CITY AND REGIONAL PLANNING 5700
CIVIL ENGINEERING 5700

Discrete-Choice Logit Models with R

Philip A. Viton

July 14, 2015

Contents

1 Introduction to R **2**
1.1 Some general tips for R 3

2 Some Screenshots of R **4**

3 Data input **8**
3.1 Reading your data 8
3.2 Checking your data 10
3.3 Saving and loading your data 11

4 Setting up your data for mlogit **12**
4.1 The dependent (choice indicator) variable 12
4.2 The choice set for each individual 13
4.3 Panel-data indicators 13
4.4 Varying choice sets 14
4.5 Simplified mlogit data specification 14

5 Formulas — specifying your model **15**
5.1 Basic formulas 15
5.2 mlogit extensions to formulas 16

6 Estimating logit models **18**
6.1 Standard logit 18
6.2 Saving your work 21
6.3 Mixed-logit models 22

6.4 Choice-based sampling 26
6.5 Other logit models 27

A The script 28

B The clogit data 30

C Binary logit 32

D Large-sample panel-data mixed logit 33

E Other statistical functions in R 35

F RStudio 35

G Further reading 37

1 Introduction to R

This note explains the basics of using the R statistics system to estimate discrete-choice logit models. The R system has several advantages : (a) it is free; (b) it runs on just about every platform including Windows, the Mac, and Linux; (c) it has perhaps the best graphics of any statistics package, including commercial; (d) it is extraordinarily comprehensive: for just about any statistical procedure you can think of, it is likely that someone has written an R package to handle it; (e) it has a comprehensive set of spatial statistics commands (including spatial econometrics), which are lacking even in many commercial programs; (f) a basic R system can take up as little as 50MB of disk space and can be run off a USB drive; (g) did I mention that it is completely free? The one potential downside is that, if you are used to a point-and-click interface (as in Systat, for example), R will take a bit of getting used to since it is a command-line system. See Appendix G for some reading suggestions.

In this note, I concentrate on using R's `mlogit` package (written by Yves Croissant) for estimation of discrete choice models. While there are other R packages that can do this — for example, `arm` and `zelig` — only `mlogit` can estimate some of the more advanced models like mixed-logit. It therefore seems worthwhile learning to use `mlogit` right away, since you'll probably end up using it eventually. As of version 0.2-2 `mlogit` can also estimate the multinomial probit model and automatically takes care of the rather complicated restrictions on its covariance matrix. See Ken Train's book, referred to in Appendix G, for more on specifying multinomial probit models.

`mlogit` is a very capable package, but there are two things it cannot do. First, it cannot compute the (conditional) distributions of the individual coefficients in a mixed-logit panel-data setting, as described in Chapter 11 of Train's book. The only packages I know that can do this are NLOGIT (an extra-cost add-on to the commercial LIMDEP package) and `mixlogit` for Stata (`mixlogit` is free, but Stata itself is commercial). Second, it will not handle choice-based sampling: see below, section 6.4.

R can be downloaded from http://www.r-project.org/ . Once it is installed, you need to add the `mlogit` package, since it is not a default package supplied with the system. Start R, then do Packages -> Install package(s) and follow the prompts.

1.1 Some general tips for R

- *R is case-sensitive* (like most Unix/Linux programs, but unlike Windows). The names `myvar` and `Myvar` do not refer to the same thing.
- The usual way to use R for estimation is to assign the result of an estimation command to a variable, and then work with the contents of that variable: see, eg, section 6, below. The standard assignment operator is `<-` (two characters, with no space between them).
- File names: under Windows, my suggestion is always to use Unix/Linux path separators (`/`). Thus, you'd say, eg, `c:/mydir/myfile.txt`. It is possible to use the normal Windows path separator (\ ) but it must be doubled up, which is inconvenient: in the example given, you'd have to write `c:\\mydir\\myfile.txt`. If you forget to do this, you'll get errors.

  Under Windows, R's case sensitivity does not extend to system-level operations. So if you want to read or write the file `myfile.txt` you could equally well call it `MyFile.txt`. The same applies to all elements of the file path.
- R's comment character is `#` : anything following it on a line is ignored.
- The easiest way to exit R is via the usual icons on the application's title bar; but you can also do it by typing `quit()` in the R console.

- R uses a package structure, and loads procedures when you ask it to: to get access to procedures in package `pck`, load the package via the command `library(pck)`. The package name is case-sensitive.

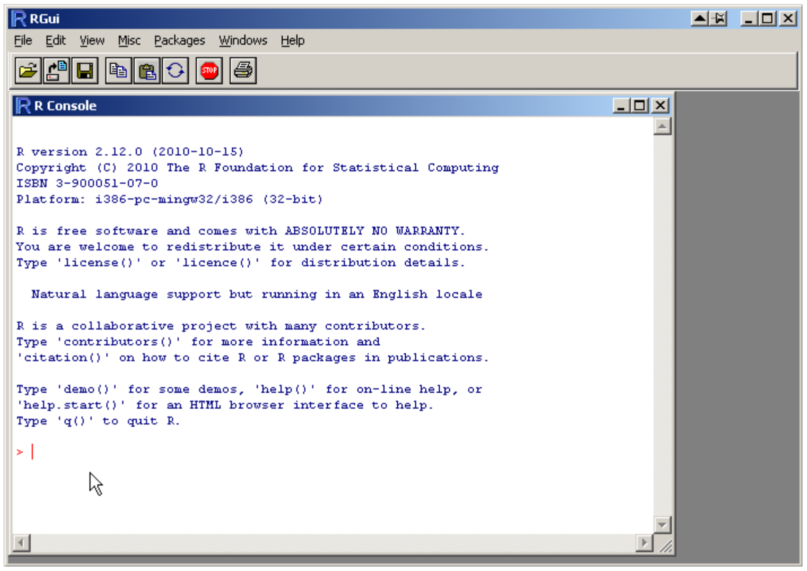
- To get help on the topic `lm` (which happens to be linear regression) do `?lm` or `help("lm")`. This works only for procedures that have already been loaded.[1] For a more general search do `help.search("mlogit")` : this searches installed packages, whether loaded or not. The result is a listing of packages which may contain the concept you're interested in, in the form `package::function`. You can then do `?package::function` (eg `?mlogit::mlogit`, since the function "mlogit" is in the package "mlogit") for more information.

2 Some Screenshots of R

Here are some pictures of R running under its default GUI interface in MS-Windows. There are other GUI interfaces which you may find more productive: one very nice multi-platform GUI is RStudio, available from http://rstudio.org/download/desktop. See Appendix F for additional information on RStudio, and a screenshot.

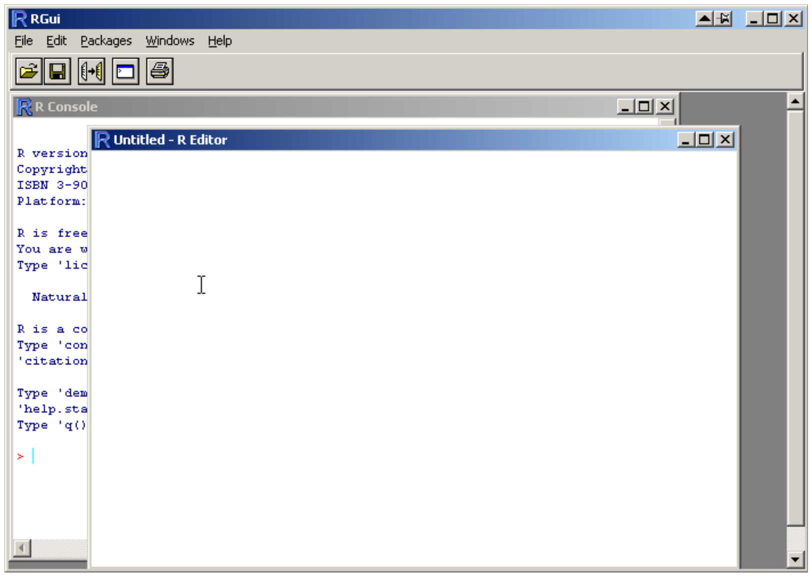
When R starts, you see its console interface: you can enter commands directly at the $>$ prompt.

[1]A large number of basic tools — including linear regression — are automatically loaded when R starts.



