TOSSM package user's manual

Dave Gregovich, Karen Martien and Mark Bravington

The Testing of Spatial Structure Models (TOSSM) package is a set of compiled functions in the 'R' language. The purpose of this document is to serve as a reference document to anyone using the TOSSM package to conduct simulation-based performance testing. We begin by introducing the TOSSM project and defining terms and concepts integral to TOSSM. We then describe the package in detail, with particular attention to the biological interpretation of some of the arguments required by the package. Though our focus here is on the TOSSM package itself, we provide Appendices describing the structure of the TOSSM datasets, which are a required input to the package. Details on how to develop an interface for testing a particular analytical method are also included.

General Introduction and definition of terms

The primary purpose of the TOSSM package is to provide a framework for testing the performance of analytical methods for defining management units from genetic data. In the TOSSM package, an analytical method's performance is tested in terms of 1) accurately inferring the spatial structure of simulated populations and 2) setting appropriate management boundaries accordingly (Figure 1).

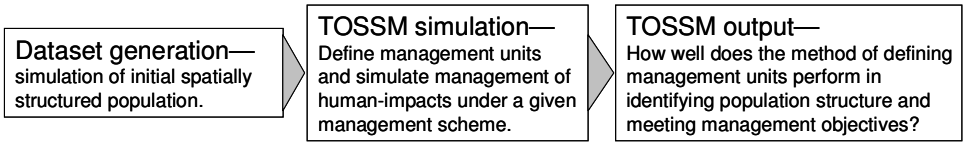


Figure 1. Overview of the TOSSM package.

Archetypes and Breeding populations (BPs)

The TOSSM datasets fall into five broad categories of population structure, which are referred to as 'Archetypes' (Figure 2). The number of simulated breeding populations (BPs) is determined by the number of breeding populations that exist in the initial, simulated dataset. These initial datasets exist as 'rmetasim' landscape objects. To generate each initial dataset, the number of BPs, carrying capacity for each BP, and a dispersal rate between BPs was specified. The TOSSM datasets, as well as further details on their generation can be found on the TOSSM website (http://swfsc.noaa.gov/tossm.aspx).

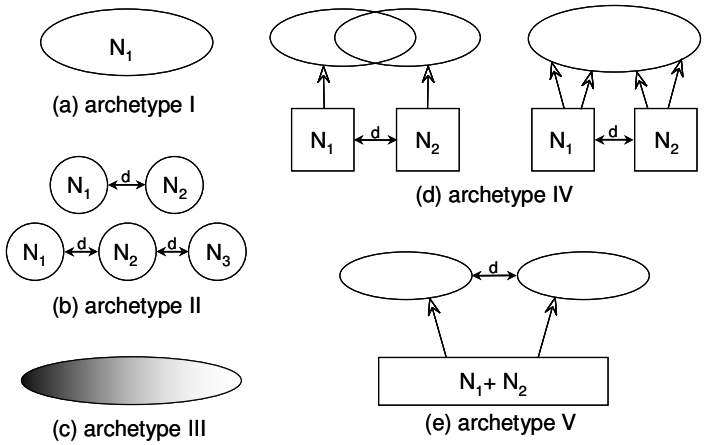


Figure 2. The five archetypes represented in the TOSSM datasets. (a) Archetype I—A single, mixed population that serves as a control. (b) Archetype II—Stepping-stone dispersal pattern between two or three populations, with only adjacent populations mixing. (c) Archetype III—Diffusion-type, where isolation across the population continuously increases with distance. (d) Archetype IV—Two discrete breeding grounds with feeding grounds that overlap partially or completely. (e) Archetype V—A single breeding population with two separate feeding grounds.

Breeding population polygons, sampling polygons, and historic harvest polygons

A TOSSM simulation is spatially explicit, with the spatial location of important components defined by polygons. These spatial components include breeding populations, genetic sampling sites, historic harvest areas, and management units. Polygons used in a TOSSM simulation are of class gpc.poly, implemented by the R package gpclib. Therefore, some of the required inputs to TOSSM are polygons of this class.

Breeding population polygons (bp.polys) define the geographic ranges of the breeding populations. They can be geographically discrete, contiguous, or overlapping (Figure 3). There must be one polygon for each breeding population. Sampling polygons define the spatial areas for which genetic samples are collected in a simulation. There is flexibility as to the number and geographic extent of these polygons. The constraints are that 1) the sampling polygons must be within the geographic extent of the breeding population polygons (i.e., you can't collect samples from a site where there is no breeding population), and 2) there can be no overlap of sampling polygons. Uniform sampling of the entire simulated landscape can be achieved by defining a single sampling polygon equal to the combined spatial extent of the breeding populations (Figure 3)

The interaction between bp.polys and sample.polys will determine whether the samples from a given sampling site come exclusively from one breeding population or represent a mixture of multiple breeding populations. Mixed samples can occur if a sample.poly is in a region of overlap between two or more bp.polys or straddles the boundary between two contiguous bp.polys.

Boundary-setting algorithm's (BSAs)

A Boundary-setting algorithm (BSA) is a function that uses genetic data to define management units. Though there are many analytical methods that accept and analyze genetic data and output information on population structure, most of them do not go so far as to explicitly define management units. Thus, while a BSA will generally have at its core an analytical method for detecting and describing population structure, it must also include a mechanism for using the results of that method to define management units. Specifically, the BSA decides if and how the breeding populations should be split spatially into management units. This could be as simple as deciding whether to manage two sampling polygons separately or as one management unit. Alternatively, if there are many sampling polygons, there could be a number of different decisions the algorithm must make about splitting or grouping the various polygons into management units. An added level of complexity is needed if the BSA works not on the sampling polygon level but at the level of the individual, in which case the BSA decides which *individuals*, each individual having x- and y-coordinates, should belong to which management unit.

A BSA must interface with the TOSSM package by accepting simulated genetic and abundance information from the package and returning a recommended way of dividing the landscape into management units. A BSA does not have to use all information about a simulated population that TOSSM makes available; different BSAs may rely on different components of the simulated data provided by a TOSSM simulation. Further details on creating a BSA are given in Appendix A.

