Intro to RMark - part 2

Philip Dixon

9/28/2021

Contents

This document describes some useful additional features of RMark, then illustrates use of Pledger and Huggins heterogeneity models. It is intended to be read after the Intro to RMark document.

The example is the Phillpot skink data set provided by Shirley Pledger. The original data file is SkinksData.txt. I have converted it to RMark format and added a categorical covariate (sex = male/female). This is artificial and not based on the real problem. The augmented data file is skinkSex.txt.

skink.txt is a copy of the skink data with two numeric covariates. These will be used later to fit regression models for the capture probability.

```
library(RMark)
```

```
## This is RMark 2.2.7
##  Documentation available at http://www.phidot.org/software/mark/rmark/RMarkDocumentation.zip
```

```
skink <- import.chdata('skink.txt', field.types = c('n', 'n'))
head(skink)
```

```
##        ch   X1   X2
## 1 0000001 0.93 2.66
## 2 0000001 0.37 3.86
## 3 1011011 0.26 4.58
## 4 1111011 0.89 6.59
## 5 1000000 0.73 3.29
## 6 0000001 0.71 3.65
```

```
skinkSex <- import.chdata('skinkSex.txt', field.types = c('f'))
head(skinkSex)
```

```
##        ch    sex
## 1 0000001   male
## 2 0000001 female
## 3 1011011 female
## 4 1111011   male
## 5 1000000   male
## 6 0000001   male
```

Notes:

- Both covariates in skink.txt will be used in regression relationships, so field.types is a vector of 2 'n's (for numeric). The covariate in skinkSex.txt defines groups so it needs to be a factor variable. field.types=c('f') indicates that.

If you get the error

Error in nchar(data$ch) : 'nchar()' requires a character vector

when running a mark() model, the problem is that R has converted the capture history (ch) to a factor variable. Many functions do this automatically. Here's how to flip it back to character:

```
skink$ch <- as.character(skink$ch)
```

Note: This will not work to convert a numeric variable to a character string. That's because numeric values lose their leading 0's. Those matter to a capture history. To illustrate with two 5 occasion capture histories:

```
temp <- c(10011, 00110)
as.character(temp)
```

```
## [1] "10011" "110"
```

You could write code to examine the length of a character string and add leading 0's when "too short''. Easier to read the data correctly at the beginning.

Fitting Mtb models

Many versions of Mtb models are overparameterized. The problem is that the first capture probability for the last occasion is estimated as 1.0. That means you think you've seen everyone; the estimated f0 = 0 and the estimated N = Mt+1. This affects models with p=ft / c=ft or with p=ft / c=f0.

The most common solution is to assume that recapture probabilites are related to first capture probabilities. The common approach is an additive shift on the logit scale: logit ci = k + logit pi. That is implemented by the Mtb model in the code below. The formula is ~time + c, share=TRUE.

One easy-to-make error is to forget the share=TRUE. Without the share=TRUE, you are fitting a model with a constant recapture probability. Note in the results below, the 2nd (Mtb.V2) and 4th (Mtb.bad) models are specified differently but they are the same model.

```
run.Mtb <- function() {
  f0 <- list(formula=~1)
  f0s <- list(formula=~1, share=TRUE)
  ft <- list(formula=~time)
  ftb <- list(formula=~time + c)
  ftbs <- list(formula=~time + c, share=TRUE)

  Mtb.V1 <- mark(skink, model='Closed',
    model.parameters=list(p=ft, c=ft))
  Mtb.V2 <- mark(skink, model='Closed',
    model.parameters=list(p=ft, c=f0))
  Mtb <- mark(skink, model='Closed',
    model.parameters=list(p=ftbs))
  Mtb.bad <- mark(skink, model='Closed',
    model.parameters=list(p=ftb))

  collect.models()
}

Mtb.all <- run.Mtb()
```

```
##
## Note: only 12 parameters counted of 14 specified parameters

## AICc and parameter count have been adjusted upward

##
## Note: only 7 parameters counted of 9 specified parameters

## AICc and parameter count have been adjusted upward
```

```
##
## Note: only 7 parameters counted of 9 specified parameters

## AICc and parameter count have been adjusted upward
```

