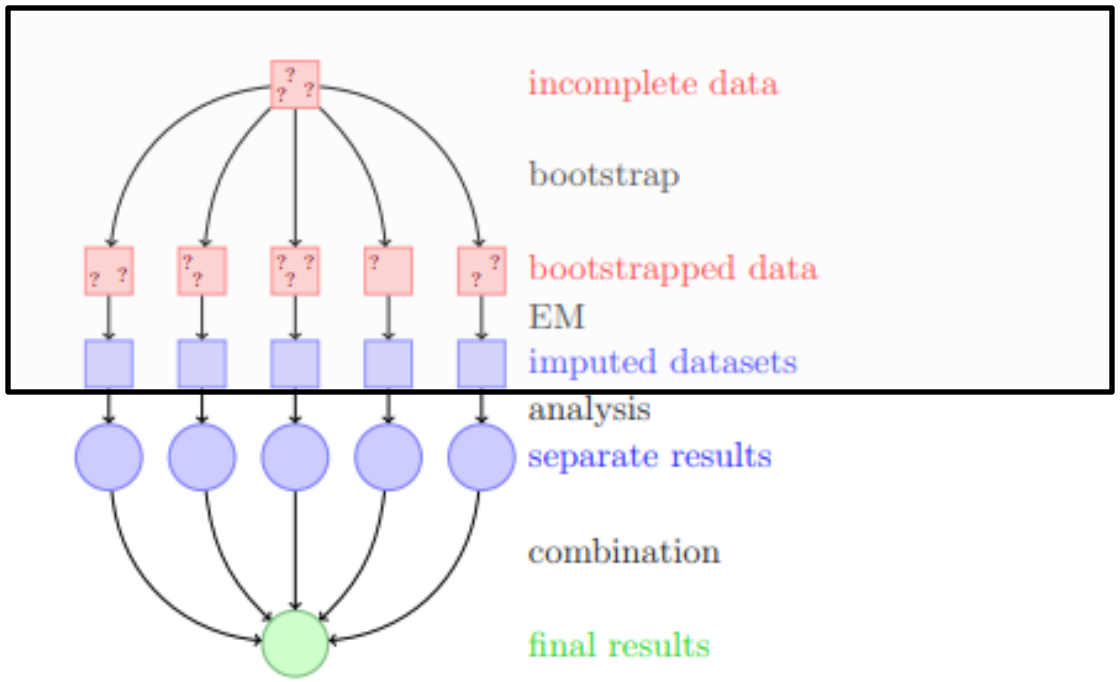
Amelia – multiple imputation in R



January 2018

Boriana Pratt, Princeton University

Missing Data

Missing data can be defined by the mechanism that leads to missingness.

Three main types of missing data ("Statistical analysis with missing data", D. Rubin, R. Little) :

- **Missing completely at random (MCAR)** – like "flipping a coin" whether to answer a question; missing data does not depend on the observed or the missing data

- **Missing at random (MAR)** – something known (from the data or another source) makes a person less likely to answer a question; missing data depends on the observed data and does not depend on non-observed data

- **Missing not at random (MNAR) (Non-ignorable missingness)–** something (not known/measured) makes a person less likely to answer a questions; the missing data depends on something not observed

Missing Data

Missing data is found quite often in practice.

How to deal with missing data?

- **Ignore**

  *listwise deletion – leads to biased estimates (**except** in the case of MCAR)*

- **Impute by guessing**

  use mean/median or nearby values – could lead to biased estimates, does not account for uncertainty of the imputed values

- **Impute probabilistically**

  multiple imputation – unbiased estimates, values are imputed with uncertainty

Multiple imputation

Steps to do multiple imputation:

1. Impute *m* values for each missing value creating *m* completed datasets.

m – between 5 and 10

2. Analyze each of these *m* completed datasets separately.

get estimates q_i (i=1,…,m) for *Q* (your quantity of interest)

3. Combine the *m* results.

take the average and adjust the SE

Multiple imputation

How to combine the *m* results:

$\overline{q} = \frac{1}{m}\sum_{i=1}^{m} q_i$ - average of estimates

for $SE_{\overline{q}}$:

- For each q_i we have $SE(q_i)$ – within variability

-$S_q^2 = \frac{\sum_{i=1}^{m}(q_i - q)^2}{m-1}$ - between variability

combine:

$(SE_{\overline{q}})^2 = \frac{1}{m}\sum_{i=1}^{m}(SE(qi))^2 + Sq^2(1 + \frac{1}{m})$

Amelia I and II

Amelia is an R package that has had two major revisions. It was originally developed by Gary King, James Honaker, Anne Joseph, and Kenneth Scheve in 2001:

- Amelia I – "Analyzing Incomplete Political Science Data: An Alternative Algorithm for Multiple Imputation" (American Political Science Review, 2001)

A newer version with GUI was released in 2011:

- Amelia II – "A program for missing Data", J. Honaker, G. King, M. Blackwell (Journal of Statistical Software, 2011)

In R/Rstudio could be installed by typing:

```
>install.packages("Amelia")
```

Then called by typing:

```
>library(Amelia)
```

If you would like to try the GUI version, type:

```
>AmeliaView()
```

Amelia – building blocks

Amelia assumes that your data is jointly distributed as multivariate normal.

