CHAPTER 2

Data formatting: the input file . . .

The first step in any analysis is gathering and collating your data. We'll assume that at the minimum, you have records for the marked individuals in your study, and from these records, can determine whether or not an individual was 'encountered' (in one fashion or another) on a particular sampling occasion.

Typically, your data will be stored in what we refer to as a 'vertical file' – where each line in the file is a record of when a particular individual was seen. For example, consider the following table, consisting of some individually identifying mark (ring or tag number), and the year. Each line in the file (or, row in the matrix) corresponds to the animal being seen in a particular year.

tag number	*year*
1147-38951	73
1147-38951	75
1147-38951	76
1147-38951	82
1147-45453	74
1147-45453	78

However, while it is easy and efficient to record the observation histories of individually marked animals this way, the 'vertical format' is not at all useful for capture-mark-recapture analysis. The preferred format is the *encounter history*. The encounter history is a contiguous series of specific dummy variables, each of which indicates something concerning the encounter of that individual – for example, whether or not it was encountered on a particular sampling occasion, how it was encountered, where it was encountered, and so forth. The particular encounter history will reflect the underlying model type you are working with (e.g., recaptures of live individuals, recoveries of dead individuals).

Consider for example, the encounter history for a typical mark-recapture analysis (the encounter history for a mark-recapture analysis is often referred to as a *capture history*, since it implies physical capture of the individual). In most cases, the encounter history consists of a contiguous series of '`1`'s and '`0`'s, where '`1`' indicates that an animal was recaptured (or otherwise known to be alive and in the sampling area), and '`0`' indicates the animal was not recaptured (or otherwise seen). Consider the individual in the preceding table with tag number '`1147-38951`'. Suppose that 1973 is the first year of the study, and that 1985 is the last year of the study. Examining the table, we see that this individual was captured and marked during the first year of the study, was seen periodically until 1982, when it was seen for the last time. The corresponding encounter-history for this individual would be: '`1011000001000`'. In other words, the individual was seen in 1973 (the starting '`1`'), not seen in 1974 ('`0`'), seen in 1975 and 1976 ('`11`'), not seen for the next 5 years ('`00000`'), seen again in 1982 ('`1`'), and then not seen again ('`000`').

While this is easy enough in principal, you surely don't want to have to construct capture-histories manually. Of course, this is precisely the sort of thing that computers are good for – large-scale data manipulation and formatting. **MARK** does not do the data formatting itself – no doubt you have your own preferred 'data manipulation' environment (**dBASE**, **Excel**, **Paradox**, **SAS**, **R**...). Thus, in general, you'll have to write your own program to convert the typical 'vertical' file (where each line represents the encounter information for a given individual on a given sampling occasion; see the example on the preceding page) into encounter histories (where the encounter history is a horizontal string).

In fact, if you think about it a bit, you realize that in effect what you need to do is to take a vertical file, and 'transpose' (or, 'pivot') it into a horizontal file – where fields to the right of the individual tag number represent when an individual was recaptured or resighted. However, while the idea of a 'transpose' or 'pivot' seems simple enough, there is one rather important thing that needs to be done – your program must insert the '`0`' value whenever an individual was not seen.

We'll assume for the purposes of this book that you will have some facility to put your data into the proper encounter-history format. For those of you who have no idea whatsoever on how to approach this problem, we provide some practical guidance in the Addendum at the end of this chapter. Of course, you could always do it by hand, if absolutely necessary!

begin sidebar

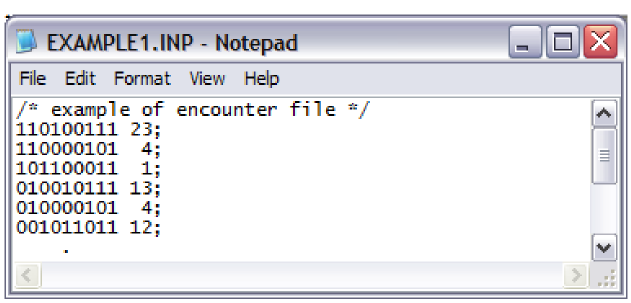
editing the .INP file

Many of the problems people have getting started with **MARK** can ultimately be traced back to problems with the `.INP` file. One common issue relates to choice of editor used to make changes/additions to the `.INP` file. You are strongly urged to avoid – as in 'like the plague' – using Windows Notepad (or, even worse, Word) to do much of anything related to building/editing `.INP` files. Do yourself a favor and get yourself a real ASCII editor. There are a number of very good 'free' editors you can (and should) use instead of Notepad (e.g., Notepad++, EditPad Lite, jEdit, and so on...)

end sidebar

2.1. Encounter histories formats

Now we'll look at the formatting of the encounter histories file in detail. It is probably easiest to show you a 'typical' encounter history file, and then explain it 'piece by piece'. The encounter-history reflects a mark-recapture experiment.


EXAMPLE1.INP - Notepad

File Edit Format View Help

```
/* example of encounter file */
110100111 23;
110000101  4;
101100011  1;
010010111 13;
010000101  4;
001011011 12;
    .
```

Superficially, the encounter histories file is structurally quite simple. It consists of an ASCII (text) file, consisting of the encounter history itself (the contiguous string of dummy variables), followed by one or more additional columns of information pertaining to that history. Each record (i.e., each line) in the encounter histories file ends with a semi-colon. Each history (i.e., each line, or record) must be the same length (i.e., have the same number of elements – the encounter history itself must be the same length over all records, and the number of elements 'to the right' of the encounter history must also be the same) – this is true regardless of the data type. The encounter histories file should have a `.INP` suffix (for example, **`EXAMPLE1.INP`**). Generally, there are no other 'control statements' or 'PROC statements' required in a **MARK** input file. However, you can optionally add comments to the `.INP` file using the 'slash-asterisk asterisk/slash' convention common to many programming environments – we have included a comment at the top of the example input file (shown at the bottom of the preceding page). The only thing to remember about comments is that they do **not** end with a semi-colon.

Let's look at each record (i.e., each line) a bit more closely. In this example, each encounter history is followed by a number. This number is the frequency of all individuals having a particular encounter history. This is not required (and in fact isn't what you want to do if you're going to consider individual covariates – more on that later), but is often more convenient for large data sets. For example, the summary encounter history

```
110000101 4;
```

could also be entered in the `.INP` files as

```
110000101 1;
110000101 1;
110000101 1;
110000101 1;
```

Note again that each line – each 'encounter history record' – ends in a semi-colon. How would you handle multiple groups? For example, suppose you had encounter data from males and females? In fact, it is relatively straightforward to format the `.INP` file for multiple groups – very easy for summary encounter histories, a bit less so for individual encounter histories. In the case of summary encounter histories, you simply add a second column of frequencies to the encounter histories to correspond to the other sex. For example,

```
110100111 23 17;
110000101  4  2;
101100011  1  3;
```

In other words, 23 of one sex and 17 of the other have history '`110100111`' (the ordering of the sexes – which column of frequencies corresponds to which sex – is entirely up to you). If you are using individual records, rather than summary frequencies, you need to indicate group association in a slightly less-obvious way – you will have to use a '`0`' or '`1`' within a group column to indicate the frequency – but obviously for one group only.

We'll demonstrate the idea here. Suppose we had the following summary history, with frequencies for males and females (respectively):

```
110000101   4   2;
```

In other words, 4 males, and 2 females with this encounter history (note: the fact that males come before females in this example is completely arbitrary. You can put whichever sex – or 'group' – you want in any column you want. All you'll need to do is remember which columns in the `.INP` file correspond to which groups. Comments are very helpful in keeping track of things.

To 'code' individual encounter histories, the `.INP` file would be modified to look like:

```
110000101  1 0;
110000101  1 0;
110000101  1 0;
110000101  1 0;
110000101  0 1;
110000101  0 1;
```

In this example, the coding '`1 0`' indicates that the individual is a male (frequency of `1` in the male column, frequency of `0` in the female column), and '`0 1`' indicates the individual is a female (frequency of 0 in the male column, and frequency of 1 in the male column). The use of one-record per individual is only necessary if you're planning on using individual covariates in your analysis.

2.1.1. Groups within groups...

In the preceding example, we had 2 groups: males and females. The frequency of encounters for each sex is coded by adding the frequency for each sex to the right of the encounter history.

But, what if you had something like males, and females (i.e., data from both sexes) and good colony and poor colony (i.e., data were sampled for both sexes from each of 2 different colonies – one classified as good, and the other as poor). How do you handle this in the `.INP` file? Well, all you need to do is have a frequency column for each (`sex.colony`) combination: one frequency column for females from the good colony, one frequency column for females from the poor colony, one frequency column for males from the good colony, and finally, one frequency column for males from the poor colony. An example of such an `.INP` file is shown below:

```
/* females, good - females, poor - males, good - males, poor */
00010        150   145   171   167;
00011        200   205   179   183;
00100        213   198   131    77;
00101         14    26    32    50;
00110         54    59    95   101;
00111         69    67    92   122;
01000         93    64    74    51;
```

As we will see in subsequent chapters, building models to test for differences between and among groups, and for interactions among groups (e.g., an interaction of sex and colony in this example) is relatively straightforward in **MARK** – all you'll really need to do is remember which frequency column codes for which grouping (hence the utility of adding comments to your `.INP` file, as we've done in this example).

2.2. Removing individuals from the sample

Occasionally, you may choose to remove individuals from the data set at a particular sampling occasion. For example, because your experiment requires you to remove the individual after its first recapture, or because it is injured, or for some other reason. The standard encounter history we have looked at so far records presence or absence only. How do we accommodate 'removals' in the `.INP` file? Actually, it's very easy – all you do is change the 'sign' on the frequencies from positive to negative. Negative frequencies indicates that that many individuals with a given encounter history were removed from the study.

For example,

```
100100 1500 1678;
100100  -23  -25;
```

In this example, we have 2 groups, and 6 sampling occasions. In the first record, we see that there were 1,500 individuals and 1,678 individuals in each group marked on the first occasion, not encountered on the next 2 occasions, seen on the fourth occasion, and not seen again. In the second line, we see the same encounter history, but with the frequencies '`-23`' and '`-25`'. The negative values indicate to **MARK** that 23 and 25 individuals in both groups were marked on the first occasion, not seen on the next 2 occasions, were encountered on the fourth occasion, at which time they were removed from the study. Clearly, if they were removed, they cannot have been seen again. So, in other words, 1,500 and 1,678 individuals recaptured and released alive, on the fourth occasion, in addition to 23 and 25 individuals that were recaptured, but removed, on the fourth occasion. So, (1,500 + 23) = 1,523 individuals in group 1, and (1,678 + 25) = 1,703 individuals in group 2, with encounter history '`100100`'.

Note: the '`-1`' code is for *removing* individuals from the live marked population. This is usually reserved for losses on capture (i.e., where the investigator captures an animal, and then, for some reason, decides to remove it from the study). The idea is that you don't want to include these 'biologist-caused' mortalities in the survival estimate.

On the other hand, if the known mortalities are *natural* (i.e., the investigator encounters a 'dead recovery'), and are not associated with the capture event itself, you have two options to get unbiased survival estimates

1. pretend you never observed the mortalities (i.e., just treat those individuals as regular releases that you never observe again). This approach is probably reasonable if the number of such mortalities is relatively small.
2. conduct a joint live recapture-dead recovery analysis with these 6 individuals treated as dead recoveries (see Chapter 9). Including the known mortalities (i.e., dead recoveries) will improve precision of your survival estimates.

