marked Package Vignette

Jeff L. Laake, Devin S. Johnson, and Paul B. Conn

Jan 16, 2013

Summary

We describe an R package called `marked` for analysis of mark-recapture data. Currently, the package is capable of fitting Cormack-Jolly-Seber (CJS) models with maximum likelihood estimation (MLE) and Bayesian Markov Chain Monte Carlo (MCMC) methods. In addition, Jolly-Seber (JS) models can be fitted with MLE. Additional models will be added as the package is updated. We provide some example analyses with the well-known dipper data and compare run timing and results with MARK. For analysis with several thousand or more capture histories and several time-varying covariates, the run times with `marked` are substantially less. By providing this open source software, we hope to expand the analyst's toolbox and enble the knowledgeable user to understand fully the models and software.

Introduction

Currently the most comprehensive software for analysis of capture-recapture data is MARK (White and Burnham 1999). MARK is a FORTRAN program for fitting capture-recapture models that are manually constructed with a graphical user interface. RMark (Laake and Rexstad 2008) is an R package that constructs models for MARK with user-specified formulas to replace the manual model creation. With RMark and MARK most currently available capture-recapture models can be fitted and manipulated in R. Additional R packages for analysis of capture-recapture data have been made available including Rcapture (Baillargeon and Rivest 2007), mra (McDonald et al. 2005), secr (Borchers and Efford 2008), BTSPAS (Schwarz et al. 2009), SPACECAP (Royle et al. 2009), and BaSTA (Colchero et al. 2012). Rcapture fits closed and open models in a log-linear framework. The mra package fits Cormack-Jolly-Seber (CJS) and the Huggins closed model with a "regression approach" to model specification. The secr and SPACECAP packages provide spatially explicit modeling of closed capture-recapture data and BTSPAS fits time-stratified Petersen models in a Bayesian framework. BaSTA estimates survival with covariates from capture-recapture/recovery data in a Bayesian framework when many individuals are of unknown age. Each package is designed for a unique niche or structure. We believe these alternative packages in R are useful because they expand the analyst's toolbox and the code is open source which enables the knowledgeable user to understand fully what the software is doing. Also, this independent innovation provides testing for existing software and potential gains in developer knowledge to improve existing software.

We developed this R package we named `marked` for analysis with marked animals in contrast to the R package unmarked (Fiske and Chandler 2011) that focuses on analysis with unmarked animals. The original impetus for the package was to implement the CJS model using the hierarchical likelihood construction described by Pledger et al. (2003) and to improve on execution times with RMark/MARK (White and Burnham 1999;Laake and Rexstad 2008) for analysis of our own large data sets with many time-varying individual (animal-specific) covariates. Subsequently, we implemented the Jolly-Seber model with the Schwarz and Arnason (1996) POPAN structure where the hierarchical likelihood construction idea extended to the entry of animals into the population. We also added a Bayesian Markov Chain Monte Carlo (MCMC) implementation of the CJS model based on the approach used by Albert and Chib (1993) for analyzing binary data with a probit regression model.

Background

We assume you know and understand mark-recapture terminology and models. For background material on capture-recapture and MARK refer to `http://www.phidot.org/software/mark/docs/book/`. In the appendix, we provide a comparison of the MARK and `marked` data and model structures and details on likelihood construction for the CJS and JS models. Our focus in the vignette will be to provide examples on the use of `marked` and simulation with timing differences.

Dipper data example

Anyone who has used RMark will find it easy to use `marked` because it has nearly identical syntax and structure with a few minor differences. The structure of the input dataframe for `marked` is also identical to RMark. Any dataframe with a character field named *ch* containing the capture history will work with `marked`. If the capture history represents more than one animal then the dataframe should contain the additional numeric field named *freq*, which is the number of animals represented by that capture history. However, it is not necessary to accumulate capture histories in the input data because the *process.data* step will accumulate duplicated records by default. There can be any number of additional fields, some of which can be used as covariates in the analysis. Some of the functions in `marked` and RMark have the same name (e.g., *process.data*, *make.design.data*), so only one of the packages should be loaded in R to avoid aliasing and resulting errors.

We will start by fitting the simplest CJS model with constant ϕ and p. The dipper data contain 294 records with the capture history (*ch*) and a factor variable *sex* with values Female or Male. Models are fitted with the function *crm* (*c*apture-*r*ecapture *m*odel). After a call to library to attach the package, and data to retrieve the dipper data from the package, the model is fitted with *crm* and assigned to the object named *model*:

```
> library(marked)
> data(dipper)
> model=crm(dipper)

Model:  CJS
Processing data
294 capture histories collapsed into  55
Creating design data. This can take awhile.
Computing initial parameter estimates
Accumulating capture histories based on design. This can take awhile.
55  capture histories collapsed into  32
Starting optimization for  2  parameters

Elapsed time in minutes:  0.0147
```

For this example, we are using the default CJS model and a default formula of ~1 (constant) for each parameter. The function *crm* calls three functions in turn: 1) *process.data* to process the data, 2) *make.design.data* to create the list of parameter-specific dataframes, and 3) *cjs*, to fit the CJS model with the defined formulas. The code reports progress and some results as it executes. We'll suppress these messages in later examples but show them here to explain some of the messages. When it processes the data, it collapses the 294 rows into 55 which are the unique number of rows in the data including *ch* and *sex*. After processing, it creates the design data and the design matrices for each parameter. Prior to the optimization, it also collapses the histories further from 55 to 32 which it can do here because *sex* is not used in either formula. The final steps are to compute initial values for the parameters and find the MLEs with the selected optimization method(s). As it progresses, the value of -2log-likelihood is reported every 100 evaluations of the objective function. It is not shown above because there were fewer than 100 function evaluations for this example. A brief listing of the model results is obtained by typing model which invokes the function *print.crm* because *class(model)=”crm”*.

```
> model

crm Model Summary

Npar :  2
-2lnL:  666.8377
AIC  :  670.8377

Beta
                 Estimate
Phi.(Intercept) 0.2420302
p.(Intercept)   2.2270627
```

The output includes the number of parameters (*npar*), -2log-likelihood, Akaike's Information Criterion (*AIC*), and the estimates for ϕ and p.

