Introduction to the '`GRTS`' package (version 0.1.6-71)

ir. Thierry Onkelinx

November 10, 2014

Contents

Chapter 1

Short introduction to Generalized Random Tessellation Stratified sampling (GRTS)

Yet to be written... [Stevens and Olsen, 1999, 2003, 2004, Theobald et˜al., 2007]

Chapter 2

Our implementation of GRTS

For the sake of simplicity we assume that we have a square grid with 8 rows and 8 columns and we would like to take a spatially balanced sample of 19 grid cells. Each grid cell has a unique set of 2D coordinates: the row id and the column id (table˜2.1).

(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)

Table 2.1: 2D coordinates of a 8 x 8 matrix

2.1 Coding of 2D coordinates into 1D address

We start the convertion by splitting the matrix in 4 submatrices (level 1). We split it in half along the x-axis and in half along the y-axis. A single binary digit for each axis is sufficient to give each submatrix a unique 2D code (table 2.2).

Instead of a two digit binary (base 2) code, we can use a single base 4 code: a number from 0 to 3 (table 2.3).

The next step is to split each submatrix again in each four subsubmatrices (level 2). Like before we number them from 0 to 3 (table 2.4). Next we prepend this number to the code of the submatrices (table 2.5).

(1,0)	(1,0)	(1,0)	(1,0)	(1,1)	(1,1)	(1,1)	(1,1)
(1,0)	(1,0)	(1,0)	(1,0)	(1,1)	(1,1)	(1,1)	(1,1)
(1,0)	(1,0)	(1,0)	(1,0)	(1,1)	(1,1)	(1,1)	(1,1)
(1,0)	(1,0)	(1,0)	(1,0)	(1,1)	(1,1)	(1,1)	(1,1)
(0,0)	(0,0)	(0,0)	(0,0)	(0,1)	(0,1)	(0,1)	(0,1)
(0,0)	(0,0)	(0,0)	(0,0)	(0,1)	(0,1)	(0,1)	(0,1)
(0,0)	(0,0)	(0,0)	(0,0)	(0,1)	(0,1)	(0,1)	(0,1)
(0,0)	(0,0)	(0,0)	(0,0)	(0,1)	(0,1)	(0,1)	(0,1)

Table 2.2: Binary 2D codes for the first level of submatrices.

1	1	1	1	3	3	3	3
1	1	1	1	3	3	3	3
1	1	1	1	3	3	3	3
1	1	1	1	3	3	3	3
0	0	0	0	2	2	2	2
0	0	0	0	2	2	2	2
0	0	0	0	2	2	2	2
0	0	0	0	2	2	2	2

Table 2.3: Base 4 1D codes for the first level of submatrices.

We keep repeating this procedure until all submatrices contain only one gridcell. In this example we have to do it only once more, resulting in table 2.6 and 2.7. When the matrix is square and the number of rows is a power of two, the procedure will take the same number of steps for each submatrix.

1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3
0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3
0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2

Table 2.4: Base 4 1D codes for the second level of submatrices.

11	11	31	31	13	13	33	33
11	11	31	31	13	13	33	33
01	01	21	21	03	03	23	23
01	01	21	21	03	03	23	23
10	10	30	30	12	12	32	32
10	10	30	30	12	12	32	32
00	00	20	20	02	02	22	22
00	00	20	20	02	02	22	22

Table 2.5: Prepending the base 4 1D codes for the second level of submatrices to the base 4 1D codes for the first level of submatrices.

1	3	1	3	1	3	1	3
0	2	0	2	0	2	0	2
1	3	1	3	1	3	1	3
0	2	0	2	0	2	0	2
1	3	1	3	1	3	1	3
0	2	0	2	0	2	0	2
1	3	1	3	1	3	1	3
0	2	0	2	0	2	0	2

Table 2.6: Base 4 1D codes for the third level of submatrices.

111	311	131	331	113	313	133	333
011	211	031	231	013	213	033	233
101	301	121	321	103	303	123	323
001	201	021	221	003	203	023	223
110	310	130	330	112	312	132	332
010	210	030	230	012	212	032	232
100	300	120	320	102	302	122	322
000	200	020	220	002	202	022	222

Table 2.7: Combining the base 4 1D codes from all levels.

