Software Manual Version 1.0.8

March 2, 2017

(Requires R 3.1.2 or higher)

Table of Contents *Page*

In publications, please cite as:

Jackson JW. Diagnostics for confounding of time-varying and other joint exposures. *Epidemiology* 2016. 27(6):859-869

Authored by: John W. Jackson

This software relies on R packages written by others. For their descriptions and license information, see:

tidyr	*https://cran.r-project.org/web/packages/tidyr/index.html*
magrittr	*https://cran.r-project.org/web/packages/magrittr/index.html*
ggplot2	*https://cran.r-project.org/web/packages/ggplot2/index.html*
gridExtra	*https://cran.r-project.org/web/packages/gridExtra/index.html*
scales	*https://cran.r-project.org/web/packages/scales/index.html*
Rmpfr	*https://cran.r-project.org/web/packages/Rmpfr/index.html*
data.table	*https://cran.r-project.org/web/packages/data.table/*
broom	*https://cran.r-project.org/web/packages/broom/index.html*

Overview

This software implements three diagnostics for confounding that can be used in sequence (see Jackson 2016). These apply to any study of multivariate exposures e.g. time-varying exposures, direct effects, interaction, and censoring. The first two diagnostics pertain to the nature of confounding in the data, while the third is meant to examine residual confounding after applying inverse probability weighting. The third diagnostic can also be used to examine residual confounding within propensity-score strata (when these are used in place of covariates in the parametric g-formula). We recommend that the diagnostics be applied to covariates that investigators will use to control for confounding. We provide tools to help focus each diagnostic on the relevant covariate history (used to control for confounding).

- Diagnostic 1 is a generalization of a "Table 1" for multivariate exposures (i.e. multiple exposures that are distinct in timing or character). It examines whether the prior covariate means are the same across exposure groups, among persons who have followed a particular exposure trajectory up to that point in time. Like a "Table 1" it is meant to describe whether exposure groups have different distributions of *prior* covariates.

- Diagnostic 2 is meant to inform whether or not g-methods are necessary to control for confounding. G-methods are required if any covariate measurement is associated with a prior exposure after adjusting for covariates that precede the exposure. Here, the diagnostic describes whether the covariate mean differs across prior exposure groups, after adjustment for covariates (that precede exposure) through inverse probability weighting or propensity score stratification.

- Diagnostic 3 is meant to be applied after investigators have applied the earlier diagnostics and have chosen to use g-methods. The form of Diagnostic 3 is similar to that of Diagnostic 1 in that it is a generalized "Table 1." The difference is that it is applied to an inverse probability weighted population, where the weights are typically designed to remove confounding. It can also be applied to evaluate residual confounding within levels of time-varying propensity-score strata.

The R-based functions presented here allow users to construct balance tables and trellised plots. They can be used to diagnose multivariate exposures that are binary or categorical. The functions can also accommodate right-censoring (including the extension for Diagnostic 2 described in Jackson 2016). While the R-based functions could in theory be applied to diagnose continuous exposures, it may be better to use a regression model for this (see sample R code at the end of the example).

The user can request time-specific tables and plots for all times, or a subset of selected times. One can alternatively request summary metrics that average over non-referent exposure values, propensity-score strata and/or exposure history, time, and segments of distance, in that order (see Jackson 2016).

To use these functions, users will need a basic understanding of data structures and data manipulation in R. They will also need to install a few add-on packages that the functions depend on (magrittr, tidyr, dplyr, ggplot2, scales, and gridExtra). These packages will need to be loaded at the beginning the R session, along with the grid package that comes with base R (see example at end of document). Note that the software will not run on R versions below 3.1.2.

Users should be aware that applying these functions to dataframes (datasets) with many observations, follow-up times, and covariates may require substantial computing memory. The approach we implement here is described in Jackson 2016 and could be adapted for other software. In addition, output from other software can be imported and used in the plotting function provided here.

Format of Input Data

The software is designed to use data that analysts may readily have on hand in analyses of time-varying exposures. This includes time-varying exposures, time-varying covariates, time-varying weights (when applicable), and time-varying propensity-score strata (when applicable). Time-varying exposure history strata may also be required, and we provide functions to create these from the exposure variables.

These time-indexed variables should be organized into a "wide" data format where each row uniquely indexes a single subject's data, so that columns index measurement of each variable at each time. The indices should be indicated with an underscore suffix followed by the time, e.g. "variable_1", "variable_2", "variable_3." No other underscores should appear in the variable name. It is fine if variables were measured at different times (e.g. "varA_1", "varB_2" "varB_4 "varC_3" "varC_5). As explained in the next section, this does not require additional arguments in the function that restructures the data.