Notes:

- You have probably noticed the notes and/or warnings about the number of parameters. For example, "only 7 parameters counted of 9 specified parameters". The initial model fit by RMark is often overparameterized, even if the correctly parameterized model is not. I ignore these warnings and trust RMark to count correctly.

```
model.table(Mtb.all)
```

```
##           p     c f0                    model npar      AICc DeltaAICc
## 3     ~time ~time ~1  p(~time)c(~time)f0(~1)   14  38.09943  0.000000
## 1 ~time + c       ~1   p(~time + c)c()f0(~1)    9  42.59257  4.493147
## 2 ~time + c    ~1 ~1 p(~time + c)c(~1)f0(~1)    9 119.60206 81.502637
## 4     ~time    ~1 ~1     p(~time)c(~1)f0(~1)    9 119.60206 81.502637
##       weight Deviance
## 3 0.90435458 253.6504
## 1 0.09564542 268.3510
## 2 0.00000000 345.3605
## 4 0.00000000 345.3605
```

Model labelling in the model.table() output

I find the usual model names too detailed. model.table() can print the names of the R object instead.

```
print(model.table(Mtb.all, model.name=F), digits=3)
```

```
##           p     c f0   model npar  AICc DeltaAICc weight Deviance
## 3     ~time ~time ~1  Mtb.V1   14  38.1      0.00 0.9044      254
## 1 ~time + c       ~1     Mtb    9  42.6      4.49 0.0956      268
## 2 ~time + c    ~1 ~1 Mtb.bad    9 119.6     81.50 0.0000      345
## 4     ~time    ~1 ~1  Mtb.V2    9 119.6     81.50 0.0000      345
```

You can also reduce the number of digits printed by default by using and explicit print() with the digits= option. This is (at least roughly) the number of displayed digits, not the number past the decimal point.

Extracting results from RMark objects

The model table is a list of lists. Each component has all the Mark results from one model. You can access those either by number or by name (another reason model.name=F is useful).

```
summary(Mtb.all[[3]])
```

```
## Output summary for Closed model
## Name : p(~time)c(~time)f0(~1)
##
## Npar :  14  (unadjusted=12)
## -2lnL:  9.73767
## AICc :  38.09943  (unadjusted=34.005941)
##
## Beta
##                   estimate          se           lcl          ucl
## p:(Intercept)    0.6400377   0.1622723  3.219841e-01    0.9580913
## p:time2         -1.7851699   0.3471140 -2.465513e+00   -1.1048265
## p:time3         -1.7386516   0.3841153 -2.491518e+00   -0.9857857
```

```
## p:time4         -2.1441150    0.4796206 -3.084172e+00    -1.2040586
## p:time5         -3.8981345    1.0318886 -5.920636e+00    -1.8756328
## p:time6         -1.8440112    0.4929493 -2.810192e+00    -0.8778305
## p:time7         19.0926480 4087.8124000 -7.993020e+03  8031.2051000
## c:(Intercept)   -0.6390797    0.2005110 -1.032081e+00    -0.2460782
## c:time3          0.6390802    0.2691890  1.114698e-01     1.1666905
## c:time4          0.5649715    0.2643388  4.686750e-02     1.0830755
## c:time5         -1.2205022    0.3178568 -1.843502e+00    -0.5975029
## c:time6          0.6672504    0.2614943  1.547216e-01     1.1797791
## c:time7          1.1643458    0.2629432  6.489771e-01     1.6797146
## f0:(Intercept) -20.5556060 5202.6420000 -1.021773e+04 10176.6230000
##
##
## Real Parameter p
##         1         2         3         4        5         6 7
##  0.654762 0.2413793 0.2499997 0.1818182 0.037037 0.2307691 1
##
##
## Real Parameter c
##          2         3         4         5         6         7
##  0.3454546 0.5000001 0.4814814 0.1347518 0.5070422 0.6283783
##
##
## Real Parameter f0
##             1
##  1.182535e-09
```

```
# or summary(Mtb.all[['Mtb.V2']])
# or summary(Mtb.all$Mtb.V2)
```