2.2 Randomisation

We randomise the above procedure at the point were we assign the number to the submatrices. Instead of assigning the numbers 0 to 3 in a systematic fashion, we do it at random. First we give an example for a 4 x 4 matrix. Table 2.8 and 2.9 indicate the randomised base 4 1D code for the level 1 and level 2 submatrices. Table 2.10 combines them into a unique code per grid cell. Table 2.11 gives the order of the grid cells based on their code in table 2.10. This order or ranking is what the `QuadratRanking()` function returns.

3	3	2	2
3	3	2	2
1	1	0	0
1	1	0	0

Table 2.8: Level 1 submatrices with randomised base 4 1D code.

2	1	3	1
3	0	0	2
2	3	3	2
1	0	0	1

Table 2.9: Level 2 submatrices with randomised base 4 1D code.

23	13	32	12
33	03	02	22
21	31	30	20
11	01	00	10

Table 2.10: Combined base 4 1D codes.

Tables˜2.12 to 2.16 give a complete example for a 8 x 8 matrix.

11	7	14	6
15	3	2	10
9	13	12	8
5	1	0	4

Table 2.11: Order of randomised base 4 1D code.

2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1

Table 2.12: Level 1 submatrices with randomised base 4 1D code.

2	2	0	0	2	2	3	3
2	2	0	0	2	2	3	3
1	1	3	3	1	1	0	0
1	1	3	3	1	1	0	0
0	0	3	3	0	0	1	1
0	0	3	3	0	0	1	1
1	1	2	2	3	3	2	2
1	1	2	2	3	3	2	2

Table 2.13: Level 2 submatrices with randomised base 4 1D code.

1	3	1	0	3	1	0	1
0	2	3	2	0	2	3	2
1	3	1	3	1	0	1	3
2	0	2	0	2	3	2	0
0	1	3	0	2	1	3	0
3	2	2	1	0	3	2	1
2	3	2	1	3	1	3	0
1	0	0	3	2	0	1	2

Table 2.14: Level 3 submatrices with randomised base 4 1D code.

122	322	102	002	323	123	033	133
022	222	302	202	023	223	333	233
112	312	132	332	113	013	103	303
212	012	232	032	213	313	203	003
000	100	330	030	201	101	311	011
300	200	230	130	001	301	211	111
210	310	220	120	331	131	321	021
110	010	020	320	231	031	121	221

Table 2.15: Combined base 4 1D codes.

26	58	18	2	59	27	15	31
10	42	50	34	11	43	63	47
22	54	30	62	23	7	19	51
38	6	46	14	39	55	35	3
0	16	60	12	33	17	53	5
48	32	44	28	1	49	37	21
36	52	40	24	61	29	57	9
20	4	8	56	45	13	25	41

Table 2.16: Order of randomised base 4 1D code.

2.3 Sampling

The procedure above generates a randomised and spatially balanced order of grid cells. Sampling n grid cells reduces to taking the first n grid cells along the randomised order. Table 2.17 is a sample of 19 grid cells from table 2.16.

X	X X	X	X
X	X	X	X
X X	X	X	X
X	X X	X	X

Table 2.17: A sample of 19 points for table 2.16

We replicate the sampling 1000 times to check whether a) each grid cell has the same probability of being selected and b) the sampling is spatially balanced.

The first assumption is checked in table 2.18. The probability of being selected is very similar for all grid cells and near to the expected probability. Note that for computational reasons we limited the number of replications to 1000. As the number of replications increases, the differences among grid cells will be smaller.

