Using RMark

Philip Dixon

September 21, 2021

Program MARK

- a powerful tool for fitting models to data on marked individuals.
- can estimate detection probability, population size, survival, immigration and lots of other useful things
- 2015: fits 155 different models
- one program and GUI that replaces a fleet of other programs
- MARK.exe: compute engine, Mark_Int.exe: GUI to setup the model

But:

- GUI requires specifying PIMs and design matrices manually
- not reproducible (Mark is point-and-click, so no code => result)
- easy to make errors

RMark:

- Replaces the GUI (Mark_Int.exe)
- Uses R to set up models, collect and organize results
- Uses MARK to do the computations
- Analysis specified by R code, so is now reproducible
- and often simpler to write
- Written and maintained by Jeff Laake, NOAA Seattle

What you need:

- MARK downloaded and installed: http://www.phidot.org/software/mark/
- RMark package installed: install.packages('RMark')
    - RMark looks for MARK in program files (x86) folder

Lots of documentation:

- MARK: Cooch and White online book (1000+ pages)
- RMark:
    - Appendix C of Cooch and White
    - ?ABeginnersGuide in RMark
    - Jeff Laake RMark Workshop notes
    - In http://www.phidot.org/software/mark/rmark/RMarkDocumentation.zip
- This document based on Jeff's notes

Example: Reid deer mice data set,

- closed population
- subset of 38 individuals
- 6 trapping occasions
- data in readCH.txt on the class web site

Just enough MARK to understand what RMark does

- Data are capture histories
    - 100111 means individual caught time 1, not caught twice, then caught last 3 times
    - don't see 000000
- If can estimate number of never seen (000000), have estimate of population size

Various models for capture probability

- one p
- time dependent
- recapture different from first capture (behavioral heterogeneity)
- time + behavior
- individual heterogeneity

MARK parameterization

- p: 6 first capture probabilities
- c: 6 recapture probabilities
- PIM: Parameter index matrix. maps model parameters to real probabilities

Model	p1	p2	p3	p4	p5	p6	c2	c3	c4	c5	c6
Constant	1	1	1	1	1	1	1	1	1	1	1
behavior	1	1	1	1	1	1	2	2	2	2	2
Time dep	1	2	3	4	5	6	2	3	4	5	6
TB	1	2	3	4	5	6	7	7	7	7	7
General	1	2	3	4	5	6	7	8	9	10	11

- Interpretation of PIMs for closed population data with t capture occasions:
    - Mark always creates t 1st capture probabilities and t-1 recapture probabilities
    - These are the 11 columns in the above matrix
    - The numbers in the matrix are the parameters specified by each model (M0, Mt, Mb, ...)
    - e.g., constant: each of the 11 probabilities is determined by parameter #1. So, all probabilities are equal.
    - e.g., time dep: p1 is determined by parameter 1, p2 and c2 are determined by parameter 2, ...
    - e.g., TB: p1 - p6 are determined by 6 parameters, all capture probabilities by parameter #7
- Alternatively, Mark uses a design matrix and linear model to model real probabilities
- General PIM, constant model: $f(\pi_i) = \beta_0$
- Allows covariate-dependent models: $f(\pi_i) = \beta_0 + \beta_1 X_i$

RMark uses design matrix approach so you don't touch PIMs

- specify model using special variable names (e.g. time) or user-given covariates
- covariates could be occasion-specific
- or individual-specific
    - Huggins models for individual heterogeneity
- but you see the results of the PIM-based setup because RMark always reports estimates for all 11 (for t=6) parameters
    - This can be very useful because it provides that you specified the model you really wanted

```
library(RMark)
```

Reading data, setting up a model, running it

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

```r
reid <- import.chdata("reidCH.txt")
head(reid)
```

```
##       ch
## 1 111111
## 2 100111
## 3 110011
## 4 110111
## 5 111111
## 6 110111
```

```r
f0 <- list(formula=~1)
mb <- mark(reid, model="Closed", model.parameters=list(p=f0, c=f0))
```

```
##
## Output summary for Closed model
## Name : p(~1)c(~1)f0(~1)
##
## Npar :  3
## -2lnL:  97.98748
## AICc :  104.0946
##
## Beta
##                   estimate        se        lcl        ucl
## p:(Intercept)   -0.6525620 0.3230650 -1.2857695 -0.0193546
## c:(Intercept)    0.4554755 0.1772735  0.1080195  0.8029316
## f0:(Intercept)   1.0401168 1.0904394 -1.0971444  3.1773781
##
##
## Real Parameter p
##          1         2         3         4         5         6
##  0.3424124 0.3424124 0.3424124 0.3424124 0.3424124 0.3424124
##
##
## Real Parameter c
##          2         3         4         5         6
##  0.6119403 0.6119403 0.6119403 0.6119403 0.6119403
##
##
## Real Parameter f0
##         1
##  2.829547
```

