtCAT Package

. . . and the preamble functions:

- `generate.mirt_object()` – generate a **mirt** object from population parameters if the MIRT model was not estimated from empirical data (alternatively, use the `mirt()`/`bfactor()` function with real data)
- `generate_pattern()` – generate potential response pattern for CAT application. Useful for Monte Carlo work, and to test the CAT designs

Setting up CATs with **mirtCAT**

```
mirtCAT(df, mo, ...)
```

- `df` is a `data.frame` object containing question stems, options, answers, and image stem paths
- `mo` is an object defined by **mirt** containing the item/group parameters

Setting up CATs with **mirtCAT**

```
mirtCAT(df, mo, ...)
```

- `df` is a `data.frame` object containing question stems, options, answers, and image stem paths
- `mo` is an object defined by **mirt** containing the item/group parameters

Depending on the purpose, one of these inputs can be missing (i.e., if only `df` supplied a survey GUI is generated; if only `mo` is supplied, an off-line CAT. Supplying both creates the adaptive GUI)

Computerized Adaptive Testing Components

1. (Pre)-Calibrated item bank (a la IRT) for a pool of items
2. Starting rules (initial $\hat{\theta}$'s, pre-CAT data collection, controlled item administration, etc)
3. Item selection algorithm
4. Updating $\hat{\theta}$ algorithm
5. Termination criteria

(Pre)-Calibration of Item Bank

Before beginning with adaptive tests, it should be noted that items could be selected using traditional (i.e., non-IRT or otherwise model-based) methods. For example,

- In sequence (typical pen-and-paper style),
- Random or quasi-random (parallel forms),
- Increasing difficulty (so-called *power tests*) in a CTT sense,
- . . . etc.

(Pre)-Calibration of Item Bank

However,

- Often desirable to select items based on measurement properties of ability (θ)
- Generally only feasible through computerized sessions where computations are performed
- Requires known (or at the very least, accurately estimated) model parameters.

(Pre)-Calibration of Item Bank

Throughout this presentation we will use the unidimensional 3PL

$$P(\theta) = g + \frac{1 - g}{1 + \exp\left[-(a\theta + d)\right]}$$

and multidimensional 3PL model

$$P(\boldsymbol{\theta}) = g + \frac{1 - g}{1 + \exp\left[-(\boldsymbol{a}'\boldsymbol{\theta} + d)\right]},$$

as well as the associated reduced 2PL and 1PL models, for illustration of the concepts. Though of course, other dichotomous and polytomous IRT models can be used in practice.

(Pre)-Calibration of Item Bank

Two approaches to setting up the item bank with **mirtCAT**. The first approach assumes we already have suitable response data for calibrating the parameters. For example, using the ubiquitous the LSAT7 dataset:

```
library(mirt)
dat <- expand.table(LSAT7)
mod <- mirt(dat, 1, itemtype='2PL', verbose=FALSE)
cfs <- coef(mod, simplify = TRUE)$items[,1:2]
```

(Pre)-Calibration of Item Bank

```
print(cfs)
```

```
##                  a1         d
## Item.1 0.9879254 1.8560605
## Item.2 1.0808847 0.8079786
## Item.3 1.7058006 1.8042187
## Item.4 0.7651853 0.4859966
## Item.5 0.7357980 1.8545127
```

(Pre)-Calibration of Item Bank with mirtCAT

The second approach (for Monte Carlo simulations) assumes that we already have calibrated IRT parameters. Here, `generate.mirt_model()` is used (see `?mirt` for item-types).

```
fixed_mod <- generate.mirt_object(cfs,
                                  itemtype = '2PL')
coef(fixed_mod, simplify=TRUE)$items
```

```
##                a1         d g u
## Item.1 0.9879254 1.8560605 0 1
## Item.2 1.0808847 0.8079786 0 1
## Item.3 1.7058006 1.8042187 0 1
## Item.4 0.7651853 0.4859966 0 1
## Item.5 0.7357980 1.8545127 0 1
```

(Pre)-Calibration of Item Bank with mirtCAT

The objects generated from either approach are of class `SingleGroupClass` from the **mirt** package. Because of this, many of the secondary functions in **mirt** can be used to investigate their properties.

- `coef()` for coefficients, including transformed classical variants (e.g., the $a(\theta - b)$ form)
- plotting methods via `plot()` and `itemplot()`
- `fscores()` for computing factor/latent trait prediction score estimates

And so on.

Example

Example 1 – Minimum off-line working example (part 1)

File for this example is located in `Example-1.R`.

