An Introduction to MBHdesign

Scott D. Foster
Data61, CSIRO, Hobart, Tasmania, Australia

Abstract

A robust scientific conclusion is the result of a rigorous scientific process. In observational ecology, this process involves making inferences about a population from a sample. The sample is crucial, and is the result of implementing a *survey design*. A good survey design ensures that the data from the survey is capable of answering the research question. Better designs, such as spatially balanced designs, will also be as precise as possible given the constraints of the budget.

The MBHdesign package is useful for creating spatially balanced designs. There are four tasks that it is intended to address: 1) designing spatially-balanced surveys using Balanced Adaptive Sampling (BAS Robertson *et al.* 2013, 2017); 2) designing and analysing spatially-balanced surveys that incorporate existing legacy sites, using Foster *et al.* (2017); 3) designing spatially-balanced transect-based surveys using the methods described in Foster *et al.* (2020), and; 4) designing spatially-balanced cluster designs Foster *et al.* (in press). The first example in this tutorial generates a point-based design and shows how legacy sites can be incorporated. There are three steps to this process:

1. Altering inclusion probabilities for spatial balance, taking into account the location of legacy sites. This is done using the function `alterInclProbs`;
2. Generating spatially balanced designs for a given set of inclusion probabilities, through the function `quasiSamp`; and
3. Analysing some (made up) data using model-based methods (using `modEsti`).

The second example in this tutorial generates a transect-based design over the same inclusion probabilities. This consists of just one substantive step: calling the `transectSamp` function.

The third example in this tutorial generates a clustered sample, where groups of sites are chosen to be spatially similar but these clusteres are spatially segregated. This process relies on use of the `quasSamp.cluster` function.

Keywords: Spatially-Balanced Survey Design, Balanced Adaptive Sampling, Transect, Cluster, R.

First Things First

Before starting with this introduction to `MBHdesign`, we need to make sure that everything is set up properly. Much of this will vary from computer to computer, but you must have a working version of R installed (preferably the recent release). This vignette was created using R Under development (unstable) (2023-09-03 r85052). It does not matter whether you prefer to use R through a development environment (such as RStudio) or through the command line – the results will be the same. There are two ways to install `MBHdesign`. The first is through

the CRAN repository. This version will not be updated very frequently, so it will not have the very latest developments (nor the associated bugs). This version can be installed by starting R and then submitting (at command line):

```
install.packages( "MBHdesign")
## or ##
devtools::install_github( repo="Scott-Foster/MBHdesign", build_vignettes=FALSE)
```

If you install from CRAN, you will be asked which repository you want to use. Just use one that is geographically close to where you are (or where your computer is). Installing from the github repos will give you the most up to date version. However, do take note of the build_vignettes=FALSE argument: the vignette can take a while to build (and has already been pre-built).

Whichever way the package is installed, you'll have to load it:

```
library( MBHdesign)
```

For illustration is is also good to fix the random number seed, so that this document is reproducible *exactly*. However, and due to R evoloving, the results may differ between different versions of R. In particular, R/3.6 could be quite different from R/3.5.

```
set.seed( 747)  #a 747 is a big plane
```

Technical note about version on CRAN. CRAN has some strict rules for packages that it hosts. One of these is that the package must not take long to build. This vignette breaks this rule... As a work-around the version of this vignette in the CRAN package uses precompiled data (with the usePrecompiledData variable set to TRUE). However, on GitHub the package does not use precompiled data. You'll see locations in the code where the running of certain functions is conditional on this variable. Please don't be confused by it.

```
#TRUE if you want some of the examples shortened by using pre-saved output
usePrecompiledData <- FALSE
```

Now, we are good to go with the rest of the introduction.

A Point-Based Design

Let's pretend that we want to generate $n = 10$ samples on a grid of points (representing the centres of a tessellation). The grid of points consists of $N = 100 \times 100 = 10000$ points in 2-dimensional space (spanning the interval $[0, 1]$ in both dimensions). Let's also pretend that there are 3 legacy sites, that have been sampled in previous survey efforts, and we wish to revisit them in the current survey. The legacy sites are located at random throughout the study area. Here, I have generated it all in R (painstakingly), but in a real application, most of this information could be read in from file.

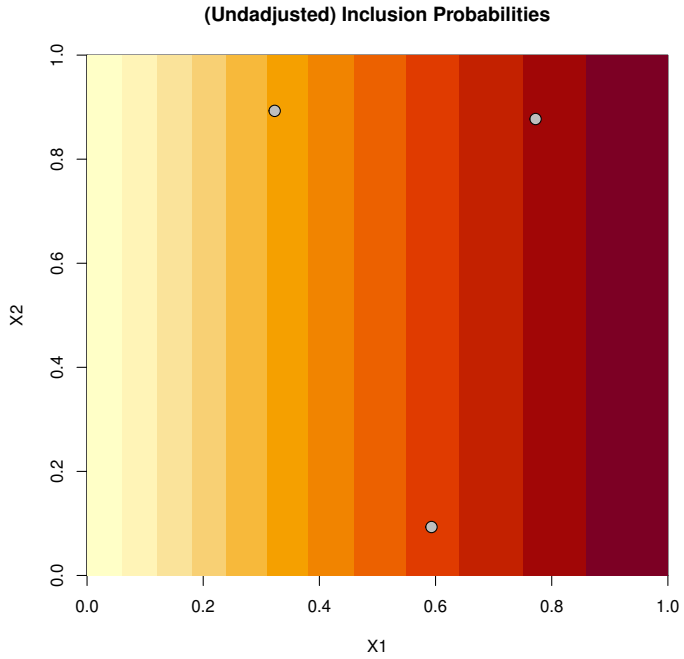
```
#number of samples
n <- 10
#number of points to sample from
N <- 100^2
#the sampling grid (offset so that the edge locations have same area)
offsetX <- 1/(2*sqrt( N))
my.seq <- seq( from=offsetX, to=1-offsetX, length=sqrt(N))
X <- expand.grid( my.seq, my.seq)
#the legacy sites (three of them)
legacySites <- matrix( runif( 6), ncol=2, byrow=TRUE)
#names can be useful
colnames( X) <- colnames( legacySites) <- c("X1","X2")
```

Inclusion Probabilities

Key to this whole design process is the concept of inclusion probabilities. Inclusion probabilities define the chance that any particular site will be part of the sample. So, if a site's inclusion probability is small, then the site is unlikely to be included into the sample. Specifying inclusion probabilities can improve efficiency of the sampling design. That is, standard errors can be reduced for a given number of samples. The 'trick' is to specify inclusion probabilities so that the sites that should have highly variable observations are sampled more often (e.g. Grafström and Tillé 2013). In ecology, variance often increases with abundance (due to Taylor's Power Law; Taylor 1961), so inclusion probabilities could be increased with abundance. If there is no knowledge about the area being sampled, then all sites should be given equal inclusion probabilities. Frequently, a formal requirement is to constrain the inclusion probabilities so that they sum to n. While `MBHdesign` does not require this constraint, it can be useful for communication purposed.

Here, we are going to pretend that there is some gradient in the variance of the population under study. We stress that this is illustrative only.

