Matching in R using the `optmatch` and `RItools` packages

Ben B. Hansen, Mark Fredrickson, Josh Buckner, Josh Errickson, and Peter Solenberger, with embedded Fortran code due to Dimitri P. Bertsekas and Paul Tseng

2019-07-12

The R Environment

All the software used in this worksheet is freely available. The R statistical package is installed for you in the lab, but you may download and install R for Windows, Mac, and Linux systems from: https://www.r-project.org.

The following document walks through a common propensity score matching work-flow in R. Example R code will appear with a `>` indicating the command prompt. You may type this code yourself — each line is a command to R. Output will follow prefaced by `##`. (In R, `#` represents a comment; any command preceded by any number of `#`'s will not be executed.) For example:

```
> 2 + 2
## [1] 4
```

(Note that when entering the code yourself, do not include the `>` in your command. Also, for longer lines of code in this document, the text may wrap onto a second line, with the second line preceded by a `+` sign. When entering the code yourself, you do not have to wrap the lines, and do not include the `+`.)

R stores data in named variables using the arrow operator:

```
> my.variable <- 2 + 2
> my.variable * 3
## [1] 12
```

Setup

Outfitting your R with the proper add-ons

R add-on packages are available to install directly from R:

```
> install.packages("optmatch")
> install.packages("RItools")
```

These commands will ask you to select a CRAN server. Any server will do. You may also be asked whether you'd like to set up a "personal library to install packages into"; if so, answer yes. (The default personal library location that R will suggest should be OK.) You'll only need to run these commands the first time you want to use `optmatch` or `RItools` on a particular computer, or when you install a new version of R.

Setting up the R environment for matching

Attach extension packages that we'll be using for matching and associated diagnostics:

```
> library(optmatch)
> library(RItools)
```

You'll do this each time you start a new R session and want to run matching commands.

1

To load the nuclear plants data, enter

```
> data(nuclearplants)
```

To see the first six rows:

```
> head(nuclearplants)
##      cost   date t1 t2  cap pr ne ct bw cum.n pt
## H 460.05 68.58 14 46  687  0  1  0  0    14  0
## I 452.99 67.33 10 73 1065  0  0  1  0     1  0
## A 443.22 67.33 10 85 1065  1  0  1  0     1  0
## J 652.32 68.00 11 67 1065  0  1  1  0    12  0
## B 642.23 68.00 11 78 1065  1  1  1  0    12  0
## K 345.39 67.92 13 51  514  0  1  1  0     3  0
```

For more on the variables here, enter

```
> help("nuclearplants")
```

You can directly access a variable within this data frame as follows. (Try typing in the commands to see what they do.)

```
> nuclearplants$pt
> table(nuclearplants$pt)
> with(nuclearplants, table(pt))
```

The variable you will have just viewed and tabulated, `pt`, is a dummy for whether the plant was built with "partial turnkey guarantees." These plants were not comparable to the others in terms of construction costs. Let's exclude them for the time being, for simplicity. To do this we'll create a data table (in R jargon, a "data frame") of just those observations for which `pt` is 0:

```
> nuke.nopt <- subset(nuclearplants, pt == 0)
```

To inspect its first six or last six entries, do

```
> head(nuke.nopt)
> tail(nuke.nopt)
```

To view this as presenting a matching problem, we'll think of plants built on the site of a previously existing plant (`pr == 1`) as the treatment group and plants on new sites (`pr == 0`) as comparisons.

Optimal pair matching and 1:k matching

Pair matching

To check the number of treated and control plants:

```
> table(nuke.nopt$pr)
##
##  0  1
## 19  7
```

To get the pair match minimizing the mean paired distance on `cap`, among all collections of 7 non-overlapping pairs, do

```
> pairmatch(pr ~ cap, data = nuke.nopt)
##    H    I    A    J    B    K    L    M    C    N    O    P    Q    R    S
## <NA>  1.2  1.1  1.1  1.2 <NA>  1.3 <NA>  1.3  1.7 <NA> <NA>  1.4  1.5 <NA>
```

```
##    T    U    D    V    E    W    F    X    G    Y    Z
## <NA>  1.6  1.4 <NA>  1.5 <NA>  1.6 <NA>  1.7 <NA> <NA>
```

