Downscaling species occupancy: an introduction and tutorial

Charles J. Marsh

July 28, 2017

Contents

1 Introduction to downscaling

In order to assess and manage the status of a species we need to know the abundance of individuals in the population(s) and their changes over time. For the vast majority of species this information is unobtainable, however one important proxy of true abundance and extinction risk is the area occupied by the species. For example, the area of occupancy (AOO) is a little-used measure of conservation status in the IUCN red list (IUCN 2014). Although easier to estimate than true abundance, the difficulty in estimating AOO lies in the extensive sampling required across the full range of the species at a grain size sufficiently fine to give meaningful estimates. For the majority of species this is still impractical or unfeasible at these grain sizes leaving a large number of unsampled cells and therefore false absences. However, as we estimate occupancy at increasing grain sizes we increase our confidence in our presence-absence predictions. Such coarse-grain atlas data, generally generated from opportunistic recording over extended periods of time, are much more widely available. However, at such coarse grain sizes we also lose resolution in our status estimates as occupancies at large grain sizes are less closely correlated with true abundance (Hartley and Kunin 2003).

A solution is to employ the occupancy-area relationship (OAR); the increase in the area occupied by a species as grain size increases (Kunin 1998). If the relationship can be described at these coarser grain sizes where confidence is high, then we can extrapolate to predict occupancy at the fine grain sizes more closely related to the true abundance and conservation status.

Many models have been proposed to describe this geometric relationship, and it appears that no one model consistently provides the best predictions (Azaele et al. 2012, Barwell et al. 2014). This package provides functions for ten commonly applied model: Nachman, power law, logistic, poisson, negative binomial, generalised negative binomial, improved negative binomial, finite negative binomial, Thomas and Hui models. The Hui model (Hui et al. 2006, 2009) is unique in that it only requires data from a single spatial scale but does require the spatial relationships (i.e. cell coordinates) of all cells. The other nine models require the occupancies at multiple grain sizes. The package then attempts to optimise the model parameters to fit the occupancy data in log-log space. Once each model is parameterised the relationships are extrapolated to estimate occupancy at finer grain sizes.

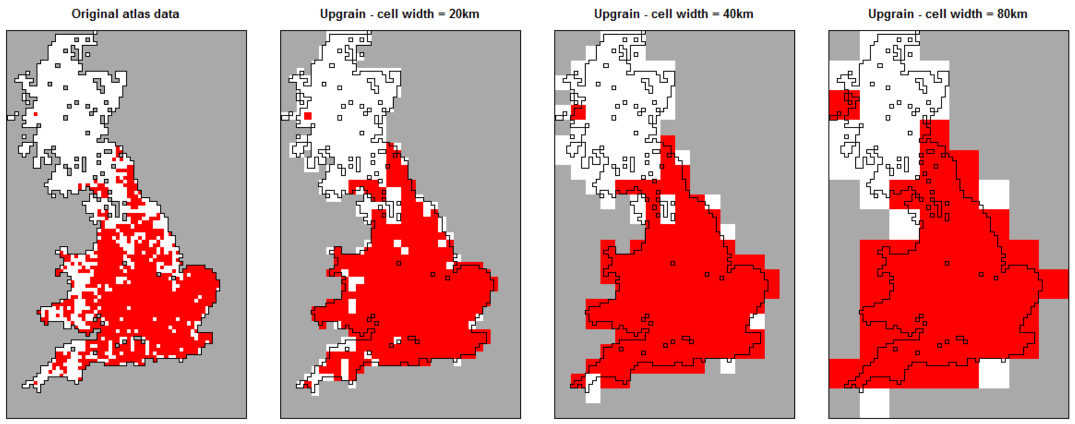

Original atlas data
Upgrain - cell width = 20km
Upgrain - cell width = 40km
Upgrain - cell width = 80km

Figure 1: **"Upraining" of atlas data at a 10km cell width to 20km, 40km and 80 km. As the grain size is increased occupancy also increases.**

As well as the fitting, prediction and plotting of the downscaling models, the package also contains functions for preparing coarse-scale data ("upgraining"; fig. 1). The tutorial vignette

"Upgraining atlas data for downscaling: threshold selection using `upgrain.threshold`" provides an overview of the different methods for achieving this.

```
vignette("Upgraining", package = "downscale")
```

For downscaling it is important to check the data for the scale of saturation and endemism. The scale of saturation is the grain size where all cells are occupied (fig. 2a) i.e. the proportion of occupancy = 1. The scale of endemism is the grain size where all presences occur in a single cell (2b). All data above these grain sizes should be discarded as they provide no further information for modelling the occupancy-area curve. The downscale functions will automatically set these occupancies to NA for modelling purposes, but this may lead to insufficient scales (less than three) remaining for modelling.

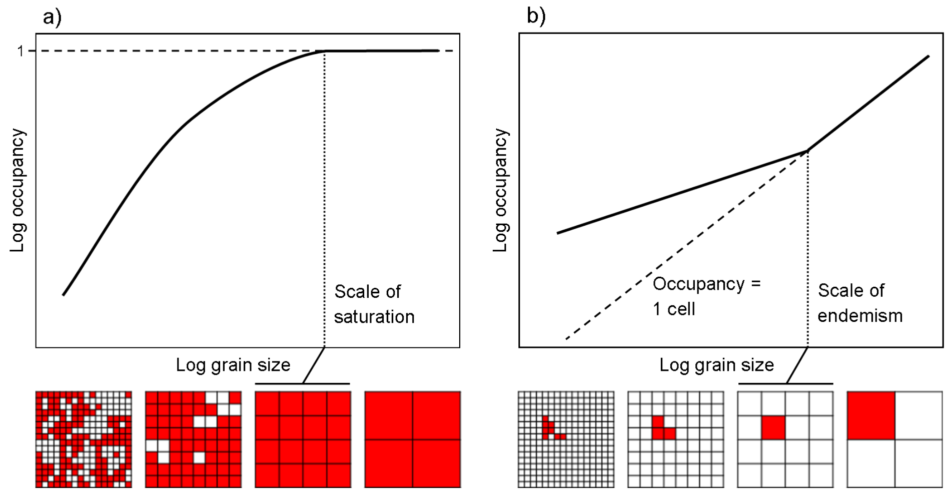


Figure 2: **Occupancy-area relationships (OAR) for two species showing a) the scale of saturation (the grain size at which all cells are occupied) and b) the scale of endemism (the scale at which only one cell is occupied).**

2 Using the downscale package

The general flow of the `downscale` package is presented in fig. 3. The package is based around seven user functions: `upgrain.threshold`, `upgrain`, `downscale`, `predict`, `hui.downscale`, `plot`, and `ensemble.downscale`. Ten downscaling models are available (Nachman, power law, logistic, poisson, negative binomial, generalised negative binomial, improved negative binomial, finite negative binomial, Thomas and Hui models). Details of all models can be found in the help files, and in the supplementary information of Barwell et al. 2014.

The user may input four types of data:

1) A data frame with columns for grain sizes (cell area) and occupancies in that order;
2) A data frame of cell coordinates and presence-absence data (presence = 1; absence = 0). Column names must be "x", "y", and "presence" in that order;
3) A raster layer of presence-absence data (presence = 1; absence = 0; no data = NA).
4) A SpatialPointsDataFrame of cell coordinates along with a data frame of presence-absence data raster layer of presence-absence data (presence = 1; absence = 0; no data = NA).

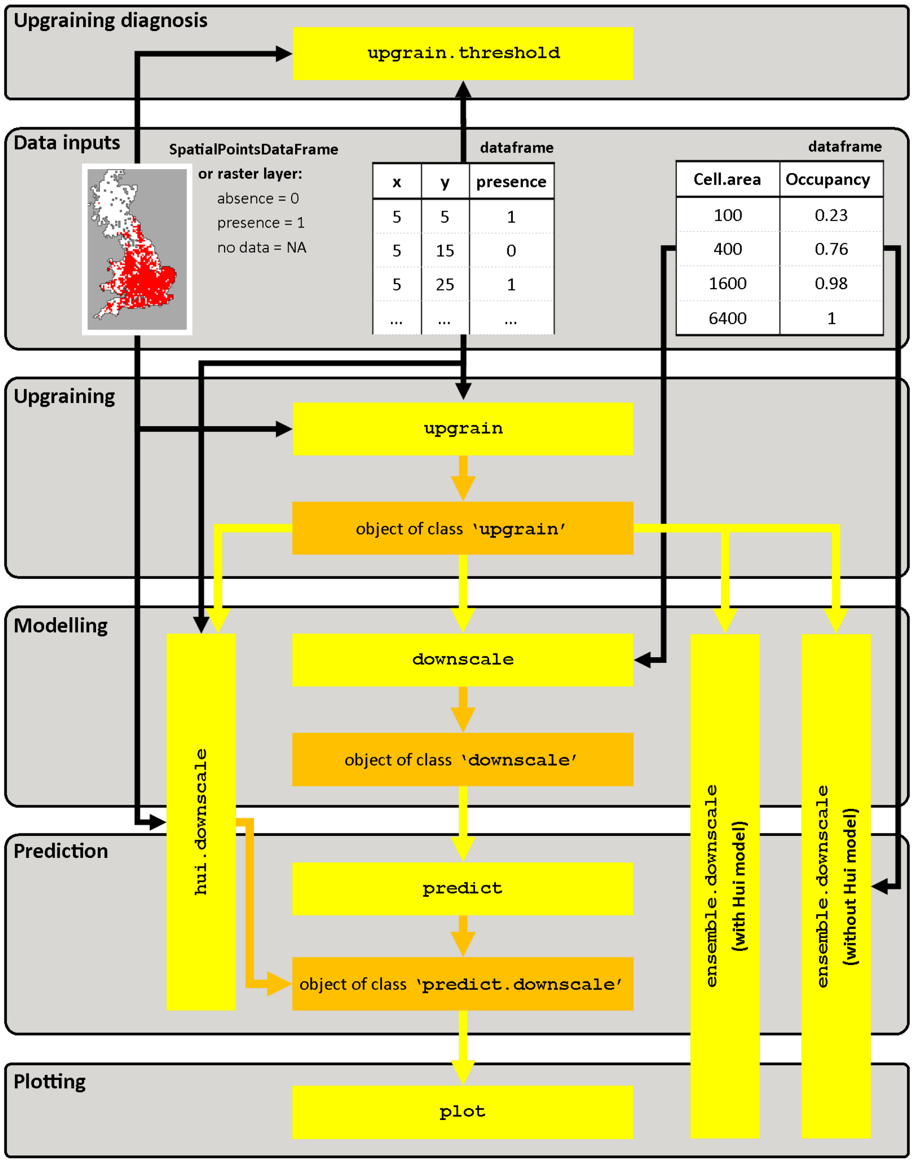


Figure 3: **Structure of the `downscale` package showing the flow between the seven functions (yellow) and the three output object classes (orange). Black arrows represent the input of raw data of three types: a raster layer of presence-absence data, a SpatialPointsDataFrame where the data are presence-absences, a data frame of cell coordinates and presence-absence for each cell, and a data frame of occupancies at coarse-grain sizes. Yellow arrows are where the output of one function may be used as the input for the next function**

To carry out downscaling with the Hui model (Hui et al. 2006, 2009) or upgraining of atlas data (and exploration of upgraining thresholds) then the input data must be of type 2, 3 or 4. The table below shows the functions to use to achieve desired objectives with regards to input data. In all cases `upgrain.threshold` can be used to explore thresholds for `upgrain`.

