# True Score Imputation with the TSI Package in R

The `devtools` library can be used to install the `TSI` package directly from Github:

```
library(devtools)
install_github('mmansolf/TSI')
library(TSI)
library(mice)
```

## Classical Test Theory with One Mismeasured Variable

### Example data

First, let's simulate some data to demonstrate true score imputation for classical test theory! Here I'm simulating two samples of true score `x`, with mean difference equal to `d`:

```
########################
#SIMULATION PARAMETERS#
set.seed(0)
n = 1000 #sample size
ratio = 0.6 #ratio of true variance to error variance; squared reliability
d = 0.5 #mean difference
mean_x = 0 #mean of true score
mean_w = 0 #mean of observed score
var_x = 1 #variance of true score
var_w = 1 #variance of observed score
```

```
#################
# SIMUALTE DATA #
x1 = rnorm(n, mean_x, sqrt(var_x))
e1 = rnorm(n, 0, sqrt(var_x * (1 - ratio) / ratio))
w1 = x1 + e1
x2 = rnorm(n, mean_x + d * sqrt(var_x), sqrt(var_x))
e2 = rnorm(n, 0, sqrt(var_x * (1 - ratio) / ratio))
w2 = x2 + e2
```

Before imputing, notice that we don't actually have a variable to relate `x` to include in an imputation model, only two groups. Let's create a dummy-coded variable `y`, where `0` corresponds to the first group and `1` corresponds to the second group, to create a data set to work with:

```
w = c(w1, w2)
y = c(rep(0, n), rep(1, n))
data_ctt = data.frame(w, y)
head(data_ctt)
#>         w y
```

```
#> 1  1.029 0
#> 2  1.177 0
#> 3  1.202 0
#> 4  0.138 0
#> 5 -0.788 0
#> 6 -1.597 0
```

You can access these data directly as the `data_ctt` object in the `TSI` package.

## True score imputation using the `TSI` function

I'll present the imputation code, then explain the pieces:

```
set.seed(0)
mice_data = TSI(data_ctt,
                os_names = "w",
                score_types = "CTT",
                reliability = ratio,
                mean = 0,
                var_ts = 1,
                mice_args = list(m = 5, printFlag = F))
#> [1] "Preliminary checks passed for true score imputation! Building call to mice..."

mice_data
#> Class: mids
#> Number of multiple imputations:  5
#> Imputation methods:
#>           w           y      true_w
#>          ""          "" "truescore"
#> PredictorMatrix:
#>        w y true_w
#> w      0 1      0
#> y      1 0      0
#> true_w 1 1      0
```

For classical test theory, the following are required:

- `os_names`: Character vector containing the names of the observed scores. Here, there is just one observed score `"w"`.
- `score_types`: Character vector of models used to generate each score. Here, the score was generated under classical test theory `"CTT"`.
- `reliability`: Numeric vector of reliability values for each observed score. Here, I copied this from the `ratio` variable defined above.
- `mean` and `var_ts`: Numeric vectors of means and variances of the underlying true scores. Here, `x` was simulated with a mean of `0` and variance of `1`.
- `mice_args`: Additional arguments passed to the `mice` function. Here, I'm specifying 5 imputations and to not print the trace output during execution.

Observe that a new variable, `true_w`, was generated as part of this call. This variable contains the imputed true score values. We can quickly look at the first imputation using the `complete` function in the `mice` package:

```
head(complete(mice_data))
#>        w y  true_w
#> 1  1.029 0  1.1793
#> 2  1.177 0  0.0141
#> 3  1.202 0  0.5073
#> 4  0.138 0 -0.3704
#> 5 -0.788 0 -0.4067
#> 6 -1.597 0 -0.7955
```

### Trying it yourself

This example is one of the examples available when running `example("TSI")`

### A couple other options

If true score imputation is required for multiple scores, all such variables must be specified in `os_names`. If these variables share imputation characteristics (`score_types`, `reliability`, `mean`, and/or `var_ts`), a single value can be specified for each, and this input is recycled for all observed scores specified in `os_names`.