The analysis proceeds as follows:

In a preliminary step, if exposure history is needed, create time-indexed exposure history variables from the time-indexed exposure variables: `makehistory.one()` or `makehistory.two()` functions.

1. Restructure this wide dataframe into a "tidy" long dataframe: `lengthen()` function

   By tidy, we mean that a row is uniquely identified by the pairing of exposure and covariate measurement times. This is will typically result in an extremely long dataframe, and could require substantial computing memory in R if there are many persons, covariates, and/or follow-up times. See Note (ii) for a solution to avoid memory issues with large and rich dataframes.

2. From the long dataframe, create a covariate balance table: `balance()` function

3. Plot the data in the covariate balance table: `makeplot()` function

Note (i) These steps should be followed in order as `lengthen()` produces the dataframe required by `balance()` which produces the dataframe used by `makeplot()`.

Note (ii) When the data have a large number of observations, covariates, and/or measurement times, memory issues can be ameliorated by using `diagnose()` to iteratively apply `lengthen()` and `balance()` by looping over covariates and additionally measurement times if desired.

Note (iii) One can encode assumptions about which covariates are necessary to adjust for confounding. This is done by removing covariate history that does *not* support exchangeability assumptions, through applying the `omit.history()` function to the "tidy" dataframe produced by `lengthen()`, `balance()`, or `diagnose()`.

Note (iv) Jackson 2016 proposes two methods for obtaining summary averages over person-time. The code here implements the standardization approach. However, regression models can be applied to the "tidy" dataframe produced by `lengthen()`. See the end of the software manual for sample code.

Note (v) The code can accept a vector of time-indexed censoring indicators (1=censored, 0=otherwise). This can be used regardless of the source of censoring (i.e. an event or some artificial rule defined by the investigator).

Note (vi) Although the code will still run in the presence of missing data, the results may not be easily interpretable (especially when there is missing data in the exposures and their history).

Function calls

`lengthen()` returns a dataframe where each record is indexed by the observation identifier, exposure measurement time, exposure value, covariate name, covariate measurement time, and possibly exposure history and/or propensity score strata. Weights will appear as additional columns.

```
lengthen(
       input               = dataframe in wide format,
       diagnostic          = diagnostic of interest e.g. 1, 2, or 3,
       censoring           = use censoring indicators/weights e.g. "yes" or "no",
       id                  = unique observation identifier e.g. "id"
       times.exposure      = a vector of exposure measurement times e.g. c(0,1,2)
       times.covariate      = a vector of covariate measurement times e.g. c(0,1,2)
       exposure            = the root name for exposure measurements e.g. "a",
       temporal. covariate = a vector of root names for covariates whose values change over time
                             e.g. c("l","m","n","o","p"),
       static.covariate    = a vector of root names for covariates whose values do not change
                            (covariates listed here should not appear in the temporal.covariate
                             argument)
       history             = the root name for history measurements e.g. "h",
       weight.exposure     = … for exposure weights e.g. "wa",
       censor              = … for censoring indicators "s",
       weight.censor       = … for censoring weights e.g. "ws",
       strata              = … for propensity-score strata e.g. "e",
       )
```

This function is designed to minimize user input by creating the covariate names automatically based on the *covariate* and *times* arguments. In the example above, it would create a lengthened dataframe based on the following covariate measurements:

```
c( "l_0","m_0","n_0","o_0","p_0",
   "l_1","m_1","n_1","o_1","p_1",
   "l_2","m_2","n_2","o_2","p_2")
```

Now, it may be that variable "n" is really a static covariate, like sex. In this case, you would only have "n_0" in the dataset, not "n_1" and "n_2". To specify a covariate like this, omit it from the *temporal.covariate* and instead include it in the *static.covariate* argument. Note that the software assumes that static covariates appear in the data with the lowest possible index specified in times. In this example, specifying "n" as a static covariate would create a lengthened dataframe based on the following covariate measurements (notice that "n_0" is included but "n_1" and "n_2" are not):

```
c("n_0",
   "l_0","m_0","o_0","p_0",
   "l_1","m_1","o_1","p_1",
   "l_2","m_2","o_2","p_2")
```

Note that the software will automatically detect and ignore covariate measurements that are not present within the input dataframe. For example, suppose variables "l*"* and "m*"* were only measured at times 0 and 2, and that l_1 and m_1 were not present in the dataframe. The software would, after detecting their absence, would create automatically lengthened dataframe based on the following covariate measurements i.e. "l_1","m_1" are excluded:

```
c( "l_0","m_0","n_0","o_0","p_0",
                "n_1","o_1","p_1",
   "l_2","m_2","n_2","o_2","p_2")
```

