An Introduction to the Floating Grid Permutation Technique

Reinder Radersma

February 27, 2014

This document describes how to use the R-package `fgpt`. The Floating Grid Permutation Technique (FGPT) offers functions which can be used for spatially restricted permutation tests. First the permutation technique will be explained. Next this documents starts with explaining the use of lower level functions on examples which broaden the understanding of the methodology and show the functionality of the functions. Lastly higher level functions will be explained which can be used to easily perform spatially restricted permutation tests.

First install and load the `fgpt` package:

```
install.packages("fgpt", repos = "http://R-Forge.R-project.org")
```

```
## Installing package into '/home/socnet/R/x86_64-unknown-linux-gnu-library/3.0'
## (as 'lib' is unspecified)
```

```
##
## The downloaded source packages are in
##   '/tmp/RtmpEUkGG0/downloaded_packages'
```

```
library("fgpt")
```

Some insight in the premutation technique is required to use this package. The Floating Grid Permutation Technique is introduced in the manuscript which introduced the R-package (Radersma and Sheldon 2014). The next paragraphs will give a short overview of the method.

1 Theory

Permutation tests are important in many scientific fields (for instance ecology and evolution) as they can deal with small sample sizes and various forms of dependencies among observations. A common source of dependence is spatial autocorrelation. Accounting for spatial autocorrelation is often crucial in eco-evolutionary studies, because many ecological and evolutionary processes are spatially restricted, such as gene flow, dispersal, mate choice, inter- and intraspecific competition, mutualism and predation.

The FGPT is a spatially restricted permutation technique for point observations with known geographical locations. Within the randomization process, the probability an observation is assigned to any of the spatial locations is a negative function of the distance between its original and assigned location. The slope of this function depends on a preset parameter and by exploring its parameter space, non-random spatial processes can be both assessed and controlled for at multiple spatial scales.

In the FGPT a set of grid cell sizes is chosen, which represents the different spatial scales that will be examined. The null distribution of observations for a particular grid cell size is calculated by first projecting the grid on a map of the geographical locations (Fig 1a). Next the grid is randomly moved and rotated (Fig 1b). Within each grid cell observations are shuffled or when appropriate sampled with replacement (bootstrapped). The statistic appropriate for testing the hypotheses is calculated. The procedure is repeated a large number of times to produce a reference null distribution of the statistic (Fig 1c). For every

observation, and for every grid cell size, a negative near linear relationship between the distance between the original and assigned location and the probability of assignment is achieved, which simulates spatial autocorrelation (Fig 1d). To deal with group membership (e.g. observations were collected in multiple years) shuffling can be restricted to observations with the same group while the statistic is calculated over all observations at once.

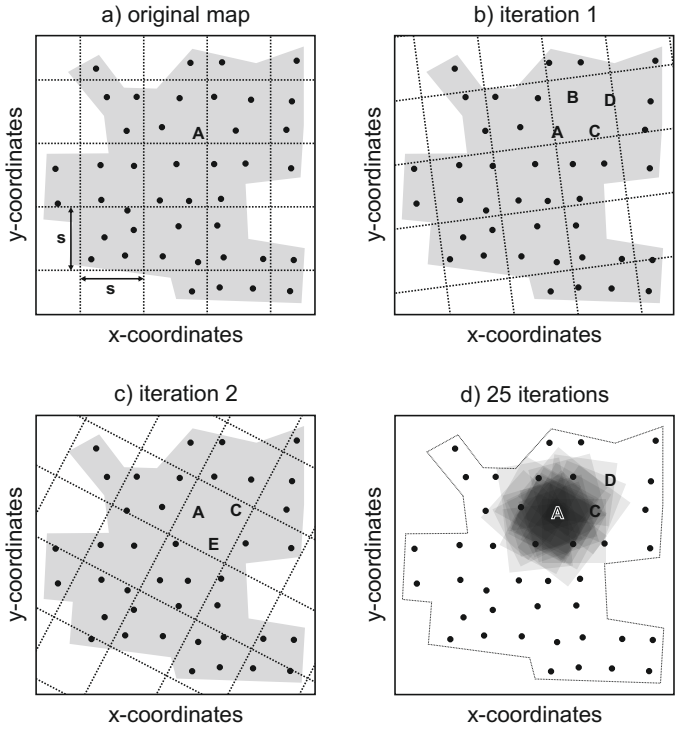


Figure 1: A graphical representation of the FGPT. The grey areas represent the study area and black dots and capital letters represent locations of observations. (a) A grid of size s is projected on the original map of the area with our focal observation marked as A. (b) At the first iteration the grid is randomly moved and rotated. Observations B, C and D share the same grid cell with A. Within each grid cell, all observations are randomized. (c) At the next iteration A shares a grid cell with only two other observations, C and E. Again all observations are randomized within grid cells, so A is randomized with C and E. (d) After many iterations (for graphical purposes we show only 25) we get a distribution for the probability an observation is assigned to any of the spatial locations. The probability that A is assigned to the location of observation C is higher than to the location of observation D. This method assumes probability of being assigned to a specific location is decreases with distance to the observed location.

2 Lower level functions

2.1 Exploring basic principles with a 1D example

I start with a small one dimensional example. Assume 10 observations, which are conveniently spaced out on a line with a spatial unit of one between all observations. So the x coordinates are in this case 1 to 10 and the y coordinates 0 for all observations.

```r
x.coor <- 1:10
y.coor <- rep(0, 10)

x.coor
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
y.coor
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

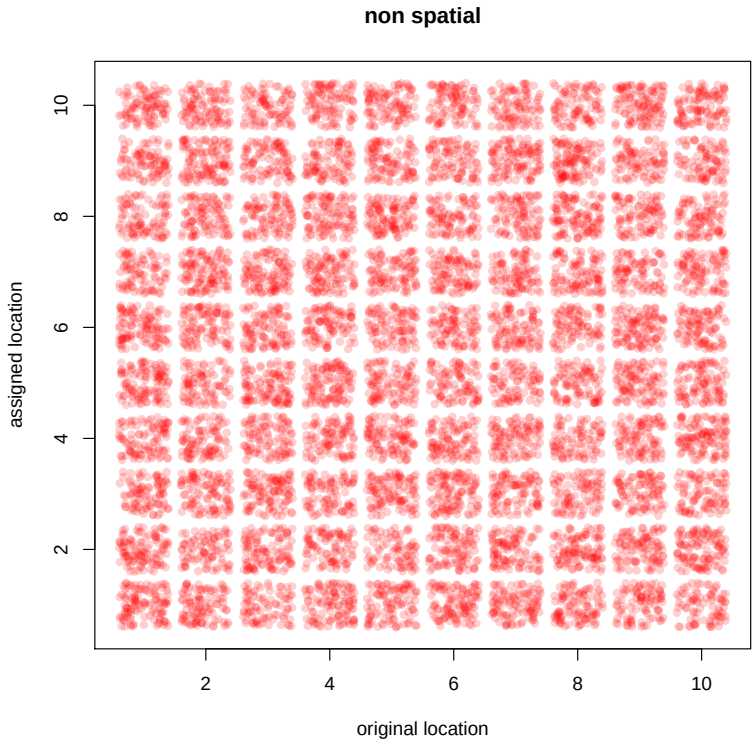
Normally those observations would not only have coordinates, but also so-called marks. Marks are for instance measurements which were taken at the geographical locations, such as traits of individual animals or plants. For convenience I take the x-coordinates as marks and will randomize the x coordinates. This way the randomization procedures are easily traceable. I first start with a non-spatial randomization. I produce 5 sets of randomized x coordinates with the R function `sample`, which is available as a base function in R.

```r
set.seed(23)
apply(array(x.coor, dim = c(10, 5)), 2, sample, size = 10)
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    6    9    6    7    8
##  [2,]    3    7    3    3    1
##  [3,]    9    4    2    9    3
##  [4,]    5    3   10    8    5
##  [5,]    7    6    9    2    7
##  [6,]    8    1    5    5    4
##  [7,]    4   10    7    6    9
##  [8,]   10    2    4   10    2
##  [9,]    2    8    1    4    6
## [10,]    1    5    8    1   10
```

A non-random randomization should not lead to a correlation between the original x coordinates and the locations they are assigned to by the randomization procedure. I can check this by running 1000 randomizations and plot the randomized sets to the x coordinates:

```r
set.seed(45)
rand.sets1 <- apply(array(x.coor, dim = c(10, 1000)), 2, sample, size = 10)
plot(as.vector(rand.sets1) + runif(10000, -0.4, 0.4), rep(1:10, 1000) + runif(10000, -0.4,
    0.4), col = "#FF000030", pch = 16, xlab = "original location", ylab = "assigned location",
    main = "non spatial")
```



With a correlation test I can check whether there is correlation between the original and assigned location.