Explanation:

- import.chdata() reads a text file with the capture data and potentially other information about individuals
  - format and contents described in Data file format (below)
- mark() actually calls 5 functions in sequence to process the data, make the design matrix, run the model (by writing a .inp file to the disk, then running MARK, which produces 3 output files), then collecting and organizing that output as R objects
  - arguments are the capture history data, the type of model, and the formulae for the two sets of parameters
  - specifics of the model specified by the formulae for each parameter
  - as model.parameters(list(parameter name = formula, . . .))

  - formulae are specified as list(formula= ~ something)
  - e.g., list(f0 = ~1) is a formula that defines a constant
  - common to define any needed formulae once, then use what you want in the model.parameters( )
  - e.g. model.parameters(list(p=f0, c=f0)) specifies a constant for p and a separate constant for c
- the output contains coefficient estimates (on link scale), back transformed coefficients (as time-specific values), and model fit information
- with big data sets want to run individual functions (only need to process the data once)
- with occasion-specific covariates, need to run individual functions
  - create design matrix data, then add information about covariates to that matrix. see ?dipper for more info

Data file format:

- Just like a MARK .inp file except without ;'s
  - can use convert.inp() if have a .inp file
- required variable is ch (capture history)
- can have a freq variable if multiple animals with same capture history
- 111111 7 is 7 animals seen every trapping night
- Can have other variables (you choose names). Need to add field.types= to import.chdata( ) to specify whether factor or numeric
- examples at the end of this document and in the RMark part 2 document

Running multiple models in RMark The preferred way to do this is:

- put all models into a dummy function
- run that function
- which packages all the information from each of the models

```
run.reid <- function() {
  f0 <- list(formula=~1)
  f0s <- list(formula=~1, share=TRUE)
  ft <- list(formula=~time, share=TRUE)
  fT <- list(formula=~Time, share=TRUE)
  ftb <- list(formula=~time)
  ftb2 <- list(formula = ~time + c, share=TRUE)

  m0 <- mark(reid, model='Closed', model.parameters=list(p=f0s))
  mt <- mark(reid, model='Closed', model.parameters=list(p=ft))
  mT <- mark(reid, model='Closed', model.parameters=list(p=fT))
  mb <- mark(reid, model='Closed', model.parameters=list(p=f0, c=f0))
  mtb <- mark(reid, model='Closed', model.parameters=list(p=ftb, c=ftb))
  mtb2 <- mark(reid, model='Closed', model.parameters=list(p=ftb2))
  return(collect.models())
}
```

Notes:

- recommended practice is to separate the formula definitions from the actual fit. That cuts the size of the mark() call, allows you to reuse formulae, and usually improves clarity of code.
- formula(=~time) specifies time as a factor. time does not need to be in the data frame. Each column of the capture history is treated as a separate time.
- formula(=~Time) specifies time as a regression: f(param) = b0 + b1*Time
- RMark uses a link function, f(), to connect the linear predictor (XB) to the parameter. Default is logit for probabilities and log for other parameters
- share = T in formula() tells RMark to use the specified models for multiple parameters,

  - so the f0s specification above says to use the same parameter for both p and c
  - the ft specification says to use the same time-specific parameters for both p and c
  - the ftb specification drops the share, so c will be estimated separately from p.
  - c could specified as time-specific values or a single parameter applied to all recapture times
  - the ftc specification says to use logit c = nu + logit(p) for the Mtb model
  - the fT specification (with capital T Time) says to use a logit linear model for time effects: logit(p) = b0 + b1 Time

- To check whether you specified the model you intended, I look at the estimated real parameter and check they have the pattern you intended. Example is in the summary() output given further down.

- model = 'Closed' specifies the type of model to be fit. This implicitly defines the parameter names for that model.

  - The MarkModels.pdf table lists all models that RMark can fit and their parameter names.
  - Models are defined by their Mark names.