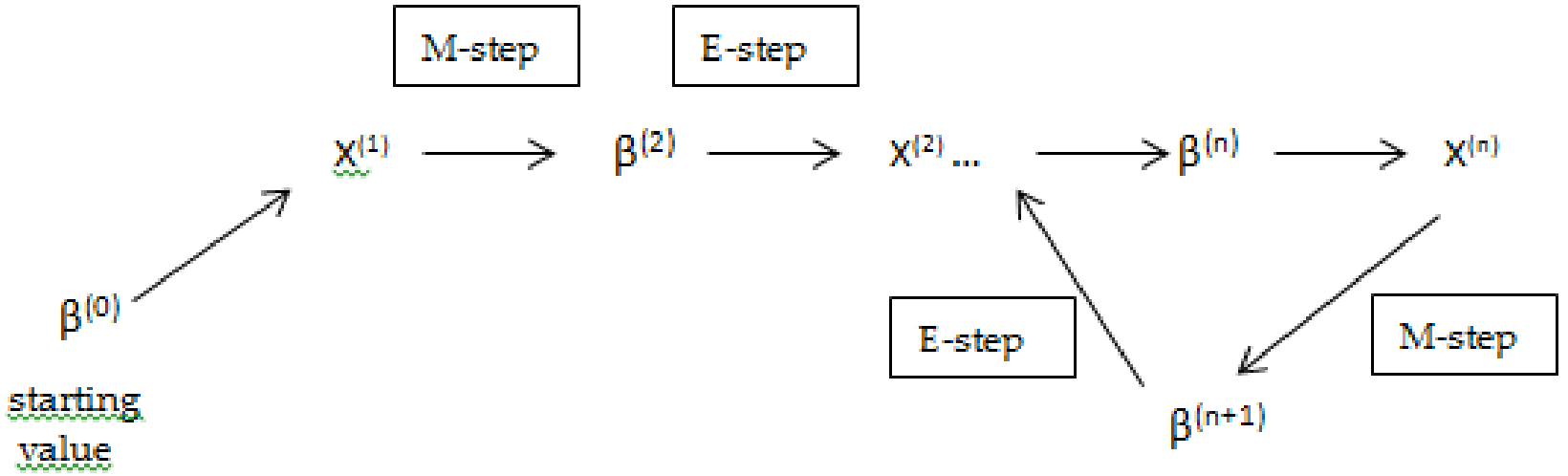
Amelia uses the following algorithm:

EM – expectation-maximization (Amelia I)

EMB – expectation-maximization with bootstraping (Amelia II)

EM algorithm

EM (expectation-maximization) algorithm alternates between an expectation (E-step) and maximization (M-step) steps until convergence. Convergence is reached when the current and previous values are close enough to each other (usually set by the analyst).



Syntax - all options

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, arglist = NULL, empri = NULL, priors = NULL, autopri = 0.05, emburn = c(0,0), bounds = NULL, max.resample = 100, overimp = NULL, boot.type = "ordinary", parallel = c("no", "multicore", "snow"), ncpus = getOption("amelia.ncpus", 1L), cl = NULL, ...)

The data

Two data files from the National Longitudinal Survey of Youth, 1979 are provided.

One is in wide format (there is a row for each person, N=11,406), the other is in long format (there is a row for each person for each survey-year).

- Nationally representative sample of 14-22-year olds surveyed every year 1979 – 1994 and then every other year until 2014
- Total people surveyed initially - 12,686 ; in 2002 – 7,724 surveyed (https://www.bls.gov/nls/handbook/2005/nlshc3.pdf)
- 1,280 – excluded (military sample, not followed after 1984),

  thus sample number is 11,406
- Longitudinal data that is provided in 'wide' format

Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, …)

The first two parameters are for specifying your data as a data frame or matrix (x) and the number of imputations (m).

For example (using data for only 1979):

```
a79<-amelia(dt79,m=5,idvars=c("id","caseid"))
```

Or (since m=5 is the default it could be ommited):

```
a79<-amelia(dt79,idvars=c("id","caseid"))
```

Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL,...)

The p2s options is for how much to be printed to the screen with 0 – for no output; the defaults is 1; and 2 – for more detailed output.

The next option - frontend is only used for GUI.

idvars is for variables that should not be included in the imputation model but would like them in the imputed datasets.

Syntax – time and cross-sectional variables

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, ...)