Quota calculating algorithm

Different management bodies have different algorithms for calculating quotas. The algorithm that the IWC uses for calculating the number of whales that can be killed in a management unit is called the catch-limit algorithm, or CLA. It calculates quotas based on the estimated abundance of a management unit and information on historic catches—see Cooke (1994) for further details. The Potential Biological Removal scheme (PBR) is the mechanism used for determining the number of animals that can killed under the U.S. Marine Mammal Protection Act (Taylor et al., 2000). It is a much simpler algorithm than the CLA, and uses only abundance estimates and their uncertainty and the maximum possible population growth rate for the species in question in order to calculate a quota. A default value for the maximum population growth rate is used when a species-specific estimate is not available.

The CLA and PBR are currently the only algorithms available within the TOSSM package for setting quotas and managing harvest. However, the modular architecture of the package makes it relatively easy for other management schemes to be added.

Harvest Interval

In TOSSM simulations, harvesting effort is not uniformly distributed across each management unit. Rather, effort is concentrated near the left edge of each management unit. This spatial bias in harvest is meant to simulate a situation in which the harvesters wish to minimize the distance they must travel in order to meet their quota, and so concentrate their effort close to their home base, which is assumed to be to the left of the study area. To implement this spatial bias, the entire simulated landscape is divided into vertical strips (harvest intervals) of equal width. In each

simulation year, TOSSM will attempt to take the entire quota for a management unit from the left-most harvest interval in that management unit. If there are not enough animals present in the first harvest interval to meet the quota, all animals in the first interval will be harvested and the program will attempt to remove the remainder of the quota from the next interval to the right. Harvest progresses toward the right until the quota has been met.

The degree to which harvest is spatially biased is controlled by changing the width of the harvest intervals. Defining intervals that are very narrow relative to the x-range of the breeding populations will result in a strong spatial bias. Setting the harvest interval width equal to the x-range of the entire simulated landscape (i.e., all breeding populations combined) will result in the harvest being taken uniformly across each management unit.

Schedule of simulation events

Before running a TOSSM simulation, the user must establish a schedule of events, which defines the simulation timeline. There are up to three distinct phases in a TOSSM simulation:

1) Historic phase – This phase allows for historic harvest that pre-dates modern management. Historic harvest can be implemented either by simply specifying the level of depletion (abundance divided by carrying capacity) at the end of the historic phase or by defining historic harvesting areas and specifying the number of individuals taken from each area in each year of the historic phase.
2) Managed phase – In this phase, management units are defined by a BSA. These management units are then managed using quotas calculated by the chosen quota calculating algorithm.
3) Recovery phase – This phase is an optional period that allows post-harvest population recovery before the end of the simulation.

In addition to defining the beginning and ending years of each of these phases, the schedule is also used to specify the years in which genetic samples are collected, abundance estimates made, management units re-defined, and quotas calculated.

A simple example

The main function in the TOSSM package is `run.tossm`. We describe all of the arguments to this function in detail in the next section. Here, we describe the minimum set of arguments that must be specified in order to run a TOSSM simulation. To do so, we work through the example included in the help file for `run.tossm`, which consists of two geographically contiguous populations and eight discrete sampling sites. There is a five year historic phase during which breeding population 1 is depleted to 30% of carrying capacity. The study area is then divided into five equally-sized management units and managed harvest proceeds for ten years using the default quota calculating algorithm (PBR). The simulation ends with a five year recovery phase. All arguments not specified are left at program defaults (see 'Details of run.tossm' for default values).

The remainder of this section presents all of the code needed to run the example, interspersed with explanatory text. Appendix B presents the code without the explanatory text, so that the user can simply cut and paste the code directly into the R command window. (Note: characters printed in `Courier New` font denote 'R' commands and functions.)

Installing the package

R can be downloaded from the CRAN website (http://cran.r-project.org). Install the program by opening the self-extracting executable and following the on-screen instructions. In order to use the TOSSM package, you must have R version 2.8 or later installed. Once R is installed, you can install the TOSSM package by opening R, going to the 'Packages' menu and selecting 'Install package(s)…' You will be presented with a list of CRAN mirrors, from which you should choose the mirror nearest you in order to achieve the highest download speed. You can then choose 'tossm' from the list of available packages. All packages required by the TOSSM package will automatically be installed as well. To use the TOSSM package type

```
library(tossm)
```

in the R command window.

Required arguments

`rland`

The first argument required by `run.tossm` is a dataset in the form of an 'rmetasim' landscape object, which represents the initial state of the simulated population. A large number of appropriate datasets, representing a wide

variety of types and degrees of population differentiation, are available for download from the TOSSM website (http://swfsc.noaa.gov/tossm.aspx). The datasets can be accessed by selecting 'Download Center' from the left menu of the website. For this example, we will use the datasets from Archetype 2, scenario 3, which is comprised of two breeding populations exchanging dispersers at a rate of $5x10^{-5}$ per year. To down load these datasets, select Archetype 2, scenario 3, replicates 1-50 from the download center, and click 'download zip file.' Extract the downloaded files to a folder called 'TOSSM.example' in My Documents.

We will start with the first replicate from this scenario, dataset `Arch2_sc3_1.rda`. To load this dataset into R, first change your working directory to the folder where you saved the datasets using the '`setwd()`' command (note that you will need to insert the correct user name in order for the following line of code to work):

```
setwd('C:/Documents and Settings/insert user name/My Documents/TOSSM.example')
```

You can now load the dataset into your workspace using the following command

```
load('Arch2_sc3_1.rda')
```

This will create in your R workspace an object called '`rland.end`', which can be passed as the first argument to `run.tossm`. Details on the structure of rmetasim landscape objects is given in Appendix C.

