Fitting Detection Functions for Line Transect Sampling with ADMB and R2admb

Jeff Laake

November 15, 2012

ADMB was designed for optimization of non-linear models. However, even if the model is non-linear, portions of the model may be linear or parameters may be linear functions of covariates through a link function (e.g., log). An example is fitting a detection function g(x) (probability of detecting an object at distance x; g(0)=1) to line transect data. Two common detection functions are the half-normal and the hazard rate which can be expressed as:

$$g(x) = \exp(-x^2/(2\sigma^2)$$

and

$$g(x) = 1 - \exp(-(x/\sigma)^{-p}) \ .$$

The objective function is the negative log-likelihood which is:

$$-\log\prod_{i=1}^{n} f(x_i) = -\log\prod_{i=1}^{n}\frac{g(x_i)}{\int_0^w g(u)du} = -\log\sum_{i=1}^{n} g(x_i) + n\log(\int_0^w g(u)du) \ ,$$

where w is the half-width of the line-transect. Often the detection function also depends on other variables ($\mathbf{z}$). These variables can change the scale (σ) and shape (p) of the detection function and their relationship can be modeled by expressing the log of the parameters as a linear function of the covariates to bound the values of the parameters to be greater than zero. To restrict the shape of the hazard function, you can also replace -p with -(1+p) so the exponent is bounded below by 1. With the inclusion of $\mathbf{z}$, the log-likelihood becomes:

$$-\log\prod_{i=1}^{n} f(x_i) = -\log\prod_{i=1}^{n}\frac{g(x_i,\mathbf{z}_i)}{\int_0^w g(u,\mathbf{z}_i)du} = -\log\sum_{i=1}^{n} g(x_i,\mathbf{z}_i) + \sum_{i=1}^{n}\log(\int_0^w g(u,\mathbf{z}_i)du)\ .$$

Covariates that explain differences in detectability as a function of distance can be incorporated into the scale parameter σ with a design matrix with a log-link to ensure $\sigma > 0$. The design matrix for each parameter is input as data. For example, if σ depends on sex, the formula would be ~sex and the design matrix would have 2 columns: an intercept and a column for male which is the amount males differ from females using the standard treatment contrast for constructing design matrices from factor variables with a formula in R. The equation for sigma would be: $\sigma = \exp(X\beta)$ where the design matrix X has a row for each observation and 2 columns, the first being all 1s and the second being 0 for females and 1 for males. The parameter vectorβ would have 2 values for the intercept and sex(male) effect. Any formula based on the data could be used for σ and p.

To make this generally useful we can structure the DATA section of the tpl file to allow any number of parameters and any number of columns for the design matrix for each parameter. This approach easily generalizes for any non-linear model where the parameters are expressed as a linear function of the covariates on the link-scale. For this specific example in distcov.tpl (shown at end of document), we also need to specify which detection function to use because that defines which parameters are estimated. The input data are:

- number of observations
- vector of distances for each observation
- transect half-width
- 1 or 2 to designate which detection function to use
- number of parameter types (not needed for this example because is depends on the detection function; but written this way to show the generality of the approach for other problems)
- the link function for each parameter (1=identity, 2=log, 3=logit); this could be expanded and only log is needed for this example
- the number of columns in the design matrix for each parameter
- a 3d array of the design matrices with the first dimension being the parameter

The only parameter in the PARAMETER section is the ragged-array beta which has a row (vector) for each parameter type and the vector contains a parameter for each column in the design matrix for that parameter. The remainder of the definitions in the section are temporary variables used to hold calculated real (inverse-link) parameters and the integral.

The PROCEDURE section distcov.tpl contains two functions:

- reals: which computes the real parameters from design matrix, beta parameters and the specified link function; additional links could be easily added.
- fct: which computes the value of the detection function for a specified distance and real parameter vector; additional detection functions could be easily added as well.

The calculation of the objective function has 2 loops. The first loops over the each parameter type and creates the real parameters for each observation and stores the values in the column of parmat for each parameter type. The second loops over the observations to calculate the negative log-likelihood value for each observation which includes computing the integral with the built-in function adromb which calls fct to integrate the detection function.

The TPL file can stand alone and any number of models can be run by modifying the DAT file. Here I demonstrate how to use R and R2admb to create the design matrix, runs the model and retrieves the results. The R code creates the DAT file from the data frame and the formula(s) for the parameters. The following is a function fitds which takes the following arguments:

- obs - data frame of observations
- width - transect half-width
- detfct - "hn" for half-normal and "haz" for hazard-rate
- scale.formula - formula for σ
- exponent.formula - formula for p only used for hazard-rate