  - Can often figure out model details by looking at the parameter names for the model.
  - I will provide model names for the models we will fit.
  - More information at the end of this document

To fit the models specified in the run.reid() function (printed output and warnings not included here)

```
reid.models <- run.reid()
```

```
##
## Note: only 6 parameters counted of 7 specified parameters

## AICc and parameter count have been adjusted upward

##
## Note: only 2 parameters counted of 3 specified parameters

## AICc and parameter count have been adjusted upward

##
## Note: only 10 parameters counted of 12 specified parameters

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

The output is the result of the collect.models() function at the end of the run.reid() function. That creates a list (class=marklist, inherits from list) of all the models in the environment.

```
print(model.table(reid.models), digits=4)
```

```
##            p     c f0                  model npar  AICc DeltaAICc   weight
## 2         ~1    ~1 ~1       p(~1)c(~1)f0(~1)    3 104.1     0.000 0.912229
## 6 ~time + c       ~1  p(~time + c)c()f0(~1)    8 110.5     6.418 0.036857
## 4     ~Time       ~1      p(~Time)c()f0(~1)    3 111.4     7.325 0.023418
## 5     ~time ~time ~1 p(~time)c(~time)f0(~1)   12 112.5     8.413 0.013588
## 1         ~1       ~1        p(~1)c()f0(~1)    2 113.6     9.466 0.008029
## 3     ~time       ~1      p(~time)c()f0(~1)    7 114.2    10.089 0.005878
##   Deviance
## 2    73.92
## 6    69.79
## 4    81.25
## 5    62.99
## 1    85.44
## 3    75.61
```

```
print(model.table(reid.models, model.name=F), digits=4)
```

```
##            p     c f0 model npar  AICc DeltaAICc   weight Deviance
## 2         ~1    ~1 ~1    mb    3 104.1     0.000 0.912229    73.92
## 6 ~time + c       ~1  mtb2    8 110.5     6.418 0.036857    69.79
## 4      ~Time       ~1    mT    3 111.4     7.325 0.023418    81.25
## 5      ~time ~time ~1   mtb   12 112.5     8.413 0.013588    62.99
## 1         ~1       ~1    m0    2 113.6     9.466 0.008029    85.44
## 3      ~time       ~1    mt    7 114.2    10.089 0.005878    75.61
```

Notes:

- model.table() prints a compact table for all models - sorted by AIC
- model.table(, model.name=F) uses the R list name to identify the models.
  - so I prefer to use descriptive names (m0, mt, mb, . . .) to store the results of mark()
- the print(, digits=4) makes the output fit on one line. Outside of RMarkdown, could just make the console window larger.

```
summary(reid.models[[2]])
```

```
## Output summary for Closed model
## Name : p(~1)c(~1)f0(~1)
##
## Npar :  3
## -2lnL:  97.98748
## AICc :  104.0946
##
## Beta
##                   estimate        se        lcl        ucl
## p:(Intercept)   -0.6525621 0.3230650 -1.2857696 -0.0193547
## c:(Intercept)    0.4554754 0.1772735  0.1080194  0.8029314
## f0:(Intercept)   1.0401169 1.0904395 -1.0971446  3.1773784
##
##
## Real Parameter p
##          1         2         3         4         5         6
##  0.3424124 0.3424124 0.3424124 0.3424124 0.3424124 0.3424124
##
##
## Real Parameter c
##          2         3         4         5         6
##  0.6119403 0.6119403 0.6119403 0.6119403 0.6119403
##
##
## Real Parameter f0
##         1
##  2.829548
```

```
summary(reid.models[['mt']])
```

```
## Output summary for Closed model
## Name : p(~time)c()f0(~1)
##
## Npar :  7  (unadjusted=6)
## -2lnL:  99.67496
## AICc :  114.184  (unadjusted=112.05505)
```