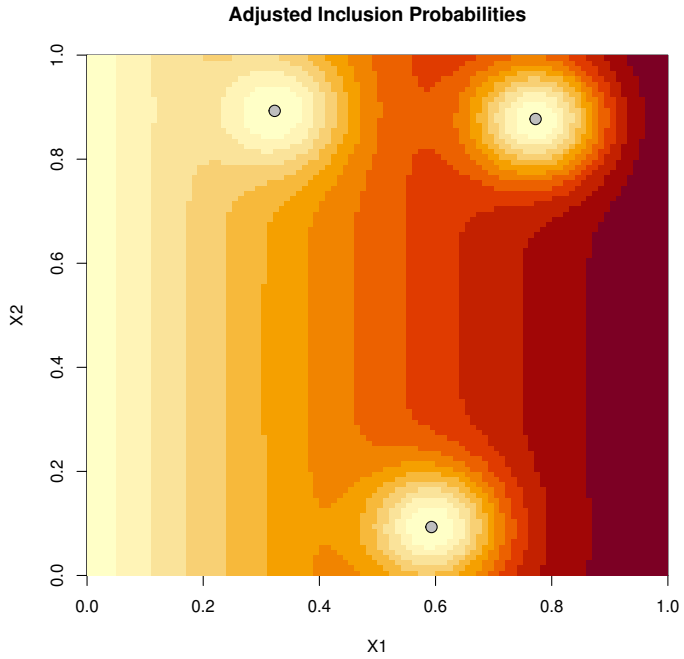
```
#non-uniform inclusion probabilities
inclProbs <- 1-exp(-X[,1])
#scaling to enforce summation to n
inclProbs <- n * inclProbs / sum( inclProbs)
#uniform inclusion probabilities would be inclProbs <- rep( n/N, times=N)
#visualise
image( x=unique( X[,1]), y=unique( X[,2]),
    z=matrix( inclProbs, nrow=sqrt(nrow(X)), ncol=sqrt(nrow( X))),
    main="(Undadjusted) Inclusion Probabilities",
    ylab=colnames( X)[2], xlab=colnames( X)[1])
#The legacy locations
points( legacySites, pch=21, bg=grey(0.75), cex=1.5)
```



Accommodating Legacy Sites

To generate a design that is spatially balanced *in both the n new sample sites and the legacy sites*, we adjust the inclusion probabilities. The adjustment (see Foster *et al.* 2017) reduces the inclusion probabilities so that sites near legacy sites are less likely to be chosen in the new sample.

```
#alter inclusion probabilities
#   so that new samples should be well-spaced from legacy
altInclProbs <- alterInclProbs( legacy.sites=legacySites,
                potential.sites=X, inclusion.probs = inclProbs)
#visualise
image( x=unique( X[,1]), y=unique( X[,2]),
    z=matrix( altInclProbs, nrow=sqrt(nrow(X)), ncol=sqrt(nrow( X))),
    main="Adjusted Inclusion Probabilities",
    ylab=colnames( X)[2], xlab=colnames( X)[1])
#The legacy locations
points( legacySites, pch=21, bg=grey(0.75), cex=1.5)
```



So, the inclusion probabilities have been reduced around the legacy sites. It is perhaps worth noting that the reduction in inclusion probabilities, due to the legacy sites, can be viewed as *sequential*. This means that the reduction for any legacy site is in addition to the reduction of all of the other legacy sites – there is no extra joint effect. Also, the adjustment is proportional to the original inclusion probability, so that a small inclusion probability and a large inclusion probability are both adjusted proportionally to the same amount.

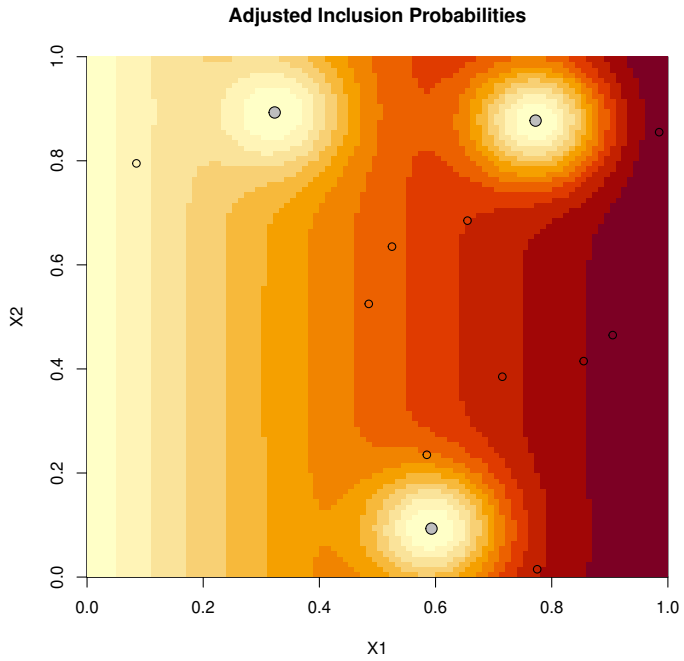
There are some other arguments to the `altInclProbs()` function (omitted for clarity here). These can be seen to refine the call and/or to make the computer to do its work quicker. Type `?altInclProbs` for more details.

Generating the Design

Irrespective of how the inclusion probabilities were obtained, we can now use them to generate a spatially balanced design.

```
#generate the design according to the altered inclusion probabilities.
samp <- quasiSamp( n=n, dimension=2,
        study.area=matrix( c(0,0, 0,1, 1,1, 1,0),ncol=2,  byrow=TRUE),
        potential.sites=X, inclusion.probs=altInclProbs)
#for faster sampling (large problems),
# consider using quasiSamp.raster, which utilises SpatRaster formats
#visualise
image( x=unique( X[,1]), y=unique( X[,2]),
    z=matrix( altInclProbs, nrow=sqrt(nrow(X)), ncol=sqrt(nrow( X))),
    main="Adjusted Inclusion Probabilities",
    ylab=colnames( X)[2], xlab=colnames( X)[1])
#The legacy locations
```

```
points( legacySites, pch=21, bg=grey(0.75), cex=1.5)
points( samp[,1:2], pch=21)
```



Voilà! A spatially balanced design that incorporates legacy sites. It is contained in the object `samp`, which looks like:

```
print( samp, row.names=FALSE)

    X1    X2 inclusion.probabilities   ID
 0.855 0.415            0.0017794542 4186
 0.485 0.525            0.0011898872 5249
 0.985 0.855            0.0018903022 8599
 0.525 0.635            0.0012643162 6353
 0.775 0.015            0.0015975614  178
 0.655 0.685            0.0014621556 6866
 0.905 0.465            0.0018436754 4691
 0.085 0.795            0.0002510468 7909
 0.585 0.235            0.0010991360 2359
 0.715 0.385            0.0015810540 3872
```

The columns of `samp` are:

- The sample locations in the `X1` and `X2` dimensions;
- The inclusion probability for that sampling location; and
- The row number (ID), of the original list of potential sites (X).