Input data type	**Objective**	**Function flow**
Data frame of cell areas and occcupancies	Downscale (excluding Hui model)	`downscale` ⇒ `predict` ⇒ `plot`
Data frame of cell coordinates and presence-absence data	Downscale (excluding Hui model)	`upgrain` ⇒ `downscale` ⇒ `predict` ⇒ `plot`
Raster layer of presence-absence data	Downscale (excluding Hui model)	`upgrain` ⇒ `downscale` ⇒ `predict` ⇒ `plot`
SpatialPointsDataFrame with presence-absence data	Downscale (excluding Hui model)	`upgrain` ⇒ `downscale` ⇒ `predict` ⇒ `plot`
Data frame of cell coordinates and presence-absence data	Downscale (including Hui model)	(`upgrain` ⇒) `hui.downscale` ⇒ `plot`
Raster layer of presence-absence data	Downscale (including Hui model)	(`upgrain` ⇒) `hui.downscale` ⇒ `plot`
SpatialPointsDataFrame with presence-absence data	Downscale (including Hui model)	(`upgrain` ⇒) `hui.downscale` ⇒ `plot`
Data frame of cell areas and occcupancies	Ensemble modelling (excluding Hui model)	`ensemble.downscale`
Data frame of cell coordinates and presence-absence data	Ensemble modelling (with or without Hui model)	`upgrain` ⇒ `ensemble.downscale`
Raster layer of presence-absence data	Ensemble modelling (with or without Hui model)	`upgrain` ⇒ `ensemble.downscale`
SpatialPointsDataFrame with presence-absence data	Ensemble modelling (with or without Hui model)	`upgrain` ⇒ `ensemble.downscale`

Table 1: **Flow of functions for downscaling with respect to the type of input data, and if the user wishes to apply the Hui model or not.**

3 Package tutorial

First, we must download the downscale package from CRAN if not already done so.

```
install.packages("downscale")
```

Then load in the library

```
library("downscale")
```

3.1 A quick example

We will start with the simplest example of using the downscaling package, where we already have occupancy data across a number of grain sizes. We first create some imaginary data; a data frame where the first column are the cell areas (grain size) and the proportion of occupancy as the second column:

```
occupancy <- data.frame(Cell.area = c(100, 400, 1600, 6400),
                        Occupancy = c(0.23, 0.56, 0.87, 1))
```

Now we use downscale to estimate the model parameters for the logistic model to the data. Note: for this type of data input we must also specify the extent (the total area over which occupancy has been measured) which is necessary for converting the modelled proportion of occupancies to area of occupancy (AOO) later on. In this imaginary data we will set extent to be 320000 km^2:

```
## fit logistic model to observed data using downscale
logis.mod <- downscale(occupancies = occupancy,
                       model = "Logis",
                       extent = 320000)

## this creates an object of class 'downscale'
logis.mod
```

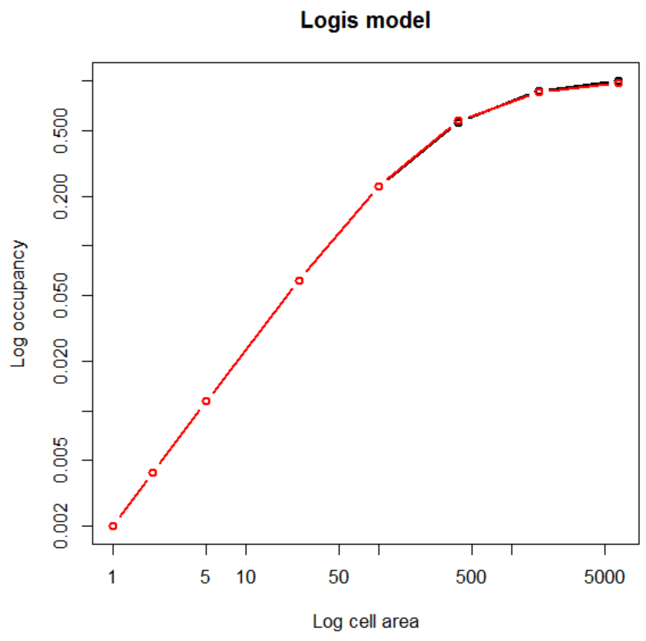
```
$model
[1] "Logis"

$pars
          C           z 
0.002014927 1.083725424 

$observed
  Cell.area Occupancy
1       100      0.23
2       400      0.56
3      1600      0.87
4      6400      1.00

$extent
[1] 320000

attr(,"class")
[1] "downscale"
```

The downscale function has estimated best-fit parameters of 0.00201 for `C` and 1.0837 for `z` for the logistic model. We then take these parameters from the '`downscale`' object to extrapolate the fitted logistic function to predict occupancies at finer grain sizes. We will first create a vector of grain sizes (cell area) to predict. If we include the original cell sizes used for modelling we can also observe the model fit.

```
## new grain sizes to predict
areas.pred <- c(1, 2, 5, 25, 100, 400, 1600, 6400)

## predict for the new grain sizes using the downscale object
logis.pred <- predict(logis.mod,
                      new.areas = areas.pred,
                      plot = FALSE)

## this creates an object of class ‘predict.downscale’
## occupancy is given as a proportion (Occupancy) and area of occupancy (AOO)
logis.pred$predicted
```

```
  Cell.area   Occupancy         AOO
1         1 0.002010876    643.4802
2         2 0.004252482   1360.7942
3         5 0.011396541   3646.8932
4        25 0.061873397  19799.4871
5       100 0.228564935  73140.7793
6       400 0.570999511 182719.8435
7      1600 0.856717834 274149.7068
8      6400 0.964106833 308514.1867
```

```
## now we can plot the predictions with log-log axes.
## Black points are the observed values, red points are the predicted values
plot(logis.pred)
```



3.2 Preparing atlas data for downscaling

For the majority of cases we will have atlas data at a single scale. For downscaling we will need to therefore upgrain the atlas data. Read in the atlas data for a hypothetical UK species provided in the package. In this case the format is a data frame of sample cell coordinates and presence-absence data but it could also be a raster layer:

```
## if it is not already loaded, load in the package
library(downscale)
data.file <- system.file("extdata", "atlas_data.txt", package = "downscale")
atlas.data <- read.table(data.file, header = TRUE)
```

The data frame must have the column names “x”, “y” and “presence”:

```
head(atlas.data)
```

```
     x  y presence
1 8170 10        0
```

```
2 8130  20          0
3 8140  20          0
4 8160  20          0
5 8170  20          0
6 8140  30          0
```

The first step is to upgrain the atlas data to calculate occupancy at larger grain sizes than the atlas data – this provides the occupancy data points to fit the different downscaling models to. If we simply increase the cell sizes of the atlas data then the extent also increases. As downscaling models requires the proportion of occupancy at each grain, if the proportions are calculated from a different total area in each case we may run in to problems. Therefore it is important that we fix the extent of all grain sizes to the extent of the largest grain size, but this means compromising between assigning unsampled cells as absences or excluding sampled cells. We therefore carry this out by applying a threshold to the proportion of a coarse-scale cell that has been sampled at the fine-scale. For example, if a 40km width cell was only sampled by a single 10km atlas absence cell within it, we may want to discard it as a) it is likely that at least one of the unsampled cells may actually be a presence (coarse-scale false absence), and b) all the unsampled atlas cells would be assigned as absences even though they have not been sampled (atlas-scale false absence). However, discarding the cell will also lead to some loss of information.

The choice of threshold can have important consequences on the model predictions and so it is highly recommended to read the vignette "Upgraining atlas data for downscaling: threshold selection using `upgrain.threshold`" and the function helpfile (`?upgrain.threshold`) for more detail on creating your multi-scale standardised atlas data:

```
vignette("Upgraining", package = "downscale")
```

The `upgrain.threshold` allows the user to explore these trade-offs through several plots, and provides four recommendations for possible threshold selections: including all the sampled cells (`All_Sampled`); including only cells at the largest grain size that were completely sampled at the atlas scale (`Sampled_Only`); a species-specific threshold that retains all species occurrences (`All_Occurrences`); and an atlas-specific threshold that maintains the same extent as the original atlas data (`Gain_Equals_Loss`).

```
## explore thresholds using upgrain.threshold
thresh <- upgrain.threshold(atlas.data = atlas.data,
                            cell.width = 10,
                            scales = 3)
```

This gives two sets of plots. First is a set of four plots that explore the trade-offs between discarding sampled cells and making assumptions about unsampled cells, which are automatically assigned as absences.