```
##
## Beta
##                    estimate           se           lcl          ucl
## p:(Intercept)   -0.4274465    0.3318809 -1.077933e+00     0.223040
## p:time2          0.5328086    0.4644356 -3.774852e-01     1.443102
## p:time3          0.1089907    0.4670112 -8.063512e-01     1.024333
## p:time4          0.4274456    0.4641207 -4.822310e-01     1.337122
## p:time5          1.0813751    0.4765166  1.474026e-01     2.015348
## p:time6          1.0813626    0.4765161  1.473910e-01     2.015334
## f0:(Intercept) -17.0460120 5843.6097000 -1.147052e+04 11436.429000
##
##
## Real Parameter p
##          1         2         3         4         5         6
##  0.3947362 0.5263162 0.4210521 0.4999998 0.6578952 0.6578924
##
##
## Real Parameter c
##          2         3         4         5         6
##  0.5263162 0.4210521 0.4999998 0.6578952 0.6578924
##
##
## Real Parameter f0
##             1
##  3.953766e-08
```

Notes:

- The results are stored as a list with one entry for each model
- summary(resultobject[[n]]) extracts the summary for model n
    - where resultobject is the name of the object storing the + e.g., reid.models for the code above, because results were stored there by reid.models <- run.reid()
- summary(resultobject[['mb']]) extracts the summary for the model called mb
- print() of a Mark Model opens the Mark output in NotePad (or other text editor)
    - just typing the model name is a default print().

    - most of the time you don't need to see the Mark output - it is voluminous
    - I generally only look at the summary() unless there is a problem.

Interpreting the results Look in the output from summary (above)

- Mark uses f0 = "number of individuals never seen" as the parameter to compute N
- the beta values are the coefficients for the linear predictors, i.e.,
    - logit for probabilities
    - log for f0
- the real parameters are the backtransformed (to natural scale) estimates
    - they are labelled by their time interval
    - so, 1 to 6 for p, 2 to 6 for c, 1 for f0

Examples of checking that you specified the model you actually wanted:

- model 2 is supposed to be the Mb model. When you look at the real parameters, you see:
- all p values are the same
- all c values are the same, but different from p
- That's what you expect for Mb
- model[['mt']] is supposed to be the Mt model. You see:

    – p values differ for each time
    – c values are the same for the matching time
    – Note p and c are not vertically aligned. First c value is for time 2.

To get the estimate of N (not f0):

```
reid.models[['mt']]$results$derived
```

```
## $`N Population Size`
##   estimate           se lcl      ucl
## 1       38 0.0002310427  38 38.00014
```

Notes:

- N is a derived parameter, computed from the estimate of f0
- Derived parameters are not automatically printed by summary()
- They are stored in the derived component of the results component of the model-specific result list.
- If there are multiple derived parameters, you'll see them individually

Defining models with time + behavioural variability Inspection of the estimates for model mtb shows this is overparameterized

- look at the estimates and variances for p[6] and f0

```
summary(reid.models[['mtb']])
```

```
## Output summary for Closed model
## Name : p(~time)c(~time)f0(~1)
##
## Npar :  12  (unadjusted=10)
## -2lnL:  87.05684
## AICc :  112.508  (unadjusted=108.07067)
##
## Beta
##                   estimate           se           lcl           ucl
## p:(Intercept)   -0.4274443 3.318810e-01     -1.077931     0.2230424
## p:time2         -0.2011643 5.493740e-01     -1.277937     0.8756087
## p:time3          0.0219794 6.228344e-01     -1.198776     1.2427349
## p:time4         -0.2657013 7.811182e-01     -1.796693     1.2652903
## p:time5          0.4274445 8.813697e-01     -1.300040     2.1549292
## p:time6         20.2358030 1.151386e+04 -22546.924000 22587.3960000
## c:(Intercept)    1.3862937 6.454973e-01      0.121119     2.6514684
## c:time3         -1.6486579 7.704478e-01     -3.158736    -0.1385801
## c:time4         -1.1786544 7.457144e-01     -2.640255     0.2829457
## c:time5         -0.5978362 7.497476e-01     -2.067341     0.8716691
## c:time6         -0.8602007 7.341964e-01     -2.299226     0.5788243
## f0:(Intercept) -20.7279210 5.710563e+03 -11213.432000 11171.9760000
##
##
## Real Parameter p
##          1         2   3         4         5 6
##  0.3947368 0.3478261 0.4 0.3333337 0.5000001 1
##
##
## Real Parameter c
##          2         3         4      5         6
##  0.7999999 0.4347826 0.5517241 0.6875 0.6285714
```

```
##
##
## Real Parameter f0
##             1
##  9.953556e-10
```

Three practical options for Mtb are:

- constrain last p to equal some other p: very arbitrary
- use a regression model for p, e.g. a logit linear model in time (formula = ~Time)
- connect p and c: e.g. a logit additive effect of 2nd capture: logit(c_i) = nu + logit(p_i). That's model mtb2.