`balance()` takes the restructured dataframe output by `lengthen()` and returns a covariate balance table (possibly stratified by exposure history and/or propensity-score strata).

```
balance (
       input            = restructured dataframe
       diagnostic       = diagnostic of interest e.g. 1, 2, or 3,
       approach         = adjustment method e.g. "none" or "weight" or "stratify",
       censoring        = use censoring indicators/weights e.g. "yes" or "no",
       scope            = report the entire trellis e.g. "all", the diagonal e.g. "recent", or
                          a summary e.g. "average",
       times.exposure   = vector of exposure measurement times e.g. c(0,1,2),
       times.covariate  = vector of covariate measurement times e.g. c(0,1,2),
       sort.order       = vector of root names for all covariates listed in the order in which
                           they should appear in the table (and also plot) e.g.
                           c("n","m","o","l","p"). To display covariates in alphabetical order
                           (the default), leave blank or type "alphabetical"
       exposure         = root name of exposure e.g. "a",
       history          = …exposure history e.g. "h",
       weight.exposure  = …IP exposure weights e.g. "wa",
       weight.censor    = …IP censoring weights e.g. "ws",
       strata           = …of propensity score strata e.g. "e",
       recency          = an integer for the relative distance between exposures and covariate
                          measurements to focus on (e.g. 0 would represent the same timing).
                          the default is 0 for Diagnostics 1 and 3, and 1 for Diagnostic 2,
       average.over     = summary level for average metrics e.g. standardize over
                          "values" or "history" or "time" or "distance",
       periods          = a list of contiguous segments of relative distance to pool over
                          e.g. list(0,1:4,5:10) would yield summaries for three segments,
       list.distance    = a vector of distances to retain after averaging over time e.g. c(0,2),
       ignore.missing.metric = "yes" or "no" depending on whether the user wishes to estimate
                           averages over person-time when there are missing values of the mean
                           difference or standardized mean difference. Missing values for the
                           standardized mean difference can occur when, for example, there is no
                           covariate variation within levels of exposure-history and measurement
                           times. If this argument is set to "no" and there are missing values,
                           the average will also be missing. If set to "yes" an average will be
                           produced that ignores missing values.
        loop            = a housekeeping argument the user can ignore. It is automatically set
                          when the balance function is called by the diagnose() function
                          described later. The default is set to "no".
       )
```

`makeplot()` takes the covariate balance table produced by `balance()` and returns a trellised plot described in Jackson 2016.

```
makeplot (
       input            = output from balance()
       diagnostic       = the diagnostic of interest e.g. 1, 2, or 3,
       approach         = the adjustment method e.g. "none" or "weight" or "stratify,
       metric           = scale e.g. "D" for mean difference, "SMD" for standardized mean
                          difference
       censoring        = use censoring indicators/weights e.g. "yes" or "no",
       scope            = report the entire trellis e.g. "all", the diagonal e.g. "recent", or
                          a summary e.g. "average",
       stratum          = the propensity-score stratum to plot
       average.over     = level of summary for average e.g. "values" or "history" or "time"
                          or "distance"
                        … additional arguments to control plot formatting parameters, see
                          example
       )
```

`makehistory.one()` will create a set of exposure history variables for a time-varying exposure. `makehistory.two()` will create a set of joint exposure history variables for each of the two time-varying exposures, properly accounting for their temporal ordering (i.e. exposure "a" precedes exposure "b" at any time t). The new history variables will use the time-indices in the exposure vectors you supply. See additional notes at end of software manual for more details.

```
makehistory.one(
       id            = unique observation identifier e.g. "id",
       input         = dataset in wide format,
       times         = a vector of measurement times e.g. c(0,1,2)
       exposure      = the root name for exposure "a",
       name.history  = desired root name for time-indexed history variables e.g. "h",
       group         = an optional baseline variable upon which to segregate the exposure
                       history. This argument provides a way to adjust the metrics for a baseline
                       covariate. For example, in the context of a trial, the grouping variable
                       could be treatment assignment. In the context of a cohort study, this
                       could be site e.g. "v".
       )

makehistory.two(
       id            = unique observation identifier e.g. "id",
       input         = dataframe in wide format,
       times         = a vector of measurement times e.g. c(0,1,2)
       exposure.a    = the root name for the first exposure e.g. "a",
       exposure.b    = the root name for the second exposure e.g."z",
       name.history.a =desired root name for the first time-indexed history variables e.g. "ha",
       name.history.b = … root name for the second time-indexed history variables e.g. "hb",
       group         = an optional baseline variable upon which to segregate the exposure
                       history. This argument provides a way to adjust the metrics for a baseline
                       covariate. For example, in the context of a trial, the grouping variable
                       could be treatment assignment. In the context of a cohort study, this
                       could be site e.g. "v".
      )
```