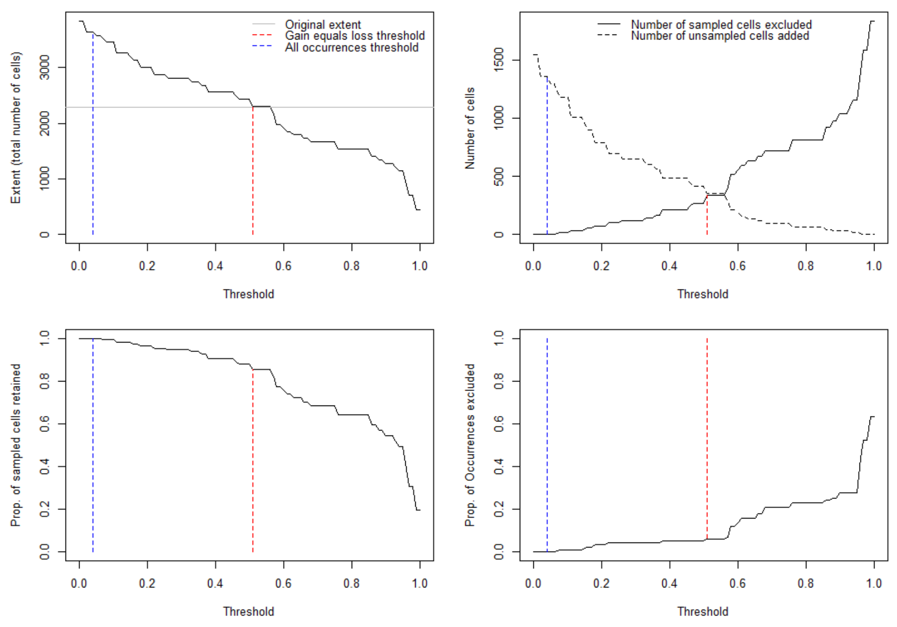
The second set of plots (hit `return` or click on the plot window to view the second window) are the standardised atlas data generated after applying the four different threshold criteria (`"All_Sampled"`, `"All_Occurrences"`, `"Gain_Equals_Loss"` and `"Sampled_Only"`).

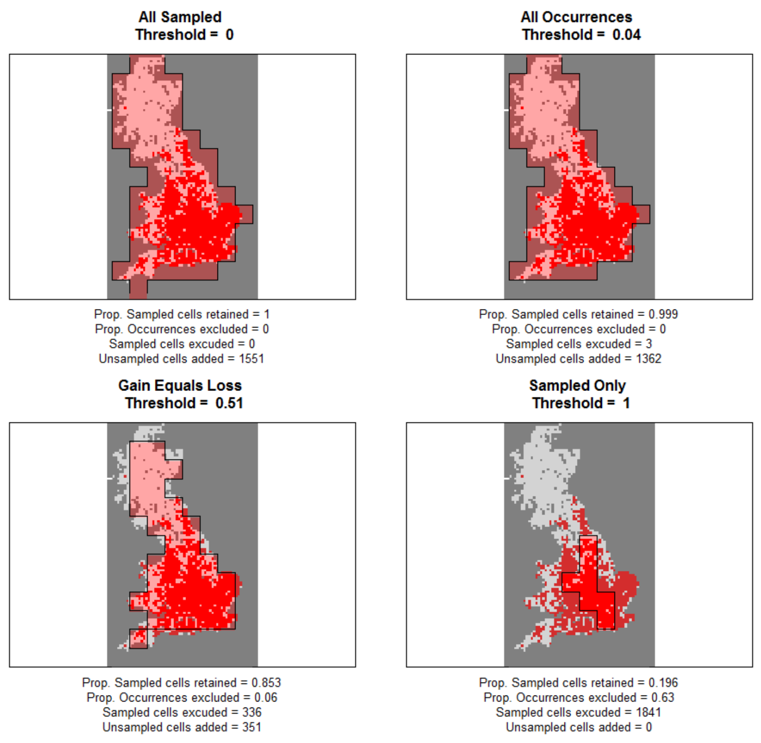
We can see the threshold values for the four threshold criteria

```
thresh$Thresholds
```

```
  All_Sampled All_Occurrences Gain_Equals_Loss Sampled_Only
1           0            0.04             0.51            1
```

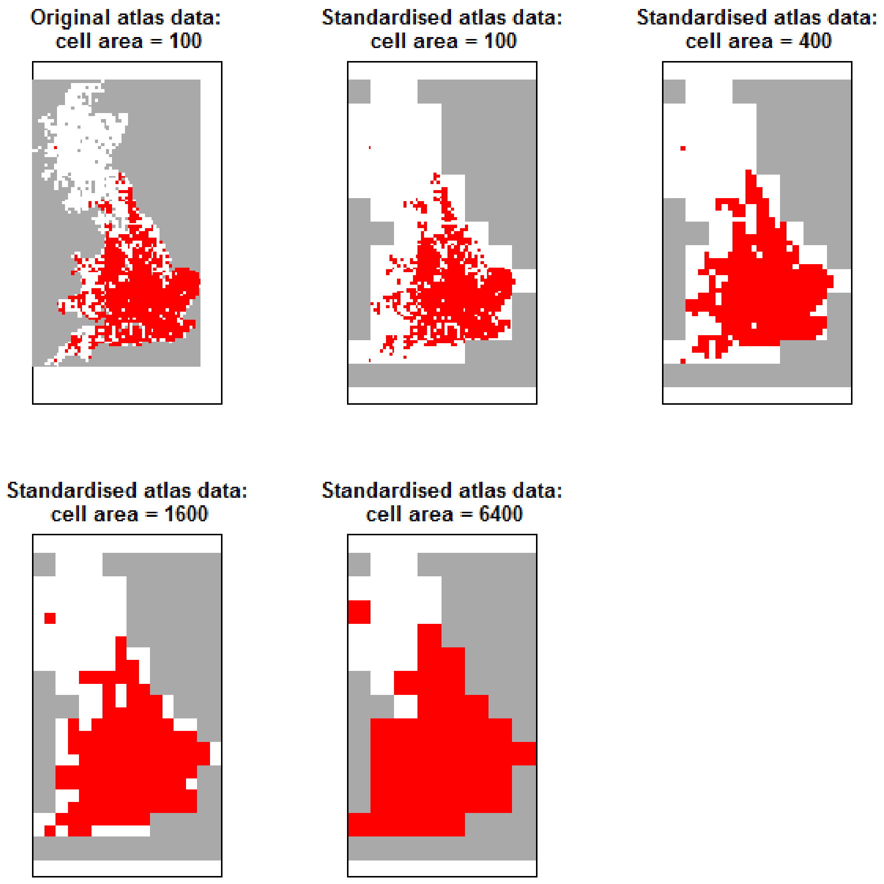
Once the user has decided on a threshold value (any value between 0 and 1 can be selected) or one of the threshold criteria, the `upgrain` function will prepare the atlas data for downscaling. For now we'll use one of the pre-defined options `"All_Occurrences"` which ensures that all




All Sampled
Threshold = 0
Prop. Sampled cells retained = 1
Prop. Occurrences excluded = 0
Sampled cells excuded = 0
Unsampled cells added = 1551
All Occurrences
Threshold = 0.04
Prop. Sampled cells retained = 0.999
Prop. Occurrences excluded = 0
Sampled cells excuded = 3
Unsampled cells added = 1362
Gain Equals Loss
Threshold = 0.51
Prop. Sampled cells retained = 0.853
Prop. Occurrences excluded = 0.06
Sampled cells excuded = 336
Unsampled cells added = 351
Sampled Only
Threshold = 1
Prop. Sampled cells retained = 0.196
Prop. Occurrences excluded = 0.63
Sampled cells excuded = 1841
Unsampled cells added = 0

occurrence records are retained. This creates an object of class `'upgrain'` which can then then be used directly as an input for `downscale`, along with plots of the original and standardised atlas data at each scale. We will save the `'upgrain'` object here for subsequent analyses in the next section.

```
## upgrain data (using All Occurrences threshold)
occupancy <- upgrain(atlas.data,
                     cell.width = 10,
                     scales = 3,
                     method = "All_Occurrences",
                     plot = TRUE)
```



3.3 Downscaling the atlas data - more detailed examples

As we can pass our '`upgrain`' object directly in to the `downscale` function we no longer require to specify the extent. Let's try the improved negative binomial model (INB) first:

```
## Improved Negative Binomial model
(inb <- downscale(occupancies = occupancy,
                  model = "INB"))

$model
[1] "INB"

$pars
         C      gamma          b
1.42670491 0.04906771 1.82970792

$observed
```

```
  Cell.area Occupancy
1       100 0.2713542
2       400 0.4122807
3      1600 0.5219298
4      6400 0.7017544

$extent
[1] 364800

attr(,"class")
[1] "downscale"
```

The downscaling functions use an optimisation procedure to fit the models to the upgrained occupancy data. Suitable starting values for model parameters are automatically inputted, however if the models aren't converging then it is possible to specify user-specific parameters. The table below shows the default starting parameters implemented.

