Missing Data in SEM

Introduction to SEM with Lavaan

Utrecht University

Kyle M. Lang

Department of Methodology & Statistics
Utrecht University

Outline

Missing Data Mechanisms

Imputation-Based Solutions

- Single Imputation
- Multiple Imputation
- MI-Based Analysis

ML-Based Solutions

- Maximum Likelihood Estimation
- Full Information Maximum Likelihood

Auxiliary Variables

What are Missing Data?

Missing data are empty cells in a dataset where there should be observed values.

- The missing cells correspond to true population values, but we haven’t observed those values.

What are Missing Data?

Missing data are empty cells in a dataset where there should be observed values.

- The missing cells correspond to true population values, but we haven’t observed those values.

Not every empty cell is a missing datum.

- Quality-of-life ratings for dead patients in a mortality study
- Firm profitability after the company goes out of business
- Self-reported severity of menstrual cramping for men
- Empty blocks of data following “gateway” items

A Little Notation

$$
\begin{aligned}
Y &:= \text{An } N \times P \text{ Matrix of Arbitrary Data}\\
Y_{mis} &:= \text{The \textit{missing} part of } Y\\
Y_{obs} &:= \text{The \textit{observed} part of } Y\\
R &:= \text{An } N \times P \text{ response matrix}\\
M &:= \text{An } N \times P \text{ missingness matrix}
\end{aligned}
$$

The R and M matrices are complementary.

- $r_{np} = 1$ means y_{np} is observed; $m_{np} = 1$ means y_{np} is missing.
- $r_{np} = 0$ means y_{np} is missing; $m_{np} = 0$ means y_{np} is observed.
- M_p is the *missingness* of Y_p.

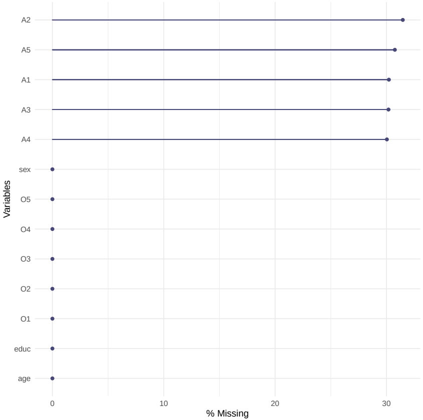
Example

```
## Load some useful packages:
library(dplyr)
library(naniar)
library(ggmice)

## Read in some data:
bfi <- readRDS("../data/bfi_datasets.rds")$incomplete %>%
    select(-matches("N\\d|C\\d|E\\d|male"))

## Compute the variablewise proportions of missing data:
bfi %>% is.na() %>% colMeans() %>% round(2)

  A1   A2   A3   A4   A5   O1   O2   O3   O4   O5  age  sex educ
0.30 0.31 0.30 0.30 0.31 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

Example

Visualize the percentages missing via **naniar**::`gg_miss_var()`.

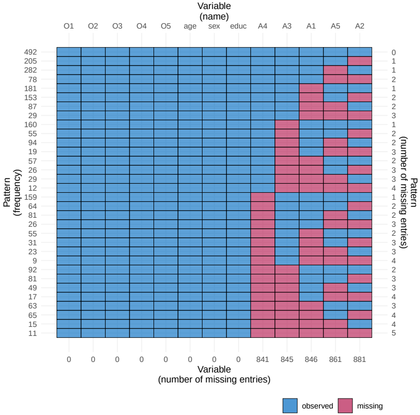
```
gg_miss_var(bfi, show_pct = TRUE)
```



Example

Visualize the missing data patterns via **ggmice**::plot_pattern().

```
plot_pattern(bfi)
```



Example

In **lavaan**, we can directly fit a model with incomplete data.

```
library(lavaan)

## Specify the measurement model:
cfaMod <- "
agree =~ A1 + A2 + A3 + A4 + A5
open  =~ O1 + O2 + O3 + O4 + O5
"

## Estimate the model:
out <- cfa(cfaMod, data = bfi, std.lv = TRUE)
```

Example

The model will estimate, just fine...

```
lavaan 0.6-11 ended normally after 21 iterations

  Estimator                                         ML
  Optimization method                           NLMINB
  Number of model parameters                        21

                                                  Used       Total
  Number of observations                           492        2800

Model Test User Model:

  Test statistic                                79.928
  Degrees of freedom                                34
  P-value (Chi-square)                           0.000

Parameter Estimates:

  Standard errors                             Standard
  Informati
```

Example

```
lavaan 0.6-11 ended normally after 21 iterations

  Estimator                                         ML
  Optimization method                           NLMINB
  Number of model parameters                        21

                                                  Used       Total
  Number of observations                           492        2800

Model Test User Model:

  Test statistic                                79.928
  Degrees of freedom                                34
  P-value (Chi-square)                           0.000

Parameter Estimates:

  Standard errors                             Standard
  Information                                 Expected
  Informati
```

Example

But not everything is as it seems.

```
lavaan 0.6-11 ended normally after 21 iterations

  Estimator                                         ML
  Optimization method                           NLMINB
  Number of model parameters                        21

                                                  Used       Total
  Number of observations                           492        2800

Model Test User Model:

  Test statistic                                79.928
  Degrees of freedom                                34
  P-value (Chi-square)                           0.000

Parameter Estimates:

  Standard errors                             Standard
  Informati
```

Default Approach

Like most software packages, **lavaan** will default to *complete case analysis* when asked to analyze incomplete data.

- In the absence of user input, this is a sensible option.
- That doesn't mean you should actually use deletion to treat the missing data in your analysis.

Default Approach

Like most software packages, **lavaan** will default to *complete case analysis* when asked to analyze incomplete data.

- In the absence of user input, this is a sensible option.
- That doesn't mean you should actually use deletion to treat the missing data in your analysis.

Complete case analysis has two major problems.

1. Throws out useful information (potentially a lot of information)
2. Probably biases parameter estimates.

To understand the second point, we need to discuss *missing data mechanisms*.

MISSING DATA MECHANISMS

Missing Data Mechanisms

Missing Completely at Random (MCAR)

- $P(R|Y_{mis}, Y_{obs}) = P(R)$
- Missingness is unrelated to any study variables.

Missing at Random (MAR)

- $P(R|Y_{mis}, Y_{obs}) = P(R|Y_{obs})$
- Missingness is related to only the *observed* parts of study variables.

Missing not at Random (MNAR)

- $P(R|Y_{mis}, Y_{obs}) \neq P(R|Y_{obs})$
- Missingness is related to the *unobserved* parts of study variables.

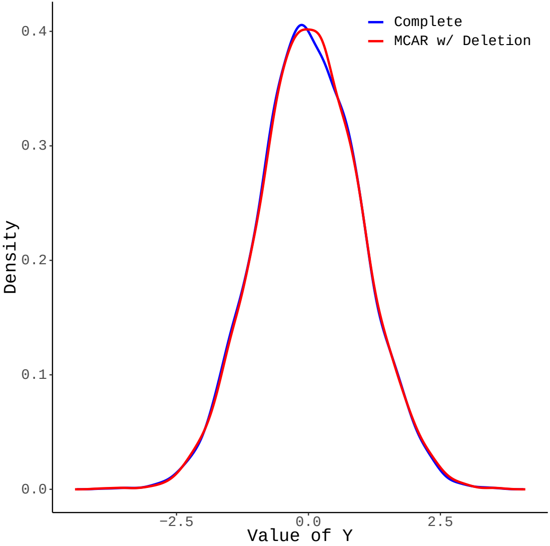
Simulate Some Toy Data

```
nObs <- 5000 # Sample Size
pm   <- 0.3  # Proportion Missing

sigma <- matrix(c(1.0, 0.5, 0.3,
                  0.5, 1.0, 0.0,
                  0.3, 0.0, 1.0),
                ncol = 3)
tmp <- rmvnorm(nObs, c(0, 0, 0), sigma)

x0 <- tmp[ , 1]
y0 <- tmp[ , 2]
z0 <- tmp[ , 3]

cor(y0, x0) # Check correlation between X and Y

[1] 0.4997145
```

MCAR Example

```
## Simulate MCAR Missingness:
mVec <- sample(1 : length(y0), size = pm * length(y0))

yMcar       <- y0
yMcar[mVec] <- NA

cor(yMcar, x0, use = "pairwise") # Look at correlation

[1] 0.5195767
```

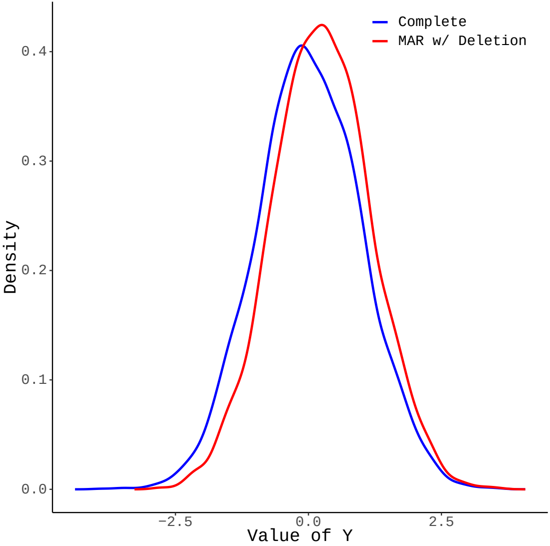
MCAR Example



MAR Example