`omit.history()` will take the dataframe produced by `lengthen()` and remove covariate measurements based on their fixed measurement time or relative distance from exposure measurements (at time t) i.e. ones that do **not** support exchangeability assumptions at time t. The *covariate.name* argument is used to name the covariate whose history you wish to modify. To process the same manipulation for a set of covariates, simply supply a vector of covariate names to *covariate.name*. The *omission* argument determines whether the covariate history is (i) set to missing for certain covariate measurement times (*omission* ="fixed" with *times*=a vector of integers) or (ii) set to missing only for covariate measurement times at or before a certain distance k from exposure measurement times (*omission* ="relative" with *distance*=some integer) or (iii) set to missing only for covariate measurements that share the same timing as exposure measurements (*omission* ="same.time"). The removed values are set to missing. For example, using the "fixed" omission option for covariate "l" at time 2 will set all data on "l" at time 2 to missing, regardless of the exposure measurement time. In contrast, using the "relative" omission option for covariate "l" with distance 2 will only set to missing data on "l" that is measured two units or more before the exposure measurement time (i.e. $t-2$, $t-3$, $t-4$ and so on). Last, using the "same.time" omission option for covariate "l" will set to missing all data on "l" that is measured at the same time as the exposure. Missing data will be ignored when this dataframe is supplied to the `balance()` function. They will not contribute to the resulting covariate balance table, nor to plots produced by `makeplot()`, nor will they contribute to any summary metrics are estimated by averaging over person-time.

```
omit.history(
     input          = restructured dataframe from lengthen() ,
     omission       = type of omission e.g. "fixed" or "relative" or "same.time"
     covariate.name = root name of the covariate e.g. "m",
     distance       = the distance between exposure and covariate measurements e.g. 2
```

```
    times           = a vector of measurement times for the covariate e.g. c(1,2,3)
    )
```

`diagnose()` is a wrapper function that calls the `lengthen()` and `balance()` functions in sequence, either in one step or iteratively across subsets of covariates and measurement times. In both cases it outputs a dataset that is suitable for plotting via the `makeplot()` function. When the user opts to not iterate over covariates and measurement times, there is no difference between calling the `lengthen()` and `balance()` functions one after the other. However, opting to iterate can be a useful way to process large and rich dataframes without otherwise requesting as much memory.

```
diagnose(
       input            = dataframe in wide format,
       diagnostic       = diagnostic of interest e.g. 1, 2, or 3,
       censoring        = use censoring indicators/weights e.g. "yes" or "no",
       approach         = adjustment method e.g. "none" or "weight" or "stratify",
       scope            = report the entire trellis e.g. "all", the diagonal e.g. "recent", or
                           a summary e.g. "average",
       id               = unique observation identifier e.g. "id"
       times.exposure   = a vector of exposure measurement times e.g. c(0,1,2)
       times.covariate    = a vector of covariate measurement times e.g. c(0,1,2)
       exposure         = the root name for exposure measurements e.g. "a",
       temporal. covariate= a vector of root names for covariates whose values change over time
                            e.g. c("l","m","n","o","p"),
       static.covariate = a vector of root names for covariates whose values do not change
                           (covariates listed here should not appear in the temporal.covariate
                            argument)
       sort.order       = vector of root names for all covariates listed in the order in which
                           they should appear in the table (and also plot) e.g.
                           c("n","m","o","l","p"). To display covariates in alphabetical order
                           (the default), leave blank or type "alphabetical"
       history          = the root name for history measurements e.g. "h",
       weight.exposure  = … for exposure weights e.g. "wa",
       censor           = … for censoring indicators "s",
       weight.censor    = … for censoring weights e.g. "ws",
       strata           = … for propensity-score strata e.g. "e",
       recency          = an integer for the relative distance between exposures and covariate
                          measurements to focus on (e.g. 0 would represent the same timing).
                          the default is 0 for Diagnostics 1 and 3, and 1 for Diagnostic 2,
       average.over     = summary level for average metrics e.g. standardize over
                          "values" or "history" or "time" or "distance",
       periods          = a list of contiguous segments of relative distance to pool over
                          e.g. list(0,1:4,5:10) would yield summaries for three segments,
       list.distance   =  a vector of distances to retain after averaging over time e.g. c(0,2),
       ignore.missing.metric = "yes" or "no" for whether the user wishes to estimate
                           averages over person-time when the balance metric has missing values.
                           For example, the standardized mean difference will be missing when
                           there is no covariate variation within levels of exposure-history and
                           measurement times. When this argument is set to "no" and there are
                           missing values, the average will also be missing. If set to "yes" an
                           average will be produced that ignores missing values.
       metric           = the metric for which the user wishes to ignore missing values as
                           specified in the 'ignore.missing.metric' argument.
        loop            = "yes" to iteratively apply balance() and lengthen() or "no" to process
                           all covariates and measurement times at once.
        loop.type       = the granularity of the loop to implement e.g. 1, 2, or 3. Specifying
                         '1' will apply balance() and lengthen() one covariate at a time,
                         processing all covariate and measurement times together. Specifying '2'
                         will loop over covariates and also prior exposure measurement times (for
                         Diagnostics 1 and 3) or covariates and also prior exposure measurement
                         times (for Diagnostic 2). Specifying '3' will, for each covariate, loop
                         over the appropriate pairing of exposure and covariate measurement times
                         for the specified Diagnostic. Generally, nested iterations as '2' and '3'
                         require less memory but could take longer to run.
       )
```