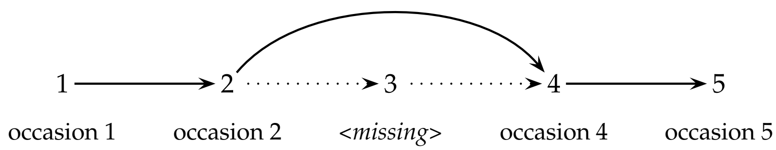
2.3. missing sampling occasions + uneven time-intervals

In the preceding, we have implicitly assumed that the sampling interval between sampling occasions is identical throughout the course of the study (e.g., sampling every 12 months, or every month, or every week). But, in practice, it is not uncommon for the time interval between occasions to vary – either by design, or because of 'logistical constraints'. In the extreme, you might even miss a sampling occasion altogether. This has clear implications for how you analyze your data.

For example, suppose you sample a population each October, and again each May (i.e., two samples within a year, with different time intervals between samples; October → May (7 months), and May → October (5 months)). Suppose the true monthly survival rate is constant over all months, and is equal to 0.9. As such, the estimated survival for October → May will be $0.9^7 = 0.4783$, while the estimated survival rate for May → October will be $0.9^5 = 0.5905$. Thus, if you fit a model without accounting for these differences in time intervals, it is clear that there would 'appear' to be differences in survival between successive samples, when in fact the monthly survival does not change over time.

So, how do you 'tell **MARK**' that the interval between samples may vary over time? In fact, depending on the particular situation, there are a couple of things you can do. First, consider the situation where you sample at all occasions, but where the interval between some of those occasions is not equal. You might think that you need to 'code' this interval information in the `.INP` file in some fashion. In fact, you don't – you specify the time intervals when you are specifying the data type in **MARK**, and not in the `.INP` file. In the `.INP` file, you simply enter the encounter histories as contiguous strings, regardless of the true interval between sampling occasions.

Alternatively, what if you're missing a sampling occasion altogether? For example, suppose you have a 5 occasion study, but for some reason, were unable to sample on the third occasion. This situation has 2 implications. First, the encounter probability on third occasion is logically 0. Second, in the absence of encounter data from the third occasion, the transition estimate for individuals alive and in the sample at occasion 2 would be from occasion $2 \rightarrow 4$, where 4 is the next sampling occasion in the data, and not $2 \rightarrow 3$.



So, in effect, missing a sampling occasion altogether is strictly equivalent to an unequal interval, at least with respect to estimating interval transition parameters, like survival.

However, it is quite different in terms of modeling the encounter probabilities. For simple unequal intervals, where all occasions are sampled (at unequal intervals), there is a parameter estimated for each encounter occasion. For missing sampling occasions, you need to account for both the unequal interval that is generated as an artifact of the missing occasion, and the fact that the encounter probability is logically 0 for the missing occasion.

In terms of formatting the data for the situation where a sampling occasion is missing, you have a couple of choices. Consider the following example encounter histories for our proposed 5 occasion study, where the third sampling occasion was missed:

```
11011   12;
10011    7;
01001    3;
```

Note that for all 3 histories, the missing third occasion is represented by a '`0`'. While technically correct, since you have explicitly entered a '`0`' for the third occasion, you will need to remember to fix the encounter parameter for that occasion to '`0`'.

An alternative approach which is available for some data types, you can indicate missing occasions explicitly in the encounter history by using a '`.`' ('dot') – see the '**`Help | Data Types`** menu option in **MARK** for a full list of data types that allow the 'dot' notation' in the encounter history for missing occasions.

For example, we would modify the preceding example histories as follows:

```
11.11   12;
10.11    7;
01.01    3;
```

There are some important considerations for handling missing occasions for data types where the transitions between occasions involve *multiple states* (e.g., multi-state models, robust design models –

see the relevant discussion in those chapters). We discuss general approaches to handling uneven time-intervals and missing sampling occasions in detail in Chapter 4.

begin sidebar

a 'dot' is not an individual covariate...

Using a 'dot' in the input file is appropriate if you have a missing occasion that applies to all of the individuals in your marked sample. It idoes not apply to the situation where (for some reason) you might not attempt to locate (encounter) a particular marked individual.

So, for example, consider the following:

```
110.1   12;
1..01    7;
1.001    3;
```

The intent with this formatting would be to indicate that the first individual was not 'searched for' on the fourth occasion, whereas the second individual was not searched for on the second and third occasions, and the final individual was not search for on the second occasion. In other words, having the sequence/pattern of 'dots' be potentially unique to the individual (analogous to an individual covariate; formatting for 'true' individual covariates is introduced in section 2.5.2).

This would be entirely incorrect – the assumption with almost all of the data types we're considering in **MARK** is that all of the marked individuals are available (or not) for possible encounter on a particular sampling occasion. In other words, if you sample your population, you are 'looking for' all of the marked individuals, not just a sub-sample of them.

So, an individual cannot have its own set of unique 'dots' to indicate you missed looking for them. A 'dot' for a particular occasion means you didn't 'look for' any of the individuals on that occasion (perhaps because you weren't able to go into the field). In that case, all of the individuals in the file would have a 'dot' for that missing occasion.

end sidebar

2.4. Different encounter history formats

Up until now, we've more or less used typical mark-recapture encounter histories (i.e., capture histories) to illustrate the basic principles of constructing an .INP file. However, **MARK** can be applied to far more than mark-recapture analysis, and as such, there are a number of slight permutations on the encounter history that you need to be aware of in order to use **MARK** to analyze your particular data type. First, we summarize in table form (top of the next page) most of the different data types **MARK** can handle, and the corresponding encounter history format.

Each data type in **MARK** requires a primary from of data entry provided by the encounter history. Encounter histories can consist of information on only live encounters (LLLL) or information on both live and dead (LDLDLDLD). In addition, some types allow a summary format (e.g., recovery matrix) which reduces the amount of input. The second column of the table shows the basic structure for a 4 occasion encounter history. There are, in fact, broad types: live encounters only, and mixed live and dead (or known fate) encounters.

For example, for a recaptures only study (i.e., live encounters), the structure of the encounter history would be 'LLLL' – where 'L' indicates information on encountered/not encountered status. As such, each '**L**' in the history would be replaced by the corresponding 'coding variable' to indicate encountered or not encountered status (usually '1' or '0' for the recaptures only history). So, for example, the encounter '1011' indicates seen and marked alive at occasion 1, not seen on occasion 2, and seen again at both occasion 3 and occasion 4.

recaptures only	LLLL
recoveries only	LDLDLDLD
both	LDLDLDLD
known fate	LDLDLDLD
closed captures	LLLL
BTO ring recoveries	LDLDLDLD
robust design	LLLL
both (Barker model)	LDLDLDLD
multi-strata	LLLL
Brownie recoveries	LDLDLDLD
Jolly-Seber	LLLL
Huggins' closed captures	LLLL
Robust design (Huggins)	LLLL
Pradel recruitment	LLLL
Pradel survival & seniority	LLLL
Pradel survival & λ	LLLL
Pradel survival & recruitment	LLLL
POPAN	LLLL
multi-strata - live and dead encounters	LDLDLDLD
closed captures with heterogeneity	LLLL
full closed captures with heterogeneity	LLLL
nest survival	LDLDLDLD
occupancy estimation	LLLL
robust design occupancy estimation	LLLL
open robust design multi-strata	LLLL
closed robust design multi-strata	LLLL

For data types including both live and dead individuals, the encounter history for the 4 occasion study is effectively 'doubled' – taking the format 'LDLDLDLD', where the 'L' refers to the live encountered or not encountered status, and the 'D' refers to the dead encountered or not encountered status. At each sampling occasion, either 'event' is possible – an individual could be both seen alive at occasion (i) and then found dead at occasion (i), or during the interval between (i) and (i+1). Since both 'potential events' need to be coded at each occasion, this effectively doubles the length of the encounter history from a 4 character string to an 8 character string.

For example, suppose you record the following encounter history for an individual over 4 occasions – where the encounters consist of both live encounters and dead recoveries. Thus, the history '10001100' reflects an individual seen and marked alive on the first occasion, not recovered during the first interval, not seen alive at the second occasion and not recovered during the second interval, seen alive on the third occasion and then recovered dead during the third interval, and not seen or recovered thereafter (obviously, since the individual was found dead during the preceding interval).

2.5. Some more examples

The **MARK** help files contain a number of different examples of encounter formats. We list only a few of them here. For example, suppose you are working with dead recoveries only. If you look at the table on the preceding page, you see that it has a format of '`LDLDLDLD`'. Why not just '`LLLL`', and using '`1`' for live', and '`0`' for recovered dead? The answer is because you need to differentiate between known dead (which is a known fate) , and simply not seen. '`0`' alone could ambiguously mean either dead, or not seen (or both!).

2.5.1. Dead recoveries only

The following is an example of dead recoveries only, because a live animal is never captured alive after its initial capture. That is, none of the encounter histories have more than one '`1`' in an **L** column. This example has 15 encounter occasions and 1 group. If you study this example, you will see that 500 animals were marked each sampling occasion.

```
000000000000000000000000000010  465;
000000000000000000000000000011   35;
000000000000000000000000001000  418;
000000000000000000000000001001   15;
000000000000000000000000001100   67;
000000000000000000000000100000  395;
000000000000000000000000100001    3;
000000000000000000000000100100   25;
000000000000000000000000110000   77;
```

Traditionally, 'dead recoveries only' data were summarized into what are known as recovery tables. **MARK** accommodates *recovery tables*, which have a 'triangular matrix form', where time goes from left to right (shown below). This format is similar to that used by Brownie *et al*. (1985).

```
 7    4   1    0   1;
      8   5    1   0;
         10    4   2;
              16   3;
                  12;
99   88 153  114 123;
```

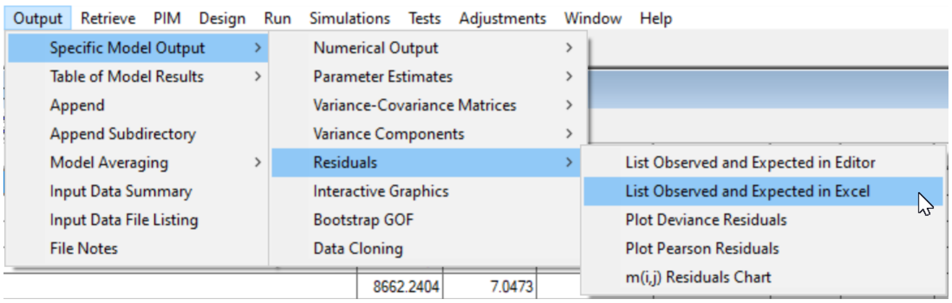
Following each matrix is the number of individuals marked each year. So, 99 individuals marked on the first occasion, of which 7 were recovered dead during the first interval, 4 during the second, 1 during the third, and so on.

`begin sidebar`

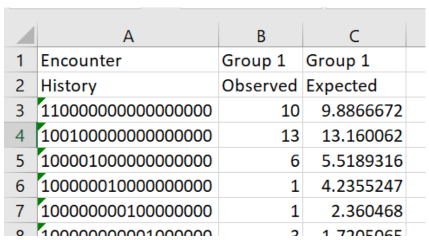
converting from 'recovery matrix' → 'recovery histories'...