```
## Simulate MAR Missingness:
mVec <- x0 < quantile(x0, probs = pm)
mean(mVec)

[1] 0.3

yMar        <- y0
yMar[mVec] <- NA

cor(yMar, x0, use = "pairwise") # Not looking so good :(

[1] 0.3822143
```

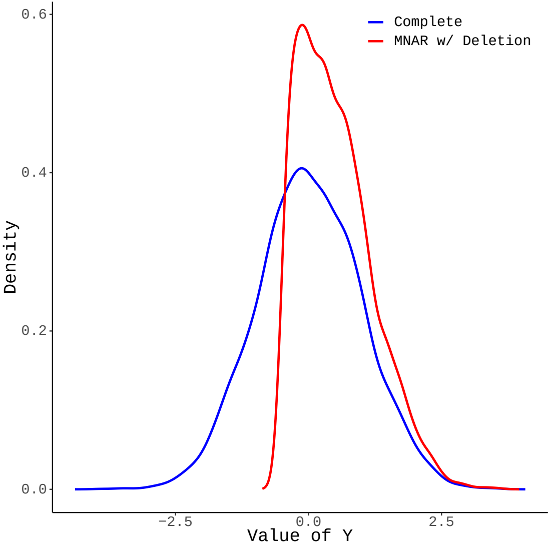
MAR Example


Complete
MAR w/ Deletion
0.4
0.3
0.2
0.1
0.0
Density
−2.5
0.0
2.5
Value of Y

MNAR Example

```
## Simulate MNAR Missingness:
mVec <- y0 < quantile(y0, probs = pm)
mean(mVec)

[1] 0.3

yMnar       <- y0
yMnar[mVec] <- NA

cor(yMnar, x0, use = "pairwise") # Hmm...looks pretty bad.

[1] 0.3902962
```

MNAR Example



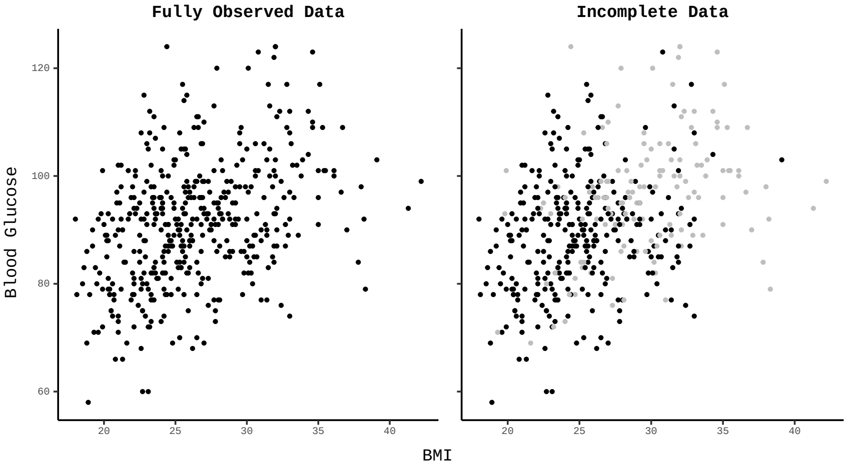
Effects of Deletion

As we saw in the preceding plots, excluding incomplete cases usually alters the variables' distributions.

- The statistics upon which we base our analyses generally summarize these distributions.
- Problems with the distributions show up as bias in the results of our anlayses.

```r
diabetes1 <- diabetes2 <- readRDS("../data/diabetes.rds")

mVec <- simLogisticMissingness0(data     = diabetes1,
                                pm       = 0.3,
                                preds    = c("bmi", "bp"),
                                stdData = TRUE)$r
diabetes2[mVec, "glu"] <- NA
```

Example


Fully Observed Data
Incomplete Data
Blood Glucose
120
100
80
60
20
25
30
35
40
BMI

Example

```
diabetes1 %>% select(bmi, glu, bp) %>% cor()

          bmi     glu        bp
bmi 1.0000000 0.38868 0.3954109
glu 0.3886800 1.00000 0.3904300
bp  0.3954109 0.39043 1.0000000

diabetes2 %>% select(bmi, glu, bp) %>% cor(use = "complete")

          bmi       glu        bp
bmi 1.0000000 0.2566595 0.2052338
glu 0.2566595 1.0000000 0.3011547
bp  0.2052338 0.3011547 1.0000000
```

Example

```
mean(diabetes1$glu)

[1] 91.26018

mean(diabetes2$glu, na.rm = TRUE)

[1] 88.86424

var(diabetes1$glu)

[1] 132.1657

var(diabetes2$glu, na.rm = TRUE)

[1] 115.8254
```

Good Methods

Multiple Imputation (MI)

- Replace the missing values with M plausible estimates
  - Essentially, a repeated application of stochastic regression imputation (with a particular type of regression model)
  - Produces unbiased parameter estimates and predictions
  - Produces “correct” standard errors, CIs, and prediction intervals
  - Very, very flexible
  - Computationally expensive

Good Methods

Full Information Maximum Likelihood (FIML)

- Adjust the objective function to only consider the observed parts of the data
    - Models are directly estimated in the presence of missing data
    - The predictors of nonresponse must be included in the model, somehow
    - Unless you write your own optimization program, FIML is only available for certain types of models
    - In linear regression models, FIML cannot treat missing data on predictors (if the predictors are taken as fixed)

Good Methods

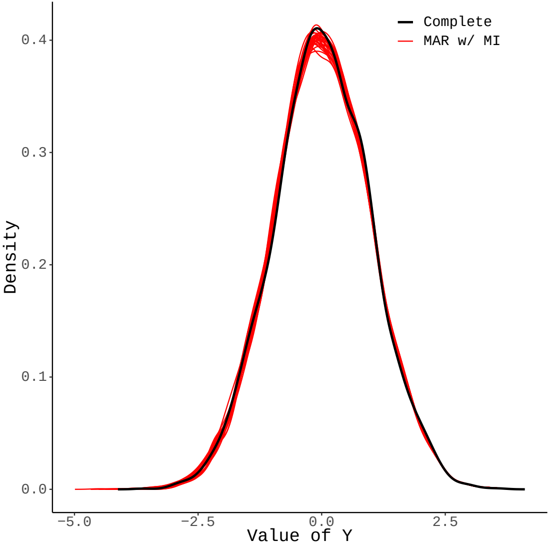
What happens when we apply MI to our previous MAR example?

```
## Estimate imputation model:
miceOut <- mice(data      = data.frame(y = yMar, x = x0),
                m         = 25,
                maxit     = 1,
                method    = "norm",
                printFlag = FALSE)

## Estimate and pool M correlations:
with(miceOut, cor(y, x))$analyses %>% unlist() %>% mean()

[1] 0.504661
```

The MI-based parameter estimate looks good.

- MI produces unbiased estimates of the parameter when data are MAR.

Good Methods



Good Methods

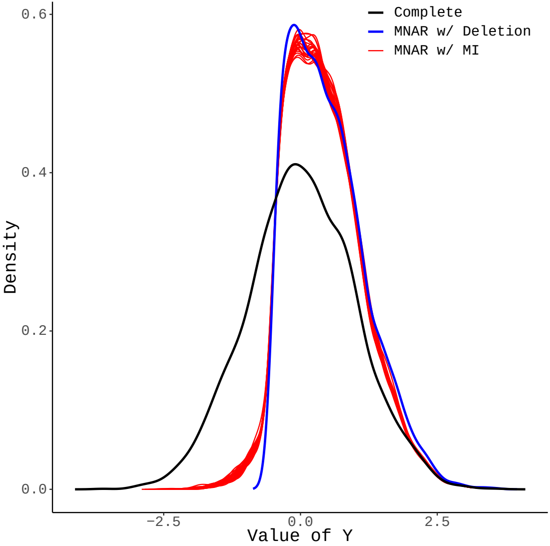
What about applying MI to our MNAR example?