Instead of specifying `mean` and `var_ts`, another option is to specify the metric of the true score if it is a z score (mean = 0, SD = 1; `metric = "z"`), T-score (mean = 50, SD = 10; `metric = "T"`), or standard score (mean = 100, SD = 15, `metrics = "standard"`). Here's an example, using the z score metric because the underlying `x` has a mean of 0 and SD of 1:

```
set.seed(0)
mice_data = TSI(data_ctt,
                os_names = "w",
                score_types = "CTT",
                reliability = ratio,
                metrics = "z",
                mice_args = list(m = 5, printFlag = F))
#> [1] "Preliminary checks passed for true score imputation! Building call to mice..."

mice_data
#> Class: mids
#> Number of multiple imputations:  5
#> Imputation methods:
#>           w           y      true_w
#>          ""          "" "truescore"
#> PredictorMatrix:
#>        w y true_w
#> w      0 1      0
#> y      1 0      0
#> true_w 1 1      0
```

## Analyzing the imputed data

Once data are generated using `TSI`, the resulting mids object can be analyzed using multiple imputation techniques. Here, we can use the `with` functionality of `mice` to predict `x` from `y`:

```
pool(with(mice_data, lm(true_w~y)))
#> Class: mipo    m = 5
#>          term m estimate    ubar        b       t dfcom    df   riv lambda
#> 1 (Intercept) 5  -0.0352 0.00093 0.000122 0.00108  1998 193.2 0.157  0.136
#> 2           y 5   0.6158 0.00186 0.000403 0.00234  1998  88.6 0.260  0.206
#>     fmi
#> 1 0.144
#> 2 0.224
```

Comparing this result to that from using the observed score `w`:

```
pool(with(mice_data, lm(w~y)))
#> Class: mipo    m = 5
#>          term m estimate    ubar b       t dfcom   df riv lambda   fmi
#> 1 (Intercept) 5  -0.0361 0.00161 0 0.00161  1998 1996   0      0 0.001
#> 2           y 5   0.6062 0.00322 0 0.00322  1998 1996   0      0 0.001
```

The results are roughly the same, because under classical test theory, the slope relating the mismeasured `x` from the measured `y` is unaffected by the measurement error. One statistic that is affected is the standard deviation of the score. To obtain this, we obtain a `list` of completed data sets `mice_data` by using the `complete` function with `"all"` as the second input,

```
mice_data = complete(mice_data, "all")
```

calculate the standard deviation of each column in each imputed data set, stored as `sds`,

```
sds = sapply(mice_data, function(d)apply(d, 2, sd))
```

and calculate the mean standard deviation across imputations:

```
apply(sds, 1, mean)
#>      w      y true_w
#>   1.30   0.50   1.01
```

The standard deviation of `w` is inflated compared to that of the underlying true score, while the standard deviation of the imputed `true_w` is unbiased.

# Expected a Posteriori Scoring with Two Mismeasured Variables

## Example data

Because true score imputation can be used in the `mice` package, it can be used in tandem with conventional multiple imputation. To illustrate, I simulated two uncorrelated, normally-distributed variables, `x` and `m`, and used them to generate a new variable `y` according to a simple regression model:

```
set.seed(1)
x = rnorm(n, 0, 1)
m = rnorm(n, 0, 1)
y = 1 + 0.4 * x + 0.6 * m + rnorm(n, 0, 1)
```

Thus, `y` is related to `m` and `w` by the equation `y = 0.4x + 0.6m + e`, where `e~N(0, 1)`, in a prototypical multiple regression configuration.

Next, I used calibrated item parameters from the NIH Toolbox emotion battery norming study (Kupst et al., 2015) to simulate item responses on the 10-item Perceived Stress Scale for each of `x` and `y`. I then scored these item responses using the same item parameters, yielding a set of T-scores and standard errors (SE's) for each record. These two quantities are provided routinely by the HealthMeasures scoring service, and while the standard errors are more often used in assessing individual change, true score imputation allows them to be used in statistical analysis of data sets with multiple observarions.

Lastly, I used a missing completely at random (MCAR) missing data model to set approximately 10% of the values of `m` and the T-score/SE pairs to missing. This "amputation" was conducted separately for `m`, for the T-score/SE pairs for `x`, and for the T-score/SE pairs for `y`.