Next two options are for time-series / longitudinal / panel data:

ts is for declaring your time variable(by giving the column name or number) and cs is for declaring your cross-sectional(/unit) variable

Here is an example with part of the output:

```
> along.out<-amelia(dtLong,m=2,p2s=2,idvars=c("idtype"),ts="year",cs="caseid")

amelia starting
beginning prep functions
Variables used:  race sex rresidence edu wth alcohol famsize region ageatint urban_rural height ev
er_asthma anychild r_hashins tnfi_trunc povstatus jobsnum numch
running bootstrap
-- Imputation 1 --
setting up EM chain indicies

  1(189)  2(184)  3(177)  4(177)  5(169)  6(160)  7(132)  8(120)  9(103) 10(96) 11(90) 12(88) 13(8
5) 14(81) 15(77) 16(68) 17(62) 18(58) 19(52) 20
(47) 21(42) 22(38) 23(37) 24(34) 25(31) 26(30) 27(30) 28(27) 29(27) 30(27) 31(26) 32(24) 33(23) 34
(21) 35(21) 36(21) 37(20) 38(18) 39(17) 40
(17) 41(17) 42(17) 43(17) 44(15) 45(15) 46(14) 47(13) 48(12) 49(11) 50(10) 51(10) 52(10) 53(10) 54
(9) 55(7) 56(7) 57(7) 58(7) 59(6) 60
(6) 61(5) 62(4) 63(4) 64(3) 65(2) 66(2) 67(2) 68(2) 69(1) 70(0)

 saving and cleaning
```

Syntax – time and cross-sectional variables

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, ...)

The next three parameters are for controlling the smooth functions of time variable to be included in the model:

polytime is for setting the power of polynomial for time effects if any should be included (could be an integer between 0 and 3)

splinetime is for controlling the cubic smoothing splines of time (should be an integer>0)

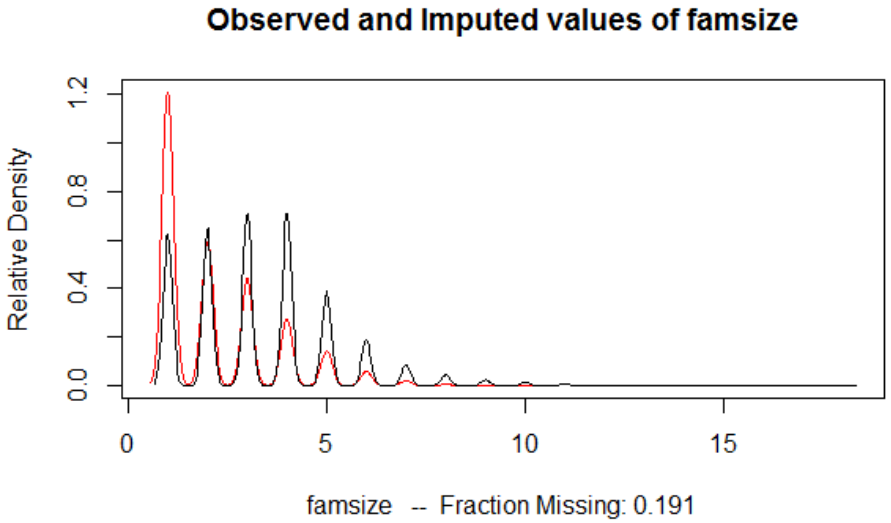
and intercs is to tell if the time effect should vary by cross-section.

Example – time and cross-sectional variables

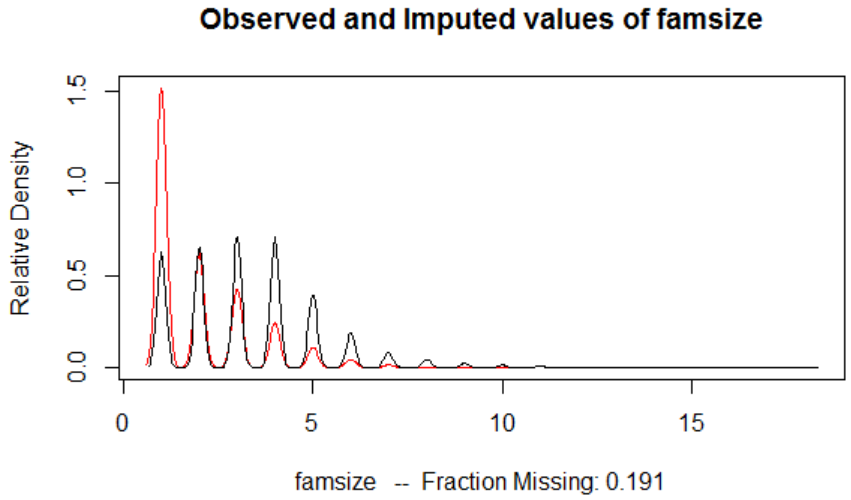
>along.out1_nt<amelia(dtLong,m=2, p2s=1,idvars=c("idtype"), ts="year", cs="caseid", noms=c("urban_rural","region","anychild","r_hashins","povstatus"), ords=c("edu","alcohol","famsize"))

>along.out1_t<amelia(dtLong,m=2, p2s=1,I dvars=c("idtype"), ts="year", cs="caseid", polytime=2, noms=c("urban_rural","region","anychild","r_hashins","povstatus"), ords=c("edu","alcohol","famsize"))

>plot(along.out1_nt,which.vars=10)



>plot(along.out1_t,which.vars=10)



Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, ...)

The lags and leads options are a way to specify columns (either by number or name) for which previous value (lag) or next value (lead) in the data should be included in the model.

Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, ...)

The startvals is for specifying starting values for the EM algorithm – there are two possible options: (the default) 0 is for using the parameter matrix from list-wise deletion model and 1 is for using the identity matrix.

The tolerance option is to specify the convergence criteria for the EM algorithm (the default here is often used in other situations when the EM algorithm is used).

Syntax – variable transformations

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, ...)

The next five options are for different types of transformations to be applied to variables in the model. These should be used when a variable is not normally distributed and the transformation would make it tend to normal distribution.

For example, a binary indicator should be listed as noms, a skewed variable would benefit from log-linear transformation (list it under logs); a count variable (always positive) could be applied the square-root transformation (list it as sqrts); ordered variable (one that, for example, gets values of 0 for 'never', 1 for 'sometimes', 2 for 'often' and 3 for 'always') should be listed as ords; logistic transformation (lgst) should be applied to proportions (usually bound between 0 and 1).

Example – variable transformations

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, ...)

```
along.out1<amelia(dtLong,m=2, p2s=1,idvars=c("idtype"), ts="year", cs="caseid",
noms=c("urban_rural","region","anychild","r_hashins","povstatus"),  ords=c("edu","alcohol","famsize")  )
```

Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, arglist = NULL, empri = NULL, priors = NULL, autopri = 0.05, emburn = c(0,0),...)

Leave the next two options- incheck and collect, as their defaults. incheck (=TRUE) will always check your input data (only if you are sure of your run and need to cut the run time down, is it advisable to change to FALSE); collect (=FALSE) is an indicator for the frequency of garbage collection (change to TRUE if you have trouble with memory, but be aware that the execution time will increase)

arglist is to specify arguments from previous Amelia run to use in this run (should be an object of class ameliaArgs), for example (using one of the examples from slide 15):

```
> along.out1_1<-amelia(dtLong,arglist=along.out1_nt$arguments, emburn=c(50,100))
```

The above code creates 5 imputations with the same variable transformations as on slide 15 and new values for 'emburn'.

Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, arglist = NULL, empri = NULL, priors = NULL, autopri = 0.05, emburn = c(0,0),...)

The next three options are for setting 'priors' which provide a way to either incorporate additional information (that would help get a better estimates for the missing values – by using priors) or a way to help the EM algorithm start at a better value (by setting empri) or move in a better direction (by using autopri).

empri and autopri work similarly to the parameter k in ridge regression (both get multiplied by N); both should be between 0 and 1 (for empri it's suggested that is <0.1)

Where priors is used for specifying prior distribution for a particular column or observation in the form of a four or five column matrix.

Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, arglist = NULL, empri = NULL, priors = NULL, autopri = 0.05, emburn = c(0,0), bounds = NULL, max.resample = 100, overimp = NULL, ...)

emburn is the way to specify how long the EM chains should be by giving a minimum and maximum length (should be a vector of two numbers - c(min, max)).

>along.out1_<-amelia(dtLong, m=2, p2s=1, idvars=c("idtype"), ts="year", cs="caseid", noms=c("urban_rural", “region", "anychild", "r_hashins", "povstatus"), ords=c("edu", "alcohol", "famsize") , emburn=c(50,100))

Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, arglist = NULL, empri = NULL, priors = NULL, autopri = 0.05, emburn = c(0,0), bounds = NULL, max.resample = 100, overimp = NULL, ...)

Another way to incorporate outside information for range of the values of a particular variable/column is to set the bounds parameter by specifying a three-column matrix with logical bounds on the imputations - (col.num, lo.bound,hi.bound).

And you could set the max.resample parameter to how many draws the algorithm should draw (in the E-step) in meeting the constraints set in bounds (after that number the imputation value will be set to the bounds).

Syntax

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, arglist = NULL, empri = NULL, priors = NULL, autopri = 0.05, emburn = c(0,0), bounds = NULL, max.resample = 100, overimp = NULL, boot.type = "ordinary", parallel = c("no", "multicore", "snow"), ncpus = getOption("amelia.ncpus", 1L), cl = NULL, ...)

overimp is a way to specify observed values to be treated as missing by giving “the address” for each in a two-column matrix - c(row, col). The package has a function “overimpute” which does this for each observed value (see slide 28?).

boot.type could be set to either “ordinary”(default) or “none” (to run the EM algorithm without bootstrapping) .

Syntax – running in parallel

amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL, ts = NULL, cs = NULL, polytime = NULL, splinetime = NULL, intercs = FALSE, lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001, logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL, incheck = TRUE, collect = FALSE, arglist = NULL, empri = NULL, priors = NULL, autopri = 0.05, emburn = c(0,0), bounds = NULL, max.resample = 100, overimp = NULL, boot.type = "ordinary", parallel = c("no", "multicore", "snow"), ncpus = getOption("amelia.ncpus", 1L), cl = NULL, ...)

