ADD IN THE EMAILS previous to this date to this list.

**2014-02-07**

(a) Version control. Put the date at the END of the file name (as a suffix as suggested in my previous email) so that all the versions of the code sort together for each file.

(b) Put the old versions in a separate sub-directory (as suggested in my previous email).

(c) In the neg.log.like function why do you parm=parm at the start of the code?

You don't need to accumulate the capture history counts into the unique capture histories (and certainly don't want to do this when you have individual covariates). There is no need to get the counts.in.order variable or anything like that. Just use the history vector directly with the associated probability etc.  Again read the MARK manual -- many studies have capture histories for INDIVIDUAL fish so there is NO need to accumulate these and it makes no sense to do this when you have individual covariates (which is part of this project)

(d) You keep changing the names of the variables. For example, you use dat, Data, for data; est, parm, par for the parameters. Use the SAME names consistently through the code. It is not necessary to have different names in different routines. This will get you confused when debugging your program trying to keep track of the various names. FOr example, use names such as

   parm  - the list of expanded parameters. This can be used to hold the initial estimates as well.

   parm.beta - the list of packed estimates on the thelogit/log scale (what MARK calls the beta parameters). This can be used to hold the packed initial estimates.

Again read the MARK manuals for details on how to structure code and use consistent variable names.

(e) In the fit.model.R code

  datalist = as.list(datacsv)

Corrected the things in this page

  Data = NULL

datalist is already a data frame which is a list. There is no need to change it to list?

Don't use the [[1]] and refer to items by name. The use of [[x]] will get you in problems as the order of items in a list is not guaranteed.

(f) initial.estimates() will need the data passed to it to create the initial estimates. You don't currently use the data, but write the code keeping in mind what it will look like in the future.

Corrected

(g) You use .01 and .99 as the lower and upper bounds, but you will be optimizing on the logit and log() scale. So these are NOT the correct bounds. Your bounds for N again should be on the log scale and not the original scale.

YOU need to change the optimization function to go from the PACKED estimates to the unpackaged estimates because you eventually will have to introduce design matrices.

(h) THefit.model needs to be a FUNCTION and not naked code. See my examples in the material that I send you before.

(i) You have expit() and logit() defined in multiple places. Put all of these helper function in a separate file that is sourced as needed. THis is not maintainable. You will eventually have to modify these functions to deal with logit(0) etc, so you want a single copy of each function.

Create a file to load all the necessary libraries and the R code similar to the SPAS.load.r function in the material that I sent you.

(j) Be consistent in the use of white space in the R code. Sometimes you have 4 blank lines, other times 2 blank links, without any apparent reasons. The filename for the file with the function in should have the same name as the function (e.g. UNPACK or PACK) etc.

(k) Put each function in a SEPARATE file (except for the helper functions such as logit, expit, etc). For example, where is the initial.estimates() function?

Corrected

(l) Be consistent how you will name functions. Some are UPPER case and some are lower case. Some have . (periods) other use underscores etc.

You need to spend more time thinking about the structure of your code before you start coding. Keep in mind what the final product will look like and how you will need to maintain the code. This is PRODUCTION code that will be released to interested readers so it has to look good and be correct.

**2014-02-26**

A similar delta method was done in the SPAS program in the fit-model routine.

Note that you should keep ALL of the parameters in each expansion, i.e. the VCV should be for the entire set of parameters and only at the final step, should you pull out the individual se etc.

You will need the full VCV to compute the estimates of the population size in each category, i.e. N-hat\_males and N-hat\_females, etc.

Completed

**2014-03-20**

You should create a function to generate the expected counts give the parameter set. You already have a function to compute the probabilities of every possible history, so it shouldn't be too difficult.

Then generate the expected counts for a set of parameter values. Pass these EXPECTED counts (with the decimal value for the count, i.e. do NOT round the counts to integers) to your function. You should get back the parameter values exactly.  This will be a way to test your program as well.

You can also use this method to estimate approximate bias and precision. See the paper by Devineau

Completed

**2014-03-28**

As the first step towards the optimization, you will need a planing tool.