```
cor.test(rep(1:10, 1000), as.vector(rand.sets1))
```

```
##
##  Pearson's product-moment correlation
##
## data:  rep(1:10, 1000) and as.vector(rand.sets1)
## t = 0.8218, df = 9998, p-value = 0.4112
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.01138  0.02781
## sample estimates:
##      cor
## 0.008218
```

So no correlation is present.

I now reproduce the previous randomizations while using the FGPT. LAgain I begin with 5 permutations to show how the results look like. To produce spatially restricted random datasets I use the function `fgperm` and introduce the spatial locations by combining the x and y coordinates into a two column matrix.

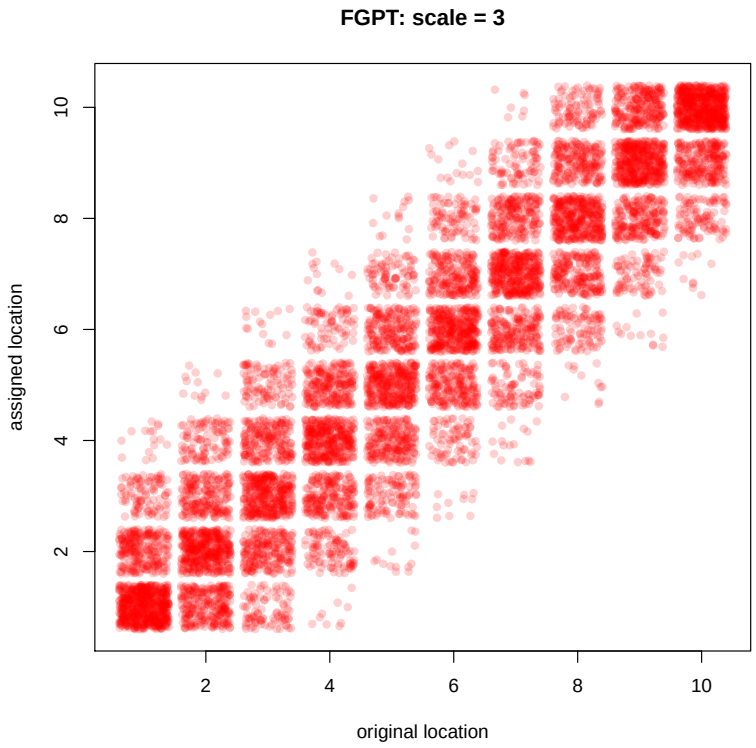
```
set.seed(72)
xy <- cbind(x.coor, y.coor)
fgperm(xy, z = x.coor, scale = 3, iter = 5, as.matrix = TRUE)
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1    3    3    1    2
##  [2,]    2    1    2    2    1
##  [3,]    4    4    1    3    3
```

```
##  [4,]    3    2    4    4    5
##  [5,]    5    7    5    5    4
##  [6,]    6    5    6    6    6
##  [7,]    7    6    8    7    8
##  [8,]    8    9    7    8    9
##  [9,]   10    8    9   10   10
## [10,]    9   10   10    9    7
```

Notice that lower values tend to stay at the top of the table and the higher values at the bottom. Now see whether the Floating Grid permutations result in a correlation between the original and assigned locations.

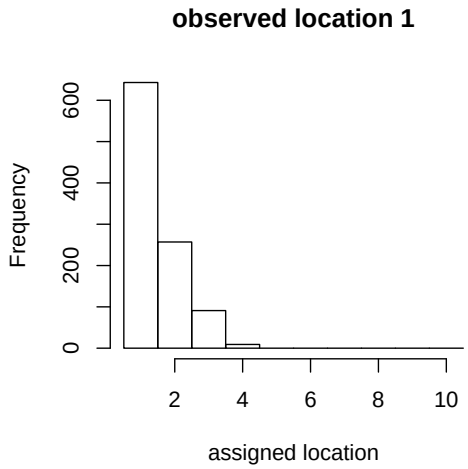
```
set.seed(24)
rand.sets2 <- fgperm(xy, z = x.coor, scale = 3, iter = 1000)
plot(unlist(rand.sets2) + runif(10000, -0.4, 0.4), rep(1:10, 1000) + runif(10000, -0.4, 0.4),
    col = "#FF000030", pch = 16, xlab = "original location", ylab = "assigned location", main = "FGPT: s
```




```
cor.test(rep(1:10, 1000), unlist(rand.sets2))

##
##  Pearson's product-moment correlation
##
## data:  rep(1:10, 1000) and unlist(rand.sets2)
## t = 254.9, df = 9998, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9283 0.9335
## sample estimates:
##    cor
## 0.9309
```

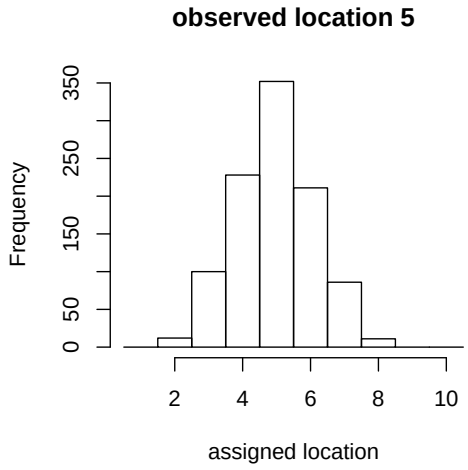
Well, there is a pretty strong correlation. Note that the observations of the original location 1 are only assigned to 1,2,3 and 4. I set `scale` to 3, which is the maximum distance observations are assigned to and happen to be the difference between x-coordinate 4 and 1. The FGPT should result in a negative near linear relationship between the distance between the observed and assigned location and the probability of assignment. I can check whether that is the case by plotting a histogram of for all observations that originate from 1.

```
hist(unlist(rand.sets2)[which(rep(1:10, 1000) == 1)], breaks = 0:10 + 0.5, xlab = "assigned location",
    main = "observed location 1")
```



Hmm, that doesn't look right. There are too many observations allocated to 1. Well, that is an edge effect. More on edge effects can be found in the paper that introduced FGPT (Radersma and Sheldon 2014). Now I try a observation in the middle, for instance 5.

```
hist(unlist(rand.sets2)[which(rep(1:10, 1000) == 5)], breaks = 0:10 + 0.5, xlab = "assigned location",
    main = "observed location 5")
```



That looks better. Now lets see what happens when I increase the scale. First to 6 and than to 1000.

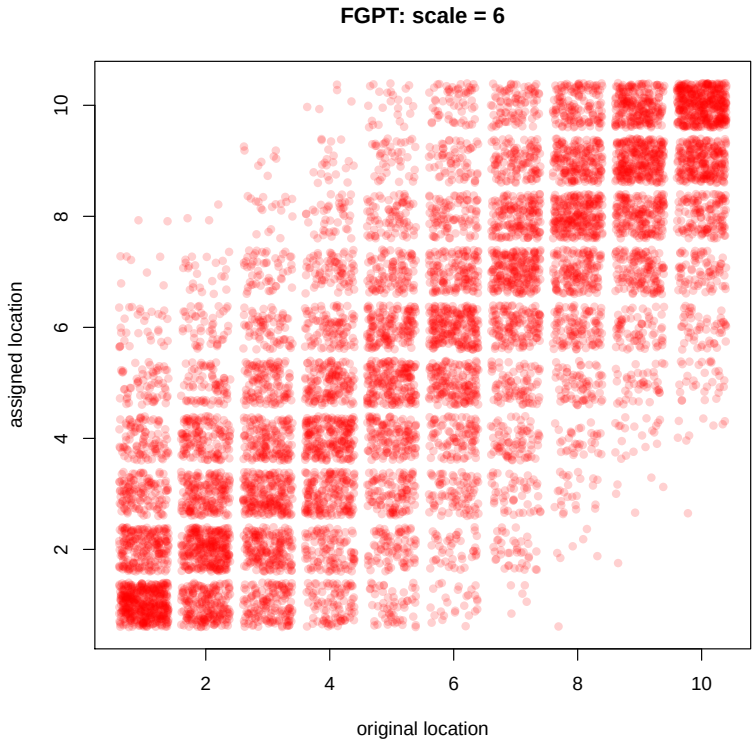
```
set.seed(7)
xy <- cbind(x.coor, y.coor)
fgperm(xy, z = x.coor, scale = 6, iter = 5, as.matrix = TRUE)
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1    3    2    1    3
##  [2,]    6    1    1    7    1
##  [3,]    2    2    3    5    2
##  [4,]    4    7    4    2    5
##  [5,]    7    8    6    3    4
##  [6,]    5    4    8    6   10
##  [7,]    3    6    7    4    7
##  [8,]    9    5    9    8    9
##  [9,]   10    9   10    9    8
## [10,]    8   10    5   10    6
```

Low values still tend to be at the top of the table and high values at the bottom. Again I plot a 1000 permutations again and check the correlation.