For a more readable report of who gets matched to whom, type

```
> print(pairmatch(pr ~ cap, data = nuke.nopt), grouped = TRUE)
##  Group Members
##     1.1    A, J
##     1.2    I, B
##     1.3    L, C
##     1.4    Q, D
##     1.5    R, E
##     1.6    U, F
##     1.7    N, G
```

For matching on both `date` and `cap`, you'd type `pairmatch(pr ~ cap + date, ...)` instead of `pairmatch(pr ~ cap, ...)`. We'll talk later about how this combines discrepancies on the two variables. For now, note the form of the output this command generates: a variable of the same length as the variables making up `nuke.nopt`, assigning a distinct name to each matched set. To fix your intuition, you might try connecting up below the units that `pairmatch()` has placed in the same matched sets.

Table 1: New-site (left columns) versus existing-site (right columns) plants. "date" is `date-65`; "capacity" is `cap-400`.

Plant	Date	Capacity	Plant	Date	Capacity
A	2.3	660	H	3.6	290
B	3.0	660	I	2.3	660
C	3.4	420	J	3	660
D	3.4	130	K	2.9	110
E	3.9	650	L	3.2	420
F	5.9	430	M	3.4	60
G	5.1	420	N	3.3	390
			O	3.6	160
			P	3.8	390
			Q	3.4	130
			R	3.9	650
			S	3.9	450
			T	3.4	380
			U	4.5	440
			V	4.2	690
			W	3.8	510
			X	4.7	390
			Y	5.4	140
			Z	6.1	730

For basic summary information about this match, try

```
> summary(pairmatch(pr ~ cap, data = nuke.nopt))
```

If you've already typed in the `pairmatch(...)` part, you can use the up-arrow, Home and End keys to avoid having to re-type. Alternatively, to assign the name "pm" to the matching result, do

```
> pm <- pairmatch(pr ~ cap, data = nuke.nopt)
```

Now, you can just type `print(pm, grouped = TRUE)` or `summary(pm)`.

The following would give a basic matched analysis of the effect of new or existing site on construction costs is given with the help of *R*'s linear modeling function. In effect, the existing site effect is estimated as one "way" in a two-way ANOVA, the other "way" being the factor variable that represents the matching result, i.e. pm.

```
> summary(lm(cost ~ pr + pm, data = nuke.nopt))
```

Matching with multiple controls

There are other types of matches you might want to try. Here's how to create matched triples (each treatment group unit is matched to two control group units):

```
> tm <- pairmatch(pr ~ cap, controls = 2, data = nuke.nopt)
```

There will be further variations suggested on the slides.

Did matching work?

It's possible to give the software an impossible list of requirements for a match. For instance, try running the following:

```
> pairmatch(pr ~ cap, controls = 3, data=nuke.nopt)
```

The problem here is that the data don't have 3 comparison units to go with each treatment unit, since we have 7 treatment units but only 19 comparison units.

Matching can also fail because the distance matrix embodies matching constraints that are impossible to meet. In these cases the matching function will generally run without complaint, although it won't create any matches. Here is an example, where the caliper is so narrow as to forbid all possible matches:

```
> pairmatch(pr ~ cap + cost, caliper=.001, data = nuke.nopt)
## Warning in fullmatch.InfinitySparseMatrix(x = x, min.controls = controls, : Matching failed. (Restri
##   Enter ?matchfailed for more info.
##    H    I    A    J    B    K    L    M    C    N    O    P    Q    R    S
## <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA>
##    T    U    D    V    E    W    F    X    G    Y    Z
## <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA>
```

Behind the scenes, the `caliper` argument restricts how the maximum distance between matched objects. For example, consider Table 1 above. Plants A and H are 1.3 units apart in `date`. If we assigned `caliper=1`, they could never be matched because they exceed the caliper limit.

If before matching you want to remove just the subjects lacking a counterpart within caliper distance, you can do `pairmatch(..., remove.unmatchables = TRUE)`. That won't help with the minuscule caliper above, but with less extreme calipers it helps you salvage a few matches.

How closely did I match?

Getting back to a matching that succeeded, note that `summary()` reports information about how close the matches are.

```
> summary(pm)
## Structure of matched sets:
## 1:1 0:1
##   7  12
## Effective Sample Size:  7
## (equivalent number of matched pairs).
```

Did matching balance the covariate?