The next set of parameters (parallel, ncpus and cl) are for running Amelia in parallel on different CPUs, or processes. Multiple imputations is an "embarrassingly parallel" situation where the same code is run multiple times. By default Amelia does not run in parallel, but if needed one could set it to either "multicore" (for non-Windows machines) or to "snow" (for Windows machines). ncpus would have to be set in both situations.

cl is for specifying the type of cluster to be used in the case of snow" parallelization.

Additional Functions

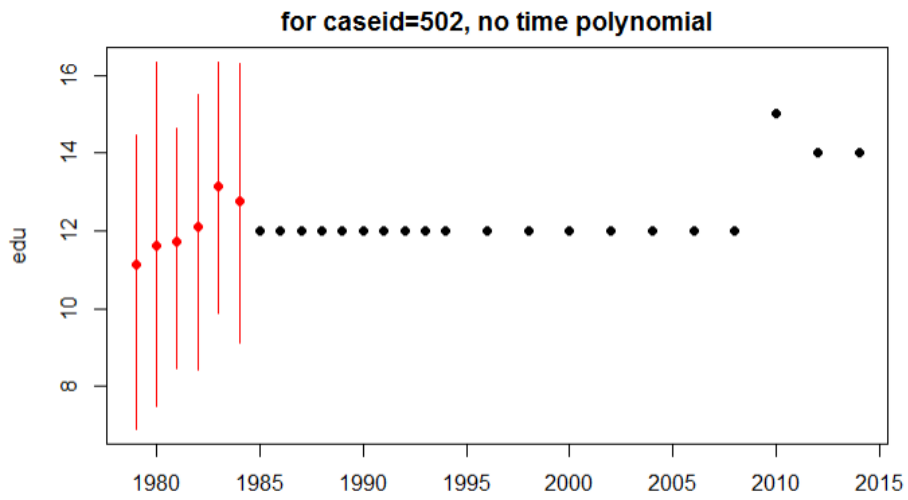
To check visually the distribution of imputations for one unit, you could use the 'tscsPlot' function:

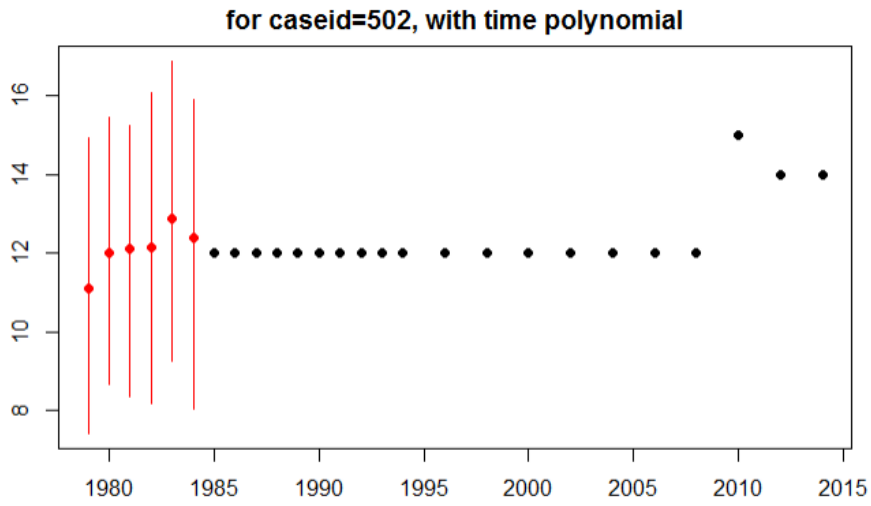
- tscsPlot(*a.out*, cs="*unit1*", var="*V1*", main="title")

Example:

>tscsPlot(along.out1_nt,cs=502,var="edu", main="for caseid=502, no time polynomial")

>tscsPlot(along.out1_t,cs=502,var="edu", main="for caseid=502, with time polynomial")