```
## Estimate imputation model:
miceOut <- mice(data      = data.frame(y = yMnar, x = x0),
                m         = 25,
                maxit     = 1,
                method    = "norm",
                printFlag = FALSE)

## Estimate and pool M correlations:
with(miceOut, cor(y, x))$analyses %>% unlist() %>% mean()

[1] 0.4116519
```

The MI-based parameter estimate is still biased.

- MI cannot correct bias in parameter estimates when data are MNAR.

Good Methods



MI Example

```r
## The mice package does MI:
library(mice)

## Multiply impute the missing data:
miceOut <- mice(data      = diabetes2,
                m         = 25,
                maxit     = 1,
                printFlag = FALSE,
                method    = "norm")
```

MI Example

```
## Complete data:
diabetes1 %>% select(bmi, glu, bp) %>% cor()

          bmi     glu        bp
bmi 1.0000000 0.38868 0.3954109
glu 0.3886800 1.00000 0.3904300
bp  0.3954109 0.39043 1.0000000

## MI:
pooledCorMat(miceOut, c("bmi", "glu", "bp"))

          bmi       glu        bp
bmi 1.0000000 0.3135162 0.3954109
glu 0.3135162 1.0000000 0.3563903
bp  0.3954109 0.3563903 1.0000000
```

MI Example

```
mean(diabetes1$glu)

[1] 91.26018

with(miceOut, mean(glu))$analyses %>% unlist() %>% mean()

[1] 90.61747

var(diabetes1$glu)

[1] 132.1657

with(miceOut, var(glu))$analyses %>% unlist() %>% mean()

[1] 123.3748
```

FIML Example

```
fit <- diabetes2 %>%
    select(bmi, glu, bp) %>%
    lavCor(missing = "fiml", output = "sampstat")

## Complete data:
diabetes1 %>% summarize(mean = mean(glu), var = var(glu))

      mean      var
1 91.26018 132.1657

## FIML:
fit %$% c(mean = mean[["glu"]], var = cov["glu", "glu"])

     mean       var
 90.82487 125.27146
```

FIML Example

```
## Complete data:
diabetes1 %>% select(bmi, glu, bp) %>% cor() %>% round(3)

      bmi   glu    bp
bmi 1.000 0.389 0.395
glu 0.389 1.000 0.390
bp  0.395 0.390 1.000

## FIML:
fit$cov %>% cov2cor()

    bmi   glu   bp
bmi 1.000
glu 0.357 1.000
bp  0.395 0.386 1.000
```

Imputation-Based Solutions

Prediction Example

To fix ideas, let's consider the *diabetes* data and the following model:

$$Y_{LDL} = \beta_0 + \beta_1 X_{BP} + \beta_2 X_{gluc} + \beta_3 X_{BMI} + \varepsilon$$

Training this model on the first $N = 400$ patients' data produces the following fitted model:

$$Y_{LDL} = 22.135 + 0.089 X_{BP} + 0.498 X_{gluc} + 1.48 X_{BMI}$$

Prediction Example

To fix ideas, let's consider the *diabetes* data and the following model:

$$Y_{LDL} = \beta_0 + \beta_1 X_{BP} + \beta_2 X_{gluc} + \beta_3 X_{BMI} + \varepsilon$$

Training this model on the first $N = 400$ patients' data produces the following fitted model:

$$Y_{LDL} = 22.135 + 0.089 X_{BP} + 0.498 X_{gluc} + 1.48 X_{BMI}$$

Suppose a new patient presents with $BP = 121$, $gluc = 89$, and $BMI = 30.6$. We can predict their LDL score by:

$$\begin{aligned} \hat{Y}_{LDL} &= 22.135 + 0.089(121) + 0.498(89) + 1.48(30.6) \\ &= 122.463 \end{aligned}$$

Imputation is Just Prediction*

In Lecture 3, you heard a bit about missing data imputation.

- Multiple imputation is one of the best ways to treat missing data.

Imputation is nothing more than a type of prediction.

1. Train a model on the observed parts of the data, Y_{obs}.
   - Train the imputation model.
2. Predict the missing values, Y_{mis}.
   - Generate imputations.
3. Replace the missing values with these predictions.
   - Impute the missing data.

Imputation can be used to support either prediction or inference.

- Our goals will dictate what type of imputation we need to do.

*Levels of Uncertainty Modeling

van Buuren (2018) provides a very useful classification of different imputation methods:

1. Simple Prediction
   - The missing data are naively filled with predicted values from some regression equation.
   - All uncertainty is ignored.
2. Prediction + Noise
   - A random residual error is added to each predicted value to create the imputations.
   - Only uncertainty in the predicted values is modeled.
   - The imputation model itself is assumed to be correct and error-free.
3. Prediction + Noise + Model Error
   - Uncertainty in the imputation model itself is also modeled.
   - Only way to get fully proper imputations in the sense of Rubin (1987).

Do we really need to worry?

The arguments against single imputation can seem archaic and petty. Do we really need to worry about this stuff?

Do we really need to worry?

The arguments against single imputation can seem archaic and petty. Do we really need to worry about this stuff?

- YES!!! (At least if you care about inference)

The following are results from a simple Monte Carlo simulation:

	Complete Data	Conditional Mean	Stochastic	MI
cor(X, Y)	0.500	0.563	0.498	0.497
Type I Error	0.052	0.138	0.120	0.054

Mean Correlation Coefficients and Type I Error Rates

Do we really need to worry?

The arguments against single imputation can seem archaic and petty. Do we really need to worry about this stuff?

- YES!!! (At least if you care about inference)

The following are results from a simple Monte Carlo simulation:

	Complete Data	Conditional Mean	Stochastic	MI
cor(X, Y)	0.500	0.563	0.498	0.497
Type I Error	0.052	0.138	0.120	0.054

Mean Correlation Coefficients and Type I Error Rates

- Conditional mean substitution overestimates the correlation effect.
- Both single imputation methods inflate Type I error rates.
- MI provides unbiased point estimates and accurate Type I error rates.

Simulate Some Toy Data

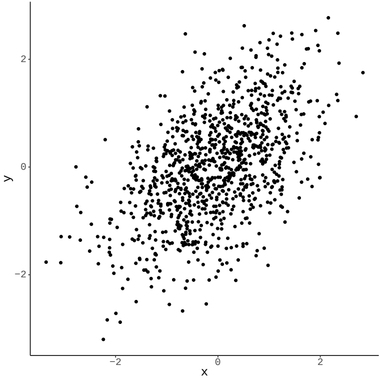
```
nObs <- 1000 # Sample Size
pm   <- 0.3  # Proportion Missing

sigma <- matrix(c(1.0, 0.5, 0.0,
                  0.5, 1.0, 0.3,
                  0.0, 0.3, 1.0),
                ncol = 3)

dat0 <- as.data.frame(rmvnorm(nObs, c(0, 0, 0), sigma))
colnames(dat0) <- c("y", "x", "z")
```

Simulate Some Toy Data

```r
## Impose MAR Nonresponse:
dat1 <- dat0
mVec <- with(dat1, x < quantile(x, probs = pm))

dat1[mVec, "y"] <- NA

## Subset the data:
yMis <- dat1[mVec, ]
yObs <- dat1[!mVec, ]
```

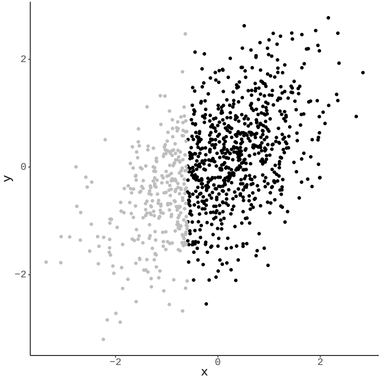
Look at the Data

```
round(head(dat0, n = 5), 3)

       y      x      z
1  0.094 -0.743  0.191
2  1.666  0.542 -0.181
3  0.208  0.617  0.711
4  0.133 -0.827  0.430
5 -0.003  0.441  0.719
```



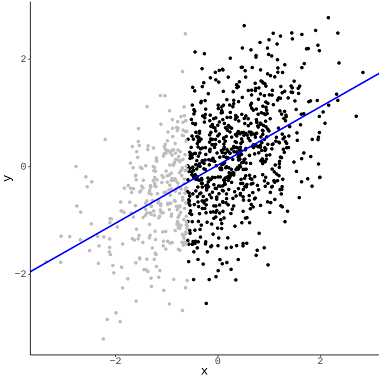
Look at the Data

```
round(head(dat1, n = 5), 3)

       y      x      z
1     NA -0.743  0.191
2  1.666  0.542 -0.181
3  0.208  0.617  0.711
4     NA -0.827  0.430
5 -0.003  0.441  0.719
```



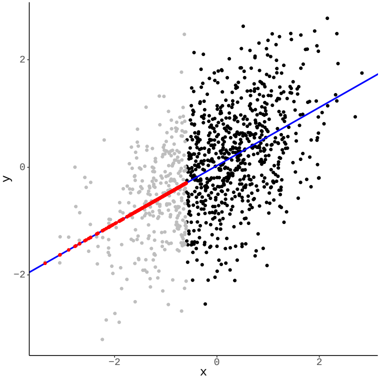
Expected Imputation Model Parameters

```
lsFit <- lm(y ~ x + z, data = yObs)

beta  <- coef(lsFit)
sigma <- summary(lsFit)$sigma

as.matrix(beta)