Analysis

After finalising the design, time comes to go and undertake the survey. For illustration, we do this *in silico* and generate observations according to a pre-defined function (following Foster *et al.* 2017, amongst others).

```
#generate some `observations' for the new sites
Z <- 3*( X[samp$ID,1]+X[samp$ID,2]) +
                        sin( 6*( X[samp$ID,1]+X[samp$ID,2]))
#and some for the legacy sites
Zlegacy <- 3*( legacySites[,1]+legacySites[,2]) +
                        sin( 6*( legacySites[,1]+legacySites[,2]))
```

These data can be analysed in two ways: 1) design-based, which uses minimal assumptions about the data; and 2) model-based, which attempts to describe more aspects of the data. See Foster *et al.* (2017) for a more complete description. For design-based analysis we take a weighted average of the estimator for the legacy sites and the estimator for the new sites. In both cases the estimates follow Horvitz and Thompson (1952). Please do read the section in Foster *et al.* (2017) for comments on estimation, it could save you some grief.

```
#the proportion of legacy sites in the whole sample
fracLegacy <- nrow( legacySites) / (n+nrow( legacySites))
#inclusion probabilities for legacy sites
#   (these are just made up, from uniform)
LegInclProbs <- rep( nrow( legacySites) / N, nrow( legacySites))
#estimator based on legacy sites only
legacyHT <- (1/N) * sum( Zlegacy / LegInclProbs)
#estimator based on new sites only
newHT <- (1/N) * sum( Z / inclProbs[samp$ID])
mean.estimator <- fracLegacy * legacyHT + (1-fracLegacy) * newHT
#print the mean
print( mean.estimator)

[1] 3.242095
```

This is pretty close to the true value of 2.9994. To get a standard error for this estimate, we use the `cont_analysis()` function from the `spsurvey`* (Dumelle *et al.* 2021), which implements the neighbourhood estimator of Stevens and Olsen (2003).

```
#load the spsurvey package
library( spsurvey)
#rescale the inclusion probs
#   (the sample frames are the same in legacy and new sites)
tmpInclProbs <- ( c( inclProbs[samp$ID], LegInclProbs) / n) *
                                                (n+nrow(legacySites))
```

*In versions of spsurvey, prior to version 5, this was acievied through `total.est()`

```
#create a temporary data frame
tmpDat <- data.frame( siteID=
                      c( samp$ID, paste0( "legacy", 1:nrow(legacySites))),
                      wgt=1/tmpInclProbs,
                      xcoord=c(samp$X1,legacySites[,1]),
                      ycoord=c(samp$X2,legacySites[,2]), Z=c(Z,Zlegacy))
#calculate the standard error
se.estimator <- cont_analysis( tmpDat, vars="Z",
                      weight="wgt",
                      xcoord="xcoord",
                      ycoord="ycoord")$Mean$StdError
#print it
print( se.estimator)

[1] 0.4186033
```

For model-based mean and standard errors we follow the 'GAMdist' approach in Foster *et al.* (2017).

```
tmp <- modEsti( y=c( Z, Zlegacy), locations=rbind( X[samp$ID,], legacySites),
        includeLegacyLocation=TRUE, legacyIDs=n + 1:nrow( legacySites),
        predPts=X, control=list(B=1000))
print( tmp)

$mean
[1] 2.259459

$se
[1] 0.3548523

$CI
    2.5%    97.5%
1.616817 2.942019
```

In this case, the standard error estimates are quite different. On average, they tend to be (when there are only a few legacy sites). Even so, this level of difference is unusual.

Last Things Last

The only remaining thing to do is to tidy up our workspace. First, to export our sample locations. Second, to remove all objects for this analysis from your workspace.

```
##write csv if wanted
#write.csv( samp, file="pointSample1.csv", row.names=FALSE)
#tidy
rm( list=c( "samp", "tmp", "se.estimator","tmpDat","tmpInclProbs",
```

```
            "mean.estimator","newHT","legacyHT","LegInclProbs",
            "fracLegacy","Zlegacy","Z","X","altInclProbs", "n", "N",
            "my.seq","inclProbs", "offsetX", "legacySites"))
```

Transect-Based Surveys

Let's now pretend that we want to generate $n = 10$ **transect** locations over a grid of points (representing the centres of a tessellation). We will use the methods described in Foster *et al.* (2020). The transects are linear and are each of length 0.15, and the grid of points consists of $N = 100 \times 100 = 10000$ points in 2-dimensional space (spanning the interval $[0, 1]$ in both dimensions). For this example, we generate all information about the survey area and inclusion probabilities but in a real application, most of this information is likely to come from file (e.g. a shapefile).

A small word of warning: this example will take a little while to run. Sorry. Whilst choices have been made to reduce the computation, at the expense of accuracy, the computation is still quite intensive.

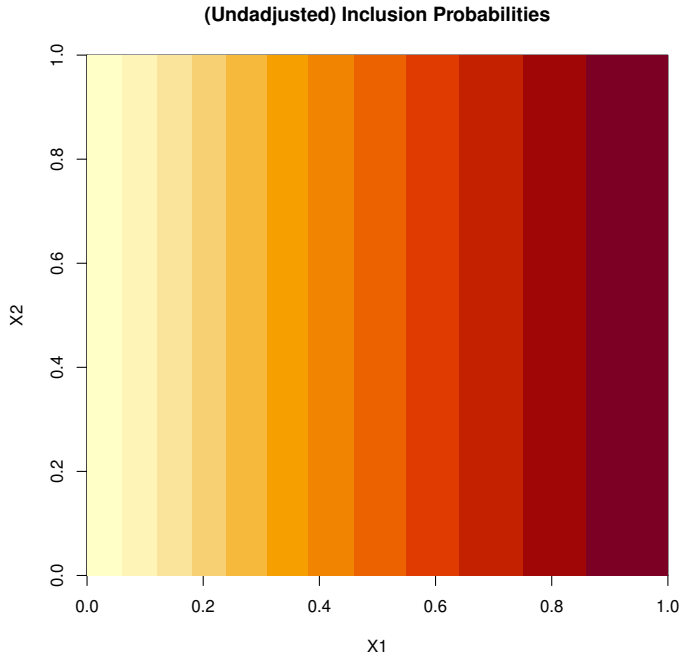
Another note of warning: while the random number seed is set, different versions of R may produce different results. This is due to R evoloving. In particular, R/3.6 could be quite different from R/3.5.

```
set.seed( 747)  #I'm currently on a 787, so it *almost* seems appropriate
#number of transects
n <- 10
#number of points to sample from
N <- 100^2
#the sampling grid (offset so that the edge locations have same area)
offsetX <- 1/(2*sqrt( N))
my.seq <- seq( from=offsetX, to=1-offsetX, length=sqrt(N))
X <- expand.grid( my.seq, my.seq)
colnames( X) <- c("X1","X2")
```

Inclusion Probabilities

Here, like in the previous example, we are going to pretend that there is some gradient in the variance of the population under study. We stress that this is illustrative only. The transect probabilities are set-up to reflect this.

```
#non-uniform inclusion probabilities
inclProbs <- 1-exp(-X[,1])
#scaling to enforce summation to n
inclProbs <- n * inclProbs / sum( inclProbs)
#uniform inclusion probabilities would be inclProbs <- rep( n/N, times=N)
#visualise
image( x=unique( X[,1]), y=unique( X[,2]),
    z=matrix( inclProbs, nrow=sqrt(nrow(X)), ncol=sqrt(nrow( X))),
    main="(Undadjusted) Inclusion Probabilities",
    ylab=colnames( X)[2], xlab=colnames( X)[1])
```