Starting Conditions

Before beginning the CAT, several conditions must be setup

- Initialization of $\hat{\boldsymbol{\theta}}$, typically set to the mean vector $\mathbf{0}$
- Prior distributions if Bayesian flavors are included when updating $\hat{\boldsymbol{\theta}}$ or in the item selection algorithms (e.g., Gaussian, uniform, etc; more on this later)
- The criteria to administer items (sequential, randomly, adaptively)
- Minimum number of items to select, initial item to administer, etc

Main Arguments

Arguments to `mirtCAT()` relevant up to this point (with default in brackets) include

- `criteria` – (`'seq'`) criteria to select items (default is sequential; more on this later)
- `start_item` – (`1`) a numeric starting item. Can also be a character similar to `criteria`
- `local_pattern` – (`NULL`) run one or more locally defined response pattern vectors off-line. Mainly used for testing, Monte Carlo simulations, and evaluating the bank of items

Main Arguments

Of particular importance is the `design` object, which will come up often.

- `design` – (`NULL`) a list of design based control parameters for adaptive and non-adaptive tests (around 20 possible inputs which can be changed). E.g., `min_items`, `thetas.start`, etc

Updating Latent Trait Prediction

After the item has been selected, and the new item response has been collected, an improved $\hat{\boldsymbol{\theta}}$ estimate is obtained using a given estimation algorithm. Methods typically rely on some variant of the likelihood/posterior

$$L(\boldsymbol{\theta}) = \prod_i \left[P_i(\boldsymbol{\theta})^{\chi_i} \cdot (1 - P_i(\boldsymbol{\theta}))^{1-\chi_i} \right] \cdot g(\boldsymbol{\theta}),$$

where χ_i is either 0 or 1 for incorrect and correct endorsement, and $g(\boldsymbol{\theta})$ is a prior distribution.

- ML for this response is just the argmax where $\forall\theta\, g(\boldsymbol{\theta}) = c$

Updating Latent Trait Prediction

- Bayesian estimates using the previous function typically use a Gaussian prior in $g(\boldsymbol{\theta})$ in computing the posterior mean (EAP) or mode (MAP) estimates, though other priors are possible (e.g., uniform, bimodal)
    - EAP estimates require numerical integration, and typically are not recommended for multidimensional CATs for that very reason
    - See the `fscores()` function in **mirt** for examples of how to define customized prior distributions
- Other estimators with alternative objective functions are possible, such as weighted likelihood estimates (aka. Jeffrey's prior), EAP for sum-scores, etc

Uncertainty in Predictions

After obtaining updated $\hat{\boldsymbol{\theta}}$ estimates it is prudent to estimate their respective precision/sampling error. This helps to track the behaviour of the CAT session, but also to determine if/when sufficient measurement precision has been reached.

- One suitable approach is to use information about the second-order Taylor-series expansion of the observed log-likelihood/log-posterior distribution (a.k.a., observed information)

$$(-\frac{\partial^2 L}{\partial \hat{\boldsymbol{\theta}} \partial \hat{\boldsymbol{\theta}}'})^{-1} = I(\hat{\boldsymbol{\theta}})^{-1} = ACOV(\hat{\boldsymbol{\theta}})$$

Uncertainty in Predictions

After obtaining updated $\hat{\boldsymbol{\theta}}$ estimates it is prudent to estimate their respective precision/sampling error. This helps to track the behaviour of the CAT session, but also to determine if/when sufficient measurement precision has been reached.

- One suitable approach is to use information about the second-order Taylor-series expansion of the observed log-likelihood/log-posterior distribution (a.k.a., observed information)

$$(-\frac{\partial^2 L}{\partial \hat{\boldsymbol{\theta}} \partial \hat{\boldsymbol{\theta}}'})^{-1} = I(\hat{\boldsymbol{\theta}})^{-1} = ACOV(\hat{\boldsymbol{\theta}})$$

- Square-root of the diagonal elements of $ACOV(\hat{\boldsymbol{\theta}})$ give suitable standard errors for each $\hat{\boldsymbol{\theta}}$ value.

Example (cont.)

Example 1 – Minimum off-line working example (part 2)

File for this example is located in `Example-1.R`.

Optimal Item Selection

The general idea for optimal item selection is that, given $\hat{\theta}_n$, find the next item such that $SE(\hat{\theta}_{n+1})$ would be the lowest possible value. In other words, pick the next item so that the precision of $\hat{\theta}$ is maximized and its sampling variability is minimized.

- This reasoning is the primary inspiration for *maximum Fisher-information*, or *maximum-information* (MI) criterion for short
- In multidimensional models, this simple criteria has multiple variations with different properties

MI Item Selection

For example, in the unidimensional 2PL model Fisher's information is defined as

$$\mathcal{F}(\theta) = a^2 P(\theta) \cdot (1 - P(\theta))$$

In practice, θ is replaced with the current $\hat{\theta}_n$ estimate and evaluated for each item remaining in the item bank.