Input (as a list)

   N - guesstimates of pop size

   lambda - guesstimates of distribution of population

   n1  - number selected at time 1 of which n1\* are "sexed"

   n2 - number selected at time 2 (in total) of which n2\* are "sexed"

   r1 - guestimate of ratio of p1m/p1f

   r2 - guestimate of ratio of p2m/p2f

From these you can "estimate" the p1m p1f, p2m p2f etc. For example give N, lambda, n1 and r1 you can figure out what p1m and p1f are etc.

Generate expected counts using the full parameter set

Estimate using expected counts

Report on Estimate, bias, and expected se.

Completed

**2014-05-14**

Here are comments on the current paper. Remember, you are writing for a Biometrics audience so you don't need all of the "simple" detains such as what is a score function, what is the likelihood function etc.

Try and remove redundant and vacuous sentences which say nothing. Every paragraph must have topic sentence as the first sentence and the rest of the paragraph must support that topic sentence. Journal space is a premium so every word must count and must be needed.

Did you get JAGS to work for the simple Petersen model? Remember you need to use the capture histories. You may have to "preprocess" the histories to make your life easier, i.e. convert alphanumeric category codes to numeric codes to make indexing easier in JAGS. Your simple Petersen model in JAGS should also include design matrices and offsets so you understand what needs to be done when you extend this to your model.

**2014-05-21**

A couple of other notes

- theDropBox should be your "official" directory for this project. You should not be copying code back and forth as that just leads to problems where code is not consistent over time. You need to develop good versioning skills for your intermediate steps, e.g. time stamp versions of papers, etc

- I also noticed that you "hard coded" the priors in the Petersen example. The alpha/beta for the beta distribution should be passed as data.

**2014-06-02**

* Debug the previous code

Debugged following files

Initial.estimates.

neg.log.likelihood

planning.tool

print.output

get.data

expected.counts

unpack.param

expected.counts

fit.model

model.comparision.table

get.data function should return silently (i.e. don' d o a str() inside the function).

Do the str() after the call.

[model.id](http://model.id/) should use the notation you used for the paper along the lines of LeBreton et al 1992 rather than the "wordy" description.

Why do you have code similar to

    observed\_count = Data$counts

    observed\_count

to list the observed counts? Just say Data$counts to list the observed counts.

Corrected the things in this page

When you fit the model with the sex ratio fixed, you used

lambdaOFFSET=c(logit(0.3298394))

Why did you hard code this? You should refer directly to the estimates returned in the previous model. Avoid hard coding most things in R.

Same think in the [model.id](http://model.id/). Use the paste() command to build the [model.id](http://model.id/)

The print.output() function should also list the design matrices and offsets.

"Initial values capture probabilities for optimization: rows are categories and columns are sampling occasions". You should attach row names to the matrices with the category labels rather than forcing people to refer back to the category labels. Ditto for MLEs and SE of MLEs"Initial category proportions for optimization: " Ditto - add names to the values corresponding to the category labels.

Use options(width=200) so that the html notebook doesn't wrap the output

Model comparison table. You also need the AICc weights.

\*\*\*\*\* print.output() function

 for(i in 1:ncats){

    cat("MLE for Population size of category " , x$rawdata$category[i], " : " ,round(x$est$Nc[i]))

    cat("\n")

  }

For loops not needed here. Use paste to create the output for all categories directly. for loops are rarely needed in R.

creat.DM.r file. Fix the name to create.DM.r. Try and be consistent between file names and the functions that they contain. This will reduce the number of errors you will make.

Corrected the things in this page

\*\*\*\*\* expected.counts() function.

Again, avoid "for" loops. Use paste to create the output directly.

  for (i in 1:length(History)){

    cat(" Expected count for the capture History ", History[i], " : " , expected.counts[i], "\n" )

  }

\*\*\*\*\* neg.log,likelihood function

- fix spelling errors, e.g.  likeligoodvelue - it makes your code look unprofessional

- why do need the neg.lg.ld function separate?