Estimates of precision are not shown because the default is *hessian*=FALSE and it must be set to TRUE to get estimates of precision. This default was chosen because the `marked` package does not count parameters from the hessian, so there is no need to compute it for each model as with MARK. The hessian may never be needed if the model is clearly inferior. Also, this allows the model to be fitted again from the final or different estimates to check for convergence without the penalty of computing the hessian at the final values each time. Separate functions (*cjs.hessian* and *js.hessian*) are provided to compute and store in the model the variance-covariance matrix from the hessian at the final estimates as shown below:

```
> model=cjs.hessian(model)
> model

crm Model Summary
```

```
Npar :  2
-2lnL:  666.8377
AIC  :  670.8377

Beta
                   Estimate        se         lcl       ucl
Phi.(Intercept) 0.2420302 0.1020079 0.04209475 0.4419656
p.(Intercept)   2.2270627 0.3252176 1.58963625 2.8644892
```

Once the hessian has been computed, printing the model will display the standard errors and 95% normal confidence intervals for the parameter estimates on the link scale (e.g., logit for ϕ and p). You can set *hessian*=TRUE in the call to *crm* if you want to compute it when the model is fitted.

You'll never fit only one model to data, so the most efficient approach is to call *process.data* and *make.design.data* separately and pass the results to *crm* so they can be used for each fitted model as shown below:

```
> dipper.proc=process.data(dipper)
> dipper.ddl=make.design.data(dipper.proc)
> Phi.sex=list(formula=~sex)
> model=crm(dipper.proc,dipper.ddl,model.parameters=list(Phi=Phi.sex),
+           accumulate=FALSE)
```

Collapsing capture history records is controlled by the *accumulate* arguments in *process.data* and *crm*. In the above example, *accumulate* was TRUE for the *process.data* step but it was turned off for the model fit because it would have not resulted in any accumulation with the sex term in the model. Typically the default values are optimal but if you are fitting many models and most are complex models, then it may save time to accumulate in the *process.data* step but not for the model fitting step.

If you fit more than a few models, use *crm.wrapper* rather than *crm*. It fits a set of models and returns a list with a model selection table that summarizes the fit of all the models. By default, *crm.wrapper* stores the model results externally and in the list it only stores the names of the files containing the models. If you set *external*=FALSE, then it will store the model results in the list as shown in the example below.

```
> dipper.proc=process.data(dipper)
> dipper.ddl=make.design.data(dipper.proc)
> fit.models=function()
+ {
+   Phi.sex=list(formula=~sex)
+   Phi.time=list(formula=~time)
+   p.sex=list(formula=~sex)
+   p.dot=list(formula=~1)
+   cml=create.model.list(c("Phi","p"))
+   results=crm.wrapper(cml,data=dipper.proc, ddl=dipper.ddl,
+                       external=FALSE,accumulate=FALSE)
+   return(results)
+ }
> dipper.models=fit.models()
```

The model selection table is displayed with:

```
> dipper.models

              model npar      AIC DeltaAIC    weight  neg2lnl
1    Phi(~sex)p(~1)    3 672.6762 0.000000 0.4241510 666.6762
3   Phi(~time)p(~1)    7 673.7301 1.053881 0.2504224 659.7301
2  Phi(~sex)p(~sex)    4 674.1518 1.475609 0.2028131 666.1518
4 Phi(~time)p(~sex)    8 675.1583 2.482104 0.1226135 659.1583
  convergence
1           0
3           0
2           0
4           0
```

A non-zero value for convergence means the model did not converge. If the models are not stored externally, an individual model can be extracted from the list with either the model number which is listed in the model table or with the model name which is the model formula specifications pasted together as shown below:

```
> dipper.models[[1]]

crm Model Summary

Npar :  3
-2lnL:  666.6762
```

```
AIC  :  672.6762

Beta
                  Estimate
Phi.(Intercept) 0.20364163
Phi.sexMale     0.07928539
p.(Intercept)   2.22748575

> dipper.models[["Phi.sex.p.dot"]]

crm Model Summary

Npar :  3
-2lnL:  666.6762
AIC  :  672.6762

Beta
                  Estimate
Phi.(Intercept) 0.20364163
Phi.sexMale     0.07928539
p.(Intercept)   2.22748575
```

If the models are stored externally, they can be retrieved with the function *load.model* as shown below:

```
> model=load.model(dipper.models[[1]])
> model

crm Model Summary

Npar :  3
-2lnL:  666.6762
AIC  :  672.6762

Beta
                  Estimate
Phi.(Intercept) 0.20364163
Phi.sexMale     0.07928539
p.(Intercept)   2.22748575
```