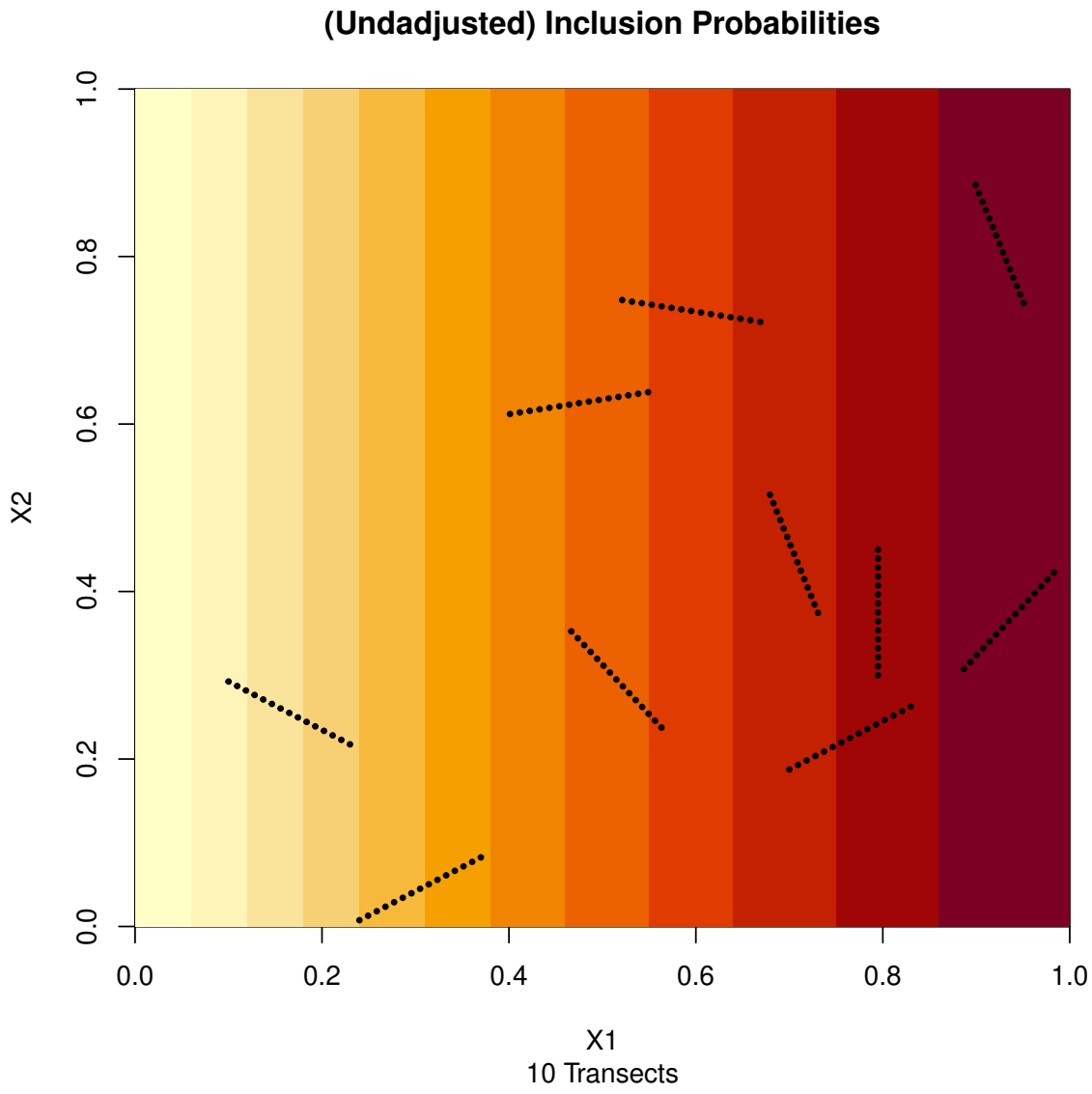
We will need to tell the transect sampling function some information about the size and shape of the transects. For this purpose, the transects should be thought of as a set of points (arranged on a line for a linear transect). For the particular case of linear transects the sampling function `transectSamp` arranges this for us, but we still need to give some information. The choices made here are compromised in favour of speed-of-execution rather than accuracy. In a real (and final) design, you may want to consider finer resolutions. These are the only control options needed in this example, but we note that there are others, which mostly control different aspects of the algorithm (not the transect setup itself).

```
#my.control is a list that contains
my.control <- list(
  #the type of transect
  transect.pattern="line",
  #the length of transect
  line.length=0.15,
  #the number of points that define the transect
  transect.nPts=15,
  #the number of putative directions that a transect can take
  nRotate=9
)
```

Take the Transect Sample

The sample is obtained by a relatively straight-forward call. They are also plotted over the inclusion probabilities.

```
#take the transect sample
if( !usePrecompiledData){
  samp <- transectSamp( n=n, potential.sites=X,
                inclusion.probs=inclProbs, control=my.control)
} else{
  samp <- readRDS( system.file(
                    "extdata", "transectSamp1.RDS",
                    package="MBHdesign"))}
image( x=unique( X[,1]), y=unique( X[,2]),
    z=matrix( inclProbs, nrow=sqrt(nrow(X)), ncol=sqrt(nrow( X))),
    main="(Undadjusted) Inclusion Probabilities",
    sub="10 Transects",
    ylab=colnames( X)[2], xlab=colnames( X)[1])
points( samp$points[,5:6], pch=20, cex=0.6)
```



Last Things Last

The only remaining thing to do is to tidy up our workspace. First, to export our sample locations. Second, to remove all objects for this analysis from your workspace.

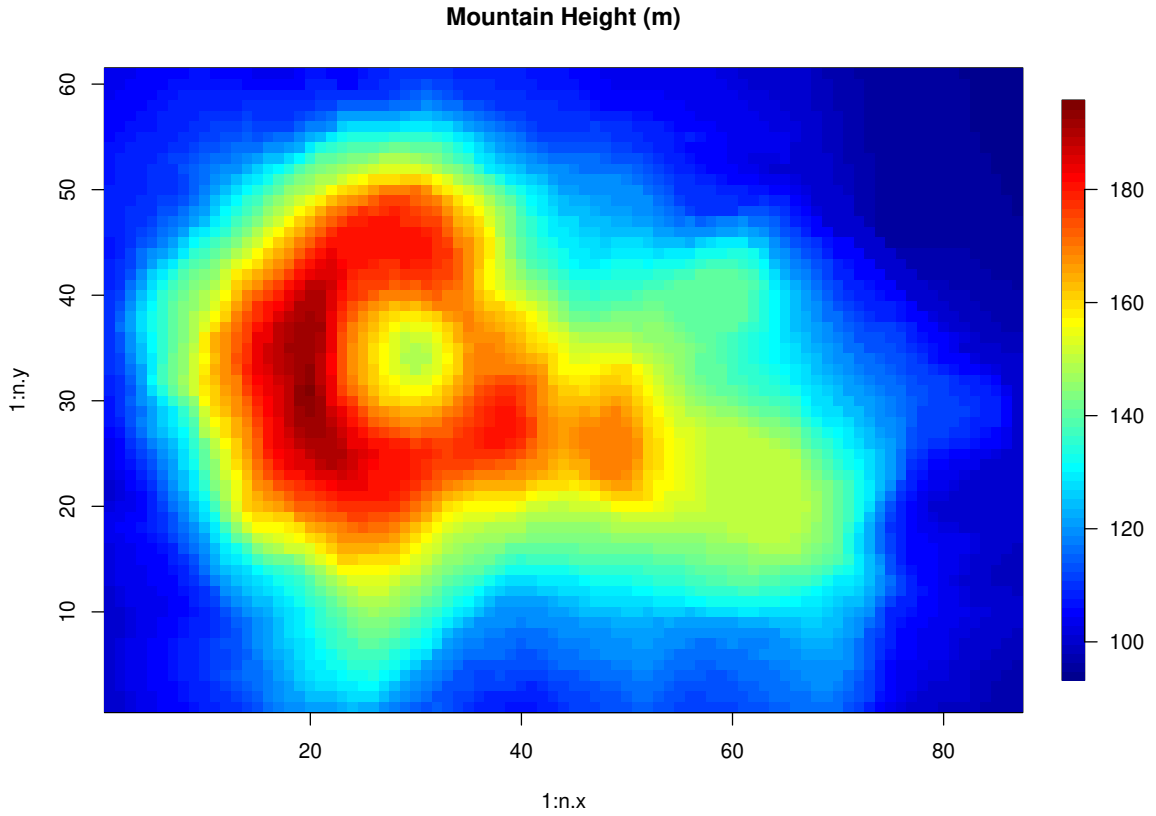
```
##write csv
#write.csv( samp$transect, file="transectSample1.csv", row.names=FALSE)
#tidy
rm( list=c( "X", "inclProbs", "samp", "my.control",
            "my.seq", "offsetX", "n", "N"))
```