You see the names of all the components by:

```
names(Mtb.all[[3]])
```

```
##  [1] "data"             "model"            "title"            "model.name"
##  [5] "links"            "mixtures"         "call"             "parameters"
##  [9] "time.intervals"   "number.of.groups" "group.labels"     "nocc"
## [13] "begin.time"       "covariates"       "fixed"            "design.matrix"
## [17] "pims"             "design.data"      "strata.labels"    "mlogit.list"
## [21] "profile.int"      "simplify"         "model.parameters" "results"
## [25] "output"
```

The results component is especially useful.
$results is a list with many components. These include $beta with the estimated parameters on the link scale and $beta.vcv with the variance covariance matrix.

I find two components of results really useful:

real: the backtransformed estimates of each capture, recapture, and f0 parameter.

derived: estimates of each derived parameter. N is the only derived parameter in the Closed, HetFull, and Huggins families of models.

If you want to see the variance covariance matrix for all derived parameters, that's in $results$derived.vcv. No, the VC matrix for the real parameters is not available. There is a list entry for real.vcv, but it's value is NULL (and it usually doesn't make sense if it was provided).

Confidence intervals for the real and derived parameters are Wald (asymptotic normal) intervals computed

on the internal scale (e.g. log for counts, logit for probabilities) then backtransformed to the real scale.

```
Mtb.all[['Mtb']]$results$real
```

```
##                estimate         se        lcl         ucl fixed    note
## p g1 t1       0.5020804  0.1107497  0.2973454   0.7061201
## p g1 t2       0.0866602  0.0499195  0.0268239   0.2462052
## p g1 t3       0.1268974  0.0718216  0.0391948   0.3411619
## p g1 t4       0.1120325  0.0662468  0.0330746   0.3175752
## p g1 t5       0.0219268  0.0148960  0.0057131   0.0804318
## p g1 t6       0.1230100  0.0730706  0.0358529   0.3460064
## p g1 t7       0.2231640  0.1243069  0.0658218   0.5394357
## c g1 t2       0.3867852  0.0426647  0.3071636   0.4729568
## c g1 t3       0.4913993  0.0404742  0.4129417   0.5702829
## c g1 t4       0.4561434  0.0391895  0.3809527   0.5333877
## c g1 t5       0.1297005  0.0269724  0.0853365   0.1922797
## c g1 t6       0.4825166  0.0385613  0.4079000   0.5579210
## c g1 t7       0.6563218  0.0367305  0.5812194   0.7243434
## f0 g1 a0 t1 51.0884370 46.0240600 11.2764230 231.4588900
```

```
Mtb.all[['Mtb']]$results$derived
```

```
## $`N Population Size`
##   estimate       se      lcl      ucl
## 1 219.0884 46.02406 179.2764 399.4589
```

Fitting Pledger models

I use the HetFull class of models for these. This implements all 8 possible combinations of time-specific, behavioural and individual heterogeneity. The HetClosed class implements the 4 combinations with behavioural effects. I have trouble fitting Mt and Mth models in the HetClosed class. Don't know if I'm omitting something obvious, but I get errors when I run HetClosed models. So, I just run the HetFull models.

The HetFull class fits models with 2 component finite mixtures for capture or recapture probabilities. If you want a mixture for any component, use a formula with +mixture in it. This can be combined with time or behaviour.

If you want to compare a model with mixtures to one without, you need to run the non-mixture model in the HetFull class. If you try to compare models from different classes, there is always the possibility that the likelihoods are not comparable. RMark will complain if you try but go ahead if you know it makes sense.