```
rand.sets3 <- fgperm(xy, z = x.coor, scale = 6, iter = 1000, add.obs = FALSE)
plot(unlist(rand.sets3) + runif(10000, -0.4, 0.4), rep(1:10, 1000) + runif(10000, -0.4, 0.4),
    col = "#FF000030", pch = 16, xlab = "original location", ylab = "assigned location", main = "FGPT: s

cor.test(rep(1:10, 1000), unlist(rand.sets3))
```

```
##
##  Pearson's product-moment correlation
##
## data:  rep(1:10, 1000) and unlist(rand.sets3)
## t = 119.1, df = 9998, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7576 0.7738
## sample estimates:
##    cor
## 0.7658
```



Still a correlation, but less steep. Note that the furthest location an observed value of 1 was assigned to is 8. This is because the near linear relationship between distance and assignment probability is tailing off to the grid cell size (i.e. `scale`) times the square root of 2. That is approxemately 8.5 in this case. So in principle values of 9 could occur, but with very small probability. Now I plot and check for a correlation when scale is 1000.

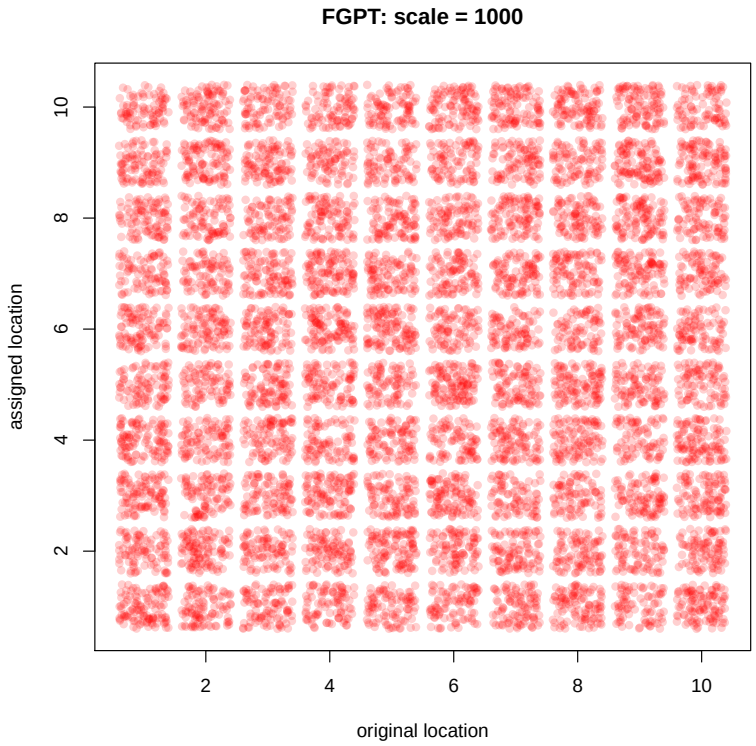
```
set.seed(6)
xy <- cbind(x.coor, y.coor)
fgperm(xy, z = x.coor, scale = 1000, iter = 5, as.matrix = TRUE)
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]   10    3    3    2    1
##  [2,]    6    8    6    7    5
##  [3,]    2    4    8    4    7
##  [4,]    1    1    4    5    8
##  [5,]    3    9    1    3    6
##  [6,]    5   10    9   10    2
##  [7,]    7    5    7    6    3
##  [8,]    9    2    2    8    4
##  [9,]    8    7   10    9   10
## [10,]    4    6    5    1    9
```

```
rand.sets4 <- fgperm(xy, z = x.coor, scale = 1000, iter = 1000, add.obs = FALSE)
plot(unlist(rand.sets4) + runif(10000, -0.4, 0.4), rep(1:10, 1000) + runif(10000, -0.4, 0.4),
    col = "#FF000030", pch = 16, xlab = "original location", ylab = "assigned location", main = "FGPT: s
cor.test(rep(1:10, 1000), unlist(rand.sets4))
```

```
##
##  Pearson's product-moment correlation
##
```

```
## data:  rep(1:10, 1000) and unlist(rand.sets4)
## t = 1.727, df = 9998, p-value = 0.08413
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.002328  0.036860
## sample estimates:
##     cor
## 0.01727
```



The results of `scale` 1000 look very similar to the results of the non spatial permutation. That makes sense, because the probability that all obervations will share the same grid cell is very large for grid cell sizes of 1000. A non spatial permutation test would yield similar results as a Floating Grid permutation test with infinite large grid cells.

2.2 Exploring functionalities with a 2D example

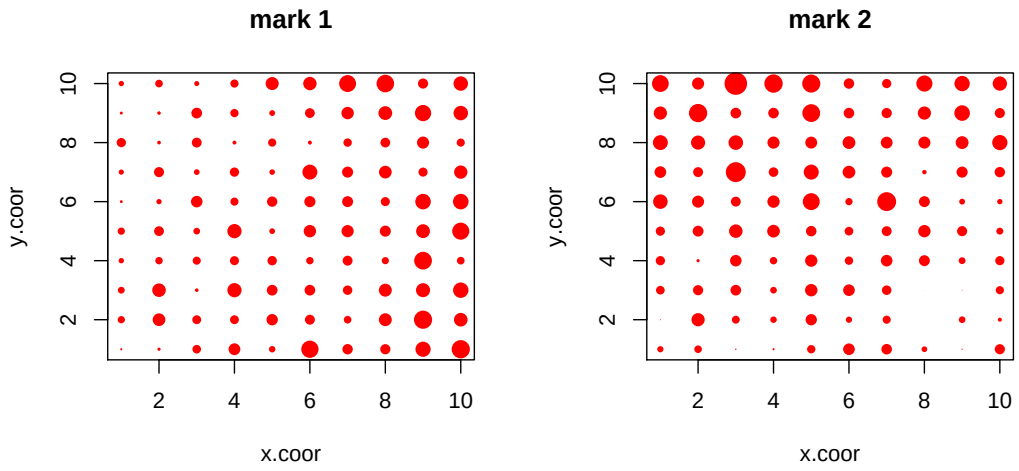
Noe I am going to produce a two dimensional example, which I use to explore the functionalities of the permutation function `fgperm` and start performing some permutation tests with the function `fgstat`.

For confenience I use observation locations with are situated in a grid.

```
x.coor <- rep(1:10, times = 10)
y.coor <- rep(1:10, each = 10)
```

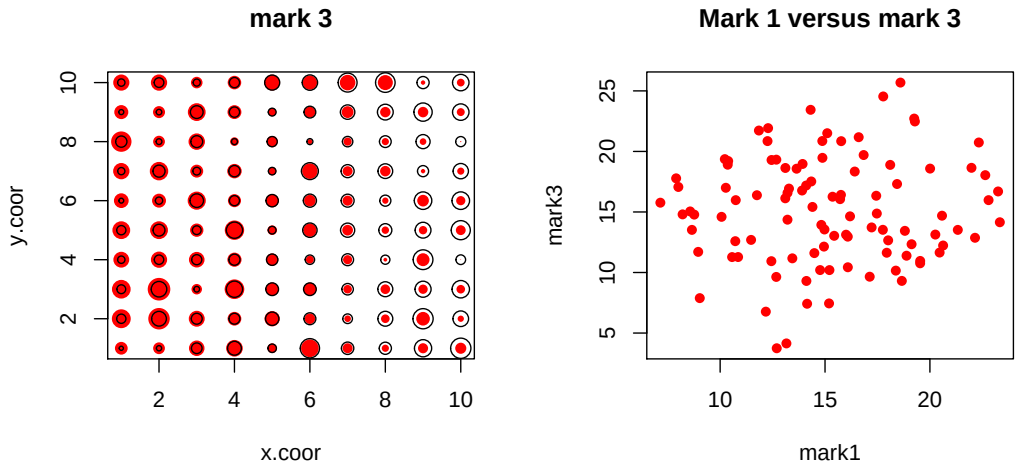
I am also going to produce three types of marks (i.e. traits) for every location and I am going to investigate whether those marks correlate with eachother. However, to show the potential of the FGPT I am going to introduce spatial autocorrelation in the marks. The first two marks are affected by space alone; the first mark shows a gradient from east to west and the second from south to north.