`bp.polys`

The breeding population polygons used to define the geographic ranges of the breeding populations should be passed as a list of polygon objects of the class `gpc.poly`. For this example, we must specify two polygons, since there are two breeding populations in the `rland` object we are using. The following commands generate a list of two rectangular polygons, the first spanning from 0 to 50 on the x-axis and from 0 to 100 on the y-axis, and the second spanning from 50 to 100 on the x-axis and 0 to 100 on the y-axis (Figure 3):

```
bp.polys<-list()
bp.polys[[1]]<-matrix(c(0,0,50,0,50,100,0,100),nrow=4,byrow=T)
bp.polys[[2]]<-matrix(c(50,0,100,0,100,100,50,100),nrow=4,byrow=T)
my.bp.polys<-lapply(bp.polys,as, 'gpc.poly')
```

`sample.polys`

As with the breeding population polygons, the polygons used to define the sampling sites should be passed as a list of polygon objects of the class `gpc.poly`. The following commands generate a list of eight discrete sampling polygons (Figure 3):

```
px <- list(c(5,20),c(30,45),c(55,70),c(80,95))
py <- list(c(10,40),c(60,90))
polys <- do.call('c',lapply(py, function(y){lapply(px,function(x){
      rbind(cbind(x,y[1]),cbind(x[2:1],y[2]))
      })}))
my.sample.polys <- lapply(polys, function(p) as(p,'gpc.poly'))
```

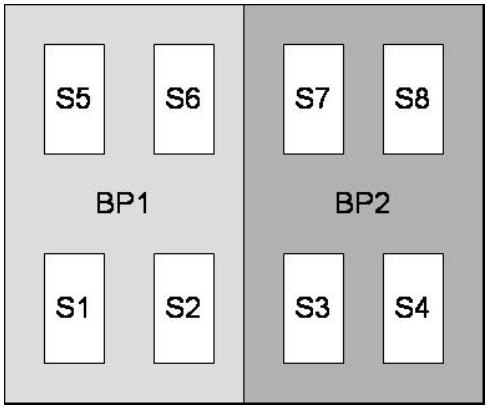


Figure 3. The breeding population polygons (BP1 and BP2) and sampling site polygons (S1-S8) used in the example.

schedule

This argument is a list specifying the timing of key events in a simulation. The list must include the following vectors as its components:

stop.years–The total number of years in the simulation
gs.years—The years in which genetic samples are taken
abund.est.years—The years in which abundance estimates are obtained
pre.RMP.years—The years predating modern management (may include historic harvest)
CLA.years—Years in which quotas are calculated using the chosen quota calculating algorithm
post.RMP.years —The years after modern management during which no harvest occurs (optional)
BSA.years—Years in which the boundary-setting algorithm is called to detect spatial structure and define management units accordingly.

These different events are contingent on each other in the following way:

- Genetic samples must be obtained before boundaries are set using the BSA (collecting them in the same simulation year is fine).
- Boundaries must be set and abundance estimates must be obtained before an initial catch limit is set. In other words, the first element in BSA.years and abund.est.years must be less than or equal to the first element in CLA.years. Subsequent calls to the quota calculating algorithm, however, can be made without additional calls to the BSA or abundance estimates.
- All CLA.years take place during the modern management phase.

Though it is possible to set up a simulation schedule manually by creating vectors for each of the schedule components and passing them as arguments to run.tossm, the TOSSM package includes a function called def.make.schedule, which is a convenient way of setting up a schedule of all simulation events. It is recommended to use def.make.schedule initially to run simulations; the user can manually customize this schedule once they are more comfortable with the package. def.make.schedule requires the following arguments:

n.pre.RMP – Number of historic harvest years (before modern management)
n.RMP – Number of years during which management units are managed according to the quotas set by the quota calculating algorithm
n.post.RMP – Number of 'recovery' years, during which no harvest occurs
abund.gap – Interval (in years) on which abundance estimates are obtained and quota calculating algorithm is called

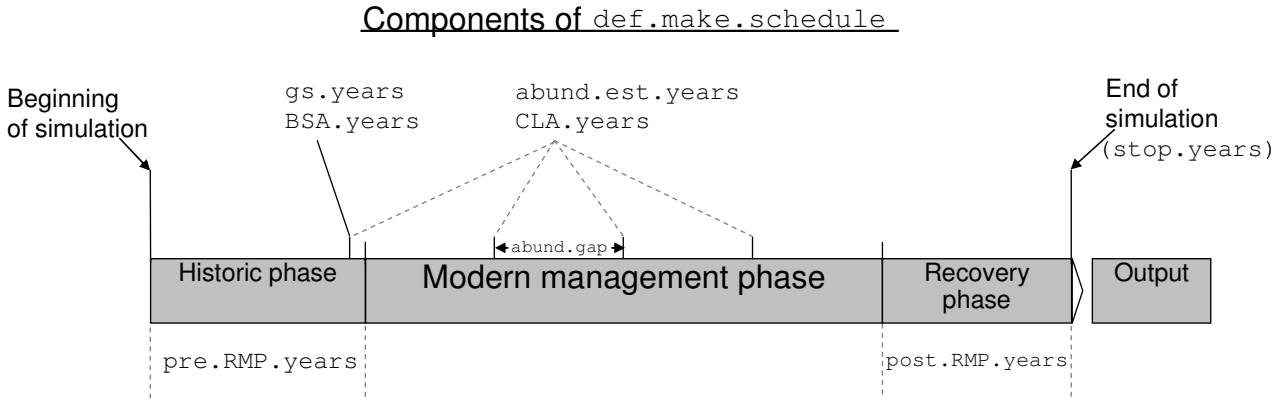
def.make.schedule generates a schedule in which genetic samples are taken (gs.years) and boundaries are set (BSA.years) in the last pre-RMP year only. Abundance estimates are obtained (abund.est.years) and a quota is calculated (CLA.years) in the first RMP year and every abund.gap years there after until the end of the RMP phase. Thus, the following call to def.make.schedule

```
my.schedule <- def.make.schedule(n.pre.RMP=5,n.RMP=10,n.post.RMP=5,abund.gap=2)
```

will return a list with the following components:

```
$stop.years
[1] 20
$gs.years
[1] 5
$abund.est.years
[1] 6 8 10 12 14
$pre.RMP.years
[1] 1 2 3 4 5
$CLA.years
[1] 6  8 10 12 14
$post.RMP.years
[1] 16 17 18 19 20
$BSA.years
[1] 5
```

These events can be envisioned schematically as follows:



n.samples

The number of genetic samples to collect within each sample polygon in each of 'gs.years' is specified as a vector, with length equal to the number of sampling polygons. Alternatively, a single number can be specified, in which case that number of samples is taken from every sample polygon.