                     [,1]
(Intercept)  0.03249194
x            0.56993731
z           -0.12558749

sigma

[1] 0.8452583
```



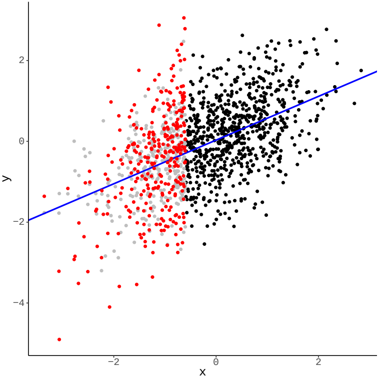
Conditional Mean Substitution

```
## Generate imputations:
imps <- beta[1] +
    beta[2] * yMis[ , "x"] +
    beta[3] * yMis[ , "z"]

## Fill missing cells in Y:
dat1[mVec, "y"] <- imps

round(head(dat1, n = 5), 3)

       y      x      z
1 -0.415 -0.743  0.191
2  1.666  0.542 -0.181
3  0.208  0.617  0.711
4 -0.493 -0.827  0.430
5 -0.003  0.441  0.719
```



Stochastic Regression Imputation

```
## Generate imputations:
imps <- imps +
    rnorm(nrow(yMis), 0, sigma)

## Fill missing cells in Y:
dat1[mVec, "y"] <- imps

round(head(dat1, n = 5), 3)

       y      x      z
1 -0.563 -0.743  0.191
2  1.666  0.542 -0.181
3  0.208  0.617  0.711
4  1.043 -0.827  0.430
5 -0.003  0.441  0.719
```



Flavors of MI

MI simply repeats a single regression imputation M times.

- The specifics of the underlying regression imputation are important.

Flavors of MI

MI simply repeats a single regression imputation M times.

- The specifics of the underlying regression imputation are important.

Simply repeating the stochastic regression imputation procedure described above won't suffice.

- Still produces too many Type I errors

	Complete Data	PN-Type	PNE-Type
cor(X, Y)	0.499	0.499	0.498
Type I Error	0.040	0.066	0.046

Mean Correlation Coefficients and Type I Error Rates

- Type I error rates for PN-Type MI are much better than they were for single stochastic regression imputation, but they're still too high.

Proper MI

The problems on the previous slide arise from using the same regression coefficients to create each of the M imputations.

- Implies that you're using the "correct" coefficients.
- This assumption is plainly ridiculous.
  - If we don't know some values of our outcome variable, how can we know the "correct" coefficients to link the incomplete outcome to the observed predictors?

Proper MI

The problems on the previous slide arise from using the same regression coefficients to create each of the M imputations.

- Implies that you're using the "correct" coefficients.
- This assumption is plainly ridiculous.
  - If we don't know some values of our outcome variable, how can we know the "correct" coefficients to link the incomplete outcome to the observed predictors?
- Proper MI also models uncertainty in the regression coefficients used to create the imputations.
  - A different set of of coefficients is randomly sampled (using Bayesian simulation) to create each of the M imputations.
  - The tricky part about implemented MI is deriving the distributions from which to sample these coefficients.

Setting Up Proper MI

Our imputation model is simply a linear regression model:

$$Y = \mathbf{X}\beta + \varepsilon$$

To fully account for model uncertainty, we need to randomly sample both β and $\text{var}(\varepsilon) = \sigma^2$.

- Question: Why do we only sample σ^2 and not ε?

Setting Up Proper MI

Our imputation model is simply a linear regression model:

$$Y = \mathbf{X}\beta + \varepsilon$$

To fully account for model uncertainty, we need to randomly sample both β and $\text{var}(\varepsilon) = \sigma^2$.

- Question: Why do we only sample σ^2 and not ε?

For a simple imputation model with a normally distributed outcome and uninformative priors, we need to specify two distributions:

1. The marginal posterior distribution of σ^2
2. The conditional posterior distribution of β

Marginal Distribution of σ^2

We first specify the marginal posterior distribution for the noise variance, σ^2.

- This distribution does not depend on any other parameters.

$$\sigma^2 \sim \text{Inv-}\chi^2\left(N - P, MSE\right) \tag{1}$$

$$\text{with } MSE = \frac{1}{N-P}\left(Y - \mathbf{X}\hat{\beta}_{ls}\right)^T\left(Y - \mathbf{X}\hat{\beta}_{ls}\right)$$

- σ^2 follows a scaled inverse χ^2 distribution.

Conditional Distribution of β

We then specify the conditional posterior distribution for β.

- This distribution is conditioned on a specific value of σ^2.

$$\beta \sim \text{MVN}\left(\hat{\beta}_{ls},\ \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\right) \tag{2}$$

- β (conditionally) follows a multivariate normal distribution.

PPD of the Missing Data

Once we've sampled our imputation model parameters, we can construct the posterior predictive distribution of the missing data.

- This is the distribution from which we sample our imputed values.
- In practice, we directly compute the imputations based on the simulated imputation model parameters.

$$
\begin{gathered}
Y_{imp} = \mathbf{X}_{mis}\tilde{\beta} + \tilde{\varepsilon} \\
\text{with } \varepsilon \sim \text{N}\left(0, \widetilde{\sigma^2}\right)
\end{gathered} \tag{3}
$$

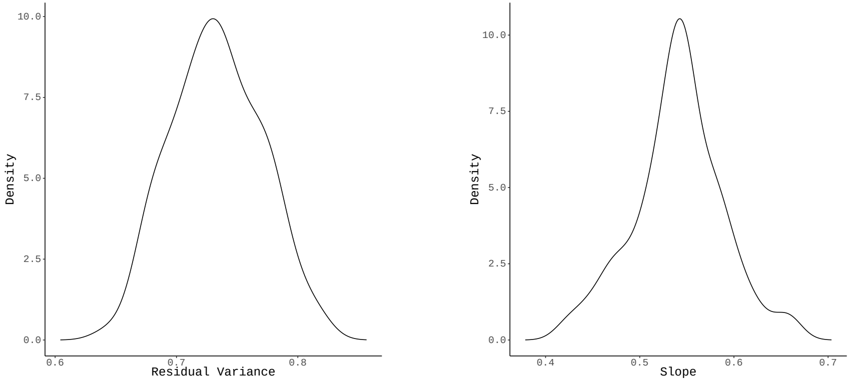
General Steps for Basic MI

With all of the elements in place, we can execute a basic MI by following these steps:

1. Find the least squares estimates of β, $\hat{\beta}_{ls}$, by regressing the observed portion of Y onto the the analogous rows of $\mathbf{X}$.
2. Use $\hat{\beta}_{ls}$ to parameterize the posterior distribution of σ^2, given by Equation 1, and draw M samples of σ^2 from this distribution.
3. For each of the σ^2_m, sample a corresponding value of β from Equation 2.
4. Plug the M samples of β and σ^2 into Equation 3 to create the M imputations.

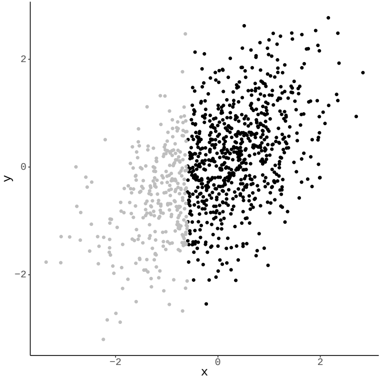
Visualizing MI

Use Bayesian simulation to estimate posterior distributions for the imputation model parameters:



Visualizing MI

Recall the incomplete data from the single imputation examples.

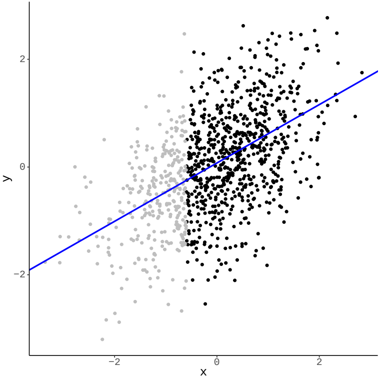


Visualizing MI

Sample values of β_0 and β_1:

- $\beta_0 = 0.075$
- $\beta_1 = 0.542$

Define the predicted best-fit line:
$\hat{Y}_{mis} = 0.075 + 0.542X_{mis}$



Visualizing MI

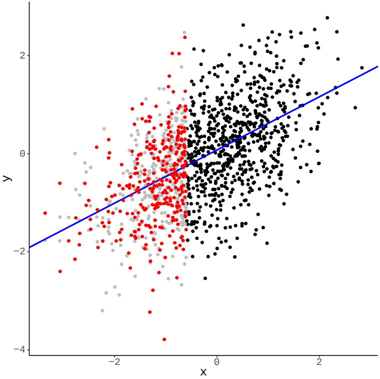
Sample a value of σ^2:

- $\sigma^2 = 0.709$

Generate imputations using the same procedure described in Single Stochastic Regression Imputation:

$$Y_{imp} = \hat{Y}_{mis} + \varepsilon$$

$$\varepsilon \sim \mathrm{N}(0, 0.709)$$

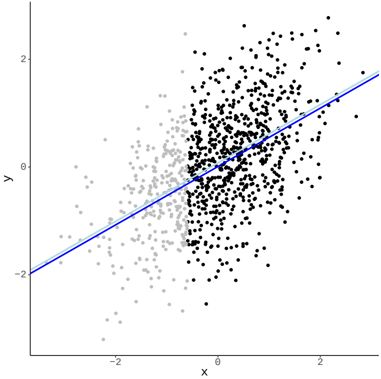


Visualizing MI

Sample values of β_0 and β_1:

- $\beta_0 = 0.009$
- $\beta_1 = 0.542$

Define the predicted best-fit line:
$\hat{Y}_{mis} = 0.009 + 0.542X_{mis}$



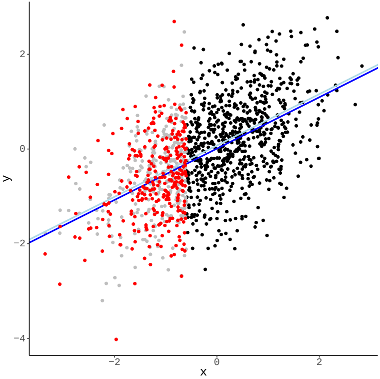
Visualizing MI

Sample a value of σ^2:

- $\sigma^2 = 0.713$

Generate imputations using the same procedure described in Single Stochastic Regression Imputation:

$$Y_{imp} = \hat{Y}_{mis} + \varepsilon$$
$$\varepsilon \sim \text{N}(0, 0.713)$$

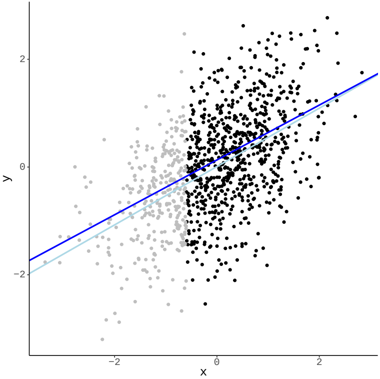


Visualizing MI

Sample values of β_0 and β_1:

- $\beta_0 = 0.132$
- $\beta_1 = 0.509$

Define the predicted best-fit line:
$\hat{Y}_{mis} = 0.132 + 0.509X_{mis}$



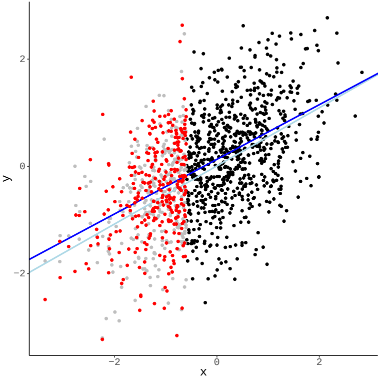
Visualizing MI

Sample a value of σ^2:

- $\sigma^2 = 0.689$

Generate imputations using the same procedure described in Single Stochastic Regression Imputation:

$$Y_{imp} = \hat{Y}_{mis} + \varepsilon$$
$$\varepsilon \sim \mathrm{N}(0, 0.689)$$

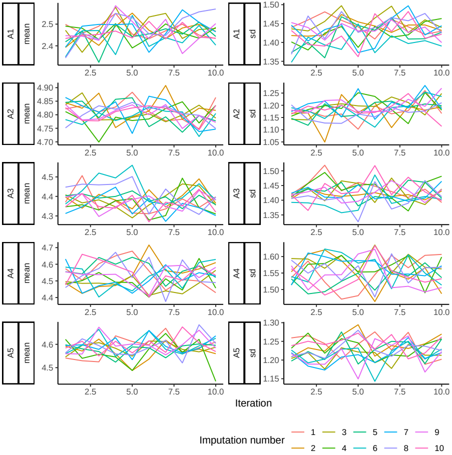


Example