```r
set.seed(29)
mark1 <- x.coor + rnorm(100, 10, 3)
mark2 <- y.coor + rnorm(100, 10, 3)
plot(x.coor, y.coor, cex = (mark1 - 5)/10, pch = 16, col = "red", main = "mark 1")
plot(x.coor, y.coor, cex = (mark2 - 5)/10, pch = 16, col = "red", main = "mark 2")
```



The third mark will correlate with the first mark. Although the spatial cline of the third mark is opposite to the cline of the first mark, I let the third mark correlate positively to the first mark at fine spatial scales. So when the first mark is at a given location relative high compared to its neighbours, the third mark will be relative high as well. This is depicted in the following graphs. In the first graph the red bubbles represent the third mark and the open circles the first mark.

```r
set.seed(9)
mark3 <- mark1 + rnorm(100, 0, 0.5) + 11 - 2 * x.coor
plot(x.coor, y.coor, cex = (mark3 - 3)/10, pch = 16, col = "red", main = "mark 3")
points(x.coor, y.coor, cex = (mark1 - 3)/10)
plot(mark1, mark3, pch = 16, col = "red", main = "Mark 1 versus mark 3")
```



Time to start using the `fgperm` function. `fgperm` is a function to produce permuted datasets in which the permutations are spatially resticted. In the previous paragraph the effect of `scale` was already investigated,

so I will not discuss its functionality here. Minimally required input for `fgperm` is a two-column matrix containing the geographical coordinates of the observations and `scale`. If I leave the argument `z` empty the function will automatically permute the indices of the spatial locations, in this case 1 to 100. I could also enter for instance the first mark in `z` to permute mark 1 rather than the indices.

```
set.seed(435)
xy <- cbind(x.coor, y.coor)
rand1 <- fgperm(xy = xy, scale = 3)
rand2 <- fgperm(xy = xy, z = mark1, scale = 3)

rand1[[1]][1:10]

##  [1] 12 14  1  4  5 27  6 28 19  9

rand2[[1]][1:10]

##  [1] 17.77 11.47  8.21 13.93 13.64 15.21 15.44 22.33 20.26 23.25
```

Whether I should permute indices or marks depends on my data and hypothesis. This will be discussed in more detail when I discuss the function `fgstat`.

By default the `fgperm` uses the Fisher-Yates shuffle, which is a sampling technique without replacement. Other sampling techniques can be passed on to `fgperm`, by changing the argument `FUN`. Here I show how I can sample with replacement, but other functions can be used as well.

```
set.seed(5)
replace <- function(x) {
    x[sample.int(length(x), replace = TRUE)]
}
rand3 <- fgperm(xy = xy, scale = 3, FUN = replace)

rand3[[1]][1:10]

##  [1]  2  1  3 25 35  7  7 20 29 19
```

As you can see index 7 is sampled twice for the first 10 locations in the first permutation. I can also change the number of iterations (default is 999), the ratio between the sides of the grid cell sizes (read the original paper for the purpose of this argument), whether I would like to add the observed values as the first permuted dataset (advisable: read Ruxton and Neuhäuser (2013) for more information on good practices in permutation techniques) and whether I would like the output to be a matrix rather than a list. A list is required for the function `fgstat`, which I will introduced later. Before moving on to `fgstat` I first introduce the function `fgploc` which has some additional functionality to `fgperm` which might come in handy in certain cases.

In `fgploc` an additional function must be added which manipulates the observations within grid cells. I can for instance scale observations within grid cells. This will give information on the deviations from the mean while correcting for any non-random variation at broader spatial scales than the grid cell size. I will show a small example, but I will not go into details for when to use this function.

```
set.seed(7)
rand4 <- fgploc(xy = xy, scale = 3, marks = mark1, FUN.mani = scale)

rand4[[1]][1:10]

##  [1]  1.2555 -0.0755 -0.9466 -0.1603  0.6360  0.5842  2.1554 -0.6509 -1.0886 -0.2938
```

Now compare the output with the output of the function `fgperm`. When I take the average of all permutated values grouped by x coordinate for both `rand4` and `rand2` an interesting difference is visible. `fgploc` results in all values to be close to 0 and not increasing with the x coordinate, while `fgperm` results in means increasing with the x coordinate. `fgploc` removed the spatial cline, however if I would increase

the grid cell size this correction disappears. Note that `rand4` contains `NaNs`. This is because the function `scale` needs at least two observations per grid cell, but some observations will not share their grid cell with any other observation (this is particulary the case close to the edge).

```
round(tapply(rand4[[1]], x.coor, mean), 2)
```

```
##     1     2     3     4     5     6     7     8     9    10
## -0.16  0.28 -0.35  0.00  0.44 -0.40  0.36 -0.23 -0.07   NaN
```

```
round(tapply(rand4[[1]], x.coor, mean, na.rm = TRUE), 2)
```

```
##     1     2     3     4     5     6     7     8     9    10
## -0.16  0.28 -0.35  0.00  0.44 -0.40  0.36 -0.23 -0.07  0.13
```

```
round(tapply(rand2[[1]], x.coor, mean), 2)
```

```
##     1     2     3     4     5     6     7     8     9    10
## 11.51 12.29 12.68 14.03 14.36 15.70 16.99 16.27 18.94 18.66
```

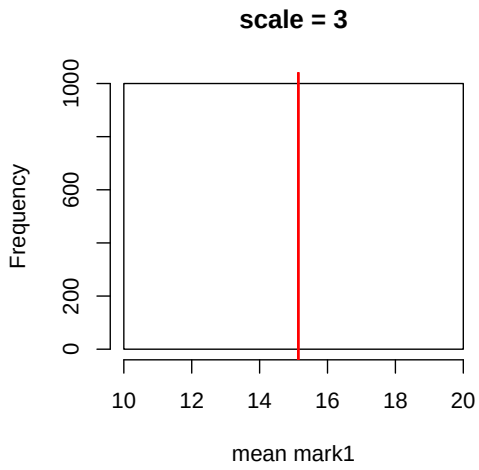
Now I am going to start analyzing the permuted datasets. I use the function `fgstat` for this. Arguments required for `fgstat` are permuted datasets which are the output of the `fgperm` or `fgploc` functions and the marks if they were not introduced in the `fgperm` function. By default `fgstat` explores the means of the permuted datasets, but other functions can be used to explore the data as well (e.g. `sd`, `var`, etc.). I first test whether the mean of mark 1 deviates from random. Before I do so, I will first add the original observations to the permuted datasets. This is not only convenient for the analyses, but also good practice (Ruxton and Neuhäuser, 2013).

```
set.seed(48)
rand1 <- fgperm(xy = xy, scale = 3, add.obs = TRUE)
rand2 <- fgperm(xy = xy, z = mark1, scale = 3, add.obs = TRUE)

calc1 <- fgstat(rand2)
hist(calc1, xlab = "mean mark1", main = "scale = 3")
abline(v = calc1[1], col = "red", lwd = 2)
calc1[1]
```

```
## observed
##    15.14
```

```
quantile(calc1, probs = c(0.025, 0.975))
```

```
##  2.5% 97.5%
## 15.14 15.14
```



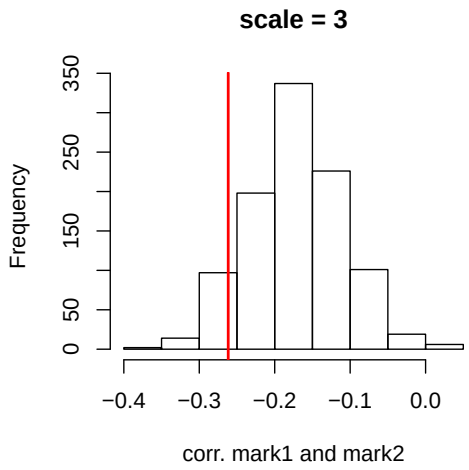
Well the mean is exactly the same for all permutations as well the observed mean. This does make sense, because a spatially restricted permutation test explores whether patterns deviate from random given the spatial distribution and not whether marks are non-randomly distributed in space. I can however test for correlations between all three marks. I start with mark 1 and mark 2. I combine mark 1 and mark 2 into a two column matrix and run the analyses. I need to add a function which produces the correlations as the argument `FUN`.