BSA, BSA.args

These arguments are used to specify the BSA that should be used to define management units and any additional arguments required for that BSA. The default values for these arguments (BSA=fixed.MU.BSA and BSA.args=list(n.mus=1)) result in the entire study area being managed as a single management unit. For this example, we will use the default BSA, but change the BSA arguments so that the study area is managed as 5 equally-sized management units:

```
my.BSA <- fixed.MU.BSA
my.BSA.args <- list(n.mus=5)
```

harvest.interval

This argument controls the width of the harvest intervals, and therefore determines the degree of spatial bias in harvest. For this example, we will set a moderately strong bias by making the harvest intervals $1/10^{th}$ the width of the study area, or 10 units wide:

```
my.harvest.interval.width <- 10
```

Running one simulation replicate

Once all of the necessary arguments have been defined, a single TOSSM simulation can be run as follows:

```
my.TOSSM.sim <- run.tossm(rland=rland.end, bp.polys=my.bp.polys,
      schedule=my.schedule, n.samples=25, sample.polys=my.sample.polys,
      BSA=my.BSA, BSA.args=my.BSA.args,
      harvest.interval=my.harvest.interval.width,plot.polys=T)
```

The performance of the BSA can be assessed by examining the abundance of the breeding populations through time (my.TOSSM.sim$abund.b), the catch in each management unit during the Modern Management Phase (my.TOSSM.sim$catches), and the amount of effort the harvesters had to expend in order to achieve that catch (my.TOSSM.sim$effort), which is expressed as the average x-coordinate of harvested individuals (assuming the 'home base' of harvesters is at x=0). Plots of the trajectories of these values can be obtained using the following function (Figure 4):

```
trajectory.plotter(my.TOSSM.sim)
```

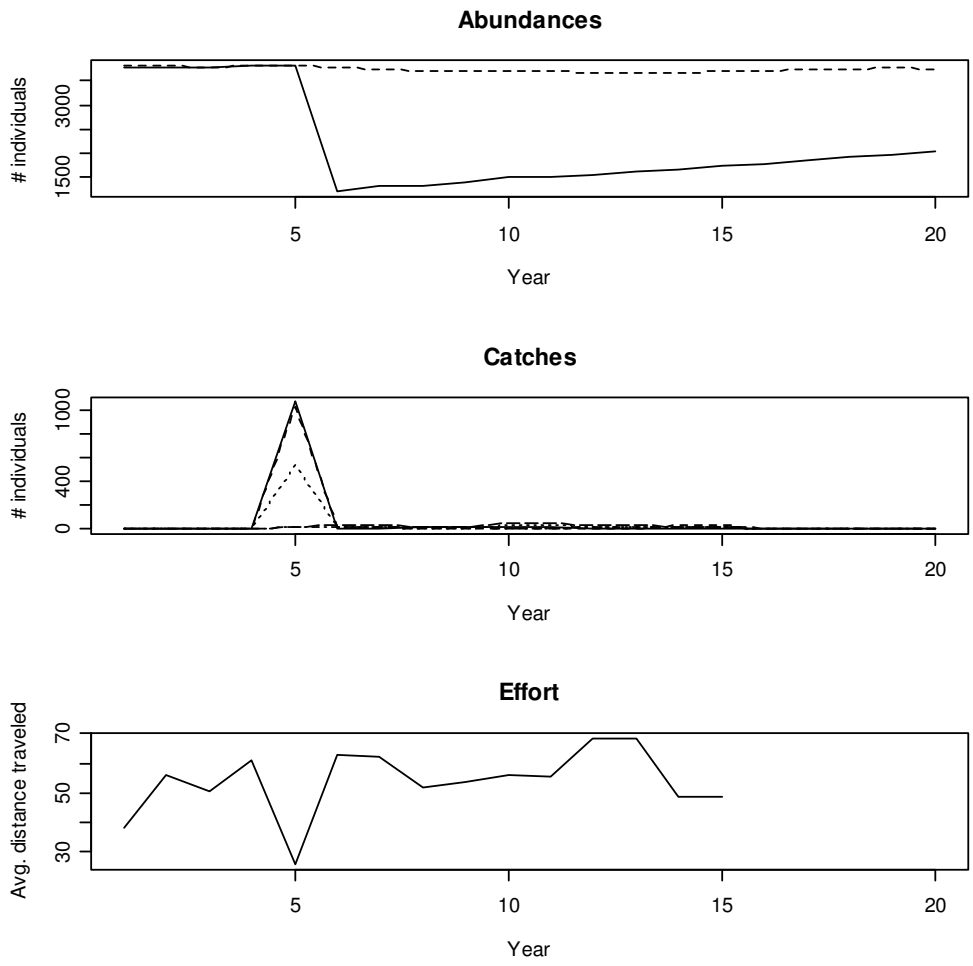


Figure 4. Graphs produced by `trajectory.plotter`.

The abundance trajectory shows the initial depletion of breeding population 1 at the start of the Modern Management Phase (year 6) to 30% of its initial abundance (the default value; see below), followed by a steady recovery. The catch trajectory shows a large spike in catch corresponding to this initial depletion.

Running multiple replicates

Assessing the performance of a BSA will require running multiple replicate simulations to look at the variability of results. This can be done by embedding the above call to run.tossm in a '`for`' loop or '`apply`' statement. The following lines of code will run five TOSSM simulations using five replicate TOSSM datasets. These datasets (Arch2_sc3_1.rda through Arch2_sc3_5.rda) should be stored in My Documents in a folder called TOSSM.example. Note that these simulations could take a few minutes to complete, depending on the speed of the computer being used.

```
num.reps <- 5
datasets <- paste("Arch2_sc3_",1:num.reps,".rda",sep="")
TOSSM.sims <- lapply(1:num.reps, function(i){
      load(datasets[i])
      run.tossm(rland=rland.end, bp.polys=my.bp.polys, schedule=my.schedule,
      n.samples=25, sample.polys=my.sample.polys, BSA=my.BSA,
      BSA.args=my.BSA.args, harvest.interval=my.harvest.interval.width)
})
```

The result (TOSSM.sims) is a list of five TOSSM objects across which summary statistics can be calculated. For instance

```
abund1.sum <- sapply(TOSSM.sims, function(i) i$abund.b[1,])
sapply(1:20,function(y) median(abund1.sum[y,]))
```

will calculate the median abundance in breeding population 1 across replicates in each simulation year.

