Lab 6 – Nest Survival Models and implementation with RMark

Note – the following was modified from a lab originally written and delivered by WFCB graduate student Joel Tebbenkamp. Any errors or omissions are most certainly Erik's fault.

Introduction

So far this semester we've conducted all of our demographic analyses using the graphical "point and click" interface of Program MARK. For most purposes, you'll find MARK to be a powerful and flexible software for doing mark-recapture and associated demographic analyses, and there's no real reason to stray away from it unless you decide to leave the world of maximum likelihood estimation and enter the realm of Bayesian inference. With that said, you may have found MARK's graphical interface a bit tedious, particularly once we started to get into more complex models where setting PIMs or building design matrices became more cumbersome. I would argue that knowing how to build models using the PIMs and DM in MARK, and how they relate to your underlying data and ultimate results, goes a long way to truly understanding the analyses you are using (and thus hopefully ensuring you don't screw something up!). For that reason learning to use MARK the old fashioned way is, again in my opinion, extremely useful. But, you will likely decide that you'd like a more streamlined approach to analyzing your data, and for that we have the R package RMark available to us.

RMark was developed by Jeff Laake, a statistician with NOAA's National Marine Mammal Lab. The main purpose of RMark is to take all of the button clicking options available in MARK, and to translate those into a code-based framework implemented through R. When we use RMark, we aren't changing anything about how we're analyzing data compared to a regular old MARK analysis. We're just using a different interface to construct and compare models. The Program MARK executable file is still doing all the heavy lifting behind the scenes, even if the graphical interface doesn't actually open .

As with everything MARK, there is substantial documentation available to support self-teaching, and within the Gentle Introduction you'll find in Appendix C 100+ pages of detailed information related to using RMark for conducting analyses. You can also look to the package vignette for RMark through the CRAN website (http://cran.r-project.org/web/packages/RMark/RMark.pdf), and there is a support forum dedicated solely to RMark on the phidot web page.

Today's exercises are designed to provide a thorough introduction to use of RMark, and we will continue to use RMark during subsequent labs. There are both advantages and disadvantages to using RMark, and some but not all of these are highlighted on the first page of Appendix C). Overall though I think there are clear benefits to doing all your data-related work, from data management through analysis and graphing, and so I encourage you to use RMark as you develop your independent projects.

Today's Analysis:

We minimally introduced RMark last week, and today we'll give it a more thorough treatment using the nest survival analysis. This analysis type is most commonly used, as the name implies, to evaluate the daily nest survival of monitored bird nests, although it can be extended to other types of data where the fate of a marked or monitored animal is known, and where the ability to detect fate = 1.0 (see Blomberg et al. 2014 Ecology and Evolution 4488-4499 for an alternative application). Normally the temporal units in a nest survival analysis are days, and survival is expressed as the daily nest survival rate, or DNS. We can model temporal, spatial, and individual variation in DNS just as we've done for apparent survival in a CJS analysis, and ultimately we can use DNS to calculate the overall probability of nest survival (which should approximate nest success) as:

$$NS = DNS^E$$

Where "E" is the nest exposure period, or the length of time an average nest is at risk of failure. If we've modelled temporal variation in DNS across the exposure period, then NS can be computed as:

$$NS = \prod_{i=1}^{E} DNS$$

Where the product of all DNS across the nest exposure period gives us the nest survival probability. These principles were pioneered by the work of Mayfield in the 1940s (e.g. Wilson Bulletin 87:456-466), and they have since been extended to the more comprehensive analytical process that we implement in MARK and other applications. For a comprehensive description of modern practices in the study of nest survival, see Dinsmore et al. 2002 (Ecology 83:3476-3488), Shaffer 2004 (Auk 121:526-540), and Rotella et al. 2004. (Animal Biodiversity and Conservation 27:187-204).

For today's lab we'll be working with nest monitoring data from greater sage-grouse in eastern Nevada, collected as part of Erik's dissertation research. This dataset was collected from 2003-2012 as part of a 10-year study of sage-grouse response to a high-capacity power transmission line. Female sage-grouse typically lay about an 8-egg clutch over the course of about 10 days, and incubate for 27 days, so we commonly use 37 days as the nest exposure period.