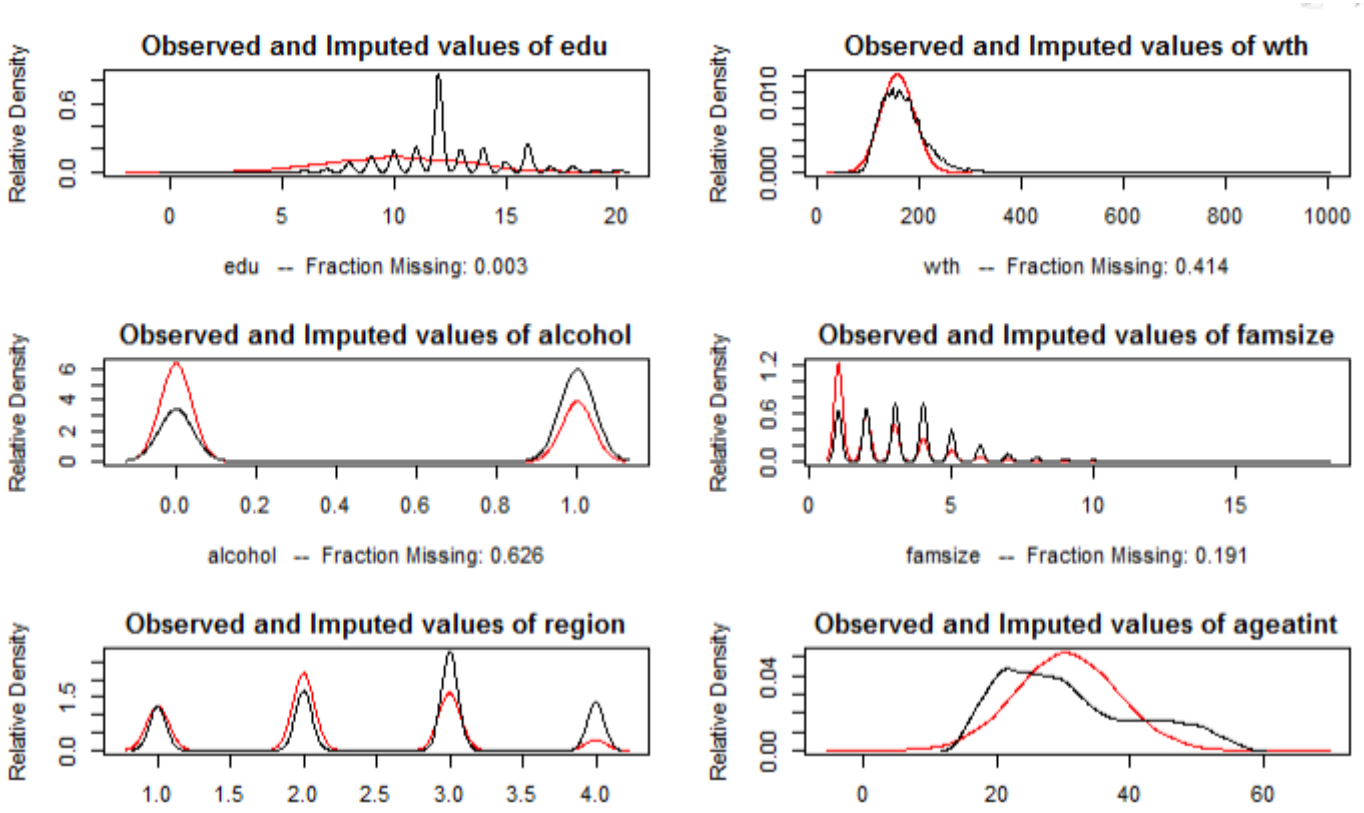
Additional Functions

To plot the densities of several variables you could use the plot function:

- plot(*a.out*, which.vars=*4:8*)

Here is an example:

>plot(along, which.vars=7:12)



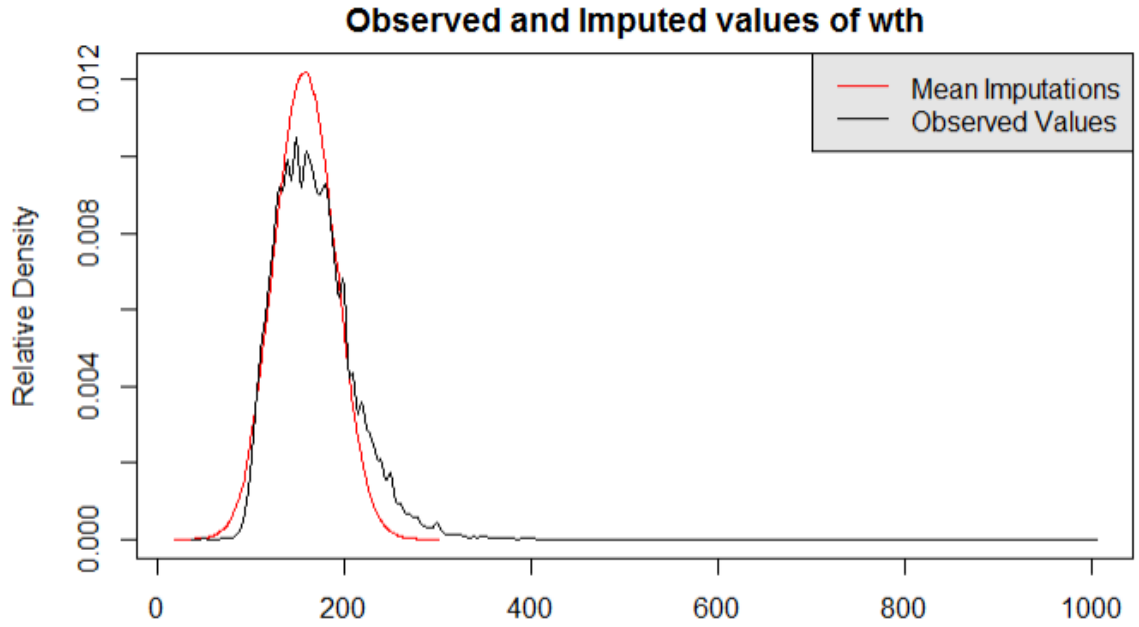
Additional Functions

To look at the observed and imputed values density for one variable/column, use:

- compare.density(*a.out*, var="*V1*")

Example:

>compare.density(along.out2, var="wth")



If the two densities are not close, you might consider changing the model for that variable.