The recovery matrix (*sensu* Brownie *et al*. 1985) is traditional, compact, and conveniently summarizes the basic dead recovery data. But this format aggregates data among individuals, making it impossible to include individual covariates that might be of interest in the analysis. To consider individual covariates, the recovery matrix must first be reformatted into typical 'encounter history' (or, in this case 'recovery history') format – the '`LDLD`' format noted earlier.

This reformatting is straightforward, requiring only a few steps. First, to do the transformation from recovery matrix → encounter histories, you need to have fit at least one dead recovery model to your dead recovery data – we're assuming that the data are initially in the 'recovery matrix' format. Once you have fit a model (and it doesn't matter which model you fit), you need to export the residuals from that model (residuals being the observed frequency of individuals with a particular recovery history, and the expected frequency, given the model and the data). To do this in **MARK**, simply select '`Output | Specific Model Output | List Observed and Expected in Excel`':



Once you've done this, you will be presented with an Excel spreadsheet consisting of 3 columns – '`history`', '`observed`' and '`expected`':



	A	B	C
1	Encounter	Group 1	Group 1
2	History	Observed	Expected
3	110000000000000000	10	9.8866672
4	100100000000000000	13	13.160062
5	100001000000000000	6	5.5189316
6	100000010000000000	1	4.2355247
7	100000000100000000	1	2.360468

For our present purpose, we want to select and export the first two columns – '`Encounter History`' and '`Observed`'. The '`Encounter History`' column contains the recovery histories we need (in '`LDLD`' format). The '`Observed`' column is simply the number (frequency) of individuals with a given history in the data. What we need to do next if we want each line in the recovery history file to correspond to a single individual is create replicates of each history, with the number of replicates being equal to the number 'observed' for that history.

While there are any number of ways you could accomplish this, the following short **R** script is probably minimum sufficient to get you started. The script assumes that you have copied the '`Encounter History`' and '`Observed`' data from the residuals spreadsheet, into an ASCII file, which is referred to in the script as `aggregated.inp` – in this file, do not add the semi-colon at the end of each line (the script will handle that for you). The script processes the data, and outputs the reformatted histories as `individual.inp`:

```
# step 1)  get aggregated encounter data
  aggregated <- read.table("C:/users/mark_user/Desktop/aggregate.inp", header=F);
  colnames(aggregated) <- c("history","observed");

# step 2) initialize the file you will output individual histories to...
 file.create("c:/users/mark_user/Desktop/individual.inp", showWarnings = FALSE)

# step 3) the main routine -- replicate each history freq times
```

```
    for (i in 1:nrow(aggregated)) {

     indiv <- replicate(aggregated$observed[i],aggregated$history[i]);
      indiv <- as.data.frame(indiv);

      # add the following at end of each history
       indiv$count <- " 1;";

  write.table(indiv, file="c:/users/egc/Desktop/individual.inp", sep=" ",
    append=TRUE , quote= FALSE, row.names=FALSE, col.names=FALSE)

  }
```

Now that you have the reformatted file (`individual.inp`), the next step is to match up each of the histories with the individual covariate(s) (section 2.5.2) you might be interested in. We'll leave this to you to work out.

end sidebar

2.5.2. Individual covariates

Finally, an example (below) of known fate data, where individual covariates are included. Comments are given at the start of each line to identify the individual (this is optional, but often very helpful in keeping track of things). Then comes the capture history for this individual, in a '`LDLDLD`...' sequence. Thus the first capture history is for an animal that was released on occasion 1, and died during the interval. The second animal was released on occasion 1, survived the interval, released again on occasion 2, and died during this second interval. Following the capture history is the count of animals with this history (always 1 in this example). Then, 4 covariates are provided. The first is a dummy variable representing age (`0` = subadult, `1` = adult), then a condition index, wing length, and body weight.

```
/* 01 */    1100000000000000    1   1   1.16   27.7   4.19;
/* 04 */    1011000000000000    1   0   1.16   26.4   4.39;
/* 05 */    1011000000000000    1   1   1.08   26.7   4.04;
/* 06 */    1010000000000000    1   0   1.12   26.2   4.27;
/* 07 */    1010000000000000    1   1   1.14   27.7   4.11;
/* 08 */    1010110000000000    1   1   1.20   28.3   4.24;
/* 09 */    1010000000000000    1   1   1.10   26.4   4.17;
```

What if you have multiple groups, such that individuals are assigned (or part of) a given group, and where you also have individual covariates? There are a couple of ways you could handle this sort of situation. You can either code for the groups explicitly in the `..INP` file, or use an individual covariate for the groups. There are pros and cons to either approach (this issue is discussed in Chapter 11).

Here is an snippet from a data set with 2 groups coded explicitly, and an individual covariate. In this data fragment, the first 8 contiguous values represent the encounter history, followed by 2 columns representing the frequencies depending on group: '`1 0`' indicating group 1, and '`0 1`' indicating group 2, followed by the value of the covariate:

```
11111111 1 0 123.211;
11111111 0 1  92.856;
```

```
11111110 1 0 122.115;
11111110 1 0 136.460;
```

So, the first record with an encounter history of '11111111' is in group 1, and has a covariate value of 123.211. The second individual, also with an encounter history of '11111111', is in group 2, and has a covariate value of 92.856. The third individual has an encounter history of '11111110', and is in group 1, with a covariate value of 122.115. And so on.

If you wanted to code the group as an individual covariate, this same input file snippet would look like:

```
11111111 1 1 123.211;
11111111 1 0  92.856;
11111110 1 1 122.115;
11111110 1 1 136.460;
```

In this case, following the encounter history, is a column of 1's, indicating the frequency for each individual, followed by a column containing a 0/1 dummy code to indicate group (in this example, we've used a '1' to indicate group 1, and a '0' to indicate group 2), followed by the value of the covariate.

A final example – for three groups where we code for each group explicitly (such that each group has it's own 'dummy column' in the input file), an encounter history with individual covariates might look like:

```
11111  1 0 0  123.5;
11110  0 1 0   99.8;
11111  0 0 1  115.2;
```

where the first individual with encounter history '11111' is in group 1 (dummy value of 1 in the first column after the encounter history, and 0's in the next two columns) and has a covariate value of 123.5, second individual with encounter history '11110' is in group 2 (dummy code of 0 in the first column, 1 in the second, and 0 in the third) and a covariate value of 99.8, and a third individual with encounter history '11111' in group 3 (0 in the first two columns, and a 1 in the third column), with a covariate value of 115.2.

As is noted in the help file (and discussed at length in Chapter 11), it is helpful to scale the values of covariates to have a mean on the interval $[0, 1]$ to ensure that the numerical optimization algorithm finds the correct parameter estimates. For example, suppose the individual covariate 'weight' is used, with a range from 1,000 g to 5,000 g. In this case, you should scale the values of weight to be from 0.1 to 0.5 by multiplying each 'weight' value by 0.0001.

In fact, **MARK** defaults to doing this sort of scaling for you automatically (without you even being aware of it). This 'automatic scaling' is done by determining the maximum absolute value of the covariates, and then dividing each covariate by this value. This results in each column scaled to between -1 and 1. This internal scaling is purely for purposes of ensuring the success of the numerical optimization – the parameter values reported by **MARK** (i.e., in the output that you see) are 'back-trasformed' to the original scale. Alternatively, if you prefer that the 'scaled' covariates have a mean of 0, and unit variance (this has some advantages in some cases), you can use the '**Standardize Individual Covariates**' option of the '**Run Window**' to perform the default standardization method (more on these in subsequent chapters).

More details on how to handle individual covariates in the input file are given in Chapter 11.

Summary

That's it! You're now ready to learn how to use **MARK**. Before you leap into the first major chapter (Chapter 3), take some time to consider that **MARK** will always do its 'best' to analyze the data you feed into it. However, it assumes that you will have taken the time to make sure your data are correct. If not, you'll be the unwitting victim to perhaps the most telling comment in data analysis: 'garbage in...garbage out'. Take some time at this stage to make sure you are confident in how to properly create and format your files.

Addendum: generating .INP files

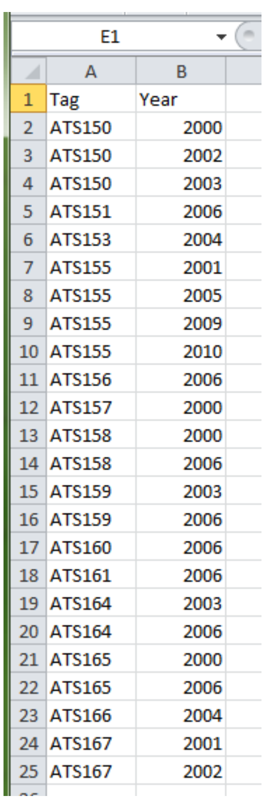
As noted at the outset of this chapter, **MARK** has no capability of generating input (`.INP`) files. This is something you will need to do for yourself. In this short addendum, we introduce two approaches to generating `.INP` files, the first based on 'Excel pivot tables', and the second using **R**. Since there are any number of different software applications for managing and manipulating data, we state for the record that we are going to demonstrate creating `.INP` files using Excel or **R**, not as a point of advocacy for either application, but owing more to the near ubiquity of both.

We also assume that in most cases, the data file that will need to be 'transformed' into a **MARK**-compatible `.INP` file consists of what we call a 'vertical array' of data – where each individual (identified by a band or tag number) encounter is entered as a separate row in the data file. One row for each encounter occasion. So, the minimum data file we consider has at least the band or tag number, and the time of the encounter (say, the year of the encounter). We seek to 'transpose' (or 'pivot') this 'vertical array' into a 'horizontal' encounter history.

Using Excel

Andrew Sterner, *Marine Turtle Research Group, University of Central Florida*

We will demonstrate the basic idea using an example where we will reformat an Excel spreadsheet containing some live encounter data (*note*: most of what follows applies generally to Access databases as well). We wish to format these data into an `.INP` file. The data are contained in the Excel spreadsheet `csj-pivot.xlsx` (note, we're clearly using Excel 2007 or later). Here are what the data look like before we transform them into an input file.


E1

	A	B
1	Tag	Year
2	ATS150	2000
3	ATS150	2002
4	ATS150	2003
5	ATS151	2006
6	ATS153	2004
7	ATS155	2001
8	ATS155	2005
9	ATS155	2009
10	ATS155	2010
11	ATS156	2006
12	ATS157	2000
13	ATS158	2000
14	ATS158	2006
15	ATS159	2003
16	ATS159	2006
17	ATS160	2006
18	ATS161	2006
19	ATS164	2003
20	ATS164	2006
21	ATS165	2000
22	ATS165	2006
23	ATS166	2004
24	ATS167	2001
25	ATS167	2002

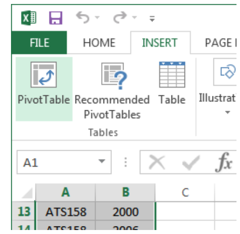
The file consists of two data columns: `TAG` (indicating the individual), and `YEAR` (the year that the individual was encountered). This data file contains encounter data for 15 marked individuals, with encounter data collected from 2000 to 2010 (thus, the encounter history will be 11 characters in length).