```
correlate <- function(x, y) {
    cor.test(x, y)$estimate
}

calc2 <- fgstat(rand1, cbind(mark1, mark2), FUN = correlate)
hist(calc2, xlab = "corr. mark1 and mark2", main = "scale = 3")
abline(v = calc2[1], col = "red", lwd = 2)
calc2[1]
```

```
## observed.cor
##      -0.2616
```

```
quantile(calc2, probs = c(0.025, 0.975))
```

```
##    2.5%   97.5%
## -0.2914 -0.0509
```



Mark 1 and mark 2 did not correlate significantly for scale 3. Now I try correlate mark 2 with mark 3 and mark 1 with mark 3.

```
calc3 <- fgstat(rand1, cbind(mark2, mark3), FUN = correlate)
calc4 <- fgstat(rand1, cbind(mark1, mark3), FUN = correlate)
hist(calc3, xlab = "corr. mark2 and mark3", main = "scale = 3")
abline(v = calc3[1], col = "red", lwd = 2)
calc3[1]
```

```
## observed.cor
##    -0.004045
```

```
quantile(calc3, probs = c(0.025, 0.975))
```

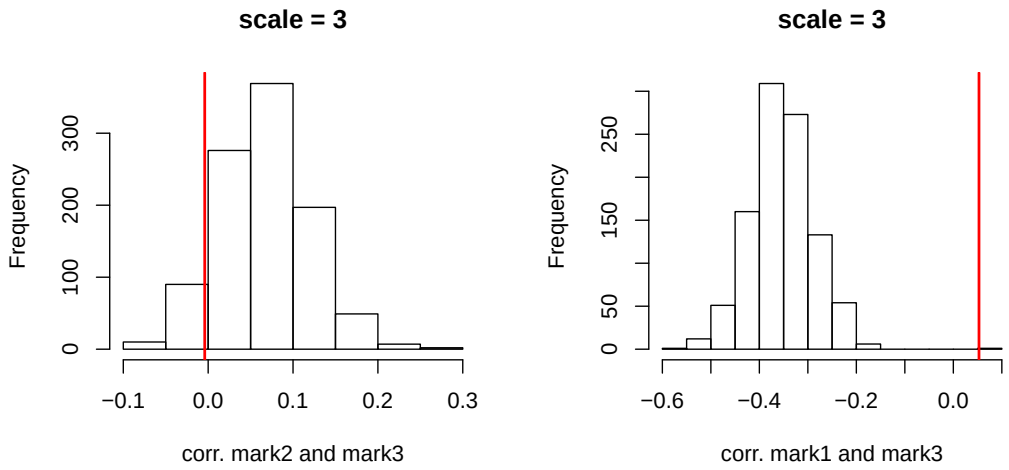
```
##     2.5%    97.5%
## -0.03445  0.17244
```

```
hist(calc4, xlab = "corr. mark1 and mark3", main = "scale = 3")
abline(v = calc4[1], col = "red", lwd = 2)
calc4[1]
```

```
## observed.cor
##       0.0529
```

```
quantile(calc4, probs = c(0.025, 0.975))
```

```
##    2.5%   97.5%
## -0.4798 -0.2280
```



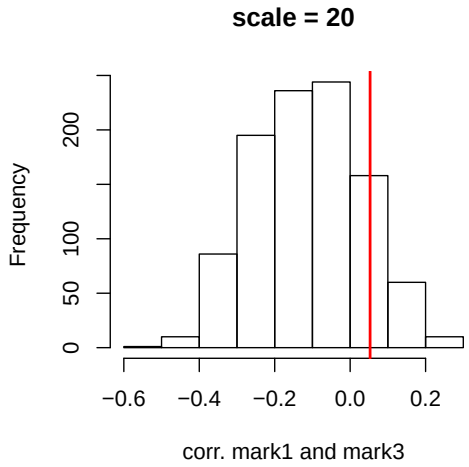
So I find a significant positive correlation between mark 1 and mark 3. This means the FGPT picks up signals of the fine-scale positive correlation between mark 1 and mark 3. No let's redo the analysis for scale 20. By using a large scale I test for broad-scale effects. Will I be able to pick up the negative correlation between mark 1 and mark 3 at broader scales?

```
rand5 <- fgperm(xy = xy, scale = 20, add.obs = TRUE)
calc5 <- fgstat(rand5, cbind(mark1, mark3), FUN = correlate)
hist(calc5, xlab = "corr. mark1 and mark3", main = "scale = 20")
abline(v = calc5[1], col = "red", lwd = 2)
calc5[1]
```

```
## observed.cor
##       0.0529
```

```
quantile(calc5, probs = c(0.025, 0.975))
```

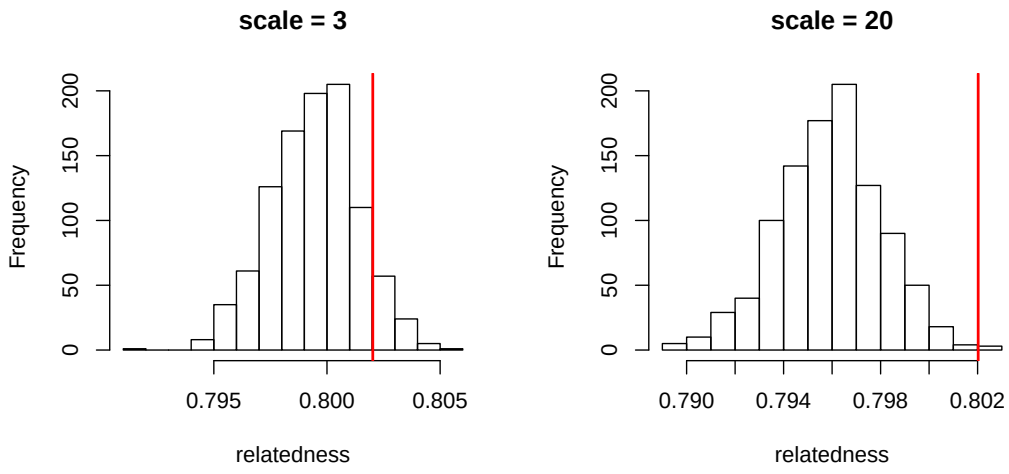
```
##    2.5%   97.5%
## -0.3697  0.1558
```



Well the correlation is still positive, but not significant anymore. I can't show there is a signifcant negative correlation at broader-scales. Whether I will be able to find such a pattern depends on many factors, such as the size of the study area and the relative strength of the fine-scale and broad-scale effects. Later in

this document I will discuss an example in which I will be able to show the presence of both negative and positive spatial effects at the same time, which occur at different scales. Other than correlations I could also investigate distances or relatedness between two observations. I could for instance look at genetic relatedness between mated pairs of individuals. To investigate distances I need a distance matrix for which one of the observations is represented by the columns and the other by the rows. I will produce a relatedness matrix and analyse it. Relatedness is in this case a function of the distance between two individuals.

```
set.seed(10)
spatial.dist <- as.matrix(dist(xy, diag = TRUE, upper = TRUE))
rel.mat <- array(rnorm(rep(1, 10000), 0.8 - (as.vector(spatial.dist)/1000), 0.02), dim = c(100,
    100))

calc6 <- fgstat(rand1, rel.mat)
hist(calc6, xlab = "relatedness", main = "scale = 3")
abline(v = calc6[1], col = "red", lwd = 2)
calc6[1]

## observed
##    0.802

quantile(calc6, probs = c(0.025, 0.975))

##   2.5%  97.5%
## 0.7955 0.8030

calc7 <- fgstat(rand5, rel.mat)
hist(calc7, xlab = "relatedness", main = "scale = 20")
abline(v = calc7[1], col = "red", lwd = 2)
calc7[1]

## observed
##    0.802

quantile(calc7, probs = c(0.025, 0.975))

##   2.5%  97.5%
## 0.7915 0.8000
```



So no significant relatedness for scale 3, but only for scale 20.

3 Higher level functions

The higher level functions of the FGPT make use of the function `fgperm` and `fgstat`, which I introduced in the previous chapter. The most important higher level function is `fgeasy`. Which is basically repeating the spatially restricted permutations with the FGPT for different grid cell sizes. There are also a summary and plotting functions. I will introduce the higher level functions by using the examples presented in the original manuscript (Radersma and Sheldon 2014).

3.1 The inbreeding avoidance example from the manuscript

I am going to use the inbreeding avoidance example from the manuscript (only the second population of this example). The code to produce the dataset is not provided here, but can be found in the supplementary materials of the paper. The dataset is however available in the package. First we load the data and explore it.