28.8%	32.2%	30.7%	31.1%	30.3%	30.9%	32.2%	27.6%
29.0%	27.9%	28.8%	28.0%	29.3%	29.8%	30.1%	29.4%
28.7%	32.4%	29.5%	31.7%	31.1%	28.3%	26.0%	29.7%
29.0%	29.1%	30.2%	28.3%	29.7%	30.2%	31.9%	30.1%
29.1%	31.6%	30.0%	29.5%	30.1%	29.2%	29.4%	29.0%
30.9%	27.5%	29.5%	30.7%	28.9%	28.7%	31.2%	29.3%
30.3%	29.9%	29.0%	29.0%	28.6%	28.7%	29.7%	30.0%
30.1%	27.5%	29.9%	31.3%	29.0%	29.9%	29.7%	30.8%

Table 2.18: Proportion of 1000 replications in which the grid cell is selected when sampling 19 grid cells using GRTS. The expected proportion is $\frac{19}{8^2} = 29.7\%$

The second assumption is checked for the level 1 submatrices in fig.˜2.3 and for the level 2 submatrices in fig.˜2.3. Each subplot is a histogram of the number of samples in each submatrix. Note that all the histogram are nearly identical. Since each submatrix represents a part of the grid, we can conclude that the GRTS sampling is spatially balanced.

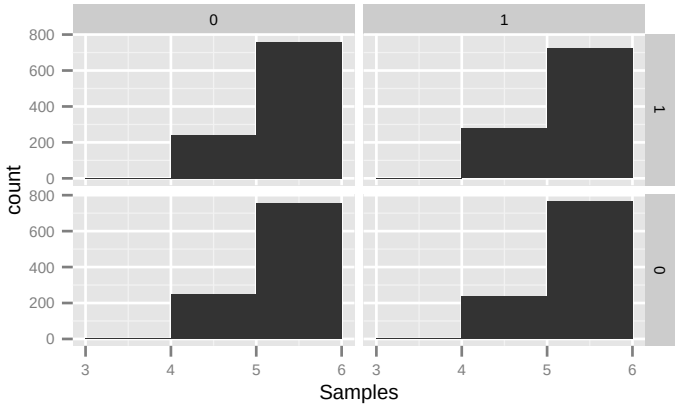


Figure 2.1: Histogram of the number of samples per level 1 submatrix.

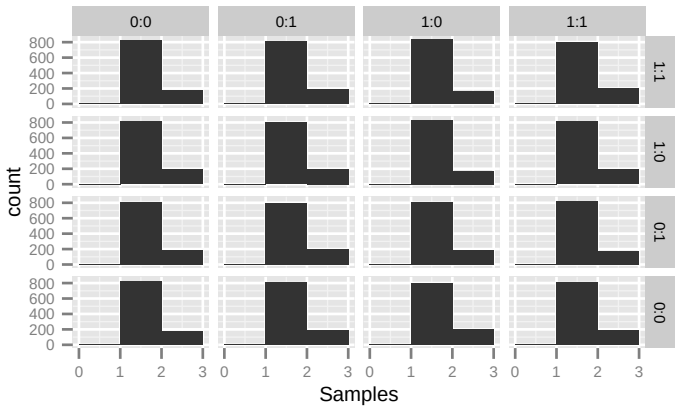


Figure 2.2: Histogram of the number of samples per level 2 submatrix.

Chapter 3

Using the package

The workhorse of the package is the `QuadratRanking()` function. This function expects a zero-filled, square matrix with the number of rows equal to a power of 2. However the function does not do any checking on those assumptions. That would be a computational burden since the function is called recursively. Therefore one should not call `QuadratRanking()` directly but use the global wrapper function `GRTS()`. This wrapper function handles more conviniently different input formats.

3.1 Calculating a GRTS randomisation for a square matrix

In case of a square matrix we just supply the number of rows to the `GRTS()` function. Note that if the number of rows is not a power of 2, then `QuadratCount()` is run with the next power of 2 as number of rows. Afterwards, the matrix is trimmed to contain the required number of rows.

```
> GRTS(8)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]   50   34   54   22   16   32   12   44
[2,]    2   18   38    6   48    0   60   28
[3,]   14   46   10   42   36   20   40    8
[4,]   62   30   26   58    4   52   24   56
[5,]   15   31   55    7   41    9   61   13
[6,]   47   63   23   39   57   25   29   45
[7,]   51   35   59   43   37    5   33   49
[8,]    3   19   11   27   53   21   17    1
```

```
> GRTS(7)
```