There is an important change in workflow when users wish to use the `diagnose()` function and also remove irrelevant covariate history from the balance table calculations and plot. Users who wish to do so will need to apply the `omit.history()` function to the dataframe output by `diagnose()`. If the user wants to remove covariate history while averaging metrics over time or distance, the user will first have to call the `diagnose()` function with the *scope* argument set to "all". The user would then apply the `omit.history()` function as many times as desired, and then use the `apply.scope()` function described next to average the metrics over person-time. This workflow change allows users to ensure that the summary metrics ignore covariate history the user deems irrelevant to confounding.

`apply.scope()` is a helper function that will take a dataframe output by `balance()` or `diagnose()`, where the *scope* argument in those functions was set to "all", and subset the table to covariate balance metrics at a certain distance (e.g. a certain recency) or produce estimates that average over person-time. This function is only useful when a user wishes to focus on proximal covariate balance metrics or produce summary estimates via `diagnose()`, but also needs to remove covariate history that is irrelevant to confounding. In this situation, the user first applies the `diagnose()` function with the *scope* argument set to "all", then applies the `omit.history()` function, followed by the `apply.scope()` function.

```
apply.scope(input = dataframe output by diagnose() or balance() function,
            diagnostic  = diagnostic of interest e.g. 1, 2, or 3,
            approach    = adjustment method e.g. "none" or "weight" or "stratify",
            scope       = report the entire trellis e.g. "all", the diagonal e.g. "recent", or
                           a summary e.g. "average",
            recency     = an integer for the relative distance between exposures and covariate
                          measurements to focus on (e.g. 0 would represent the same timing).
                          the default is 0 for Diagnostics 1 and 3, and 1 for Diagnostic 2,
            average.over = summary level for average metrics e.g. standardize over
                          "values" or "history" or "time" or "distance",
            periods      = a list of contiguous segments of relative distance to pool over
                          e.g. list(0,1:4,5:10) would yield summaries for three segments,
            list.distance =  a vector of distances to retain after averaging over time e.g.
                             c(0,2),
            sort.order  = vector of root names for all covariates listed in the order in which
                           they should appear in the table (and also plot) e.g.
                           c("n","m","o","l","p"). To display covariates in alphabetical order
                           (the default), leave blank or type "alphabetical"
            ignore.missing.metric = "yes" or "no" depending on whether the user wishes to
                           estimate averages over person-time when there are missing values of the
                           mean difference or standardized mean difference. Missing values for the
                           standardized mean difference can occur when, for example, there is no
                           covariate variation within levels of exposure-history and measurement
                           times. If this argument is set to "no" and there are missing values,
                           the average will also be missing. If set to "yes" an average will be
                           produced that ignores missing values.
            metric      = the metric for which the user wishes to ignore missing values as
                           specified in the 'ignore.missing.metric' argument.
                        )
```

Required Arguments for Function Calls

For all functions users must specify the *diagnostic*, *approach*, *scope*, and *censoring* arguments. Depending on how these are specified, other arguments may be required:

Required arguments for `lengthen()` by diagnostic, approach, and censoring arguments			
Diagnostic	***Approach***	***Censoring***	***Additional Required Arguments (in addition to Scope)***
1	"none"	"no"	id, exposure, temporal.covariate or static.covariate, times.exposure, times.covariate , history
		"yes"	(previous) + censor
2	"weight"	"no"	id, exposure, temporal.covariate or static.covariate, times.exposure, times.covariate, history, weight.exposure
		"yes"	(previous) + censor
	"stratify"	"no"	id, exposure, temporal.covariate or static.covariate, times.exposure, times.covariate, strata
		"yes"	(previous) + censor
3	"weight"	"no"	id, exposure, temporal.covariate or static.covariate, times.exposure, times.covariate, history, weight.exposure
		"yes"	(previous) + censor
	"stratify"	"no"	id, exposure, temporal.covariate or static.covariate, times.exposure, times.covariate, history, strata
		"yes"	(previous) + censor

Required arguments for `balance()` by diagnostic, approach, and censoring arguments			
Diagnostic	***Approach***	***Censoring***	***Additional Required Arguments (in addition to Scope)***
1	"none"	"no"	exposure, history, times.exposure, times.covariate
		"yes"	(previous)
2	"weight"	"no"	exposure, history, times.exposure, times.covariate, weight.exposure
		"yes"	(previous) + weight.censor
	"stratify"	"no"	exposure, times.exposure, times.covariate, strata,
		"yes"	(previous) + weight.censor
3	"weight"	"no"	exposure, history, times.exposure, times.covariate, weight.exposure
		"yes"	(previous) + weight.censor
	"stratify"	"no"	exposure, history, times.exposure, times.covariate, strata
		"yes"	(previous) + weight.censor

Required arguments for `makeplot()` by diagnostic and approach arguments		
Diagnostic	***Approach***	***Additional Required Arguments (in addition to Scope and Metric)***
1	"none"	---
2	"weight"	---
	"stratify"	Stratum
3	"weight"	---
	"stratify"	Stratum

Note (i) The `makeplot()` function also requires the *metric* argument.

Note (ii) For the `balance()` and `makeplot()` functions, specifying *scope*="average", will require you to specify an option for the *average.over* argument. If you chose *average.over*="strata" then you do not need to choose a value for the *stratum* argument.

Note (iii) Specifying *scope*="recent" will allows you to specify an option for the *recency* argument in `balance()` i.e. compute metrics at a specific exposure-covariate distance of your choosing.

Note (iv) `diagnose()` has the combined requirements of `lengthen()` and `balance()`.

Additional Notes

Format of initial dataset

- As stated earlier, the input dataset should have one record per observation (wide format) with the timing of variables indexed by an underscore followed by the time index (**underscores should NOT appear anywhere else in the variable name**). Any indexing scheme can be used (e.g. `"var_1","var_4","var_9")`, but it may be easiest to assign zero as the baseline index and increase it by one the unit for each subsequent measurement (e.g. `"var_0","var_1","var_2")`.

- The common referent value—to which all other exposure levels are compared—should be coded as the lowest value.

- Censored data should contain a vector of time-indexed censoring indicators (1=censored, 0 otherwise) for the `lengthen()` function.

Alignment of Censoring weights

- Generally speaking, the code asks for separate exposure and censoring weights. This is so because the `lengthen()` function will align censoring weights with exposure times, in the case of Diagnostics1 and 3, or with covariate times in the case of Diagnostic 2.

- The `balance()` function takes the product of exposure and censoring weights during the estimation process.

Time-indices for multivariate exposures and point exposures

- The functions provided here were developed for time-varying exposures and treat covariates as if they precede exposure when they share the same time index. What follows next is a workaround for multivariate exposures that generally applies when, for some times, exposures precede covariates.

  For multivariate joint exposures, some covariates L may intercede between the exposures, as in the example of exposures A and Z in the eAppendix of the Jackson 2016. Specifically, it may be the case that (i) at each time t, exposure $A(t)$ affects covaraites $C(t)$ which affect exposure $Z(t)$, and (ii) covariates $C(t)$ affect subsequent exposures $A(t+k)$ and $Z(t+k)$ and also the outcome Y. The functions could be used as they are to assess confounding for the second exposure Z (since both covariates L and C precede Z); a workaround to assess confounding for the first exposure A would be to increase, by one unit (or some value appropriate for the data's indexing scheme), the indices for all covariates L that occur after A for any given time t (i.e. covariate index → covariate index+1). The functions can then be used to examine confounding for exposure A. Another approach for exposure A would be to remove the history on L measured at the same time as the exposure A using `omit.history()` on the dataframe produced by `lengthen()`, but this only works for Diagnostics 1 and 3.