Our challenge, then is to take this 'vertical' file (one record per individual each year encountered), and 'pivot' it horizontally. For example, take the first individual in the file, `ATS150`. It was first encountered in 2000, again in 2002, and again (for the final time) in 2003. The second individual, `ATS151`, was seen for the first time in 2006, and then not seen again. The third individual, `ATS153`, was seen in 2004, and not seen again after that. And so on. If we had to generate the `.INP` file by hand for these individuals, their encounter histories would look like:

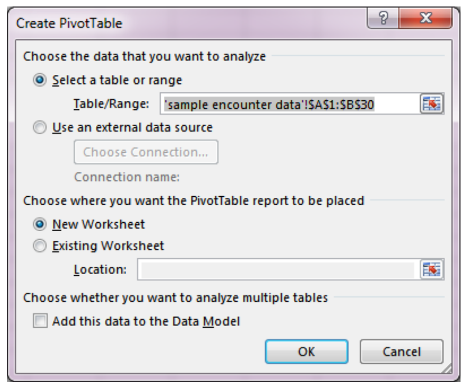
```
/* ATS150 */ 10110000000   1;
/* ATS151 */ 00000010000   1;
/* ATS153 */ 00001000000   1;
```

As it turns out, we can make use of the 'pivot table' in Excel (and some simple steps involving 'search and replace' and the `CONCATENATE` function), to generate exactly what we need. The process can be more involved for more complicated data types (e.g., robust design), but the basic principle of 'pivoting' applies.

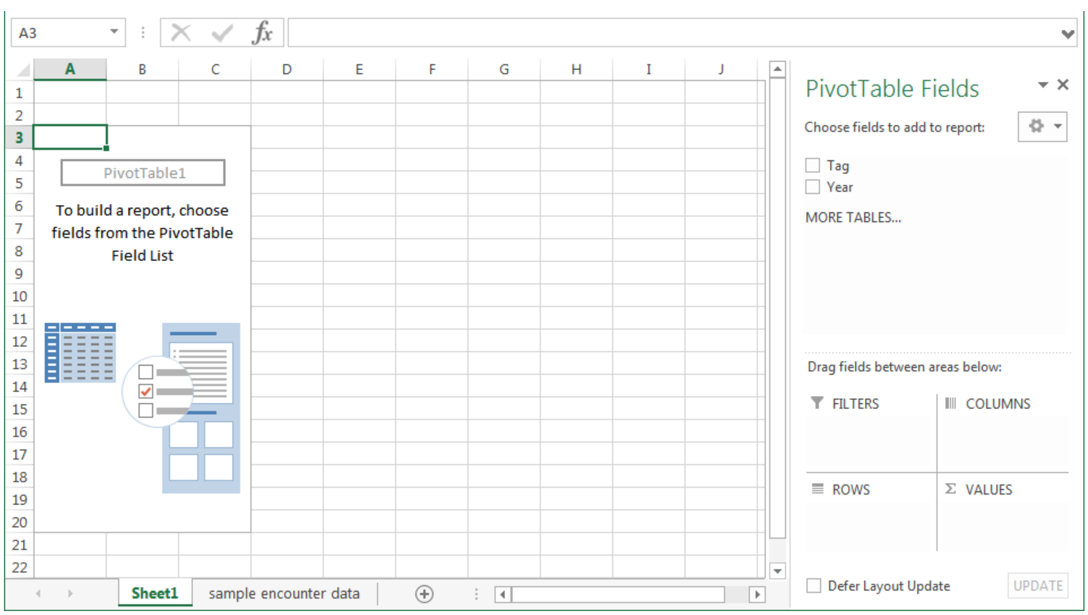
Here are the basic steps. First, we select the rows and columns containing the data. Then, select '**`Insert | PivotTable | PivotTable`**', as shown to the right (make sure you select **`PivotTable`** and not **`PivotChart`**).



This will bring up a dialog window (below) related to the data you want to use, and where you want the pivot table to be placed. We strongly recommend you put the pivot table into a '**`New Worksheet`**' (this is selected by default).:

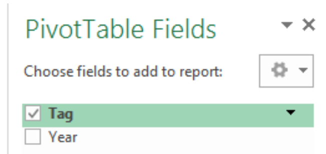


The '**`Table/Range`**' field will already be filled with the rows and columns of the data you selected. Once you click '**`OK`**', you will be presented with the template from which you will generate the pivot table:

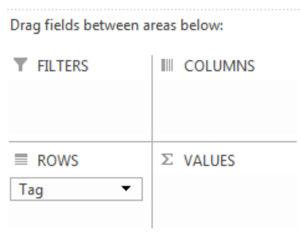


All you really need to do at this point is specify the '**`row labels`**', the '**`column labels`**', and the '**`values`**' (on the right hand side of the template).

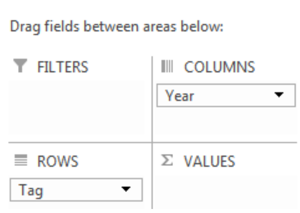
So, to specify the row labels, we simply select '**tag**'



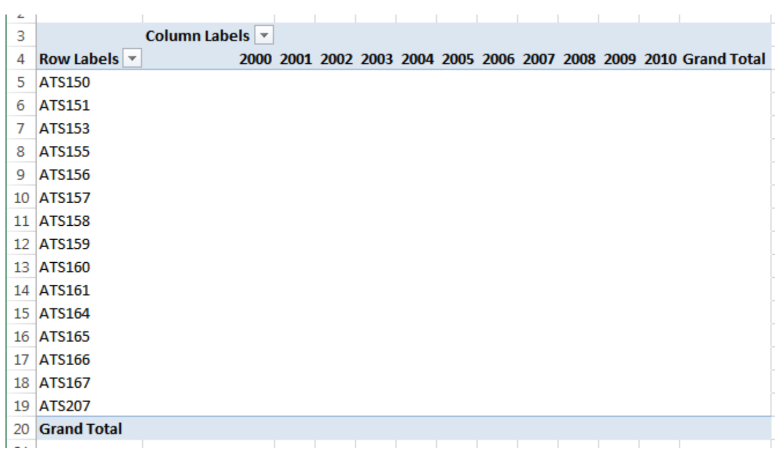
and then drag the '**tag**' field down to the '**row labels**' box at the bottom-right, if it isn't there already (this largely depends on which version of Excel you might have):



Then, do the same thing for the '**Year**' field: select '**Year**', and drag it down to the '**column labels**' box.



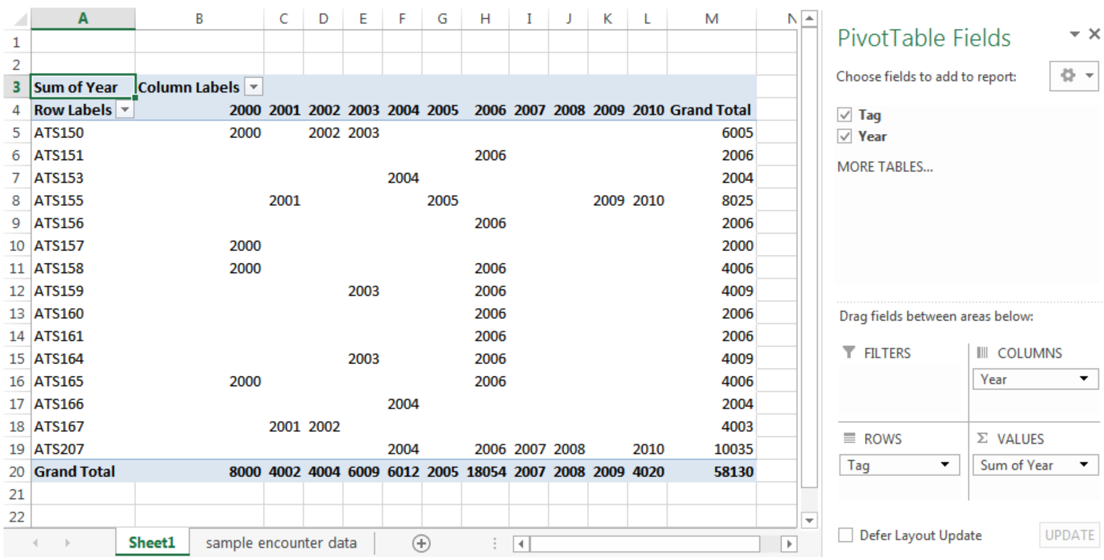
Once you have done this, you will quickly observe that a table (the 'pivot table') has been inserted into the main body of the template (below):



3		Column Labels											
4	Row Labels		2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	Grand Total
5	ATS150													
6	ATS151													
7	ATS153													
8	ATS155													
9	ATS156													
10	ATS157													
11	ATS158													
12	ATS159													
13	ATS160													
14	ATS161													
15	ATS164													
16	ATS165													
17	ATS166													
18	ATS167													
19	ATS207													
20	Grand Total													

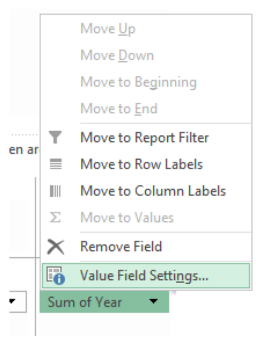
The table has row labels (individual tag numbers) and column labels (the years in your data file), plus some additional rows and columns for '**`Grand total`**' (reflecting the fact that pivot tables were designed primarily for business applications).

However, at present, there is nothing in the table (all of the cells are blank). Again, this may differ depending on the version of Excel you're using. Now, drag the '**`year`**' field label down to the '**`values`**' box in the lower right-hand corner.

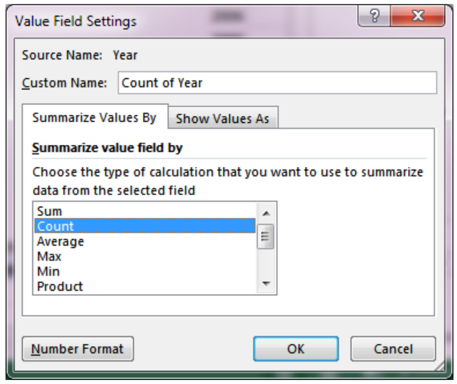


What we see is that the year during which an encounter has occurred for a given individual has been entered explicitly into the table, in the column corresponding to that year.

But, for an encounter history, we want a '`1`' to indicate the encounter year, not the year itself, and a '`0`' to indicate a year when an encounter did not occur. Achieving the first objective is easy. Simply pull down the '**`Sum of Year`**' menu, and select '**`Value Field Settings...`**':



Then, switch the '**`Summarize value field by`**' selection from '**`Sum`**' to '**`Count`**', as shown on the top of the next page.



As soon as you do this, then all of the years in the pivot table will be changed to 1. Why? Simple – all you've told the pivot table to do is count the number of times a year occurs in a given cell. Since the data file contains only a single record for each individual for each year it was encountered, then it makes sense that the tabulated '`Count`' should be a 1. Moreover, now the '`Grand Total`' rows and columns have some relevance – they indicate the number of individuals encountered in a given year (column totals), or the number of times a given individual was caught over the interval from 2000 to 2010 (row totals).

OK, on to the next step – putting a '`0`' in the blank cells for those years when an individual wasn't caught. This sounds easy enough in principle – a reasonable approach would be to select the rows and columns, and execute a 'search and replace', replacing blank cells with '`0`'. In fact, this is exactly what we want to do. However, for various reasons, you can't actually edit a pivot table. What you need to do first is select and copy the rows and columns (including the row labels, but excluding row and column totals), and paste them into a new worksheet. Then, simply do a '`Find & Select`'), replacing blanks (simply leave the '`Find what'` field empty) with a '`0`'.