The data will include two grouping variables, one for the year that the study was conducted (Year) and one that defines hen age (Age) as Juvenile (first nesting season), Adult (second or later nesting season) and Unknown (age was not or could not be established.

Also included in this data file are a number of individual covariates that describe the composition of sage-grouse nesting habitat, as measured during the study. These variables are generally of two types: site-level variables measured directly on the ground at each nest site, and landscape-scale variables measured using GIS. For lab today we've distilled the large number of variables that were collected during the study (>25) down to a more manageable handful for the purpose of our exercise. These variables include:

DLek – the distance (m) of a nest from the nearest sage-grouse breeding lek.

Sage – The proportional cover of sagebrush (*Artemisia* spp.) shrubs within 100m^2 of a sage-grouse nest, measured using line intercept transects.

NonSage - The proportional cover of shrubs other than sagebrush within 100m^2 of a sage-grouse nest, measured using line intercept transects.

RoadD500 – The density of roads within 500 m of a nest.

All1000 – the proportional coverage of sagebrush landcover within 1000 m of a nest.

NMG_Ht – the average height of grasses within 1 m^2 of the nest.

AvgGrsHt100 – the average height of grasses within 100 m^2 of the nest.

Date – Ordinal date expressed as day of the year, where Jan. 1 = Day 1.

The last variable is not a habitat variable per se but will allow us to account for potential within-year variable in the nest survival data. This data set is organized slightly different than a normal nest survival analysis. Many researchers define their survival history according to the day of the nesting season, where day 1 reflects the first day a nest is found, and days count up until the last nest is active. So if you found a nest on May 1st, and the last nest hatched on June 30, your nest survival history would consist of 61 days. A nest found on June 2nd would be found on day 33 (important for reasons that will be apparent later). In the analysis we're working with today, we've defined the history based on the maximum length of nest exposure, which was 44 days, and the day the nest is found is synonymous with its age (days past laying of the first egg). This is important to keep in mind and will introduce some subtle differences in terms of how we interpret results relative to other studies that rely on day of the nesting season to define a survival history (described below).

For more information about these data, see *Gibson et al. 2015. Auk 132:397-407 or Gibson et al. 2016 Condor 118:689-702.*

Part 1. First steps:

Note – The text below includes both step-by-step instructions and code that can be copied and pasted into RStudio. The code is highlighted in italics and indented, whereas the instructions are in standard font. But because I haven't used "#" to close out the instructional text, you can't copy and paste the whole

thing into R; it will be read it as code. So be sure to only copy italicized text. I've also included all code as an annotated .R file on Blackboard; you can choose for yourself whether you'd like to copy/paste or work with the existing script.

1.1. Getting Started.

First, open RStudio and install and load the "RMark" and "rgl" packages.

> *#install.packages("RMark")*
>
> *#install.packages("rgl")*
>
> *library(RMark)*
>
> *library(rgl)*

Though it shouldn't be a problem on the lab computers it is important to keep in mind that when using RMark you must also have program MARK installed in your computer, because as noted above when running models RMark is actually calling program MARK to do the work. You will find two different data files on Blackboard. SAGRNest.csv is the data file we will use in RMark, and SAGRMark.inp is the same data in input file format that could be run with the conventional Mark program. I've provided them both so you can compare them to each other. Each of these files contain certain information required for a nest survival model that must appear in the correct order, and in the case of RMark, have the proper column headings. These include (in order and with appropriate column heading):

> **ID** (the identifier of a nest)
>
> **FirstFound** (the date on which the nest was first discovered)
>
> **LastPresent** (the date the nest was last observed to be active)
>
> **LastCheched** (the last date the nest was checked)
>
> **Fate** – 0 for successful and 1 for unsuccessful
>
> **Freq** (how many nests share this history – typically 1)

Each file also contains a number of individual covariates we will be including in our analysis and are described below. Otherwise there are a few formatting differences between the files. The inp file lacks column headings, and uses the "/**/" convention around the ID term to signify that the ID value for each bird is not part of the encounter history. You should also notice that year is labeled as a group in the inp file, while in the csv data file I've included it as a covariate value. When running nest models with multiple years of data using MARK, year must be considered a group, which is true of all categorical variables with greater than 3 levels when implemented in MARK. Implementing group structure in RMark is a bit more straightforward, and can be accommodated with covariate columns specified as factors.

