Interfacing ADMB with R

Most researches that would find ADMB beneficial to their work already use the free statistical package R. Hence, to facilitate the use of ADMB, it is important to link ADMB with R. Currently there are several approaches that have been used to link ADMB with R. The simplest is to run an ADMB executable from within R and pass data using files. This is the system that is used for the ADMB based GLMM function for R (http://otter-rsch.com/admbre/examples/glmmadmb/glmmADMB.html). ADMB also has the capacity to create DLLS with R, but this has been infrequently used. Anders Nielsen has created an R script for reading in ADMB default output files (See Appendix I) Steve Martel, from UBC, has created an R script to read ADMB report files (See Appendix II). The function is capable of reading single variables, vectors, and 2-D arrays (including ragged arrays) and produces a list object. Mike Prager and his NMFS team has created ADMB2R, a collection of AD Model Builder routines for saving complex data structures into a file that can be read into R with a single command. ADMB2R provides the means to transfer data structures significantly more complex than simple tables. Jon Schnute has developed a set of R functions and a related GUI (PBSadmb) to facilitate the use of ADMB and to interface ADMB with R. In addition to reading model output into R, PBSadmb has other features like facilitating instillation and compiling code. Ben Bolker has developed a similar set of R scripts. John Nash has developed a Google Summer of Code project to interface ADMB with R. FLR is a comprehensive R project for fisheries realated analyses and would benify greatly from interfacing ADMB with R (http://flr-project.org/).

Despite several projects working on interfacing ADMB with R, a more complete interface with R is needed so that complete control of ADMB can be conducted within R. The list below summarizes some of the functionality needed when interfacing ADMB with R. Appendix III provides specifications of some R functions that are missing from the current projects.

- Reading in default ADMB output
- Reading in custom output
- Creating data and pin files
- Reading in data and pin files
- Writing tpl files
- Compiling and running admb programs

- Converting R models

ADMB2R

Mike Prager and his colleagues at NMFS have developed middleware to interface modeling codes to the statistics package R. They use their interface for several purposes: to examine model results and diagnostics during stock-assessment workshops; to automate graph and table creation for formal reports; to transfer model output to R for ad hoc analyses and forecasting; and to store model data for analyses conducted months or years later.

They created a set of ADMB libraries that write ADMB data structures into a text file that is easily readable by R with a single function call. When read, the data form an R list, whose components may be vectors, matrices, character values, or contained lists (e.g., lists of matrices).

http://www.sefsc.noaa.gov/mprager/rinter.html

PBSadmb

Jon Schnute and Rowan Haigh of the Pacific Biological Station in Nanaimo Canada have developed a set of R functions and a related GUI to facilitate the use of ADMB and to interface ADMB with R. PBSadmb comes with a version of ADMB and uses Rtools to install a free C++ compiler (GCC), which facilitates the installation of ADMB. Menus on the GUI call the R functions to carry out the translation to C++, compilation, linking, and running of the ADMB code and configuring run options. The GUI brings up your choice of editor so you can create or edit the ADMB files (e.g. tpl, dat, and pin). The output can be loaded from the ADMB created text files into R with a click of a button on the GUI or by calling the R functions directly from R. The GUI includes a library of examples files to help guide new users. PBSadmb includes several features that aid the use of ADMB within R such as ensuring that variable names match between the source file for ADMB (called a template) and a corresponding source file for R. It also has several features to aid in the development of Bayesian MCMC models such as plotting posterior traces, densities, and pair wise correlations. PBSadmb can be downloaded at: http://code.google.com/p/pbs-admb/ and http://cran.r-project.org/

R2admb

Ben Bolker has developed an interface for ADMB with a function for calling ADMB and returning the results as an object of class "admb", and print/coef/summary assessors etc.

http://r-forge.r-project.org/projects/r2admb/

Google School of Code

Google Summer of Code (http://code.google.com/soc/) is a global program that offers student developers stipends to write code for various open source software projects. Google works with several open source, free software, and technology-related groups to identify and fund several projects over a three month period. Through Google Summer of Code, accepted student applicants are paired with a mentor or mentors from the participating projects, thus gaining exposure to real-world software development scenarios and the opportunity for employment in areas related to their academic pursuits. In turn, the participating projects are able to more easily identify and bring in new developers.

John Nash (Telfer School of Management of the University of Ottawa) has put together a proposal for the GSoC to integrate R with Automatic Differentiation Tools (http://rwiki.sciviews.org/doku.php?id=developers:projects:gsoc2010:adinr). One of the tools that he has identified is ADMB.

