Appendix: Reading and Saving Data

5.1 Importing External Data

Although it is possible to type data directly into MATLAB, as has been done in most of the examples in this book, this is rarely the most effective way to operate with real data sets except when they are very small. In general, you will either be using data that has been collected by other people and is in a format of their choosing, or you will be using data tabulated by yourself in a format of your own choosing. We term such data "external" since it is typically formatted outside of MATLAB, perhaps using a spreadsheet or database program. This section describes how to read such data into MATLAB.

In all cases, we will assume that the external data are already stored in one or more computer files. These files might be in a format for a specific program (for example, an Excel or Lotus or some other spreadsheet or some database program).

To illustrate, we we assume that you have a file containing the following experimental information pertaining to a fictional antibiotic effectiveness experiment, perhaps stored in a spreadsheet program. (If you are using a word processor to create tables of data, make sure to put tabs or commas rather than spaces between data entries and to save your work as an ASCII file. Alternatively, consider switching to a spreadsheet that allows you to save your work as a comma or tab separated ascii file.)

Placebo			Antibiotic		
Age	Gender	Cell Count	Age	Gender	Cell Count
32	M	234	27	F	64
45	M	73	19	M	124
23	F	97	34	F	204

```
    F      103              41      M      51
                            26      M      62
```

This is an unrealistically small data set, however it serves to illustrate how one might import a larger data set. (We use this example to show how one might deal with a difficult case. The format is not ideal; see the end of this section for a better suggested format for this type of data.)

Note that the data fall into two groups: 5 people were given the antibiotic and 4 people were given an placebo. The same three measurements, age, sex, and a count of cells of a certain type were made in all 9 subjects. For the 4th placebo-group subject, the age is missing.

These are the basic principles you should keep in mind when importing data into MATLAB:

- MATLAB deals best with purely numerical data. This means that when you have non-numerical categorical data (for example, M for male, F for female) it is best to convert the data to numerical codes (for example 0 for male and 1 for female). When converting from the categorical data to the numerical codes, you will need to use the facilities of some external program, such as Excel, Lotus, or a database program.

  Numbers can contain decimal points, but not commas. For example, 83932232.23 is a valid number, but $83,932,232.23$ is not. Numbers with commas will confuse MATLAB.

- MATLAB deals best with vectors or matrices of numbers. A vector is a single column or row of numbers, a matrix is a rectangular array of numbers. If you have more than one vector in a data set, and the vectors are not all the same length, you will need to use a separate computer files, one file for each group of vectors of the same length. For example, in the antibiotic data given above, there are all together six vectors of data, which can be grouped into two matrices, one for the placebo group (matrix size, 4×3) and one for the antibiotic group (matrix size, 5×3). Each of these matrices needs to be stored in a separate data file before reading it into MATLAB.

- Non-numerical data will confuse MATLAB. Depending on how the data are read into MATLAB, it will convert non-numerical data to zeros, or will fail to read the data file at all. So, you should delete titles and column headings (e.g., “Placebo”, “Antibiotic”, “Age”, “Gender”, “Cell Count” in the above example) before reading the data into MATLAB. However, titles and column headings can be extremely important in managing your raw data files, so you will probably want to make a copy of your raw data files, delete the titles and headings from the copy, and read the copy into MATLAB.

- Missing data is best represented as the three letter sequence `NaN` with the correct capitalization. Other forms of representing missing data (e.g., blank spaces, non-numeric codes such as `.`, and so on) will often be turned into `0` by MATLAB. If you choose, you can represent missing data in your raw data files with a numerical code, such as `-9999`, however doing so runs the risk that your numerical code might happen to match an actual data value.
- Once you have read the data into MATLAB, it is easy to store it in a format that MATLAB will read easily for future processing.
- If you maintain a master set of data files, and then process copies of these for reading into MATLAB, be aware that changes or edits to your master files will *not* automatically appear in the MATLAB files. You will, of course, have to re-process the master files to incorporate any changes in them.

Keeping these principles in mind, we will work through the steps of reading a data file, like that in the antibiotic example, into MATLAB. (The data file is stored in text form in the `resamp/examples` directory in the file `dataexample.txt` and in Excel spreadsheet form in `dataexample.xls`.) We will work from the spreadsheet file, as data sets are often collected in such spreadsheets.

1. Convert the non-numeric categorical codes into numerical codes. We converted M to 0 and F to 1. (This was done in the spreadsheet program using the "Replace" menu command.) Make sure to write down the values of the numerical codes for later reference.
2. The contents of the missing data cells were replaced with the letters `NaN`.
3. Each of the rectangular arrays of data, the 4×3 array for the placebo data and the 5×3 array for the antibiotic data, were separately copied to its own worksheet. Only the numerical data and not the titles and column headings were copied. (The resulting Excel spreadsheet is stored in `dataexample2.xls`.) These worksheets were then separately saved to files. Either the "comma delimited" (CSV) or the "text (tab delimited)" file formats work well.