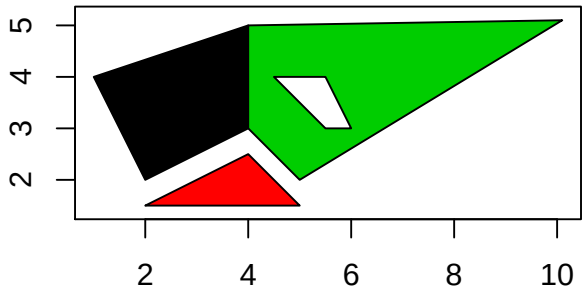
```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   26   13    5   40    4   27   46
```

```
[2,]    1   36   15   28   14   39   22
[3,]   47   34   43   31   18   42    0
[4,]   10   23    7   19    6   30   25
[5,]    8   20   24   48   45   33   41
[6,]   32   44   11   35    9   21   17
[7,]   37    2   29   16   38    3   12
```

3.2 Calculation a GRTS randomisation for polygons

Typically GIS-polygons are used to localise the study area. To accomodate this situation, `GRTS()` can handle objects of the class `SpatialPolygons`. Let's first create an object with a hypothetical study area consisting of 3 polygons: one island polygon and two adjacent polygons of which one contains a hole.

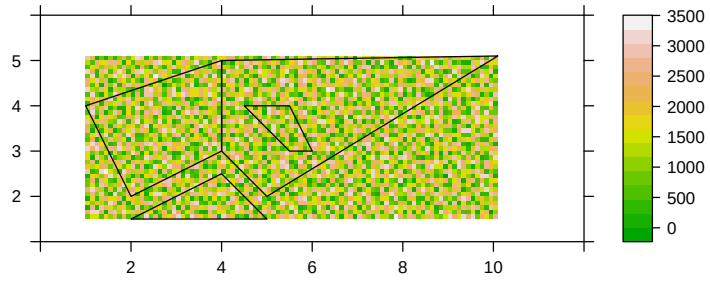
```
> #define a SpatialPolygons object
> Sr1 <- Polygon(cbind(c(2, 4, 4, 1,2), c(2, 3, 5, 4, 2)))
> Sr2 <- Polygon(cbind(c(5, 4, 2, 5), c(1.5, 2.5, 1.5, 1.5)))
> Sr3 <- Polygon(cbind(c(4, 4, 5, 10.1, 4), c(5, 3, 2, 5.1, 5)))
> Sr4 <- Polygon(cbind(c(4.5, 5.5, 6, 5.5, 4.5), c(4, 3, 3, 4, 4)), hole = TRUE)
> Srs1 <- Polygons(list(Sr1), "s1")
> Srs2 <- Polygons(list(Sr2), "s2")
> Srs3 <- Polygons(list(Sr3, Sr4), "s3/4")
> SpP <- SpatialPolygons(list(Srs1,Srs2,Srs3), 1:3)
> plot(SpP, col = 1:3, pbg="white", axes = TRUE)
```



When we pass a `SpatialPolygons` object to `GRTS()` we must specify the `cellsize` argument. This defines dimensions of a single grid cell and is in the

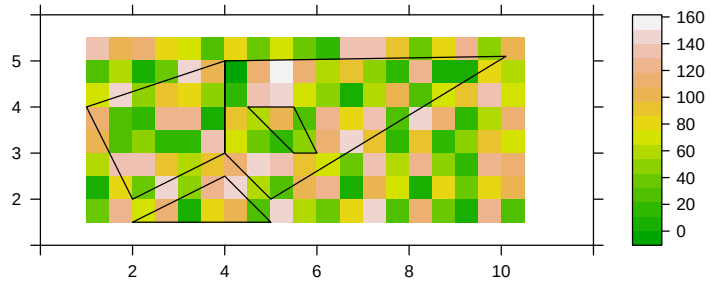
same units as the coordinates of the polygons. The variable `Ranking` from the GRTS output contains the randomised order of the grid cells.

```
> pls <- list("sp.polygons", SpP, col = "black", first = FALSE)
> output <- GRTS(SpP, cellsize = 0.1)
> limits <- apply(cbind(bbox(output), bbox(SpP)), 1, function(x){
+   range(pretty(x))
+ })
> spplot(output, sp.layout = list(pls), col.regions = terrain.colors(100),
+        scales = list(draw = TRUE), xlim = limits[, 1], ylim = limits[, 2])
```



