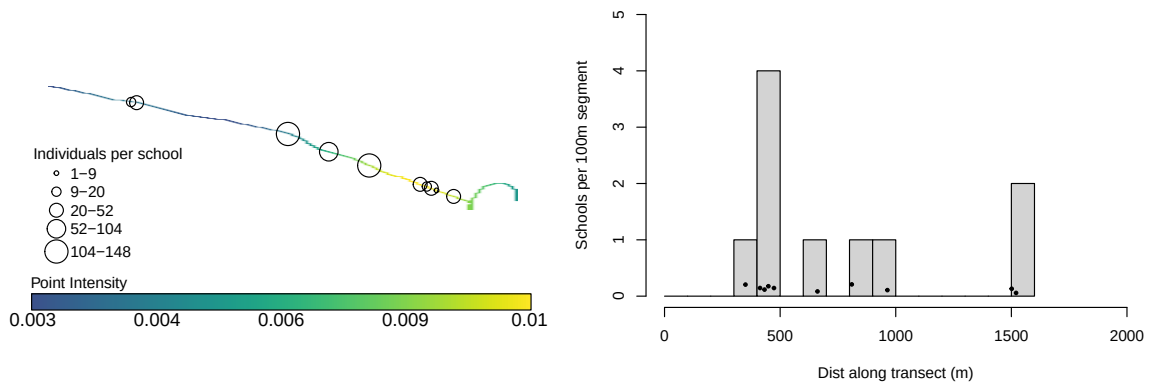
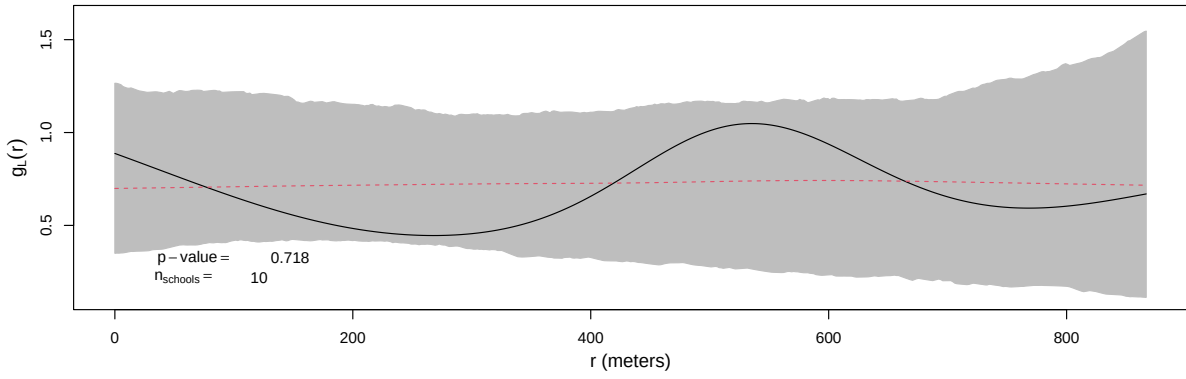
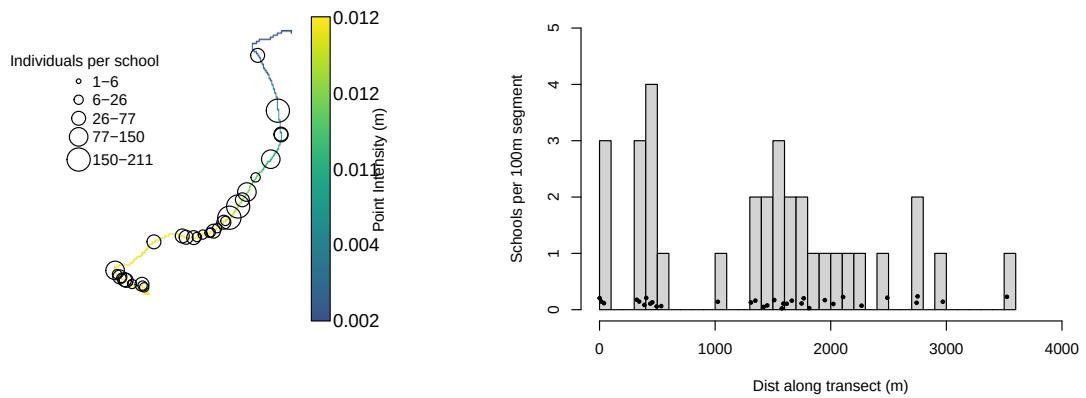
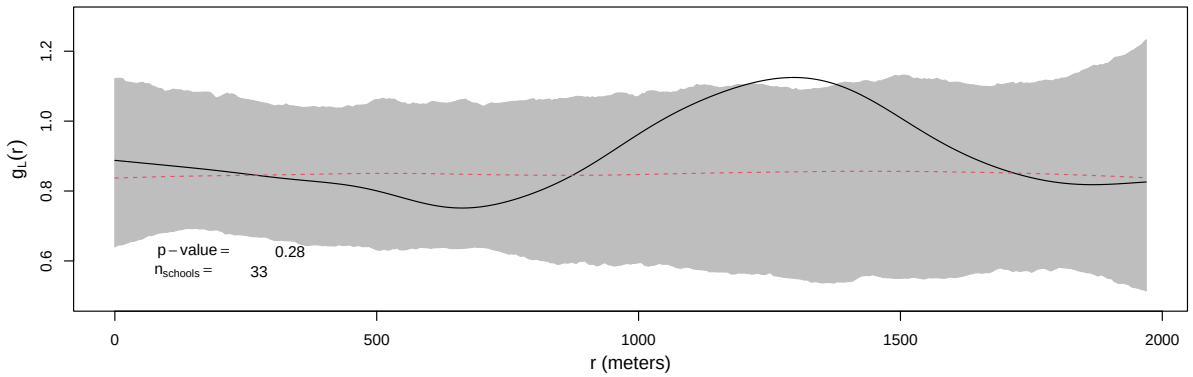