- In unidimensional models, maximum Fisher information is equivalent to minimum expected error

$$SE(\theta) = \frac{1}{\sqrt{\mathcal{F}(\theta)}}$$

MI Item Selection

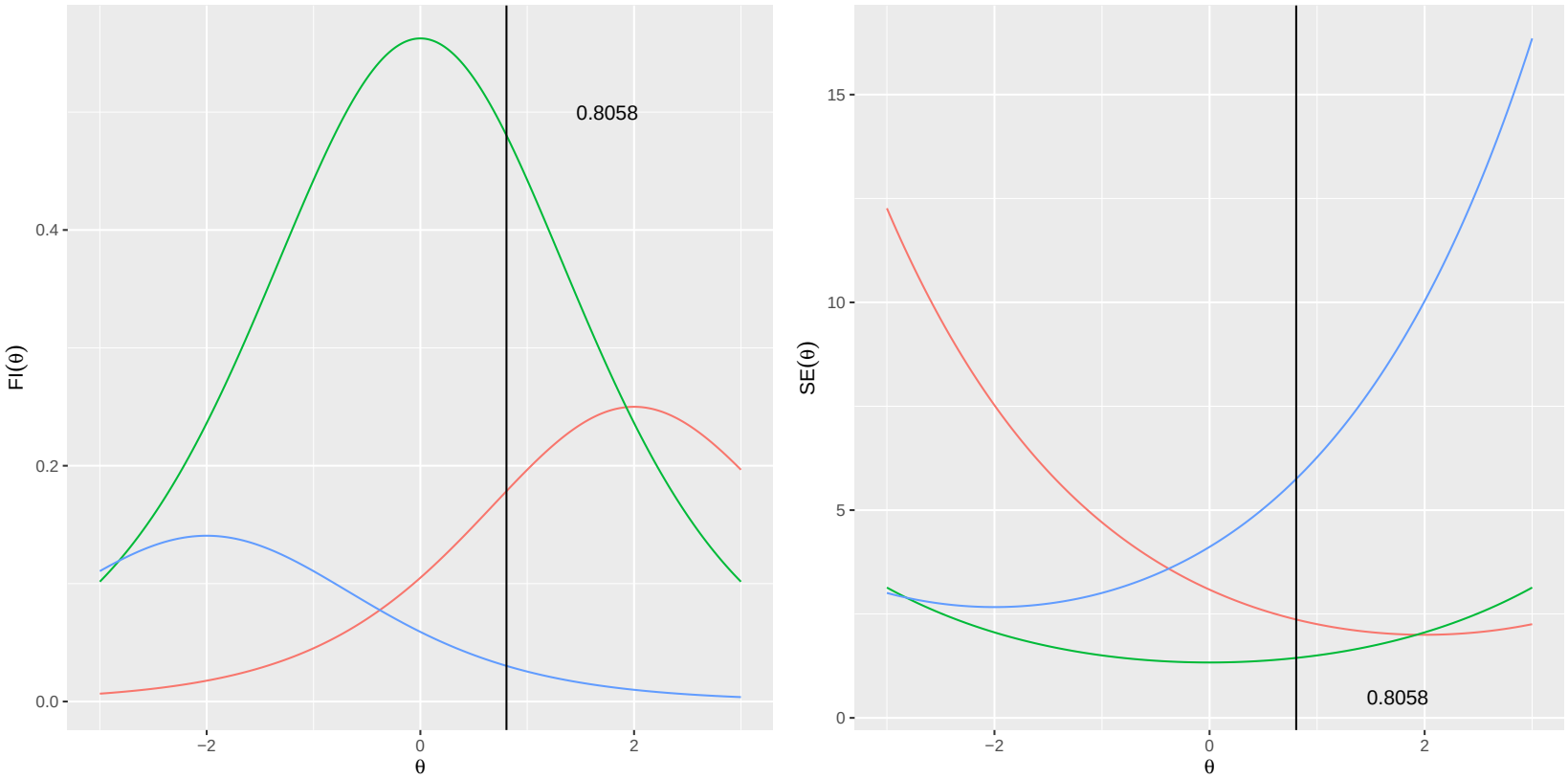

Figure 1: Information and standard errors for three items given $\theta = .8058$

MI Item Selection

- Note that there is an inherent limitation with the MI approach. Namely, assuming $\hat{\theta}_n \approx \theta$ may lead to local item selection problems (particularly early in the CAT where $\hat{\theta}_n$ is very different from θ)
    - This is generally why other selection criteria have been proposed (e.g., Kullback-Leibler information, continuous entropy, MLWI, MPWI, MEI, MEPV, etc)
- Also, large/wide information functions make it more temporally likely an item will be selected given the current $\hat{\theta}$ due to their wider range of being a maximum. Hence, items might be selected too often compared to others in the bank, leading to *exposure control* issues

Item selection Features with **mirtCAT**

Item selection is controlled with the `criteria` input, with several options to choose from:

- Unidimensional: `MI`, `MEPV`, `MLWI`, `MPWI`, `IKLP`, `IKL`, `IKLn`, `IKLPn`
- Multidimensional: `Drule`, `Arule`, `Trule`, `Wrule`, `Erule`, and posterior weighted variants (e.g., `DPrule`)
- Both: `KL`, `KLn`, `seq`, `random`

Main Arguments

Some more arguments to **mirtCAT** relevant up to this point (with default in brackets)

- `method` – (`'MAP'`) $\boldsymbol{\theta}$ estimation method. Can be any of the options available via the `mirt::fscores()` function
- `design` – (`list()`) ... E.g., `quadpts`, `theta_range`, `KL_delta`, `weights`
- `preCAT` – (`list()`) Information for stage prior to beginning the CAT (mainly useful when using ML opitimizer for $\hat{\theta}$)

Terminate Test

Finally, the test is checked regarding whether it should be terminated (bracket terms are associated with the `design` input with their defaults).

- $SE(\hat{\boldsymbol{\theta}})$ based on the observed information is less than some predefined tolerance (`min_SEM = 0.3`)
- A predefined number of items have been administered (`max_items = Inf`)
- The change in $\hat{\boldsymbol{\theta}}_n$ and $\hat{\boldsymbol{\theta}}_{n+1}$ is less than some $\boldsymbol{\delta}$ (`delta_thetas = NA`)
- Individuals can be classified above or below predefined cutoffs (`classify = NA`)
- Testing time has run out (`max_time = Inf`)

Example

Example 2 – Adaptive test

File for this example is located in `Example-2.R`.

Exercise

Unidimensional CAT

The first exercise involves building the fundimental elements to study a unidimensional CAT bank with the available off-line components. This exercise file is located in `Exercise_1.pdf`

Multidimensional Item Selection

Compared to unidimensional IRT models, multidimensional models tend to be slightly more technically complicated for a couple of reasons.

1. Items may measure more than one trait at once,
2. Redundant information may exist if the latent traits are correlated or items contain related material (testlets),
3. Some traits may be more important to measure than others (think bifactor model)

When one or more of these criteria occur it may be more beneficial to perform a multidimensional CAT instead of multi-unidimensional CATs.

Multidimensional Item Selection

The selection process generally involves evaluating the new $\mathcal{F}_p(\boldsymbol{\theta})$ matrices for all remaining items in the pool.

- E.g., information for M2PL with two factors is

$$\mathcal{F}(\boldsymbol{\theta}) = \begin{pmatrix} a_1^2 & a_1 a_2 \\ a_1 a_2 & a_2^2 \end{pmatrix} P(\boldsymbol{\theta}) \cdot (1 - P(\boldsymbol{\theta}))$$

- Similar to the unidimensional information, except now multiple a parameters are included and the result is a matrix, not a scalar

Multidimensional Item Selection

To compare matrices we should collapse all $\mathcal{F}_p(\boldsymbol{\theta})$ into a scalars using suitable criteria. One such criteria is the maximum determinant rule (Segall, 1996)

$$\text{D-rule} = |TI_n(\boldsymbol{\theta}) + \mathcal{F}_p(\boldsymbol{\theta})|$$

where $TI_n(\boldsymbol{\theta})$ is the information matrix given the n previous answered items. This optimally decreases the expected volume in $ACOV(\boldsymbol{\theta})$.

Multidimensional Item Selection

If prior information about the relationship between the latent traits should be included in the item selection process then a posterior selection rule should be used.

For example, when a Gaussian prior is assumed

$$\text{DP-rule} = |TI_n(\boldsymbol{\theta}) + \mathcal{F}_p(\boldsymbol{\theta}) + \Sigma^{-1}|$$

where Σ is the variance-covariance matrix between the $\boldsymbol{\theta}$ values.