```
## Impute the missing data 10 times:
miceOut <- mice(data   = bfi,
                m      = 10,
                maxit  = 10,
                method = "pmm",
                seed   = 235711,
                print  = FALSE)

## Check convergence via PSR factor:
rhat(miceOut)

  Variable MissProp RHat_Mean RHat_Variance
1       A1 30.21429 1.0097875      1.118038
2       A2 31.46429 0.9937958      1.016108
3       A3 30.17857 1.1010929      1.063382
4       A4 30.03571 1.0307582      1.039938
5       A5 30.75000 1.0201913      1.084875
```

Example

```
ggmice::plot_trace(miceOut)
```

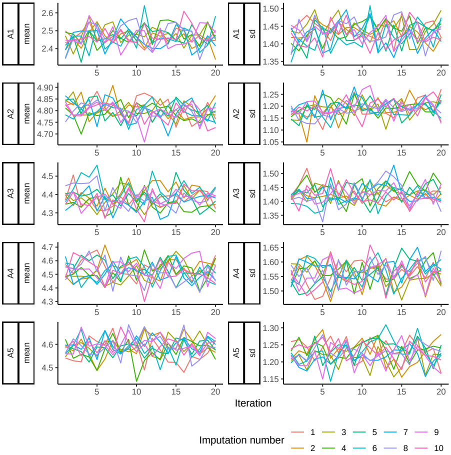


Example

```
## Extend the Markov chains by 10 interations:
miceOut <- mice.mids(miceOut, maxit = 10, printFlag = FALSE)

## Check convergence via PSR factor:
rhat(miceOut)

  Variable MissProp RHat_Mean RHat_Variance
1       A1 30.21429 0.9991332     1.0269467
2       A2 31.46429 0.9974385     1.0004546
3       A3 30.17857 1.0990821     1.0404245
4       A4 30.03571 1.0017656     0.9991882
5       A5 30.75000 1.0302434     1.0520755
```

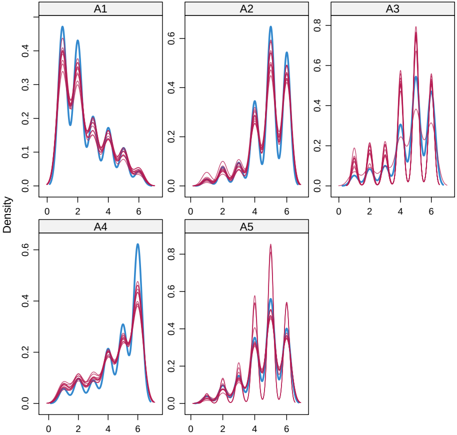
Example