The result is shown below:

	A	B	C	D	E	F	G	H	I	J	K	L
1	ATS150	1	0	1	1	0	0	0	0	0	0	0
2	ATS151	0	0	0	0	0	0	1	0	0	0	0
3	ATS153	0	0	0	0	1	0	0	0	0	0	0
4	ATS155	0	1	0	0	0	1	0	0	0	1	1
5	ATS156	0	0	0	0	0	0	1	0	0	0	0
6	ATS157	1	0	0	0	0	0	0	0	0	0	0
7	ATS158	1	0	0	0	0	0	1	0	0	0	0
8	ATS159	0	0	0	1	0	0	1	0	0	0	0
9	ATS160	0	0	0	0	0	0	1	0	0	0	0
10	ATS161	0	0	0	0	0	0	1	0	0	0	0
11	ATS164	0	0	0	1	0	0	1	0	0	0	0
12	ATS165	1	0	0	0	0	0	1	0	0	0	0
13	ATS166	0	0	0	0	1	0	0	0	0	0	0
14	ATS167	0	1	1	0	0	0	0	0	0	0	0
15	ATS207	0	0	0	0	1	0	1	1	1	0	1

(Alternatively, if you navigate to '`PivotTable | PivotTable Name | Options`', you will see an option to specify what an empty cell should show. Simply change it to a '`0`').

We're clearly getting closer. All that remains is to do the following. First, we remember that each line of the encounter history file must end with a frequency – where each line in the file corresponds to a single individual, then this frequency is simply '`1;`'. So, we simply enter '`1;`' into column **`K`**, and copy it down for as many rows as there are in the data (there are a number of ways to copy a value down a set of rows – we'll assume here you know of at least one way to do this).

Now, for a final step – we ultimately want an encounter history (`.INP` file) where the encounters form a contiguous string (i.e., no spaces). We can achieve this relatively easily by using the **`CONCATENATE`** function in Excel. Simply click the top-most cell in the next empty column (column **`N`** in our example), and then go up into the function box, and enter

```
=CONCATENATE("/* ",A1," */ ",B1,C1,D1,E1,F1,G1,H1,I1,J1,"  ",K1)
```

In other words, we want to 'concatenate' (merge together without spaces), various elements – some from within the spreadsheet, others explicitly entered (e.g., the delimiters for comments, so we can include the tag information, and some spacer elements).

Once you execute this cell macro, you can copy it down in column **`N`** over all rows in the file. If you manage to do this correctly, you will end up with a spreadsheet looking like the one below:

N1 =CONCATENATE("/* ",A1," */ ",B1,C1,D1,E1,F1,G1,H1,I1,J1,K1,L1," ",M1)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	ATS150	1	0	1	1	0	0	0	0	0	0	0	1;	/* ATS150 */ 10110000000 1;		
2	ATS151	0	0	0	0	0	0	1	0	0	0	0	1;	/* ATS151 */ 00000010000 1;		
3	ATS153	0	0	0	0	1	0	0	0	0	0	0	1;	/* ATS153 */ 00001000000 1;		
4	ATS155	0	1	0	0	0	1	0	0	0	1	1	1;	/* ATS155 */ 01000100011 1;		
5	ATS156	0	0	0	0	0	0	1	0	0	0	0	1;	/* ATS156 */ 00000010000 1;		
6	ATS157	1	0	0	0	0	0	0	0	0	0	0	1;	/* ATS157 */ 10000000000 1;		
7	ATS158	1	0	0	0	0	0	1	0	0	0	0	1;	/* ATS158 */ 10000010000 1;		
8	ATS159	0	0	0	1	0	0	1	0	0	0	0	1;	/* ATS159 */ 00010010000 1;		
9	ATS160	0	0	0	0	0	0	1	0	0	0	0	1;	/* ATS160 */ 00000010000 1;		
10	ATS161	0	0	0	0	0	0	1	0	0	0	0	1;	/* ATS161 */ 00000010000 1;		
11	ATS164	0	0	0	1	0	0	1	0	0	0	0	1;	/* ATS164 */ 00010010000 1;		
12	ATS165	1	0	0	0	0	0	1	0	0	0	0	1;	/* ATS165 */ 10000010000 1;		
13	ATS166	0	0	0	0	1	0	0	0	0	0	0	1;	/* ATS166 */ 00001000000 1;		
14	ATS167	0	1	1	0	0	0	0	0	0	0	0	1;	/* ATS167 */ 01100000000 1;		
15	ATS207	0	0	0	0	1	0	1	1	1	0	1	1;	/* ATS207 */ 00001011101 1;		
16																

All that remains is to select column **`N`** (which contains the formatted, concatenated encounter histories), and paste them into an ASCII text file. (A reminder here that you should avoid – as in 'like the plague' – using Word or Notepad as your ASCII editor. Do yourself a favor and get yourself a real ASCII editor.

Sampling occasions with no data...

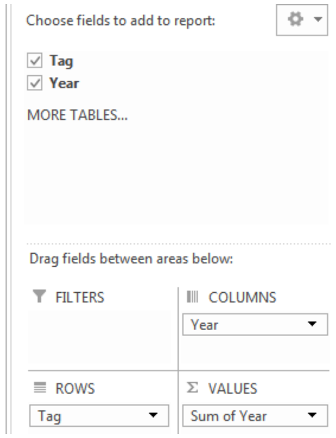
Suppose that you have no encounter data for a particular sampling occasion (meaning, no newly marked individuals, or encounters with previously marked individuals). Although not common, such a circumstance can arise for various reasons.

Let' consider how to handle this sort of situation, using a subset of the of the data contained in the `csj-pivot.xlsx` spreadsheet we used in the preceding section. Here, we'll simply drop the encounter data for the individual with the tag '`ATS207`'. We'll call this edited spreadsheet `csj-pivot_missing.xlsx`. After dropping individual '`ATS207`', we now have 14 individuals.

What about the range of years? If you look at `csj-pivot_missing.xlsx`, we see that the first encounter year is 2000, and the asst encounter year is still 2010. So, our encounter histories should still be 11 characters long.

We'll run through the 'pivot table' approach we just demonstrated in the preceding section, to see if in fact it works with the edited spreadsheet. We'll skip most of the various steps (which are identical to what was described earlier), and simply look at the pivot table (shown at the top of the next page).

3	Sum of Year	Column Labels									
4	Row Labels	2000	2001	2002	2003	2004	2005	2006	2009	2010	Grand Total
5	ATS150	2000		2002	2003						6005
6	ATS151							2006			2006
7	ATS153					2004					2004
8	ATS155		2001				2005		2009	2010	8025
9	ATS156							2006			2006
10	ATS157	2000									2000
11	ATS158	2000						2006			4006
12	ATS159				2003			2006			4009
13	ATS160							2006			2006
14	ATS161							2006			2006
15	ATS164				2003			2006			4009
16	ATS165	2000						2006			4006
17	ATS166					2004					2004
18	ATS167		2001	2002							4003
19	Grand Total	8000	4002	4004	6009	4008	2005	16048	2009	2010	48095



Superficially, looks very much the same as the pivot table generated for the full spreadsheet data. Ah, but look closely. Notice that there are columns for the years 2000 → 2006, and 2009 → 2010. But, there are no columns for 2007 and 2008! Why? Simple – with individual '`ATS207`' dropped from the data set, there simply are no encounter data for 2007 and 2008.

In fact, this is a significant issue. If we hadn't noticed the 'missing years', and simply proceeded with the remaining steps, we'd end up with encounter histories that 'look' correct, but which would be entirely wrong. The only clue we'd have, if we were paying attention, is that instead of being 11 characters long, they'd be only 9 characters long – because of the two missing years.

So, what do you do to deal with this problem? There are at least 2 options. First, you can simply copy the pivot table to a new sheet, and insert 2 new columns, for 2007 and 2008, respectively (as shown below – we've indicated the two new columns using a red font for the column labels):

	A	B	C	D	E	F	G	H	I	J	K	L
1	Row Label	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
2	ATS150	2000		2002	2003							
3	ATS151							2006				
4	ATS153					2004						
5	ATS155		2001				2005				2009	2010
6	ATS156							2006				
7	ATS157	2000										
8	ATS158	2000						2006				
9	ATS159				2003			2006				
10	ATS160							2006				
11	ATS161							2006				
12	ATS164				2003			2006				
13	ATS165	2000						2006				
14	ATS166					2004						
15	ATS167		2001	2002								

For the 'missing years', you can choose to either (i) fill them with `0`'s, along with the other zero cells of the table, or (ii) fill the columns for 2007 and 2008 with 'dots' (`.`). For some data types (see '**`Help | Data Types`** menu option in **MARK**), the 'dot' is interpreted as a missing occasion, which can be somewhat more convenient that manually setting the encounter probability to `0` for those years.

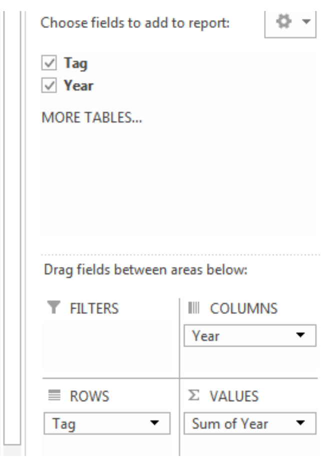
Alternatively, you can enter a 'fake' individual into your spreadsheet right at the start, before you begin constructing the pivot table. For this 'fake' individual, you simply add as many rows (where each row corresponds to a sampling occasion) as you have occasions over the whole data set (in this example, 11 occasions, from 2000 → 2010).

So, for this example,

23	ATS166	2004
24	ATS167	2001
25	fake	2000
26	fake	2001
27	fake	2002
28	fake	2003
29	fake	2004
30	fake	2005
31	fake	2006
32	fake	2007
33	fake	2008
34	fake	2009
35	fake	2010

Why do this? By introducing this 'fake' individual, where there is an encounter record (i.e., row) for every possible year in the data set. If you think about it, this should ensure that when we 'pivot' the spreadsheet there is a column for all years. If we try it, this is exactly what we see:

3	Sum of Year	Column Labels										
4	Row Labels	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	Grand
5	ATS150	2000		2002	2003								
6	ATS151							2006					
7	ATS153					2004							
8	ATS155		2001				2005				2009	2010	
9	ATS156							2006					
10	ATS157	2000											
11	ATS158	2000						2006					
12	ATS159				2003			2006					
13	ATS160							2006					
14	ATS161							2006					
15	ATS164				2003			2006					
16	ATS165	2000						2006					
17	ATS166					2004							
18	ATS167		2001										
19	fake	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	
20	Grand Total	10000	6003	4004	8012	6012	4010	18054	2007	2008	4018	4020	



So, we now simply select the 'real data' from the pivot table (as shown below), and copy it out to a new sheet, and proceed from there.