Graphical Representation of D-rule with the ACOV

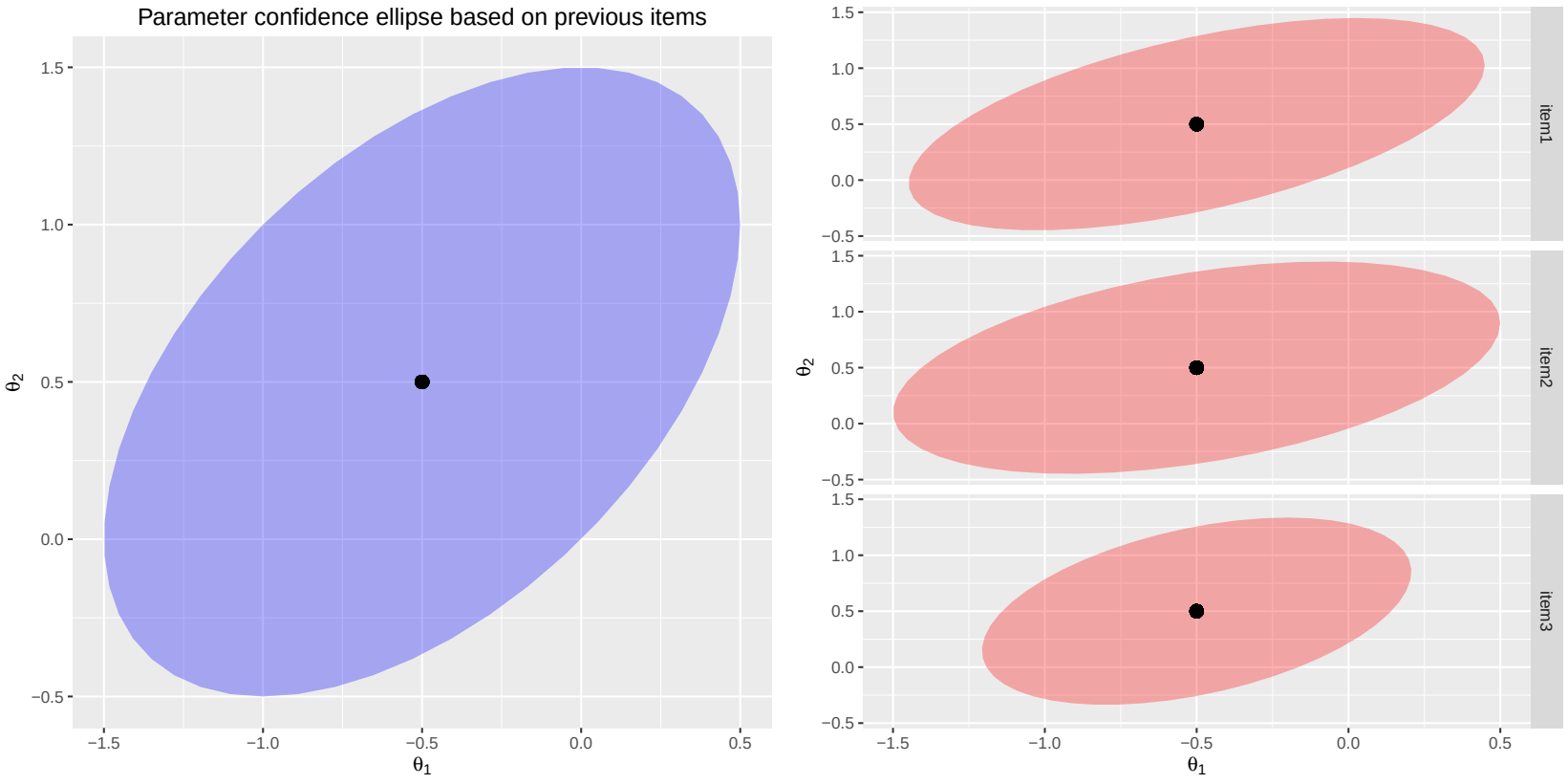

Figure 2: 95% Confidence ellipses for three items given $\boldsymbol{\theta} = [-0.5, 0.5]$

Multidimensional Item Selection

Alternative selection schemes are possible, such as the (weighted) trace of the information matrix

$$\text{T-rule} = \text{Tr}((\mathit{TI}_n(\boldsymbol{\theta}) + \mathcal{F}_p(\boldsymbol{\theta}))\mathbf{W})$$

or $\mathit{ACOV}(\boldsymbol{\theta})$

$$\text{A-rule} = \text{Tr}((\mathit{TI}_n(\boldsymbol{\theta}) + \mathcal{F}_p(\boldsymbol{\theta}))^{-1}\mathbf{W})$$

where $\mathbf{W}$ is a diagonal weight matrix for each trait, and posterior variants readily available. For the unweighted variants of these measures $\mathbf{W}$ is simply an identity matrix.

Multidimensional Item Selection

Another algorithm that favours measuring particular traits better relative than others in the CAT is the weighted rule

$$\text{W-rule} = \mathbf{w}'(TI_n(\boldsymbol{\theta}) + \mathcal{F}_p(\boldsymbol{\theta}))\mathbf{w},$$

which is simply a quadratic form computation, where $\mathbf{w}$ is a weight vector. Posterior variant similar to the DP-rule is also possible.

$$\text{WP-rule} = \mathbf{w}'(TI_n(\boldsymbol{\theta}) + \mathcal{F}_p(\boldsymbol{\theta}) + \Sigma^{-1})\mathbf{w},$$

Multidimensional Item Selection

A difficulty with these multidimensional criteria is that sometimes they aren't computable.