Appendix I: Reading ADMB default output into R

This R function written by Anders Nielsen will read in the default ADMB output files. (The “file” argument is the directory and root of the ADMB program e.g. file=“C:/ADMB/examples/simple”)

```
read.fit<-function(file){
# Function to read a basic AD Model Builder fit.
# Use for instance by:
# simple.fit <- read.fit('c:/admb/examples/simple')
#
# Then the object 'simple.fit' is a list containing sub-objects
# 'names', 'est', 'std', 'cor', and 'cov' for all model
# parameters and sdreport quantities.
#
  ret<-list()
  parfile<-as.numeric(scan(paste(file,'.par', sep=''),
  what='', n=16, quiet=TRUE)[c(6,11,16)])
  ret$nopar<-as.integer(parfile[1])
  ret$nlogl<-parfile[2]
  ret$maxgrad<-parfile[3]
  file<-paste(file,'.cor', sep='')
  lin<-readLines(file)
  ret$npar<-length(lin)-2
  ret$logDetHess<-as.numeric(strsplit(lin[1], '=')[[1]][2])
```

```
sublin<-lapply(strsplit(lin[1:ret$npar+2], ' '),function(x)x[x!
=''])
ret$names<-unlist(lapply(sublin,function(x)x[2]))
ret$est<-as.numeric(unlist(lapply(sublin,function(x)x[3])))
ret$std<-as.numeric(unlist(lapply(sublin,function(x)x[4])))
ret$cor<-matrix(NA, ret$npar, ret$npar)
corvec<-unlist(sapply(1:length(sublin), function(i)sublin[[i]]
[5:(4+i)]))
ret$cor[upper.tri(ret$cor, diag=TRUE)]<-as.numeric(corvec)
ret$cor[lower.tri(ret$cor)] <- t(ret$cor)[lower.tri(ret$cor)]
ret$cov<-ret$cor*(ret$std%o%ret$std)
return(ret)
}
```

Appendix II: Reading ADMB custom output into R

Steve Martell has written a useful R function that reads the contents of a report file (or any output file) and stores the contents in R in the form of a list object. This function is capable of reading single variables, vectors, and 2-D arrays (including ragged arrays). The R-code was inspired by some earlier code developed by George Watters. The format of *.tpl code requires the object name be printed first then the value(s), e.g.:

```
report<<"Biomass"<<endl<<bt<<endl;
```

Once the above line has been read into R the bt vector will be available as A$Biomass, where A is the list object, and Biomass is the name of the bt vector. To use this function, copy and past the R-code below and save this file as "reptoRlist.R". Then source this file in R (or put it at the top of your R-script). The function requires a file name argument (e.g., fn="MyModel.rep").

To read the contents of a report file, simply use:

A=reptoRlist(fn)

Then all of the objects in your report file will be stored in the list object A.

```
reptoRlist = function(fn)
{
  ifile=scan(fn,what="character",flush=T,blank.lines.skip=F,quiet
  =T)
  idx=sapply(as.double(ifile),is.na)
  vnam=ifile[idx] #list names
  nv=length(vnam) #number of objects
  A=list()
  ir=0
  for(i in 1:nv)
  {
    ir=match(vnam[i],ifile)
    if(i!=nv)
       irr=match(vnam[i+1],ifile) else irr=length(ifile)+1 #next
  row
    dum=NA
    if(irr-ir==2)
       dum=as.double(scan(fn,skip=ir,nlines=1,quiet=T,what=""))
    if(irr-ir>2)
      dum=as.matrix(read.table(fn,skip=ir,nrow=irr-ir-1,fill=T))
    if(is.numeric(dum))#Logical test to ensure dealing with
  numbers
    {
      A[[ vnam[i ]]]=dum
    }
  }
  return(A)}
```

Appendix III: Specification of R functions to generate dat and pin files for the PBSadmb R project

The objective of these functions is to generate dat and pin files used with AD Model Builder (ADMB) directly from R. This will facilitate the use of R to create data that can be used in ADMB based programs.

Function name: **make.dat.object**

This function creates an R list that contains objects that correspond to the init_ variables defined in the DATA_SECTION of an ADMB tpl file. The objects in the list are in the same order as in the tpl file and, if possible, have the same dimentions.