To make the analysis more interesting, we will add some covariates for ϕ and p. For ϕ, we will add a static covariate *weight* which is a random value between 1 and 10. For ϕ, we also add a time-varying covariate *Flood* which is the same for all dippers but varies by time with a 0 value for times 1,4,5,6 and a value of 1 for times 2 and 3. For p, we will add a time-varying individual covariate *td* (trap dependence) which is the 0/1 value of the capture from the previous occasion. Static covariates are entered in the dataframe in a single column and time-varying covariates have a column and name for each occasion with the appropriate time appended at the end of each name. In this case, *Flood* will be *Flood1,...,Flood6* and for *td* it will be *td2,...,td7* because time for ϕ is based on the time at the beginning of the interval and for p it is the time for the capture occasion. Below the names of the covariates in the dataframe are shown after they are created:

```
> data(dipper)
> # Add a dummy weight field which are random values from 1 to 10
> set.seed(123)
> dipper$weight=round(runif(nrow(dipper),0,9),0)+1
> # Add Flood covariate
> Flood=matrix(rep(c(0,1,1,0,0,0),each=nrow(dipper)),ncol=6)
> colnames(Flood)=paste("Flood",1:6,sep="")
> dipper=cbind(dipper,Flood)
> # Add td covariate, but exclude first release as a capture
> # splitCH and process.ch are functions in the marked package
> td=splitCH(dipper$ch)
> td=td[,1:6]
> releaseocc=process.ch(dipper$ch)$first
> releaseocc=cbind(1:length(releaseocc),releaseocc)
> releaseocc=releaseocc[releaseocc[,2]<nchar(dipper$ch[1]),]
> td[releaseocc]=0
> colnames(td)=paste("td",2:7,sep="")
> dipper=cbind(dipper,td)
> # show names
> names(dipper)
```

```
 [1] "ch"     "sex"    "weight" "Flood1" "Flood2" "Flood3" "Flood4"
 [8] "Flood5" "Flood6" "td2"    "td3"    "td4"    "td5"    "td6"
[15] "td7"
```

Next we process the data with the default CJS model and make the design data with parameters that identify which covariates to use for each parameter. By default, each covariate in the dataframe is added to the design data of each parameter but the argument static can be used to limit the variables appended and the time-varying argument is needed to identify which covariates vary by time and should be collapsed into a single column. If we did not include the static argument for ϕ, then *weight* and each of the *td* columns would also be included and for p, weight and each of the *Flood* columns would be included. If *td* was added to the time-varying argument for ϕ and *Flood* was added for p, then we would have also needed to add *td1* and *Flood7* due to the different time labels for ϕ and p. We have no intention to use those variables so it is best to specify what is needed. After specifying the design lists for ϕ and p, they are used in the call to *make.design.data* and the resulting dataframe names are shown below:

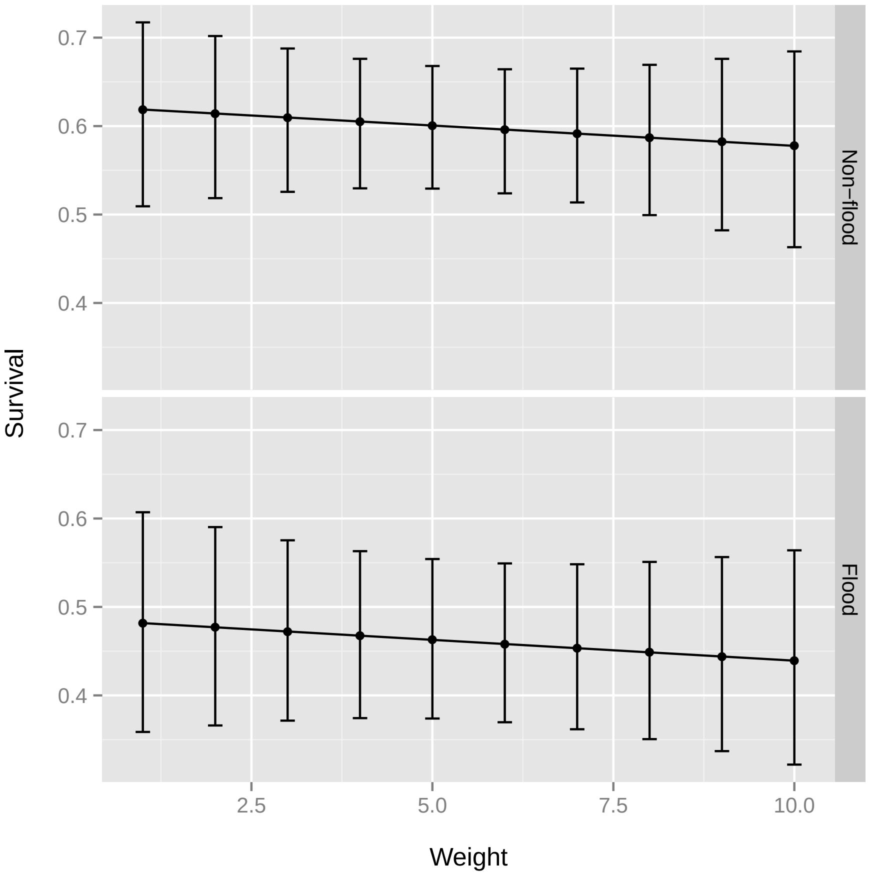
```
> # Process data
> dipper.proc=process.data(dipper)
> # Create design data with static and time varying covariates
> design.Phi=list(static=c("weight"),time.varying=c("Flood"))
> design.p=list(static=c("sex"),time.varying=c("td"),
+                         age.bins=c(0,1,20))
> design.parameters=list(Phi=design.Phi,p=design.p)
> ddl=make.design.data(dipper.proc,parameters=design.parameters)
> names(ddl$Phi)
> names(ddl$p)
```

Next we define the models for ϕ and p that we want to fit and call crm.

```
> Phi.sfw=list(formula=~Flood+weight)
> p.ast=list(formula=~age+sex+td)
> model=crm(dipper.proc,ddl,hessian=TRUE,
+             model.parameters=list(Phi=Phi.sfw,p=p.ast))
```