\*\*\*\*\* unpack.parm() function

- eight separate special cases! Really! This is ugly and the hard way to do it. Recode this function so that it runs through the parameters sequentially and deals with design matrices with 0 columns in a nice fashion as they are encountered. What will you do if you have 5 different design matrices. Would you code all 2\*\*5 combinations!

Corrected the things in this page

**2014-06-03**

As an "approximate" solution, combine the U0 and UU histories into a U\* history. The MSS then reduces to one statistic/parameter (don't forget that the MSS must sum to n which also reduces the number of statistics).

You should now try and find explicit solutions using this "reduced" data.

Do this after you get the optimization working when I'm away until mid-July.

I notice that you are still using the base R plot() and lapply() commands. These should not be used any more -- the ggplot2 package provides much superior plotting facilities and the plyr() package is preferred over the apply() family.

Why did you choose the alabama package? It doesn't look like it is being maintained (last revision in 2011) and there are equivalent packages in the stat and Rsolvnp packages.

Because you have only a single linear (cost) constraint, you don't have to minimize the variance over all 4 variables. Once you have n1, n1.star, n2 chosen, the value of n2.star can be solved for to match the cost exactly.

This will eliminate one of the loops in your code for the contour plots. You can likely now remove all the for loops and replace them by a expand.grid(on n1, n1.star, n2), solve for n2\*star and then compute the variance of N at each entry in the grid.

Similarly, when using the optimizer directly, set the design matrix to have only 3 parameter (n1, n1\*star, n2). Solve for n2star directly in the function before finding the variance. You will have to deal with cases where the n2star is not feasible (likely set it to the boundary which will give a higher than expected variances so the optimization routines will likely choose another solution).

This means that your problem is considerably easier to solve as the routines don't have to numerically constrain the 4 n values to match the cost function. This equality constraint makes the problem numerically much more difficult to solve.

Your optimal.allocation() function has the cost values passed to it, but never uses it. It actually doesn't do any optimal allocation - so you should change the name to something more suitable.

Your planning.tool.for.optimization() function extract the possible capture histories from the Data. This is incorrect.  You need to make a list of all the possible capture histories rather than relying on the observed data otherwise you may miss some histories.

Corrected the things in this page

**2014-06-04**

Hi Lasantha -- here are some things to work on  when I'm away over the next month.  If you have questions about what you will be doing over the next month, you need to ask me this week as I'm only here 1 day next week (Wednesday) before mid-july. I will have some email access, but at irregular times.  You may wish to create a "paper" with the activities below and questions/answers for each section so you don't forget.

These are standard actions that need to be done for any applied paper.  After you finish a section, add it to the writeup you have so far on the paper.

You will need a single file that runs though all of the analyses for the Millelacs example for publication purposes so you should try and generate a nice file. This should NOT include all of the steps used in testing/exercising your code which should be in a separate file.

 You will need to spend sufficient time each day on the various activities if you want this project to move forward in a timely fashion. You should likely reduce your TA activities next term to ensure that you have sufficient time to devote to the project - definitely no teaching!

 After the list below is complete, this is about 1/2 of the paper. Don't forget we next deal with the cases with individual covariates (e.g. length) measured on each fish in addition to the categories. These covariates also affect the catchability. Refer to the paper outline that I sent you way back when we started this project. We will work though a similar list of activities as the initial case with the individual covariates, i.e. a MLE and bayesian solution. Refer to several papers by Huggins on how to deal with individual covariates.

Planning

- get the optimization working;

- generate some nice contour plots. Use the design matrix approach to fix n1 and n2 and then find the optimal n1.star and n2.star values rather doing a large loop or grid search. You may need to smooth the contour plot to get nice contour lines.

- investigate the optimal allocation when the sex ratio is known.  Explain any changes from the above and if there are no changes explain why. Give intuitive justifications for the changes (if any)

- investigate the optimal allocation when the capture rates of males and females is equal either at time 1 or time 2 or both. Explain any changes from the above and if there are no changes explain why. Give intuitive justifications for the changes (if any)

- generate some data where the capture probabilities vary between sexes and/or sample times and then fit a model where the probabilities are equal. Use the Devineau paper method to estimate the power of your study to detect such differences. Verify the power using a simulation study.