First, we'll read in the text file and give it a name. I like using the ".df" in the name to remind me that it is a data frame.

```
nest.df= read.table("C:/Users/Joel T/Documents/RMark_Class/GSG_NV1.txt", header=T)
```

Check the structure of the variables within the data frame.

```
str(nest.df)
```

There are 2 variables in the data frame that should be listed as factors. Those are "Year" and "HenAge" You should notice that Age is already listed as a factor variable since the information the data frame consisted of character strings, and R was able to interpret this. However, year will be listed as a number when it should be a factor, so we'll need to change it. There are several ways of doing this but an easy way is as follows:

```
nest.df$Year<- factor(nest.df$Year)
```

Check the structure again to make sure it worked using the str() command. Let's also check a summary of the data using the summary() command. For factor variables it will provide the name and the number of occurrences within each level. For the numeric (continuous) variables it will return descriptive statistics.

```
summary(nest.df)
```

Once the variables are correctly specified as either numeric or factors, the first step to setting up an analysis is to process the data to create a file within R that MARK can work with when RMark executes comments. This is essentially the same as what we do when we begin a new MARK file, but here executed with a command statement.

```
ns1.process=process.data(nest.df, model="Nest", nocc=44, groups=c("Year", "HenAge"))
```

This is a fairly bare-bones command that tells mark we want to use a nest survival model, with 44 occasions and 2 grouping variables. You'll also notice we did not need to specify covariate values as RMark will read these directly from the data file. The following are some of the arguments that can be included when processing the data:

model: the type of analysis model (e.g., "CJS", "Known", "POPAN", etc.);

begin.time: the time of the first capture/release occasion for labeling

time.intervals: the lengths of the time intervals between capture occasions

groups: the list of factor variables in the data to define group

initial.ages: the age of animals at first capture/release corresponding to the levels of the age grouping variable (**age.var**)

nocc: number of capture/encounter occasions

As is clear from the data that was just processed it is not necessary to define all of these different attributes, just as it's not necessary to use all the options every time you begin an analysis in MARK. But, you will probably always need to specify the model type and number of capture occasions (nocc). If you have any categorical variables it is also necessary to identify these as groups. This is a step where, if you are running a different analysis on your own, you'd need to do enough background reading to determine which of these options were necessary and important for your particular analysis.

Our next step will use the processed data to create what is known as design data (often noted with a ".ddl")

ns1.ddl<-make.design.data(ns1.process)

The design data provides RMark with critical information related to data structure, such as the time structure, group membership, and age or cohort structure, which allows RMark to build an appropriate design matrix for a specified model when calling MARK. Design data are parameter-specific; if we were modelling phi and p, as in the CJS, then we'd define design data for each parameter included in the analysis. This step is effectively the same as setting the PIMs in MARK, and in the case of this simple nest survival model we don't need to do anything more complex when setting the PIM structure.

1.2. Running models and viewing results

Now that we've set our basic model parameters, we can run a model. To do this, use the mark() command and specify the processed data (ns1.process), the design data (ns1.ddl), and the RMark function "model.parameters." In RMark we can specify the null model using a "~1" notation, which is general R-speak for an intercept-only model, i.e. the dot model. So the notation to run a dot model from our data is as follows:

S.dot<-mark(ns1.process, ns1.ddl, model.parameters=list(S=list(formula=~1)))

When you run this command you'll get an instant printout of the results for all the models. You can also use the *output = FALSE* option to suppress that printout, which you will probably want to do once we move towards running comprehensive model sets where it becomes tedious to have dozens of results flashing through the screen. Because this is the null model there isn't too much of interest here, but one thing it does tell us is that the daily survival probability across all nests was 0.96049.