A Harder/Constrained Transect-Based Survey

The design problem described in the previous section is pretty straight-forward as the sample space doesn't have any constraints upon it. Here, we make things more complicated in this respect, by forcing transects to go down-hill. This mimics towed-platforms, such as underwater image tools, and was enforced in the seamount example in Foster *et al.* (2020). As an illustration, we utilise the volcano altitude dataset from the excellent `MASS` package (Venables and Ripley 2002). The inclusion probabilities here are assumed to be uniform, so that the transects will be spatially-balanced, and even, throughout the study area.

Forcing down-hill transects isn't the only type of constraint. However, it is illustrative. Another type of constraint that we regularly encounter is, when sampling some certain types of seamounts, the desire to run the transects from 'peak-to-base' so that all transects must start at the highest point of the seamount. This kind of constraint can be incorporated within `MBHdesign` using the function `findTransFromPoint`, which only allows transects from a specified set of points.

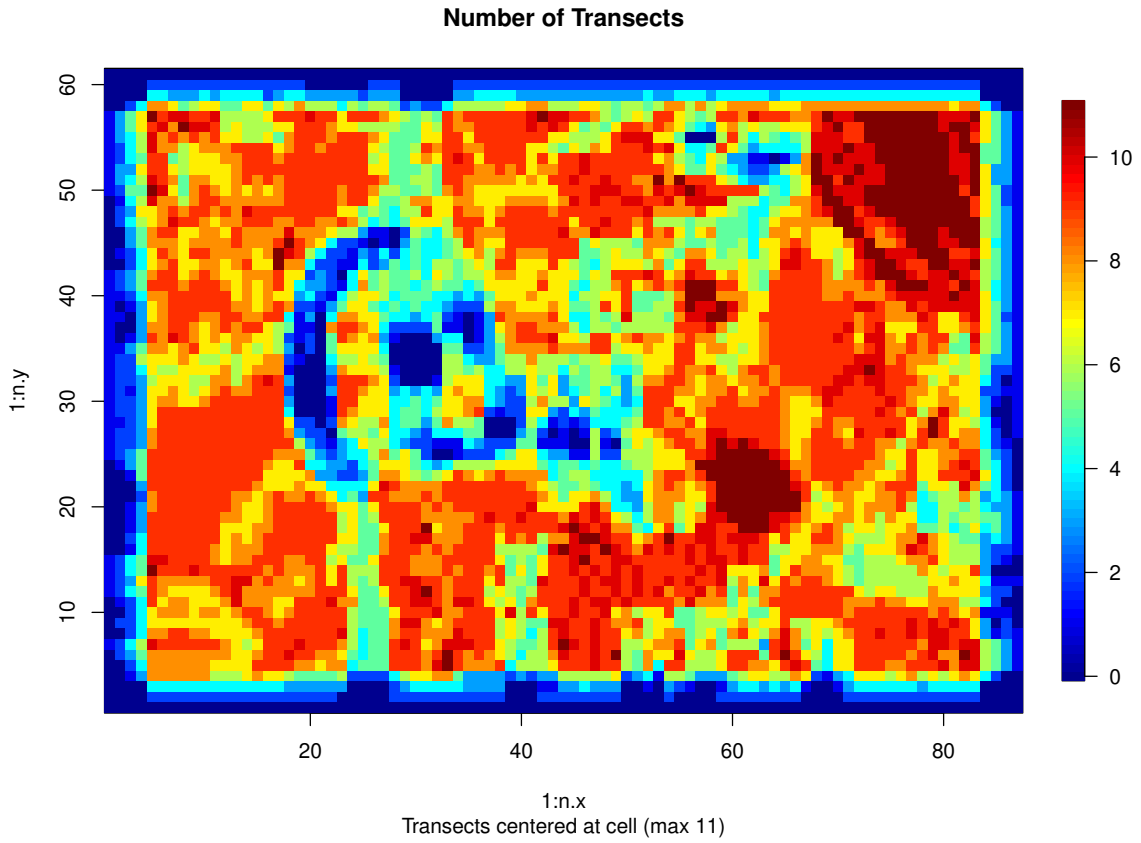
```
library( MASS)  #for the data
library( fields)  #for image.plot
#library( MBHdesign) #for the spatial design and constraints
set.seed( 717)  #Last plan I was on
#number of transects
n <- 20
#load the altitude data
data( volcano)  #this is a matrix
n.x <- nrow( volcano)
n.y <- ncol( volcano)
image.plot( x=1:n.x, y=1:n.y, z=volcano, main="Mountain Height (m)", asp=1)
#format for MBHdesign functions
pot.sites <- expand.grid( x=1:n.x, y=1:n.y)
pot.sites$height <- as.vector( volcano)
#details of the transects (see Details section in ?transectSamp)
vol.control <- list( transect.pattern="line", transect.nPts=10,
                     line.length=7, nRotate=11, mc.cores=1)
#In a real application, transect.nPts and nRotate may need to be increased
#1 cores have been used to ensure generality for all computers.
# Use more to speed things up
```



So, that is the data. Note that there are locations where transects could descent and then ascend. The 'hole' (crater) in the middle of the mountain is the largest and most obvious example. To avoid transects that ascend and descend, we use an `MBHdesign` function