Model	**Parameter 1**	**Parameter 2**	**Parameter 3**
`Nachman`	`"C"` = 0.01	`"z"` = 0.01	
`PL`	`"C"` = 0.01	`"z"` = 0.01	
`Logis`	`"C"` = 0.01	`"z"` = 0.01	
`Poisson`	`"gamma"` = 1e-8		
`NB`	`"gamma"` = 0.01	`"k"` = 0.01	
`GNB`	`"C"` = 0.00001	`"z"` = 1	`"k"` = 0.01
`INB`	`"C"` = 1	`"gamma"` = 0.01	`"b"` = 0.1
`FNB`	`"N"` = 10	`"k"` = 10	
`Thomas`	`"rho"` = 1e-8	`"mu"` = 10	`"sigma"` = 1

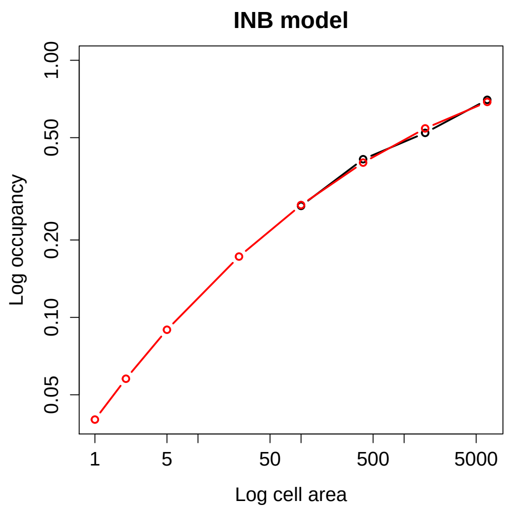
If using your own parameters, they must be in the form of a list with the same parameter names (take particular note of capitals) as the original starting parameters:

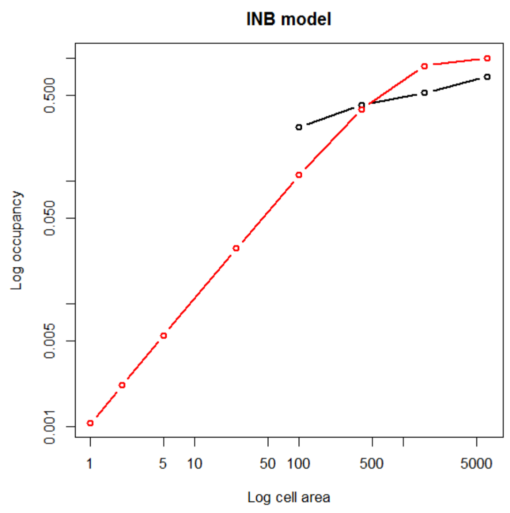
```
## Specifying the starting parameters
params.new <- list("C" = 0.1, "gamma" = 0.00001, "b" = 0.1)
inb.new <- downscale(occupancies = occupancy,
                     model = "INB",
                     starting_params = params.new)
```

We can visually compare the two to see which has a better fit by extrapolating the modelled curves to finer grain sizes using `predict` as before (plotting can be called directly from `predict` or through `plot`). The first plot is the prediction using the original parametersm and the second plot using the new parameters (a much worse fit in this case):

```
## plot the predictions of two FNB models using predict.downscale
inb.pred <- predict(inb,
                    new.areas = c(1, 2, 5, 25, 100, 400, 1600, 6400),
                    plot = TRUE)
inb.pred.new <- predict(inb.new,
                        new.areas = c(1, 2, 5, 25, 100, 400, 1600, 6400),
                        plot = TRUE)
```

The Thomas model involves an integration process that can be time-consuming to run. For this reason the user may alter the tolerance during integration – the finer the tolerance the more accurate the prediction but the longer the computation time. It can therefore be a good idea to initially try a larger tolerance value than the default ($1e^{-6}$) in order to ascertain if the





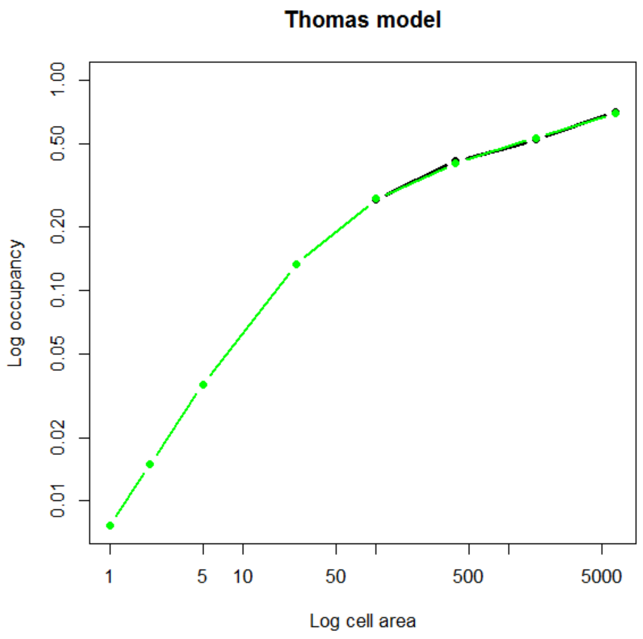
starting parameters are likely to be correct. You can then always use the parameter estimates as the starting parameters when using a smaller tolerance value.

```
## Thomas model
thomas <- downscale(occupancies = occupancy,
                    model = "Thomas",
                    tolerance = 1e-3)

## the tolerance can also be set for the predict function
thomas.pred <- predict(thomas,
                       new.areas = c(1, 2, 5, 25, 100, 400, 1600, 6400),
                       tolerance = 1e-6)

## When plotting the results we can also change the look of the graphics
plot(thomas.pred,
     col.pred = "green",  # change the colour of the prediction
     pch = 16,            # change point character
     lwd.obs = 3)         # change line width of the observed data
```

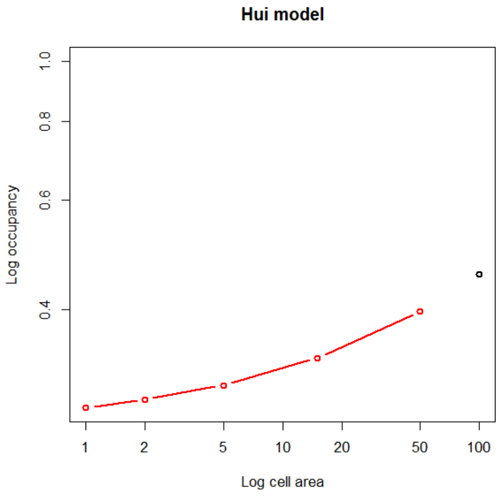
The Hui model is slightly different from the other downscaling models in that it does not need occupancy from multiple scales. Instead, it only takes the coordinates of presence-absence data at the atlas scale and uses this to calculate occupancy at finer grain sizes. For this reason it is implemented using a seperate function, `hui.downscale`, which in effect runs `downscale` and `predict.downscale` in a single step.

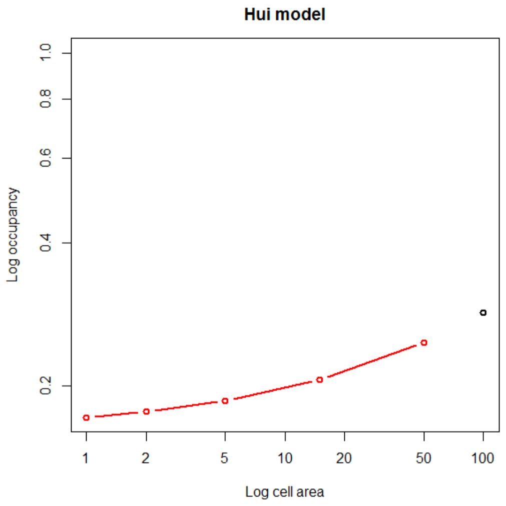
The input data must either be a presence-absence raster layer of the atlas data, or a data frame of cell coordinates and presence-absence data. Additionally the function requires the cell widths of the input data, and if using a data frame as the input data, the total extent, and the grain sizes (cell area) for which we wish to predict occupancy. These must be smaller than the cell area of the input data. Like the Thomas model, the tolerance can be specified if the results appear inaccurate (set tolerance to a smaller number) or takes extensive programming time (set tolerance to a larger number).

```
## Hui model using a data frame as input
hui <- hui.downscale(atlas.data,
                     cell.width = 10,
```



```
                     extent = 320000,
                     new.areas = c(1, 2, 5, 15, 50))

## the output is a normal 'predict.downscale' object
plot(hui)

## Or we can use the 'upgrain' object as input
hui <- hui.downscale(occupancy,
                     cell.width = 10,
                     new.areas = c(1, 2, 5, 15, 50),
                     plot = TRUE)
```





It is critical to note here that the proportion of occupancies are very different between the two plots. This is because the extents are different between the original atlas data 320000 km^2 and the standardised atlas data 364800 km^2. If comparing predictions using multiple models it is crucial to use the same standardised data in all cases, or else only compare the converted area of occupancies (AOO) and not the proportion of occupancies.

3.4 Ensemble modelling

No single model appears to provide the most accurate fine-scale occupancy predictions in all cases, and it is difficult to predict which model will in a given situation. The ensemble function will model and predict occupancy for multiple models simultaneously, and also applies a simple model averaged prediction (the means of the log occupancies). Some or all of the models can be selected. Again, lets start where our data is a data frame of occupancies at each grain size:

```
## hypothetical occupancy data
occupancy <- data.frame(Cell.area = c(100, 400, 1600, 6400),
                        Occupancy = c(0.23, 0.56, 0.87, 1))

## grain sizes (cell areas) to predict
areas.pred <- c(1, 2, 5, 25, 100, 400, 1600, 6400)
```