- When $TI_n(\boldsymbol{\theta}) + \mathcal{F}_p(\boldsymbol{\theta})$ is not invertible then the D-rule and A-rule cannot be used.
  - This often occurs in tests with so-called "simple structure", where all cross-loadings are 0
- Including a prior density information can be helpful (e.g., DP-rule), otherwise alternative selection algorithms should be adopted (e.g., T-rule, W-rule)

Multidimensional Models with **mirtCAT**

With **mirtCAT**:

- Multidimensional CATs are defined in the same way as unidimensional CATs. However, multiple slope parameters are included in the `mirt()/bfactor()` or `generate.mirt_object()` functions to indicate the measurement of more than one trait.
    - Covariation between the latent traits may also be included
- As well, you may be interested in multiple response patterns within a given CAT design rather than simply studying a given response pattern. This leads into **mirtCAT**s Monte Carlo simulation capabilities.

Example

Example 3 – MCAT with Monte Carlo simulations

File for this example is located in `Example-3.R`.

Exercise

Multidimensional CAT

Similar to the first exercise, however this involves studying the properties of a multidimensional CAT session using off-line components. As well, multiple response patterns are studied for a Monte Carlo-esk designs in R. Exercise file is located in `Exercise_2.pdf`

Building GUIs with mirtCAT

Building GUIs with mirtCAT

You should be more comfortable with the more basic inputs of mirtCAT by now, so let's turn to implementing these concepts in real-time.

GUIs with **mirtCAT**

```
mirtCAT(df, mo, ...)
```

- `df` is a `data.frame` object containing question stems, options, answers, and image stem paths
- `mo` is an object defined by **mirt** containing the item/group parameters

Depending on the purpose, one of these inputs can be missing (only `df` supplied, a survey GUI is generated, only `mo` is supplied, an off-line CAT; supplying both creates the adaptive GUI)

GUIs with **mirtCAT**

```
mirtCAT(df, mo, ...)
```

- `df` is a `data.frame` object containing question stems, options, answers, and image stem paths
- `mo` is an object defined by **mirt** containing the item/group parameters

Depending on the purpose, one of these inputs can be missing (**only `df` supplied, a survey GUI is generated**, only `mo` is supplied, an off-line CAT; **supplying both creates the adaptive GUI**)

The Mighty df Object

The `df` object contains many important components for the GUI by reserving a number of keywords. Each row corresponds to a given item to be administered. The following are the three essential input components.

- `Question` – Character vector for the respective item stem.
- `Option.#` – Response options, where # is replaced by 1, 2, ..., k. `NA`'s used as placeholders if items have different k's
- `Type` – The type of response stimuli to present: `radio`, `select`, `text`, `slider`, ...

The `Question` and `Option` inputs are also interpreted using MathJaX; hence, math equations are rendered automatically.

The Mighty df Object (optional arguments)

- `Answer` or `Answer.#` – Character vector indicating the correct answer(s) to a given question. Failing this, see the `AnswerFuns()` input for a functional approach (e.g., `text` inputs that require string matches with `grepl()`)
- `...` – additional arguments passed to the respective **shiny** constructor functions (e.g., `step`, `ticks`, `...`), or otherwise additional arguments to track for the item (more on this later)

shinyGUI

Much like the important `design` input, lower-level options in the generated GUI are controlled by the `shinyGUI` list. Many of these inputs require at least some familiarity with functions from **shiny**.

- Visual-based inputs such as `title`, `authors`, `instructions`, `firstpage`, `lastpage`, `begin_message`, `stem_default_formats`, `css`, `theme`, `ui`,
- Control-based inputs such as `password`, `temp_file`, `forced_choice`, `stop_App`, `time_before_answer`, and
- Information collection inputs such as `demographics` and `demographics_inputIDs`

Example

Example 4 – A Simple Survey GUI

File for this example is located in `Example-4.R`.

Integration with Off-line Components

- Obviously, supplying an `mo` object, and modifying the `criteria` input to some adaptive criterion, will invoke a CAT session
- The `design`, `start_item`, `preCAT`, `...`, etc, inputs are also supported because the two approaches use the same software engine

This slide is almost not needed because this is pretty obvious, no? Maybe take a deep breath, because the hard work is basically done...

Example

Example 5 – A CAT with formative assessment

File for this example is located in `Example-5.R`.

Exercise

Building a CAT GUI

This concludes the long build-up on how to build a CAT-GUI with **mirtCAT**. You should have a good idea of how to construct (at least a basic) CAT with an associated GUI element. The exercise file is located in `Exercise_3.pdf`

Advanced CAT Topics and Features

Advanced CAT Topics and Features

This section deals with theoretical and practical topics that I would consider more advanced, and likely require some experimenting with before becoming comfortable. The following topics are presented in no particular order, but include features which some may find attractive.

Fair Warning

Here be dragons! Or, at least some other kind of animal.

Designing Original Graphical Stimuli (DOGS)

If default stem-presentation formats are not to your liking, or you have other external files (e.g., HTML) you would like to include instead, then the following keywords to `df` can be used.