3	Sum of Year	Column Labels										
4	Row Labels	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
5	ATS150	2000		2002	2003							
6	ATS151							2006				
7	ATS153					2004						
8	ATS155		2001				2005				2009	2010
9	ATS156							2006				
10	ATS157	2000										
11	ATS158	2000						2006				
12	ATS159				2003			2006				
13	ATS160							2006				
14	ATS161							2006				
15	ATS164				2003			2006				
16	ATS165	2000						2006				
17	ATS166					2004						
18	ATS167		2001									
19	fake	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
20	Grand Total	10000	6003	4004	8012	6012	4010	18054	2007	2008	4018	4020

Both approaches are fairly straightforward to implement, and successfully address the problem of 'missing occasions'. If there is an advantage of the 'fake individual' approach, it is because it is easy to implement as a general practice. Meaning, for any data set, you simply (i) scroll to the bottom, and (ii) add encounter rows for each occasion in the study for a 'fake individual', (iii) run through the process of pivoting the spreadsheet, and then (iv) copy the 'real data' to a new sheet for the final processing.

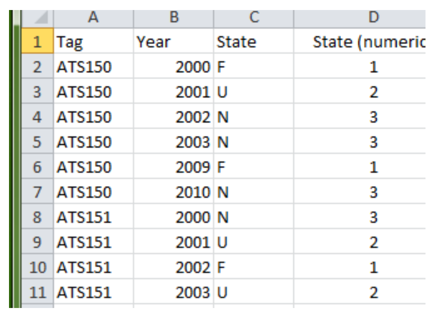
Other data types

Here we will consider 2 other data types, the robust design, and multi-state. Clearly, there are more data types in **MARK**, but these two represent very common data types, and if you understand steps in formatting `.INP` files for these two data types, you'll more than likely be able to figure out other data types on your own.

multi-state

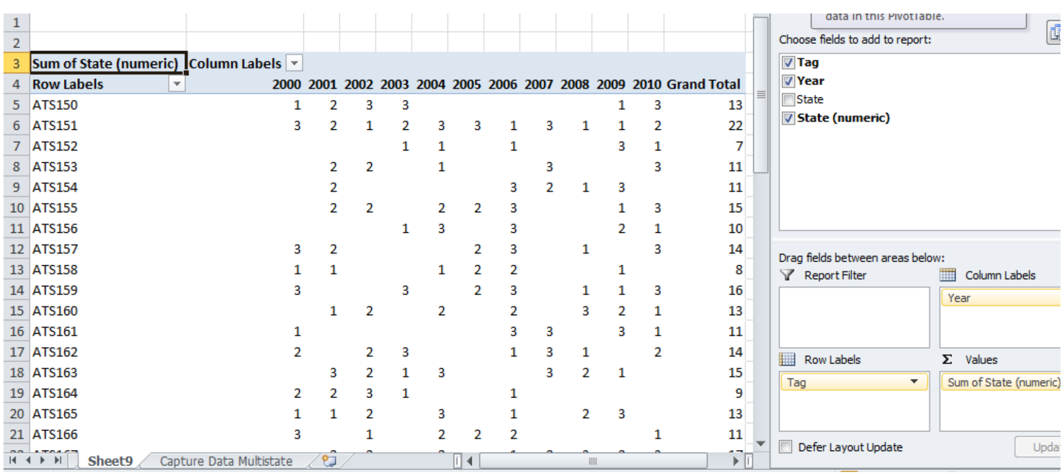
Here we will demonstrate formatting an `.INP` file for a multi-state data set (see Chapter 10). The encounter data we will use are contained in the Excel spreadsheet `MS-pivot.xlsx`. The file consists of 3 columns: `TAG` (indicating the individual), `YEAR` (the year the individual was encountered), and the `STATE` (for this example, there are 3 possible states: `F`, `U`, `N`).

We start by noting that `STATE` is a character (i.e., a letter). This might seem perfectly reasonable, since the most appropriate state name (indicator) might be a character. Unfortunately, Excel can't handle characters in the table cells when you pivot the table. As such, you first need to (i) select the column containing the state variable, (ii) copy this into the first empty column, and (iii) execute a '**`Find and Replace`**' in this column, such that you change `F` $\longrightarrow$ `1`, `U` $\longrightarrow$ `2`, and `N` $\longrightarrow$ `3`. Once finished, your Excel spreadsheet shoot look something like what is shown to the right.



	A	B	C	D
1	Tag	Year	State	State (numeric
2	ATS150	2000	F	1
3	ATS150	2001	U	2
4	ATS150	2002	N	3
5	ATS150	2003	N	3
6	ATS150	2009	F	1
7	ATS150	2010	N	3
8	ATS151	2000	N	3
9	ATS151	2001	U	2
10	ATS151	2002	F	1
11	ATS151	2003	U	2

Next, select the data, and insert a Pivot Table into a new sheet in the spreadsheet. Drag `TAG` to the '**`Row Labels`**' box, `YEAR` to the '**`Column Labels`**' box, and `State (numeric)` to the '**`Values`**' box:



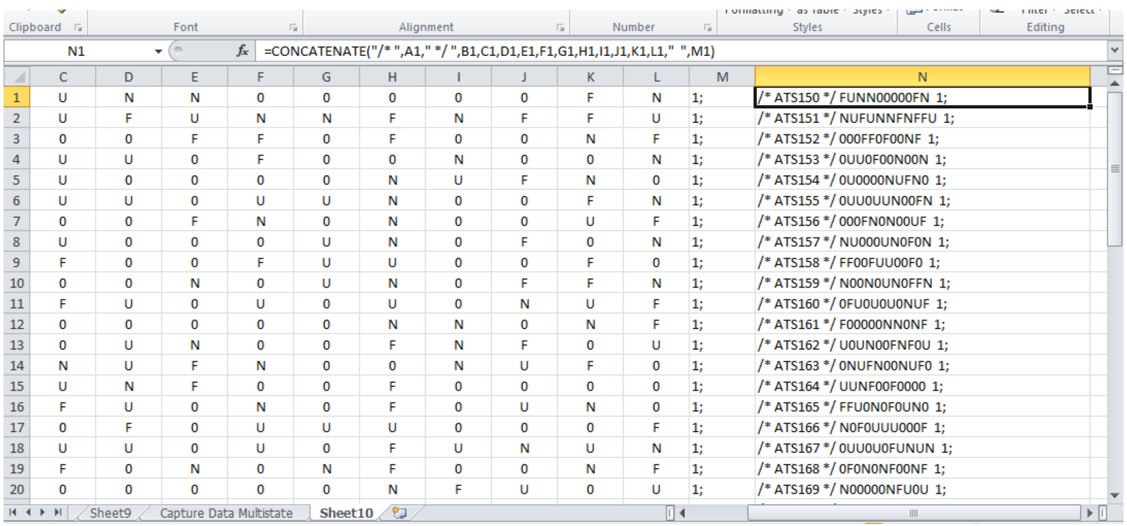
3	Sum of State (numeric)	Column Labels											
4	Row Labels	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	Grand Total
5	ATS150	1	2	3	3						1	3	13
6	ATS151	3	2	1	2	3	3	1	3	1	1	2	22
7	ATS152				1	1		1			3	1	7
8	ATS153		2	2		1			3			3	11
9	ATS154		2					3	2	1	3		11
10	ATS155		2	2		2	2	3			1	3	15
11	ATS156				1	3		3			2	1	10
12	ATS157	3	2				2	3		1		3	14
13	ATS158	1	1			1	2	2			1		8
14	ATS159	3			3		2	3		1	1	3	16
15	ATS160		1	2		2		2		3	2	1	13
16	ATS161	1						3	3		3	1	11
17	ATS162	2		2	3			1	3	1		2	14
18	ATS163		3	2	1	3			3	2	1		15
19	ATS164	2	2	3	1			1					9
20	ATS165	1	1	2		3		1		2	3		13
21	ATS166	3		1		2	2	2				1	11

Next, copy the `TAGS`, `YEARS` and table values to a new worksheet. Then '**`Find and Replace`**' all the blank cells with zeros. At this point, you have a decision to make: you can either (i) '**`Find and Replace`**' the states from numeric back to their original character values (i.e., 1 ⟶ `F`, 2 ⟶ `U` and 3 ⟶ `N`), or (ii) leave the states numeric, and simply inform **MARK** what the states mean. For this example, we'll '**`Find and Replace`**' the states from numeric back to their original character values. Finally, add a column of '`1;`' to the new worksheet.

Then click the top-most cell in the next empty column (column **`L`** in our example), and then go up into the function box, and enter

```
=CONCATENATE("/* ",A1," */ ",B1,C1,D1,E1,F1,G1,H1,I1,J1,K1,L1,"  ",M1)
```

In other words, we want to 'concatenate' (merge together without spaces), various elements – some from within the spreadsheet, others explicitly entered (e.g., the delimiters for comments, so we can include the tag information, and some spacer elements). Once you execute this cell macro, you can copy it down in column **`L`** over all rows in the file. The final worksheet should look something like the one shown at the top of the next page. At this point, you simply copy your concatenated encounter histories from column **N** into an editor, and save into an `.INP` file.


Clipboard Font Alignment Number Styles Cells Editing

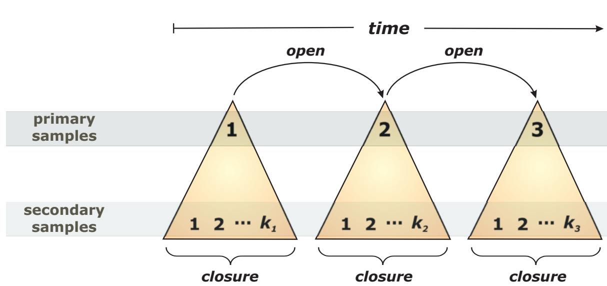
N1 =CONCATENATE("/* ",A1," */ ",B1,C1,D1,E1,F1,G1,H1,I1,J1,K1,L1," ",M1)

	C	D	E	F	G	H	I	J	K	L	M	N
1	U	N	N	0	0	0	0	0	F	N	1;	/* ATS150 */ FUNN00000FN 1;
2	U	F	U	N	N	F	N	F	F	U	1;	/* ATS151 */ NUFUNNFNFFU 1;
3	0	0	F	F	0	F	0	0	N	F	1;	/* ATS152 */ 000FF0F00NF 1;
4	U	U	0	F	0	0	N	0	0	N	1;	/* ATS153 */ 0UU0F00N00N 1;
5	U	0	0	0	0	N	U	F	N	0	1;	/* ATS154 */ 0U0000NUFN0 1;
6	U	U	0	U	U	N	0	0	F	N	1;	/* ATS155 */ 0UU0UUN00FN 1;
7	0	0	F	N	0	N	0	0	U	F	1;	/* ATS156 */ 000FN0N00UF 1;
8	U	0	0	0	U	N	0	F	0	N	1;	/* ATS157 */ NU000UN0F0N 1;
9	F	0	0	F	U	U	0	0	F	0	1;	/* ATS158 */ FF00FUU00F0 1;
10	0	0	N	0	U	N	0	F	F	N	1;	/* ATS159 */ N00N0UN0FFN 1;
11	F	U	0	U	0	U	0	N	U	F	1;	/* ATS160 */ 0FU0U0U0NUF 1;
12	0	0	0	0	0	N	N	0	N	F	1;	/* ATS161 */ F00000NN0NF 1;
13	0	U	N	0	0	F	N	F	0	U	1;	/* ATS162 */ U0UN00FNF0U 1;
14	N	U	F	N	0	0	N	U	F	0	1;	/* ATS163 */ 0NUFN00NUF0 1;
15	U	N	F	0	0	F	0	0	0	0	1;	/* ATS164 */ UUNF00F0000 1;
16	F	U	0	N	0	F	0	U	N	0	1;	/* ATS165 */ FFU0N0F0UN0 1;
17	0	F	0	U	U	U	0	0	0	F	1;	/* ATS166 */ N0F0UUU000F 1;
18	U	U	0	U	0	F	U	N	U	N	1;	/* ATS167 */ 0UU0U0FUNUN 1;
19	F	0	N	0	N	F	0	0	N	F	1;	/* ATS168 */ 0F0N0NF00NF 1;
20	0	0	0	0	0	N	F	U	0	U	1;	/* ATS169 */ N00000NFU0U 1;

Sheet9 Capture Data Multistate Sheet10

robust design

For our final example, we consider formatting an `.INP` file for a robust design analysis (the robust design is covered in Chapter 15). In brief, the robust design combines closed population samples embedded (nested) within open population samples. Consider the following figure:



As shown, there are 3 'open population' samples (known as primary period samples). Between open samples, population abundance can change due to emigration, death, immigration or birth. Within each open sample period are embedded k 'closed population' (or secondary) samples. The trick here is to encode the encounter history taking into account the presence of both primary and secondary samples (where the number of secondary samples may vary among primary samples). As you might expect, the greater complexity of the RD encounter file might require a somewhat higher level of Excel proficiency than the first two examples we discussed earlier.