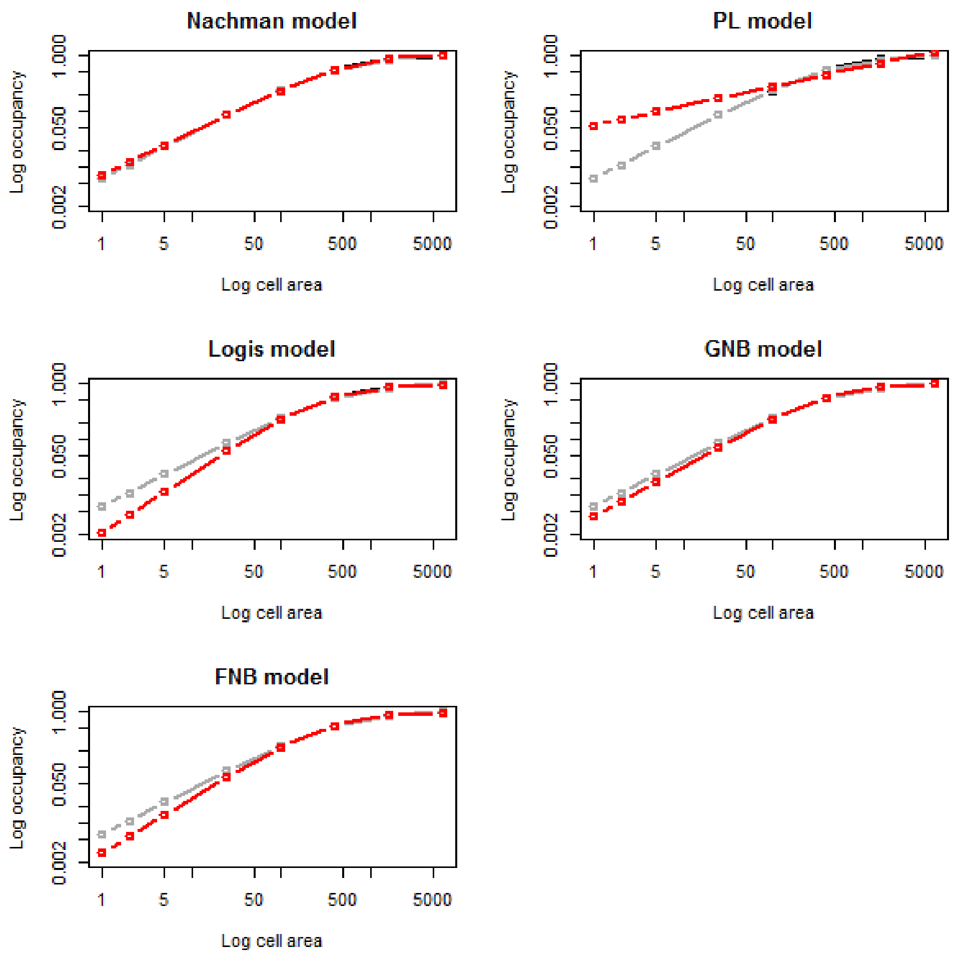
The `ensemble.downscale` function does the modelling and predicting in a single step so we need a few more arguments than when just using `downscale`: the cell areas of the fine grain sizes we wish to predict, the total extent and the models we wish to apply. Also note, with this type of input data we can not apply the Hui model.

```
ensemble <- ensemble.downscale(occupancy,
                               new.areas = areas.pred,
                               extent = 320000,
                               models = c("Nachman",
                                          "PL",
                                          "Logis",
                                          "GNB",
                                          "FNB"),
                               plot = TRUE)
```

```
Nachman model is running...  complete
PL model is running...  complete
Logis model is running...  complete
GNB model is running...  complete
FNB model is running...  complete
```

```
## the model averaged predictions are in grey and the model predictions in red
## to print the predicted proportion of occupancies for each model
ensemble$Occupancy
```

```
  Cell.area    Nachman        PL      Logis        GNB        FNB      Means
1         1 0.00708498 0.0558199 0.00201088 0.00405949 0.00284089 0.00620137
2         2 0.01217964 0.0711372 0.00425248 0.00764569 0.00566805 0.01098108
3         5 0.02485181 0.0980179 0.01139654 0.01758727 0.01406806 0.02330096
4        25 0.08522024 0.1721162 0.06187339 0.07373553 0.06710741 0.08520463
5       100 0.23247468 0.2795353 0.22856494 0.23018897 0.22857598 0.23910431
6       400 0.54430571 0.4539953 0.57099951 0.55812208 0.56665466 0.53692736
7      1600 0.90314840 0.7373372 0.85671783 0.87738733 0.87016143 0.84685851
8      6400 0.99902662 1.1975149 0.96410683 0.98448699 0.97632517 1.02083964
```



```
## and print the predicted area of occupancies (AOO) for each model
ensemble$AOO
```

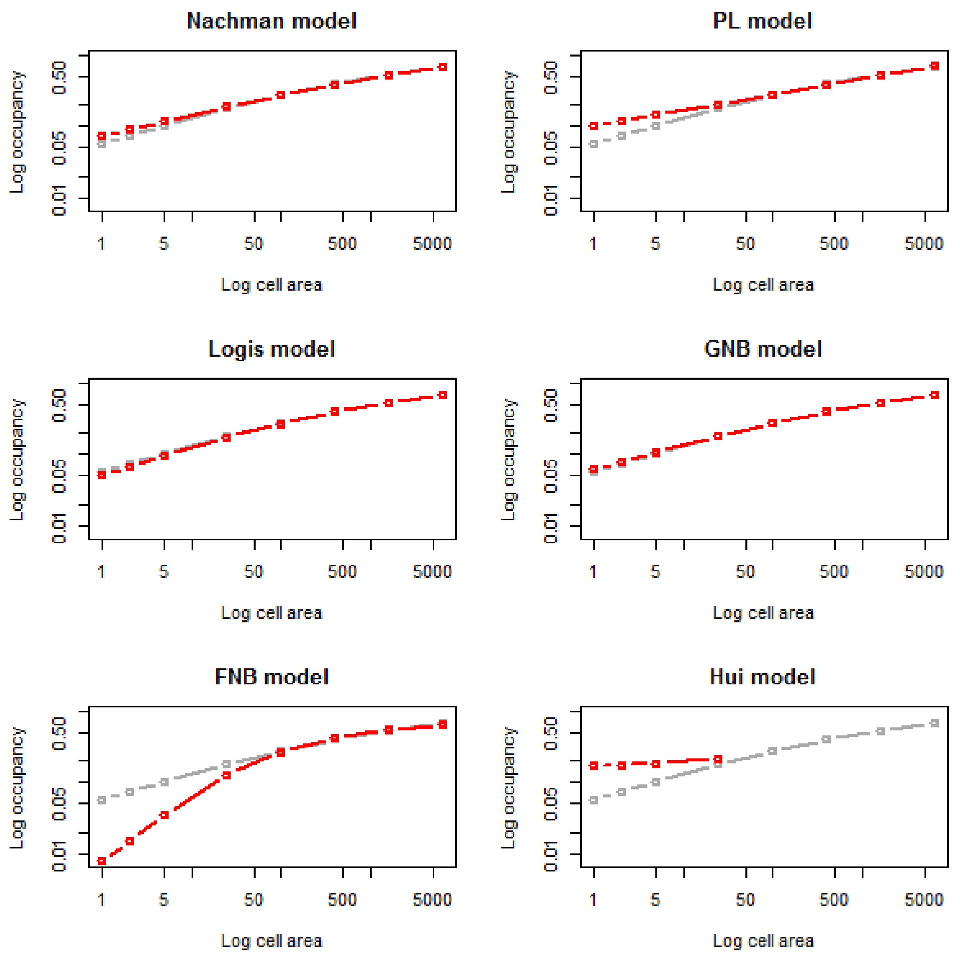
```
  Cell.area    Nachman        PL       Logis        GNB         FNB      Means
1         1   2267.193  17862.39    643.4802   1299.035    909.0862   1984.439
2         2   3897.484  22763.91   1360.7942   2446.623   1813.7772   3513.945
3         5   7952.579  31365.75   3646.8932   5627.925   4501.7776   7456.307
4        25  27270.478  55077.20  19799.4871  23595.370  21474.3707  27265.480
5       100  74391.897  89451.29  73140.7793  73660.469  73144.3146  76513.378
6       400 174177.827 145278.51 182719.8435 178599.064 181329.4907 171816.756
7      1600 289007.489 235947.91 274149.7068 280763.946 278451.6571 270994.723
8      6400 319688.520 383204.78 308514.1867 315035.837 312424.0534 326668.684
```

Alternatively, the input data may be an object of class ‘`upgrain`’, which also allows us to run the Hui model as long as we specify the cell width:

```
## read in atlas data
data.file <- system.file("extdata", "atlas_data.txt", package = "downscale")
atlas.data <- read.table(data.file, header = TRUE)
```

```
## upgrain data (using All Occurrences threshold)
occupancy <- upgrain(atlas.data,
                     cell.width = 10,
                     scales = 3,
                     method = "All_Occurrences",
                     plot = FALSE)

## ensemble modelling
ensemble <- ensemble.downscale(occupancy,
                               new.areas = areas.pred,
                               cell.width = 10,
                               models = c("Nachman",
                                          "PL",
                                          "Logis",
                                          "GNB",
                                          "FNB",
                                          "Hui"),
                              plot = TRUE)
```

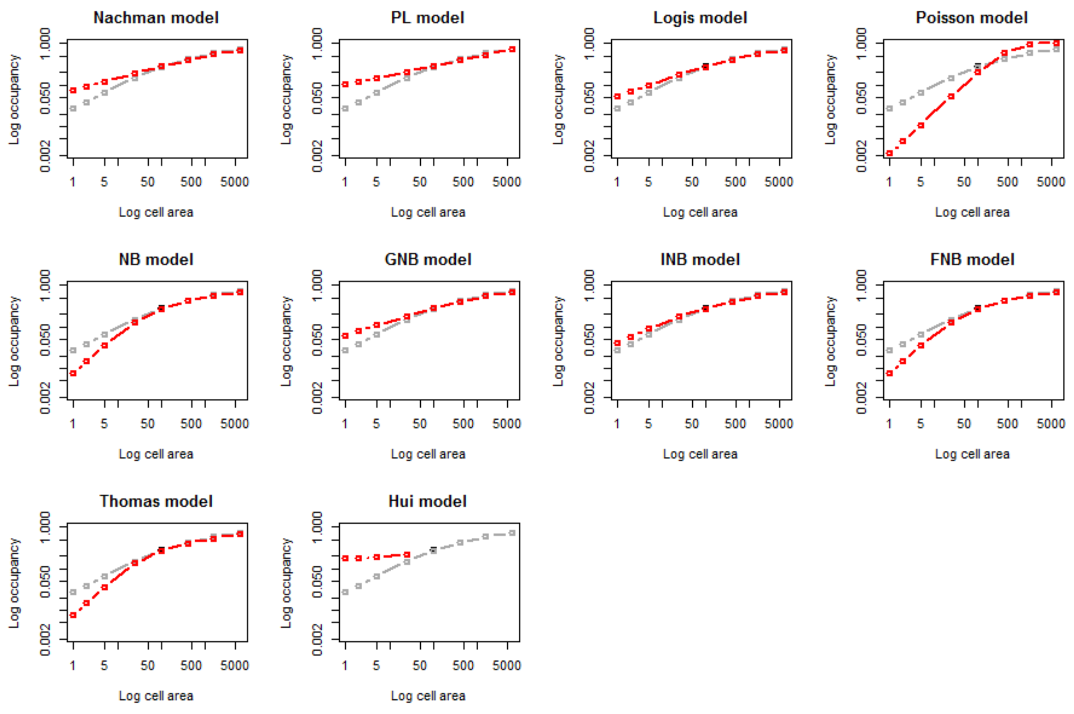
```
Nachman model is running...  complete
PL model is running...  complete
Logis model is running...  complete
GNB model is running...  complete
FNB model is running...  complete
Hui model is running...  complete
```



