For Topology 4/10/2020

Fred Kaesmann

4/7/2020

From my notes from Monday (4/7/2020):

Example filter function:

$$\Sigma = \text{ Sample } = \{(x_i, y_i)\}$$

Program cover with resolution and gain

- If given a resolution and gain from user
- and some given height function (example above)
- Determine the overlap of covers (intersections of pullbacks)
- cluster points in overlap (user's choice)
- cluster become the connected components

Small program for Wednesday:

1. Input:
   - N: points in R^2 with $y \geq 0$
   - r: resolution value
   - g: gain
2. Output: Under r and g values given
   - Produce the preimages of the covering intervals
   - "If you're feeling froggy:" Maybe consider an instersection matrix for when pullbacks nontrivially intersect?

Part 1

```
PullBackCovers <- function(x, y, resolution, gain){

  ##### Create New Data Frame from x and y Values Entered ####
  df <- data.frame(
    x = x,
    y = y
  )

  ##### Initial Info ####
  y_max <- max(df$y)                                 # When to stop
  increment <- resolution * (1 - gain)               # Lower bound iterations
  n_sets <- ceiling(y_max / increment)               # Number of Sets

  ##### Cover Boundaries ####
  # Initialize Variables
  lower <- 0 - increment
  upper <- lower + resolution
  covers <- list()
  for (cover_number in c(1:n_sets)) {
    covers[[cover_number]] <- df[df$y > lower & df$y < upper, ] # List entry cover
    lower <- (increment * cover_number)                         # Lower Bound
    upper <- lower + resolution                                 # Upper Bound

  }
  ##### Meaningful List Entry Names ####
  names(covers) <- paste0("Cover_", c(1:n_sets))

  ##### Return the list ####
  return(covers)
}
```

```
covers <- PullBackCovers(runif(10)*10, runif(10)*10,
                           resolution =  4,
                                 gain =  0.75)
print(covers[c(1:2)])
```

```
## $Cover_1
##          x         y
## 6 5.882575 0.6042926
##
## $Cover_2
##           x        y
## 1  3.855625 3.511748
## 10 1.292005 3.734862
```

Part 2

```r
CoverAdjacency <- function(covers, binary = TRUE) {
  ##### Adjacency Matrix ####

  ##### Packages ####
  library(dplyr, warn.conflicts = F, quietly = T)

  ##### Iteration Ranges ####
  list_length <- length(covers)
  list_range <- c(1:list_length)

  ##### Initialize Empty Matrix #####
  adj_mat <- matrix(nrow = list_length,
                   ncol = list_length,
                   dimnames = list(
                     paste0("C",list_range),
                     paste0("C",list_range)
                   ))

  ##### ith Row, jth Column ####
  for (i in list_range) {
  for (j in list_range) {

   ##### Binary Adjacency Matrix ####
   if (binary == TRUE) {
     if (nrow(anti_join(covers[[i]], covers[[j]], by = c("x", "y"))) < nrow(covers[[i]])){
       adj_mat[i, j] <- 1
     } else {
       adj_mat[i, j] <- 0
     }
    ##### Count the Overlapped Points ####
   } else {
     adj_mat[i, j] <- nrow(covers[[i]]) - nrow(anti_join(covers[[i]], covers[[j]], by = c("x", "y")))
   }

  }
  }

  ##### Return the Adjacency Matrix ####
  return(adj_mat)
}
```

```r
##### Binary Adjacency Matrix ####
CoverAdjacency(covers)
```

```
##     C1 C2 C3 C4 C5 C6 C7 C8 C9 C10
## C1   1  0  0  0  0  0  0  0  0   0
## C2   0  1  1  1  0  0  0  0  0   0
## C3   0  1  1  1  0  0  0  0  0   0
## C4   0  1  1  1  1  1  1  0  0   0
## C5   0  0  0  1  1  1  1  1  0   0
## C6   0  0  0  1  1  1  1  1  1   0
## C7   0  0  0  1  1  1  1  1  1   1
## C8   0  0  0  0  1  1  1  1  1   1
## C9   0  0  0  0  0  1  1  1  1   1
## C10  0  0  0  0  0  0  1  1  1   1
```

```r
##### Count the Points in Common ####
CoverAdjacency(covers, binary=FALSE)
```

```
##     C1 C2 C3 C4 C5 C6 C7 C8 C9 C10
## C1   1  0  0  0  0  0  0  0  0   0
## C2   0  2  2  2  0  0  0  0  0   0
## C3   0  2  2  2  0  0  0  0  0   0
## C4   0  2  2  4  2  2  2  0  0   0
## C5   0  0  0  2  4  4  4  2  0   0
## C6   0  0  0  2  4  6  6  4  2   0
## C7   0  0  0  2  4  6  7  5  3   1
## C8   0  0  0  0  2  4  5  5  3   1
## C9   0  0  0  0  0  2  3  3  3   1
## C10  0  0  0  0  0  0  1  1  1   1
```