Comparing overt biases before and after matching. An assessment of the unmatched difference between the groups on `cap` can be had via:

```
> cap.noadj <- lm(cap ~ pr, data = nuke.nopt)
> summary(cap.noadj)
```

The output is suppressed, as most of it is not relevant to balance. This variation hones in on the part that is:

```
> summary(lm(cap ~ pr, data = nuke.nopt))$coeff["pr",]
##    Estimate Std. Error    t value   Pr(>|t|)
## 79.7368421 92.7031668  0.8601307  0.3982280
```

(Note again the use of square brackets, [and], for specifying subsets of a matrix. With *R* one has to carefully distinguish square brackets, curly brackets and parentheses.)

Here is a parallel calculation that takes the match `pm` into account.

```
> summary(lm(cap ~ pr + pm, data = nuke.nopt))$coeff["pr",]
##   Estimate Std. Error    t value   Pr(>|t|)
##  1.7142857  5.1395492  0.3335479  0.7500689
```

The RItools package's `xBalance` function zeroes in on balance, and facilitates checking balance on multiple variables at the same time. Here are some examples:

```
>
> xBalance(pr ~ cap + t2, report="all", data=nuke.nopt)
##      strata unstrat
##      stat      pr=0     pr=1 adj.diff adj.diff.null.sd std.diff      z
## vars
## cap        803.263 883.000   79.737            92.220    0.380  0.865
## t2          59.526  66.857    7.331             4.547    0.738  1.612
## ---Overall Test---
##        chisquare df p.value
## unstrat      2.71  2   0.258
## ---
## Signif. codes:  0 '***' 0.001 '** ' 0.01 '*  ' 0.05 '.  ' 0.1 '   ' 1
> xBalance(pr ~ cap + t2 + strata(pm),
+          data=nuke.nopt,
+          report=c("adj.mean.diffs", "std", "z"))
##      strata    Unadj                          pm
##      stat   adj.diff std.diff       z     adj.diff std.diff      z
## vars
## cap        79.73684 0.38030  0.86464      1.71429  0.00818  0.35698
## t2          7.33083 0.73777  1.61209      4.71429  0.47444  1.16892
```

Exercises.

1. Compare `pm`, `tm` and the unmatched samples in terms of balance on `t2`.
2. Compare `pm`, `tm` and the unmatched samples in terms of balance on `date`.
3. Compare `pm` to Mahalanobis pair matching on `t1` in terms of balance on `date`.
4. Compare Mahalanobis pair matching on `cap` and `date` to Mahalanobis pair matching on `cap`, `date` and each of `t1,t2`. Add the last two variables in one at a time, so that you're comparing a total of three matches. Compare on balance in `cap` and `t2`.

Section "Checking balance in general", below, presents convenient ways to do balance assessment for many variables at once. Before getting to that let's discuss try matching with propensity scores.