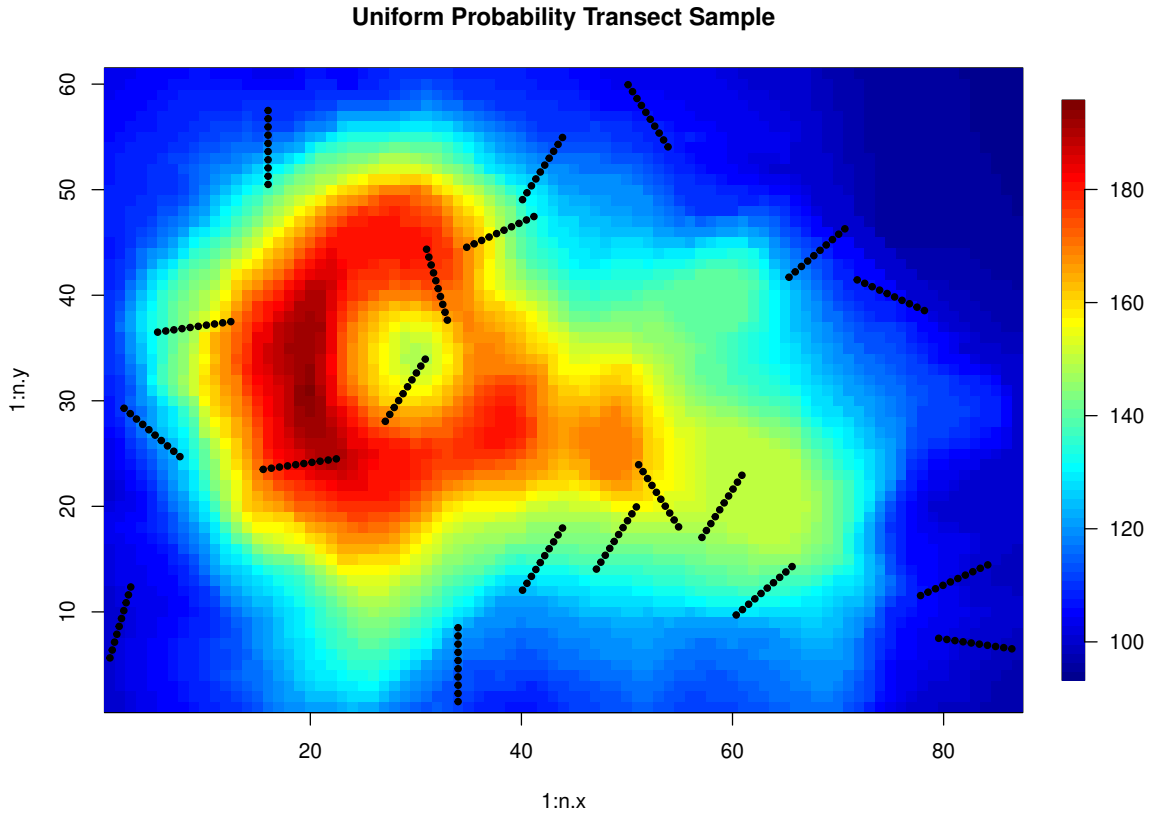
```
if( !usePrecompiledData){
  vol.constraints <- findDescendingTrans(
        potential.sites = pot.sites[,c("x","y")],
        bathy=pot.sites$height, in.area=rep( TRUE, nrow( pot.sites)),
        control=vol.control)
} else{
  vol.constraints <- readRDS( system.file(
                    "extdata", "transectConstraints1.RDS",
                    package="MBHdesign"))}
#this is a matrix with nrow given by the number of sites and ncol by
#   the number of rotations around each site
print( dim( vol.constraints))

[1] 5307   11

#The contents describe how the transect lays over the landscape
#So, there are 15592 putative transects that ascend and descend
#   (and can't be used in the sample)
```

```
table( as.vector( vol.constraints))


         descend descendAndNA_NaN        upAndDown
           34397              8388            15592

#convert to TRUE/FALSE
#Note that the final possible transect type ('descendAndNA') is
#   not present in these data
#If present, we would have to decide to sample these or not
vol.constraints.bool <- matrix( FALSE, nrow=nrow( vol.constraints),
                                ncol=ncol( vol.constraints))
vol.constraints.bool[vol.constraints %in% c("descend")] <- TRUE
#Let's get a visual to see what has just been done.
tmpMat <- matrix( apply( vol.constraints.bool, 1, sum), nrow=n.x, ncol=n.y)
image.plot( x=1:n.x, y=1:n.y, z=tmpMat,
            main="Number of Transects",
            sub="Transects centered at cell (max 11)", asp=1)
#There aren't any transects that are centred on ridges or depressions.
```



The task is to now sample the reduced sampling frame. Fortunately, with the constraint matrix just produced, this is pretty easy.

```
#take the sample
if( !usePrecompiledData){
  volSamp <- transectSamp( n=n, potential.sites=pot.sites[,c("x","y")],
                           control=vol.control,
                           constrainedSet=vol.constraints.bool)
} else{
  volSamp <- readRDS( system.file(
                     "extdata", "transectSamp2.RDS",
                     package="MBHdesign"))}
#visualise the sample
image.plot( x=1:n.x, y=1:n.y, z=volcano,
            main="Uniform Probability Transect Sample", asp=1)
points( volSamp$points[,c("x","y")], pch=20)
```



The only remaining thing to do is to tidy up our workspace. First, to export our sample locations. Second, to remove all objects for this analysis from your workspace.

```
##write csv
#write.csv( volSamp$transect, file="volcanoSample1.csv", row.names=FALSE)
#tidy
rm( list=c( "volSamp", "tmpMat", "vol.constraints.bool",
   "vol.constraints", "vol.control", "pot.sites",
```

```
  "n", "n.x", "n.y", "volcano"))
```

Clustered Surveys

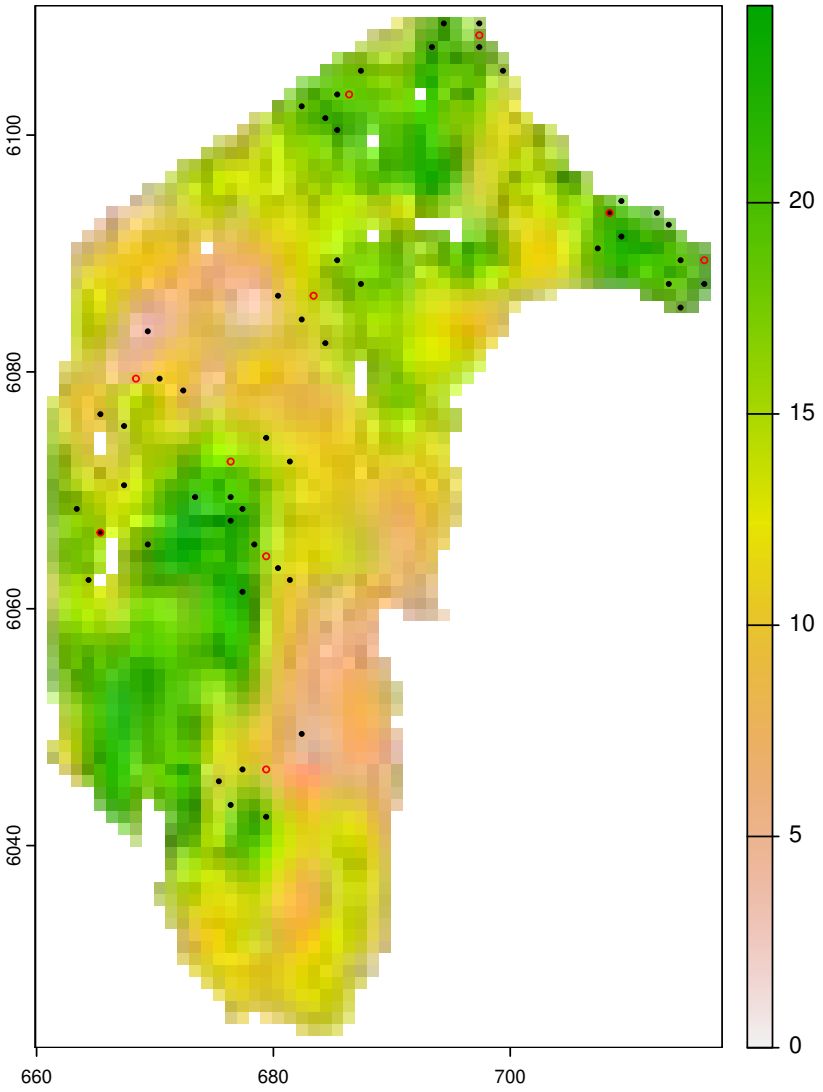
The very nature of spatially-balanced designs implies that the sampling locations will be well spread out in space. Whilst this is statistically appealing (as it is usually efficient) it can also create logistic difficulties – for example transit time between sampling locations is large. To overcome this logistical constraint, the function `quasiSamp.cluster` has the ability to group sampling locations into clusters. Spatial balance is maintained at both the between-cluster and within-cluster levels (see Foster *et al.* in press, for details). A clustered sample, consisting of 10 clusters or 5 samples, will now be generated using an exaggerated version of data representing Soil Moisture at the Root Zone (SMRZ) over a 1km raster of the Australian Capital Territory (see Frost *et al.* 2018).

```
#need raster functions
library( terra)
#import example data
egDat <- rast(system.file(
                    "extdata", "ACT_DemoData.grd",
                    package="MBHdesign"))$soilMoisture
values( egDat) <- ( values( egDat) - min( values( egDat), na.rm=TRUE)) * 5
```