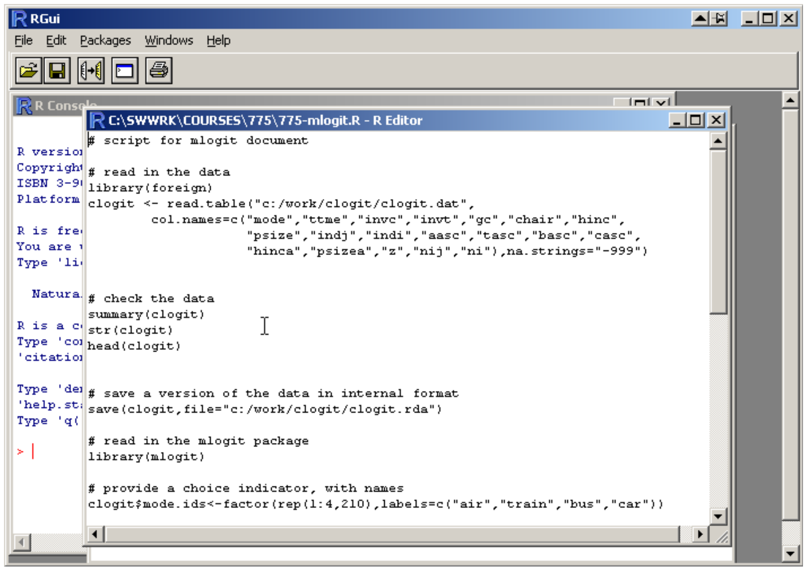
```
R version 2.12.0 (2010-10-15)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

But my suggestion is that you record your commands in a script file, which you can then save and re-run if you need to to. Also, it is easier to correct typos from a script file. To start a blank script file, click on File -> New Script. The picture below shows an empty script window: note that the icons have changed. To save your script, click on the diskette icon (second from the left) or do File -> Save. The usual extension for scripts is `.R` but it can be anything you like. R scripts are plain ASCII files, which makes editing them easy, even outside the R system.



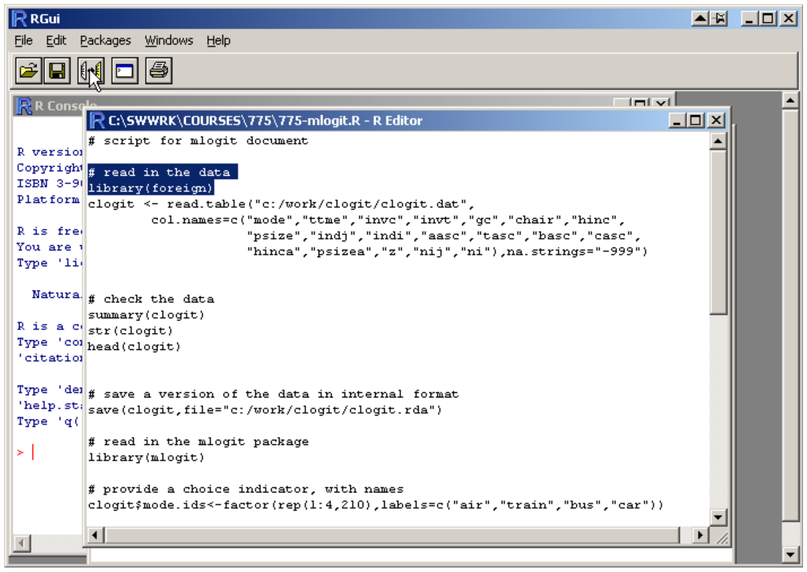
```
R version
Copyright
ISBN 3-90
Platform:

R is free
You are w
Type 'lic

  Natural

R is a co
Type 'con
'citation

Type 'dem
'help.sta
Type 'q()

>
```

For our work here, I have already created a script containing my commands, so I just read it in via File->Open or the open-script icon (left-most icon). The result is shown in the next picture:


RGui
File Edit Packages Windows Help
R Console
C:\SWWRK\COURSES\775\775-mlogit.R - R Editor

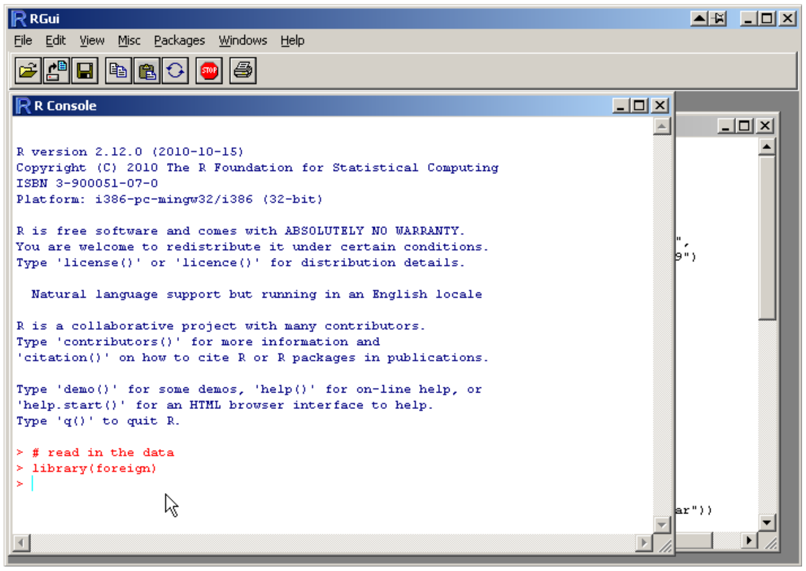
```
# script for mlogit document

# read in the data
library(foreign)
clogit <- read.table("c:/work/clogit/clogit.dat",
        col.names=c("mode","ttme","invc","invt","gc","chair","hinc",
                    "psize","indj","indi","aasc","tasc","basc","casc",
                    "hinca","psizea","z","nij","ni"),na.strings="-999")


# check the data
summary(clogit)
str(clogit)
head(clogit)


# save a version of the data in internal format
save(clogit,file="c:/work/clogit/clogit.rda")

# read in the mlogit package
library(mlogit)

# provide a choice indicator, with names
clogit$mode.ids<-factor(rep(1:4,210),labels=c("air","train","bus","car"))
```

Once I have something in my script window, I can highlight one or more lines and then send them to the main engine for processing. In the picture below, I highlight the line to load the library (`foreign`) that I'll need to process a csv file (more details in the next section). I then click on the Run Line or Selection icon (third from left, note the cursor arrow in the picture) and the selection is executed.


RGui
File Edit Packages Windows Help
R Console
C:\SWWRK\COURSES\775\775-mlogit.R - R Editor

```
# script for mlogit document

# read in the data
library(foreign)
clogit <- read.table("c:/work/clogit/clogit.dat",
        col.names=c("mode","ttme","invc","invt","gc","chair","hinc",
                    "psize","indj","indi","aasc","tasc","basc","casc",
                    "hinca","psizea","z","nij","ni"),na.strings="-999")


# check the data
summary(clogit)
str(clogit)
head(clogit)


# save a version of the data in internal format
save(clogit,file="c:/work/clogit/clogit.rda")

# read in the mlogit package
library(mlogit)

# provide a choice indicator, with names
clogit$mode.ids<-factor(rep(1:4,210),labels=c("air","train","bus","car"))
```

As we can see in the picture below, the selection is echoed to the main R window, including the comment.



```
R version 2.12.0 (2010-10-15)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> # read in the data
> library(foreign)
> 
```

This is the recommended procedure for analysis using R: record commands in a script (window) and then send them to the R system for execution.

3 Data input

All the material in this section applies to R in general, and is independent of the `mlogit` package.

3.1 Reading your data

The `foreign` package provides tools to get your data into (and out of) R, including input from data files created by other statistical package, like SAS or SPSS.[2] By way of example, I shall use the `clogit` data set, which is also used by the Limdep system.[3] This dataset reports the result of a survey of the intercity modal choices of 210 Australians, conducted by David Hensher. (As it happens, this is a *choice-based* dataset, but we shall ignore this until section 6.4). Each individual faces a

[2]For SAS there is also a more advanced package, `SASxport`.
[3]See Appendix B for links to this data.

choice set consisting of 4 alternatives: air, train, bus and car.

The file `clogit.dat` contains 840 (= 210 individuals $\times$ 4 choices per individual) rows and 19 columns. Each row is purely numeric and each column is separated from the next by one or more spaces. To read in the data, we first load the `foreign` package, and then use the package's `read.table` function:

```
library(foreign)
clogit <- read.table("c:/work/clogit/clogit.dat",
        col.names=c("mode","ttme","invc","invt","gc","chair","hinc",
                    "psize","indj","indi","aasc","tasc","basc","casc",
                    "hinca","psizea","z","nij","ni"),na.strings="-999")
```

The result is that the data is read into a special R object called a dataframe, which we name `clogit`.[4] Data frames are containers for data: they can contain variables of any type, the only restriction being that each variable must have the same number of observations (rows).

Some notes on this:

- We provide variable names via the `col.names` argument. The syntax `c(...)` stands for the *concatenation* of what is inside, here a comma-separated list of names in quotes. If you omit the `col.names` argument, then the variables will automatically be named `V1, V2, ....`[5]
- The `na.strings` argument tells R what counts as missing data. By default, missing data is assumed to be coded by the string "`NA`". In this case the data was exported from Limdep, which uses −999 as its missing-data code, so we need to tell R that entries of −999 are not real data.
- For Windows users, it may be simplest to format your data as a comma-separated-values file (this is easy to do from Excel), in which case the `foreign` package provides a function `read.csv` which does most of the work for you. All we would need to do in this case is:
  `clogit <- read.csv("c:/work/clogit/clogit.csv").`
  To export the dataframe `clogit` as a csv file, do:

[4] See Appendix B for a full explanation of what the variables are.

[5] Another way to provide variable names is to include them on the first line of the data set, where they must have the same separators as the data itself (here, spaces). In this case you'd omit the `col.names` argument and instead say `header=TRUE` .

`write.csv(clogit, file="c:/mydir/myfile.csv")`
There are some additional options to `write.csv` that you may want to review (for example, whether to include row labels, how to handle missing data). Do `?write.csv` for more information.

3.2 Checking your data

Once the data has been read in, you will probably want to see what's there, to reassure yourself that everything has gone smoothly. R provides a number of ways to do this: `summary(clogit)` will provide some basic descriptive statistics on each variable (including the number of missing observations, if any are detected), while `str(clogit)` provides more technical information, including data types and the first few observations on each variable. (`str` stands for "structure"). To see the beginning of the dataset you can use the `head` function. This will try to be intelligent about how many observations it displays, but you can control it by providing an optional second integer argument: `head(clogit,20)` will display the first 20 observations. If the integer is negative, you will see the *last* observations. Here is the beginning of what each of these produces:

```
> summary(clogit)
      mode            ttme             invc              invt
 Min.   :0.00   Min.   : 0.00   Min.   :  2.00   Min.   :  63.0
 1st Qu.:0.00   1st Qu.: 0.75   1st Qu.: 23.00   1st Qu.: 234.0
 Median :0.00   Median :35.00   Median : 39.00   Median : 397.0
 Mean   :0.25   Mean   :34.59   Mean   : 47.76   Mean   : 486.2
 3rd Qu.:0.25   3rd Qu.:53.00   3rd Qu.: 66.25   3rd Qu.: 795.5
 Max.   :1.00   Max.   :99.00   Max.   :180.00   Max.   :1440.0

> str(clogit)
'data.frame':   840 obs. of  19 variables:
 $ mode  : num  0 0 0 1 0 0 0 1 0 0 ...
 $ ttme  : num  69 34 35 0 64 44 53 0 69 34 ...
 $ invc  : num  59 31 25 10 58 31 25 11 115 98 ...
 $ invt  : num  100 372 417 180 68 354 399 255 125 892 ...
 $ gc    : num  70 71 70 30 68 84 85 50 129 195 ...
 $ chair : num  0 0 0 0 0 0 0 0 0 0 ...
 $ hinc  : num  35 35 35 35 30 30 30 30 40 40 ...
 $ psize : num  1 1 1 1 2 2 2 2 1 1 ...
 $ indj  : num  -2 0 0 1 -2 0 0 1 -2 0 ...

>head(clogit)
```

```
     mode ttme invc invt gc chair hinc psize indj indi aasc tasc basc casc hinca
1    0    69   59  100 70     0   35     1   -2    0    1    0    0    0    35
2    0    34   31  372 71     0   35     1    0   -1    0    1    0    0     0
3    0    35   25  417 70     0   35     1    0   -1    0    0    1    0     0
4    1     0   10  180 30     0   35     1    1    1    0    0    0    1     0
5    0    64   58   68 68     0   30     2   -2    0    1    0    0    0    30
6    0    44   31  354 84     0   30     2    0   -1    0    1    0    0     0
     psizea z nij ni
1      1 0   1  2
2      0 1   3  2
3      0 1   3  2
4      0 1   3  2
5      2 0   1  2
6      0 1   3  2
```

You can have several dataframes active in the same R session, and dataframes can contain variables with the same names. To refer to a variable within a particular dataframe, you use the dollar-sign syntax: `clogit$mode` refers to the variable `mode` in the `clogit` dataframe.

3.3 Saving and loading your data

For large datasets, or datasets that require extensive processing in R in order to make them useful, you may want to save them in R's internal format. (For small datasets that don't require much processing, there's very little advantage to this). You can do this with the `save` function:

```
save(clogit,file="c:/work/clogit/clogit.rdata")
```

The extension `.rdata` for R's internal data format is conventional, but not required. This saves the object (here, the `clogit` dataframe) named in the first argument. If you omit the first argument, then the entire contents of the workspace is saved. Once this has been done, you can use the `load` command to retrieve your data:

```
load("c:/work/clogit/clogit.rdata")
```

4 Setting up your data for mlogit

Once you've read in your data, you may need to make some adjustments in order to make it usable by `mlogit`.

4.1 The dependent (choice indicator) variable

The first thing to do is to ensure that you have a correctly coded choice variable. This will usually be straightforward, but remember that if you need to create such a variable yourself, it must be added to the dataframe. `mlogit` recognizes three ways to specify the choice variable (what we called y_{ij} in class):

1. It can be a $0/1$ indicator, with a 0 indicating an unchosen alternative and 1 indicating which alternative was actually selected. For the `clogit` dataset, the variable `mode` is of this type, so there's nothing more you need to do.

2. It can be a logical variable, where the special name `TRUE` indicates the alternative chosen, and `FALSE` indicates an alternative not chosen. Suppose that your data had a variable `cny` coded as `"c"` for 'chosen" and `"nc"` for "not chosen". You could convert it to `TRUE`/`FALSE` as follows: `clogit$choice<-clogit$cny=="c"`. This says that the new variable is `TRUE` when `cny` equals "`c`", and (by implication) `FALSE` otherwise.[6] Note the dollar syntax on the left of the assignment: we want the new variable to be part of the `clogit` dataframe.

3. It can be a *factor* taking on the values `yes` or `no`. In R, a factor is an efficient way of storing a variable whose values are from a (typically small) number of distinct possibilities. Instead of storing the full value for each observation, R will store a list of the possible values and then, for each observation, an integer indicating which one applies. This can save space, especially when the values are repeated character strings. To create this type of choice indicator, try `clogit$choice<-factor(clogit$mode,labels=c("no","yes"))`

[6] Note the doubled-up equal signs: if you don't do this and say instead `clogit$cny="c"`, you won't get what you were expecting: the single = is an assignment operator, and the result is that `cny` would have each element set to the character "c".

4.2 The choice set for each individual

We also need a way to tell the program what each individual's choice set is, which will be referred to in estimation commands as the value of `alt.var`. For this data set it's simple, since each individual faces the same four alternatives, in the given order. We can indicate the choice set by a vector of integers `1,2,3,4` repeated 210 times (since there are 210 individuals in the dataset). The resulting variable, which must be a factor and included in the dataframe, and which we'll name `mode.ids`. can be created as follows:

```
clogit$mode.ids<-factor(rep(1:4,210))
```

It would also be useful to provide names for each alternative, which we can do using an extended form of the `factor` command, as follows:

```
clogit$mode.ids<-factor(rep(1:4,210),
                 labels=c("air","train","bus","car"))
```

It's not necessary to provide names for the alternatives: if we don't do so, `mlogit` will number the alternatives, and you'd see alternative-specific constants as `1:intercept`, `2:intercept, ...` . Clearly, it's more informative to provide your own names.

4.3 Panel-data indicators

If your data contains repeated observations on each individual (as `clogit` does not), called a panel dataset, you may also need to tell the program which observations apply to which individual. (For discrete-choice models this will be relevant only if you will be estimating mixed-logit models). You do this by another factor variable, which will be referred to in estimation commands as the value of `id.var`. For example, suppose the data has three consecutive observations on each individual, with each individual facing the same 4 alternatives. Then the data for individual 1 would be the first 12 rows, individual 2 would be the second 12, and so on. We an construct a panel indicator (call it `indivs`) as follows, bearing in mind that we have 210 individuals:

```
clogit$indivs<-factor(rep(1:210,each=12))
```

The result is 12 1's, followed by 12 2's etc.

4.4 Varying choice sets

If the individuals in your sample face different-sized choice sets (for example, one individual doesn't own a car, or, in a stated-preference experiment, was not presented with all the options defined by the experiment) you need to indicate this too. You do this via another variable, which will be referred to as the `chid.var` variable in estimation commands.

There are two possibilities. In a non-panel dataset (each decision-maker appears just once in the dataset), suppose you have a variable (say `indiv`) that identifies the decision-maker. This will be repeated the correct number of times to allow the program to identify how many alternatives that individual faced. In a panel-data setting, the individual identifier will not be enough. In this case you must provide another variable that identifies the "choice setting". Suppose for example that in a panel-data setting individual 1 faced 3 alternatives in her first decision setting and 4 alternatives the next time she chose. In this case the individual identifier will be the same for all 7 rows, and you would need to set up a choice-setting identifier variable, for example `chid<-c(1,1,1,2,2,2,2, ...)` (assuming that individual 1's data was in the first 7 rows).

4.5 Simplified mlogit data specification

When you run a statistical command in R you need to tell it which data set to operate on. In the case of `mlogit` this involves a bit more than specifying just the name of the dataframe (see section 6.1), and can get a bit cumbersome. If you plan on running several models with the same dataset, it may be convenient to create a special form of the data that `mlogit` understands: this will make command specification simpler and estimation slightly faster.

To do this, we need to tell `mlogit` once-and-for-all what the choice variable is, and the values of `alt.var`, for a panel dataset, `id.var`; and if the choice set varies, `chid.var`. For `clogit` we can do this as follows, where we name the `mlogit` form of the data `CLOGIT` (in capitals, remember that R is case-sensitive):[7,8]

[7]The reason for the `shape` argument is that `mlogit` will also recognize an alternative `wide` form of the data in which each individual's decision is represented on one line, rather than on as many lines as there are alternatives, as here. See the `mlogit` write-up for more detail on this.

[8]One slight disadvantage here is that attempting to edit the `mlogit` form of the dataframe converts it to a standard dataframe, so you will want to do this only when the data is cleaned up and ready for