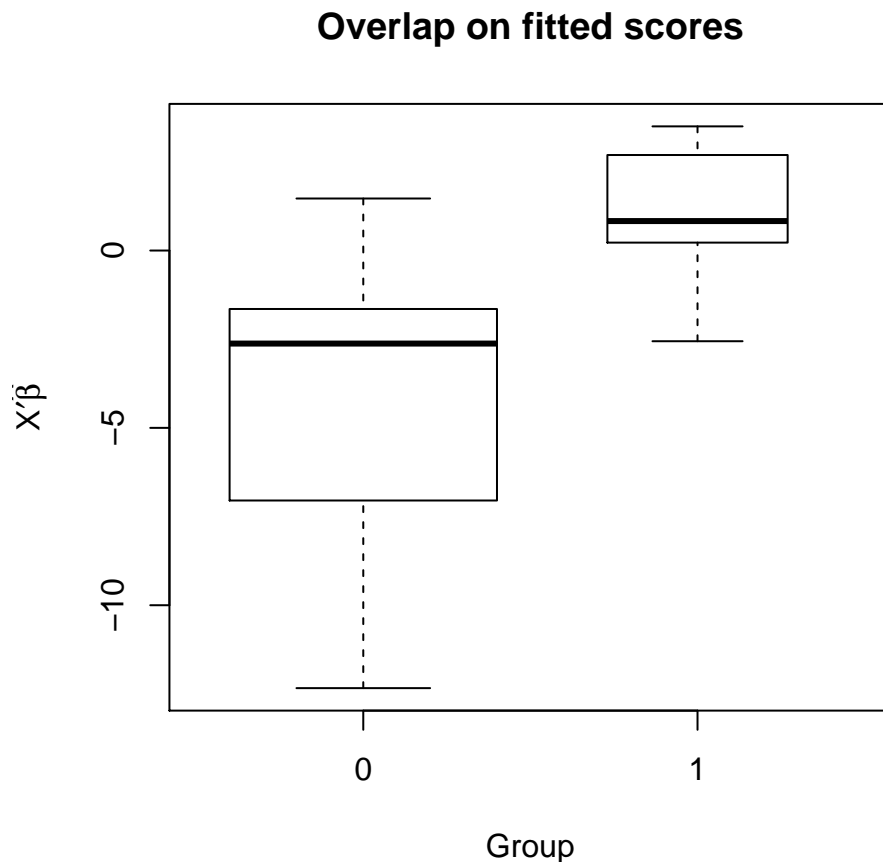## Propensity Score Matching

Propensity score fitting in *R*

Logistic regression models are fit in *R* using the function `glm()`, with "`family`" argument set to "`binomial`."
Example:

```
> psm <- glm(pr ~ date + t1 + t2 + cap + ne + ct + bw + cum.n + pt,
+            family = binomial, data = nuclearplants)
```

The fitted logistic regression is then stored in the object "`psm`." The propensity scores can be accessed with `psm$fitted.values` (estimated probabilities) or `scores(psm)` (estimated logits of probabilities).

It's often a good idea to compare the groups' distributions on the propensity score.

```
> boxplot(psm)
```


Overlap on fitted scores

The groups do overlap, if not greatly. It may be wise to restrict the sample to the region of overlap, at least roughly. A propensity caliper would help with this.

First, lets match directly on the propensity score without restricting the sample to the region of overlap.

```
> ps.pm <- pairmatch(psm, data = nuclearplants)
> summary(ps.pm)
## Structure of matched sets:
## 1:1 0:1
##  10  12
```

```
## Effective Sample Size:  10
## (equivalent number of matched pairs).
```

To restrict to the overlapping region, we want to imply a caliper to the distances generated by the propensity score model. To do this requires a more explicit generation of the match, involving separate steps for generation of the distances followed by matching upon those distances.

First, we create a distance matrix based upon `psm`:

```
> psm.dist <- match_on(psm, data=nuclearplants)
```

`psm.dist` is a matrix with an entry corresponding to the distance between each potential pair of treatment and control units. We can caliper directly on this distance matrix,

```
> caliper(psm.dist, 2)
##       control
## treated  H   I   J   K   L   M   N   O   P   Q   R   S   T U   V   W   X
##       A Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf 0   0 Inf Inf
##       B Inf   0 Inf Inf Inf   0   0   0 Inf   0   0   0   0 0   0   0 Inf
##       C Inf   0   0 Inf Inf   0   0   0 Inf   0   0   0   0 0 Inf   0   0
##       D Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf   0 Inf 0   0 Inf Inf
##       E Inf   0 Inf Inf Inf   0   0   0 Inf   0   0   0   0 0   0   0   0
##       F Inf   0 Inf Inf Inf   0 Inf   0 Inf Inf   0   0   0 0   0 Inf Inf
##       G Inf   0 Inf Inf Inf   0   0   0 Inf   0   0   0   0 0   0   0 Inf
##       a Inf   0 Inf Inf Inf Inf Inf Inf Inf Inf   0   0   0 0   0 Inf Inf
##       b Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf 0   0 Inf Inf
##       c Inf   0 Inf Inf Inf   0   0   0 Inf   0   0   0   0 0   0   0 Inf
##       control
## treated  Y   Z d   e   f
##       A Inf Inf 0 Inf Inf
##       B Inf   0 0 Inf   0
##       C Inf   0 0 Inf   0
##       D Inf Inf 0 Inf Inf
##       E Inf   0 0 Inf   0
##       F Inf   0 0 Inf   0
##       G Inf   0 0 Inf   0
##       a Inf   0 0 Inf   0
##       b Inf Inf 0 Inf Inf
##       c Inf   0 0 Inf   0
```

Entries which are `Inf` will never be matched. Adding the caliper to `psm.dist` will disallow matching between units which differ by more than 2 standard deviations.