In this example (data in `RD-pivot.xlsx`), we assume primary samples from 2000-2010. Within each primary period, we have 4 secondary samples, which occur from May 1 to May 15 (secondary sample 1), May 16 to May 30 (secondary sample 2), June 1 to June 15 (secondary sample 3), and June 16 to June 30 (secondary sample 4). For each secondary sample, an encountered individual is recorded only once. We imagine that your data are stored in the following way. For each individual (`TAG`), for each primary sample (`YEAR`), you have a series of columns, one for each secondary sampling period.

	A	B	C	D	E	F	G
1	Tag	Date	Year				
2							
3	ATS150		2000		5/30/12		6/17/12
4	ATS150		2001	5/2/12			6/24/12
5	ATS150		2002			6/4/12	
6	ATS150		2003		5/23/12	6/2/12	6/25/12
7	ATS150		2009				
8	ATS150		2010	5/6/12	5/17/12	6/10/12	6/17/12
9	ATS151		2000	5/13/12	5/18/12		
10	ATS151		2001	5/7/12			
11	ATS151		2002			6/10/12	6/18/12
12	ATS151		2003			6/1/12	6/16/12
13	ATS151		2004	5/6/12	5/25/12	6/10/12	6/19/12
14	ATS151		2005	5/11/12	5/18/12		
15	ATS151		2006	5/11/12	5/28/12		
16	ATS151		2007	5/9/12			
17	ATS151		2008	5/2/12			
18	ATS151		2009				6/23/12
19	ATS151		2010			6/12/12	6/25/12
20	ATS152		2003	5/12/12	5/25/12	6/3/12	6/29/12

For example, in the preceding figure, we see that individual with tag '`ATS150`' was observed during primary sample, 2000, 2001, 2002, 2003, 2009, and 2010. In 2000, the individual was not observed during the first secondary sample (May 1 to May 15), was observed during the second secondary sample (May 16 to May 30), was not observed during the third secondary sample (June 1 to June 15), and was observed during the fourth and final secondary sample (June 16 to June 30). In contrast, in 2010, the individual with tag '`ATS1150`' was observed in all 4 secondary samples.

Now, you may be wondering why we've entered dates in terms of 2012, even for primary encounter years <2012. For example, for '`ATS150`', we enter '`5/30/12`' as the date for the encounter during the second secondary sample period. We need to do this in order to make use of some very handy Excel functions. For example, consider the '`year`' function. This function extracts the year associated with a given date (such that if you type in '`=year(B2)`' and `B2` is a date, it will return the year associated with that date. So, for robust design data, you may have intervals (for a secondary sample period) spanning from 5/1/12 to 5/15/12, and you want to know if the encounter date falls between them.

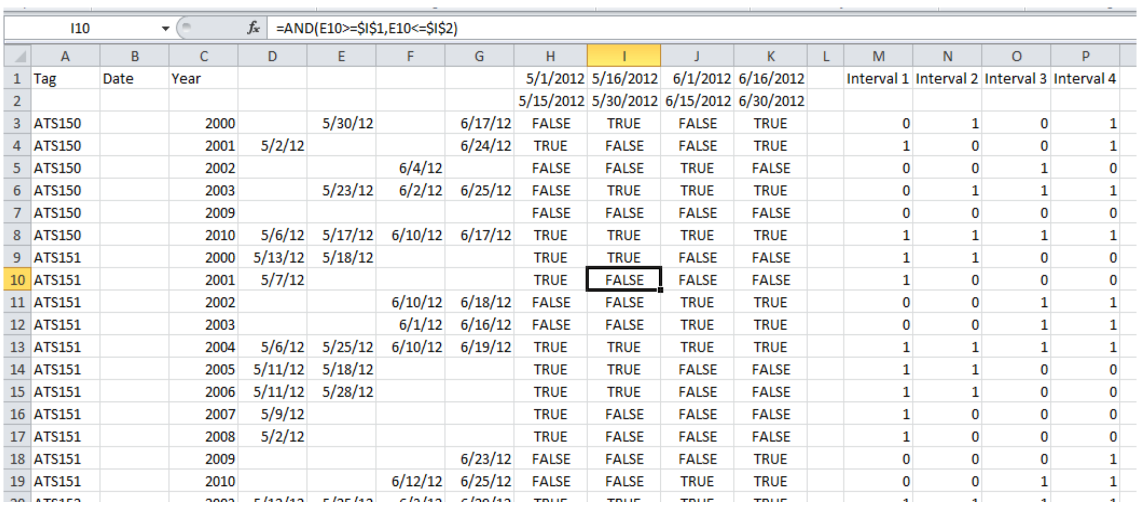
All you need to do is

- use the `AND` function to determine if a date falls within a given range. For example, in cell `H3` in the spreadsheet, we enter

```
=AND(D3>=H1,D#<=H2)
```

- What you are asking Excel is: "Is `D3` (my date of capture) greater than or equal to my first date, `5/1/12`, and less than or equal to `5/15/12`". We do the same thing for each of the other 3 secondary sample periods.
- This may seem a bit odd at first but keep in mind that Excel treats all dates as a number of days since January 1, 1900 or 1904 (depending on which version of Excel you are using)
- The `AND` function will return a `TRUE` value is the criteria in the parenthesis are met or a `FALSE` value if they are not
- Once you have got all of your `TRUE` and `FALSE` values copy them into a separate set of columns. Note that instead of just '**`paste`**'or '**`ctrl+v`**', you want to right click and '**`paste special`**' and select the '**`Values`**' box. This tells Excel to just give you the displayed number text or whatever appears in the box without any of the underlying formulas.
- Now you can '**`Find and Replace`**' `TRUE` with `1` and `FALSE` with `0`

These steps (and cell macros) are shown in worksheet '`RD within season period trick`'. At this point, you will se something that look like



I10 =AND(E10>=$I$1,E10<=$I$2)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Tag	Date	Year					5/1/2012	5/16/2012	6/1/2012	6/16/2012		Interval 1	Interval 2	Interval 3	Interval 4
2								5/15/2012	5/30/2012	6/15/2012	6/30/2012					
3	ATS150		2000		5/30/12		6/17/12	FALSE	TRUE	FALSE	TRUE		0	1	0	1
4	ATS150		2001	5/2/12			6/24/12	TRUE	FALSE	FALSE	TRUE		1	0	0	1
5	ATS150		2002			6/4/12		FALSE	FALSE	TRUE	FALSE		0	0	1	0
6	ATS150		2003		5/23/12	6/2/12	6/25/12	FALSE	TRUE	TRUE	TRUE		0	1	1	1
7	ATS150		2009					FALSE	FALSE	FALSE	FALSE		0	0	0	0
8	ATS150		2010	5/6/12	5/17/12	6/10/12	6/17/12	TRUE	TRUE	TRUE	TRUE		1	1	1	1
9	ATS151		2000	5/13/12	5/18/12			TRUE	TRUE	FALSE	FALSE		1	1	0	0
10	ATS151		2001	5/7/12				TRUE	FALSE	FALSE	FALSE		1	0	0	0
11	ATS151		2002			6/10/12	6/18/12	FALSE	FALSE	TRUE	TRUE		0	0	1	1
12	ATS151		2003			6/1/12	6/16/12	FALSE	FALSE	TRUE	TRUE		0	0	1	1
13	ATS151		2004	5/6/12	5/25/12	6/10/12	6/19/12	TRUE	TRUE	TRUE	TRUE		1	1	1	1
14	ATS151		2005	5/11/12	5/18/12			TRUE	TRUE	FALSE	FALSE		1	1	0	0
15	ATS151		2006	5/11/12	5/28/12			TRUE	TRUE	FALSE	FALSE		1	1	0	0
16	ATS151		2007	5/9/12				TRUE	FALSE	FALSE	FALSE		1	0	0	0
17	ATS151		2008	5/2/12				TRUE	FALSE	FALSE	FALSE		1	0	0	0
18	ATS151		2009				6/23/12	FALSE	FALSE	FALSE	TRUE		0	0	0	1
19	ATS151		2010			6/12/12	6/25/12	FALSE	FALSE	TRUE	TRUE		0	0	1	1

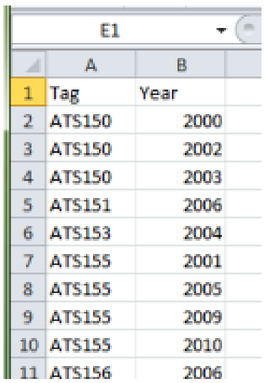
At this point, the remaining steps are similar to the same steps we used for CJS and MS data types (as described earlier). You simply

1. copy the the data to a new worksheet (shown in '`capture data-robust design`')
2. Select the data, then '**`Insert | Pivot Table | Pivot Table`**'
3. Drag `Tag` to '**`Row Label`**', `Year` to '**`Column Label`**'
4. Now here is another difference for the RD: there are multiple occasions per year. So just drag each one to the values box in the order that they occur!
5. concatenate into a contiguous encounter history, and you're done. Have a look at the worksheet '`RD Input Construction`' for what it should look like.

Using R

Brandon Merriell & Cory Snyder, *Environmental and Sustainability Sciences, Cornell University*

We will demonstrate the basic idea using an example where we will reformat an existing Excel spreadsheet containing some live encounter data (*note*: most of what follows applies generally to Access databases as well). We wish to format these data into an `.INP` file. The data are contained in the Excel spreadsheet `csj-pivot.xlsx` (the same example spreadsheet used in the preceding section demonstrating the use of a 'pivot table' in Excel). Here are what the data look like before we transform them into an input file:



	A	B
1	Tag	Year
2	ATS150	2000
3	ATS150	2002
4	ATS150	2003
5	ATS151	2006
6	ATS153	2004
7	ATS155	2001
8	ATS155	2005
9	ATS155	2009
10	ATS155	2010
11	ATS156	2006

The file consists of two data columns: `TAG` (indicating the individual), and `YEAR` (the year that the individual was encountered). This data file contains encounter data for 15 marked individuals, with encounter data collected from 2000 to 2010 (thus, the encounter history will be 11 characters in length).