If we want to run all ten models we can specify `models = "all"`. Once again, we can set the tolerance values for the modelling (`tolerance_mod`) and prediction (`tolerance_pred`) of the Thomas model and the Hui model (`tolerance_hui`) to improve processing times or accuracy.

```
ensemble <- ensemble.downscale(occupancy,
                               new.areas = areas.pred,
                               cell.width = 10,
                               models = "all",
                               tolerance_mod = 1e-3,
                               plot = TRUE)
```

```
Nachman model is running...  complete
PL model is running...  complete
Logis model is running...  complete
Poisson model is running...  complete
NB model is running...  complete
GNB model is running...  complete
INB model is running...  complete
FNB model is running...  complete
Thomas model is running...  complete
Hui model is running...  complete
```



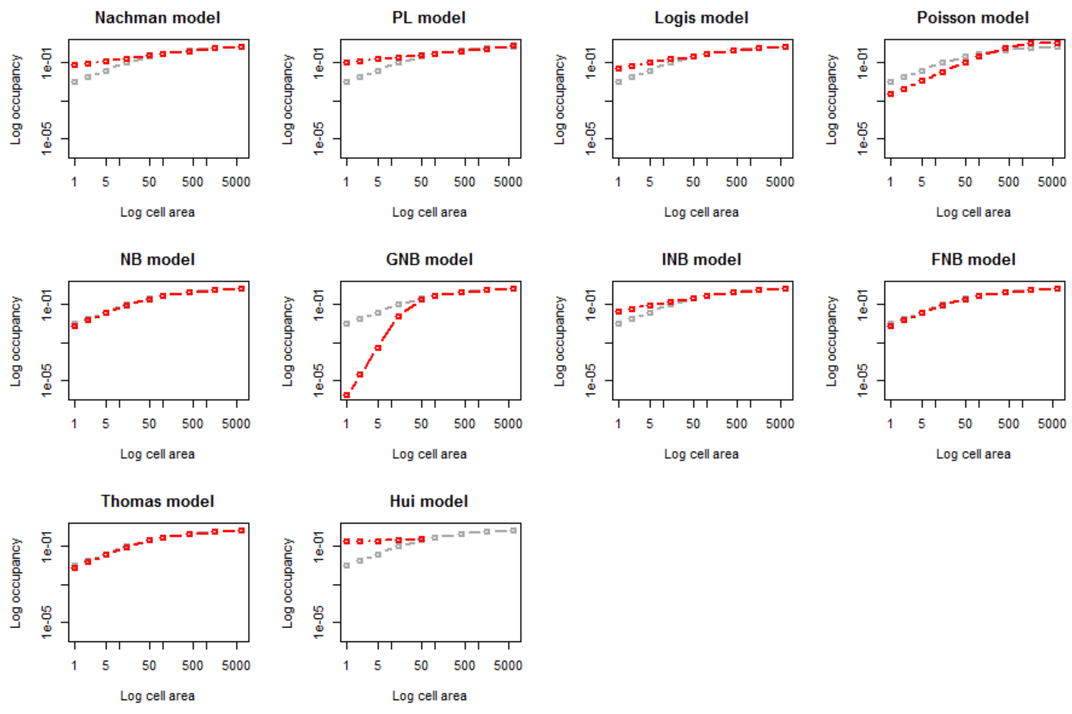
We can also specify the starting parameters for specific models. For each model the starting parameters should be in the form of a list as before, and each model list is an item in a combined list:

```
## Specifying starting parameters for Nachman and GNB models
new.params <- list(Nachman = list("C" = 0.1, "z" = 0.01),
                   GNB = list("C" = 0.1, "z" = 1, "k" = 0.01))
new.params
```

```
$Nachman
$Nachman$C
[1] 0.1

$Nachman$z
[1] 0.01


$GNB
$GNB$C
[1] 0.1

$GNB$z
[1] 1

$GNB$k
[1] 0.01
```

```
ensemble <- ensemble.downscale(occupancies = occupancy,
                               new.areas = c(1, 2, 5, 15, 50, 100, 400, 1600,
                                             6400),
                               cell.width = 10,
                               models = "all",
                               tolerance_mod = 1e-3,
                               starting_params = new.params,
                               plot = TRUE)
```

```
Nachman model is running...  complete
PL model is running...  complete
Logis model is running...  complete
Poisson model is running...  complete
NB model is running...  complete
GNB model is running...  complete
INB model is running...  complete
FNB model is running...  complete
Thomas model is running...  complete
Hui model is running...  complete
```



3.5 Creating atlas data from point records

It may be that instead of having pre-existing atlas data, we may need to create our own coarse-scale data from point records (for example herbarium records or GBIF data).

The grain size (cell width) needs to be carefully chosen so that we can best meet the assumption that all cells have been sampled. The larger the grain size the greater our confidence in each cell's status, but the further we will have to downscale and the fewer coarse-grain data points we will have for model fitting. If there are cells or regions where we do not expect this to be the case it may be best to change these to NAs rather than assign them as absences.

The library `rgbif` will automatically harvest GBIF data for a desired species for a specified region.

```
## if you need to, install the packages
install.packages("rgbif")

## load in the necessary libraries
library(rgbif)
library(downscale)
```

We'll get the UK records for the chalkhill blue (*Polyommatus coridon*), a butterfly species with a patchy breeding distribution largely in the south of the UK. We will confine ourselves to records only from the UK (`gbifopts = list(country = "GB")`).

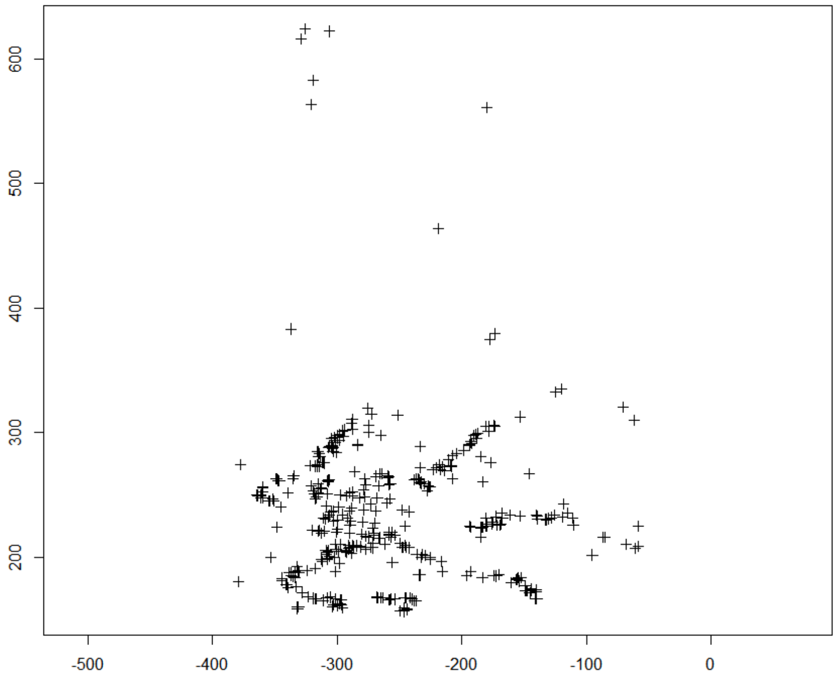
```
records <- occ_search(scientificName = "Polyommatus coridon",
                      country = "GB",
                      limit = 10000,
                      hasCoordinate = TRUE,
                      return = "data")

## extract just the coordinates of the occurrences
records.coords <- SpatialPoints(data.frame(Lon = records$decimalLongitude,
                                           Lat = records$decimalLatitude),
                                proj4string = CRS("+proj=longlat +datum=WGS84
                                                   +ellps=WGS84"))
```

The coordinates are in latitude/longitude based upon the WGS84 coordinate system. However, a lot of the point records for UK butterflies are collected based upon the British National Grid ("OSGB 36"), which can be seen in the grid-like pattern of some of the points so it makes sense to reproject the points to this coordinate system. As an added advantage we can also specify the units as km to make our upgraining easier.

```
## reproject the coordinates to British National Grid
records.coords <- spTransform(records.coords,
                              CRS("+proj=lcc
                                   +lat_1=49.8333339 +lat_2=51.16666733333333
                                   +lat_0=90 +lon_0=4.367486666666666
                                   +x_0=150000.01256 +y_0=5400088.4378
                                   +ellps=intl +units=km +no_defs"))
```

We can have a quick look at the point records if we like.