The details of `run.tossm`

The main function in the TOSSM package is `run.tossm`. The following is the full compliment of arguments to `run.tossm`, including their defaults. This information is duplicated in the help file for `run.tossm` (accessed by typing `?run.tossm` at the R command prompt).

```
run.tossm(rland=NULL, bp.polys, schedule=NULL, n.samples=NULL, sample.polys,
       initial.depletion=.30, historic.removals=NULL, historic.polys=NULL,
       BSA=fixed.MU.BSA, BSA.args=list(n.mus=1), stop.years=schedule$stop.years,
       gs.years=schedule$gs.years, abund.est.years=schedule$abund.est.years,
       pre.RMP.years=schedule$pre.RMP.years, CLA.years=schedule$CLA.years,
       post.RMP.years=schedule$post.RMP.years, BSA.years=schedule$BSA.years,
       harvest.interval, genetic.sampler=def.genetic.sampler, abund.for.10pc.CV=70000,
       quota.calc=PBR, quota.args=list(r.max=0.04,F.r=1,multiplier=1), CLA.prog=NULL,
       CLA.dir=NULL, plot.polys=FALSE, seed=-1)
```

`rland` – a dataset in the form of an 'rmetasim' landscape that represents the initial state of the simulated population. Many such datasets, along with further details on how those datasets were the generated, are available at the TOSSM website (http://swfsc.noaa.gov/tossm.aspx). Details on the structure of `rmetasim` landscape objects is given in Appendix C.

`bp.polys` - A list of polygons (class `gpc.poly`) representing the spatial extent of breeding populations.

`n.samples` – A vector indicating the number of genetic samples to collect within each sample polygon in each of `gs.years`. If a single number is given, then that number of samples is taken from every sample polygon.

`sample.polys` – A list of polygons (class `gpc.poly`) representing the geographic extent of sampling areas. These polygons must not overlap, and must be fully contained within one or more breeding population polygons represented by `bp.polys`.

`initial.depletion` – A vector indicating the level to which populations are depleted at the beginning of the modern management phase. The length of the vector should equal the number of populations. If not, the end of the vector will be padded with 0.99. Thus, the default is that population 1 is depleted to 0.30 of carrying capacity, while all other populations are at 0.99 of K. The initial depletion argument is only used if `historic.removals` is null, otherwise the `initial.depletion` argument will be ignored.

`historic.removals` – This argument is used to specify in detail any historic harvest before the period in which modern management takes place. It consists of a matrix with each row representing a single `pre.RMP.year` and each column representing an area of historic harvest. The entries are the number of animals killed in each historic harvest area in each `pre.RMP.year`. Note that values of 0 in this matrix will be automatically changed to 1 by `run.tossm` due to idiosyncracies of the CLA quota calculating program, MANAGE-D.exe. If `historic.removals` is set, the argument `initial.depletion` will be ignored.

`historic.polys` – A list of polygons (class `gpc.poly`) representing the geographic extent of historic harvest areas. As with the sampling polygons, these historic harvest polygons must not overlap each other, and must be fully contained within the `bp.polys`.

`BSA` – A function used to set management unit boundaries. Leaving the `BSA` and `BSA.args` arguments to `run.tossm` at the defaults (`BSA=fixed.MU.BSA` and `BSA.args= list(n.mus=1)`) results in all breeding

population polygons being combined into a single management unit, and is therefore useful for making sure that `run.tossm` is running properly before attempting to use a different BSA. See Appendix A for the details of setting up a BSA.

`BSA.args` – A list of up to two extra arguments passed to the boundary-setting algorithm. The arguments required vary between BSAs. For instance, `fixed.MU.BSA`, the package default, requires a single extra argument (`n.mus`) specifying the number of MUs into which the simulated landscape should be developed, while `hyptest.network.BSA` requires one argument (`sig.level`) specifying the α level at which the *p*-value from a G-test is considered statistically significant. Because the components of `BSA.args` can themselves be lists of arbitrary length, it is possible to write BSAs that require more than two arguments. For instance, `structure.BSA` effectively requires 12 arguments, which are organized into two lists called `mainparams` and `BSA.params`.

`schedule` – A list specifying the timing of key events during the simulation. The schedule can be generated manually or using the helper function `def.make.schedule`. See 'A simple example' for details.

`stop.years, gs.years, abund.est.years, pre.RMP.years, CLA.years, post.RMP.years, BSA.years` – integer vectors dictating the timing of events in `schedule`. If values are set for any of these arguments, they will overwrite the corresponding values contained in the argument `schedule`.

`harvest.interval` – A vector of length one, which specifies the width of the harvest intervals into which the study area is divided. This argument controls the degree to which harvest is spatially concentrated within a management unit. Specifying a harvest interval width that is very small relative to the x-range of the breeding populations will maximize the degree to which harvest is spatially concentrated. Setting the harvest interval width equal to the x-range of the entire simulated
landscape (i.e., all breeding populations combined) will result in the harvest being taken uniformly across each management unit.

`genetic.sampler` – The genetic sampler dictates the sampling design used for obtaining genetic samples. The default (`def.genetic.sampler`) draws a sample that is random with respect to age, sex, and all other demographic characteristics. The distribution of samples within sampling sites is proportional to density. If density is uniform across a sampling sites, samples will be uniformly distributed. Otherwise, the density of samples will reflect the density of animals across the sampling site. The default sampler should be fine for most applications, but the user can supply an alternate sampler if desired.

`abund.for.10pc.CV` – The coefficient of variation (CV) of the abundance estimates is assumed to be proportional to 1/sqrt(abund). The constant of proportionality is chosen so that the CV is 0.1 at the value specified by this argument. The default value for this argument (70,000) results in a CV of 0.30 when abundance equals 7,500.

`quota.calc` – The algorithm to use for calculating catch quotas. There are two quota calculating algorithms included with the TOSSM package - `CLA` (IWC, 1994), which is the algorithm used by the International Whaling Commission to set quotas, and `PBR` (Taylor et al. 2000), which is the algorithm for calculating quotas under the U.S. Marine Mammal Protection Act.

`quota.args` – Extra arguments passed to the quota calculating algorithm. The three arguments currently accepted are `multiplier`, `r.max` and `F.r`. `multiplier` is used to scale the quota-- default is 1. `multiplier` is used by both CLA and PBR. `r.max` and `F.r` are only used by `PBR`. `F.r` is the recovery factor and `r.max` is the maximum population growth rate for the species in question (Taylor et al., 2000). The default value for `r.max` (0.04) is the default value used for PBR calculations for any cetacean species for which a species-specific estimate is not available.

`CLA.prog` – The path to the Fortran program that implements the CLA. The default should be correct unless MANAGE-D.exe or MANAGE-D (in the case of unix-alikes) has been moved subsequent to installation of the TOSSM package.