We can access the simulated data using the `data_eap` data set in the `TSI` package.

```
head(data_eap)
#>     Fx   Fy SE.Fx SE.Fy       m
#> 1 43.5   NA  3.43    NA  1.1350
#> 2 56.1 45.7  3.59  3.65  1.1119
#> 3   NA 63.2    NA  3.34 -0.8708
#> 4 65.7 66.2  3.51  3.42  0.2107
#> 5 55.9   NA  3.78    NA  0.0694
#> 6 42.1 51.8  3.56  3.66 -1.6626
```

## True score imputation using the `TSI` function

The code for imputation with two EAP-scored variables is nearly identical to that above for CTT:

```
set.seed(0)
mice_data = TSI(data_eap,
                os_names = c("Fx", "Fy"),
                se_names = c("SE.Fx", "SE.Fy"),
                metrics = "T",
                score_types = "EAP",
                separated = T,
                ts_names = c("Tx", "Ty"),
                mice_args = c(m = 5, maxit = 5, printFlag = F))
#> [1] "Preliminary checks passed for true score imputation! Building call to mice..."

mice_data
#> Class: mids
#> Number of multiple imputations:  5
#> Imputation methods:
#>          Fx          Fy       SE.Fx       SE.Fy           m          Tx
#>       "pmm"       "pmm"       "pmm"       "pmm"       "pmm" "truescore"
#>          Ty
#> "truescore"
#> PredictorMatrix:
#>       Fx Fy SE.Fx SE.Fy m Tx Ty
#> Fx     0  1     1     1 1  0  0
#> Fy     1  0     1     1 1  0  0
#> SE.Fx  1  1     0     1 1  0  0
#> SE.Fy  1  1     1     0 1  0  0
```

```
#> m      1  1     1     1 0  0  0
#> Tx     1  1     1     1 1  0  0

head(complete(mice_data))
#>     Fx   Fy SE.Fx SE.Fy       m   Tx   Ty
#> 1 43.5 49.1  3.43  3.28  1.1350 41.4 47.5
#> 2 56.1 45.7  3.59  3.65  1.1119 57.8 51.8
#> 3 42.1 63.2  3.36  3.34 -0.8708 43.5 65.7
#> 4 65.7 66.2  3.51  3.42  0.2107 69.7 69.4
#> 5 55.9 84.1  3.78  3.51  0.0694 55.6 83.5
#> 6 42.1 51.8  3.56  3.66 -1.6626 45.7 58.8
```

A few differences:

- Two new variables were created: `Tx` and `Ty`. These names were specified using the optional `ts_names` input to `TSI`, which works as follows:
  - If `ts_names` is not specified, the prefix `"TRUE_"` is prepended to each of `os_names` to produce the names of the resulting imputed true score variables.
  - If `ts_names` is specified, these names are used instead, with each element of `ts_names` denoting the name of the imputed true score variable to be generated from the respective variable in `os_names`.
- Two variables are specified in `os_names` because both `Fx` and `Fy` are T scores.
- The character vector `se_names`, which contains the names of the standard error variables corresponding to each observed score in `os_names`, must be specified for EAP scores and have the same length as `os_names`.
- A `T`-score metric was used, consistent with how the Perceived Stress Scale is scored in the NIH Toolbox.
- `score_types` contains `"EAP"` because NIH Toolbox scores the Percieved Stress Scale using EAP scoring
- The extra argument `separated` specifies whether true score imputation uses an average standard error (`separated = F`), which runs faster but doesn't account for differential measurement error of the observed scores for each respondent, or whether separate standard errors are used for each value of each observed score (`separated = T`), which runs slower but accounts for differential measurement error.

Note that although there are two variables measured with error (`Fx` and `Fy`), only one value is provided for `metrics`, `score_types`, and `separated`, resulting in these settings being applied to all variables measured with error.

## Analyzing the imputed data

Once data are generated using `TSI`, the resulting mids object can once again be analyzed using multiple imputation techniques. For reference, estimating this model using the true scores, with no missing data, yields:

```
lm(y~x + m)
#>
#> Call:
#> lm(formula = y ~ x + m)
#>
#> Coefficients:
#> (Intercept)            x            m
#>       1.016        0.449        0.622
```

Using the observed T scores yields:

```
pool(with(mice_data, lm(Fy~Fx + m)))
#> Class: mipo    m = 5
#>          term m estimate     ubar        b      t dfcom   df   riv lambda   fmi
#> 1 (Intercept) 5    39.06 2.521042 1.751999 4.6234   997 18.7 0.834  0.455 0.505
#> 2          Fx 5     0.39 0.000981 0.000602 0.0017   997 21.4 0.737  0.424 0.471
#> 3           m 5     5.21 0.086023 0.019439 0.1094   997 79.0 0.271  0.213 0.233
```

Observe that the regression coefficient associated with `Fx`, the T-score for `x`, is between 0 and 1, because both T scores are on the same metric; therefore, this estimate can be directly compared to that from the true, complete data above. In contrast, the `m` variable was not linearly transformed and therefore this estimated coefficient should be divided by 10 because the SD of T scores is 10. Placing this estimate back on the original metric by dividing by 10 yields 0.521. Both regression coefficients are biased.