(Note that this differs from the previous use of `caliper` directly in `pairmatch`, where the caliper is applied directly to the distances instead of smartly upon the standard deviations between the propensity scores.)

Combining the above, we can now

```
> ps.pm2 <- pairmatch(psm.dist, data = nuclearplants)
> ps.pm3 <- pairmatch(psm.dist + caliper(psm.dist, 2), data = nuclearplants)
> all.equal(ps.pm, ps.pm2, check.attributes=FALSE)
## [1] TRUE
> all.equal(ps.pm, ps.pm3, check.attributes=FALSE)
## [1] "8 string mismatches"
> summary(ps.pm3)
## Structure of matched sets:
## 1:1 0:1
```

```
##  10  12
## Effective Sample Size:  10
## (equivalent number of matched pairs).
```

Or you could match within calipers of the propensity score on some other distance, perhaps Mahalanobis distances based on selected covariates as recommended by Rubin and Thomas (2000, JASA) and others. For Mahalanobis matching on `date`, `cap` and the propensity score, for instance, combined with a propensity caliper of 1 pooled sd.

```
> mhd1 <- match_on(pr ~ date + cap + scores(psm), data=nuclearplants)
> mhpc.pm <- pairmatch(mhd1, caliper=1, data=nuclearplants)
> summary(mhpc.pm) # oops
## Structure of matched sets:
## 1:1 0:1
##  10  12
## Effective Sample Size:  10
## (equivalent number of matched pairs).
> mhpc.pm <- pairmatch(mhd1, caliper=2, data=nuclearplants)
> summary(mhpc.pm) # better!
## Structure of matched sets:
## 1:1 0:1
##  10  12
## Effective Sample Size:  10
## (equivalent number of matched pairs).
```

Checking balance in general

The `RItools` package has a convenient function for checking balance on many variables simultaneously. To get a sense of what it does, try this:

```
> library(RItools)
> xBalance(pr ~ date + t1 + t2 + cap + ne + ct + bw + cum.n, data = nuclearplants)
> xBalance(pr ~ date + t1 + t2 + cap + ne + ct + bw + cum.n + pt +
+            strata(ps.pm2) -1, # the `-1` just focuses the output a little
+          data = nuclearplants)
```

It can in the same display compare matching to no matching, on any of a number of axes. Here is a demonstration:

```
> myb <- xBalance(pr ~ date + t1 + t2 + cap + ne + ct + bw + cum.n +
+                 strata(ps.pm2),
+                 data = nuclearplants,
+                 report = c("adj.means", "std.diffs",
+                            "z.scores", "chisquare.test"))
> plot(myb)
```