- `Stem` – a character vector of absolute or relative paths pointing external markdown (.md) or HTML (.html) files
- `StemExpression` – A logical vector indicating whether the `Question` input should be evaluated in R first. This allows for HTML to be generated via **shiny** functions (e.g., `Question = "h3('This is my question.')"`)

The latter allows for simple low-level tweaks with **shiny** code for DOGS with CATs (e.g., for more controlled font styles, audio, video, etc.)

DOGS with Functions

Alternatively, if the `Type` argument is not to your specification then DOGS directly using lower-level **shiny** code is an option via the `customTypes` input.

- Each function must have the argument `inputID` and `df_row`
  - the former represents the associated `inputID` name to be used in **shiny**'s response functions
  - `df_row` is the indexed row from the `df` object
- The name supplied to `customTypes` is then used as the `Type` argument in `df`

DOGS named Doug

```
Doug <- function(inputId, df_row){
   list(h3(df_row$Question),
     radioButtons(inputId = inputId,
                  label='', selected = '',
                  choices = with(df_row,
                                 c(Option.1, Option.2)))
     )
}

df <- data.frame(Type = 'good_DOGS',
    Question = '\"The better I get to know men,
       the more I find myself loving dogs.\"',
    Option.1='Charles de Gaulle',
    Option.2='Phil Chalmers', stringsAsFactors=FALSE)
results <- mirtCAT(df = df,
                   customTypes = list(good_DOGS=Doug))
```

Example

Example 6 – Custom GUI elements

File for this example is located in `Example-6.R`.

Customized Item Selection

Perhaps you require an item-selection scheme beyond the simple ones provided by the `criteria` input. To create a custom item selection map

```
myfun <- function(design, person, test) ...
result <- mirtCAT(...,
                  design = list(customNextItem=myfun))
```

- Function returns either an integer, indicating the next item to administer, or `NA` to terminate the session
- `design` (S4), `test` (S4), `person` (RefClass) contain all the information about the session
- Can be indexed directly (`str()`), or more safely with `extract.mirtCAT()` (see next slide)

Extractor Functions

- In an attempt to expose and document internal properties useful for front-end users, I've exposed elements of the `person`, `design`, and `shinyGUI` objects by using the extractor function `extract.mirtCAT()` (see also `extract.mirt()`).

Two other helper functions to be aware of are

- `findNextItem(...)` – as it says. Find the next item given the criteria, returning a single integer number
- `computeCriteria(...)` – for computing the actual values of a given criteria for each remaining item in the pool (e.g., the MI values for each respective item)

Customized Item Selection

Additional user-defined properties about the test and respective participants can be included via the `design` inputs

- `test_properties` – a `data.frame` object with as many rows as items, and
- `person_properties` – a `data.frame` object with as many rows as participants (in GUIs there's obviously 1 row)

Extracting from these objects using `if-then` strategies makes for a fully controlled CAT session given demographic and test structure information.

Passing Responses to `fscores()`

Because **mirtCAT** often calls **mirt** to estimate various aspects within the CAT, re-estimating abilities once the test is complete is also possible.

```
responses <- summary(result,
                     sort=FALSE)$scored_responses
fscores(mirt_object, response.pattern = responses,
        method = "ML")
```

Passing Responses Back to **mirt**

In the same light as with `fscores()`, data can be extracted from **mirtCAT** and added to the original data used to calibrate the item parameters. Provides a fluid bidirectional work-flow for collecting responses and calibrating parameters.

```
old_model <- mirt(old_dat, ...)
result <- mirtCAT(df, old_model, ...)
responses <- summary(result,
                     sort = FALSE)$scored_responses

# re-estimate model, run mirtCAT() again....ad nosium
new_dat <- rbind(old_dat, responses)
new_model <- mirt(new_dat, ...)
result <- mirtCAT(df, new_model, ...)
responses <- summary(result,
                     sort = FALSE)$scored_responses
```

Customized θ Updating Algorithms

Much the same as `customNextItem()`, $\hat{\theta}$ can be updated manually using a new input to `design`

```
myfun <- function(design, person, test){
    fs <- fscores(...)
    person$thetas <- ...
}
result <- mirtCAT(...,
            design = list(customUpdateThetas=myfun))
```

- Function should return `invisible()` because it's just about modifying the `RefClass`
- Gives the administrative control over how θ estimates and their SEs are realized

Example

Example 7 and 7b – Customized selection and updating

Files for this example are located in `Example-7.R` and `Example-7b.html`.

Content Balancing and Constraints

Two often important consideration when implementing CATs are

1. Ensure that various 'Content Areas' are sampled from a sufficient number of times during the test (e.g., a certain proportion of algebra and geometry items should appear), and
2. Various item-level constraints must be satisfied (e.g., item 1 and item 10 cannot appear in the same session).