Below we create a range of data values to compute predicted ϕ values and then plot the results for Flood and non-Flood years for a range of weights. Not surprising that the slope for weight is nearly 0 because the weight values were generated randomly.

```
> newdipper=expand.grid(sex=c("Male","Female"),weight=1:10,Flood1=0,
+       Flood2=1,Flood3=1,Flood4=0,Flood5=0,Flood6=0,td2=0,td3=c(0,1),
+       td4=c(0,1),td5=c(0,1),td6=c(0,1),td7=c(0,1))
> reals=predict(model,newdata=newdipper,se=TRUE)
> library(ggplot2)
> reals$Phi$Flood=factor(reals$Phi$Flood,labels=c("Non-flood","Flood"))
> ggplot(reals$Phi,aes(weight,estimate,ymin=lcl,ymax=ucl))+
+        geom_errorbar(width=0.2)+geom_point()+geom_line()+
+        xlab("\nWeight")+ylab("Survival\n")+facet_grid(Flood~.)
```



Fitting an MCMC CJS model (*probitCJS*) to the dipper data is quite similar to the MLE models with the added arguments *burnin* and *iter* to control the number of *burnin* iterations and the number of iterations to perform after the *burnin* process. The following fits Phi(˜Flood)p(˜sex+time) with low values set for *burnin* and *iter* to reduce the vignette build time.

```
> data(dipper)
> # Add Flood covariate
> Flood=matrix(rep(c(0,1,1,0,0,0),each=nrow(dipper)),ncol=6)
> colnames(Flood)=paste("Flood",1:6,sep="")
> dipper=cbind(dipper,Flood)
> design.parameters=list(Phi=list(time.varying="Flood"))
> model.parameters=list(Phi=list(formula=~Flood),
+                        p=list(formula=~time+sex))
> MCMCfit=crm(dipper,model="probitCJS",
+             model.parameters=model.parameters,
+             design.parameters=design.parameters,
+             burnin=1000,iter=5000)

Model:  probitCJS
Processing data
Creating design data. This can take awhile.
Computing initial parameter estimates
probitCJS MCMC beginning...
p model =  ~ time + sex
phi model =  ~ Flood
Approximate time till completion:  0.6  minutes
10 % completed
20 % completed
30 % completed
40 % completed
50 % completed
60 % completed
70 % completed
80 % completed
90 % completed
100 % completed

Elapsed time in minutes:  0.8598
```

The results shown are the mode,mean,standard deviation, and 95% highest posterior density interval for the beta parameters and unique ϕ and p real values.

```
> # beta estimates
> MCMCfit

crm Model Summary
```

```
Npar :  9

Beta
                      mode       mean         sd   CI.lower
Phi.(Intercept)  0.2749370  0.2912696 0.08973318  0.1106013
Phi.Flood       -0.3884804 -0.3753850 0.14141597 -0.6533261
p.(Intercept)    0.4140836  0.5698078 0.44489929 -0.2487042
p.time3          0.5139672  0.6586387 0.59991937 -0.4850090
p.time4          0.6657875  0.6543644 0.55642292 -0.4855225
p.time5          0.6820132  0.6657598 0.51173699 -0.3420197
p.time6          0.8461588  0.7910958 0.54636214 -0.3259645
p.time7          0.4996805  0.5179868 0.64625588 -0.7060412
p.sexMale        0.2548546  0.2671861 0.29836671 -0.2965319
                  CI.upper
Phi.(Intercept)  0.4628545
Phi.Flood       -0.1025575
p.(Intercept)    1.4814044
p.time3          1.8677067
p.time4          1.7002608
p.time5          1.6692771
p.time6          1.8130904
p.time7          1.8843851
p.sexMale        0.8708236

> # real estimates
> MCMCfit$results$reals

$Phi
  Flood      mode      mean         sd  CI.lower  CI.upper
1     0 0.6088605 0.6141262 0.03412873 0.5440337 0.6782657
2     1 0.4674608 0.4666748 0.04259277 0.3869100 0.5513335

$p
   time    sex      mode      mean         sd  CI.lower  CI.upper
1     2 Female 0.7596788 0.6979662 0.14002079 0.4240942 0.9427035
2     2   Male 0.8507216 0.7765668 0.12293162 0.5375297 0.9796021
3     3 Female 0.9379705 0.8684767 0.08557616 0.7050317 0.9978447
4     3   Male 0.9593349 0.9113565 0.06872156 0.7728171 0.9993625
5     4 Female 0.9122068 0.8747917 0.07150282 0.7386733 0.9874568
```

```
6      4   Male 0.9522966 0.9177248 0.05605280 0.8091453 0.9970443
7      5 Female 0.9078881 0.8806079 0.06007884 0.7688724 0.9853112
8      5   Male 0.9539749 0.9223662 0.04762355 0.8303261 0.9942904
9      6 Female 0.9305683 0.8999823 0.05843383 0.7853789 0.9919248
10     6   Male 0.9636290 0.9360712 0.04520629 0.8431578 0.9988252
11     7 Female 0.8342433 0.8330626 0.09818177 0.6618383 0.9998414
12     7   Male 0.9369966 0.8887917 0.07566673 0.7475864 0.9996799
```