Completed

Model assessment from the MLE model

You need to investigate the fit of the model to data.

- make residual plots similar to those produced by program MARK. You will have to consolidate the histories together in cases where the raw data is not already consolidated. Don't forget to deal with cases where the raw data is missing some histories. The residuals should  also be standardized.

- do a non-parametric bootstrap goodness of fit using the deviance from the saturated model as the measure of lack of fit. Again this is similar to what is done in program MARK. Create a histogram of the bootstrap deviances and show where the observed deviance fits. Also use the Tukey statistic as outlined in the Brooks paper below as well. Compute the p-value of the gof test for both statistics. Explain why one is preferable to the other and under what conditions each should be used.

Non- parametric bootstrap did not work. Did the parametric bootstrap

Tukey measure removes the need to pool small cells to avoid overweighting

Tukey statistic = sum( sqrt(observed) – sqrt(expected) ) \*\*2

- generate some data where the assumptions of the model are violated and see if the above two assessments capture the problem. [Typical problems are heterogeneity in catchability among the animals within a category; failure of the non-death assumptions; etc.

Generating data is complicated. If data generated assuming failure of non-death assumptions, the fog plot cannot capture the problem

- generate some data where the capture-probabilities vary between categories and/or time; fit a model where they are forced to be equal (similar to the power study). Show the residual and gof plots - what features of these plots show extent of the violations of assumptions.

Completed. Capture histories MM, FF,UU don’t have bigger residual compared to others. 0M, OF, 0U, M0, F0, U0 have higher residuals depend on capture probabilities on the categories on each occasion. For an example capture probability for category M is very small or very large compared to others, then 0M and M0 capture histories have very big residual.

Closed form solutions

- see my earlier email about pooling two of the histories and using the reduced set of statistics to come up with closed-form estimates.

Compare impact of unknown ability to sex fish

- see my earlier email about computing what would happen if you sex the fish exactly with the same total effort and you could do a proper fully-stratified experiment and analysis.

Obtained the closed form solutions and computed the estimates using fully stratified experiment. Fully stratification method gives almost the same estimates as partial stratification method.

Problem in computing the SE

Bayesian model

- implement the model in jages using the capture histories (and not the sufficient statistics).

- it should deal with more than 2 categories just like your program

- you will need to generate plots of the posterior distributions for the important parameters overlaid with the the prior distribution for the parameters. Assess the amount of overlap to see if the prior has too much weight. [read the paper <http://link.springer.com/chapter/10.1007/978-0-387-78151-8_48> and related papers for details.]

- generate bayesian p-values plots to assess model fit. See the paper by Steve Brooks et al in Statistical Science.

**2014-06-05**

For the tasks that I gave you when I'm away, I expect general functions for many of them. For example, for the power comparisons, you should write a function that takes two model specifications and find the power to detect the difference between the models; for the residual plot, write a function that takes the fitted model and creates the residual plot (as a ggplot object) that is returned; for the contour plot, take a model specification and a "centre point" (e.g. the optimal allocation) and then create and return the contour plot (as a ggplot object) and a list of the contour points and their n1, n1\*, n2, n2\* values etc.

Other changes that I noticed.

get.data()

- you require the histories to be in a specific order. Why? What section of the code requires this? A sample mean doesn't depend on the order of the data, so why should these routines?

expected.counts()

- why does this generate the expected values ONLY for the observed history. We went through this in another function - ALL of the histories should be generated here. It should NOT depend on the raw data.

- lots of spelling errors. [Hint: Open your function in a word process such a WordPad to see the spelling errors.

fit.model()

- lacks documentation on the input parameters. ALL functions need clear and explicit documentation.

model.comparison.table()

- spelling errors

- add AICc model weight column

neg.log,likelihood()

- spelling errors

print.output()

- see previous emails about needed changes.

unpack.parm()

- see previous emails about needed changes

planning.tool()

- these functions should be in the Functions directory as with all other functions.