Here's code to fit 3 possible models with mixtures and one without. Mtb is fitting the non-mixture time + c model. Results are the same as before, except the # parameters is counted differently. Mh1 has a mixture for capture probabilities (same for 1st and recapture). Mh2 has a mixture for capture probabilities and a constant recapture probability.

```
run.pledger <- function() {
  ftbs <- list(formula = ~time + c, share=TRUE)
  fms <- list(formula = ~mixture, share=TRUE)
  fmtbs <- list(formula = ~mixture + time + c, share=T)

  Mtb <- mark(skink, model='FullHet',
    model.parameters=list(p=ftbs))
  Mh <- mark(skink, model='FullHet',
    model.parameters=list(p=fms))
  Mtbh <- mark(skink, model='FullHet',
    model.parameters=list(p=fmtbs))
```

```
  collect.models()
  }

fit.pledger <- run.pledger()
```

```
##
## Note: only 9 parameters counted of 10 specified parameters

## AICc and parameter count have been adjusted upward
```

```
print(model.table(fit.pledger, model.name=F), digits=4)
```

```
##   pi                    p c f0 model npar   AICc DeltaAICc weight Deviance
## 3 ~1 ~mixture + time + c   ~1  Mtbh   11 -94.67       0.0      1    127.0
## 2 ~1           ~time + c   ~1   Mtb   10  44.63     139.3      0    268.4
## 1 ~1            ~mixture   ~1    Mh    4  97.95     192.6      0    333.8
```

Notes on fitting Pledger models

Notice that the non-mixture Mtb still prints out like a mixture. You will see that each component has the same probabilities. And, you get an error about a singular parameter for beta 1. That's the mixture probability (pi). pi is irrelevant when the two components are identical. Ignore this.

Also, make sure to check the results, not just AICc, to see whether a mixture is reasonable. If one component has essentially no contribution (pi = 0.001), you have no mixture heterogeneity. The individuals have 1 component, even if 2 were "estimated".

Fiting Huggins models

The heterogeneity in Huggins's models is specified as a function of covariates. Those covariates are specified in the model formula. You can include some or all, or none (no heterogeneity). The model type I use is Huggins.

The general form is then the same as fitting a Pledger model. All models need to be in the Huggins class, even if no heterogeneity. This is especially important for Huggins because that likelihood is conditional on being seen at least once (so can measure covariates).

```
run.huggins <- function() {
  ftbs <- list(formula = ~time + c, share=TRUE)
  fms <- list(formula = ~ X1+X2, share=TRUE)
  fmtbs <- list(formula = ~ X1 + X2 + time + c, share=T)

  Mtb <- mark(skink, model='Huggins',
    model.parameters=list(p=ftbs))
  Mh <- mark(skink, model='Huggins',
    model.parameters=list(p=fms))
  Mtbh <- mark(skink, model='Huggins',
    model.parameters=list(p=fmtbs))

  collect.models()
}

fit.huggins <- run.huggins()
```

```
print(model.table(fit.huggins, model.name=F), digits=4)
```

```
##                     p c model npar AICc DeltaAICc weight Deviance
```

```
## 3 ~X1 + X2 + time + c    Mtbh   10 1285       0.0      1     1265
## 2           ~time + c     Mtb    8 1428     142.5      0     1655
## 1            ~X1 + X2      Mh    3 1473     188.4      0     1467
```

Notes on interpreting Huggins output

The beta table has the regression coefficients. The covariate model is logit(p) = X beta, so the coefficients act on the logit scale. You can estimate capture (or recapture probability for each individual two ways:

- use the covariate.predictions() function in RMark - see the documentation for that function. This computes both the real probabilities for specified covariate values and their variance-covariance matrix.
- use the regression coefficients and measured covariate(s) to calculate logit(p) for each individual, then back transforming: p = 1/(1+exp(-logit)).

When a model includes time, time is modeled as a factor and RMark uses the default "set first to 0" treatment for factor effects. That's why there is no estimated value for time1. It's 0.

The real parameters (for p or c) are the estimated probabilities for an individual with the average covariate value(s). For the skink data, the average covariate values are X1: 0.467 and X2: 4.329. On the logit scale an individual with those covariate values would have logit(p) = -2.277 + (-0.1037)*0.467 + 0.4507*4.329 = -0.375. Backtransforming, p = 1/(1+exp(-(-0.378))) = 0.407, which is what RMark reports.

Neither N nor f0 are parameters in a Huggins model. Mark estimates those as derived parameters. Those values, with se and 95% confidence intervals are available in the derived parameter results.

```
fit.huggins$Mtbh$results$derived
```

```
## $`N Population Size`
##   estimate       se      lcl      ucl
## 1 253.1356 43.61293 201.0519 387.2936
```