Validation and timing

For a comparison of execution times and to validate the results against RMark/MARK, we replicated the dipper data twenty-fold so there were roughly 6,000 records, added a random weight field rounded to values 1 to 10, added individual time-varying covariates *td* and *age* as shown in the following code:

```
> data(dipper)
> dipper=dipper[rep(1:nrow(dipper),each=20),]
> dipper$weight=round(runif(nrow(dipper),0,10),0)
> dipper$region=factor(floor(runif(nrow(dipper),1,10.99)))
> td=splitCH(dipper$ch)
> td=td[,1:6]
> colnames(td)=paste("td",2:7,sep="")
> dipper=cbind(dipper,td)
> age=t(apply(process.ch(dipper$ch,all=TRUE)$Fplus,1,cumsum))
> colnames(age)=paste("age",1:7,sep="")
> dipper=cbind(dipper,age)
```

We defined the following set of 8 models for ϕ and 7 models for p and fitted all 56 models with the `marked` and RMark packages.

```
> Phi.1=list(formula=~1)
> Phi.2=list(formula=~weight+age)
> Phi.3=list(formula=~time+weight)
> Phi.4=list(formula=~sex+weight)
> Phi.5=list(formula=~sex+weight+age)
> Phi.6=list(formula=~region+weight+age)
> Phi.7=list(formula=~region+time+weight)
> Phi.8=list(formula=~region+sex+weight)
> p.1=list(formula=~1)
> p.2=list(formula=~time)
> p.3=list(formula=~sex)
> p.4=list(formula=~td)
> p.5=list(formula=~td+sex)
> p.6=list(formula=~td+sex+age)
> p.7=list(formula=~region+sex+age)
>
>
```

The models were fitted with `marked` in 18.09 minutes with the R MLE implementation, 18.03 minutes with the ADMB implementation (*use.admb*=TRUE) and 41.38 minutes with RMark/MARK. MARK was using 7 CPUs for those timings and `marked` was using 1 CPU. The time difference was larger prior to the recent implementation of parallel processing in MARK. As more group structure and time-varying individual covariates are added the time difference increases substantially. As a comparison of convergence performance, we subtracted the -log-likelihood value of the RMark model fit from the equivalent value from `marked`. A negative value implies a better fit (convergence) for the `marked` package and vice-versa for a positive value. With the initial runs of each package using their default starting values there were 9, 11, and 4 models with a difference exceeding 0.01 for R (*use.admb*=FALSE) versus ADMB (*use.admb*=TRUE), R versus MARK and ADMB versus MARK. The range of -loglikelihood differences was from -9.2973E-03 to 1.3911E-01, -1.3142E+01 to 1.3611E-01, and -1.3142E+01 to 9.0000E-03 for the three comparisons respectively. ADMB had the lowest negative log-likelihood for all of the models. The models that differed had little model weight and either had confounded parameters (e.g., Phi(time)p(time)) or had large negative or positive parameters as the real parameters approached boundaries of 0 or 1. With the exception of the 1 model in MARK most converged to similar values in the initial run. We re-ran the set of MARK models using starting values from the models fit with `marked`, and then ran the `marked` models with starting values from the models re-fitted with MARK. After re-running models the differences in -log-likelihood values between ADMB and MARK ranged from -1.0000E-02 to 9.0000E-03 showing that the likelihood for both packages are the same and the initial differences resulted from differences in starting values and tolerance criterion.

References

Albert, J. and Chib, S. (1993). Bayesian-analysis of binary and polychotomous reponse data. *Journal of the American Statistical Association*, 88(422):669–679.

Baillargeon, S. and Rivest, L. P. (2007). Rcapture: Loglinear models for capture-recapture in R. *Journal of Statistical Software*, 19(5):1–31.

Borchers, D. L. and Efford, M. G. (2008). Spatially Explicit Maximum Likelihood Methods for Capture-Recapture Studies. *Biometrics*, 64(2):377–385.

Colchero, F., Jones, O. R., and Rebke, M. (2012). BaSTA: an R package for Bayesian estimation of age-specific survival from incomplete mark-recapture/recovery data with covariates. *Methods in Ecology and Evolution*, 3:466–470.

Fiske, I. J. and Chandler, R. B. (2011). unmarked : An R Package for fitting hierarchical models of wildlife occurrence and abundance. 43(10):1–23.

Laake, J. and Rexstad, E. (2008). RMark – an alternative approach to building linear models in MARK. In Cooch, E. and White, G. C., editors, *Program MARK: A Gentle Introduction*.

Manly, B. F. J. and Parr, M. J. (1968). A new method of estimating population size, survivorship, and birth rate from recapture data. *Transactions of the society for British Entomology*, 18:81–89.

McDonald, T. L., Amstrup, S. C., Regehr, E. V., and Manly, B. F. J. (2005). Examples. In Amstrup, S. C., McDonald, T. L., and Manly, B. F. J., editors, *Handbook of capture-recapture analysis*, pages 196–265. Princeton University Press, Princeton, New Jersey USA.

Pledger, S., Pollock, K. H., and Norris, J. L. (2003). Open capture-recapture models with heterogeneity: I. Cormack-Jolly-Seber model. *Biometrics*, 59(4):786–794.

Royle, J. A., Karanth, K. U., Gopalaswamy, A. M., and Kumar, N. S. (2009). Bayesian inference in camera trapping studies for a class of spatial capture-recapture models. *Ecology*, 90(11):3233–44.

Schwarz, C. J. and Arnason, A. N. (1996). A general methodology for the analysis of capture-recapture experiments in open populations. *Biometrics*, 52(3):860–873.

Schwarz, C. J., Pickard, D., Marine, K., and Bonner., S. J. (2009). Juvenile salmonid outmigrant monitoring evaluation, Phase II, september 2009. Unpublished Report prepared for the Trinity River Restoration Program, Weaverville, CA.

White, G. C. and Burnham, K. P. (1999). Program MARK: survival estimation from populations of marked animals. *Bird Study*, 46:120–139.