```
CLOGIT<-mlogit.data(clogit,shape="long",
                      choice="mode",alt.var="mode.ids")
```

Note that this is a non-panel dataset and everyone faces the same-size choice set, so we do not need to specify either `id.var` or `chid.var`.

The simplified dataset is much like the original — you can inspect it by any of the methods discussed in section 3.2— but there is also a subsidiary dataframe containing the various identification indices. You can inspect this via `head(index(CLOGIT),`*n*`)` where *n* is the number of rows you want to see.

5 Formulas — specifying your model

Many estimation commands in R (including linear regression and the discrete-choice models estimated by `mlogit`) use a special syntax called a *formula* to specify linear-in-parameters models. Here I discuss only the basics: try `?formula` for more information.[9]

5.1 Basic formulas

Formulas have the name of the dependent variable on the left, and the specification of the independent variables is on the right, with the two parts separated by a tilde (~). The right-hand side (the independent variables) can take several forms:

- It can consist of a series of names of variables, separated by + signs. Note that since you will be telling the estimation function which dataset the formula applies to, you don't need to use the dollar syntax. For example, for the specification $\mathtt{H1} = \beta_0 + \beta_1\mathtt{V1} + \beta_2\mathtt{V2} + \beta_3\mathtt{V3}$ you would write `H1 ~ V1 + V2 + V3`.
- In general, an intercept/constant (in the above example, β_0) is automatically included. If you don't want this, you can cause it to be omitted by specifying

use. .

[9]There is also a clever syntax for modifying an existing formula (for example, when you want to remove one variable and add another) without having to re-type everything, but personally I find it easier, when working in a script, just to cut-and-paste. See `?update` for more on this modification facility.

`-1` on the right-hand side. Thus `H1 ~ -1 + V1 + V2 + V3` would give you the model $\texttt{H1} = \beta_1\texttt{V1} + \beta_2\texttt{V2} + \beta_3\texttt{V3}$.

- For transformed variables, it is not necessary to construct them explicitly in the dataframe (though you can do so if you wish). You can specify the transformation directly in the formula: for example, suppose you wanted the specification $\texttt{H1} = \beta_0 + \beta_1\texttt{V1} + \beta_2\texttt{V2} + \beta_3 \ln(\texttt{V3})$, where ln is the natural log function. A formula expressing this would be `H1 ~ V1 + V2 + log(V3)`.

  There is one complication. Suppose you wanted to include the sum of `V1` and `V2`. Specifying `H1~V1+V2` wouldn't do what you want — it would include `V1` and `V2` individually, but *not* their sum. You need to "protect" the specification by enclosing it in the `I()` function: thus `H1~I(V1 + V2)`. Protection is not needed when the transformation involves a function, that is, something that takes arguments in parentheses (eg `exp()`). But it is needed for transformations like the power transform: write `I(V1^2)` to include the square of `V1`.

- Finally, note that you can save a formula into a variable (eg, `modelA<-H1 ~ V1 + V2`) and then supply the name of the variable whenever a formula is called for.[10]

5.2 mlogit extensions to formulas

`mlogit` provides some useful extensions to the formula concept, specially adapted to discrete choice models. First, there are extensions which facilitate estimating models with alternative-specific constants.

- Requesting an intercept — which as noted above is the default — results in a full set of alternative-specific constants ("mode specific dummys" for transportation mode choice models) being included.[11]

[10]One slight disadvantage of this is that, as we will see below, the estimation results report the formula you provided. If you provide a variable instead of an explicit formula, then just the name of the variable is reported, which is uninformative. Of course, you can usually deduce the formula from the estimation results.

[11]The package documentation says that to omit the alternative-specific constants you must include `0` or `-1` among the `part_2` variables, and that a `-1` in the `part_1` variables will be ignored. At least as of `mlogit` version 0.2-3 this does not seem to be correct: any of these specifications, including `-1` in `part_1`, will cause the alternative-specific constants to be omitted.

- If there are J alternatives in the choice set, then at most $J - 1$ alternative-specific dummys can be estimated. By default, `mlogit` will delete the *first* dummy (the one corresponding to the first-named alternative, which for the `clogit` dataset is air). If you don't like this — for example because you want your results to be comparable to someone else's — you can tell `mlogit` which dummy to omit via the `reflevel` argument to the estimation function**.** For example, specifying `reflevel="train"` would cause the alternative-specific constant for the "train" alternative to be omitted.

The second set of extensions deal with interactions of variables and variables that do not vary with the alternatives. `mlogit` allows the right-hand side of formulas to be specified in three parts, separated by a vertical bar, which we can represent symbolically as

```
H1 ~ part_1 | part_2 | part_3
```

- `part_1` is the "basic" specification, as discussed above. Typically, variables in this part will vary with the alternative under consideration (for example, in a mode choice model, they might describe a mode's cost (fare) or travel times.
- `part_2` consists of the names of variables whose values do not vary by alternative (individual-specific variables, for example, gender or perhaps age).

  One standard way to get estimable coefficients for individual-specific variables is to "attach" the variable to one alternative, and to fill in zeros for the other alternatives. A variable constructed in that way can be included in the `part_1` list, and you get a single coefficient for the variable. But it is possible to specify that the coefficients vary over the alternatives: this would be done by interacting the variables with the estimable alternative-specific dummys. This is what `part_2` allows you to do, without needing to create the interacted variables directly.

  For example, suppose you have an individual-specific variable `hinc`, the individual's income. If you include this variable among the `part_2` variables, you will estimate the model $\ldots \beta_1(\texttt{hinc} \times \texttt{M}_1) + \beta_2(\texttt{hinc} \times \texttt{M}_2) \ldots$ where $\texttt{M}_i$ is a mode-i dummy, that is, a variable that takes the value 1 for mode i and zero otherwise. In other words, `hinc` gets one weight (β_1) when we are constructing the utility function for mode 1, and another (β_2) in the utility function for mode 2, etc.[12]

[12] Note that this will only work if `hinc` is coded for all alternatives, as it is in this dataset (see the `str` display on p. 10)

- Finally, `part_3` allows you to specify that variables describing characteristics of the *alternatives* have coefficients that differ across alternatives.

  For example, suppose you have a variable `time`, representing on-vehicle travel time. The standard specification, with `time` included in the `part_1` variables, would result in a specification $\cdots + \beta_t \texttt{time} + \ldots$, that is, with all modes' times having the same weight (β_t). On the other hand, if you include `time` among the `part_3` variables, it will result in a specification $\cdots + \beta_1(\texttt{time} \times \texttt{M}_1) + \beta_2(\texttt{time} \times \texttt{M}_2) + \ldots$ where $\texttt{M}_i$ is an mode-*i* dummy. Thus, `time` gets one weight (β_1) in the utility function for mode 1, and another (β_2) for mode 2, etc. In other words, each mode's travel time has a different weight.

6 Estimating logit models

We can now pull all this together, and see how to estimate actual discrete-choice models. The basic idea is to construct the model command in a script; then run the command, saving the result to a variable; and then see what you've got by printing a summary.[13]

6.1 Standard logit

For our first example, suppose we want to estimate a model in which the independent variables are waiting time, generalized cost, plus a full set of alternative-specific dummys. We begin by loading the mlogit package, if it hasn't already been loaded, via `library(mlogit)` :

```
>library(mlogit)
```

We then construct our model, and run it, assigning the result to `res1`.[14]

[13] You can of course do all this from the R console, but I think it is easier to work from a script session.

[14] The reason for assigning the result to a variable is that several post-estimation commands use the special internal structure of the assigned result. If you didn't assign the result, you'd deprive yourelf of the ability to use those commands.

```
>res1<-mlogit(mode~ttme+gc, data=clogit, shape="long",
                alt.var="mode.ids")
```

When we run this line, there is no response from the system, just a new input prompt. To see what we've got, we use the `summary` method on our saved results. In this case what we get is:

```
>summary(res1)

Call:
mlogit(formula = mode ~ttme + gc, data = clogit, shape = "long",
    alt.var = "mode.ids", method = "nr", print.level = 0)

Frequencies of alternatives:
    air   train     bus     car
0.27619 0.30000 0.14286 0.28095

nr method
5 iterations, 0h:0m:1s
g'(-H)^-1g = 0.000221
successive fonction values within tolerance limits

Coefficients :
                    Estimate Std. Error t-value  Pr(>|t|)
train:(intercept) -1.8533538  0.3700925 -5.0078 5.505e-07 ***
bus:(intercept)   -2.5656173  0.3843251 -6.6756 2.461e-11 ***
car:(intercept)   -5.7763487  0.6559187 -8.8065 < 2.2e-16 ***
ttme              -0.0970904  0.0104351 -9.3042 < 2.2e-16 ***
gc                -0.0157837  0.0043828 -3.6013 0.0003166 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-Likelihood: -199.98
McFadden R^2:  0.29526
Likelihood ratio test : chisq = 167.56 (p.value=< 2.22e-16)
```

As we can see, the block beginning `Coefficients` has the usual estimation results: the estimated coefficients themselves, their estimated standard errors, the t-statistics, the probability, under the null hypothesis that the true value of the coefficient is zero, of observing a t-value greater than the computed one; and finally a graphical indication (stars) of the significance level of the coefficient. The line

`McFadden R^2` refers to what we have called McFadden's ρ^2 statistic in class.[15]

Note that we needed to include information on the shape of the dataset, and on the choice set via the `alt.var` argument. This can be cumbersome to type if you are running many models, which is why we created the `mlogit`-specific form of the dataset, `CLOGIT` (in caps), see section 4.5. With this to hand, the estimation command can be abbreviated to:

```
res2<-mlogit(mode~ttme+gc,data=CLOGIT)
summary(res2)
```

which should give exactly the same results.

We now illustrate the `mlogit`-specific extensions to formulas. First, we include income (`hinc`) in the `part_2` variables:

```
>res3<-mlogit(mode~ttme + gc | hinc,  data=CLOGIT)
>summary(res3)

Coefficients :
                   Estimate Std. Error t-value  Pr(>|t|)
train:(intercept) -0.3249576  0.5763335 -0.5638  0.572866
bus:(intercept)   -1.7445354  0.6775004 -2.5750  0.010025 *
car:(intercept)   -5.8747921  0.8020903 -7.3244 2.400e-13 ***
ttme              -0.0954602  0.0104732 -9.1147 < 2.2e-16 ***
gc                -0.0109273  0.0045878 -2.3818  0.017226 *
train:hinc        -0.0511880  0.0147352 -3.4739  0.000513 ***
bus:hinc          -0.0232100  0.0162306 -1.4300  0.152712
car:hinc           0.0053735  0.0115294  0.4661  0.641163
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-Likelihood: -189.53
McFadden R^2:  0.33209
Likelihood ratio test : chisq = 188.47 (p.value=< 2.22e-16)
```

(only partial results shown). We now have income interacted with the three (estimated) mode-choice dummys. Income has a different coefficient for each alternative.

[15]The result of the command (here, `res1`) is a special structure containing results and information about the estimation run; these can be extracted separately (for example, to construct a customized report). You can see what's here by `str(res1)`.

Next, we also make the coefficient of gc (modal generalized cost) vary by mode, by including it in the part_3 area of the formula:

```
>res4<-mlogit(mode~ttme | hinc | gc,  data=CLOGIT)
>summary(res4)

Coefficients :
                    Estimate Std. Error t-value  Pr(>|t|)
train:(intercept)  1.8492221  1.0578547  1.7481 0.0804489 .
bus:(intercept)    0.2332096  1.2150069  0.1919 0.8477885
car:(intercept)   -3.5030068  1.1092306 -3.1581 0.0015883 **
ttme              -0.0962846  0.0104908 -9.1780 < 2.2e-16 ***
train:hinc        -0.0547168  0.0150964 -3.6245 0.0002895 ***
bus:hinc          -0.0249718  0.0163743 -1.5251 0.1272444
car:hinc           0.0034443  0.0119876  0.2873 0.7738634
air:gc             0.0104705  0.0091005  1.1505 0.2499210
train:gc          -0.0088560  0.0050541 -1.7522 0.0797317 .
bus:gc            -0.0070853  0.0074225 -0.9546 0.3397949
car:gc            -0.0110825  0.0056119 -1.9748 0.0482870 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Log-Likelihood: -184.95
McFadden R^2:  0.34822
Likelihood ratio test : chisq = 197.62 (p.value=< 2.22e-16)
```

Again, we show only partial results. Note that the gc variable is now interacted with dummys for *all* the modes: this is possible because gc varies with alternatives, not with individuals. The bottom line is that gc’s coefficient now differs by alternative. It’s also worth noting that the significance levels have changed quite a bit, suggesting that this specification may not be appropriate (at least for this dataset).

6.2 Saving your work

Once you’ve estimated your model, you will probably want to save your results in order to use them elsewhere (for example, in a report). You can do this in a number of ways:

- Perhaps the simplest way is to copy the results (what appears on screen when you run the `summary` command) to the clipboard, and then open a text editor like Notepad or a word-processor document, paste in the clipboard contents, and save the result.

- You can do the same thing under program control (ie as part of a script) by redirecting output to a file instead of to the screen. For example:

  ```
  outfile<-"c:/aaa/res3.txt"
  sink(outfile,append=FALSE)

  cat("Here are my results:\n")
  summary(res3)
  sink()
  ```

  The first line sets up the name of the output file, the next redirects output to that file: the effect of `append=FALSE` is that any existing file `res3.txt` will be over-written: you may wish to change this to `append=TRUE` in appropriate circumstances. The next line prints a bit of explanatory text to the file (the `\n` stands for a newline). Finally we print the summary of our results (`summary(res3)`) and close the file (`sink()`), which returns printing to the screen.[16]

- If you are a LaTeX user, you may wish to explore the `xtable` package, which will write the results of `summary` to a file that LaTeX can process. There is also the `sweave` package, which will enable you to write "interactive" documents, in which R will be run automatically and the results imported into your document each time you use LaTeX to typeset your work.

6.3 Mixed-logit models

One reason for preferring `mlogit` over other R packages that can estimate basic discrete choice models is that it can also estimate the mixed logit model, which is a way of formulating models that do not involve the IIA property. This model considers the coefficients themselves to be random variables ("random parameters model") and then estimates the multidimensional integrals that define the choice probabilities using Monte Carlo simulation. That is, we draw random numbers

[16]You don't have to specify `outfile` separately: it can be included in the `sink` command (`sink("c:/aaa/res3.txt",append=FALSE)`). One reason to do it as shown in the text is when you want to write multiple results to the same file (in which case you'd use `append=TRUE`): you need specify the output file only once (for example at the beginning of your script), and thereafter you can re-open it with `sink(outfile,append=TRUE)`.

from the assumed joint probability distributions of the coefficients, and compute the (conditional) choice probabilities. We repeat this many times, and average the results. This average is an unbiased estimate of the unconditional choice probabilities. In order to carry this out — note that it is going to be *much* more time-consuming than the basic model, so be prepared — you need to tell the estimation routine a few more things:

- You need to tell it which parameters are random, and what distribution(s) to use. You do this via the `rpar` argument. This is the concatenation `c(...)` of comma-separated pairs of the form *var*=`"`*spec*`"` where *var* is the name of a variable in the formula and *spec* is the distributional specification: `n` for normal, `t` for triangular, `ln` for log-normal, `cn` for censored normal, `tn` for truncated normal, or `u` for uniform.[17]

- You need to tell it how to generate draws from the specified distributions. By default pseudo-random numbers are used for the draws; but the literature suggests that it may be more efficient to use quasi-random Halton numbers (more efficient in the sense that you can get by with fewer draws, thus saving both time and space). If you choose this option, you need to include `halton=NA` in the model specification.[18]

- You need to tell it how many draws to use. You do this via the `R=`*n* specification, where *n* is the number of draws.

- By default, the random parameters are assumed to be independent. If you include `correlation=TRUE` in the model specification, you get correlated random parameters. Note that this can add significantly to the number of parameters being estimated. If you allow for correlated random parameters it is important to realize that what is being estimated is *not* the individual variances and covariances, but rather the Choleski decomposition of the covariance matrix. To translate the Choleski matrix into a covariance matrix, do `cov.mlogit(res)`, where `res` is an mlogit results structure.

- If you have a panel data set and you want to take this into account, you need to tell the program that, too: include `panel=TRUE` in the command.[19]

[17]Note that you must still enclose this specification in a `c()` structure even if you are specifying only one random parameter.

[18]The effect of `NA` here is that Halton numbers are constructed based on successive primes starting with 3. It is possible to modify this.

[19]If you create the special `mlogit` form of the dataset as in section 4.5 you will have had to specify which variable is the panel indicator (via `id.var="`*xxx*`"`).

Here is an example. We suppose that the alternative-specific constants for air, bus and train have independent (uncorrelated) normal distributions, and that we want to use 500 Halton draws in order to estimate the choice probability integrals. Because `mlogit` would normally omit the air dummy, we tell it to omit the car dummy instead. We also request some feedback on the estimation progress by setting `print.level` to 1. The model specification would be:[20,21,22]

```
> res5<-mlogit(mode~ttme+gc,data=CLOGIT,reflevel="car",
+       rpar=c("air:(intercept)"="n","bus:(intercept)"="n",
               "train:(intercept)"="n"),
+       R=500,halton=NA,print.level=1)
> summary(res5)
```

and here is the output (we omit the feedback output):

```
Call:
mlogit(formula = mode ~ttme + gc, data = CLOGIT, reflevel = "car",
    rpar = c("air:(intercept)" = "n", "bus:(intercept" = "n",
        "train:(intercept)" = "n"), R = 500, halton = NA, print.level=1)

Frequencies of alternatives:
    car     air   train     bus
0.28095 0.27619 0.30000 0.14286

bfgs method
15 iterations, 0h:0m:38s
```

[20]The specification of random parameters for the alternative-specific constants has changed from `mlogit` version 0.1-8. The old version had `rpar=c(altair='n',altbus='n',alttrain='n')`. If you get an error here, try estimating model without random parameters (like model `res4` above), and note how the mode-specific dummys are reported; then use that syntax in the `rpar` argument.

[21]The same idea as in the previous footnote applies to logical variables. If `var1` is a variable taking the values `TRUE` or `FALSE`, then including it among the independent variables in a standard logit model will result in a coefficient estimate for the variable `var1TRUE`. If you want its coefficient to be normally distributed, you would specify this as `var1TRUE='n'` (and not as `var1='n'`). The error message if you forget this will involve a phrase like `'names' attribute [1] must be the same length as the vector [0]`, which is attempting to tell you that plain `var1` is not in fact included among the independent variables. Similarly for factor variables: the bottom line is that variables named in the `rpar` list need to match the way they appear in a non-mixed-logit model.

[22]Note on quoting variable names in the `rpar` specification: you do not usually need to quote the variable names (but you can if you wish). However, if a variable name contains parentheses (as the names of the alternative-specific constants now always do) then it *must* be quoted, as in the example. Either single or double quotes will work.