Appendix A: Linear networks and PCFs for all transects

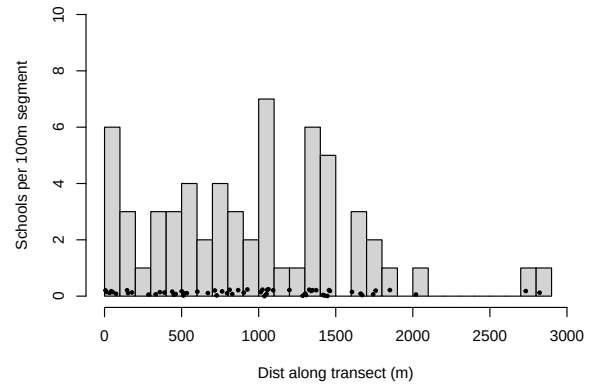
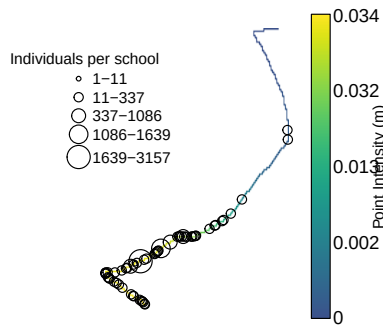
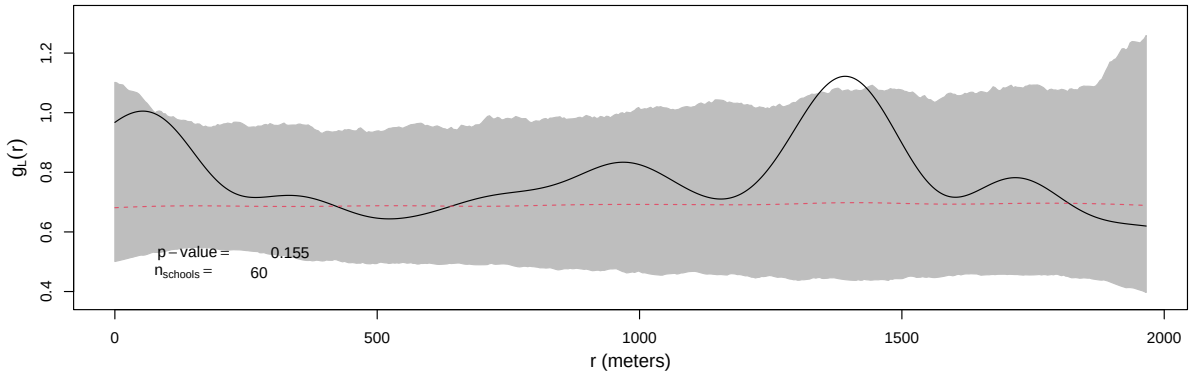