```
data("Pmajor")

str(xy)
```

```
##  num [1:200, 1:2] 46.6 150 300.6 390.8 513.9 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "locx" "locy"
```

```
str(rel2)
```

```
##  num [1:200, 1:200] 0.00407 0.00543 0 0 0 ...
```

In short, I have a population of 200 breeding pairs of birds. Because of limited dispersal there is isolation by distance (individuals are more closely related to individuals which live close by than further away). I want to test for inbreeding aviodance (the tendancy to avoid pairing with a closely related individual).

The data consists of a two column table containing the geographical locations of all breeding pairs (`xy`) and a matrix with the relateness between all males and females (`rel2`). I analyze the data with the function `fgeasy` and use `summary` and `plot` to explore the results. I run only 99 iterations to save some time, but 9999 would be advisable.

```
set.seed(78)
fg1b <- fgeasy(xy = xy, marks = rel2, iter = 99)
```

```
##
## ====================================
##  Floating Grid Permutation Technique
## ====================================
##
## Progress bar for spatially restricted permutations
## 0% |-----------------------------------------------| 100%
##    ================================================
##
## Non-spatial permutation test is now running.......
##
##
## The analysis took 13.51 secs
```

```
summary(fg1b)
```

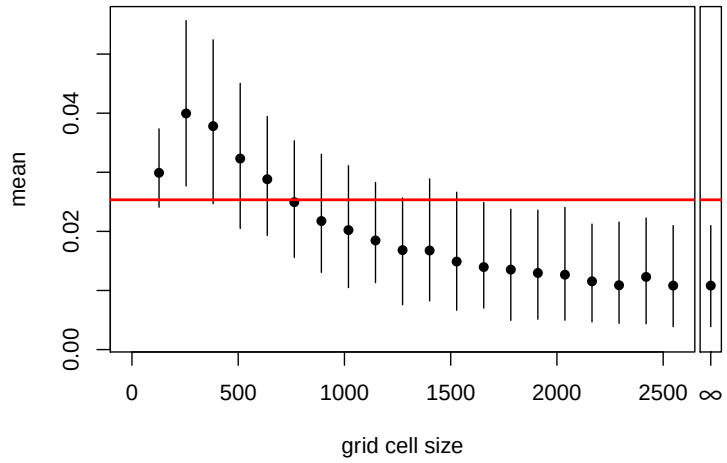
```
##
```

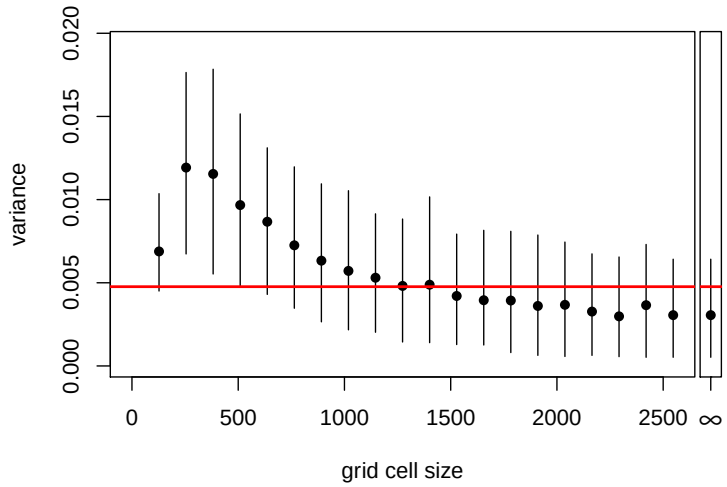
```
## ====================================
##  Floating Grid Permutation Technique
## ====================================
##
## Number of scales: 21
## Number of iterations per scale: 99
##
## -------------------
##  Results for mean:
## -------------------
##
##  cell_size observed    mean lower_limit upper_limit P_value % failures
##      127.4  0.02534 0.02991    0.024086     0.03734    0.18          0
##      254.7  0.02534 0.03995    0.027685     0.05566    0.04          0
##      382.1  0.02534 0.03781    0.024695     0.05241    0.10          0
##      509.4  0.02534 0.03232    0.020502     0.04505    0.32          0
##      636.8  0.02534 0.02882    0.019331     0.03946    0.50          0
##      764.1  0.02534 0.02495    0.015593     0.03533    0.90          0
##      891.5  0.02534 0.02175    0.013065     0.03306    0.56          0
##     1018.8  0.02534 0.02022    0.010483     0.03113    0.36          0
##     1146.2  0.02534 0.01847    0.011329     0.02828    0.16          0
##     1273.5  0.02534 0.01683    0.007583     0.02567    0.12          0
##     1400.9  0.02534 0.01676    0.008229     0.02888    0.22          0
##     1528.2  0.02534 0.01489    0.006649     0.02663    0.08          0
##     1655.6  0.02534 0.01396    0.007023     0.02489    0.04          0
##     1782.9  0.02534 0.01353    0.004947     0.02375    0.02          0
##     1910.3  0.02534 0.01296    0.005133     0.02360    0.04          0
##     2037.6  0.02534 0.01267    0.004966     0.02405    0.06          0
##     2165.0  0.02534 0.01154    0.004688     0.02123    0.02          0
##     2292.3  0.02534 0.01089    0.004446     0.02156    0.02          0
##     2419.7  0.02534 0.01230    0.004356     0.02226    0.02          0
##     2547.0  0.02534 0.01082    0.003872     0.02098    0.02          0
##        Inf  0.02534 0.01082    0.003872     0.02098    0.02          0
##
##  Results for variance:
## ----------------------
##
##  cell_size observed     mean lower_limit upper_limit P_value % failures
##      127.4 0.004765 0.006884   0.0045056    0.010358    0.10          0
##      254.7 0.004765 0.011926   0.0067336    0.017643    0.02          0
##      382.1 0.004765 0.011544   0.0055292    0.017838    0.04          0
##      509.4 0.004765 0.009672   0.0047906    0.015151    0.06          0
##      636.8 0.004765 0.008671   0.0043022    0.013113    0.12          0
##      764.1 0.004765 0.007253   0.0034689    0.011969    0.24          0
##      891.5 0.004765 0.006329   0.0026516    0.010948    0.58          0
##     1018.8 0.004765 0.005714   0.0021715    0.010535    0.72          0
##     1146.2 0.004765 0.005304   0.0020267    0.009144    0.84          0
##     1273.5 0.004765 0.004810   0.0014410    0.008833    0.94          0
##     1400.9 0.004765 0.004884   0.0013992    0.010163    0.92          0
##     1528.2 0.004765 0.004206   0.0012861    0.007926    0.70          0
##     1655.6 0.004765 0.003950   0.0012594    0.008153    0.58          0
##     1782.9 0.004765 0.003933   0.0008034    0.008092    0.68          0
##     1910.3 0.004765 0.003607   0.0006361    0.007870    0.40          0
##     2037.6 0.004765 0.003674   0.0005697    0.007447    0.64          0
```

```
##     2165.0 0.004765 0.003269   0.0006349    0.006736    0.46          0
##     2292.3 0.004765 0.002982   0.0005590    0.006550    0.26          0
##     2419.7 0.004765 0.003653   0.0005203    0.007304    0.56          0
##     2547.0 0.004765 0.003055   0.0005208    0.006417    0.32          0
##        Inf 0.004765 0.003055   0.0005208    0.006417    0.32          0
```

Both the mean and the variance are explored by `fgeasy`. The method automatically selects 20 grid cell sizes and also produces a non-spatial permutation test (`Inf`). The output is rather self-explainatory, except maybe the `% of failure`. When there are missing values in the observed dataset, the Floating Grid Method will not always be able to calculate the statistics. This will be particulary the case for very small grid cell sizes. The percentage of failing permutations is printed. Now plot the results.

```
plot(fg1b)
```





In this case the means are of most interest. There is inbreeding avoidance in the population, because at the finer spatial scales individuals are less related than expected. WI can also choose the grid cell size myself, to for instance focus on a certain spatial scale. Here I explore fine spatial scales.