- The code can also be tricked to handle multivariate point exposures by simply adding a subscript "_0" to each exposure and covariate when using `lengthen()`, and then specifying "0" for exposure and covariate times when using `balance()`.

Multivariate time-varying exposures or point exposures

When the exposure is multivariate, the idea is to diagnose each exposure separately (see eAppendix of Jackson 2016). From the perspective of using the R-functions, the only difference is to use exposure history based on all exposures that comprise the multivariate exposure. It is important that such joint exposure history accurately reflect the ordering of each component exposure. The function `makehistory.two()` creates an appropriate joint exposure history for each of two exposures, assuming that exposures in its argument *list.exposure.a* (e.g. A) precede those in *list.exposure.b* (e.g. Z) at any given index as described in the eAppendix of Jackson 2016. In that example, exposure $A(t)$ always precedes exposure $Z(t)$ such that the joint history of $A(2)$ is $A(1), A(0), Z(0)$ while the joint history of $Z(2)$ is $A(1), A(0), Z(1), Z(0)$. If one exposure does not precede the other, investigators will still need to use an appropriate joint exposure history and can specify either order as desired. Note that the exposure history produced by the function `makehistory.two()` will be inappropriate if the relative ordering of $A(t)$ and $Z(t)$ varies over time.

Averaging over person-time

When using the `balance()` , `diagnose()`, or `apply.scope()` functions, specifying *average.over*="average" and *average.over*="time" will return balance metrics for each "distance" value. The output can be subset to specific distances of interest e.g. *k*=0 and *k*=2 by supplying a vector to *list.distance* e.g. c(0,2) but this is optional. Specifying *average.over*="distance", you can opt to average within segments of distance using the *periods* argument (leaving this blank will average over all distance values). The *periods* argument requires a list of contiguous numeric vectors e.g. list(0,1:4,5:10). For Diagnostic 3 this would report metrics at time t, averages over times $t-1$ to $t-4$, and averages over times $t-5$ to $t-10$. For Diagnostics 1 and 3 the entire range should lie between 0 and t. For Diagnostic 2 the entire range should lie between 1 and t.

Residual confounding for parametric g-formula w/ propensity score stratification

- Jackson 2016 emphasizes Diagnostic 3 to describe residual confounding in a weighted population (for marginal structural models). This can be accomplished by specifying *diagnostic*=3 and *approach*="weight". Any weight can be used.

- In the eAppendix of Jackson 2016, there is an alternative version of Diagnostic 3 that describes residual confounding within a propensity-score stratified population (for a special case of the parametric g-formula). This is done by specifying *diagnostic*=3 and *approach*="stratify". Note that one can average these metrics over propensity score strata, exposure history, time, and distance (i.e. by specifying *scope*= "average" and choosing *average.over*="strata" or higher).

Notes on investigator supplied data

These functions can diagnose confounding for a single exposure or each distinct exposure (in a multivariate exposure) as long as the user provides appropriate history, inverse probability weights, propensity score strata, and censoring indicators. See Jackson 2016 for details. Note that those particular specifications may not apply to the user's causal question (e.g. the user has data where covariates are measured after exposure for every time point, instead of before exposure). The `make.history()` functions return nonsense when exposures are partially missing.

A warning on required arguments

The *diagnostic*, *approach*, *scope*, and *censoring* arguments for the `lengthen()`, `balance()`, `diagnose()`, and `makeplot()` functions are required and must be identical or else the functions will return errors or incorrect results.