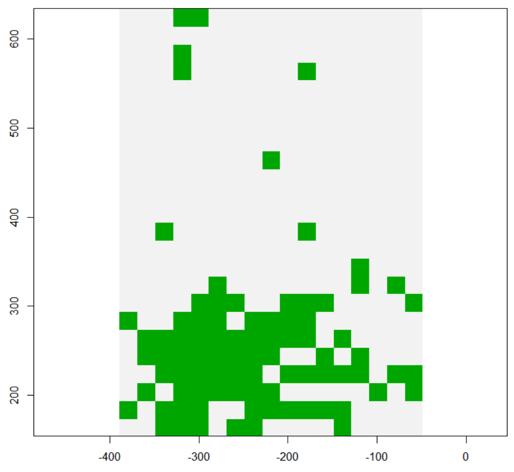
```
plot(records.coords, axes = T)
```



Now we have to convert these points in to a coarse-scale raster. The simplest method is to bound our raster by the limits of the location coordinates. Careful thought must also be put in to the grain size. It must be large enough that we are confident it is an accurate representation of presence-absence, but also small enough to allow upgraining to give at least three spatial scales worth of occupancy data for fitting the downscaling models. In the UK butterflies are generally sampled at a 10 km grid cell, so we'll set a grain size of 20 km width (400 km^2), which gives us a little more certainty in our atlas data but will still comfortably allow us to upgrain to give three scales (400, 1600, 6400 km^2).

```
## set grain size as 20 km
cell.width <- 20

# extract extent of coordinates
coords.extent <- extent(records.coords)

## create a blank raster to fit the coordinates (note the addition of half a
## cell width on all sides)
gbif_raster <- raster(xmn = coords.extent@xmin - (cell.width / 2),
                      xmx = coords.extent@xmax + (cell.width / 2),
                      ymn = coords.extent@ymin - (cell.width / 2),
                      ymx = coords.extent@ymax + (cell.width / 2),
                      res = cell.width)

## assign cells with presence records as 1
gbif_raster <- rasterize(records.coords, gbif_raster, field = 1)
```

```
## convert cells with NA (no records) to 0
gbif_raster[is.na(gbif_raster)] <- 0

plot(gbif_raster, legend = FALSE)
```



As our area is rectangular we should not be too worried about setting our thresholds for upgraining, and so we can choose the “All_Sampled” option to maintain all data.

```
occupancy <- upgrain(gbif_raster,
                     scales = 2,
                     method = "All_Sampled")
```

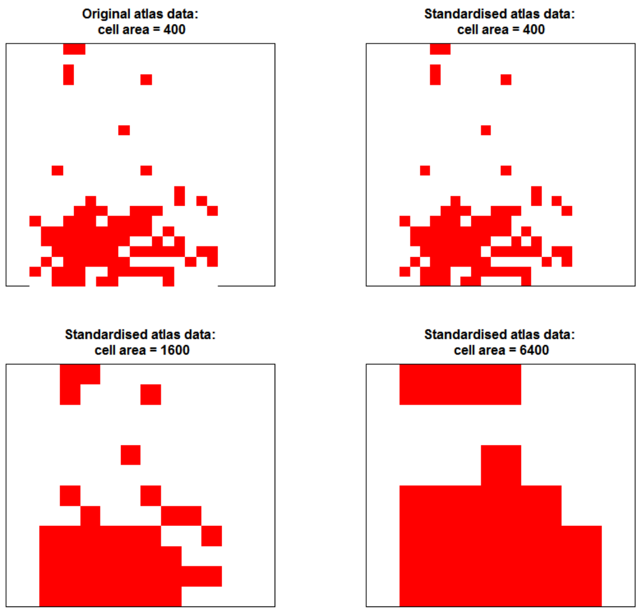
We can see there has not been much increase in extent after upgraining:

```
## The extent of the original atlas data
occupancy$occupancy.orig[1, 2]
```

```
[1] 163200
```

```
## The extent of the standardised atlas data
occupancy$extent.stand
```

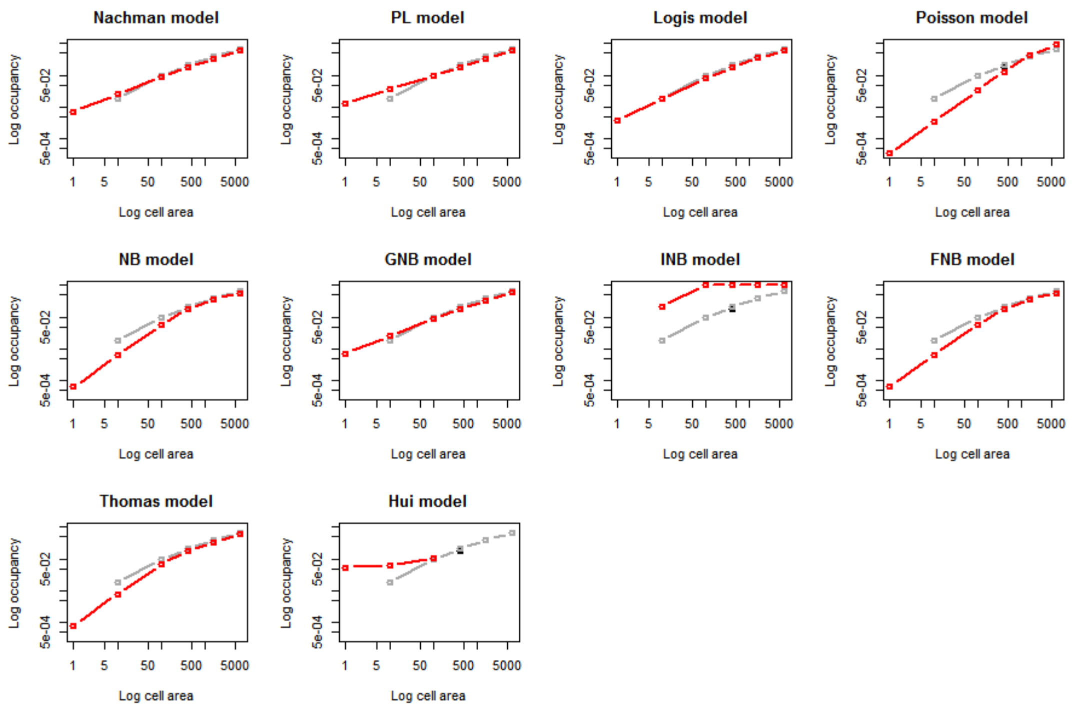
```
[1] 192000
```



Now we can run our ensemble downscaling models:

```
ensemble <- ensemble.downscale(occupancy,
                               models = "all",
                               new.areas = c(1, 10, 100, 400, 1600, 6400),
                               tolerance_mod = 1e-3)
```

```
Nachman model is running...  complete
PL model is running...  complete
Logis model is running...  complete
Poisson model is running...  complete
NB model is running...  complete
GNB model is running...  complete
INB model is running...  complete
FNB model is running...  complete
Thomas model is running...  complete
Hui model is running...  complete

Warning message:
In predict.downscale(object = mod, new.areas = new.areas, extent = extent,  :
  Predicted results may be innaccurate: one or more 0's predicted.
```



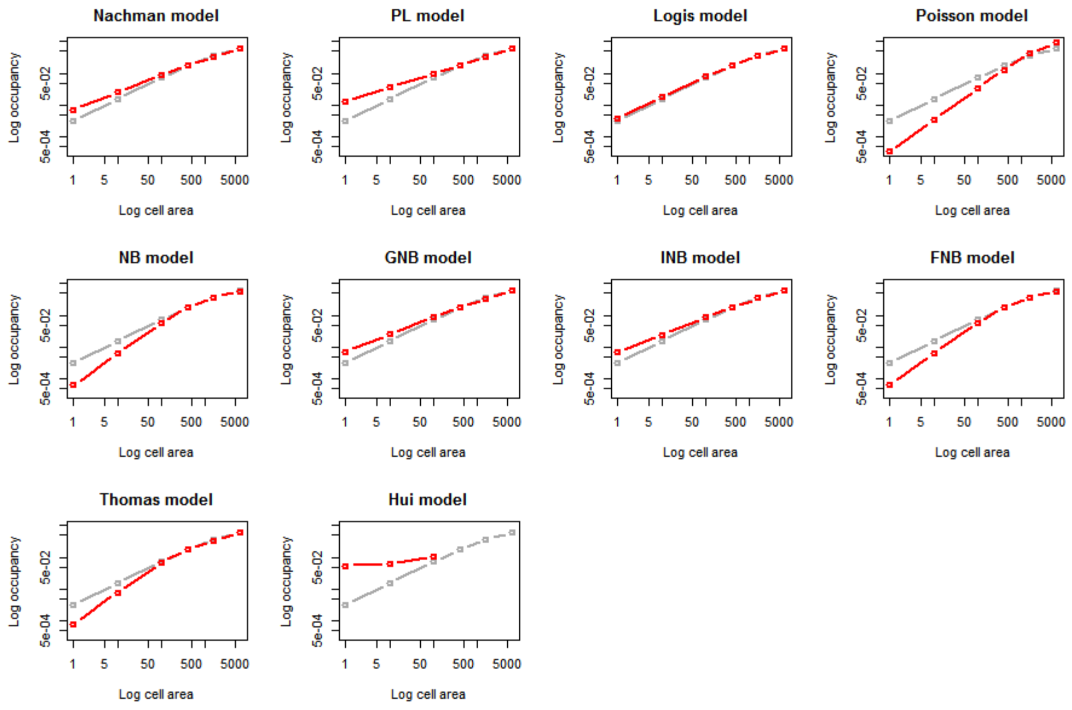
The INB model has not converged satisfactorily and thrown up a warning message (it has predicted a 0 at the finest grain size which we know to be impossible). This highlights the importance of visually inspecting the model fits. We can try tweaking it's starting parameters to see if we can get a better fit using the `starting_params` argument:

```
ensemble <- ensemble.downscale(occupancy,
                               models = "all",
                               new.areas = c(1, 10, 100, 400, 1600, 6400),
                               tolerance_mod = 1e-3,
                               starting_params = list(INB = list(C = 10,
                                                                 gamma = 0.01,
                                                                 b = 0.1)))
```

```
Nachman model is running...  complete
PL model is running...  complete
Logis model is running...  complete
Poisson model is running...  complete
NB model is running...  complete
GNB model is running...  complete
INB model is running...  complete
FNB model is running...  complete
Thomas model is running...  complete
Hui model is running...  complete
```