South Patuxent – 15 Jun 2016



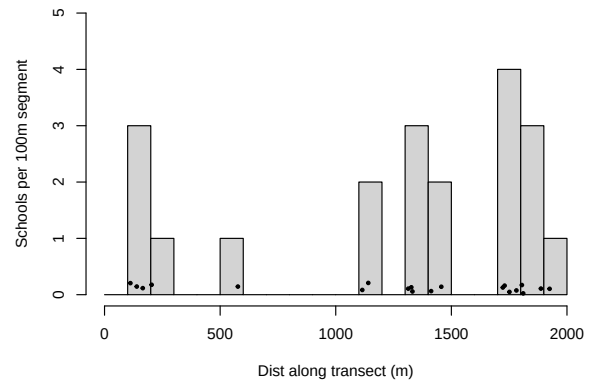
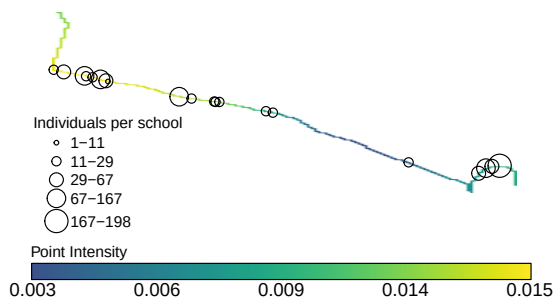
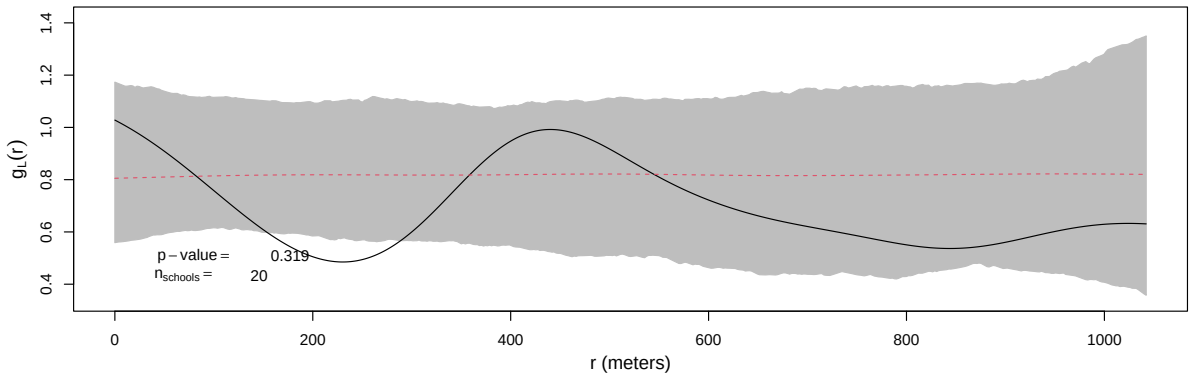
St. Leonard Creek – 15 Jun 2016