Finally, using true score imputation yields:

```
pool(with(mice_data, lm(Ty~Tx + m)))
#> Class: mipo    m = 5
#>          term m estimate     ubar        b        t dfcom    df   riv lambda
#> 1 (Intercept) 5   38.683 2.489222 3.91293 7.18474   997  9.12 1.886  0.654
#> 2          Tx 5    0.423 0.000963 0.00133 0.00256   997 10.02 1.656  0.623
#> 3           m 5    5.919 0.100030 0.03769 0.14526   997 38.91 0.452  0.311
#>     fmi
#> 1 0.711
#> 2 0.681
#> 3 0.344
```

Bias has been substantially reduced, and the fractions of missing information `fmi` now represent both information lost to missingness and information lost to measurement error.

#### Trying it yourself

This example is one of the examples available when running `example("TSI")`

## Calling the `mice` Function Directly with `method = "truescore"`

The `TSI` function generates a call to the `mice` function, but the `mice` function can also be called directly to perform true score imputation. The core of the `TSI` package is a function named `mice.impute.truescore` which can be called from the `mice` package by setting `method = "truescore"` for each variable to be imputed using this method. These calls can get complicated, so for most purposes we hope that the `TSI` function can provide a convenient means to conduct true score imputation, but when an analyst needs to modify this call, they can do so by working with `mice` directly.

### Classical test theory

One convenience provided by the `TSI` function is that it creates the true score variables automatically. When using `mice` directly, one must first add empty (all `NA` values) variables to the data set corresponding to the true scores:

```
data_ctt_2 = data_ctt
data_ctt_2$true_w = NA
```

Then, the `mice` function is called with `method = "truescore"`:

```
set.seed(0)
mice_data = mice(data_ctt_2, m = 5,
                 blocks = list("true_w"),
                 method = "truescore",
                 calibration = list(os_name = "w",
                                    score_type = "CTT",
                                    reliability = ratio,
                                    mean_ts = mean_x,
                                    var_ts = var_x),
                 predictorMatrix = matrix(c(1, 1, 0), 1, 3, byrow = T),
                 printFlag = F,
                 remove.constant = F)
mice_data
#> Class: mids
#> Number of multiple imputations:  5
#> Imputation methods:
#>      true_w
#> "truescore"
#> PredictorMatrix:
#>        w y true_w
#> true_w 1 1      0
```

A few notes:

- Each set of imputed variables must be specificed as a block using the `blocks` argument, which contains a list of vectors with variable names per block.
- Each block must have a matching `method` listed under the `method` argument; here, we're only using true score imputation, so this is simply `"truescore"`.
- The flag `remove.constant = F` is needed to prevent `mice` from automatically discarding the true score variable `true_w` for containing only `NA` values.
- Although not necessary for this example, we specified a `predictorMatrix`, where each row corresponds to a `block` and each column indicates whether the corresponding variable in the data set is included (`1`) or not (`0`) in the imputation model. For more information on these inputs, see the documentation for the `mice` function at `?mice`. Lastly:
- Calibration information, including reliability and the mean and variances of true and observed scores, must be specified in an additional argument named `calibration`. The elements of the list for classical test theory must be the following, in any order, and all must be named as follows:
  - `os_name` (character vector of length one): Name of the observed score variable in the data set.
  - `score_type` (character vector of length one): `"CTT"` for classical test theory.
  - `reliability` (numeric scalar or vector with length equal to `nrow(data)`: An estimate of the reliability of the observed scores as measures of the true score. If a scalar is provided, the same reliability is assumed for all values of the observed score.
  - `mean_ts` and `var_ts` (numeric scalars): Mean and variance of true scores, respectively, specified as in the `TSI` function. Note that specifying `metric` as a shortcut is not available when using `mice` directly.