Example code

```
##########################
##LOAD DATA AND PACKAGES##
##########################

#NOTE THAT THIS CODE REQUIRES R VERSION 3.1.2 OR HIGHER

install.packages(c("magrittr","tidyr","dplyr","ggplot2","gridExtra","scales","broom"),dependencies=TRUE)
library(magrittr) #last tested on magrittr v1.5
library(tidyr)    #last tested on tidyr v0.6.0
library(dplyr)    #last tested on dplyr v0.5.0
library(ggplot2)  #last tested on ggplot2 v2.1.0
library(grid)     #this comes with base R
library(gridExtra)#last tested on gridExtra v2.0.1
library(scales)   #last tested on ggplot2 v0.4.0
library(Rmpfr)    #last tested on Rmpfr v0.6-0
library(broom)    #last tested on broom v0.4.1
path   <- "C:\\"
#for mac use one slash
indata.small <- read.csv(paste(path,"example_sml.csv",sep=""))
#indata.large <- read.csv(paste(path,"example_lrg.csv",sep=""))
source(paste(path,"RFunctions_1_0_8.r",sep=""))

#####################################################################
##Example: Diagnostic 3 for a time-varying exposure without censoring
#####################################################################

#PRELIMINARY STEP: MAKE EXPOSURE HISTORY
mydata <- indata.small
mydata.history <- makehistory.one(input=mydata,exposure="a",name.history="h",times=c(0,1,2))

#STEP 1: RESTRUCTURE THE DATA
mydata.long <- lengthen(
  input=mydata.history,
  diagnostic=3,
  censoring="no",
  id="id",
  times.exposure=c(0,1,2),
  times.covariate=c(0,1,2),
  exposure="a",
  temporal.covariate=c("l","m","o"),
  static.covariate=c("n","p"),
  history="h",
  weight.exposure="wax"
)

#example of how to remove relative covariate history
mydata.long.omit <- omit.history(input=mydata.long,
  omission="relative",
  covariate.name=c("l","m","o"),
  distance=1
  )

#STEP 2: CREATE BALANCE TABLE
mytable <- balance (
input=mydata.long.omit,
diagnostic=3,
approach="weight",
censoring="no",
scope="all",
times.exposure=c(0,1,2),
times.covariate=c(0,1,2),
exposure="a",
history="h",
weight.exposure="wax",
sort.order= c("l","m","o","n","p")
)

#STEP 3: PLOT BALANCE METRIC
myplot <- makeplot (
```

```
input=mytable,
diagnostic=3,
approach="weight",
scope="all",
metric="SMD"
)
#The following formatting arguments for makeplot() are optional (defaults shown).

#label.exposure="A",                          #exposure label
#label.covariate="C",                         #covariate label
#lbound=-1,                                   #lower bound for x-axis
#ubound=1,                                    #upper bound for x-axis
#ratio=2,                                     #plot aspect ratio
#text.axis.title=8,                           #title font size
#text.axis.y=6.5,                             #y-axis (covariate names) font size
#text.axis.x=6.5,                             #x-axis font size
#text.strip.y=10,                             #row panel label font size
#text.strip.x=10,                             #column panel label font size
#point.size=.75,                              #dot size
#zeroline.size=.1,                            #thickness of zero line on x-axis
#refline.size=.1,                             #thickness of reference line on x-axis
#refline.limit.a=-.25,                        #location for reference line 1 on x-axis
#refline.limit.b=0.25,                        #location for reference line 2 on x-axis
#panel.margin.size=.75,                       #space between panels
#axis.title="Mean Difference",                #or "Standardized Mean Difference" (x-axis title)
#label.width=15                               #width of panel label text (before wrapping text)

#STEP 4: SAVE BALANCE TABLE AND PLOT
#write.csv(mytable,paste(path,"mytable.csv",sep=""))
#ggsave(filename=paste(path,"myplot.pdf",sep=""))

################################################
##Example of Regression Approach for Diagnostic 1
################################################

library(broom) #need for tidy()

#create balance dataset
mydata.long <- lengthen(input=mydata,
  diagnostic=1,
  censoring="no",
  id="id",
  times.exposure=c(0,1,2),
  times.covariate=c(0,1,2),
  exposure="a",
  temporal.covariate=c("l","m","n","o","p"),
  history="h"
)

##MAKE BALANCE TABLE USING REGRESSION##

#create balance table
mydata.long.reg <- mutate(mydata.long,time=time.exposure,distance=time.exposure-time.covariate,history=h)
output <- mydata.long.reg %>%
  group_by(name.cov) %>% #note, you can include other stratifying variables here or in the model
    filter(time.exposure>=time.covariate) %>%
      do(tidy(lm(formula=value.cov~a+time+distance+history,.))) %>% #same model form used for every covariate
        filter(term=="a1") %>% ungroup()

table.reg <- output %>%
               select(name.cov,estimate) %>%
                 rename_("D"="estimate")

print(table.reg)
#write.csv(table.reg,paste(path,"table_regression.csv"))
#NOTE: This code applies the same model parameterization for each covariate (relying on a strong assumption).

### COMPARE THAT TO A DIRECT CALCULATION & STANDARDIZATION ###

table.std <- balance(input=mydata.long,
  diagnostic=1,
  approach="none",
  censoring="no",
  scope="average",
  average.over="distance",
  times.exposure=c(0,1,2),
  times.covariate=c(0,1,2),
  exposure="a",
  history="h"
)
```

```
print(table.std)
#write.csv(table.std,paste(path,"table_standardization.csv"))
```