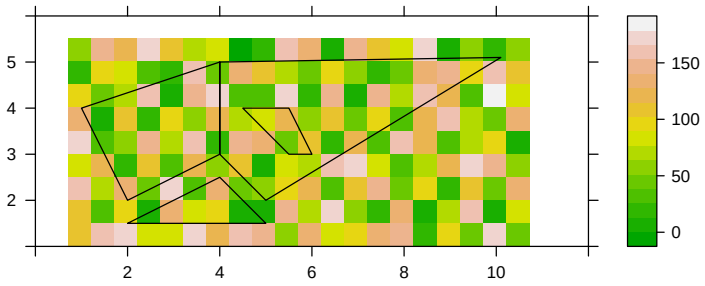
Changing the cellsize impacts the resolution of the grid.

```
> output <- GRTS(SpP, cellsize = 0.5)
> limits <- apply(cbind(bbox(output), bbox(SpP)), 1, function(x){
+   range(pretty(x))
+ })
> spplot(output, sp.layout = list(pls), col.regions = terrain.colors(100),
+        scales = list(draw = TRUE), xlim = limits[, 1], ylim = limits[, 2])
```



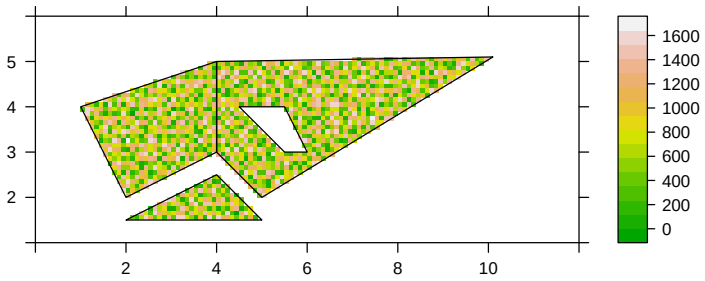
By default the grid starts at the south-west corner (minimum of both coordinates) of the bounding box of the polygon object. When we specify `RandomStart = TRUE`, this origin is shifted at random in both directions between `0` and `cellsize` units.

```
> output <- GRTS(SpP, cellsize = 0.5, RandomStart = TRUE)
> limits <- apply(cbind(bbox(output), bbox(SpP)), 1, function(x){
+   range(pretty(x))
+ })
> spplot(output, sp.layout = list(pls), col.regions = terrain.colors(100),
+        scales = list(draw = TRUE), xlim = limits[, 1], ylim = limits[, 2])
```



Another optional argument is `Subset = TRUE`. In this case the grid will be subsetted and only the grid cells who's centroid is located in one of the polygons are retained.

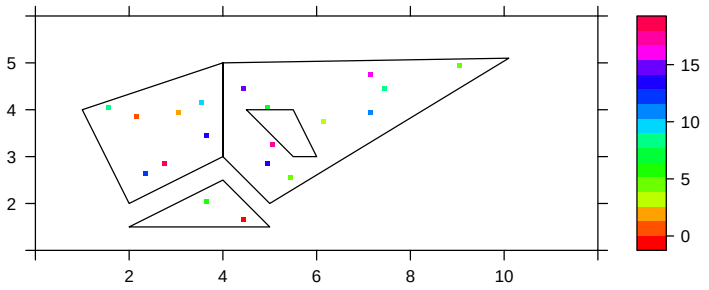
```
> output <- GRTS(SpP, cellsize = 0.1, Subset = TRUE)
> limits <- apply(cbind(bbox(output), bbox(SpP)), 1, function(x){
+   range(pretty(x))
+ })
> spplot(output, sp.layout = list(pls), scales = list(draw = TRUE),
+        col.regions = terrain.colors(100),
+        xlim = limits[, 1], ylim = limits[, 2])
```



Suppose we want a sample of 19 points. After the GRTS randomisation, we select the 19 grid cells with the lowest ranking.

```
> n <- 19
> #calculate the treshold value
```

```
> MaxRanking <- max(head(sort(output$Ranking), n))
> #do the selection
> Selection <- subset(output, Ranking <= MaxRanking)
> spplot(Selection, sp.layout = list(pls), scales = list(draw = TRUE),
+        col.regions = rainbow(n),
+        xlim = limits[, 1], ylim = limits[, 2])
```