Take a look at the PIM structure for this model using the following code. The "S" denotes that we are looking at the PIM's that correspond to survival probability value, the only parameter in a nest survival model. If we were running an analysis with multiple parameters (e.g. Ψ, Φ, and p), specifying the parameter would be more important and necessary to view parameter-specific PIMs.

```
PIMS(S.dot, "S", simplified=T)
```

You should see just a whole bunch of ones repeated across a large number of groups, just as we would see in MARK. To export the results to a text window simply type in the name of model, which will return a notepad output similar to what you would see from MARK.

```
S.dot
```

There is a lot of information on the page but the most relevant information will be at the bottom which includes the AIC value, deviance, etc. as well as the real and beta estimates. Notice that in MARK if you right click on a model and click on the "Output in Notepad" option, you'll get the same results. If you want to report only say the real parameter estimates, you can do that using

```
S.dot$results$real
```

Now we'll run through a few models where we apply different sources of variation on daily nest survival in the form of continues predictor variables. First, let's run a model that considered the effects of "Sage" on nest survival – this will test whether the overhead cover of sagebrush shrubs within 5m of the nest influence the daily survival rate.

```
S.Sage<-mark(ns1.process, ns1.ddl,
model.parameters=list(S=list(formula=~Sage)))
```

This is for all purposes the same code we ran above for the dot model, except that we've changed the model name and have specified ~Sage as the model formula instead of ~1. Check the PIMs for this model

```
PIMS(S.Sage, "S", simplified=T)
```

You should notice that the PIM's for this model look the same as the PIM's for the dot model. Think for a second why this would be the case? Recall that when fitting a covariate effect to a model, it really takes up only a single parameter, so in practice it can be fit to a model where the PIMs have been constrained to the dot structure. We typically did not bother constraining the PIMs as such in MARK, because within the Design Matrix we could apply the covariate across all parameter rows. But when RMark fits a covariate without a group or time structure, it will base in on the dot model PIMs. This should make no difference in the outcome of the model.

One other interpretation we can make from this model, without having gone through model selection as of yet, is to examine the beta coefficient for the sagebrush effect within the model printout, or by asking for those results specifically

```
S.Sage$results$beta
```

Where you should see that the 95% confidence intervals widely overlap 0.0, suggesting not much support for this effect.

Next let’s run a model that contains a “Year” effect on daily nest survival. Remember that Year is included in the dataset as a categorical grouping effect, and we’ve already defined it as a factor.

```
S.Year<-mark(ns1.process, ns1.ddl,
model.parameters=list(S=list(formula=~Year)))
```

Take a look at the PIM’s.

```
PIMS(S.Year, "S", simplified=T)
```

It will take a little scrolling up and down but you’ll notice the parameter index values are consistent among year groups (e.g. 2004 = 3, 2005 = 1, 2006 = 4); this is the same approach we would use in the PIM’s in MARK to test for general differences in nest success among years. So in this case RMark is building a group model by adjusting the PIMs, just as we’d have done during a regular Mark analysis. Let’s finish up by running one more model where we test the effect of hen age, again as a grouping variable.

```
S.Age<-mark(ns1.process, ns1.ddl,
model.parameters=list(S=list(formula=~HenAge)))
```

Now let’s look at an AIC table of our model results so far, which will take 2 steps. First, you need compile the models you want to compare into a single object (in this case named “GSG.results”) using the “collect.models” command.

```
GSG.results<- collect.models()
```

To view the AIC table type in the name of the object that you just created.

```
GSG.results
```

As a HUGE caveat here, collect.models() will grab all mark model objects that are present in your R environment, regardless of whether they are part of this analysis or not. The recommendation of the RMark authors is to begin and save each new analysis you conduct as its own .Rdata file, which will limit the number of concurrent R objects you have open at a given time. We will see an alternative approach to building models shortly that will also alleviate this issue.

1.3. Additive and interactive models

Once you’ve got running single variable models down, considering additive effects of multiple variables, or interactions among two variables, is quite simple. For an additive model with the variables “Dlek” and “All1000” the code would be as follows:

```
S.Dlek.and.All1000<- mark(ns1.process, ns1.ddl,
model.parameters=list(S=list(formula=~Dlek+All1000)))
```

For naming the model I used “.and.” to indicate that it was an additive model. For the formula all I had to do was add a “+” between the 2 variables. Similarly, if I wanted to use the same 2 variables to produce a model with an interaction, I would use a ”*”. Pretty simple, just make sure you name it something different… I use an “.X.”

*S.Dlek.X.All1000<-mark(ns1.process, ns1.ddl, model.parameters=list(S=list(formula=~Dlek*All1000)))*

When considering time structure, you can either look at time trends or produce separate, independent, survival estimates for each day of the nesting season. To denote a time trend use the word “Time” with a capital “T” as the model variable. To estimate separate survival rates for each day use the word “time” with a lowercase ”t.”