- lots of spelling errors

Corrected

- planning.tool.for.optimization()

   - k1, k2 k3 -- meaningless variable names

  - why do you select k1[1] etc. The ORDER of histories should NOT matter.

  - the expected history() function should generate ALL of the histories – see above.

  - I don't understand why you need to generate the log.probability.histories and then find the expected counts when you already a function to generate expected counts (see above).

Corrected the things in this page

**2014-06-05**

Make sure that your code doesn't break when the histories are in a different order or broken into smaller chunks.

corrected

**2014-06-06**

Now that you have the design matrix working for the planning allocation, use it to simplify the problem. For example, rather than using n1, n1.star, n2 with complex constraints, create a design matrix where the underlying parameters are n1.u n1.star, n2 where n1u+n1.star=n1. This way all of n1.u, n1.star, n2 just have a simple constraint that they must be positive. Because you compute n2.star based on the cost, this converts the problem into a simple minimization subject to simple boundary constraints (all of n1.u, n1.star, n2 must be >0) and n2.star is derived based on the linear constraint. Now you can use the standard optimization routines such as nlminb.

Your other code allows for an arbitrary number of categories. Your optimization code allows for only two categories (typically M and F). that is ok, but you should do a test and issue a stop() command if the number of categories is not equal to 2.

**2014-06-07**

Below is some costing information for MilleLacs. Use this to figure out some appropriate costs for the the various phases.

Here the costs are in minutes, but the same idea. Use the observed data from MilleLacs to compute an approximate total cost to use in the example.

The "sexing" is for the second phase -- in the MilleLacs remember that virtually fish were sexed in the first capture occasion.

You should also think about optimal allocation when sexing cost is 0; when the sexing cost is high etc as a discussion part of your paper. In moderate cost cases, my guess is that once you get 400-1000 fish sexed in each phase, you should stop sexing as this will give you an estimate of the observed sex ratio in the catch (confounded with the capture rates) to within about 3 percentage points 19/20 times is likely "good enough". I would start your optimal allocation run with n1.star and n2.star at about 1000 fish and see what your program does.

Remember that you need to give "intuitive explanations" for the optimal allocation that field crews can understand in your paper. You should add your "explanations" to the to-do list for use later when writing your paper.

**2014-06-09**

I had a quick look this morning...you are doing this the hard way.

For example, you have

    temp1 = match(substr(Data$History,1,1),c("0") ) # histories with 0 in the first digit

    u\_2 = sum(Data$counts[which(temp1==1)]) # number of unmarked individuals caught in sample 2

    n\_1 = n - u\_2

Why not simply

  n1 <- sum(Data$counts[ ! substr(Data$History,1,1) == "0')]

Similarly for n2 and m2 etc.

Only use the match function if you want to do a table lookup, i.e. several values to be tested for etc.

The which function is not necessary either if your selection vector is already of type logical.

You have several code fragments of the form

    [which(temp3==1)]

Again this is not needed. temp3 is a logical vector, so you just have to say

    [temp3]

to select the elements based on the logical vector temp3.

corrected

**2014-06-11**

Some more changes

print.model()

- why don't the estimates of the pop sizes and the se align for all of the categories? There appears to an extra blank in the F category estimates.

- add commas (,) in the pop size estimates because they are larger than 10,000.

- you should put the offset to the right of the design matrices rather than underneath.

- add row labels to the design matrices

Residual plot

- X axis label should read Expected counts

- residual table should include the residual

residual.plot() function

- you need to remove the comments about the for loop

- the ggplot code needs to be "prettied" up by align the code fragments etc. It is too messy to read.

- this call should go in the fit.model() routine and add the residual plots and the residual table to the model object rather than having the pair of code

     fit.model()....

     residual.plot()...

- add the results of the residual.plot directly to the model object so it "stays" with the model information. This also makes the print.model() function cleaner as you can't pass the wrong residual plot with the model plot then.