```
summary(reid.models[['mtb2']])
```

```
## Output summary for Closed model
## Name : p(~time + c)c()f0(~1)
##
## Npar :  8
## -2lnL:  93.85477
## AICc :  110.5123
##
## Beta
##                   estimate        se        lcl       ucl
## p:(Intercept)  -0.5700003 0.3795046 -1.3138292 0.1738287
## p:time2         0.0441047 0.5212030 -0.9774533 1.0656627
## p:time3        -0.6469194 0.6052358 -1.8331817 0.5393428
## p:time4        -0.5101460 0.6580226 -1.7998704 0.7795783
## p:time5         0.0241109 0.7094466 -1.3664044 1.4146262
## p:time6        -0.0739973 0.7415602 -1.5274554 1.3794608
## p:c             1.2634764 0.6971101 -0.1028594 2.6298121
## f0:(Intercept)  1.2596006 1.4988303 -1.6781068 4.1973080
##
##
## Real Parameter p
##          1         2        3         4         5         6
##  0.3612368 0.3714747 0.228479 0.2534783 0.3668186 0.3443434
##
##
## Real Parameter c
##          2         3         4         5      6
##  0.6764666 0.5116371 0.5457046 0.6720754 0.6501
##
##
## Real Parameter f0
##         1
##  3.524014
```

Model averaging Mark parameters Can do model averaging across a list of Mark models very easily

```
model.average(reid.models)
```

```
##    par.index  estimate         se
## 1          1 0.3478323 0.07635719
## 2          2 0.3494347 0.08099012
## 3          3 0.3453501 0.08524154
## 4           4 0.3469126 0.08903639
## 5           5 0.3553349 0.09817983
## 6           6 0.3623058 0.12269873
## 7           7 0.6116021 0.06050249
## 8           8 0.6010348 0.05903126
## 9           9 0.6054390 0.05013745
## 10         10 0.6139526 0.04805464
## 11         11 0.6133941 0.04666495
## 12         12 2.7110810 3.18183256
```

Notes:

- averages the real parameters
  - the last real parameter is the MA estimate of f0 = # not seen
  - to get a model averaged estimate of N, add f0 and # individuals seen at least once
  - since # seen is treated as a known constant, se of N = se of f0
- includes all the models included in the reid.models list
- uses the revised variance estimator
  - adding revised=F shifts to Buckland estimator
- Parameters are listed by number. Look at the order of real parameters to identify them
  - so for these data, the 1 - 6 row are for the 6 1st capture probabilities
  - 7-11 are the 5 recapture probabilities
  - 12 is f0
- if you want to omit a model, remove it from the "model collector" function, and refit the models.
  - here, model average of f0 is not useful because it's not identified in model mtb. Would want to rerun model list without mtb. mtb2 is just fine.

It is possible to get RMark to model average derived parameters, e.g. N, instead of adding f0 to # seen. Jeff Laake has code to do that, which would be useful when you want to automate producing a report.

Including covariates Either for capture parameters or # not seen

Mark is especially useful when the data has groups of individuals defined by values of a measured variable. Covariates that vary between individuals need to be handled differently, e.g. by a Huggins model.

The reidCH.txt file adds sex (2 levels, m and f) and age (3 levels, 0, 1, and 2) information to each capture history. Sex is the first "extra" variable and will be treated as a factor. We will focus on sex.

```r
reids <- import.chdata('reidCHs.txt', field.types=c('f', 'n') )
head(reids)

run.reid2 <- function() {
  f0 <- list(formula=~1)
  fs <- list(formula=~sex)

  mb <- mark(reids, model='Closed', model.parameters=list(p=f0, c=f0, f0=fs),
    groups=c('sex') )
  mps <- mark(reids, model='Closed', model.parameters=list(p=fs, c=fs, f0=fs),
    groups=c('sex') )
  mcs <- mark(reids, model='Closed', model.parameters=list(p=f0, c=fs, f0=fs) ,
    groups=c('sex') )
  mpcs <- mark(reids, model='Closed', model.parameters=list(p=fs, c=fs, f0=fs),
    groups=c('sex') )
  return(collect.models())
}

reid.models2 <- run.reid2()
```

Notes:

- field.types= specifies type for each variable (other than ch and freq). ‘f’ is factor, ‘n’ is numeric

To fit a factor variable, you need to tell RMark, and hence MARK, that there are groups in the data. This creates duplicate sets of PIMs, one for each group. These share parameters unless you tell RMark that the parameter varies by the group. The f0 formula specifies a single value for all groups. The fs formula species differences between groups.