Our challenge, then is to take this 'vertical' file (one record per individual each year encountered), and 'pivot' it horizontally. For example, take the first individual in the file, `ATS150`. It was first encountered in 2000, again in 2002, and again (for the final time) in 2003. The second individual, `ATS151`, was seen for the first time in 2006, and then not seen again. The third individual, `ATS153`, was seen in 2004, and not seen again after that. And so on.

If we had to generate the `.INP` file by hand for these individuals, their encounter histories would look like:

```
/* ATS150 */ 101100000   1;
/* ATS151 */ 000000100   1;
/* ATS153 */ 000010000   1;
```

Accomplishing this is relatively straightforward in **R**, using the capabilities of the **`reshape`** package (Wickham & Hadley 2007). Basically, the **`reshape`** package lets you "melt" data so that each row is a unique id-variable combination. Then you "cast" the melted data into any shape you would like (see the package documentation for details, and examples).

As noted in the preceding section on using a pivot table in Excel, one technical 'hassle' you might face is how to handle occasions (say, years in this example) where there are simply no data (i.e., occasions with no initial capture-marking events, and no reencounter events of previously marked individuals). The 'challenge' is to find a way to (i) identify the missing occasions (years, for this example), (ii) add encounter columns consisting of all `0`'s for those missing years, and (iii) make sure the resulting set of encounter columns is ordered in the correct chronological (ordinal) sequence.

There are probably several ways to accomplish this in **R**. A 'brute force' approach, as introduced in the preceding section on using Excel to format a CJS `.INP` file using pivot tables, is to simply introduce encounter data for a 'fake individual' before formatting the data. We'll return to that in a moment. For now, we'll consider a 'general approach' using **R**, that automatically detects and accounts for 'missing occasions'. This approach is based on comparing the set (list) of occasions (years) that are in the encounter file against a canonical list of all the occasions (years) that occur between the first and last encounter years. We will first confirm that the script works for a data set without any missing occasions (`csj-pivot.xlsx`), and then test the approach to handling 'missing occasions' using a subset of these data, after dropping the encounter data for individual with tag '`ATS207`' (`csj-pivot_missing.xlsx`).

Here is the script:

```
# simple code to generate encounter history .inp files for MARK from CJS data

# clean up a few things, and set wd
rm(list=ls(all=TRUE))
getwd()
setwd("C:/Users/markbook/Desktop")

# load package "reshape" first
library(reshape)

# read in the data - assumes .csv file minimially contains individual identification field
# called 'Tag' and the encounter year as 'Year'. Following code assumes 'Tag' is in the
# first column, and 'Year' is in some other column (say, the second column).
data=read.csv("cjs-pivot.csv")
data=data.frame(data); data

# now, we reshape the data using melt
transform=melt(data, id.vars="Tag")
pivot=cast(transform, Tag ~ value)
pivot[is.na(pivot)]=0

# turns all non-zero matrix elements (for year, not tag) into 1
# following assumes Tag is in the first column of pivot
pivot[,2:ncol(pivot)][pivot[,2:ncol(pivot)] != 0] = 1

##  now get everything ready to output the CH ##

# find length of history
lh <- max(data$Year)-min(data$Year)+2;

# following code needed to accommodate any missing years of data. Basic logic
# is to identify missing occasions by comparing columns in pivot table with
# a canonical list (occStr) of occasions (years) that should be in the table...

occStr <- seq(min(data$Year),max(data$Year),1);  # vector of years you want
occStr <- as.character(occStr);                  # convert to character string
occStr <- c("Tag",occStr);                       # pre-pend tag onto occStr

Missing <- setdiff(occStr,names(pivot))   # Find names of missing columns
pivot[Missing] <- 0                       # Add them, filled with '0's or dots
pivot <- pivot[occStr]                    # sort in ordinal sequence

#
# now do formatting of encounter history file
#

pivot$eh <- apply(pivot[2:lh],1,paste,collapse="") # concatenates encounter columns into eh
pivot[2:lh] <- NULL # drops individual encounter columns
```

```
# create commented tag
pivot$Tag <- paste("/*", pivot$Tag, "*/", sep=" ")

# sort by descending encounter histories
pivot <- pivot[order(pivot$eh,decreasing=TRUE),]

# tack on the frequency for the individual
pivot$end <- "1;";

# write out the input file
write.table(pivot,file="cjs-pivot.inp",sep="   ",quote=F,col.names=F,row.names=F);
```

The output file `cjs-pivot.inp` (shown below) is exactly what we want – the commented individual tag, followed by the encounter history as a contiguous string length 11, consisting of `1`'s and `0`'s, followed by a '`1;`' to indicate the individual frequency:

```
|...+....1....+....2....+....3...
/* ATS150 */   10110000000   1;
/* ATS158 */   10000010000   1;
/* ATS165 */   10000010000   1;
/* ATS157 */   10000000000   1;
/* ATS167 */   01100000000   1;
/* ATS155 */   01000100011   1;
/* ATS159 */   00010010000   1;
/* ATS164 */   00010010000   1;
/* ATS207 */   00001011101   1;
/* ATS153 */   00001000000   1;
/* ATS166 */   00001000000   1;
/* ATS151 */   00000010000   1;
/* ATS156 */   00000010000   1;
/* ATS160 */   00000010000   1;
/* ATS161 */   00000010000   1;
```

Sampling occasions with no data...

Now, does the script work if there are 'missing occasions'? We'll look at the data contained in the spreadsheet `csj-pivot_missing.xlsx`, which is a subset of these full dataset, after dropping the encounter data for individual with tag '`ATS207`'. As a result, in this subset of the data, there are no encounter data for 2007 and 2008.

If we run through the first few lines of the **R** script, and have a look at the the first few rows of the `pivot` dataframe before we have the script 'look' for the missing occasions, we see that the `pivot` dataframe looks much the same as what we saw using Excel – and (looking closely), we see that we're missing occasions columns) for 2007 and 2008:

```
> pivot
     Tag 2000 2001 2002 2003 2004 2005 2006 2009 2010
1  ATS150 2000   NA 2002 2003   NA   NA   NA   NA   NA
2  ATS151   NA   NA   NA   NA   NA   NA 2006   NA   NA
3  ATS153   NA   NA   NA   NA 2004   NA   NA   NA   NA
4  ATS155   NA 2001   NA   NA   NA 2005   NA 2009 2010
5  ATS156   NA   NA   NA   NA   NA   NA 2006   NA   NA
6  ATS157 2000   NA   NA   NA   NA   NA   NA   NA   NA
```

OK, so how do we fix the problem? First, we'll change all the non-zero encounter 'years' to `1`'s, and all the `NA`'s to `0`'s.

Then, we have the script determine which 'occasions' are missing. We do this be generating a 'canonical' list of the years that should be in the file (2000 $\longrightarrow$ 2010), which we call '`occStr`':

```
occStr <- seq(min(data$Year),max(data$Year),1);  # vector of years you want
occStr <- as.character(occStr);                  # convert to character string
occStr <- c("Tag",occStr);
```

Now, the key step – we use the `setdiff` function to compare the canonical list of occasions (`occStr`), with the list of occasions that actually are in the data file (using `names(pivot)`). We then add the columns to the pivot dataframe, set them to zero (we could use 'dots', if desired), and then sort them

```
Missing <- setdiff(occStr, names(pivot))   # Find names of missing columns
pivot[Missing] <- 0                        # Add them, filled with '0's
pivot <- pivot[occStr]                     # sort in ordinal sequence
```

The resulting pivot dataframe now has columns for the missing occasions (years) 2007 and 2008 have been inserted, made all `0`'s, and then sorted in the correct ordinal order. Here are the first few rows:

```
> pivot
     Tag 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
  1  ATS150    1    0    1    1    0    0    0    0    0    0    0
  2  ATS151    0    0    0    0    0    0    1    0    0    0    0
  3  ATS153    0    0    0    0    1    0    0    0    0    0    0
  4  ATS155    0    1    0    0    0    1    0    0    0    1    1
  5  ATS156    0    0    0    0    0    0    1    0    0    0    0
  6  ATS157    1    0    0    0    0    0    0    0    0    0    0
```

Pretty slick, eh? Note that for the 'missing years', you can choose to either (i) fill them with `0`'s (as shown above), or (ii) fill the columns for 2007 and 2008 with 'dots' (`.`). For some data types (see '**`Help | Data Types`** menu option in **MARK**), the 'dot' is interpreted as a missing occasion, which can be somewhat more convenient that manually setting the encounter probability to `0` for those years.

The remaining lines in the script handle the final formatting (concatenating the encounter data into a contiguous string, adding a few bits like commented tags, and the '`1;`' frequency column, and finally outputting everything to the `.INP` file).

Of course, none of this is needed if you use the 'fake individual approach' noted earlier. As discussed in the preceding section on using pivot tables in Excel, you simply enter a 'fake' individual into your spreadsheet right at the start, before you begin 'reshaping' the dataframe. For this 'fake' individual, you simply add as many rows (where each row corresponds to a sampling occasion) as you have occasions over the whole data set (in this example, 11 occasions, from 2000 $\longrightarrow$ 2010).

So, for this example,

23	ATS166	2004
24	ATS167	2001
25	fake	2000
26	fake	2001
27	fake	2002
28	fake	2003
29	fake	2004
30	fake	2005
31	fake	2006
32	fake	2007
33	fake	2008
34	fake	2009
35	fake	2010

Why do this? By introducing this 'fake' individual, where there is an encounter record (i.e., row) for every possible year in the data set. If you think about it, this should ensure that when we 'pivot' the spreadsheet there is a column for all years.

This is exactly what we see – if you look at the `pivot` dataframe before the script starts 'looking' for missing occasions, you'll see that including the 'fake individual' has already accomplished what we needed – since all occasions (years) are now in the dataframe:

```
> pivot
      Tag 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
1  ATS150    1    0    1    1    0    0    0    0    0    0    0
2  ATS151    0    0    0    0    0    0    1    0    0    0    0
3  ATS153    0    0    0    0    1    0    0    0    0    0    0
4  ATS155    0    1    0    0    0    1    0    0    0    1    1
5  ATS156    0    0    0    0    0    0    1    0    0    0    0
6  ATS157    1    0    0    0    0    0    0    0    0    0    0
7  ATS158    1    0    0    0    0    0    1    0    0    0    0
8  ATS159    0    0    0    1    0    0    1    0    0    0    0
9  ATS160    0    0    0    0    0    0    1    0    0    0    0
10 ATS161    0    0    0    0    0    0    1    0    0    0    0
11 ATS164    0    0    0    1    0    0    1    0    0    0    0
12 ATS165    1    0    0    0    0    0    1    0    0    0    0
13 ATS166    0    0    0    0    1    0    0    0    0    0    0
14 ATS167    0    1    1    0    0    0    0    0    0    0    0
15   fake    1    1    1    1    1    1    1    1    1    1    1
```

All that remains is to format the data frame, by concatenating the encounter data into a contiguous string, adding a few bits like commented tags, and the '`1;`' frequency column, outputting everything to the `.INP` file, and then deleting the row corresponding to the 'fake individual'.

So, the **R** code to 'find and insert' missing occasions as shown in the script would only be needed if you didn't first introduce the 'fake individual' into the original spreadsheet. Use whichever approach you find easiest to understand, and modify.

Other data types

Here we will consider 2 other data types, the robust design, and multi-state. Clearly, there are more data types in **MARK**, but these two represent very common data types, and if you understand steps in formatting `.INP` files for these two data types, you'll more than likely be able to figure out other data types on your own.

Coming soon...