The 2 models would be written as follows, here for the trend model:

S.Trend<-mark(ns1.process, ns1.ddl, model.parameters=list(S=list(formula=~Time)))

And here for the full daily survival probability:

S.Day<-mark(ns1.process, ns1.ddl, model.parameters=list(S=list(formula=~time)))

You may have gotten a red error message when you ran this model. If so, it indicated that a number of parameters that were in the model were not identified as estimable parameters and therefore would not influence the penalty term in AICc. In this case, it adjusted the number of parameters to appropriately reflect the model. However, examples like this help to emphasize the importance of identifying the number parameters that should be included in each model (by hand or in your head) and make sure the number of parameters listed is the same that should be present. Lack of estimation here results from the fact that even with over 300 nests there isn’t enough data to estimate nest survival independently across all 44 days.

Also of importance, because this model is not converging on all parameters we should be cautious when making any sorts of interpretations from it. One word of caution I would give you once you start running code-based models, which can be cranked through quite quickly, is to not neglect your due diligence of scrutinizing your model and model selection results.

1.4. Quadratic effects

With any continuous covariate, including a linear Time trend, we can add a quadratic term to reflect a nonlinear relationship. In R a quadratic relationship can be included using the following formula notation: “VariableX + I(VariableX^2)” For modeling a quadratic effect of Time, the code would look like the following:

```
S.Time2<-mark(ns1.process, ns1.ddl,
model.parameters=list(formula=~Time+I(Time^2))
```

1.5. Considering additional temporal constraints

Recall that our history is defined by nest age, rather than day of the nesting season. When data are organized like this there are some calls that you can implement in RMark to look at temporal variance in nest survival, and given the way these data are organized "time" is synonymous with nest age. For example, we might hypothesize that daily nest survival differs between the laying phase and incubation phase. During laying, the female is only occasionally on the nest to lay an egg and then leaves. During incubation females are on the nest for most of the day. For greater sage grouse it is assumed that on average laying takes 10 days. Therefore, we can classify day 1 through 10 as laying in day 11 onward as incubation. To achieve this we need to go back to the design data that we made in the beginning and make some modifications to include a new grouping variable that partitions our daily history into laying and incubation periods. The following code achieves this.

```
ns1.ddl<-add.design.data(ns1.process, ns1.ddl, parameter="S", type="time",
bins=c(1,11,44), name="NestStage", right=F, replace=T)
```

Because we do not want to make new design data, but just modify the current design data, we can use the add.design.data() command. There are several pieces of information one needs to include with this command

Name of the processed data

Name of the design data - note: this is the same as the name of the object you are creating/modifying

The **parameter for which the design data corresponds** - e.g. S, Phi, p

Parameter type - does the modification apply to "age", "time", or "cohort"

bins defines the constraints you're putting on the parameter, i.e. which time intervals should be considered similar or different. In this case, we are saying that occasions 1 through 10 will be considered different from 11 through 44.

name the name assigned to the variable in design data. This will be the new variable name that will be applied in subsequent model statements.

Replace if TRUE, replace any variable with same name as name

Right If TRUE, bin intervals are closed on the right

Check a summary of the newly modified design data after we've incorporated the new variable "NestStage."

summary(ns1.ddl$S$NestStage)

The top line of the resulting output shows how the capture history is divided in order to create a stage for laying and a stage for incubation. A ")" means the interval is open on the right which infers that value is not included in the interval. Whereas a square bracket ("[" or "]") is for a closed interval which means the interval end point is included. So in this case the first interval includes days 1 through 10, and the second interval days 11 through 44. The 2nd line provides the count of the number of unique paramters (group by day combinations) within each time block.

Now let's run a model testing the effect of the nesting stage.

S.NestStage<-mark(ns1.process, ns1.ddl, model.parameters=list(S=list(formula=~NestStage)))

Checking the results from this model, you should see that it is applying a different daily survival probability depending on the day within the history. This technique was useful in this specific case for evaluating differences in laying stages, but it could be used in any case where you want to group time intervals. The example we used earlier in MARK was to run the Phi(Flood) model for the dipper data.