Appendix

Model construction

The manner in which models are constructed in the `marked` package is different than in MARK. We start by describing how MARK specifies and constructs models and how the `marked` package differs. The likelihood for CJS/JS is a product multinomial and the multinomial cell probabilities are functions of the parameters. We will focus on CJS which has two types of parameters: p (capture/recapture probability) and ϕ (apparent survival). Each cell is associated with a release cohort and recapture time (occasion). The cohorts can be further split into groups based on categorical variables (e.g., sex). In MARK, a parameter index matrix (PIM) is used to specify the parameters. Each cell can have a unique index for each type of parameter type (e.g., p and ϕ). As a very simple example, we will consider a CJS model for a single group with k=4 occasions. For this structure the PIMS are triangular with a row for each cohort released at each occasion (time) and the column representing the occasions (times) following the release. Using an unique index for each potential parameter, the PIMS would be:

ϕ				p			
	Time				Time		
Cohort	2	3	4	Cohort	2	3	4
1	1	2	3	1	7	8	9
2		4	5	2		10	11
3			6	3			12

The index 5 represents survival of the second release cohort between occasions 3 and 4 and index 8 represents recapture probability of the first release cohort on the third occasion. A separate set of PIMS is created for each group defined in the data. For this example, if there were g groups there would be g*12 possible unique indices.

Some reduced models with constraints on parameters can be constructed by modifying the PIMS. For example, a model with constant survival and capture probability (Phi(.)p(.)) could be specified by setting all of the PIM values to 1 for the ϕ PIM and 2 for the p PIM. Not all reduced models can be specified by modifying the PIMS and more generally reduced models are constructed with a design matrix which is a set of linear constraints. Each parameter index specifies a row in the design matrix which can be used to apply constraints on the parameters and to relate covariates to the parameters. The PIM approach minimizes the work of manually creating a design matrix by reducing the set of potential parameters and thus the number of rows in the design matrix. For our example, there would be 12 rows in the design matrix and each column in the design matrix (X) represents one of the parameters in the vectorβ. If a logit link is used, the real parameters θ (e.g., ϕ and p in CJS) are:

$$\theta = \frac{1}{1 + \exp(-X\beta)} \tag{1}$$

To specify the Phi(.)p(.) model, the design matrix (X) would have 12 rows and 2 columns. The first column in X would have a 1 in the first six rows (indices 1-6) and a 0 in the last six rows (indices 7-12). The second column would have 0 in the first six rows and a 1 in the last 6 rows. The resulting model would have two parameters with the first representing ϕ and the second p.

To automate the creation of design matrices with formula, the RMark package creates “design data” which are data about the parameters like cohort, time, group, and any related variables. However, having a single X becomes problematic with individual covariates (variables that differ for animals in the same group/cohort). In MARK, these individual covariates are entered into the design matrix as a covariate name. When the real parameters are computed, the name of the covariate is replaced with the value for an animal. If there are only a few individual covariates and small number of animals, the time required for replacement and computation is minor, but when the individual covariates differ in time, there is a different covariate name for each time and most rows in the design matrix must be recomputed which results in longer execution times for large models.

Even if computation time is not limiting, we agree with McDonald et al. (2005) that the PIMs used in MARK can be quite confusing to the novice user because they are an additional layer of abstraction from the data. PIMS are useful for manual model creation, but they become an unnecessary nuisance when models and design matrices are created with formula. Explicit PIMS can be avoided by having an underlying real parameter for each animal for each occasion regardless of the release cohort. It can be viewed as a rectangular matrix with a column for each animal and a row for each occasion. That was the solution of McDonald et al. (2005) who in their mra package use a rectangular covariate matrices as predictor for the real parameters as in standard regression.

We use a similar approach that we believe is simpler for the user. For a specified model structure (e.g., CJS), the `marked` package creates from the raw data, a list of dataframes. Each dataframe contains the covariate data for a type of parameter in the model (e.g., ϕ and p). The dataframes contain a record for each animal-occasion and the columns are the covariates. Each record contains the covariate values for that animal-occasion. If the covariate is constant across time then the value is the same value in each record for an animal and the value may be different for each occasion if the covariate is time-varying. Time-varying covariates must be named in the raw data in a certain manner and specified in the processing step. Each column in the dataframe is a vector that is equivalent to one of the covariate matrices in mra; however, rather than specifying the model as a set of covariate matrices, the standard R model formula (e.g., ˜time+sex) can be used with the dataframe for each parameter which is even closer to a typical regression analysis.

A separate dataframe is created for each parameter to allow different data (e.g., time values for ϕ and p) and number of occasions (e.g., ϕ and p in CJS) for each parameter. For ϕ and p in the CJS model, there are $n(k-1)$ rows for n animals and $k-1$ occasions for each parameter but time is labeled by 1 to $k-1$ for ϕ and 2 to k for p. For the JS model, there are $n(k-1)$ records for survival probability and entry probability, but for capture probability there are nk records because the initial capture event is modeled.

The parameter-specific animal-occasion dataframes are automatically created from the user's data which contain a single record per animal containing the capture history and any covariates. Some covariates like cohort, age, and time are generated automatically for each record. Other static and time-varying covariates specified in the data are added. Static variables (e.g., sex) are repeated for each occasion for each animal. Time-varying covariates are specified in the data using a naming convention of "vt" where "v" is the covariate name and "t" is a numeric value for the time (occasion) (e.g., td19 contains the value of covariate td at time 19). The time-varying covariates are collapsed in the animal-occasion dataframes to a single column with the name identified by "v" and each record contains the appropriate value for the occasion. All of this is transparent to the user who only needs to specify a dataframe, the type of model (e.g., CJS), the variables that should be treated as time-varying, and the formula for each parameter.