```r
ggmice::plot_trace(miceOut)
```



Example

```
mice::densityplot(miceOut)
```



Doing MI-Based Analysis

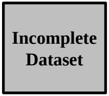
An MI-based data analysis consists of three phases:

1. The imputation phase
   - Replace missing values with M plausible estimates.
   - Produce M completed datasets.

2. The analysis phase
   - Estimate M replicates of your analysis model.
   - Fit the same model to each of the M datasets from Step 1.

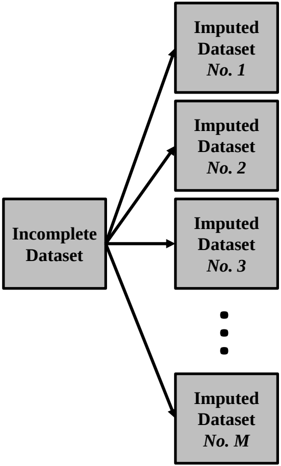
3. The pooling phase
   - Combine the M sets of parameter estimates and standard errors from Step 2 into a single set of MI estimates.
   - Use these pooled parameter estimates and standard errors for inference.

MI-Based Analysis



Incomplete Dataset

MI-Based Analysis



Incomplete Dataset

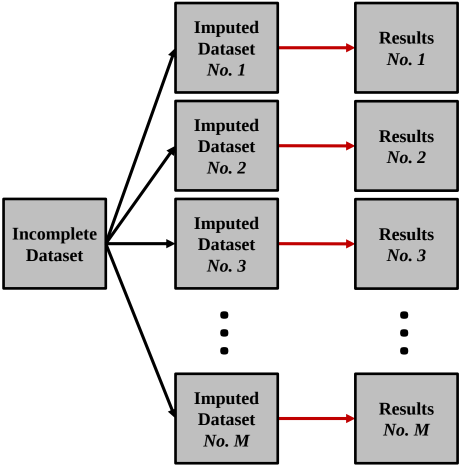
Imputed Dataset *No. 1*

Imputed Dataset *No. 2*

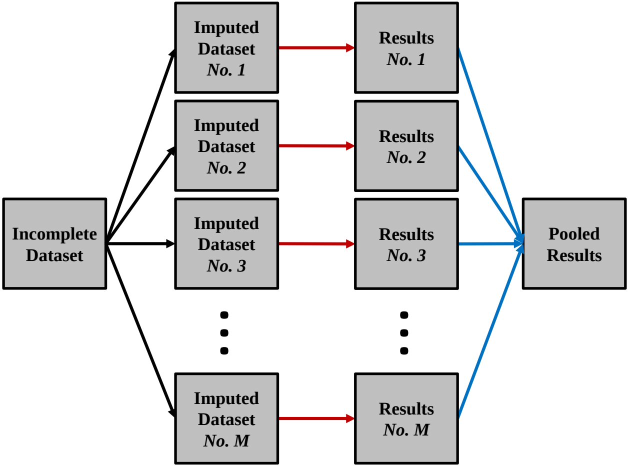
Imputed Dataset *No. 3*

Imputed Dataset *No. M*

MI-Based Analysis



MI-Based Analysis



Pooling MI Estimates

Rubin (1987) formulated a simple set of pooling rules for MI estimates.

- The MI point estimate of some interesting quantity, Q^*, is simply the mean of the M estimates, $\{\hat{Q}_m\}$:

$$Q^* = \frac{1}{M} \sum_{m=1}^{M} \hat{Q}_m$$

Pooling MI Estimates

The MI variability estimate, T, is a slightly more complex entity.

- A weighted sum of the *within-imputation* variance, W, and the *between-imputation* variance, B.

$$
\begin{aligned}
W &= \frac{1}{M} \sum_{m=1}^{M} \widehat{SE}_{Q,m}^{2} \\
B &= \frac{1}{M-1} \sum_{m=1}^{M} \left(\hat{Q}_m - Q^*\right)^2 \\
T &= W + \left(1 + M^{-1}\right) B \\
&= W + B + \frac{B}{M}
\end{aligned}
$$

Inference with MI Estimates

After computing Q^* and T, we combine them in the usual way to get test statistics and confidence intervals.

$$t = \frac{Q^* - Q_0}{\sqrt{T}}$$

$$CI = Q^* \pm t_{crit}\sqrt{T}$$

We must take care with our *df*, though.

$$df = (M - 1)\left[1 + \frac{W}{(1 + M^{-1})\,B}\right]^2$$

Fraction of Missing Information

In Lecture 4, we briefly discussed a very desirable measure of nonresponse: *fraction of missing information* (FMI).

$$FMI = \frac{r + \frac{2}{(df+3)}}{r+1} \approx \frac{(1+M^{-1})B}{(1+M^{-1})B + W} \rightarrow \frac{B}{B+W}$$

where

$$r = \frac{(1+M^{-1})B}{W}$$

The FMI gives us a sense of how much the missing data (and their treatment) have influence our parameter estimates.

- We should report the FMI for an estimated parameter along with other statistics (e.g., t-tests, p-values, effect sizes, etc.).

Example: Analysis & Pooling

Analyze the multiply imputed datasets and pool results:

```r
library(semTools)

## semTools::cfa.mi() will do both the analysis and pooling phases:
miOut <-
    cfa.mi(cfaMod, data = complete(miceOut, "all"), std.lv = TRUE)
```

Example: Analysis & Pooling

```
Latent Variables:
                   Estimate  Std.Err  t-value  P(>|t|)      FMI
  agree =~
    A1                0.560    0.034   16.331    0.000    0.303
    A2               -0.770    0.027  -28.983    0.000    0.556
    A3               -1.014    0.030  -33.803    0.000    0.556
    A4               -0.708    0.035  -19.977    0.000    0.399
    A5               -0.817    0.028  -28.815    0.000    0.280
  open =~
    O1                0.635    0.030   21.272    0.000    0.004
    O2               -0.633    0.041  -15.333    0.000    0.014
    O3                0.838    0.033   25.309    0.000    0.026
    O4                0.344    0.033   10.566    0.000    0.014
    O5               -0.641    0.035  -18.367    0.000    0.010
```

Example: Analysis & Pooling

```
Covariances:
                   Estimate  Std.Err  t-value  P(>|t|)       FMI
  agree ~~
    open             -0.299    0.029  -10.220    0.000     0.305

Variances:
                   Estimate  Std.Err  t-value  P(>|t|)       FMI
   .A1                1.697    0.056   30.215    0.000     0.389
   .A2                0.776    0.032   24.268    0.000     0.325
   .A3                0.771    0.041   18.707    0.000     0.423
   .A4                1.725    0.059   29.171    0.000     0.373
   .A5                0.889    0.036   24.420    0.000     0.469
   .O1                0.878    0.037   23.974    0.000     0.004
   .O2                2.048    0.071   28.769    0.000     0.008
   .O3                0.785    0.046   17.043    0.000     0.037
   .O4                1.369    0.045   30.594    0.000     0.003
   .O5                1.357    0.050   26.897    0.000     0.008
```

Model Fit

We cannot simply average the χ^2 statistics as we do the parameter estimates.

- We need to do some fancy processing to get a correctly distributed fit statistic.

Three different formulations have been proposed, and they are typically designated D_1, D_2, and D_3.

1. D_1
2. D_2
3. D_3

Model Fit

```
what <- c("chisq", "df", "pvalue", "fmi", "rmsea", "cfi", "tli")
fitMeasures(miOut, test = "D3")[what] %>% round(3)

  chisq       df   pvalue      fmi    rmsea      cfi      tli
279.070   34.000    0.000    0.417    0.051    0.909    0.880

fitMeasures(miOut, test = "D2")[what] %>% round(3)

  chisq       df   pvalue      fmi    rmsea      cfi      tli
243.995   34.000    0.000    0.472    0.047    0.926    0.902
```

Model Fit

```
## Define a restricted model:
cfaMod2 <- paste(cfaMod, "agree ~~ 0 * open", sep = "\n")

## Estimate the restricted model and pool the results:
miOut2 <-
    cfa.mi(cfaMod2, data = complete(miceOut, "all"), std.lv = TRUE)

## Test the constraint via nested model comparison:
anova(miOut2, miOut, test = "D3")

     F    df1    df2 pvalue   ariv    fmi
93.961  1.000 62.196  0.000  0.323  0.244

anova(miOut2, miOut, test = "D2")

      F      df1      df2  pvalue    ariv     fmi
 89.292    1.000 114.627   0.000   0.389   0.280
```

Model Fit

```
## Define a model with some parameter labels:
cfaMod3 <- "
agree =~ l11 * A1 + l21 * A2 + l31 * A3 + l41 * A4 + l51 * A5
open  =~ l12 * O1 + l22 * O2 + l32 * O3 + l42 * O4 + l52 * O5
"

## Estimate the model and pool the results:
miOut3 <-
    cfa.mi(cfaMod3, data = complete(miceOut, "all"), std.lv = TRUE)

## Define some constraints:
cons <- "
l11 == l12
l21 == l22
l31 == l32
l41 == l42
l51 == l52
"
```

Model Fit

```
## Test the constraints via multivariate Wald tests:
lavTestWald.mi(miOut3, constraints = cons, test = "D1")

      F     df1     df2  pvalue    ariv     fmi
413.268   5.000 570.423   0.000   0.352   0.260

lavTestWald.mi(miOut3, constraints = cons, test = "D2")

      F     df1     df2  pvalue    ariv     fmi
435.383   5.000 105.124   0.000   0.298   0.230
```

Model Fit

```
## Define a model with some parameter labels:
cfaMod4 <- paste(cfaMod3, cons, sep = "\n")

## Estimate the model and pool the results:
miOut4 <-
    cfa.mi(cfaMod4, data = complete(miceOut, "all"), std.lv = TRUE)

anova(miOut3, miOut4, test = "D2")

      F     df1     df2  pvalue    ariv     fmi
404.410   5.000 178.423   0.000   0.214   0.176
```

Model Fit

```r
## Define some syntax to allow cross-loadings from the "agree"
## factor to the "open" items:
adds <- "agree =~ O1 + O2 + O3"

## Test the parameter relaxations via a score test:
stD2 <- lavTestScore.mi(miOut, add = adds, test = "D2")
stD1 <- lavTestScore.mi(miOut, add = adds, test = "D1")
```

Model Fit

```
stD2

$test

total score test:

   test      F df1     df2 p.value  ariv   fmi
1 score 21.809   3 213.475       0 0.211 0.174

$uni

univariate score tests:

        lhs op rhs      F df1     df2 p.value   riv   fmi
1 agree=~O1 ==   0  3.954   1 681.076   0.047 0.130 0.115
2 agree=~O2 ==   0 40.767   1 418.527   0.000 0.172 0.147
3 agree=~O3 ==   0 29.511   1 335.876   0.000 0.196 0.164
```

Model Fit

```
stD1

$test

total score test:

   test      F df1      df2 p.value  ariv   fmi
1 score 23.414   3 1503.432       0 0.131 0.116