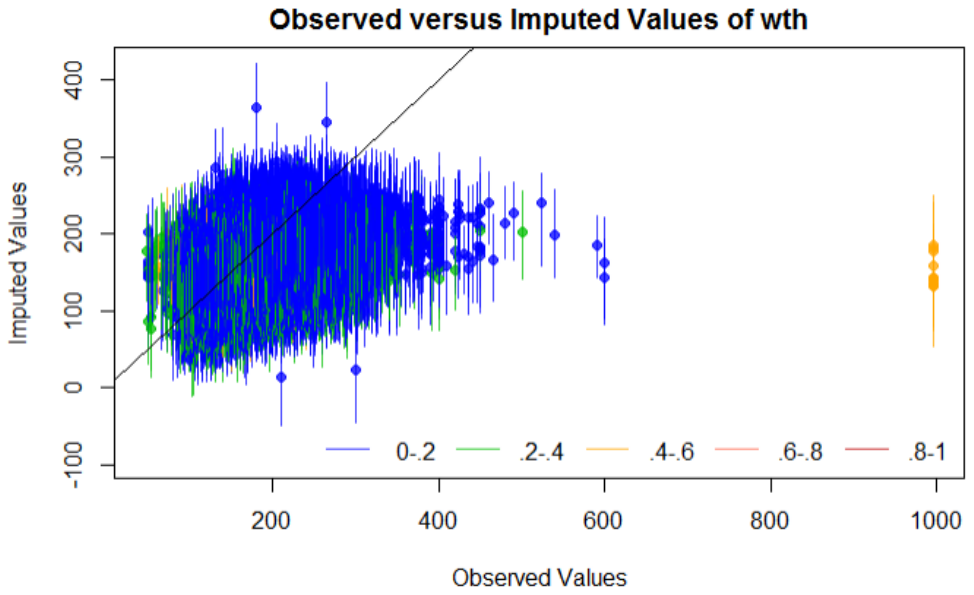
Additional Functions

Another diagnostic tool available to use is overimputation (treating each observed value as though it's missing, generating imputed value for it and then plotting the results). This is done by the function overimpute:

- overimpute(*a.out*, var="*V1*")

An example:

>overimpute(along.out2, var="wth")



The color of the line (see the legend for codes) represents the fraction of missing observations.

Note: This usually take a bit of time.

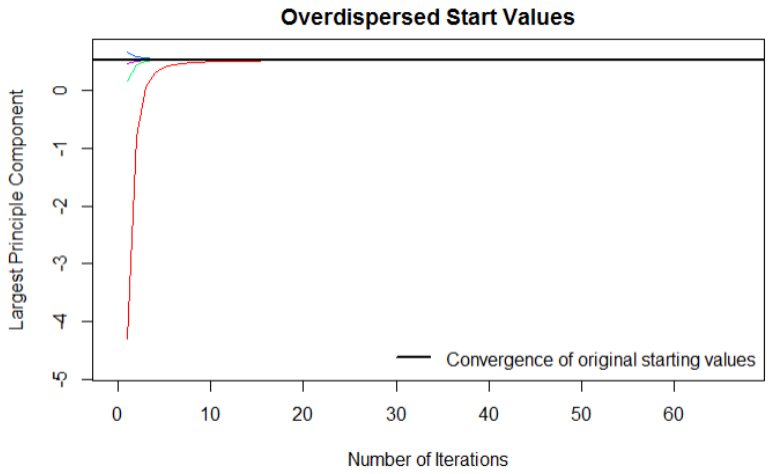
Additional Functions

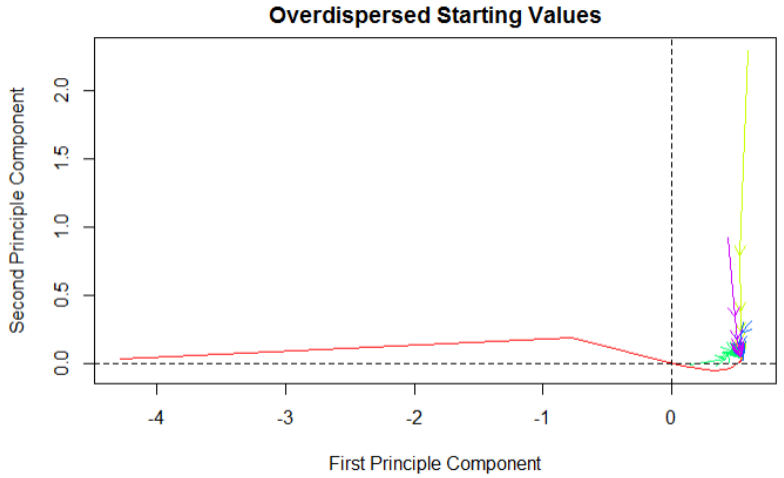
Another diagnostic check - for how well the EM algorithm did, in particular, whether it found the global maximum of the likelihood function, is the 'disperse' function:

- disperse(*a.out*, dmis=*1*, m=*5*)

This function produces two kinds of plots depending on the value of dmis (1 or 2):

- disperse(along.out2, dims=1, m=5)
- >disperse(along.out2, dims=2, m=5)





Both show convergence for 5 different starting values for EM chains (because m=5). The left plot shows only the first PC (principal component), where the plot on the right shows the first two principal components (this is controlled by the dmis parameter).