Standardized Differences
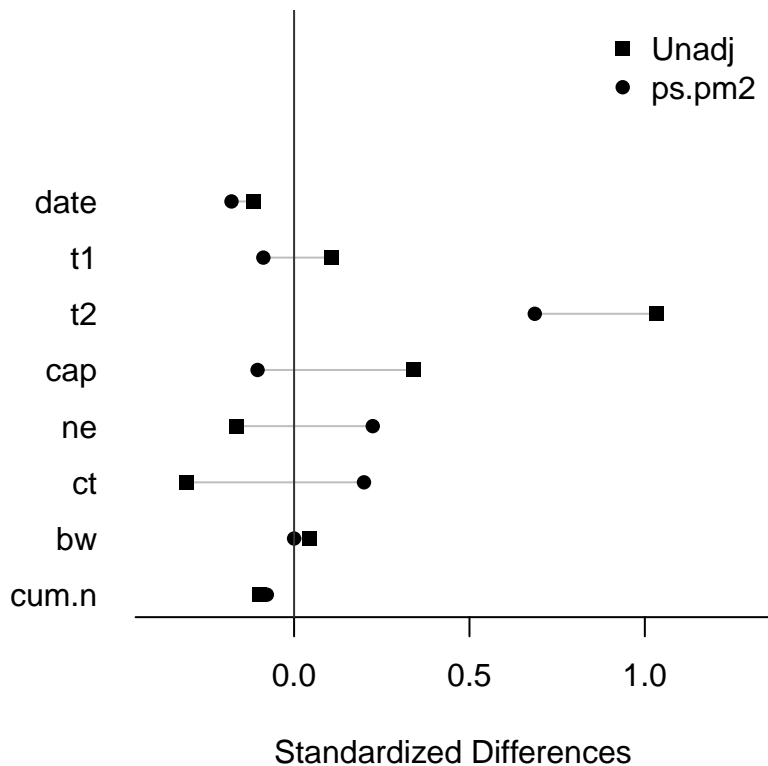
```
> print(myb, digits=1)
##       strata  Unadj                              ps.pm2
##       stat     pr=0   pr=1 std.diff     z          pr=0    pr=1 std.diff     z
## vars
## date          68.62  68.50    -0.11   -0.31        68.68  68.50    -0.18   -0.36
## t1            13.64  14.00     0.11    0.28         14.30  14.00    -0.09   -0.17
## t2            59.32  69.10     1.03    2.47   *     62.60  69.10     0.69    1.43
## cap          805.18 869.80     0.34    0.89        889.60 869.80    -0.10   -0.29
## ne            0.27   0.20     -0.16   -0.43         0.10   0.20      0.22    0.58
## ct            0.45   0.30     -0.31   -0.81         0.20   0.30      0.20    0.45
## bw            0.18   0.20      0.05    0.12         0.20   0.20      0.00    0.00
## cum.n         8.73   8.10     -0.10   -0.26         8.60   8.10     -0.08   -0.19
## ---Overall Test---
##       chisquare df p.value
## Unadj        11  8     0.2
## ps.pm2        9  8     0.4
## ---
## Signif. codes:  0 '***' 0.001 '** ' 0.01 '*  ' 0.05 '.  ' 0.1 '   ' 1
```

For a very compact representation of the assessment, call `summary()` on the match, passing along the fitted propensity model as a second argument. If you've got `RItools` loaded, then this will call `xBalance` in the background, reporting the summary chi-square test results.

```
> summary(ps.pm2, psm)
## Structure of matched sets:
## 1:1 0:1
##  10   12
## Effective Sample Size:  10
## (equivalent number of matched pairs).
##
## Balance test overall result:
##   chisquare df p.value
##         10  9   0.351
```

Exercise.

Try out {at least 3} different combinations of propensity score and Mahalanobis matching. Identify the matches that gives:

1. the "best balance overall," as measured by Chi-square statistics;
2. the "best balance overall," as indicated by the largest standardized difference among the covariates;
3. the best balance on `date` and `cap`, as measured by the larger of the standardized differences for these two variables; and
4. the best balance on `date` and `cap`, as measured by the larger of the standardized differences for these two variables, among those matches for which the overall imbalance p-value is no more than .1.

Other topics in matching

Full matching and matching with a varying number of controls

Try out for yourself and compare:

```
> summary(fullmatch(pr ~ date + cap, data = nuke.nopt))
> summary(fullmatch(pr ~ date + cap, data = nuke.nopt, min = 1))
> summary(fullmatch(pr ~ date + cap, data = nuke.nopt, min = 2, max = 3))
```

Subclassification before matching

Recall that the data set `nuclearplants` had 32 observations, 6 of which we excluded. These were plants built under "partial turnkey" guarantees (`pt == 1`), for which costs are difficult to compare with other plants. We might include the excluded plants by matching them only among themselves. Then we need to subclassify prior to matching.