Now let's generate another AIC table with all the results we've generated so far.

GSG.results<-collect.models()

GSG.results

Part 2. Using a function to efficiently analyze a comprehensive model set:

When 1st learning how RMark works it is useful to write out the formulas for each model in order to get a sense of how the processed data and design data are being used to create the specified model, and to evaluate results as you get them to make sure your model statements are doing what you think they are doing. However, there is a more organized and efficient way to create a model set that can be achieved by using a function in R.

When using R if you reuse the name of an object it will generally replace it with whatever the most recently defined object contains. To avoid replacing all of the models and objects we've already created, let's re-process the data and make new design data.

ns.process<-process.data(nest.df, model="Nest", nocc=44, groups=c("Year", "HenAge"))

ns.ddl<-make.design.data(ns.process)

First, we need to name the function that will build and run a full suite of models. The information contained between the "{ and }" is the function that will run when you call its name. The following is pretty long but includes annotation throughout to describe what is being done,

```
nest.models=function ()

{

## Null model

S.dot=list(formula=~1)

##Note what pieces of information are included/excluded when writing this

##formula compared to what you did previously

## single-variable models

S.Year=list(formula=~Year)

S.Date=list(formula=~Date)

S.HenAge=list(formula=~HenAge)

S.Dlek=list(formula=~Dlek)

S.Sage=list(formula=~Sage)

S.NonSage=list(formula=~NonSage)

S.RoadD500=list(formula=~RoadD500)

S.All1000=list(formula=~All1000)

S.NMG_Ht=list(formula=~NMG_Ht)

S.AvgGrsHt100=list(formula=~AvgGrsHt100)

## end of single-variable models


##Time-varying Models

S.Time=list(formula=~Time)

S.Time2=list(formula=~Time+I(Time^2))

S.time=list(formula=~time)


##The following provides information that applies to all models in this function
## Therefore, it was not necessary to add this information to each model as done
## previously when running one model at a time
cml=create.model.list("Nest")
  results=mark.wrapper(cml, data=ns.process,)
  return(results)
} ## end function
```

Now we'll designate an object to store the results (nest.results) and run the function "nest.models()."

```
nest.results<- nest.models()
```

View the AIC table.

```
nest.results
```

Using this function streamlines writing out all of the models, however, in order to view the results of an individual model there is now an extra step. Since all of the models are contained within the "nest.reults" object is necessary to use the "$" to retrieve the desired model that is contained within "nest.reults." For example, to view the results of "S.Dlek" in a text window:

```
nest.results$S.NonSage
```

Or just the beta coefficients

```
nest.results$S.NonSage$results$beta
```

Lastly, we can generate model-averaged parameter estimates from our model set using the model.average() function.

```
Avg.est<- model.average(nest.results)
```

This will give a print out of all 989 possible parameters (days*years*ages) results here, so there is a lot of redundancy and, because age, day, and year were not supported in model selection the variability among the estimates are really not that interesting. So, lets extract just the first 44 values so that we can estimate the overall nest success probability for a typical nesting attempt.

```
Avg.est.44<- Avg.est[1:44, ]
```

Since the mean nesting interval is 37 days long, we can calculate nest success as the product of the first 37 probability values, which are contained in the second column

```
NS.37<- product(Avg.est.44[1:44, 2])
NS.37
```

This tells us that the expected nest success rate, based on our model-averaged parameter estimates, should be approximately 17.3 percent

Part 3. Graphing model results

Once we've got our MARK results contained conveniently within R, we can take advantage of R's superior graphing capabilities to present the results for papers, presentations, etc. What follows is a fairly straightforward graphing exercise to show the relationship between non-sagebrush shrub cover at nests and predicted daily nest survival rates using the ggplot package.