Content Balancing

Simple methods for 1) include

- Classifying items according to content groups (i.e., a proportion) and selecting the items throughout the test to minimize the observed versus required proportions, or
- Approaching the issue with MCATs, where for example bi-factor models are used in which each specific trait is related to a particular 'content' area and items are selected from accordingly

Constraints

mirtCAT supports 5 item-level constraints

- `not_scored` – indicating the item should be administered but not scored (just response collected)
- `excluded` – exclude item for all participants
- `independent` – declare which items should never appear in the same session

Constraints

For testlets in particular, the following constraints are useful:

- `ordered` – if one item is to be administered in a set, then the remaining items in said set must be administered in sequence
- `unordered` – if one item is selected for the this set then the remaining items must be administered. However, the order does not matter

Content Balancing and Constraints

That said, there are a number of limitations in vanilla **mirtCAT**s approach.

- Proportionality constraints are stochastic approximations only. Bifactor approach is also stochastic
- Exact constraints generally not possible (e.g., a 30-30-40 split between item clusters)
- With a large number of constraints there can be situations in which the tests specifications cannot occur (e.g., running out of items before a certain content constraints are reached)

This leads us into a newly implemented feature known as *optimal test design*.

Optimal Test Designs

- Technique based on using an *integer solver* for a high-dimensional optimization given known item properties
- Hundreds of constraints can be defined and guaranteed to be satisfied for each participant (so long as an initial solution is possible; though this may lead to a BAT)
- Categorical and quantitative constraint criteria possible, which can be updated on-the-fly as well, making the selection algorithm rather flexible
- In **mirtCAT** used within a `customNextItem()` definition after passing a `constr_fun()` argument

Optimal Test Designs

E.g., Exactly 10 items should be administered, item 7 must be included, and items 1 and 2 should not be included in the same test (though neither must be selected).

```
constr_fun <- function(person, test, design){
        nitems <- extract.mirt(test@mo, 'nitems')
        lhs <- matrix(0, 3, nitems)
        lhs[1, ] <- 1
        lhs[2, 7] <- 1
        lhs[3, c(1,2)] <- 1
        data.frame(item=lhs, relation=c("==", "==", "<="),
                   value=c(10, 1, 1))
}
```

The constraints are $1 \cdot x_1 + \ldots + 1 \cdot x_n = 10$, $1 \cdot x_7 = 1$, and $1 \cdot x_1 + 1 \cdot x_2 + \ldots + 0 \cdot x_n \leq 1$, where the x_i's are binary.

Shadow CAT

$$Optimal\ test\ design + CAT = Shadow\ CAT$$

- Really, it's that simple. Combine an adaptive selection criteria with the optimal test design strategy.
- When selecting each item, an optimization is performed where the next most optimal item criteria is selected subject to the constraint that all the item-level requirements are achieved
- This guarantees that the constraints are properly met in practice for each participant

Shadow CAT Testing with mirtCAT

Given a `constr_fun` definition passed to `mirtCAT()`.

```
customNextItem <- function(person, design, test){
    # Compute objective for optimizer
    obj <- computeCriteria(person=person, design=design,
                           test=test, criteria='MI')
    # pass objective to findNextItem()
    item <- findNextItem(person=person, design=design,
                         test=test, objective=obj)
    item
}
```

Example

Example 8 and 8b – Optimal test assembly and Shadow CAT

Files for this example are located in `Example-8.R` and `Example-8b.html`.

Features Not Discussed/Demonstrated

- Pre-CAT designs to collect data responses prior to beginning the CAT
- Exposure control settings (i.e., random sampling, Sympson-Hetter)
- Support for nearly all models in **mirt**, which can be mixed (e.g., 70% 3PL models, 30% GPCM)
- Two-step control of internal objects (required when hosting **mirtCAT** on a remote server)
- Exposing all internal objects to manually control each aspect of the internal objects manually (perhaps for easier hosting on third-party software such as **Concerto**)

. . . to name a few. These are left as exercises to the viewer.

Exercise

Advanced features in **mirtCAT**

If you're feeling adventurous, the exercise file `Exercise_4.pdf` includes a number of more advanced CAT applications using many of the customization and lower-level features in **mirtCAT**.

Additional Information

mirt and **mirtCAT** are actively being developed by yours truly. The development versions of the packages are obtainable on Github and have a handful of examples on the associated wiki's.

- Github: `https://github.com/philchalmers/mirt` and `https://github.com/philchalmers/mirtCAT`
- Github wiki's: `https://github.com/philchalmers/mirt/wiki` and `https://github.com/philchalmers/mirtCAT/wiki`

For general questions about the packages, or other IRT related topics on analyzing data, there also is a Google mailing list which is free to join.

'Till Next Time

I hope you enjoyed this presentation, and I hope you learned something new about CATs and DOGS (and maybe BATs?) with **mirtCAT**.[1]

Thank you!

[1] No animals were harmed during the making of these slides.