`CLA.dir`—The location of some temporary files created by the CLA program.

`plot.polys` – A flag indicating whether or not the breeding polygons, sampling polygons, and historic removal polygons should be plotted at the beginning of each simulation. Though the default is FALSE, if a problem with the polygons is detected by `tossm.diagnostics`, the polygons will be plotted regardless of the value of this argument.

`Seed` – An integer. If seed>0, it ensures a reproducible sequence of datasets.

***The output of* `run.tossm`**

The results of a TOSSM simulation are output as a list of class `tossm.obj`. These components of this list are as follows:

`abund.b` – a matrix tracking the abundance of the breeding populations through the simulation.

`catches` – a matrix tracking the catch that takes place in each management unit through the simulation.

`effort` – a vector tracking the average distance (in the x-dimension only) that hunters had to travel in order to catch the animals killed during the simulation (assuming that hunters are based at the left-hand edge of the landscape).

`mu.hist` – a list consisting of the management unit/historic polygons, and the catches taken from each management unit/historic polygon, in each year in which animals are harvested.

`est.abund.mu` – the abundance estimates for each managment unit throughout the simulation.

`var.abund.mu` – the variance of abundance estimates for each management unit throughout the simulation.

`gs` – a list containing all of the data from genetic samples taken during the simulation by year and by sampling area polygon. The length of the list is equal to the number of years in the simulation. List components are NULL for years in which no samples were taken. For years with samples, the list component is a sub-list of length equal to the number of sampling polygons. The sample for each year/sampling area combination is also attributed with IDs of the individuals sampled as well as spatial coordinate information. This information is stored in the attribute '`coords`' and can be accessed using the R command `unclass (e.g., unclass(attr(gs[[yr]][[samp.site]],'coords')).`

`agg.gs, agg.gtypes` – agg.gs is the same as `gs` but with genetic samples from each sampling polygon aggregated by year. Thus it is a list of length equal to the number of sampling polygons. Each element of the list is a three-dimensional array containing the genetic data for each sample collected from that sampling polygon during the entire simulation. The dimensions of the array are the number of samples (gs.years*n.samples) X the number of loci X two. The final dimension is used to store the two alleles at each locus. Haploid loci have NaN as their second allele. Agg.gtypes is similar to agg.gs, except that the data from each sampling polygon are stored in a two-dimensional matrix. Each row in the matrix represents a single sample. There is one column for each haploid locus followed by two columns for each diploid locus. Thus, the format of agg.gtypes is very similar to that required by many analytical methods, such as STRUCTURE. In both agg.gs and agg.gtypes, each list element (representing a single sampling polygon) is attributed with IDs of the individuals sampled, the birth year of each individual, and the spatial coordinates where each individual was sampled. This information is stored in the attribute '`coords`' and can be accessed using the R command `unclass (e.g., unclass(attr(agg.gtypes[[samp.site]], 'coords'))`

`agg.gfreq` – a single 3-D array with dimension (number of sampling polygons, number of loci, max number of alleles at a locus) containing allele frequencies for each locus within each sampling polygon.

`call` – the original call to run.tossm.

`seed` – the seed used for the random number generator.

The first three components in the above list should be sufficient for assessing the performance of a BSA with respect to meeting management objectives. `abund.b` can reveal whether the abundances of populations in the

simulation were maintained at or above some minimum sustainable level, for instance, 50% of carrying capacity. `effort` and `catches` indicate what impact the management units defined by the BSA had on resource utilization. The performance of a BSA should be considered 'good' when catch is maximized and effort minimized, while the populations are still maintained at a relative abundance consistent with the conservation goals of the management scheme.

`MU.hist` provides details about the types of mistakes a BSA commonly makes, which can help the BSA developer to improve the performance of their method. For instance, a BSA may consistently fail to detect small populations, or it might define the correct number of MUs most of the time, but place the boundaries in the wrong place.

The remaining outputs are included as diagnostics for ensuring the simulation ran properly, and for use in subsequent analysis of the data generated during the simulation.

Acknowledgements
Funding was provided by NMFS and the IWC Secretariat. Thanks to Barbara Taylor for reviewing this document, and to Charles Edwards for help his comments on the TOSSM package and its documentation.

References

Cooke, J.G. 1994. The management of whaling. Aquatic Mammalogy 20:129-135.
Excoffier, L., G. Laval, and S. Schneider. 2005. Arlequin ver. 3.0: An integrated software package for population genetics data analysis. Evolutionary Bioinformatics Online. **1**:47-50.
Glaubitz, J.C. 2004. CONVERT: A user-friendly program to reformat diploid genotypic data for commonly used population genetic software packages. Molecular Ecology Notes. **4**:309-310.
IWC. 1994. The Revised Management Procedure (RMP) for Baleen Whales. Rep. int. Whal. Commn., **44,** 145-167.
IWC. 2004. Report of the Workshop to design simulation-based performance tests for evaluation methods used to infer population structure from genetic data. Journal of Cetacean Research and Management. **6**(Suppl.):469-485.
Laval, G. & Excoffier L. 2004. SIMCOAL 2.0: a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history. Bioinformatics, **20**:2485-2487.
Martien, K. 2006. Progress on TOSSM dataset generation. Paper SC/58/SD2 submitted to the Annual Meeting of the Scientific Committee of the International Whaling Commission, St. Kitts, June 2006.
Strand A.E. 2002. METASIM 1.0: an individual-based environment for simulating population genetics of complex population dynamics. Molecular Ecology **2**:373-376.
Taylor, B.L., P.R. Wade, D.P. DeMaster, and J. Barlow. 2000. Incorporating uncertainty into management models for marine mammals. Conservation Biology 14:1243-1252.

Appendix A: Writing a BSA

The BSA needs in essence to do two things: 1) analyze the genetic and/or abundance data and 2) set management unit boundaries. The analysis is likely to be done exclusively by an 'outside' program not written in 'R'. However, it may be helpful to do some intermediate data processing in R to make the data supplied by the TOSSM package friendly to the analytical method being used. Similarly, it may take some processing to convert the output of an analytical method to a format accepted by `run.tossm`. Regardless, there needs to be at least some 'R' code written, even if just a 'wrapper' that allows the analytic method and the TOSSM package to interface.