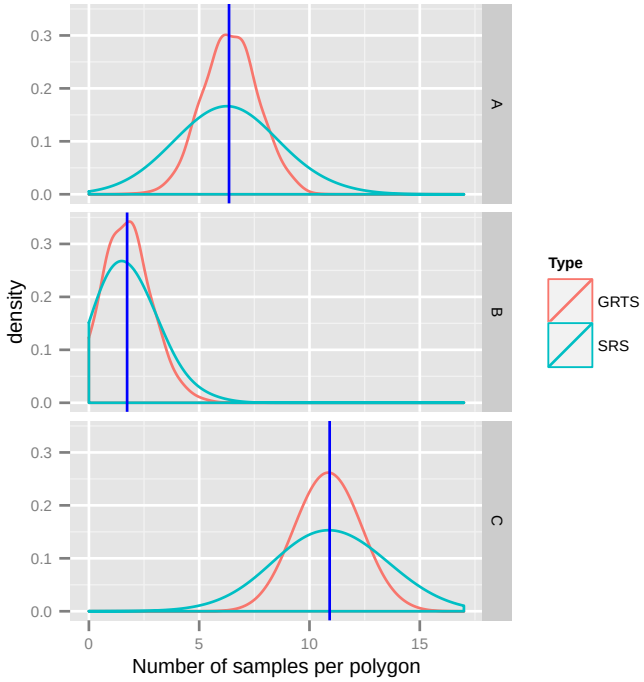


Let us test whether GRTS does a better job at generating a spatially balanced sample than a simple random sample (SRS). First we take a GRTS sample of 19 and count the number of points in each polygon. We repeat this several times and look at the distribution of the number of samples per polygon. We do the same thing for a simple random sample. The expected number of samples per polygon is the sample size multiplied with the relative area of the polygon.

The figure below shows the distribution with the number of samples for the three polygons. Polygon A is medium sided polygon without hole. Polygon B is the small triangular polygon. Polygon C is the large polygon with a hole. The expected number of samples is indicates with a blue line. The distribution from the GRTS sampling have a smaller variance than the simple random sampling, indicating that GRTS sampling is more spatially balanced.

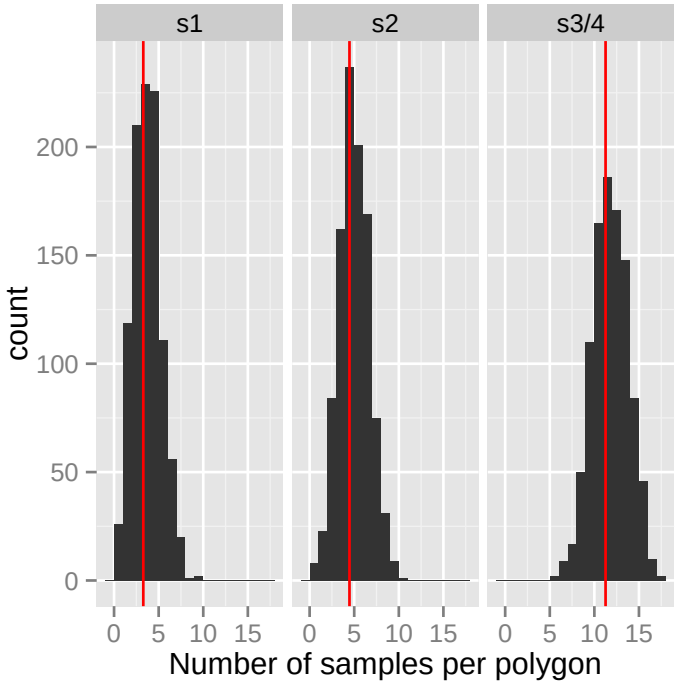
```
> testGRTS <- t(replicate(reps, {
+   #do the randomisation
+   output <- GRTS(SpP, cellsize = 0.1, Subset = TRUE, RandomStart = TRUE)
+   #calculate the treshold value
+   MaxRanking <- max(head(sort(output$Ranking), n))
+   #do the selection
+   Selection <- subset(output, Ranking <= MaxRanking)
+   #do the overlay
+   table(Polygon = factor(over(Selection, SpP), levels = 1:3,
+     labels = c("A", "B", "C")))
+ }))
> testGRTS <- melt(data = testGRTS)
> testGRTS$Type <- "GRTS"
> testSRS <- t(replicate(reps, {
+   Selection <- spsample(SpP, n = n, type = "random")
```

```
+   table(Polygon = factor(over(Selection, SpP), levels = 1:3,
+     labels = c("A", "B", "C")))
+ }))
> testSRS <- melt(data = testSRS)
> testSRS$Type <- "SRS"
> test <- rbind(testGRTS, testSRS)
> areas <- sapply(SpP@polygons, function(x){
+   tmp <- sapply(x@Polygons, function(y){
+     c(ifelse(y@hole, -1, 1), y@area)
+   })
+   sum(tmp[1, ] * tmp[2, ])
+ })
> reference <- data.frame(Polygon = factor(c("A", "B", "C")),
+   Expected = n * areas / sum(areas))
> ggplot(test) +
+   geom_density(aes(x = value, colour = Type), adjust = 2) +
+   geom_vline(data = reference, aes(xintercept = Expected), colour = "blue") +
+   facet_grid(Polygon ~ .) +
+   xlab("Number of samples per polygon")
```