$uni

univariate score tests:

        lhs op rhs      F df1     df2 p.value   riv   fmi
1 agree=~01 ==   0  3.989   1 467.030   0.046 0.090 0.083
2 agree=~02 ==   0 42.262   1 277.159   0.000 0.122 0.108
3 agree=~03 ==   0 31.201   1 474.650   0.000 0.089 0.082
```

ML-Based Solutions

FIML Intuition

FIML is an ML estimation method that is robust to ignorable nonresponse.

- FIML partitions the missing information out of the likelihood function so that the model is only estimated from the observed parts of the data.

After a minor alteration to the likelihood function, FIML reduces to simple ML estimation.

- So, let's review ML estimation before moving forward.

Maximum Likelihood Estimation

ML estimation simply finds the parameter values that are “most likely” to have given rise to the observed data.

- The *likelihood* function is just a probability density (or mass) function with the data treated as fixed and the parameters treated as random variables.
- Having such a framework allows us to ask: “Given that I’ve observed these data values, what parameter values most probably describe these data?”

Maximum Likelihood Estimation

ML estimation is usually employed when there is no closed form solution for the parameters we seek.

- This is why you don’t usually see ML used to fit general linear models.

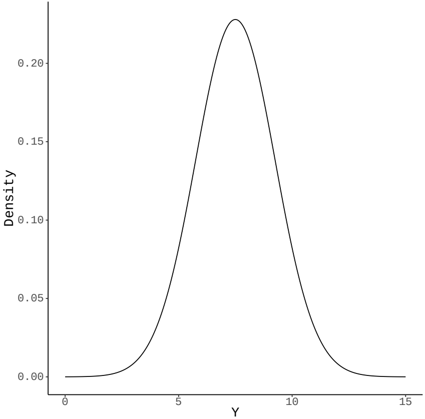
After choosing a likelihood function, we iteratively optimize the function to produce the ML estimated parameters.

- In practice, we nearly always work with the natural logarithm of the likelihood function (i.e., the *loglikelihood*).

Likelihoods

Suppose we have the following model:

$$Y \sim \mathrm{N}\left(\mu, \sigma^2\right).$$



Likelihoods

For a given Y_n, we have:

$$P\left(Y_n|\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(Y_n-\mu)^2}{2\sigma^2}}. \tag{4}$$

If we plug estimated parameters into Equation 4, we get the probability of observing Y_n given $\hat{\mu}$ and $\hat{\sigma}^2$:

$$P\left(Y_n|\hat{\mu},\hat{\sigma}^2\right) = \frac{1}{\sqrt{2\pi\hat{\sigma}^2}} e^{-\frac{(Y_n-\hat{\mu})^2}{2\hat{\sigma}^2}}. \tag{5}$$

Applying Equation 5 to all N observations and multiplying the results produces a *likelihood*:

$$\hat{L}\left(\hat{\mu},\hat{\sigma}^2\right) = \prod_{n=1}^{N} P\left(Y_n|\hat{\mu},\hat{\sigma}^2\right).$$

Likelihoods

We generally want to work with the natural logarithm of Equation 5. Doing so gives the *loglikelihood*:

$$\begin{aligned}\hat{\mathcal{L}}\left(\hat{\mu},\hat{\sigma}^2\right) &= \ln \prod_{n=1}^{N} P\left(Y_n|\hat{\mu},\hat{\sigma}^2\right) \\ &= -\frac{N}{2}\ln 2\pi - N\ln\hat{\sigma} - \frac{1}{2\hat{\sigma}^2}\sum_{n=1}^{N}\left(Y_n-\hat{\mu}\right)^2\end{aligned}$$

ML tries to find the values of $\hat{\mu}$ and $\hat{\sigma}^2$ that maximize $\hat{\mathcal{L}}\left(\hat{\mu},\hat{\sigma}^2\right)$.

- Find the values of $\hat{\mu}$ and $\hat{\sigma}^2$ that are *most likely*, given the observed values of Y.

Likelihoods

Suppose we have a linear regression model:

$$Y = \beta_0 + \beta_1 X + \varepsilon,$$

$$\varepsilon \sim \mathrm{N}\left(0, \sigma^2\right).$$

This model can be equivalently written as:

$$Y \sim \mathrm{N}\left(\beta_0 + \beta_1 X, \sigma^2\right)$$

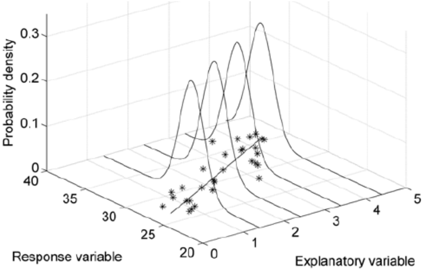


Image retrieved from: `http://www.seaturtle.org/mtn/archives/mtn122/mtn122p1.shtml`

Likelihoods

For a given $\{Y_n, X_n\}$, we have:

$$P\left(Y_n | X_n, \beta_0, \beta_1, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\left(Y_n - \beta_0 - \beta_1 X_n\right)^2}{2\sigma^2}}. \tag{6}$$

If we plug our estimated parameters into Equation 6, we get the probability of observing Y_n given $\hat{Y}_n = \hat{\beta}_0 + \hat{\beta}_1 X_n$ and $\hat{\sigma}^2$.

$$P\left(Y_n | X_n, \hat{\beta}_0, \hat{\beta}_1, \hat{\sigma}^2\right) = \frac{1}{\sqrt{2\pi\hat{\sigma}^2}} e^{-\frac{\left(Y_n - \hat{\beta}_0 - \hat{\beta}_1 X_n\right)^2}{2\hat{\sigma}^2}} \tag{7}$$

Likelihoods

So, our final loglikelihood function would be the following:

$$\begin{aligned}\hat{\mathcal{L}}\left(\hat{\beta}_0, \hat{\beta}_1, \hat{\sigma}^2\right) &= \ln \prod_{n=1}^{N} P\left(Y_n | X_n, \hat{\beta}_0, \hat{\beta}_1, \hat{\sigma}^2\right) \\ &= -\frac{N}{2} \ln 2\pi - N \ln \hat{\sigma} - \frac{1}{2\hat{\sigma}^2} \sum_{n=1}^{N} \left(Y_n - \hat{\beta}_0 - \hat{\beta}_1 X_n\right)^2 .\end{aligned}$$

Example

```
## Fit a model:
out1 <- lm(ldl ~ bp + glu + bmi, data = diabetes)

## Extract the predicted values and estimated residual
## standard error:
yHat <- predict(out1)
s    <- summary(out1)$sigma

## Compute the row-wise probabilities:
pY <- dnorm(diabetes$ldl, mean = yHat, sd = s)

## Compute the loglikelihood, and compare to R's version:
sum(log(pY)); logLik(out1)[1]

[1] -2109.939
[1] -2109.93
```

Multivariate Normal Distribution

The PDF for the multivariate normal distribution is:

$$P(\mathbf{Y}|\boldsymbol{\mu},\Sigma) = \frac{1}{\sqrt{(2\pi)^P|\Sigma|}} e^{-\frac{1}{2}(\mathbf{Y}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{Y}-\boldsymbol{\mu})}.$$

So, the multivariate normal loglikelihood is:

$$\mathcal{L}\left(\boldsymbol{\mu},\Sigma\right) = -\left[\frac{P}{2}\ln(2\pi) + \frac{1}{2}\ln|\Sigma| + \frac{1}{2}\right]\sum_{n=1}^{N}(\mathbf{Y}_n - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{Y}_n - \boldsymbol{\mu}).$$

Which can be further simplified if we multiply through by -2:

$$-2\mathcal{L}\left(\boldsymbol{\mu},\Sigma\right) = \left[P\ln(2\pi) + \ln|\Sigma|\right]\sum_{n=1}^{N}(\mathbf{Y}_n - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{Y}_n - \boldsymbol{\mu}).$$

Steps of ML

1. Choose a probability distribution, $f(Y|\theta)$, to describe the distribution of the data, Y, given the parameters, θ.
2. Choose some estimate of θ, $\hat{\theta}^{(i)}$.
3. Compute each row's contribution to the loglikelihood function by evaluating: $\ln\left[f\left(Y_n|\hat{\theta}^{(i)}\right)\right]$.
4. Sum the individual loglikelihood contributions from Step 3 to find the loglikelihood value, $\hat{\mathcal{L}}$.
5. Choose a "better" estimate of the parameters, $\hat{\theta}^{(i+1)}$, and repeat Steps 3 and 4.
6. Repeat Steps 3 – 5 until the change between $LL^{(i-1)}$ and $LL^{(i)}$ falls below some trivially small threshold.
7. Take $\hat{\theta}^{(i)}$ as your estimated parameters.

ML Example

Recall the nth observation's contribution to the multivariate normal loglikelihood function:

$$\mathcal{L}\left(\boldsymbol{\mu}, \Sigma\right)_n = -\frac{P}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma| - \frac{1}{2}(\mathbf{Y}_n - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{Y}_n - \boldsymbol{\mu}).$$

It turns out that this function is readily available in R via the **mvtnorm** package:

```r
## Vector of row-wise contributions to the overall LL:
ll0 <- dmvnorm(y, mean = mu, sigma = sigma, log = TRUE)
```

ML Example

We can wrap the preceding code in a nice R function:

```
## Complete data loglikelihood function:
ll <- function(par, data) {
    ## Extract the parameter matrices:
    p  <- ncol(data)
    mu <- par[1:p]

    ## Populate sigma from its Cholesky factor:
    sigma <- vecChol(tail(par, -p), p = p, revert = TRUE)

    ## Compute the row-wise contributions to the LL:
    ll0 <- dmvnorm(data, mean = mu, sigma = sigma, log = TRUE)

    sum(ll0)# return the overall LL value
}
```

ML Example

We’ll also need the following helper function:

```
## Convert from covariance matrix to vectorized Cholesky factor and back.
vecChol <- function(x, p, revert = FALSE) {
    if(revert) {
        tmp                  <- matrix(0, p, p)
        tmp[!lower.tri(tmp)] <- x
        crossprod(tmp)
    }
    else
        chol(x)[!lower.tri(x)]
}
```

ML Example

The **optimx** package can numerically optimize arbitrary functions.

- We can use it to (semi)manually implement ML.