There are a number of examples of mixing and migration models that have been proposed for use as analytic methods. It is foreseen that some of these methods can stand alone or be combined to supply the information desired to analyze TOSSM simulated data and set management boundaries. Whatever the method used, the 'R' code written for the BSA must begin as in the following example:

```
My.BSA<- function(genetic samples, abundance estimates, variances, catches by year
and sampling polygon, optional param1, optional param2){
```

The first four of these arguments are passed automatically to the BSA function by `run.tossm`, so the function must accept these four arguments whether it uses them or not. The last two arguments can optionally be used to accept any additional information that might be required to run the BSA function. The two optional arguments are specified in the argument `BSA.args`, and can be named as the user wishes.

`run.tossm` can also provide the BSA with genetic data aggregated across years, as well as information on allele frequencies across sampling polygons. These are not automatically available, but can be made available to the BSA function by including the following line in the BSA function:

`agg.gs.tseries()`—creates the objects `agg.gs, agg.gtypes, n.loci, n.areas,` and `n.alleles`

A BSA must return to run.tossm a list of management unit polygons. These polygons should be of class `gpc.poly`, implemented via the package `gpclib`. The management units output from the BSA must completely cover the extent of the breeding population polygons, yet not overlap each other. There is an important distinction between two types of potential BSAs:

1) Those BSAs that simply assign each `sample.poly` to a management unit. In this type of BSA, statistical analysis of genetic samples will be done at the level of the sampling polygon. For such BSAs, the helper function `MU.poly.generator` may be of use to the BSA developer. `MU.poly.generator` generates a bounding box around the breeding population polygons, and within this box a 60X60 grid of cells. Each cell is assigned the management unit of the `sample.poly` which it is closest to. Grid cells that share a management unit are then joined together. The resulting management units meet the requirements of a BSA, as they are non-overlapping and cover the entire simulated study area.

2) Those BSAs that work at the level of individual genetic samples. This type of BSA might disregard which sampling polygon samples come from. In this case, management unit boundaries could quite possibly bisect sampling polygons and/or, conversely, assign individuals from different sampling polygons to the same management unit. `MU.poly.generator` will not be of use to those developing such BSAs, and the developer is tasked with ensuring that a list of non-overlapping management unit polygons that cover the entire study area is returned from the BSA.

Appendix B

The following code can be cut and pasted directly into the R command window (after filling in the correct user name in the third line) in order to run the 'simple example' outlined in the user's manual. Note that this example assumes that R and the TOSSM package have been installed, and the datasets `Arch2_sc3_1.rda` through `Arch2_sc3_5.rda` are stored in the directory My Documents/TOSSM.example.

```
#load library and dataset
library(tossm)
setwd('C:/Documents and Settings/insert user name/My Documents/TOSSM.example')
load('Arch2_sc3_1.rda')

#define bp.polys and sample.polys
bp.polys<-list()
bp.polys[[1]]<-matrix(c(0,0,50,0,50,100,0,100),nrow=4,byrow=T)
bp.polys[[2]]<-matrix(c(50,0,100,0,100,100,50,100),nrow=4,byrow=T)
my.bp.polys<-lapply(bp.polys,as,"gpc.poly")
px <- list(c(5,20),c(30,45),c(55,70),c(80,95))
py <- list(c(10,40),c(60,90))
polys <- do.call('c',lapply(py, function(y){lapply(px,function(x){
      rbind(cbind(x,y[1]),cbind(x[2:1],y[2]))
      })}))
my.sample.polys <- lapply(polys, function(p) as(p,'gpc.poly'))

#establish simulation schedule
my.schedule <- def.make.schedule(n.pre.RMP=5,n.RMP=10,n.post.RMP=5,
   abund.gap=2)

#define BSA, BSA arguments, and harvest interval width
my.BSA <- fixed.MU.BSA
my.BSA.args <- list(n.mus=5)
my.harvest.interval.width <- 10

#run simulation
my.TOSSM.sim <- run.tossm(rland=rland.end, bp.polys=my.bp.polys,
      schedule=my.schedule, n.samples=25, sample.polys=my.sample.polys,
      BSA=my.BSA, BSA.args=my.BSA.args,
      harvest.interval=my.harvest.interval.width,plot.polys=T)

#plot results
trajectory.plotter(my.TOSSM.sim)

#run five replicate simulations
num.reps <- 5
datasets <- paste("Arch2_sc3_",1:num.reps,".rda",sep="")
TOSSM.sims <- lapply(1:num.reps, function(i){
      load(datasets[i])
      run.tossm(rland=rland.end, bp.polys=my.bp.polys, schedule=my.schedule,
      n.samples=25, sample.polys=my.sample.polys, BSA=my.BSA,
      BSA.args=my.BSA.args, harvest.interval=my.harvest.interval.width)
})

#calculate median abundance across replicates for each year
abund1.sum <- sapply(TOSSM.sims, function(i) i$abund.b[1,])
sapply(1:20,function(y) median(abund1.sum[y,]))
```

Appendix C: Components of an Rmetasim landscape object

The TOSSM package uses the package **rmetasim** (Strand 2002) for all population projections. The basic object created and manipulated by Rmetasim is a landscape. An **rmetasim** landscape is a required argument to run.tossm. The user does not need a detailed understanding of the components of a landscape object in order to use run.tossm. Nonetheless, a basic understanding of the components of a landscape will likely be useful.

A landscape object is a list of the following six components:

- `intparam`
- `switchparam`
- `floatparam`
- `demography`
- `loci`
- `individuals`

In the following sections, we give a brief overview of each of these components. At the end of each section, we provide one or two examples of how to access the different components described. The examples all assume that the landscape object being accessed is called 'rland.end', as that is the name given to the landscape in each of the TOSSM datasets.

intparam

`intparam` is a list containing all of the integer parameters of a landscape. It's components are:

- `habitats` – The number of habitats (i.e., breeding populations) present in the landscape
- `stages` – The number of life history stages
- `locusnum` – The number of loci
- `numepochs` – The number of epochs. The TOSSM datasets all consist of a single epoch
- `currentgen` – The number of years (not generations!) the landscape has been projected through. For all TOSSM datasets, `currentgen` equals 1000, as they were simulated for 1000 years.
- `currentepoch` – The current epoch. Epochs are numbered starting at 0. All TOSSM datasets are in epoch 0.
- `totalgens` – The total number of years (not generations!) that the landscape can be projected. Note that **rmetasim** will not project a landscape beyond this number, so before any of the TOSSM datasets can be used in further simulations, this parameter must be changed to a larger number. However, totalgens is automatically increased an appropriate amount by `run.tossm` for `tossm` simulations
- `numdemos` – The number of different demographies available for this landscape. For the TOSSM datasets, this parameter is always equal to 1.
- `maxlandsize` – The maximum total abundance (summed across all habitats) possible for this landscape.
- `nextid` – The unique identifier that will be assigned to the next individual born in the landscape.