With the R function model.matrix, the formula is applied to the dataframe to create the design matrix for each parameter (e.g, X_ϕ,X_p). They are each equivalent to a portion of the design matrix in MARK for an animal which are then “stacked on top of one another” to make a matrix for all animals. For maximum likelihood estimation (MLE), equation 1 (or similar inverse link function) is applied and the resulting vector is converted to a matrix with n rows and a column for each required occasion ($k-1$ or k). For Bayesian MCMC inference in `marked`, only$X\beta$ are needed for updating.

Model fitting

Currently there are only three types of models implemented in the `marked` package: 1) CJS, 2) JS, and 3) probitCJS. The first two are based on MLE and the third is an MCMC implementation as described by (Johnson et al in prep). The likelihoods for CJS and JS were developed hierarchically as described by Pledger et al. (2003) for CJS. For simplicity we only consider a single animal to avoid the additional subscript. Let ω be a capture history vector having value ω_j= 1 when the animal was initially captured and released or recaptured at occasion j and ω_j= 0 if the animal was not captured on occasion j, for occasions j=1,...k. The probability of observing a particular capture history $Pr(\omega)$ for CJS can be divided into two pieces ($Pr(\omega) = Pr(\omega_1)Pr(\omega_2)$: 1) ω_1 is the portion of the capture history from the initial release (i.e., first 1) to the last occasion it was sighted (i.e., last 1), and 2) ω_2 is from the last 1 to the last occasion. $Pr(\omega_1)$ is easily computed because the time period when the animal was available for capture is known but $Pr(\omega_2)$ is more difficult to compute because it is unknown whether the animal survived until the last occasion, or if it died, when it died. Typically, $Pr(\omega_2)$ has been computed recursively (Nichols 2005). A hierarchical construction is more direct and understandable. As described in Pledger et al. (2003), we let f be the occasion an animal was released and let d be the occasion after which the animal is no longer available to be recaptured due to death or termination of the study. Then $Pr(\omega|\, f) = \sum_d Pr(\omega|\, f,\, d)Pr(d\,|\, f)$ where the conditional probability $Pr(\omega|\, f,\, d)$ is:

$$Pr(\omega|\, f,\, d) = \prod_{j=f+1}^{d} p_j^{\omega_j}(1-p_j)^{(1-\omega_j)}$$

and the departure probability is

$$Pr(d\,|\, f) = \left(\prod_{j=f}^{d-1} \phi_j\right)(1-\phi_d)$$

Viewed in this way, it is possible to construct the likelihood values for all of the observations with a set of matrices as we describe in the Appendix.

The hierarchical approach is easily extended to JS. Now the capture history has an additional component with $Pr(\omega) = Pr(\omega_0)Pr(\omega_1)Pr(\omega_2)$ where ω_0 is the portion of the capture history vector from the first occasion (j=1) to the occasion the animal was first seen (j=f). The $Pr(\omega_0)$ is similar to $Pr(\omega_2)$ but now it is e, the occasion at which the animal was first available for capture (i.e., entered in the prior interval) that is unknown. The CJS likelihood provides $Pr(\omega|\, f)$ so we only need to compute $Pr(\omega_0) = \sum_{e=0}^{f-1} Pr(\omega_0|\, e)\pi_e$ where π_e is the probability of an animal enters the population in the interval between occasion e and e+1 ($\sum_{e=1}^{k-1} \pi_e = 1 - \pi_0$; pent parameters in MARK and specified as β by Schwarz and Arnason (1996)) and

$$Pr(\omega_0|\, e) = \left(\prod_{j=e+1}^{f-1} (1 - p_j) \right) p_f$$

The JS likelihood also contains a component for animals that entered but were never seen. The details for the JS likelihood construction are provided in the Appendix.

The MLEs are obtained numerically by finding the minimum of the negative log-likelihood using optimization methods provided through the optimx R package (Nash and). The default optimization method is BFGS but you can specify alternate methods and several methods used independently or in sequence as described by Nash and (). Initial values for parameters can either be provided as a constant (e.g., 0) or as a vector from the results of a previously fitted similar model. If initial values are not specified then they are computed using general linear models (GLM) that provide approximations. Using the underlying idea in Manly and Parr (1968) we compute initial estimates for capture-probability p for occasions 2 to $k - 1$ using a binomial GLM with the formula for p which is fitted to a sequence of Bernoulli random variables that are a subset of the capture history values y_{ij} $i = 1, ..., n$ and $j = f_i + 1, ..., l_i - 1$ where f_i and l_i are the first and last occasions the i^{th} animal was seen. A similar but more ad-hoc idea is used for ϕ. We know the animal is alive between f_i and l_i and assume that the animal dies at occasion $l_i + 1 \leq k$. We use a binomial GLM with the formula for ϕ fitted to the y^*_{ij} $i = 1, ..., n$ and $j = f_i + 1, ..., l_i + 1 \leq k$ where y^*_{ij}=1 for $j = f_i + 1, ..., l_i$ and y^*_{ij}=0 for $j = l_i + 1 \leq k$. For ϕ and p, a logit link is used for MLE and a probit link for MCMC. For JS, a log link is used for f^0, the number never captured and the estimate of super-population size is the total number of individual animals plus f^0. Also, for JS a multinomial logit link is used for π_e. The initial value is set to 0 for any β that is not estimated (e.g., p_K in CJS or p_1,π_e, N for JS).