Multiple residual plots

grid.arrange() uses the "Pen-on-paper-paper' paradigm of plotting like Base R graphics. You should avoid using any methods that use this obsolete paradigm.

model.comparison.r

- you have comments that refer to QAICc. You are not using QAICc. DO you understand the difference between AICc and QAICc?

Model ids

Sometimes you say lambda(c), sometimes you say lambda\_(MLE) (i.e. with an underscore). Be consistent everywhere.

Corrected

**2014-06-18**

Looks pretty good. A few suggestions

(a) You will want to do this AFTER you have already fit a model. So just pass the model results from a previous fit to this function. This contains the data and design matrices so you don't have to pass these as well. The model results also contains the MLE etc so you don't need to refit the data.

So your function will start

     function( [model.info](http://model.info/), nboots=1000)

Also set a default value for the number of bootstrap samples (say 1000).

(b) Good use of the plyr() library with no for loops. You are now getting the hand of this

(c) You also need to compute the gof p-value, i.e what proportion of the bootstrap deviances exceed the observed deviance.

(d) Comparing the saturated model and the fitted model is a bit tricky.

As you see in your code, the saturated model is conditional on "n" the total number of animals seen because you would need an estimate of N which isn't obtainable in the saturated model. But this isn't a problem... think of the observed vs expected histories - you can't use the 00 history in the computations because the count of the 00 history is not observable.

So .. what you need to do is find the saturated model ONLY using the observed histories (as you did). This is conditioning on "n" - the total of the observed counts. Now you need to adjust the likelihood of the fitted model to the bootstrap samples (and the original fit) to condition on the observed counts (n).

This isn't too bad because the likelihood can be partitioned into the part involving N, and n\_00 and a conditional part involving n. You will need to subtract from to the computed log-likelihood the gamma terms involving N  and deal with the division by p\_00.

Corrected

**2014-06-18**

The saturated model is conditional on "n" (since you compute the probability of each history as n\_history / n. The factorial terms should only include n and the n\_history -- no N (as it cannot be computed in the saturated model).

The fitted model likelihood needs to subtract the (N-n)log(p\_00) and the N! and (N-n)! terms, but you need to condition on the observed captured histories to translate it to a conditional (on the observed histories) multinomial - refer back to the Sananthanan papers that your read earlier.

The deviance should be close to the sum of the (obs-exp)\*\*2/exp from the residual plots as both are asymptotically equivalent. You don't have to compute p\_00 as you can get this from the expected counts that appear in the residual plot?

Corrected

For computing the p-value, you have

   p.value = length(Deviance.boots[Deviance.boots>Deviance.obs])/n.boots

Just use

   p.value = mean( Deviance.boots > Deviance.obs)

Remember, that True/False are treated as 1/0 in arithmetic computations

I see the p\_00 is easy to compute so not necessary to use the expected counts method.

IN the  fitted.model.log.likelihood() function, don't forget to add the n! term when you find the conditional log-likelihood. Don't also, you may want to rename the function as it will eventual return the conditional log-likelihood, rather than the log-likelihood.

You should also restructuring your code to put everything together for ONE replicate of a bootstrap, i.e.

- generate the bootstrap sample

- compute the saturated log-likelihood\

- fit the model

- find the conditional log-likelihood

- return the deviance.

Corrected

rather than generating a whole bunch of bootstrap sample, then processing each sample in turn etc. Your method can also be parallelized, but it is much more difficult to do and understand.

Then you can use rdply() to replicate the bootstrapping multiple time. The way your current code is structured, you can't "parallelize" things easily. By re-structuring the code into one routine that does everything for a single replicate of a bootstrap sample, it is easily parallelizable.

Corrected

**2014-06-21**

Bootstrap comments

Looks pretty good. Some comments

(a) Split the title line in the bootstrap plot using \n to start a new line with the model id on the second line

(b) Add the gof p-value to the plot.

(c) Rather than using frequency on the Yaxis, use a "probability" value. ggplot can produce these automatically.

(d) One of the options in ggplot for histograms is for the number of bar. You would not have to set the width to get the right number of bars.