```
set.seed(11)
fg1c <- fgeasy(xy = xy, marks = rel2, scale.seq = c(1, 100, 200, 300, 400, 500, 800), iter = 99)
```

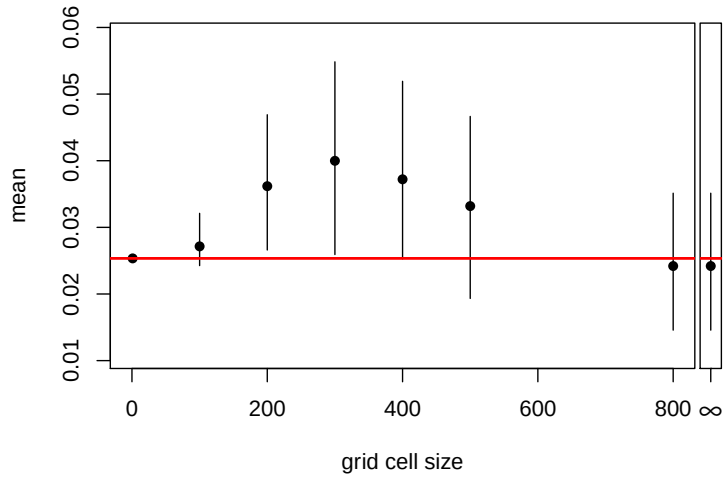
```
##
```

```
## ====================================
##  Floating Grid Permutation Technique
## ====================================
##
## Progress bar for spatially restricted permutations
## 0% |-------------------------------------------------| 100%
##    ================================================
##
## Non-spatial permutation test is now running.......
##
##
## The analysis took 5.86 secs
```

```
summary(fg1c)
```

```
##
## ====================================
##  Floating Grid Permutation Technique
## ====================================
##
## Number of scales: 8
## Number of iterations per scale: 99
##
## -------------------
##  Results for mean:
## -------------------
##
##  cell_size observed    mean lower_limit upper_limit P_value % failures
##          1  0.02534 0.02534     0.02534     0.02534    1.00          0
##        100  0.02534 0.02715     0.02426     0.03210    0.46          0
##        200  0.02534 0.03618     0.02660     0.04690    0.04          0
##        300  0.02534 0.03999     0.02592     0.05484    0.04          0
##        400  0.02534 0.03721     0.02526     0.05192    0.08          0
##        500  0.02534 0.03319     0.01933     0.04665    0.28          0
##        800  0.02534 0.02420     0.01457     0.03513    0.78          0
##        Inf  0.02534 0.02420     0.01457     0.03513    0.78          0
##
##  Results for variance:
## ----------------------
##
##  cell_size observed     mean lower_limit upper_limit P_value % failures
##          1 0.004765 0.004765    0.004765    0.004765    1.00          0
##        100 0.004765 0.005565    0.004557    0.007600    0.42          0
##        200 0.004765 0.010230    0.005511    0.015272    0.02          0
##        300 0.004765 0.012357    0.006560    0.018846    0.04          0
##        400 0.004765 0.011513    0.006090    0.018245    0.02          0
##        500 0.004765 0.009904    0.004411    0.016046    0.10          0
##        800 0.004765 0.006931    0.002652    0.012007    0.30          0
##        Inf 0.004765 0.006931    0.002652    0.012007    0.30          0
```

```
plot(fg1c, plane = 1)
```



Interesting to see is that the permuted data for `scale = 1` does not have any error bars. This is also visible in the summary table. This is because all permutations are similar to the observed data. The minimum distance between two geographical locations is much larger than 1, so the permutation always result in the assigning the observed pairs together.

3.2 The assortative pairing example from the manuscript

Now I move on to the next example to show some other functionalities. First I explore the dataset.

```
data("Gpulex")

str(Gp.xy)

##  num [1:200, 1:2] 0.257 2.956 0.516 8.505 0.4 ...

str(f.pheno)

##  num [1:200] 10.32 12.34 9.37 13.53 9.38 ...

str(m.pheno)

##  num [1:200] 16.1 16.3 18 20.9 15.9 ...
```

This dataset is a simulation of a population of freshwater Crustaceans living in a river section. There is spatial segregation for body size in the population due to spatial variation in water current. I will test here whether there is only habitat segregation or also size assortative pairing.

The FGPT can deal with missing values. In fact it can deal with distances or correlations between two traits, even if the traits never share a location. The function will, in that case, not able to calculate the statistics for the observed values. Best to analyse those cases with the lower level functions. First I remove a few observations to show that the method can deal with missing values. I remove 5 observation for males and 5 for females.

```
set.seed(9)
remove.f <- sample(200, 5)
remove.m <- sample(200, 5)

remove.f

## [1] 45  5 42 43 87
```

```
remove.m
```

```
## [1]  27  78  74 132 195
```

```
f.pheno[remove.f] <- NA
m.pheno[remove.m] <- NA
```

Now I run the analysis.

```
set.seed(456)

fg2a <- fgeasy(xy = Gp.xy, marks = cbind(f.pheno, m.pheno), correlate = "pearson", iter = 99)
```

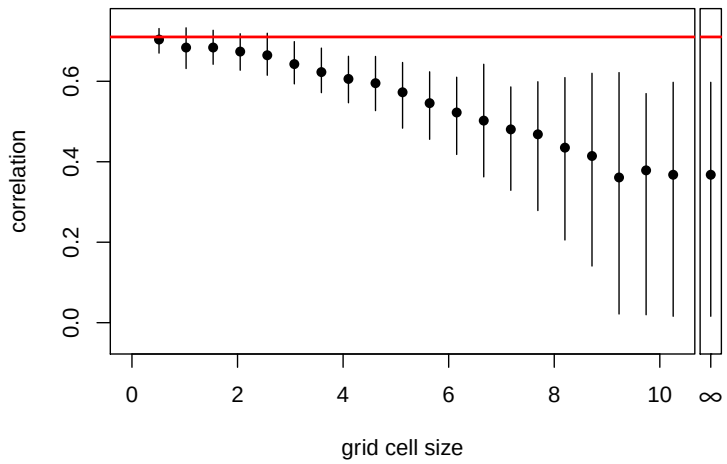
```
##
## ======================================
##  Floating Grid Permutation Technique
## ======================================
##
## Progress bar for spatially restricted permutations
## 0% |-------------------------------------------------| 100%
##    ================================================
##
## Non-spatial permutation test is now running.......
##
##
## The analysis took 12.32 secs
```

```
summary(fg2a)
```

```
##
## ======================================
##  Floating Grid Permutation Technique
## ======================================
##
## Number of scales: 21
## Number of iterations per scale: 99
##
## -------------------------
##  Results for pearson correlation:
## -------------------------
##
##  cell_size observed   mean lower_limit upper_limit P_value % failures
##     0.5127   0.7102 0.7038     0.67003      0.7311    0.68          0
##     1.0253   0.7102 0.6839     0.63169      0.7329    0.24          0
##     1.5380   0.7102 0.6839     0.64226      0.7268    0.22          0
##     2.0506   0.7102 0.6739     0.62748      0.7182    0.14          0
##     2.5633   0.7102 0.6647     0.61500      0.7194    0.10          0
##     3.0759   0.7102 0.6427     0.59357      0.6986    0.02          0
##     3.5886   0.7102 0.6229     0.57174      0.6825    0.02          0
##     4.1012   0.7102 0.6059     0.54661      0.6622    0.02          0
##     4.6139   0.7102 0.5953     0.52713      0.6617    0.02          0
##     5.1266   0.7102 0.5727     0.48347      0.6467    0.02          0
##     5.6392   0.7102 0.5456     0.45577      0.6235    0.02          0
##     6.1519   0.7102 0.5225     0.41817      0.6101    0.02          0
##     6.6645   0.7102 0.5023     0.36246      0.6423    0.02          0
##     7.1772   0.7102 0.4805     0.32902      0.5860    0.02          0
##     7.6898   0.7102 0.4682     0.27874      0.5989    0.02          0
```

```
##     8.2025   0.7102 0.4351     0.20558      0.6092    0.02          0
##     8.7151   0.7102 0.4143     0.14079      0.6199    0.02          0
##     9.2278   0.7102 0.3609     0.02158      0.6216    0.02          0
##     9.7405   0.7102 0.3786     0.01970      0.5695    0.02          0
##    10.2531   0.7102 0.3677     0.01590      0.5974    0.02          0
##        Inf   0.7102 0.3677     0.01590      0.5974    0.02          0
```