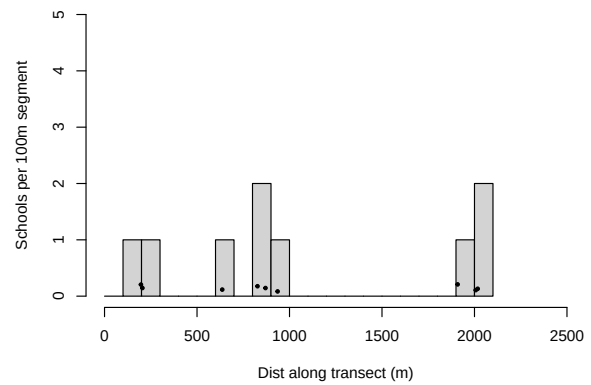
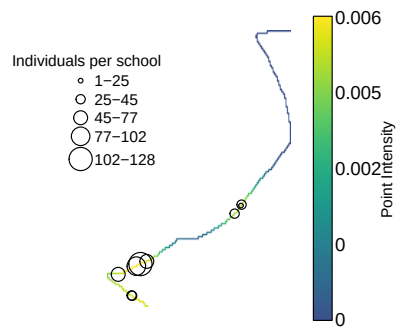
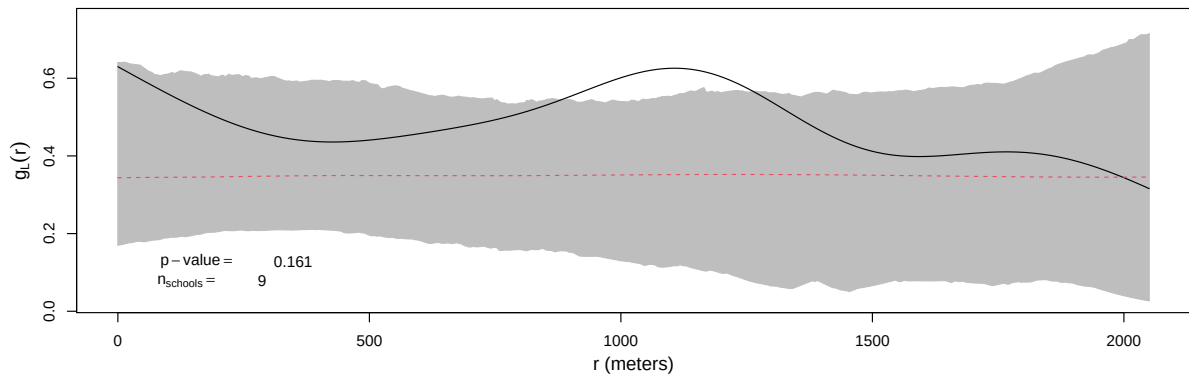
St. Leonard Creek – 28 Jun 2016



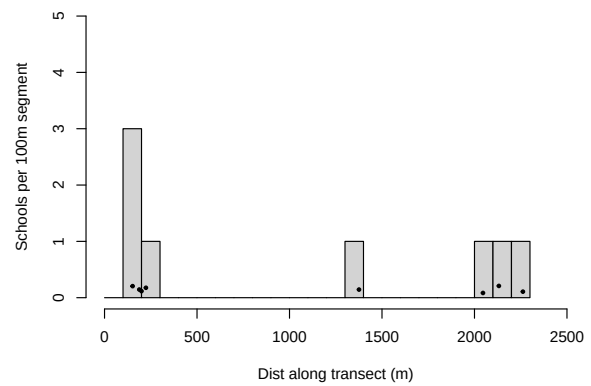