This is a common and useful operation, and `optmatch` is designed to help you do it via a `strata` term in the matching formula.

```
> pairmatch(pr ~ date + cap + scores(psm), data=nuclearplants)
##    H    I    A    J    B    K    L    M    C    N    O    P    Q    R    S
## <NA>  1.1  1.1 <NA>  1.2 <NA> <NA>  1.4  1.3  1.3 <NA> <NA> <NA>  1.2 <NA>
##    T    U    D    V    E    W    F    X    G    Y    Z    d    e    f    a
##  1.8  1.7  1.4  1.5  1.5 <NA>  1.6  1.6  1.7 <NA> <NA>  1.9 <NA> 1.10  1.8
##    b    c
##  1.9 1.10
> pairmatch(pr ~ date + cap + scores(psm) + strata(pt), data=nuclearplants)
##    H    I    A    J    B    K    L    M    C    N    O    P    Q    R    S
## <NA>  0.1  0.1 <NA>  0.2 <NA> <NA>  0.4  0.3  0.3 <NA> <NA> <NA>  0.2 <NA>
##    T    U    D    V    E    W    F    X    G    Y    Z    d    e    f    a
## <NA>  0.7  0.4  0.5  0.5 <NA>  0.6  0.6  0.7 <NA> <NA>  1.2  1.1  1.3  1.1
```

```
##     b     c
##   1.2   1.3
```

Distances

You'll often want to do several variations on a match. It may save computation and typing time to store the distance you're using to match, if you're going to re-use that distance. To do so, you'll have to explicitly separate distance-making and matching, two steps that we've merged together thus far. We'll use the `match_on()` to create distances.

```
> cap.dist <- match_on(pr ~ cap, data = nuke.nopt)
> pm1 <- pairmatch(pr ~ cap, data=nuke.nopt)
> pm2 <- pairmatch(cap.dist, data=nuke.nopt)
> all.equal(pm1, pm2, check.attributes = FALSE)
## [1] TRUE
> summary(pm2)
## Structure of matched sets:
## 1:1 0:1
##   7  12
## Effective Sample Size:   7
## (equivalent number of matched pairs).
```

What does a matching distance look like? Here's the upper-left corner of one of them:

```
> round(cap.dist[1:3, 1:3], 1)
##            control
## treatment   H   I   J
##         A 1.8 0.0 0.0
##         B 1.8 0.0 0.0
##         C 0.6 1.2 1.2
```

(Note the use of square brackets, `[` and `]`, for specifying rows and columns of the distance matrix. If you find that this isn't working on a distance that you've produced, try `as.matrix(my.dist)[1:3,1:3]` or similar.)

Matching with a caliper of 2 pooled standard deviations on the `cap` variable:

```
> round(cap.dist + caliper(cap.dist, 2), 1)
##          control
## treated   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W
##        A 1.8 0.0 0.0 Inf 1.2 Inf 1.3 Inf 1.3 Inf 0.1 1.0 1.4 1.0 0.1 0.7
##        B 1.8 0.0 0.0 Inf 1.2 Inf 1.3 Inf 1.3 Inf 0.1 1.0 1.4 1.0 0.1 0.7
##        C 0.6 1.2 1.2 1.5 0.0 1.7 0.1 1.2 0.2 1.4 1.1 0.1 0.2 0.1 1.3 0.4
##        D 0.7 Inf Inf 0.1 1.4 0.3 1.2 0.1 1.2 0.0 Inf 1.5 1.2 1.5 Inf 1.8
##        E 1.7 0.1 0.1 Inf 1.1 Inf 1.2 Inf 1.2 Inf 0.0 1.0 1.3 1.0 0.2 0.7
##        F 0.7 1.1 1.1 1.5 0.0 1.8 0.2 1.3 0.2 1.4 1.1 0.1 0.2 0.1 1.2 0.4
##        G 0.6 1.2 1.2 1.5 0.0 1.7 0.1 1.2 0.1 1.4 1.1 0.1 0.2 0.1 1.3 0.4
##          control
## treated   X   Y   Z
##        A 1.3 Inf 0.3
##        B 1.3 Inf 0.3
##        C 0.2 1.4 1.5
##        D 1.2 0.0 Inf
##        E 1.3 Inf 0.4
##        F 0.2 1.4 1.4
##        G 0.2 1.3 1.5
```

Entries of `Inf` or `NaN` in a distance matrix are interpreted as forbidden matches. Thus

```
> pairmatch(cap.dist + caliper(cap.dist, 2), data = nuke.nopt)
```

matches on `cap`, insisting that paired units not differ by more than 2 pooled SDs in `cap`. If you would prefer to set a requirement on how much paired units can differ in `cap` in its original units, rather than standard units, then you would have to create `cap.dist` in a somewhat different way. There's an example on the help page for `caliper()`. Enter `help(caliper)` at the R command line.