3.3 Unequal probability sampling

```
> Weights <- data.frame(
+   ID = c("s1", "s2", "s3/4"),
+   Weight = c(1, 5, 2)
+ )
> rownames(Weights) <- Weights$ID
> Weights$Expected <- n * areas * Weights$Weight / sum(areas * Weights$Weight)
> SpP <- SpatialPolygonsDataFrame(SpP, data = Weights)
> SpP$ID <- factor(SpP$ID)
> test <- replicate(reps, {
+   GRTSorder <- GRTS(SpP, cellsize = 0.1, Subset = TRUE)
+   GRTSorder$Weight <- over(GRTSorder, SpP[, "Weight"])$Weight / (max(SpP$Weight))
+   GRTSorder$Weight <-
+     GRTSorder$Weight *
+     pmin(
+       1,
+       length(GRTSorder) / sum(GRTSorder$Weight)
+     )
+   GRTSups <-
+     GRTSorder[
+       rbinom(
+         length(GRTSorder),
+         size = 1,
+         prob = GRTSorder$Weight
+       ) == 1,
+       "Ranking"]
+   table(
+     over(
+       GRTSups[order(GRTSups$Ranking) <= n, ],
+       SpP[, "ID"]
+     )$ID,
+     useNA = 'ifany')
+ })
> test <- melt(t(test))
> colnames(test) <- c("Run", "ID", "Estimate")
> test <- merge(test, Weights)
> ggplot(test, aes(x = Estimate)) +
+   geom_histogram(binwidth = 1) +
+   geom_vline(aes(xintercept = Expected), colour = "red") +
+   xlab("Number of samples per polygon") +
+   facet_wrap(~ID)
```


s1
s2
s3/4
count
200
150
100
50
0
0
5
10
15
Number of samples per polygon

Bibliography

Don˜L Stevens and Anthony˜R Olsen. Spatially Restricted Surveys Over Time for Aquatic Resources. *Journal of Agricultural, Biological, and Environmental Statistics*, 4(4):415–428, 1999.

Don˜L. Stevens and Anthony˜R. Olsen. Variance estimation for spatially balanced samples of environmental resources. *Environmetrics*, 14(6):593–610, 2003.

Don˜L. Stevens and Anthony˜R. Olsen. Spatially balanced sampling of natural resources. *Journal Of The American Statistical Association*, 99(465):262–278, 2004. doi: 10.1198/016214504000000250.

David Theobald, Don Stevens, Denis White, N.˜Urquhart, Anthony Olsen, and John Norman. Using GIS to Generate Spatially Balanced Random Survey Designs for Natural Resource Applications. *Environmental Management*, 40 (1):134–146, jul 2007. doi: 10.1007/s00267-005-0199-x.