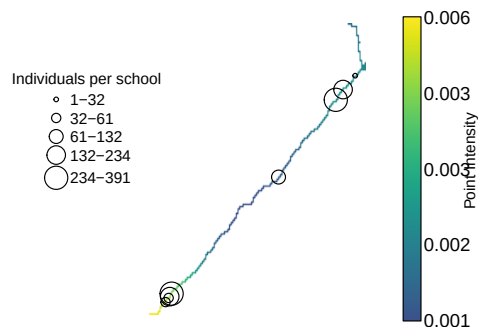
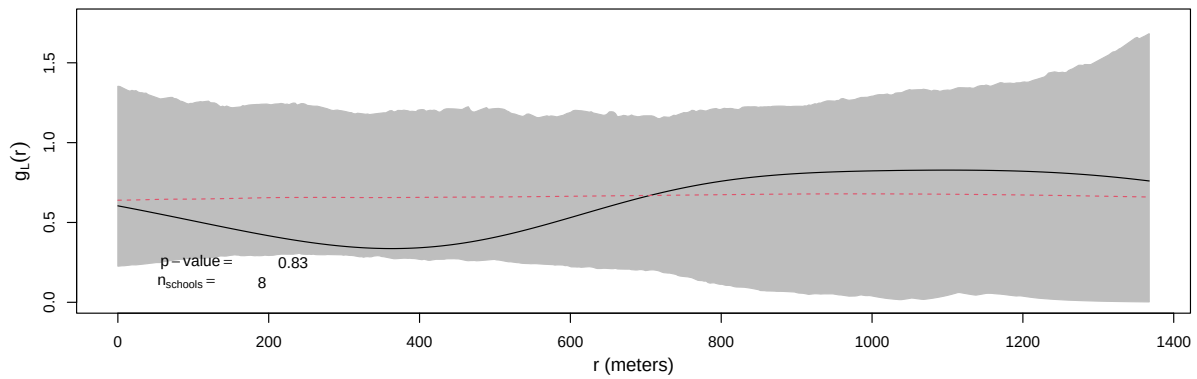
South Patuxent – 14 Jul 2016



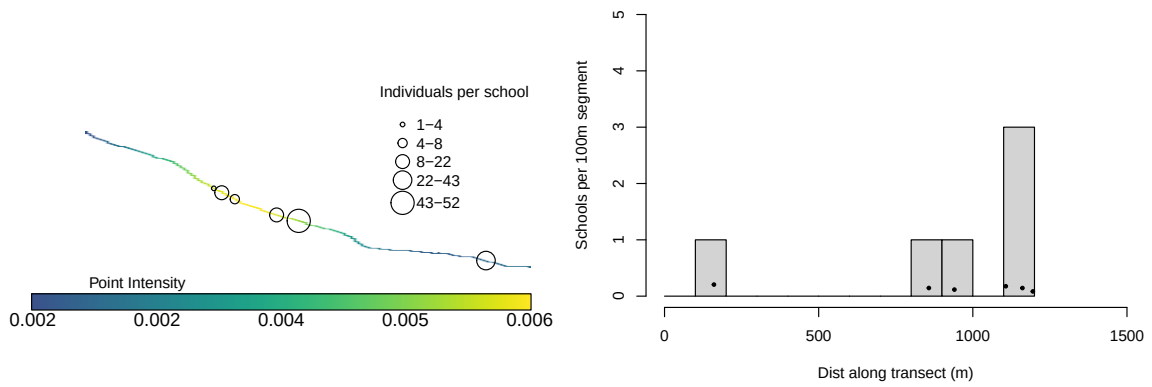