Using R for matching and another program for matched analysis

If you prefer to do your main work in another statistical package, you can use R for matching and balance assessment, and then re-import the matched data back into your preferred package for the main analysis. A typical work flow might look something like this:

1. Load data into your primary statistical package.
2. Preprocess data (handle missing values, combine variables into single measures, etc).
3. Export data in a format R can read.
4. Load data into R.
5. Perform matching and balance testing.
6. Append matches to your data and export in a format your statistical package can read.
7. Load appended data into your primary package and perform analyses.

You already know how to perform the matching and balance testing, so what remains are the import and export steps, along with appending your matches to your original data.

When importing and exporting data, you must select a data format. In a broad sense, you have two options: proprietary data formats (for example `.dta` for Stata) or open standards (for example `.csv`, Comma Separated Values). The advantage of proprietary formats is that they may include additional information for R to use, such as the labels on a categorical variable. The potential pitfall is that R might not know how to read your particular file type. Open formats like CSV are easy to read and write, but you may lose variable names, labels, or other special forms of data. For reading and writing proprietary formats, consider the use of the `foreign` library in R. See for example `read.dta` and `write.dta`:

```
> library(foreign)
> ?read.dta
> ?write.dta
```

For much more detail on this topic, see the "R Data Import/Export" manual.

If you use CSV files, consult your statistical package's manuals for details on how to export your data as a CSV. Once you have exported your data into a file, open the file (using a text editor, not Microsoft Word) to see if there is a header row of variable names. Then, to import your data use the following:

```
> my.plants <- read.csv("nuclearplants.csv", header = TRUE)
```

If you do not have a header on your data change to `header = FALSE`.

At this point, you may proceed to conduct your matching as usual (for example using an example from earlier in this document). When you are done, you will need to append your match to your original data. The safest way to do this is using the following code snippet:

```
> plant.match <- pairmatch(pr ~ cap, data = my.plants)
> my.plants.extended <- data.frame(my.plants, matches = plant.match, check.rows=TRUE)
```

Finally, export your data as a .csv file (or a proprietary format such as with `write.dta` if that is what you are using):

```
> write.csv(my.plants.extended, file = "nuclearplants-with-matches.csv")
```

As one final note, many of the balance tests included in **RItools** are available to Stata users directly from Stata. Information is available at the **RItools** webpage at {http://www.jakebowers.org/RItools.html}.

Trying it out on your own

For another interesting (toy) data set, do

```
> data(tli, package="xtable")
> head(tli)
##   grade sex disadvg ethnicty tlimth
## 1     6   M     YES HISPANIC     43
## 2     7   M      NO    BLACK     88
## 3     5   F     YES HISPANIC     34
## 4     3   M     YES HISPANIC     65
## 5     8   M     YES    WHITE     75
## 6     5   M      NO    BLACK     74
```

You might compare test scores for kids with `disadvg=="YES"` to those of kids with`disadvg=="NO"` using propensity matching, in some combination with Mahalanobis matching and caliper matching. A check of propensity overlap may inform your decision as to whether to include a propensity caliper. Be sure to check for balance, and do check the structure of the matched sets.

Three sources of *real* data can be gotten as follows. First, Paul Rosenbaum has posted many of the data sets discussed in his *Design of Observational Studies* (2010) to his web site. If you have an active internet connection then you can get them by (as of this writing):

```
> download.file("http://www-stat.wharton.upenn.edu/~rosenbap/DOSdata.RData",
+                destfile="./DOSdata.RData")
> load("./DOSdata.RData")
```

Second, the "`lalonde`" data set, discussed by Lalonde (1986, *Am. Econom. Rev.*), Dehejia and Wahba (1999, *JASA*) and Smith and Todd (2005, *J. Econom.*), is bundled with several *R* packages, including "{arm}" and "`Matching`." To get it:

```
> install.packages("arm", dep=T) # if not already installed
> data(lalonde, package="arm")
> help("lalonde", package="arm")
```

Third, the data used by Connors et al (1996, *J. Am. Med. Assoc.*) to examine costs, benefits and risks associated with right heart catheterization is bundled with Frank Harrell and collaborators' "**Hmisc**" package.

```
> install.packages("Hmisc", dep=T) # if not already installed
> Hmisc:::getHdata(rhc, what = "all")
```