Take the Cluster Sample

The process of taking a clustered sample is almost as straightforward as a regular point-based sample, and the process mostly follows that of taking a point sample. However, some extra information is needed about the clusters themselves. The function quasiSamp.cluster performs all the necessary computations and produces an obvious output. We note that quasiSamp.cluster, like quasiSamp.raster, accepts only raster objects for the inclusion probabilities.

```
set.seed( 727)
#take the cluster sample
#increase mc.cores for faster processing
if( !usePrecompiledData){
  samp <- quasiSamp.cluster( nCluster=10, clusterSize=5, clusterRadius=5,
                                inclusion.probs = egDat, mc.cores=1)
} else{
  samp <- readRDS( system.file(
                    "extdata", "clusterSamp1.RDS",
                    package="MBHdesign"))}
#plot it over the egData data
plot( egDat)
#the sample points
points( samp$x, samp$y, pch=20, cex=0.5)
#the centres of the clusters
# (not sample points but potentially useful nevertheless)
points( attr( samp, "clusterDes")[,c("x","y")], pch=1, col='red', cex=0.5)
```

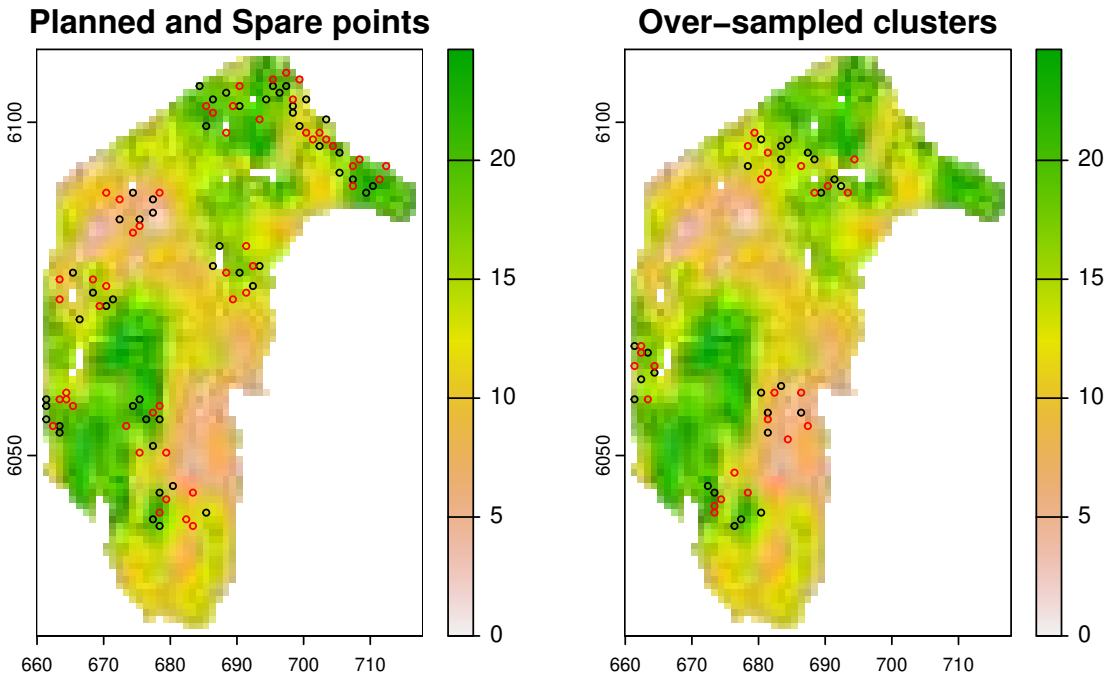


Take a Clustered Oversample

If/when an over-sample is required (e.g. , lar08,van18) so that a survey can be expanded at future dates, including replacement of non-response sites (missing values), then an extra step is required. This is because the working inclusion probabilities, which are calculated within `quasiSamp.clust`, are calculated for the number of *planned observations* and not the size of the *planned over-sample*. Within the framework of MBHdesign this is achieved by first calculating the working inclusion probabilities (see Foster *et al.* in press) for the 'correctly sized' design and then useing these as the basis for an over-sample.

In the design that follows, an over sample of clusters (15 drawn for 10 initial target) is taken. The planned cluster size is 5, but 10 are taken for each cluster to allow for drop-out. A property of BAS is that any contiguous set of samples is spatially-balanced, and this applies within each cluster and between clusters too.

```
#Create the working probabilties for the correct sized cluster.
if( !usePrecompiledData){
  workProbs <- alterInclProbs.cluster( nCluster=15, clusterSize=5,
                mc.cores=1, clusterRadius=5, inclusion.probs=egDat)
} else{
  workProbs <- readRDS( system.file(
                    "extdata", "clusterWorkProbs1.RDS",
                    package="MBHdesign"))}
#take the (over-sample)
set.seed( 747)
overSamp <- quasiSamp.cluster( nCluster=15, clusterSize=10,
                clusterRadius=5, working.inclusion.probs = workProbs)
#plot the results
par( mfrow=c(1,2))
plot( egDat, main="Planned and Spare points")
#the planned sample
points( overSamp[overSamp$cluster<=10 & overSamp$point<=5,c("x","y")], cex=0.5)
#the over-sample (within clusters 1:10)
points( overSamp[overSamp$cluster<=10 & overSamp$point>5,c("x","y")],
                cex=0.5, col='red')
plot( egDat, main="Over-sampled clusters")
#the overs-sampled clusters (themselves oversampled)
points( overSamp[overSamp$cluster>10 & overSamp$point<=5,c("x","y")], cex=0.5)
points( overSamp[overSamp$cluster>10 & overSamp$point>5,c("x","y")],
                cex=0.5, col='red')
```