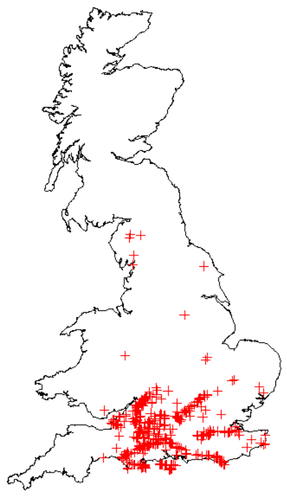
```
## And the predicted mean area of occupancies for each grain size
ensemble$AOO[, c("Cell.area", "Means")]
```

```
  Cell.area       Means
1         1    617.8954
2        10   2966.7477
3       100  13958.6322
4       400  32781.6381
5      1600  68197.5079
6      6400 117253.2752
```



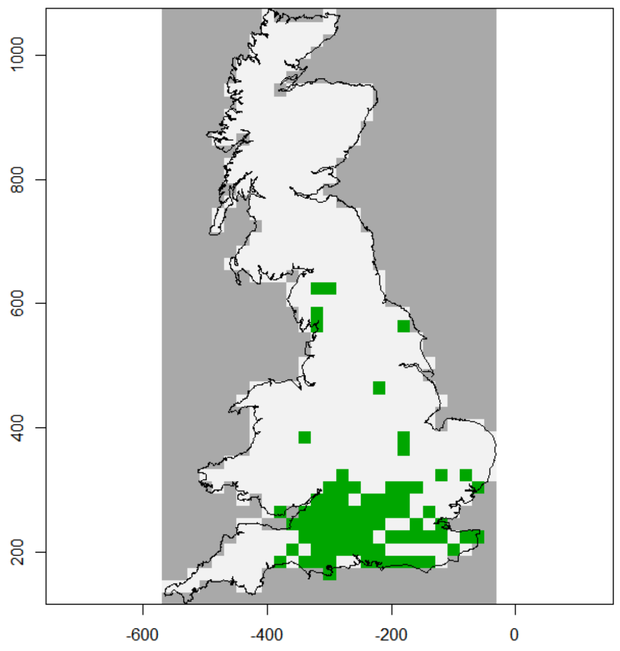
So far we have simply drawn a rectangle around our points, but perhaps we have a better idea of the possible range limits of the species. In our case, it is probably sensible to set the extent as mainland UK. A shapefile of the UK has been provided which we can load in.

```
uk <- system.file("extdata", "UK.shp", package = "downscale")
uk <- shapefile(uk)

## reproject to be the same coordinate system as our GBIF data (in km)
uk <- spTransform(uk,
                  CRS("+proj=lcc +lat_1=49.8333339 +lat_2=51.16666733333333
                       +lat_0=90 +lon_0=4.367486666666666 +x_0=150000.01256
                       +y_0=5400088.4378 +ellps=intl +units=km +no_defs"))

## plot our GBIF records on top of the UK polygon
plot(uk)
plot(records.coords, add = TRUE, col = "red)
```



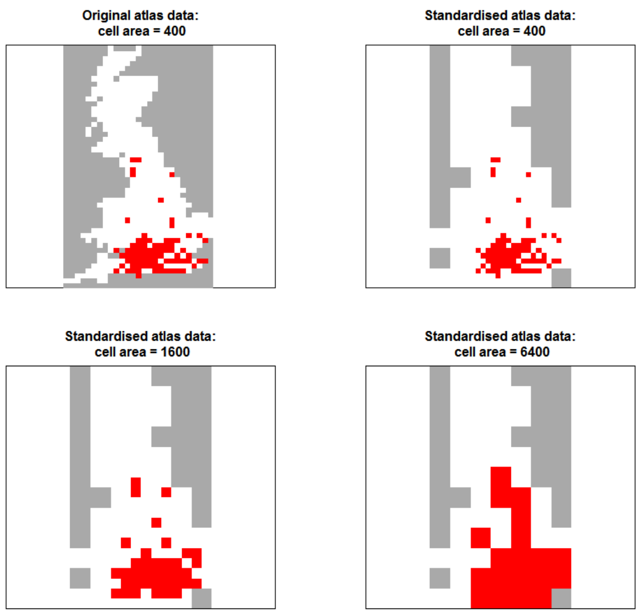
Now, we make our raster of presence-absence the same way as before, except this time we set the extent to be the same as that of the polygon. We then mask the raster file with the UK polygon so that any cells outside this polygon are assigned as NA (unsampled cells):

```
## create a blank raster with the same extent as the UK polygon
gbif_raster <- raster(ext = extent(uk),
                      res = cell.width)

## assign cells with presence records as 1
gbif_raster <- rasterize(records.coords, gbif_raster, field = 1)

## convert cells with NA (no records) to 0
gbif_raster[is.na(gbif_raster)] <- 0

## mask the raster to the UK polygon, so cells outside the polygon are NA
gbif_raster <- mask(gbif_raster, uk)

## plot the masked atlas raster and overlay with the UK polygon
plot(gbif_raster, legend = FALSE)
plot(uk, add = TRUE)
```



Now, we just upgrain and downscale as before:

```
occupancy <- upgrain(gbif_raster,
                     scales = 2,
                     method = "All_Sampled")
```

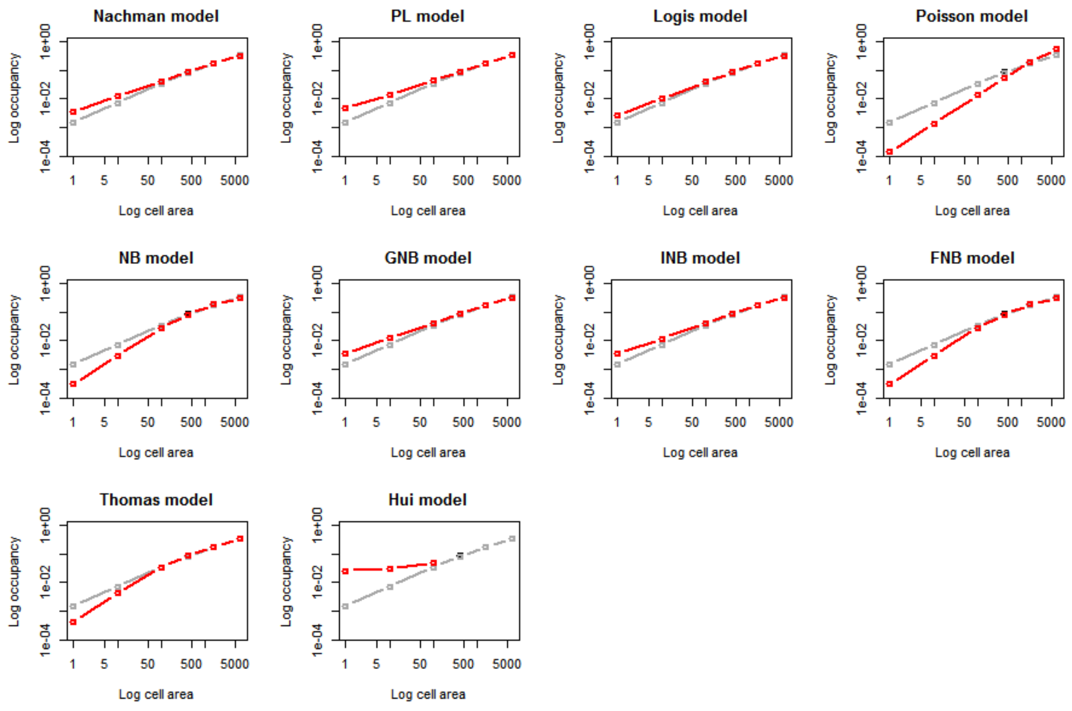


```
ensemble <- ensemble.downscale(occupancy,
                               models = "all",
                               new.areas = c(1, 10, 100, 400, 1600, 6400),
                               tolerance_mod = 1e-3,
                               starting_params = list(INB = list(C = 10,
                                                                  gamma = 0.01,
                                                                  b = 0.1)))
```

```
Nachman model is running...  complete
PL model is running...  complete
Logis model is running...  complete
Poisson model is running...  complete
NB model is running...  complete
GNB model is running...  complete
INB model is running...  complete
FNB model is running...  complete
Thomas model is running...  complete
Hui model is running...  complete
```

```
## And the predicted mean area of occupancies for each grain size
ensemble$AOO[, c("Cell.area", "Means")]
```

```
  Cell.area       Means
1         1    665.2036
2        10   3353.7223
3        25   6408.7945
4       100  17003.7297
5       400  43131.0685
6      1600  98390.5784
7      6400 183332.8997
```



If we want to compare predicted occupancy between the two methods we must compare the converted area of occupancies (AOO), not the proportion of occupancies as these are calculated from different extents. In this case the estimates for grain sizes of 1 km and 10 km from the bounded rectangle (AOO = 618 km^2 and 2967 km^2) are quite a bit larger than the estimates using the full mainland UK (AOO = 556 km^2 and 2635 km^2), most likely as a lot of cells in the atlas generated by the bounded rectangle would actually be sea. This highlights the care that is needed in selecting the bounding extent, the method and number of scales to upgrain, and the grain size we wish to predict occupancy for.

4 Bibliography

Azaele, S., S. J. Cornell, and W. E. Kunin. 2012. Downscaling species occupancy from coarse spatial scales. Ecological Applications 22:1004–14.

Barwell, L. J., S. Azaele, W. E. Kunin, and N. J. B. Isaac. 2014. Can coarse-grain patterns in insect atlas data predict local occupancy? Diversity and Distributions 20:895–907.

Hartley, S., and W. E. Kunin. 2003. Scale dependence of rarity, extincition risk, and conservation priority. Conservation Biology 17:1–12.

Hui, C., M. A. McGeoch, B. Reyers, P. C. le Roux, M. Greve, and S. L. Chown. 2009. Extrapolating population size from the occupancy-abundance relationship and the scaling pattern of occupancy. Ecological Applications 19:2038–2048.

Hui, C., M. A. McGeoch, and M. Warren. 2006. A spatially explicit approach to estimating species occupancy and spatial correlation. Journal of Animal Ecology 75:140–147.

IUCN. 2014. Guidelines for using the IUCN Red List categories and criteria.

Kunin, W. E. 1998. Extrapolating species abundance across spatial scales. Science 281: 1513–1515.