Likelihood Details

Here we provide some details for the likelihoods that are computed in cjs.lnl and cjs.f for the CJS model and in js.lnl for the JS model. Let ϕ,M p be $n \times k$ matrices which are functions of the parameters whereϕ_{ij} is the survival probability for animal i from occasion $j-1$ to j ($\phi_{i1} = 1$), m_{ij} is the probability of dying in the interval j to $j+1$ ($m_{ik} = 1$) and p_{ij} is the capture probability for animal i on occasion j ($p_{i1} = 0$). In addition we define a series of matrices and vectors computed from the data in the function process.ch. Let C be an $n \times k$ matrix of the capture history values and $f_i$and l_i are the first and last occasions the i^{th} animal was seen. Derived from those values are $n \times k$ matrices L, F and F^+ which contain 0 except that $L_{ij} = 1$ $j \geq l_i$,$F_{ij} = 1$ $j \geq f_i$ and $F^+_{ij} = 1$ $j > f_i$. The likelihood calculation has the following steps where the matrix multiplication is element-wise and 1 and represents an $n \times k$ matrix where each element is 1:

1. Construct the $n \times k$ matrix $\phi' = 1 - F^+ + \phi F^+$which is the ϕ matrix modified such that $\phi_{ij} = 1$ for $j \leq f_i$.
2. Construct the $n \times k$ matrix $\phi*$from ϕ' where $\phi_{ij}* = \prod_{i=1}^{j} \phi'_{ij}$.
3. Construct the $n \times k$ matrix $p' = 1 - F^+ + F^+(Cp + (1-C)(1-p))$.
4. Construct the $n \times k$ matrix $p*$from p' where $p_{ij}* = \prod_{i=1}^{j} p'_{ij}$.
5. Compute $n \times 1$ vector of probabilities of observed capture histories $Pr(\omega_i)$ $i = 1, ..., n$ which are the sums of the rows of $LM\phi^* p^*$.
6. Compute the log-likelihood $\sum_{i=1}^{n} \ln(Pr(\omega_i))$

The R code translates from the mathematics quite literally as:

```
> Phiprime=1-Fplus + Phi*Fplus
> Phistar=t(apply(Phiprime,1,cumprod))
> pprime=(1-Fplus)+Fplus*(C*p+(1-C)*(1-p))
> pstar=t(apply(pprime,1,cumprod))
> pomega=rowSums(L*M*Phistar*pstar)
> lnl=sum(log(pomega))
```

While this code is simple it is faster if the apply is replaced with a loop because there are far more rows than occasions or replaced with compiled code which we did with a FORTRAN subroutine (cjs.f called from cjs.lnl).

The Jolly-Seber likelihood can be partitioned into 3 components: 1) CJS likelihood for $Pr(\omega_1)Pr(\omega_2)$ treating the first “1” as a release, 2) a likelihood component for $Pr(\omega_0)$; entry and first observation, 3) a component for those that entered before each occasion but were never seen. We define an additional $n \times k$ matrix π which are the entry probabilities into the population (specified as β by Schwarz and Arnason (1996)) with the obvious constraint that $\sum_{j=0}^{k-1} \pi_{ij} = 1$ and N_g $g = 1, ..., G$ is the abundance of animals in each of the defined G groups (e.g., male/female) that were in the population at some time (super-population size). $N_g = n_g + f_g^0$ where n_g is the number observed in the group and f_g^0 are the estimated number of animals in the group that were never seen.

The same hierarchical approach used for CJS can be used for the second component. Construct the capture history probability for a given entry time and then sum over all possible entry times. . The second component is constructed with the following steps:

1. Construct the $n \times k$ matrix $E = (1 - p)\phi(1 - F) + F$,
2. Construct the $n \times k$ matrix $E*$where $E_{ij}* = \prod_{l=j}^{k} E_{il}$,
3. Compute $n \times 1$ vector of probabilities $Pr(\omega')$ which are the sums of the rows of $E * (1 - F^+)\pi$ multiplied by the vector p_{if_i} $i = 1, ..., n$,
4. Compute the log-likelihood $\sum_{i=1}^{n} \ln(Pr(\omega_i'))$.

For the third likelihood component for missed animals, we constructed $G \times k$ dummy capture histories of all 0’s except for a “1” at the occasion the animals entered. From the CJS portion of the code, we obtained p_{gj}^0 the probability that an animal released in group g on occasion j would never be captured. The final log-likelihood component is:

$$\sum_{g=1}^{G} f_g^0 \ln \left[\sum_{j=1}^{k} \pi_{g,j-1}(1 - p_{gj})p_{gj}^0 \right] + \ln(N_g!) - \ln(f_g^0!)$$

The JS log-likelihood is the total of the 3 components plus $-\sum_{g=1}^{G}\sum_{j=1}^{k} \ln(n_{gj}!)$ which do not depend on the parameters but is added after optimization in function js, to be consistent with the output from MARK.

With the structure we have used, the design matrices can become quite large and available memory could become limiting. For the design matrices, we use sparse matrices with the R package Matrix (citation). In addition, for MLE analysis, we use the following to reduce the required memory:

1. We reduce the data to $n^* \leq n$ individuals by aggregating records with identical data and using the frequencies $(f_1, ..., f_{n^*}; n = \sum_{i=1}^{n^*} f_i)$ in the likelihood.

2. We construct the design matrix X incrementally with a user-specified size of data chunk that is processed at one time.

3. We retain only the rows of X which are unique for the design and including any fixed parameters which can be animal-specific.

For Bayesian MCMC inference, we cannot aggregate records but we reduce the required memory by:

1. Eliminating the unused animal-occasion data for occasions prior to and including the release occasion for the animal, and

2. Storing only the unique values of the real parameters which are unique rows of $X_\phi$and X_p.