(The saved files are in `placebo.csv` and `antibiotic.csv` for the comma-separated format, and in `placebo.txt` and `antibiotic.txt` for the tab-delimited format. Of course, you will only use one format or the other, depending on what is most convenient given your spreadsheet software.)

4. From the MATLAB command window, read in the files you have just saved, giving each an appropriate variable name. For the comma-delimited format, the commands are[1]

```
≫  antibiotic = ...
        csvread('c:/resamp/examples/antibiotic.csv')
  antibiotic   27 1 64
               19 0 124
               34 1 204
               41 0 51
               26 0 62
≫  placebo = csvread('c:/resamp/examples/placebo.csv')

  placebo  32 0 234
           45 0 73
           23 1 97
           NaN 1 103
```

   For the tab-delimited format, the respective commands would be

```
≫  antibiotic = load('antibiotic.txt');
≫  placebo = load('placebo.txt');
```

   If you need to use some delimiter other than a tab or a comma, you can use `dlmread`, which allows you to specify the delimiter as in

```
≫  m = dlmread('antibiotic.txt', ';')
```

   Of course you will need to replace the name `'antibiotic.txt'` with the name of the folder in which the data are contained, as well as the data files name. For instance,
   `'c:/resamp/examples/antibiotic.txt'`.

5. Now the data are read into MATLAB. If you used a numerical

[1]Use the name of the directory where MATLAB is installed, which we are assuming is `c:/resamp/examples`. If you are not sure where this directory is, type the command

```
≫  which birthday
```

which will look for the directory that contains the example `birthday`.

code for missing data, and not NaN, then use RECODE to convert the missing data code to NaN.

If your data was a single-column or single-row vector, you are now finished. If your data are in matrix form, you may prefer to give each column of the matrix a different name, but this is optional. To do this, use the SEPMATRIX command:

```
≫   sepmatrix placebo agep sexp countp
≫   sepmatrix antibiotic agea sexa counta
```

Note that `sepmatrix` takes arguments that are *not* in parentheses. The first argument is the name of the matrix to be divided into columns. The remaining arguments are the names of the variables to be used. If you re-use a variable name, the old data will be overwritten. Avoid using names such as `count` that are the same as the name of a function.

Optional Once you have gone through the labor of importing external data, you may want to save the data in MATLAB format so that it can be easily read into future sessions of MATLAB. See Sec. **??**.

For the placebo vs. antibiotic data, a more natural format for the raw data files might be as a single matrix

Type	Age	Gender	Cell Count
P	32	M	234
P	45	M	73
P	23	F	97
P		F	103
A	27	F	64
A	19	M	124
A	34	F	204
A	41	M	51
A	26	M	62

The information about whether each person recieved a placebo or antibiotic is coded as a P or A in the first column. This has two advantages: it is somewhat easier to read into MATLAB since only one matrix is involved (although the P and A will need to be converted to numerical codes); for resampling operations involving the null hypothesis that the antibiotic is no different from the placebo, all the data for both types is already in a single vector.

Once the categorical data has been converted to numerical codes, with say 0 for P and 1 for A, and that each column has been saved as a variable (`type`, `age`, `sex`, `cnt` — note that we didn't use `count` because there is a function of this name), we can easily pull out the placebo and antibiotic data for separate processing. For example,

```
≫  cnta = cnt(type==1);
≫  cntp = cnt(type==0);
≫  mean(cnta) ⇒ ans:  101
≫  mean(cntp) ⇒ ans:  126.75
```

5.2 Saving and Reading MATLAB Variables for Internal Use

Once you have gone through the work of reading external data into MATLAB, or have generated new results using MATLAB, you may want to save the variables you have created in a format that is easily re-read into MATLAB. This will allow you to start up a new MATLAB session with variables you created in a previous session with little work and a minimal possibility of error.

The easiest way to save your variables is to use the FILE/SAVE SESSION menu item from the MATLAB menu bar This saves all the variables in the session. (See Steps **??** and **??** of Appendix **??**.)

If you want to save just some of your variables, you can use the `save` and `load` commands on the MATLAB prompt line.

The `save` and `load` commands are used for this purpose.

To save a set of variables, use `save`

```
≫  save myfile x y z ...
```

where `myfile` is the name of the file that will hold the data and `x`, `y`, `z` and so on are the names of the variables to be saved. You can use `*` as a wildcard character to indicate one or more variables of the given pattern. For example,

```
≫  save myfile *
```

Any previous data stored in `myfile` will be deleted. If you want to add new variables to `myfile`, use `-append` as the last argument to `save`, as in

```
≫  save myfile newx newy -append
```

To load in the data from `myfile`, simply give the command

```
≫  load myfile
```

This will load all of the variables with the names used when they were saved.