The resulting `mice_data` object is the same, imputation error notwithstanding, from that created in the classical test theory example above:

```
pool(with(mice_data, lm(w~y)))
#> Class: mipo    m = 5
#>          term m estimate    ubar b       t dfcom   df riv lambda   fmi
#> 1 (Intercept) 5  -0.0361 0.00161 0 0.00161  1998 1996   0      0 0.001
#> 2           y 5   0.6062 0.00322 0 0.00322  1998 1996   0      0 0.001

pool(with(mice_data, lm(true_w~y)))
#> Class: mipo    m = 5
#>          term m estimate    ubar        b       t dfcom    df   riv lambda
#> 1 (Intercept) 5  -0.0352 0.00093 0.000122 0.00108  1998 193.2 0.157  0.136
#> 2           y 5   0.6158 0.00186 0.000403 0.00234  1998  88.6 0.260  0.206
#>     fmi
#> 1 0.144
#> 2 0.224

mice_data = complete(mice_data, "all")
sds = sapply(mice_data, function(d)apply(d, 2, sd))
apply(sds, 1, mean)
#>      w      y true_w
#>   1.30   0.50   1.01
```

## Expected a posteriori scoring for item response theory

For EAP scores, we once again need to add empty true score variables to the data set manually:

```
data_eap_2 = data_eap
data_eap_2$Tx = NA #nolint
data_eap_2$Ty = NA #nolint
```

Then, we need a more complicated call to `mice`:

```
set.seed(0)
mice_data = mice(data_eap_2, m = 5, maxit = 5,
                 method = c("pmm", "pmm", "pmm", "pmm", "pmm",
                            "truescore", "truescore"),
                 blocks = list(Fx = "Fx", Fy = "Fy",
                               SE.Fx = "SE.Fx", SE.Fy = "SE.Fy", m = "m",
                               Tx = "Tx", Ty = "Ty"),
                 blots = list(Tx = list(calibration = list(os_name = "Fx",
                                                           se_name = "SE.Fx",
                                                           score_type = "EAP",
                                                           mean = 50,
                                                           var_ts = 100,
                                                           separated = T)),
                              Ty = list(calibration = list(os_name = "Fy",
                                                           se_name = "SE.Fy",
                                                           score_type = "EAP",
                                                           mean = 50,
                                                           var_ts = 100,
                                                           separated = T))),
                 predictorMatrix = matrix(c(0, 1, 1, 1, 1, 0, 0,
                                            1, 0, 1, 1, 1, 0, 0,
```

```
                                        1, 1, 0, 1, 1, 0, 0,
                                        1, 1, 1, 0, 1, 0, 0,
                                        1, 1, 1, 1, 0, 0, 0,
                                        1, 1, 1, 1, 1, 0, 0,
                                        1, 1, 1, 1, 1, 0, 0), 7, 7, byrow = T),
              printFlag = F,
              remove.constant = F)
mice_data
#> Class: mids
#> Number of multiple imputations:  5
#> Imputation methods:
#>          Fx          Fy       SE.Fx       SE.Fy           m          Tx
#>       "pmm"       "pmm"       "pmm"       "pmm"       "pmm" "truescore"
#>          Ty
#> "truescore"
#> PredictorMatrix:
#>       Fx Fy SE.Fx SE.Fy m Tx Ty
#> Fx     0  1     1     1 1  0  0
#> Fy     1  0     1     1 1  0  0
#> SE.Fx  1  1     0     1 1  0  0
#> SE.Fy  1  1     1     0 1  0  0
#> m      1  1     1     1 0  0  0
#> Tx     1  1     1     1 1  0  0
```

A few extra things to note:

- A separate `block`, and corresponding `method` and row of `predictorMatrix`, must be specified for each variable because all variables have missing data
- Separate `blots` are needed for the two true scores, with different inputs for each. A few things to note here:
- Lastly, the `predictorMatrix` argument must be specified in order to avoid using observed scores for `x` and `y` to predict true scores on `y` and `x`, respectively. Based on (unpublished) simulation results, it seems the best way to specify the predictor matrix for use in `mice` is for true scores to be predicted from all observed variables but *not* to predict other missing data from the imputed true scores. This is the default behavior when the `TSI` function is used, and we recommend, unless further research identifies otherwise, that the same be done when using this function to interact with `mice` directly.

### Trying it yourself

These examples are both available when running `example("mice.impute.truescore")`