Access example:

```
> rland.end$intparam$habitats
[1] 2
```

switchparam

`switchparam` is a list of parameters that are used to switch features of **rmetasim** on and off. They can only take the values of 0 and 1. `randepoch` and `randdemo` are only relevant to landscapes that include more than one epoch or demography, respectively, and thus are not relevant to the TOSSM datasets. `multp` applies only in cases where females give birth to more than one offspring at a time, and is therefore also not relevant to the TOSSM datasets. `densdepdemo` indicates whether or not density dependence is used, and is set to 1 for all TOSSM datasets.

Access example:

```
> rland.end$switchparam$densdepdemo
[1] 1
```

floatparam

`floatparam` contains a single component, selfing, to indicate the rate of self-fertilization. It is set to 0 for all TOSSM datasets.

Access example:

```
> rland.end$floatparam$selfing
```

```
[1] 0
```

demography

The demography component of an **rmetasim** landscape holds all of the demographic information necessary for the projection. It consists of three components – `localdem`, `localdemK`, and `epochs`. `localdem` and `localdemK` are each lists containing the life history matrices for the populations near zero population density and at carrying capacity, respectively. The TOSSM datasets only consist of a single demography, so `localdem` and `localdemK` are each lists of length 1.

epochs is a list of length equal to the number of epochs that have been defined for the landscape. For all of the TOSSM datasets, there is a single epoch, so the length of the epochs list is 1.

Each epoch consists of the following eight named components:

- `RndChooseProb` – This variable is not relevant to the TOSSM datasets, as it only applies to landscapes containing more than one demography
- `StartGen` – The simulation year in which the epoch in question commences. This variable is set to 0 for all TOSSM datasets, since the datasets only contain a single epoch.
- `Extinct` – A vector of length equal to the number of habitats indicating the probability that each habitat goes extinct in a given simulation year. The extinction probability of all habitats in all TOSSM datasets is zero.
- `Carry` – A vector of length equal to the number of habitats indicating the carrying capacity of each habitat.
- `Localprob` – A vector of length equal to the number of demographies indicating the probability with which each demography is chosen each year. The TOSSM datasets all contain a single demography, so this variable is set to 1
- `S`, `R`, and `M` – matrices governing the movement of individuals and gametes between habitats. These are referred to as the 'landscape matrices' in the **rmetasim** documentation.

Access example:

```
> rland.end$demography$epochs[[1]]$Carry
[1] 3750 3750
```

loci

`loci` is, not surprisingly, a list of all of the genetic loci simulated as part of a landscape. For each locus, the following information is available:

- `type` – An integer indicating the mutation model a locus follows. Options are infinite allele mutation model (type=251), stepwise mutation model (type=252) or DNA sequence (type=253).
- `ploidy` – An integer indicating whether the locus is haploid (ploidy=1) or diploid (ploidy=2).
- `trans` – The mode of transmission for the locus. 0=biparental inheritance, 1=uniparental inheritance.
- `rate` – The mutation rate of the locus
- `alleles` – A list of all alleles at the locus. Each element of this list (i.e., each allele) is itself a list with the following components:
  - `aindex` – The index assigned to the allele
  - `birth` – The simulation year in which the allele originated
  - `prop` – The frequency of the allele in the landscape (all habitats combined)
  - `state` – The state of the allele. For microsatellite markers, the state is equivalent to the allele length. For DNA sequence markers, the state is the nucleotide sequence.

Access example:

```
> rland.end$loci[[1]]$ploidy
[1] 1

> rland.end$loci[[1]]$alleles[[1]]$prop
[1] 0.0001296008
```

individuals

The `individuals` component is a matrix containing all of the demographic and genetic information for each individual in the landscape. It is the component most likely to be accessed by a user of the TOSSM package. The `individuals` matrix has one row for each individual. The first six columns of the matrix contain demographic information, while the remaining columns contain the genetic information. Contents of the columns are as follows:

- Column 1 – This column indicates both the habitat that an individual belongs and the life history stage the individual is in. Assuming that an individual is from habitat x and in life history stage y, the value in this column will be (x * number of habitats) + y. Both habitats and stages are numbered starting at 0, so an individual with a 0 in column 1 is from habitat 0 and in life history stage 0.
- Column 2 – This column is not currently used by **`rmetasim`** and is set to 0 for all individuals.
- Column 3 – The simulation year in which the individual was born.

- Column 4 – A unique identifier for the individual
- Columns 5&6 – The unique identifiers for the individual's mother and father, respectively.
- Columns 7 and higher – The genetic data associated with the individual. There is one column for each haploid locus and two for each diploid locus. In the TOSSM datasets, column 7 holds the mtDNA haplotype, columns 8 and 9 hold the two alleles for locus 1, columns 10 and 11 hold the alleles for locus 2, etc. The alleles/haplotypes are represented in the individuals matrix by their allele index. The information stored in the `loci` component of the landscape can be used to associate allele indices with mtDNA haplotype sequences and microsatellite repeat lengths

Access example:

```
> rland.end$individuals
      [,1] [,2] [,3]   [,4]   [,5]   [,6] [,7] [,8] [,9] [,10]...
 [1,]    0    0  978 473189 462079 461701   67    3    6     8...
 [2,]    0    0  979 473573 454805 460043   43    3    3     2...
 [3,]    0    0  979 473587 463469 461103   37    4    4     1...
 [4,]    0    0  981 474371 469827 463031   43    4   13    11...
 [5,]    0    0  982 474836 469144 462530   36    3    4     1...
 [6,]    0    0  982 474870 466020 458638   43    3    3     1...
 [7,]    0    0  982 474945 465830 455850   43    1    3     5...
 [8,]    0    0  983 475360 455667 464958   69    4    4     8...
 [9,]    0    0  983 475434 462993 463021   42    3    3     1...
[10,]    0    0  983 475480 472180 460168   25    2    5    12...
  .
  .
  .
```