```
## Subset the 'diabetes' data:
dat1 <- diabetes[ , c("bmi", "ldl", "glu")] %>% as.matrix()

## Choose some starting values:
m0   <- rep(0, 3)
s0   <- diag(3) %>% vecChol()
par0 <- c(m0, s0)

## Use optimx() to numerically optimize the LL function:
mle <- optimx(par     = par0,
              fn      = ll,
              data    = dat1,
              method  = "BFGS",
              control = list(maximize = TRUE, maxit = 1000)
              )

Maximizing -- use negfn and neggr
```

ML Example

Finally, let’s check convergence and extract the optimized parameters:

```
## Check convergence:
mle[c("convcode", "kkt1", "kkt2")]

     convcode kkt1 kkt2
BFGS        0 TRUE TRUE

## Get the optimize mean vector and covariance matrix:
muHat    <- mle[1:3]
sigmaHat <- mle[4:9] %>% as.numeric() %>% vecChol(p = 3, revert = TRUE)
```

ML Example

	bmi	ldl	glu
ML	26.376	115.440	91.260
Closed Form	26.376	115.439	91.260

Estimated Means

	bmi	ldl	glu
bmi	19.476	35.014	19.697
ldl	35.014	922.859	101.375
glu	19.697	101.375	131.865

ML Covariance Matrix

	bmi	ldl	glu
bmi	19.520	35.093	19.742
ldl	35.093	924.955	101.605
glu	19.742	101.605	132.166

Closed Form Covariance Matrix

From ML to FIML

The nth observation's contribution to the multivariate normal loglikelihood function would be the following:

$$\mathcal{L}\left(\boldsymbol{\mu}, \Sigma\right)_n = -\frac{P}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma| - \frac{1}{2}(\mathbf{Y}_n - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{Y}_n - \boldsymbol{\mu}). \tag{8}$$

From ML to FIML

The nth observation's contribution to the multivariate normal loglikelihood function would be the following:

$$\mathcal{L}\left(\boldsymbol{\mu}, \Sigma\right)_n = -\frac{P}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma| - \frac{1}{2}(\mathbf{Y}_n - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{Y}_n - \boldsymbol{\mu}). \tag{8}$$

FIML just tweaks Equation 8 a tiny bit:

$$\mathcal{L}\left(\boldsymbol{\mu}, \Sigma\right)_{fiml,n} = -\frac{P}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma_q| - \frac{1}{2}(\mathbf{Y}_n - \boldsymbol{\mu}_q)^T\Sigma_q^{-1}(\mathbf{Y}_n - \boldsymbol{\mu}_q).$$

Where $q = 1, 2, \ldots, Q$ indexes response patterns.

FIML Example

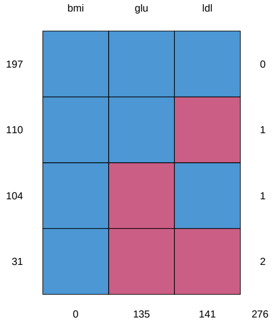
First things first, we need to punch some holes in our example data.

```r
## Impose MAR missing:
dat2 <- imposeMissData(data    = dat1,
                       targets = c("ldl", "glu"),
                       preds   = "bmi",
                       pm      = 0.3,
                       types   = c("low", "high"),
                       stdDat  = TRUE)
```

Visualize the Response Patterns

The data contain 4 unique response patterns.

- We'll define 4 different version of μ and Σ.
- We'll calculate each individual loglikelihood contributions using the appropriate flavor of μ and Σ.



FIML Example

```r
## Compute the within-pattern contributions to the LL:
ll0 <- function(i, mu, sigma, pats, ind, data) {
    ## Define the current response pattern:
    p1 <- pats[i, ]

    if(sum(p1) > 1) # More than one observed variable?
        dmvnorm(x     = data[ind == i, p1],
                mean  = mu[p1],
                sigma = sigma[p1, p1],
                log   = TRUE)
    else
        dnorm(x    = data[ind == i, p1],
              mean = mu[p1],
              sd   = sqrt(sigma[p1, p1]),
              log  = TRUE)
}
```

FIML Example

```r
## FIML loglikelihood function:
llm <- function(par, data, pats, ind) {
    ## Extract the parameter matrices:
    p  <- ncol(data)
    mu <- par[1:p]

    ## Populate sigma from its Cholesky factor:
    sigma <- vecChol(tail(par, -p), p = p, revert = TRUE)

    ## Compute the pattern-wise contributions to the LL:
    ll1 <- sapply(X     = 1:nrow(pats),
                  FUN   = ll0,
                  mu    = mu,
                  sigma = sigma,
                  pats  = pats,
                  ind   = ind,
                  data  = data)

    sum(unlist(ll1))
}
```

FIML Example

```r
## Summarize response patterns:
pats <- uniquecombs(!is.na(dat2))
ind  <- attr(pats, "index")

## Choose some starting values:
m0   <- colMeans(dat2, na.rm = TRUE)
s0   <- cov(dat2, use = "pairwise") %>% vecChol()
par0 <- c(m0, s0)

## Use optimx() to numerically optimize the LL function:
mle <- optimx(par     = par0,
              fn      = llm,
              data    = dat2,
              pats    = pats,
              ind     = ind,
              method  = "BFGS",
              control = list(maximize = TRUE, maxit = 1000)
              )
```

FIML Example

Check convergence and extract the optimized parameters:

```
## Check convergence:
mle[c("convcode", "kkt1", "kkt2")]

     convcode kkt1 kkt2
BFGS        0 TRUE TRUE

## Get the optimize mean vector and covariance matrix:
muHat1    <- mle[1:3]
sigmaHat1 <- mle[4:9] %>% as.numeric() %>% vecChol(p = 3, revert = TRUE)
```

FIML Example

Just to make sure our results are plausible, we can do the same analysis using the `cfa()` function from the **lavaan** package:

```
## Define the model in lavaan syntax:
mod <- "
bmi ~~ ldl + glu
ldl ~~ glu
"

## Fit the model with lavaan::cfa():
fit <- cfa(mod, data = dat2, missing = "fiml")

## Extract the estimated parameters:
muHat2    <- inspect(fit, "est")$nu
sigmaHat2 <- inspect(fit, "theta")
```

FIML Example

	bmi	ldl	glu
Manual	26.376	115.326	91.376
Lavaan	26.376	115.329	91.377

Estimated Means

	bmi	ldl	glu
bmi	19.475	26.375	22.197
ldl	26.375	933.224	113.550
glu	22.197	113.550	130.717

Manual FIML Covariance Matrix

	bmi	ldl	glu
bmi	19.476	26.395	22.199
ldl	26.395	933.277	113.579
glu	22.199	113.579	130.721

Lavaan FIML Covariance Matrix

Auxiliary Variables

Satisfying the MAR Assumption

Like MI, FIML also requires MAR data.

- Parameters will be biased with MAR is violated.

Unlike MI, FIML directly treats the missing data while estimating the analysis model.

- The MAR predictors must be included in the analysis model.
- Otherwise, FIML reduces to pairwise deletion.

If the MAR predictors are not substantively interesting variables, naively included them in the analysis model can change the model's meaning.

Saturated Correlates Technique

Graham (2003) developed the *saturated correlates* approach to meet two desiderata:

1. Satisfy the MAR assumption by incorporating MAR predictors into the analysis model.
2. Do not affect the fit or substantive meaning of the analysis model.

The approach entails incorporating the MAR predictors via a fully-saturated covariance structure:

1. Allow all MAR predictors to co-vary with all other MAR predictors.
2. Allow all MAR predictors to co-vary with all observed variables in the analysis model (or their residuals).

References

Graham, J. W. (2003). Adding missing-data-relevant variables to FIML-based structural equation models. *Structural Equation Modeling: A Multidisciplinary Journal*, *10*(1), 80–100. doi: 10.1207/S15328007SEM1001_4

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys* (Vol. 519). New York, NY: John Wiley & Sons.

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Boca Raton, FL: CRC Press.