For MATLAB experts: If you have a complicated data format to read in, consider writing a special-purpose program using the built-in MATLAB commands `fscanf`, `fopen`, and so on.

5.3 Exporting MATLAB Results

You can easily save a MATLAB matrix to a computer file in a format that can be read by other programs, for example word processors or spreadsheets. Save each matrix into a separate file, but the matrix can have as many columns as you like. The commands are:

- to save the matrix in plain ASCII FORMAT, which is readable by many word processors or spreadsheets.
  ```
  ≫   save outfile.txt mymatrix -ascii
  ```

- to save the matrix in comma-delimited format
  ```
  ≫   csvwrite('outfile.csv', mymatrix)
  ```

- to save the matrix with a delimiter other than comma, for example using ;
  ```
  ≫   dlmwrite('outfile.txt', mymatrix, ';')
  ```

5.4 Missing Data

Sometimes when collecting data, a given piece of data is simply not available, or was incorrectly recorded and subsequently edited out of the record. When you have a single vector of data, such missing values can simply be ignored. However, when there is more than one item of data for each case, as in the placebo example above where the age data is missing for subject 4, it is necessary to put in a placeholder to preserve the right relationship between the other measurements.

In Resampling Stats, missing data is represented by the code `NaN`. This representation has the benefit that it is clear exactly what data is missing. It also means that computations done with missing data carry through the fact that some data is missing. For example, the short data set

```
≫   data = [1 2 NaN 3 4 NaN 6];
```

has two missing data points. As the next command shows, the mean of this data is missing!

```
≫   mean(data)
  ans:     NaN
```

This is actually the correct answer, since we have no idea what the missing data should be.

We can cleanse the data of missing values by using WEED and ISMISSING. Examples of this are given in the documentation for WEED and ISMISSING. To illustrate briefly:

```
≫  cleandata = weed(data, ismissing(data))
   ans:    1 2 3 4 6
≫  mean(cleandata) ⇒ ans:  3.2
```

If you have more than one related data vector, with missing data in different positions in the different vectors (as in the placebo example above, where there is missing data only in the age column, and not in the gender or "cell count" columns), you may want to clean the vectors separately or simultaneously, depending on the situation. In general, if you are performing an operation that treats each vector separately, you will likely want to clean that vector individually. This would be the case, for example, in computing the mean of each vector. In situations where two or more vectors are being combined, for example computing the correlation coefficient or doing multiple regression, you will want to clean the vectors simultaneously so that a missing value in any one vector renders invalid the entries in the corresponding position in the other vectors. (See WEED for how to clean vectors simultaneously.)

Some programs, like PLOT and HISTOGRAM, automatically ignore missing data in a sensible way.

5.5 Saving Figures

You have just created a figure which you wish to place in a document you are writing. You can do this by first saving your figure to a file using the built-in `print` command. (This command can either print or save the figure to a file.) Then, you can read in the saved figure file into whatever other software you are using to compose your document.

The first step is to decide what format the figure should be in. This is a matter between you and your word-processing software. There are many formats available. Some typical ones are

postscript a device-independent format, meaning that the printed quality will typically be high. There are many variations of this format, of which "encapsulated postscript" is often the most appropriate for inclusion in another document. This is the format used for figures in this book.

tiff a bitmap-based format, highly portable but often of low quality.

jpeg a compressed format often used on the WWW.

A complete list of the formats available from MATLAB can be gotten via the command

```
≫   help print
```

Once you have picked a format, note the corresponding format used by MATLAB (for example, `-depsc` for color encapsulated postscript, or `-djpeg90` for JPEG).

Remember that each figure is created in a MATLAB "figure window" which has a name like "Figure 1" or "Figure 2." To save the figure in figure window 1, for example, in encapsulated postscript format in a file called `myfig.eps`, give the command

```
≫   print -deps -f1 myfig.eps
```

You can give a complete directory path to specify a particular directory, for example

```
≫   print -deps -f1 c:/myfiles/mydocuments/myarticle/myfig.eps
```

Occasionally, after a figure has been created and saved, you realize that there is a typographical error, or that you want to change the labels, or that the size of the labels or axes limits are not what you would like. It is possible to save a MATLAB figure as an "m-file" which is a file containing the MATLAB instructions needed to regenerate the figure and which also contains all of the data that appears in the figure. In order to change the figure, you can either edit the m-file and then run it from within MATLAB, or you can run the m-file and then give additional MATLAB plotting commands, or both. To run an m-file, simply give the name of the m-file as a command (after changing to the appropriate directory using the MATLAB command `cd`).

You can save a figure as an m-file by using the "file/save-as" menu from the top of the figure window.