```
g'(-H)^-1g = 8.22E-07
gradient close to zero

Coefficients :
                       Estimate Std. Error t-value  Pr(>|t|)
air:(intercept)       6.1592355  1.1386852  5.4091 6.335e-08 ***
train:(intercept)     5.1095985  0.8174536  6.2506 4.088e-10 ***
bus:(intercept)       4.1487347  0.9042209  4.5882 4.471e-06 ***
ttme                 -0.1144355  0.0199023 -5.7499 8.932e-09 ***
gc                   -0.0308410  0.0077524 -3.9782 6.943e-05 ***
sd.air:(intercept)    2.9380453  0.9163970  3.2061  0.001346 **
sd.train:(intercept)  0.0382613 21.4589826  0.0018  0.998577
sd.bus:(intercept)    0.0016575 76.6920349  0.0000  0.999983
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-Likelihood: -197.47
McFadden R^2:  0.3041
Likelihood ratio test : chisq = 172.58 (p.value = < 2.22e-16)

random coefficients
                  Min.  1st Qu.   Median     Mean  3rd Qu. Max.
air:(intercept)   -Inf 4.177554 6.159236 6.159236 8.140917  Inf
bus:(intercept)   -Inf 4.147617 4.148735 4.148735 4.149853  Inf
train:(intercept) -Inf 5.083792 5.109599 5.109599 5.135405  Inf
```

Note that in the case of the random parameters, the "Estimates" are in fact the estimates of the means of the (here, normal) distributions, and not of the individual coefficients (βs). This is made clearer in the "random coefficients" block of results. The estimates denoted `sd.air:(intercept)` (etc) represent (in this case) the estimated standard deviations of the random coefficients: in this example we estimate that the air dummy is normal with mean 6.1436 and variance $2.9768^2 = 8.8613$, ie $N(6.1436, 8.8613)$.[23] Note that the results here strongly suggest that the train and bus dummys are *not* in fact random (because we cannot reject the null hypothesis of zero standard deviations).

The interpretation of the random parameters as means and standard deviations is specific to the normal distribution: if for example we had specified that the air dummy had a uniform distribution, we would be estimating the mean of the distribution, plus its "spread", namely how far from the mean the distribu-

[23]Note that the estimates for these dummys in the "coefficients" block of results are the same as the "means" in the "random coefficients" block of results.

tion extends in either direction. However, the second parameter would still be listed as “`sd`” whether or not it was a standard deviation. You can get a description of what the random parameters actually mean by doing, eg, `rpar(res5)` (where the argument is the name of a saved mixed-logit model result), which will list the information for all the random parameters, or, if you just need to know about one of them, `rpar(res5,"air:(intercept)")`. The result is one line per parameter, so there’s usually little to be gained by not asking for them all. If you need to, you can extract a line of the “random coefficients” block via, eg, `summary(rpar(res5,"air:(intercept)"))`, which can of course be saved to a variable for later use.

If you try to include an individual-specific variable (like `hinc` in this dataset) in the `rpar` list (for example, `rpar=c(hinc"="n"))` , you will get an error. This is understandable, since you are actually creating $J-1$ new variables in the specification, and it is unclear how you want all of them to be treated. The solution is to name the variables whose coefficients you wish to make random *exactly* as they appear in the listing for a non-random-parameters model. In this case, an example would be: `rpar=c("train:hinc="n")`. Note that because of the separator, the variable name must be quoted.

You should also be aware of the notation used for estimated coefficients in mixed-logit models with correlated random parameters. A coefficient name like `V1:V2` is the coefficient of variable `V1` interacted with variable `V2`. But if you specify that these two variables have correlated random parameters, you will also see results in the form `V1.V2`, ie with a period (dot) between the two names, rather than a colon. This notation is used for the row-`V1`, column-`V2` element of the Choleski decomposition of the covariance matrix.

6.4 Choice-based sampling

As we have noted, the `clogit` dataset is not a random sample, but choice-based.[24] Choice-based sampling may be appropriate when we wish to over-sample some lightly used alternatives in order to ensure that their characteristics are represented in the dataset, or because they are simpler to collect.

However, this makes the likelihood function wrong (since it is based on the

[24]The name arises because instead of picking out people at random (say from the phone book) we sample them by surveying them at their chosen modes (say, on the bus, or at the parking lot). Thus our sample depends on the choices they actually made.

assumption of random sampling). The fix is to correct the likelihood function by weighting the individual observations. Let π_i be the sample proportion using mode i, and let π_i^* be the population proportion using this mode. Then each observation relating to mode i must be weighted by $w_i = \pi_i^* / \pi_i$.[25]

Update, December 2014: Contrary to what earlier editions of this guide said, it seems that `mlogit` cannot handle choice-based-sampling. The `weights` parameter appears to do something quite different, namely allow you to weight the individual observations (as opposed to the choices). (It does the same thing as `weights` in the standard linear-regression routine `lm`). It would probably be quite easy to modify the code to handle choice-based sampling, but I've not had occasion to work on this. I did send a query about this to Yves Croissant, but have received no response. Note that NLOGIT can handle this.

6.5 Other logit models

`mlogit` can also estimate several other logit-like models, including:

- The heteroskedastic extreme-value model (HEV). This is a generalization of standard logit that relaxes the "identical distribution" assumption governing the random terms. This results in cross-elasticities that are different for different alternatives, and can be considered another way of relaxing the IIA property. This model must also be estimated by simulation.
- The nested logit model. This is another way of relaxing the IIA property. We partition the choice set into "nests" (groups of alternatives) such that IIA holds between alternatives in a nest, but does not hold across nests. This model does not require simulation for estimation. One disadvantage of nested logit over the HEV and mixed logit models is that we need to specify the grouping of the alternatives *a priori* rather than allowing the data to tell us what's going on. Of course, you can experiment with various specifications (groupings), and see which works best.[26]

[25]Note that the π_i^* require external information. For `clogit` the π_i^* are: air = 0.14 ; train = 0.13 ; bus = 0.09 ; and car = 0.64. Compare these with the sample proportions as reported in the estimation results. For the clogit dataset, the weights turn out to be: air = 0.506897; train = 0.433333; bus = 0.629987; car = 2.27799. But as mentioned in the text, the package cannot as of now make use of this.

[26]One standard way to do this is to choose the nesting pattern that results in the highest value of the maximized log-likelihood function.

See the `mlogit` help files for details of how to estimate these models.

- Much of the non-transportation literature refers to the model we have concentrated on here as the "conditional logit model", and reserves the name "logit model" for a formulation in which the independent variables do not vary over alternatives (are individual-specific). In this model, we have

  $$P_{ij} = \frac{e^{x_i \beta_j}}{\sum_m e^{x_i \beta_m}}$$

  where now the *coefficients* (βs) vary over the alternatives.[27] Standard example: an individual's choice of an occupation. Here the alternatives are, for example, soldier, teacher, politician; and the independent variables describe, not the alternatives, but the characteristics of the decision-maker: gender, college GPA, years served in prison, etc.

  This version of the model can be estimated using `mlogit` by including the individual-specific variables in what we called above the `part_2` variables, and omitting (except perhaps for alternative-specific constants) any `part_1` variables.

A The script

Here is a complete listing of the script I used to do the work described here.

```
# script for mlogit document. Note that the file locations refer to
# my hard disk; and you will need to alter these for your use

# this works in mlogit 0.2-2 and involves a change of syntax for the
# mxl, where the alt specific dummys are random. See model 5 below.

# read in the data
library(foreign)
clogit <- read.table("c:/work/clogit/clogit.dat",
                     col.names=c("mode","ttme","invc","invt","gc","chair","hinc",
                       "psize","indj","indi","aasc","tasc","basc","casc",
                       "hinca","psizea","z","nij","ni"),na.strings="-999")
```

[27] For identification, we must normalize one of the β_j, usually to zero.