Combining Amelia runs

ameliabind(*a.out1, a.out2*)

where *a.out1, a.out2* are amelia objects

If you have run Amelia twice on the same data with the same parameter settings, you could combine the two amelia objects with 'ameliabind'.

For example:

>along.out1<-amelia(dtLong, m=3, p2s=1, idvars=c("idtype"), ts="year", cs="caseid", noms=c("urban_rural","region","anychild","r_hashins","povstatus"), ords=c("edu","alcohol","famsize"))

>along.out2<-amelia(dtLong,m=2, p2s=2, idvars=c("idtype"), ts="year", cs="caseid", noms=c("urban_rural","region","anychild","r_hashins","povstatus"), ords=c("edu","alcohol","famsize"))

>along<-ameliabind(along.out1,along.out2)

```
> along

Amelia output with 5 imputed datasets.
Return code:  1
Message:  Normal EM convergence

Chain Lengths:
--------------
Imputation 1:  61
Imputation 2:  47
Imputation 3:  70
Imputation 4:  75
Imputation 5:  71
```

```
> along.out1

Amelia output with 3 imputed datasets.
Return code:  1
Message:  Normal EM convergence.

Chain Lengths:
--------------
Imputation 1:  61
Imputation 2:  47
Imputation 3:  70
```

```
> along.out2

Amelia output with 2 imputed datasets.
Return code:  1
Message:  Normal EM convergence.

Chain Lengths:
--------------
Imputation 1:  75
Imputation 2:  71
```

Saving the imputations

write.amelia(obj=*a.out* , separate = TRUE, file.stem= *fname*, format="dta", impvar = "imp", orig.data = TRUE, ...)

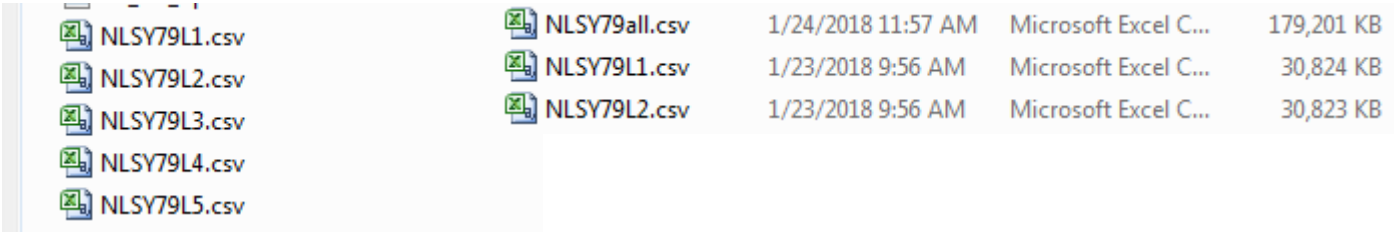
To save the imputed datasets use the 'write.amelia' function. A format could be specified for the files, so they could be read easily in another program. The available formats are: "csv", "dta", "table" . By default separate files are written, if you want all in one – set separate=FALSE.

Example:

```
> write.amelia(obj=along,file.stem="NLSY79L")
```

```
>write.amelia(obj=along,separate=FALSE,
file.stem="NLSY79all")
```

And in my folder:



References

- **Inference and missing data**, D. Rubin 1976 - http://qwone.com/~jason/trg/papers/rubin-missing-76.pdf
- **Statistical Analysis with Missing Data,** Little & Rubin, 2002 - http://onlinelibrary.wiley.com/book/10.1002/9781119013563

- **Analyzing Incomplete Political Science Data: An Alternative Algorithm for Multiple Imputation** , G. King, J. Honaker, A. Joseph, k. Scheve, American Political Science Review, 2001 - https://gking.harvard.edu/files/gking/files/evil.pdf
- **A program for missing Data**, J. Honaker, G. King, M. Blackwell (Journal of Statistical Software, 2011 - https://gking.harvard.edu/files/gking/files/amelia_jss.pdf

- EM algorithm: **Maximum Likelihood from Incomplete Data via the EM Algorithm**, Dempster, Laird, Rubin, Stat. Soc. Ser. B 39, 1–38 (1977) - https://www.cse.iitb.ac.in/~pjyothi/cs753/EM.pdf

Thank you.

Other packages in R for multiple imputations…

- **MICE** ("Multivariate Imputation via Chained Equations") - uses chained equations starting with the least missing
- **missForest** (an implementation of randomForest) – non-parametric algorithm
- **Hmisc** – this package has two main functions for imputation – 'impute' and 'aregImpute', later one of which uses predictive mean matching
- **mi** ("multiple imputation with diagnostics") – imputes from the conditional distribution of a variable given the observed and imputed values of the other variables; automatically detects irregularity in the data like high collinearity of variables