To create a graph the 1st order of business is specify the values of NonSage for which we want to compute the real parameter estimates. This will provide the range of x-axis values for our graph. This is simple enough to do using the seq() function

```
NonSage.values<-seq(-0.64, 6.24, by=0.2)
```

Where recall that our covariates are z-standardized, hence the negative starting values. Coincidentally, we can improve the code above by pulling the starting and ending values directly from our data, removing the need to look up the min and max values ourselves.

```
NonSage.values<-seq(min(nest.df$NonSage),max(nest.df$NonSage), by=0.2)
```

The covariate.predictions() function in RMark creates predictions from Mark models, and in this application is most similar to the covariate graphing feature (and its output) from MARK. It will produced model-averaged survival estimates from a model list, or estimates from a single model if that is specified. It also requires that you input a range of covariate values if the model contains a covariate, in this case "NonSage."

```
DSRbyNonSage<-covariate.predictions(nest.results$S.NonSage,
data=data.frame(NonSage=NonSage.values), indices=c(1))
```

Take a look at the data to see what was produced. This output contains two different elements, the first being an output with the real estimates, SE, and CIs plus some other associated values for each level of the values specified in the data= statement above. The second component is the variance-covariance matrix.

```
DSRbyNonSage
```

This output has a bunch of superfluous information we don't exactly need, so let's extract just the estimates, SE, and CIs from these results and append into them into their own data.frame() that we can work with. Because the output contains both the estimates and the vcv matrix, the $estimates component is a necessary addition to the statement

```
NonSage.pred<- data.frame(DSRbyNonSage$estimates[,5:8])
```

From here it's simple to append the z-transformed values to the data frame, which is facilitated by the fact that the predictions were generated from NonSage.values, so the ordering is the same.

```
NonSage.pred$NonSage.z<- NonSage.values
```

And lastly we need to calculate the back-transformed real values from the Z-standardized values so that we can put the actual, rather than transformed, values on the x axis of our graph. The values here are SD=0.104026872 and X-bar=0.16546597 for NonSage, so given the backtransformation:

```
NonSage.pred$NonSage<-
NonSage.pred$NonSage.z*0.104026872+0.16546597
```

From here we can create a sexy plot showing the effect of non-sage cover on the daily nest survival rate using ggplot package. As mentioned in last week's lab, there is a ggplot tutorial posted on blackboard, and you can find substantial supporting documentation for the package online. I really like it for its graphing capacities, but you could also accomplish this with base R graphing.

```
library(ggplot2)

Nonsage.plot<- ggplot(data=NonSage.pred, aes(x=NonSage, y=estimate))+
  geom_ribbon(aes(ymin=lcl, ymax=ucl), fill= "gray")+
  geom_line() +
  ylab ("Daily Nest Survival") +
  xlab ("Non-Sagebrush Shrub Cover")+
  theme(text= element_text(size=16))

Nonsage.plot
```

Let's take a quick second to appreciate what we just accomplished in R, relative to the steps that would be required from MARK. First, we built and ran a set of models using a streamlined function which took ~10 seconds to run. Then, we extracted results from those models in the form of AIC tables, real estimates and beta coefficients, and also generated a predictive graph of one covariate effect. In MARK itself, even once you become very proficient at model building, this would have consumed maybe, say, and hour of your time pointing and clicking. AND, importantly, doing it over again would have taken another hour of your time. With RMark you might spend and equivalent, or even greater amount of time writing and trouble-shooting your code the first time around. HOWEVER, if you ever need to revise your analysis (and you WILL need to revise your analysis, probably multiple times) it becomes a trivial exercise in highlighting your code and hitting run again. So there is clearly a great deal of efficiency to be gained in working with RMark.

Lab 6 Assignment

On Blackboard you will find a file labelled MALLNest.csv, which contains Mallard nesting data provided as part of the MARK supporting data files. This file contains mallard nest monitoring data from North Dakota during the year 2000. Nests are grouped by cover types, which include roadsides (i.e. ditches), setlands, planted areas (i.e. row crops), and native grassland cover. Also provided are covariates that describe a Robel Pole reading, which is a measure of vertical vegetative cover at the nest (larger value equals more vegetation and hence better cover), the proportion of the surrounding landscape comprised of grasslands, and the age of the nest, in days, when it was found. For this last variable, 0 would reflect a nest found at the beginning of incubation, and a negative number reflects a nest found during laying. This analysis defines the history based on the day of the overall nesting season, rather than the age of the nest.

From these data, you should be able to conduct an RMark analysis that evaluates support for each of the covariates, the group effect, and I would also like to know if nest survival changed throughout the course of the nesting season. Please include your R script and a word document with the AIC table and the **model-averaged** parameter estimates.