Last Things Last

The only remaining thing to do is to tidy up our workspace. First, to export our sample locations. Second, to remove objects for this analysis from your workspace.

```
##write csv
#write.csv( as.data.frame( overSamp),
#      file="clusterSamp1.csv", row.names=FALSE)
#tidy
rm( list=c("egDat","overSamp","workProbs","samp","usePrecompiledData"))
```

Acknowledgements

This package started life as part of the Marine Biodiversity Hub, a collaborative partnership supported through funding from the Australian Government's National Environmental Science Program. Our goal is to assist decision-makers to understand, manage and conserve Australia's environment by funding world-class biodiversity science. NESP Marine Biodiversity Hub partners include the Institute for Marine and Antarctic Studies, University of Tasmania; CSIRO, Geoscience Australia, Australian Institute of Marine Science, Museum Victoria, Charles Darwin University, University of Western Australia, NSW Office of Environment and Heritage, NSW Department of Primary Industries and the Integrated Marine Observing System. You can check out the project here.

Since this initial project, its life has been moulded by the needs of other projects – notably the National Koala Monitoring Program.

References

Dumelle M, Kincaid TM, Olsen AR, Weber MH (2021). *spsurvey: Spatial Sampling Design and Analysis*. R package version 5.0.1.

Foster SD, Hosack GR, Lawrence E, Przeslawski R, Hedge P, Caley MJ, Barrett NS, Williams A, Li J, Lynch T, Dambacher JM, Sweatman HP, Hayes KR (2017). "Spatially balanced designs that incorporate legacy sites." *Methods in Ecology and Evolution*, **8**(11), 1433–1442. ISSN 2041-210X. doi:10.1111/2041-210X.12782. URL http://dx.doi.org/10.1111/2041-210X.12782.

Foster SD, Hosack GR, Monk J, Lawrence E, Barrett NS, Williams A, Przeslawski R (2020). "Spatially balanced designs for transect-based surveys." *Methods in Ecology and Evolution*, **11**(1), 95–105. doi:10.1111/2041-210X.13321. https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.13321, URL https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13321.

Foster SD, Lawrence E, Hoskins A (in press). "Spatially-Clustered Survey Designs." *Journal of Agricultural, Biological and Environmental Statistics*.

Frost A, Ramchurn A, Smith A (2018). "The Australian landscape water balance model (AWRA-L v6). Technical description of the Australian water resources assessment landscape model version 6."

Grafström A, Tillé Y (2013). "Doubly balanced spatial sampling with spreading and restitution of auxiliary totals." *Environmetrics*, **24**(2), 120–131. ISSN 1099-095X. doi:10.1002/env.2194. URL http://dx.doi.org/10.1002/env.2194.

Horvitz D, Thompson D (1952). "A Generalization of Sampling Without Replacement From a Finite Universe." *Journal of the American Statistical Association*, **47**(260), 663–685. ISSN 01621459. URL http://www.jstor.org/stable/2280784.

Robertson B, McDonald T, Price C, Brown J (2017). "A modification of balanced acceptance sampling." *Statistics & Probability Letters*, **129**, 107 – 112. ISSN 0167-7152. doi:https://doi.org/10.1016/j.spl.2017.05.004. URL http://www.sciencedirect.com/science/article/pii/S0167715217301797.

Robertson BL, Brown JA, McDonald T, Jaksons P (2013). "BAS: Balanced Acceptance Sampling of Natural Resources." *Biometrics*, **69**(3), 776–784. ISSN 1541-0420. doi:10.1111/biom.12059. URL http://dx.doi.org/10.1111/biom.12059.

Stevens D, Olsen A (2003). "Variance estimation for spatially balanced samples of environmental resources." *Environmetrics*, **14**(6), 593–610. ISSN 1099-095X. doi:10.1002/env.606. URL http://dx.doi.org/10.1002/env.606.

Taylor L (1961). "Aggregation, Variance and the Mean." *Nature*, **189**(4766), 732–735. URL http://dx.doi.org/10.1038/189732a0.

Venables W, Ripley B (2002). *Modern Applied Statistics with S. Fourth Edition*. Springer.

Appendix

Computational details

This vignette was created using the following R and add-on package versions

- R Under development (unstable) (2023-09-03 r85052), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_AU.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_AU.UTF-8`, `LC_COLLATE=en_AU.UTF-8`, `LC_MONETARY=en_AU.UTF-8`, `LC_MESSAGES=en_AU.UTF-8`, `LC_PAPER=en_AU.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_AU.UTF-8`, `LC_IDENTIFICATION=C`
- Time zone: `Australia/Hobart`
- TZcode source: `system (glibc)`
- Running under: `Ubuntu 20.04.6 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: fields 15.2, knitr 1.43, MASS 7.3-60.1, Matrix 1.5-3, MBHdesign 2.3.14, sf 1.0-13, spam 2.8-0, spsurvey 5.5.0, survey 4.1-1, survival 3.5-7, terra 1.7-39, viridisLite 0.4.1

- Loaded via a namespace (and not attached): abind 1.4-5, AlgDesign 1.2.0, assertthat 0.2.1, boot 1.3-28.1, cachem 1.0.4, callr 3.7.3, class 7.3-22, classInt 0.4-9, cli 3.6.1, codetools 0.2-19, compiler 4.4.0, crayon 1.5.2, crossdes 1.1-1, DBI 1.1.3, deldir 1.0-9, devtools 2.4.3, digest 0.6.31, dotCall64 1.0-1, dplyr 1.0.10, e1071 1.7-13, ellipsis 0.3.2, evaluate 0.21, fansi 1.0.4, fastmap 1.1.0, fs 1.6.2, generics 0.1.3, geometry 0.4.7, glue 1.6.2, gtools 3.9.2, highr 0.8, KernSmooth 2.23-22, lattice 0.21-8, lifecycle 1.0.3, lme4 1.1-27.1, lpSolve 5.6.18, magic 1.6-1, magrittr 2.0.3, maps 3.3.0, memoise 2.0.0, mgcv 1.9-0, minqa 1.2.4, mitools 2.4, mvtnorm 1.2-3, nlme 3.1-163, nloptr 1.2.2.2, parallel 4.4.0, pillar 1.9.0, pkgbuild 1.2.0, pkgconfig 2.0.3, pkgload 1.3.2, prettyunits 1.1.1, processx 3.8.1, proxy 0.4-27, ps 1.7.5, purrr 1.0.1, R6 2.5.1, randtoolbox 2.0.4, Rcpp 1.0.11, remotes 2.4.2, rlang 1.1.1, rngWELL 0.10-9, rprojroot 2.0.3, rstudioapi 0.13, sampling 2.9, sessioninfo 1.1.1, splines 4.4.0, tibble 3.2.1, tidyselect 1.2.0, tools 4.4.0, units 0.8-2, usethis 2.1.6, utf8 1.2.3, vctrs 0.6.3, withr 2.5.0, xfun 0.40

Affiliation:

Scott D. Foster
CSIRO
Marine Laboratories
GPObox 1538
Hobart 7001
Australia E-mail: scott.foster@csiro.au