```
model.table(reid.models2, model.name=F)
```

```
##      p    c   f0 model npar     AICc DeltaAICc     weight Deviance
## 1   ~1   ~1 ~sex    mb    4 156.0251  0.000000 0.60980882 103.6059
## 2   ~1 ~sex ~sex   mcs    5 158.0402  2.015118 0.22264676 103.5301
## 3 ~sex ~sex ~sex  mpcs    6 159.9952  3.970088 0.08377221 103.3753
## 4 ~sex ~sex ~sex   mps    6 159.9952  3.970088 0.08377221 103.3753
```

```
summary(reid.models2[['mcs']])
```

```
## Output summary for Closed model
## Name : p(~1)c(~sex)f0(~sex)
##
## Npar :  5
## -2lnL:  147.77
## AICc :  158.0402
##
## Beta
##                   estimate        se        lcl       ucl
## p:(Intercept)   -0.5360076 0.3116887 -1.1469175 0.0749022
## c:(Intercept)    0.5031035 0.2484298  0.0161811 0.9900258
## c:sexm          -0.0976383 0.3547110 -0.7928719 0.5975953
## f0:(Intercept) -0.2050929 1.9364435 -4.0005222 3.5903364
## f0:sexm        -0.1852332 2.2605105 -4.6158340 4.2453675
##
##
## Real Parameter p
##                      1         2         3         4         5         6
## Group:sexf 0.3691168 0.3691168 0.3691168 0.3691168 0.3691168 0.3691168
## Group:sexm 0.3691168 0.3691168 0.3691168 0.3691168 0.3691168 0.3691168
##
##
## Real Parameter c
##                      2         3         4         5         6
## Group:sexf 0.6231884 0.6231884 0.6231884 0.6231884 0.6231884
## Group:sexm 0.6000000 0.6000000 0.6000000 0.6000000 0.6000000
##
##
## Real Parameter f0
##                      1
## Group:sexf 0.8145716
## Group:sexm 0.6768361
```

Notes:

- You notice that the real parameters tables now have 2 rows, one for each sex. If you had 4 groups, you would get four rows.
- Model mcs has the same 1st capture probabilities and different recapture probabilities for the two sexes.

    - In the output, the two rows of p real parameters are the same (no sex diff. in the model)
    - The two rows of c real parameters are different (model includes a sex difference)
- The beta coefficients table includes an intercept and sexm effect.
    - RMark uses the default R parameterization of factor effects models
    - intercept is the estimated mean (here on the logit scale) for the base level of the factor. In the default R parameterization, the base level is the first level if the factor. f is alphabetically before m, so female is the base level.
    - sexm is the estimated difference between male and female, as male mean - female mean.
- All models include f0=fs. That specifies different "# not seen" for the two sexes. I suggest this is biologically necessary when you specify groups.
    - That's because groups make MARK create two separate sets of data and PIMs. If you don't use a group effect for f0, you are forcing the # unseen males to equal # unseen females. Unlikely.

    - So, if I use groups, I always include f0=grouping variable
- If you use a factor variable and groups for one model, you need to use groups for all models (see model mb above). That's because MARK needs to have the same sets of PIMs for all models.

Cleaning up after Mark

Mark leaves lots of files in your working folder

- Each time you run MARK, you get 4 files, even when rerunning the same model
- some needed - those for "current" models. rest are excess
- cleanup() deletes excess files. Will confirm each one unless you trust RMark and tell it not to ask

```
cleanup(ask=F)
```

Models that MARK and RMark can fit

- Conditional (Huggins) and unconditional (Otis) closed population models: population size
- Pledger mixture models for individual heterogeneity
- Cormack-Jolly-Seber: survival from live recaptures
- Brownie: dead recoveries
- reverse-time (Pradel): immigration / birth rate / lambda
- robust design
- multistate versions of most of above
- many of above with random effects or mixtures
- occupancy models
- nest survival

A full list with model names, parameter lists, and relevant examples is in MarkModels.pdf in the RMark folder (usually My Documents/R/win-library/version/RMark) where version is the R version, e.g. 3.6 or 4.0.

As of 2015, RMark can fit 97 of the 155 models that MARK will fit. Most of the not-currently-supported models are generalizations of supported models that allow mis-identification of marks (e.g., genotyping error).