(e) The deviances seems awfully large. The deviance value should be fairly close to a chi-square 1 distribution in the case of a unrestricted model and should be close to the sum of the squared residuals (the pearson chi-square test based on the standardized residuals) is asymptotically equivalent.

I think you are missing one of the terms in the correction for conditioning on "n". Don't forget that when you condition on "n", you need to divide all of the probabilities by 1-p00 to account for the conditioning on "n". These get multiplied by the cell numbers etc.... Once you take logs this may "fix" the large deviation values.

(f) Also give the deviance plot for some of the other models, esp the models that clearly don't fit.

Corrected

(g) Getting the residual plots for the top 4 models is done the hard way. Here is what you have to do... You have a list of all the models passed to your model comparison (n.models) and the sorted list of models by AIC of which you want k=min(4, n.models)

- use ldply to extract the model names from the list of all of the models.Use an implicit function rather than an explicit function, i.e.

  model.names <- ldply(model.list, function(x){return(x$[model.id](http://model.id/))})

- use the outer() function to compare the list of top 4 (or less) in the AIC table to all the model ids. This gives you a k x n.models matrix-

- multiple this by 1:n.model to give the index list to the model list for the top 4

- use llply to extract the residual plot from this index list to give the list of residual plots (in order). Again, use an implicit function call rather than an explicit function.

- because this list can have 4 or 3 or 2 or 1 plots, you will need to use the do.call() function to pass the list of plots to the grobArrange function() function, Don't forget to add an element to the of residual plots for the number of rows for grobArrange.

Again, don't use grid.arrange as this does the plot without creating an object using the pen-on-paper paradigm.

Corrected

something wrong with the deviance gof plots.

Bootstrapping of histories will not provide any additional information from the usual analyses and it generates estimates similar to obseved data

whatever the model we consider, the deviance using the bootstrap data is always around the deviance for the observed data.  Therefor observed deviance is reasonably likely and that's why the p-value is large.

Only thing is the wrong models have a very large deviance.

MARK uses parametric bootstrap.

**2014-06-23**

Do a parametric bootstrap rather than a non-parametric bootstrap.

Completed

**2014-06-23**

parametric bootstrap

I had a quick look - looks pretty good. A few comments

(a) Always draw the vertical line for the observed deviance but it is not necessary to put into a legend.

Did you compare the computed deviance to the sum(standardized residuals\*\*2). they should be similar (I think they are)

Can you put the p-value and observed deviance inside the graph (rather than in the title).

See <http://stackoverflow.com/questions/11019692/place-annotation-at-the-top-of-a-series-of-histograms-in-ggplot2-using-a-for-loo>

for an example.

(b) I notice you used the labs() function in ggplot. You will likely find it easier to use the xlab(), ylab() and ggtitle() separately. I.e. rather than

  labs(x = "Deviance",y = "Density",title = paste("Parametric Bootstrap Sampling Distribution for Deviance \n model: ", [model.id](http://model.id/), "\n \n p-value = ", p.value,sep=" " ))+

use

    xlab("Deviace")+ylab("Density)+

    ggtitle(paste(....))

It just makes the code code cleaner and gets rid of the long lines.

(c) You still need to fix the title for the residual plot (put the model on the second line like you did for the deviance plot)

(d) Add the model p(c+t) theta(t) lambda(c). See the MARK manual for the definition of the g+t models in capture-recapture.

(e) The power study should allow comparisons between any set of nested models, not just differences in the capture-probabilities. You will need to modify the function to accept a complete specification for both model, i.e. both model need the complete set of design matrices and offsets.

You should NOT hard code the two model in this power function, i.e. you have hard coded the df = 3 etc. This should be computed based on the models.

Corrected

Again, put the lower inside the plot window (see (a) above).

Chose two alternative models where the models are closer to verity that the power/simulation give approximately the same value.

You should also test your function with the same model given in both the restricted and unrestricted model to see what happens. What you do you expect to see and do you see this?

(f) Lot of spelling errors again in the functions. You should fix these as you go along. Open the final R files in a text editor that has a spelling checker.

Corrected