With the arguments specified, it creates the necessary DAT file to run the model.

```
> fitds=function(obs,width,detfct="hn",scale.formula=~1,
+                  exponent.formula=~1)
+ {
```

```
+   # create scale design matrix with formula and data
+   scale_mat=model.matrix(scale.formula,obs)
+   if(detfct=="haz")
+     exponent_mat=model.matrix(exponent.formula,obs)
+   # write out data file
+   con=file(paste(tplfile,".dat",sep=""),open="wt")
+   writeLines(as.character(nrow(obs)),con)
+   write(obs$distance,con,ncolumns=1)
+   writeLines(as.character(width),con)
+   if(detfct=="haz")
+   {
+      writeLines("2",con)
+      writeLines("2",con)
+      writeLines("2  2",con)
+      writeLines(paste(ncol(scale_mat)," ",
+                       ncol(exponent_mat),sep=""),con)
+      write(t(scale_mat),con,ncolumns=ncol(scale_mat))
+      write(t(exponent_mat),con,ncolumns=ncol(exponent_mat))
+   }else
+   {
+      writeLines("1",con)
+      writeLines("1",con)
+      writeLines("2",con)
+      writeLines(paste(ncol(scale_mat),sep=""),con)
+      write(t(scale_mat),con,ncolumns=ncol(scale_mat))
+    }
+   close(con)
+   run_admb(tplfile)
+   results=read_admb(tplfile)
+   cnames=paste("scale:",colnames(scale_mat),sep="")
+   if(detfct=="haz")
+     cnames=c(cnames,paste("exponent:",colnames(exponent_mat),sep=""))
+   names(results$coefficients)=cnames
+   rownames(results$vcov)=cnames
+   colnames(results$vcov)=cnames
+   return(results)
+ }
```

To run the model, we need to compile the TPL file, write the data, run the model and

extract the results. Here we use the modularized approach by calling compile_admb, run_admb, and read_admb because we only want to compile the TPL file once and then run each model by modifying the data file. The following example uses the golf tee data that are contained in the package mrds (mark-recapture distance sampling) that is on CRAN.

```
> tplfile="distcov"
> # compile tpl file
> compile_admb(tplfile)
> # get data from golf tee data in mrds
> library(mrds)
> data(book.tee.data)
> obs=book.tee.data$book.tee.dataframe
> obs=obs[obs$observer==1,]
> obs=obs[obs$detected==1,]
> # fit for different models
> model1=fitds(obs,4,"haz",~1,~sex)
> model1

Model file: distcov
Negative log-likelihood: 152.601
Coefficients:
   scale:(Intercept) exponent:(Intercept)         exponent:sex
             0.719816             1.670280            -1.773860

> model2=fitds(obs,4,"haz",~sex,~sex)
> model2

Model file: distcov
Negative log-likelihood: 150.171
Coefficients:
   scale:(Intercept)            scale:sex exponent:(Intercept)
             0.364933             0.711980             0.841306
        exponent:sex
             0.691031

> model3=fitds(obs,4,"hn",~sex+size)
> model3
```

```
Model file: distcov
Negative log-likelihood: 150.081
Coefficients:
scale:(Intercept)         scale:sex        scale:size
        0.1709850         0.5865760         0.0262098

> model4=fitds(obs,4,"hn",~sex+exposure)
> model4

Model file: distcov
Negative log-likelihood: 149.69
Coefficients:
scale:(Intercept)         scale:sex    scale:exposure
       -0.0105602         0.7154600         0.2711800

>
```

```
DATA_SECTION
    init_int n;                          // number of distances
    init_vector xs(1,n);                 // distances
    init_number width;                   // truncation half-width of transect
    init_int ifct;                       // type of detection function 1=hn, 2=haz
    init_int pt;                         // number of parameter types
    init_ivector links(1,pt);            // link number but would rather use strings; not sure how 1=identity, 2=log, 3=logit
    init_ivector k(1,pt);                // vector of number of parameters for each type; cols in design matrix
    init_3darray dm(1,pt,1,n,1,k);       // design matrices - one for each parameter type
PARAMETER_SECTION
    init_matrix beta(1,pt,1,k);          // beta parameters for each parameter type
    matrix parmat(1,pt,1,n);             // matrix of parameter values; 1 to n and 1 to pt types of parameters
    vector par;                          // holds par for an observation likelihood calculation
    number mu;                           // holds single integral fct(x)
    objective_function_value f;          // negative log-likelihood
PROCEDURE_SECTION
    int i, j;
// Create matrix of real parameter values which is pt rows and n columns
    for (i=1;i<=pt;i++)
    {
     parmat(i)=reals(dm(i),beta(i),links(i));
    }
// loop over each observation computing sum of log-likelihood values
    f=0;
    for (j=1;j<=n;j++)
    {
       par=column(parmat,j);
       mu=adromb(&model_parameters::fct,0,width,8);
       f+= -log(fct(xs(j))) + log(mu);
    }

////////////////////////////////
// Computes reals from betas
////////////////////////////////
FUNCTION dvar_vector reals(dmatrix& dm, dvar_vector& beta, int ilink)
// dm is the design matrix
// beta is vector of parameters - length macthes ncol(dm)
// ilink is type of link function
     dvar_vector tmp;
          if(ilink==1)
          tmp=dm*beta;
     if(ilink==2)
          tmp=exp(dm*beta);
     if(ilink==3)
          tmp=1/(1+exp(-dm*beta));
     return tmp;
//////////////////////////////////////////////////////////
// Computes normalizing constant int fct(x) - 0 to width
//////////////////////////////////////////////////////////
FUNCTION dvariable fct(const dvariable& x)
// x is integration variable
// ifct is index for function read from data
    dvariable tmp;
    if(ifct==1)
      tmp=exp(-.5*x*x/(par(1)*par(1)));
    if(ifct==2)
    {
      if(x<0.0000001)
            tmp=1;
          else
```

```
    tmp=1-exp(-(pow(x/par(1),-(1+par(2)))));
}
return tmp;
```

∞