```
# check the data
summary(clogit)
str(clogit)
head(clogit)


# save a version of the data in internal format
save(clogit,file="c:/work/clogit/clogit.rdata")

# read in the mlogit package
library(mlogit)

# read in the data we saved
load(file="c:/work/clogit/clogit.rdata")

# provide a choice indicator, with names
clogit$mode.ids<-factor(rep(1:4,210),labels=c("air","train","bus","car"))

# for convenience, create a special form of the dataset:
# note that we exploit the case sensititivty here: clogit is the original
# dataset, while CLOGIT (all-caps) is the version for use with the mlogit package.
CLOGIT<-mlogit.data(clogit,shape="long",choice="mode",alt.var="mode.ids")


# first model : standard logit. We save the results into a variable
# and then view them. The first command uses the un-fixed dataset, while the
# second uses the mlogit-specific dataset, and is clearly easier to type in.
# Both produce the same output.
res1<-mlogit(mode~ttme+gc, data=clogit, shape="long",
                alt.var="mode.ids")
summary(res1)

res2<-mlogit(mode~ttme+gc,data=CLOGIT)
summary(res2)

# model with income interacted with the mode-specific dummys
res3<-mlogit(mode~ttme+gc | hinc, data=CLOGIT)
summary(res3)

 # model with gc varying by mode
res4<-mlogit(mode~ttme | hinc | gc, data=CLOGIT)
summary(res4)

# mixed logit model  in which the alt-specific vars
# have independent normal distributions. We use Halton numbers for
# efficiency in computation, and use R=500 in our simulations

# note that the syntax for specifying random alt-specific dummys has changed.
# we set print.level to 1 to get some feedback
res5<-mlogit(mode~ttme+gc,data=CLOGIT,reflevel="car",
      rpar=c("air:(intercept)"="n","bus:(intercept)"="n",
      "train:(intercept)"="n"),R=500,halton=NA,print.level=1)
summary(res5)

# print info on what was estimated for a random parameter
rpar(res5,"air:(intercept)")

# same model, but multinomial probit (not discussed in the text)
res6<-mlogit(mode~ttme+gc,data=CLOGIT,reflevel="car",
             R=500,halton=NA,probit=TRUE,print.level=1)
summary(res6)
# Omega-sub-i for each mode:
res6$omega
```

B The clogit data

Here is a description of the data in the `clogit` dataset : as already noted, this is a choice-based survey of the intercity mode choices of 210 Australians, gathered by David Hensher. There are 4 modes, in order: air, train, bus, car. Each individual is represented by 4 rows of data, one row for each mode. (This is the so-called “long” form of the data).

Variable	**Description**
`mode`	0/1 variable, with a 1 indicating the position of the mode selected.
`ttme`	terminal waiting time, minutes
`invc`	in-vehicle cost, minutes
`invt`	in-vehicle time, minutes
`gc`	generalized cost = `invc + (invt` × `value-of-time)`
`chair`	dummy, = 1 if chosen mode is air
`hinc`	household income, in thousands of $AUS
`psize`	travelling party size
`indj`	−2 for not-air ; otherwise the same as `mode`
`indi`	dummy for air / not-air : −1 for the other modes
`aasc`	alternative-specific dummy for air mode
`tasc`	alternative-specific dummy for train mode
`basc`	alternative-specific dummy for bus mode
`casc`	alternative-specific dummy for car mode
`psizea`	`psize` × `aasc`
`hinca`	household income, “attached” to the air mode
`z`	`tasc + basc + basc` = Dummy variable for Not Air
`nij`	1 if `aasc` = 1 and 3 if `aasc` = 0
`ni`	2 = number of branches in tree

Some of these variables (eg `nij`, `ni`) are pre-constructed variables that were used in the estimation of a nested logit model in Limdep. In R we do not need the alternative-specific variables (`aasc` etc): we could remove them from the dataframe by commands like `clogit$aasc<-NULL` .

The full dataset is available on the C&RP 5700 website in a number of forms:

- Plain text: this zip file contains two versions: `clogit.dat` which has just the data (corresponds to the discussion in the text) and `clogit-hdr.dat`, which is the same data, but this time with a header line, so you don’t have to name the variables yourself.
- Comma-separated values (with header line).
- Excel 2003 file

C Binary logit

Most packages estimate the binary logit model with a dataset that has one row per individual: this is clearly space-efficient, but it will not work for `mlogit`. If you have such a dataset and you want to estimate only a "standard" binary logit model, it would probably be easier to use another package (eg, `glm`). But if you want to estimate a binary mixed-logit model, then `mlogit` may be your only option. In this case you will need to convert your data to a long-form dataset with 2 rows per individual: here is a sketch of how to do it, using the R package `gdata` (which you will need to install yourself, since it is not standard).

We assume that `old` is a dataframe in 1-line-per-individual format with no missing data.[28] `new` will be a long-form (2 lines per individual) dataframe. Begin by focusing on any purely identification variables in `old`: you will want the data in these columns to be the same for both rows for each individual of the new dataframe (these will be variables that you will not be using as part_1 independent variables, since they will be the same for both alternatives). A panel indicator is of this type, if your dataset is a panel dataset. You should also include variables for which a numeric 0 isn't valid (eg date or time variables). We interleave this with itself.

```
zcols<-c("state_name","indiv_name","indiv_address")
new<-gdata::interleave(old[,zcols],old[,zcols])
```

Next, pick the rest of the data, and interleave it with a conformable matrix of zeros (note the order in which we do this):[29]

```
zzcols<-setdiff(names(old),zcols)
nnew<-gdata::interleave(0*old[,zzcols],old[,zzcols])
```

Put the two parts together:

[28] If there is missing data, then an alternative strategy is: (1) create a dataframe A1 with any character data from the original dataset, and A2 as a copy of A1. Then create a dataframe B1 with all the numeric data from the original dataset. Create B2 as B1 multiplied by 0; this will preserve `NA`s. Create AA by interleaving A1 and A2, and BB by interleaving B1 and B2. Finally create the new dataset by `new<-cbind(AA,BB)`. Now fix up the choice variable, as in the main text. You will need to add a line to the effect `if (is.na(old[r]) next` before `if (old[r] ...` in order to handle any missing data in the choice variable

[29] You could also select columns numerically, so that for example `zcols<-c(1,5,10,11)`. Then `zzcols` would be defined by `zzcols<-setdiff(1:dim(old)[2],zcols)`

```
new<-cbind(new,nnew)
```

Finally, fix up the choice variable, which we assume is named "choice". If the model is $\Pr[y = 1] = x'\beta$ then we want the two rows to be (1,0) when the original choice indicator was 0, and (0,1) when it was 1. Remembering that R indices start from 1, the following code does that:

```
z<-unlist(list(c(0,1),c(1,0))[1+new$choice])
new$choice<-z
```

At this point, `new` is a long-form dataframe for binary choice that is usable by `mlogit`. You may want to estimate a standard logit model using mlogit and comparing it to what you get with a 1-line-per-individual dataset with another logit package, just to be sure.

Subsetting: remember that most columns of `new` have zeros in each individual's first row. Therefore, selecting on such a variable will typically pick out only the second data row for each person, which is not what we want. But this is easy to fix: we check the condition, pull out all the even-numbered elements of the result and then replicate them:

```
sub<-new$myvar==1975
sub<-rep(sub[seq(from=2,to=length(sub),by=2)],each=2]
```

At this point `new[sub,]` should give us the subset we want.

D Large-sample panel-data mixed logit

The essential difference between mixed-logit in a panel-data (as against a cross-sectional) setting is that with panel data the individual choice probabilities involve a *product* of logits, one logit for each choice setting that the panel member is observed to encounter.

Ordinarily this will not be a problem, but there is one situation where an issue can arise. Suppose you observe an individual in *many* choice settings, where "many" could mean many hundreds. In most economic datasets this is probably unlikely, but consider a political science application where one is studying the votes of individual judges on an appellate court, where a judge could easily be represented by over a thousand votes, depending on how long he or she has been on the bench. Now suppose that at the current iteration of the maximization algorithm, the computed choice probabilities for this panel member are low. Then the contribution to the likelihood will involve a low probability raised to a large power. This can cause an underflow in R's floating-point arithmetic, with the result that the panel-data choice probability for this panel member will be reported as zero, and the log of zero (needed for the log-likelihood) is minus-infinity.[30]

When this happens, R (or at least optimizer used by `mlogit`) does not issue a warning about an underflow. Instead, it reports an opaque error:

```
Error in if (abs(x - oldx) < ftol) { :
 missing value where TRUE/FALSE needed
```

and stops. The natural tendency at this point is to take seriously the reference to a "missing value" and think that there must be something wrong with the data or perhaps the model specification; but that is not necessarily so. You can check whether the problem is underflow at the initial point by re-running the mixed-logit model with `iterlim=1` and `print.level=1`. This will run 1 iteration of the maximum simulated likelihood procedure, and if underflow is a problem you will see this in the reported likelihood value.[31] In my limited experience, if there are no problems with the likelihood at the initial point, the optimization makes it unlikely that there will be problems at later trials.

As far as I can tell, when this problem occurs, your only option is to-re-run the model on a (random) sample of the original data, such that the underflow does not occur. Finding a suitable sample size will probably involve a certain amount of trial-and-error.[32]

[30]For a concrete example (32-bit R 3.0.2 for MS Windows) suppose that the average logit for the chosen alternative for a panel member is 0.4 and that the number of choice settings is 814. R reports the product 0.4^{814} as zero.

[31]Because the optimizer used by `mlogit` minimizes, the report will that the function value is plus-infinity rather than minus-infinity.

[32]This actually happened to me once, and I spent the best part of a week single-stepping through `mlogit` code trying to figure out what was wrong with my data.

E Other statistical functions in R

Here is a brief list of some other regression functions implemented in R.

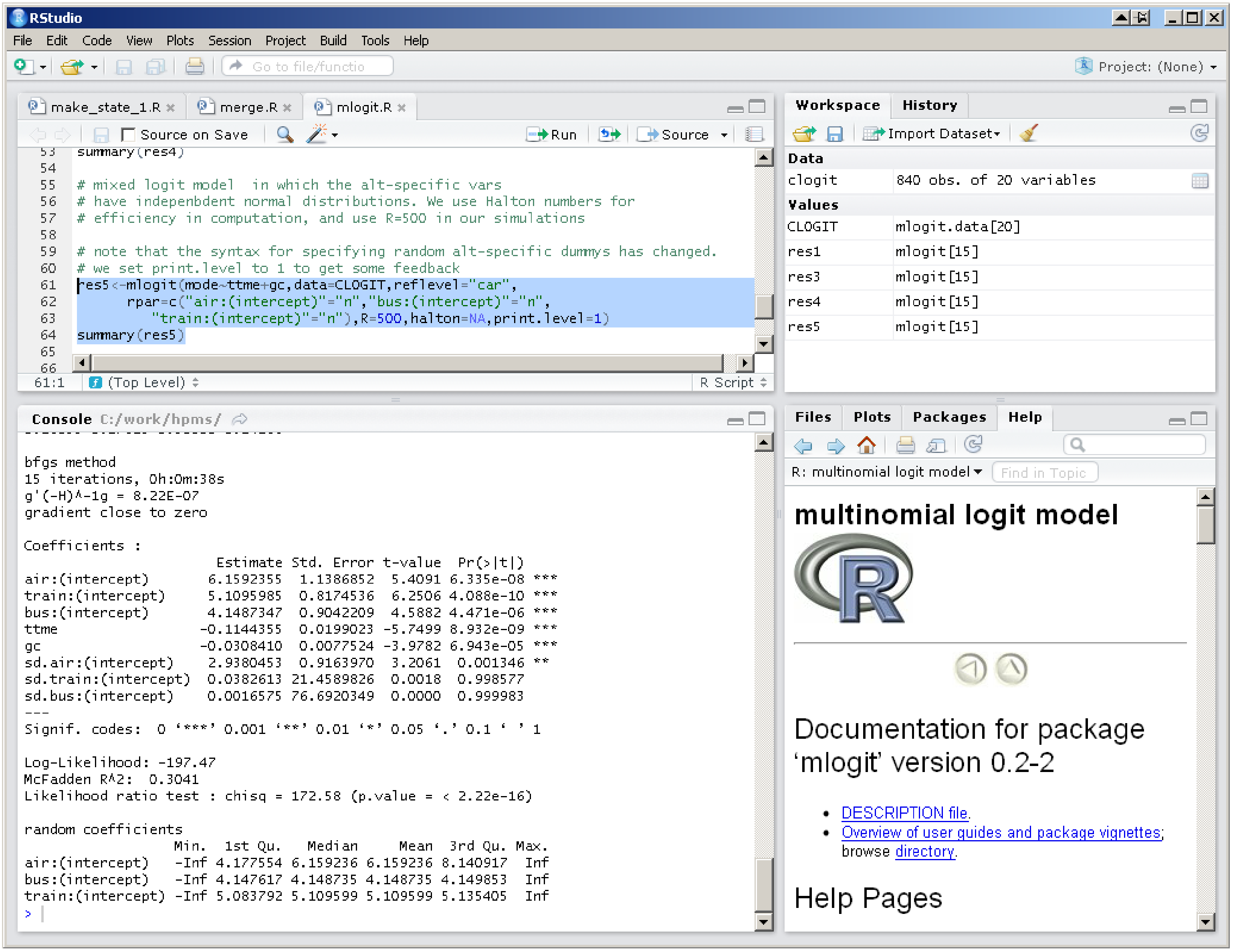
Model	**Package**	**Function**
Linear cross-sectional regression	(default)	`lm()`
Panel-data linear regression	`plm`	`plm()`
Simultaneous equation models, including SUR	`systemfit`	`systemfit()`
Generalized linear models	`glm`	`glm()`
Binary logit	`glm`	`glm` with `link="logit"`
Binary probit	`glm`	`glm` with `link="probit"`
Spatial cross-sectional regression[33]	`spdep`	(various)
Spatial panel data regression	`splm`	(various)
Bayesian spatial probit	`spatialprobit`	`sar_probit_mcmc()`
Spatial logit or probit via GMM	`McSpatial`	`splogit(), spprobit()`

Note: the `spatialprobit` package is an emulation in R of James LeSage's MATLAB function. However, it appears to be much faster, due to clever use of sparse-data techniques: for example, a model with 10,000 locations estimates quite quickly. See Stefan Wilhelm and Miguel Godinho de Matos. "Estimating spatial probit models in R". *The R Journal*, 5 (1) : 130–143, June 2013 for a discussion which also contains R code for setting up spatial probit sampling experiments in several contexts.

F RStudio

The next page has a picture of an alternative GUI, RStudio (running on Windows, though it is also available for Linux and the Mac). Download it from http://rstudio.org/download/desktop.

[33] You can also do spatial cross-sectional regression using the `splm` package, and though it's a bit more complicated, it may be worth considering in the interests of not having to learn too many packages.



```
summary(res4)

# mixed logit model  in which the alt-specific vars
# have indepenbdent normal distributions. We use Halton numbers for
# efficiency in computation, and use R=500 in our simulations

# note that the syntax for specifying random alt-specific dummys has changed.
# we set print.level to 1 to get some feedback
res5<-mlogit(mode~ttme+gc,data=CLOGIT,reflevel="car",
      rpar=c("air:(intercept)"="n","bus:(intercept)"="n",
         "train:(intercept)"="n"),R=500,halton=NA,print.level=1)
summary(res5)
```

Data

clogit	840 obs. of 20 variables

Values

CLOGIT	mlogit.data[20]
res1	mlogit[15]
res3	mlogit[15]
res4	mlogit[15]
res5	mlogit[15]

```
bfgs method
15 iterations, 0h:0m:38s
g'(-H)^-1g = 8.22E-07
gradient close to zero

Coefficients :
                       Estimate Std. Error t-value  Pr(>|t|)
air:(intercept)       6.1592355  1.1386852  5.4091 6.335e-08 ***
train:(intercept)     5.1095985  0.8174536  6.2506 4.088e-10 ***
bus:(intercept)       4.1487347  0.9042209  4.5882 4.471e-06 ***
ttme                 -0.1144355  0.0199023 -5.7499 8.932e-09 ***
gc                   -0.0308410  0.0077524 -3.9782 6.943e-05 ***
sd.air:(intercept)    2.9380453  0.9163970  3.2061  0.001346 **
sd.train:(intercept)  0.0382613 21.4589826  0.0018  0.998577
sd.bus:(intercept)    0.0016575 76.6920349  0.0000  0.999983
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Log-Likelihood: -197.47
McFadden R^2:  0.3041
Likelihood ratio test : chisq = 172.58 (p.value = < 2.22e-16)

random coefficients
                  Min.  1st Qu.   Median     Mean  3rd Qu. Max.
air:(intercept)   -Inf 4.177554 6.159236 6.159236 8.140917  Inf
bus:(intercept)   -Inf 4.147617 4.148735 4.148735 4.149853  Inf
train:(intercept) -Inf 5.083792 5.109599 5.109599 5.135405  Inf
>
```

multinomial logit model

Documentation for package ‘mlogit’ version 0.2-2

- DESCRIPTION file.
- Overview of user guides and package vignettes; browse directory.

Help Pages

From the top-left, clockwise:

- A script window. Note the syntax highlighting. In addition, RStudio features syntax completion: if you type, for instance, an opening parenthesis, the program actually gives you a matched set, and all you need to do is type in whatever belongs inside. (You can turn this off if you don't like it). There are provisions for multiple scripts, each selected via a tab. Like the standard R GUI, you can select portions of the script, and then send them to the R system for execution.
- A data window. The data objects are listed: note that they include some of the variables storing estimation results. Clicking on the little icon to the right of the `clogit` dataframe allows you to edit the dataframe. Though R also has a small editor, RStudio's is much simpler to use.
- A help window, open to the main `mlogit` help page. This is also somewhat more convenient than R's help system, which opens up in a browser window; in my experience this takes more time to start than I'd like. If you click on the Packages tab you will get a list of all packages currently installed in your system.
- A console window, showing the results of executing a script command. This is very much like the console window in the standard R GUI. One problem is that (unlike standard R) you cannot print the contents of the console. All you can do is copy the contents to the clipboard, paste it into a text file (also available in RStudio) and print the contents of the text file.

G Further reading

Here are some books that can help you get started with R. They are in the Springer "Use R!" series, and if you are an OSU student working from an OSU computer you should be able to read and/or save them online: see `http://www.springer.com/series/6991?detailsPage=titles`.

- Phil Spector. *Data Manipulation With R*. Use R! Springer, Secaucus, N.J., 2007.
- Christian Kleiber and Achim Zeileis. *Applied Econometrics with R*. Use R! Springer, Secaucus, N.J., 2008

In addition, there are a number of excellent free introductions to R available from the R website: for example

- "An Introduction to R" by W. N. Venables and D. M. Smith
- "icebreakeR" by Andrew Robinson
- "An Introduction to R: Software for Statistical Modelling and Computing" by Petra Kuhnert and Bill Venables

All these cover R's graphics capabilities, which have been bypassed here. Josh Mills at Purdue University has written two superb summaries: "Estimation of Statistical Models in R" and "Special Topics in Estimating Statistical Models in R". He tends to prefer to use the `zelig` package as a consistent interface to estimation. Do a Google search to find his write-ups.

Finally, remember that there is a wonderful book on simulation techniques in discrete choice analysis, which is available in a free download:

- Kenneth E. Train. *Discrete Choice Methods with Simulation*. Cambridge University Press, Cambridge, UK, 2002, available at `http://elsa.berkeley.edu/books/choice2.html`.