```
plot(fg2a)
```



In this case the Pearson Product Momentum correlation makes sense, but when dealing with non-normally distributed data also the Spearman Rank correlation or the Kendall's tau correlation can be used. The FGPT calls upon the `cor.test` function, so for more information on those correlations check the help pages `?cor.test`.

There is also an alternative way to analyse this data. Instead of looking for correlations between the male and the female phenotypes I could look at the differences between them. To do so I first need to produce a matrix which contains all possible combinations of males and females and next run the analyses.

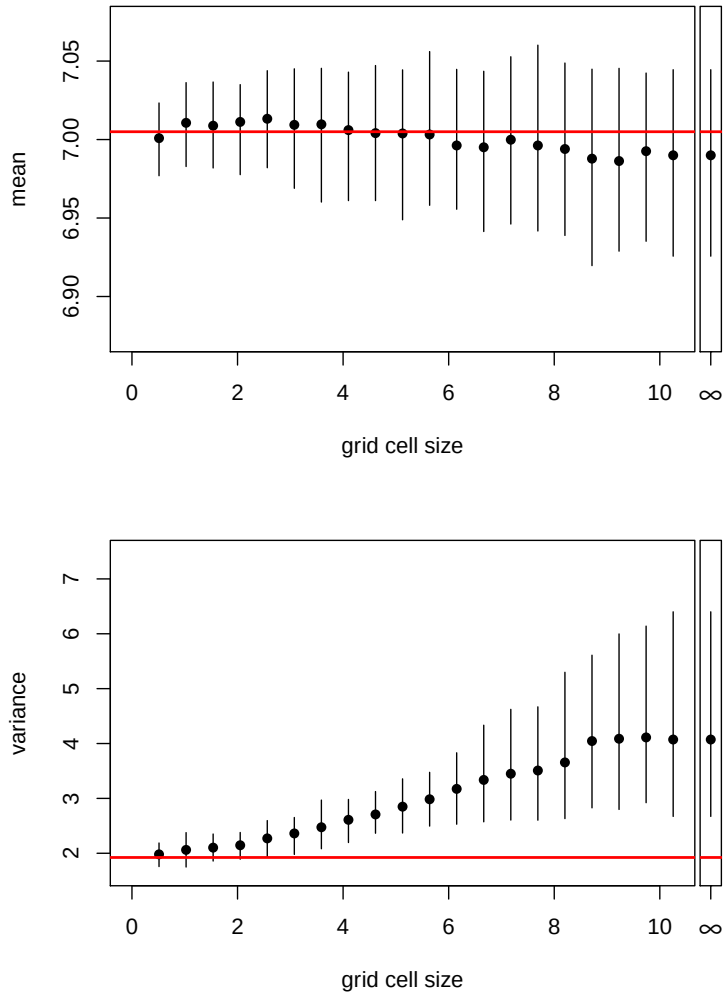
```
size.diff <- matrix(m.pheno, nrow = 200, ncol = 200) - matrix(f.pheno, nrow = 200, ncol = 200,
    byrow = TRUE)

set.seed(99)
fg2c <- fgeasy(xy = Gp.xy, marks = size.diff, iter = 99)
```

```
##
## ======================================
##  Floating Grid Permutation Technique
## ======================================
##
## Progress bar for spatially restricted permutations
## 0% |-------------------------------------------------| 100%
##    ================================================
##
## Non-spatial permutation test is now running.......
##
##
## The analysis took 13.29 secs
```

```
summary(fg2c)

##
## ====================================
##  Floating Grid Permutation Technique
## ====================================
##
## Number of scales: 21
## Number of iterations per scale: 99
##
## -------------------
##  Results for mean:
## -------------------
##
##  cell_size observed  mean lower_limit upper_limit P_value % failures
##     0.5127    7.005 7.001       6.977       7.023    0.68          0
##     1.0253    7.005 7.011       6.983       7.036    0.62          0
##     1.5380    7.005 7.009       6.982       7.037    0.76          0
##     2.0506    7.005 7.011       6.978       7.035    0.54          0
##     2.5633    7.005 7.013       6.982       7.044    0.68          0
##     3.0759    7.005 7.009       6.969       7.045    0.82          0
##     3.5886    7.005 7.010       6.960       7.045    0.80          0
##     4.1012    7.005 7.006       6.961       7.043    0.94          0
##     4.6139    7.005 7.004       6.961       7.047    0.94          0
##     5.1266    7.005 7.004       6.949       7.044    0.98          0
##     5.6392    7.005 7.003       6.958       7.056    1.00          0
##     6.1519    7.005 6.996       6.956       7.045    0.76          0
##     6.6645    7.005 6.995       6.941       7.043    0.84          0
##     7.1772    7.005 7.000       6.946       7.053    0.90          0
##     7.6898    7.005 6.996       6.942       7.060    0.78          0
##     8.2025    7.005 6.994       6.939       7.049    0.74          0
##     8.7151    7.005 6.988       6.920       7.045    0.56          0
##     9.2278    7.005 6.986       6.929       7.045    0.56          0
##     9.7405    7.005 6.993       6.935       7.042    0.80          0
##    10.2531    7.005 6.990       6.926       7.044    0.78          0
##        Inf    7.005 6.990       6.926       7.044    0.78          0
##
##  Results for variance:
## ----------------------
##
##  cell_size observed  mean lower_limit upper_limit P_value % failures
##     0.5127    1.923 1.980       1.758       2.187    0.58          0
##     1.0253    1.923 2.062       1.749       2.376    0.42          0
##     1.5380    1.923 2.103       1.858       2.351    0.18          0
##     2.0506    1.923 2.146       1.892       2.379    0.14          0
##     2.5633    1.923 2.271       1.935       2.596    0.04          0
##     3.0759    1.923 2.361       1.979       2.652    0.04          0
##     3.5886    1.923 2.474       2.082       2.969    0.02          0
##     4.1012    1.923 2.609       2.196       2.981    0.02          0
##     4.6139    1.923 2.708       2.365       3.125    0.02          0
##     5.1266    1.923 2.849       2.369       3.356    0.02          0
##     5.6392    1.923 2.984       2.495       3.475    0.02          0
##     6.1519    1.923 3.173       2.530       3.831    0.02          0
##     6.6645    1.923 3.337       2.573       4.334    0.02          0
```

```
##     7.1772    1.923 3.449       2.605       4.622    0.02          0
##     7.6898    1.923 3.508       2.603       4.668    0.02          0
##     8.2025    1.923 3.655       2.632       5.298    0.02          0
##     8.7151    1.923 4.045       2.828       5.609    0.02          0
##     9.2278    1.923 4.087       2.799       5.997    0.02          0
##     9.7405    1.923 4.111       2.922       6.140    0.02          0
##    10.2531    1.923 4.073       2.672       6.400    0.02          0
##        Inf    1.923 4.073       2.672       6.400    0.02          0
```

```
plot(fg2c)
```



In this case I focus on the variance rather than the mean. I don't expect the mean to be informative, because the mean is pretty much independent of the spatial scale. There is only a bit of variation in the mean for different permutations, because of the missing values. Because of the missing values, not all individuals will be paired in every permutation which results in some variation in the average difference between female and male individuals. The variance plot looks very similar to the correlation plot (though upside down). The fact that the variance is lower than expected for grid cell size of 3.6 and larger suggests that pairs are better matched with respect to body size than expected by change.

4 Concluding remarks

The FGPT is a spatially resticted permutation technique, which has the ability to separate various sources of spatial autocorrelation, given they act on different spatial levels and the study area is suffiently large. The `FGPT` package offers functions to perform spatially restricted permutations, calulate statistics for permuted dataset and functions to test for correlations or distances given spatial autocorrelation and to summary and plot the results. The method is free of assumptions regarding the spatial scale at which spatial processes occur. Therefore it is very suitable for systems in which there is no extensive knowledge on the scale of spatial processes and can be used to study for instance animal or plant populations that show genotypic or phenotypic variation in space.

5 References

- Radersma R and Sheldon BC, 2014. A new permutation test to explore and control for spatial autocorrelation in point pattern data. *Submitted*.
- Ruxton GD and Neuhäuser M, 2013. Improving the reporting of P-values generated by randomization methods. *Methods in Ecology and Evolution* 4:1033-1036.