Parameters

tplfile – name and directory of the tpl file

Actions

Find the DATA_SECTION of the tpl file

Find the variables defined by the prefix int_

Create an R list object containing each of the int_ variables in order with the appropriate dimensions (if possible, otherwise use default dimensions)

Populate the objects with zeros

Add an associated object for each variable containing the definition from the tpl file [e.g. “init_number Y(StartYear,EndYear)”]

Create a warning message if the object dimensions can not be completely defined if the dimensions are based on an _init variable or a non _init variable

Variables to be implemented

int, ivector, imatrix, number, vector, matrix, 3darray, 4darray, 5darray, 6darray, 7darray

Function name: **populate.dat.object**

This function populates (and optionally creates) an R list that contains objects that correspond to the init_ variables defined in the DATA_SECTION of an ADMB tpl file. The objects are populate from a ADMB dat file that corresponds to the TPL file.

Parameters

datfile – name and directory of the dat file

datlist – dat list that is to be populated (could be the return object not a parameter; if null generate dat list)

tplfile – name and directory of the tpl file

Actions

Read in the data from the dat file

Populate each object in the dat list with the data in order

If prior objects are used to dimention the object, use that data. This may require simultaneous reading of the TPL file

If the datlist parameter is null, create a new dat list

Create a warning if the variable is a ragged array (i.e its dimensions are based on a vector or an array)

Function name: **write.dat.object**

This function creates an ADMB dat file using the information contained in a R list object.

Parameters

datfile – name and directory of the dat file to create or overwrite

datlist – dat list that is to be used to create the dat file

Actions

Write the dat list to the file in order

Function name: **check.dat.object**

This function checks to ensure that a n R list that contains objects that correspond to the init_ variables defined in the DATA_SECTION of an ADMB tpl file correctly corresponds to the tpl file.

Parameters

datlist – dat list that is to be populated (could be the return object not a parameter; if null generate dat list)

tplfile – name and directory of the tpl file

Actions

Find the DATA_SECTION of the tpl file

Find the variables defined by the prefix int_

Compare the R list object with each int_ variable, in order, and with the appropriate dimensions (the dimensions may depend on previously defined objects in the R list)

Function name: **make.pin.object populate.pin.object write.pin.object**

These functions create, populate, and write to file an R list that contains objects that correspond to the init_ variables defined in the PARAMETER_SECTION of an ADMB tpl file.

Repeat the above using the pin file and the PARAMETER_SECTION

Notes
Need to deal with additional prefixes of bounded_ and dev_.

Need to deal with number_vector etc.

May need to read the DATA_SECTION to get the dimensions of variables

Variables to be implemented

number, vector, matrix, 3darray, 4darray, 5darray, 6darray, 7darray

Random effects parameters

Other issues

Implementation of ragged arrays

Example code (poorly written by Maunder and fixed by Taylor)

```
make.dat.object<-function(tplfile="C:\\Documents and
Settings\\mmaunder\\My Documents\\Work\\admb
project\\Rproject\\test.tpl",ncols=15)
{
    Data<-list()

    rawtpl <-
read.table(file=tplfile,col.names=c(seq(1,ncols,by=1)),fill=T,q
uote="",colClasses="character",nrows=-1)

    print("***** Start looking for section *****")
    startrow <- grep("DATA_SECTION",rawtpl[,1])
    endrow <- grep("PARAMETER_SECTION",rawtpl[,1])
    if(length(startrow)>0 & length(endrow)>0) print("found
section heads")

    print("***** Start looking for data variables*****")
    initrows <- grep("init_",rawtpl[,1])
    initrows <- initrows[initrows>startrow & initrows<endrow]
    Ninits <- length(initrows)
    if(Ninits>0) print(paste("found",Ninits,"rows containing
'init_'"))
    nameslist <- rep(NA,Ninits)

    for(i in 1:Ninits)
    {
        irow <- initrows[i]
        type <- substr(rawtpl[irow,1],6,nchar(rawtpl[irow,1]))
        object <- rawtpl[irow,2]
        print(paste("type =",type))
```

```
        print(paste("object =",object))

        # split apart object at any punctuation mark
        objectparts <- strsplit(object,"[[:punct:]+' ']")[[1]]
        nameslist[i] <- objectparts[1]

        dims <- as.numeric(objectparts[-1])
        print(dims)
        dims <- dims[!is.na(dims)]
        print(dims)

        if(type=="number") Data[[i]] <- 0
        if(type=="vector")
        {
            Data[[i]] <- rep(0,dims[2]-dims[1]+1)
        }
        if(type=="matrix")
        {
            ncol <- dims[4] - dims[3] + 1
            nrow <- dims[2] - dims[1] + 1
            Data[[i]] <- matrix(0,nrow=nrow,ncol=ncol)
        }
        print(Data)
    }
    names(Data) <- nameslist
    return(Data)
}

dat1 = make.dat.object('c:/ss/tests/makedat/example.tpl')
```