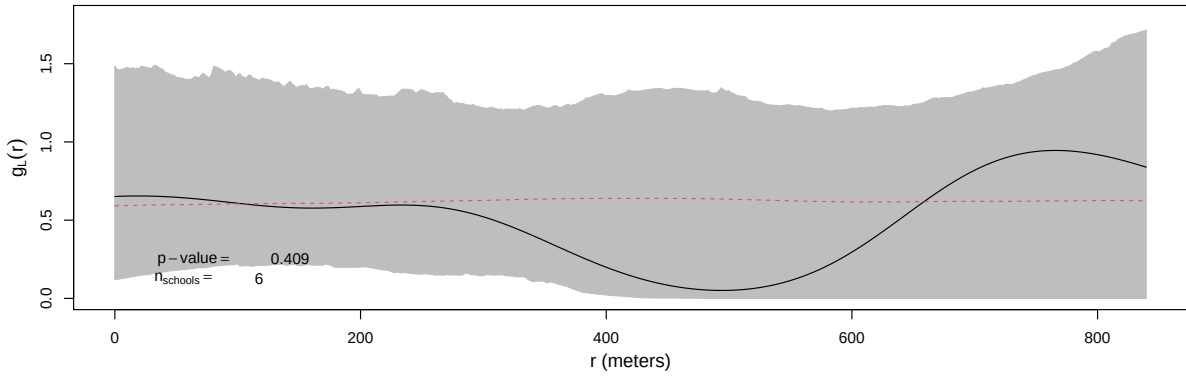
St. Leonard Creek – 14 Jul 2016



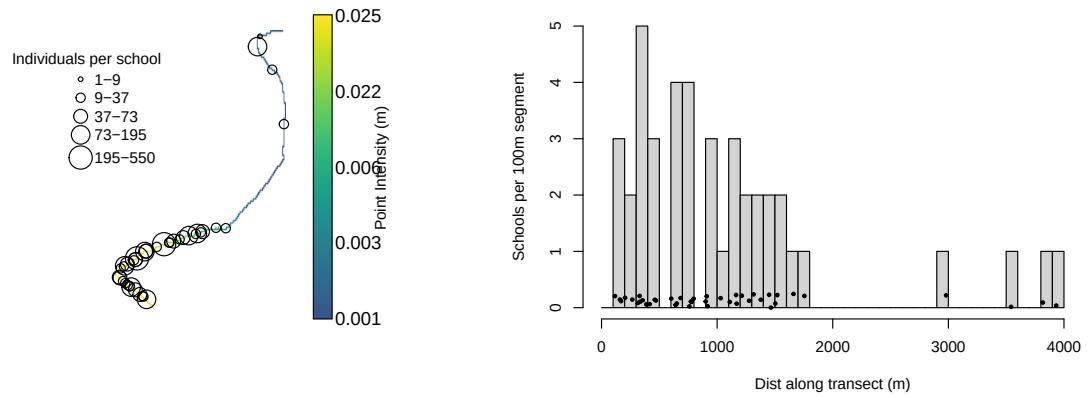
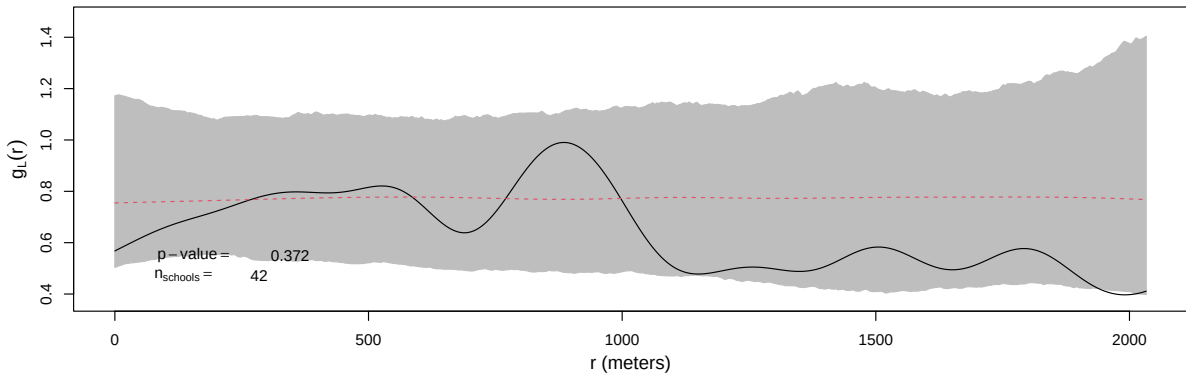
North Patuxent – 14 Jul 2016



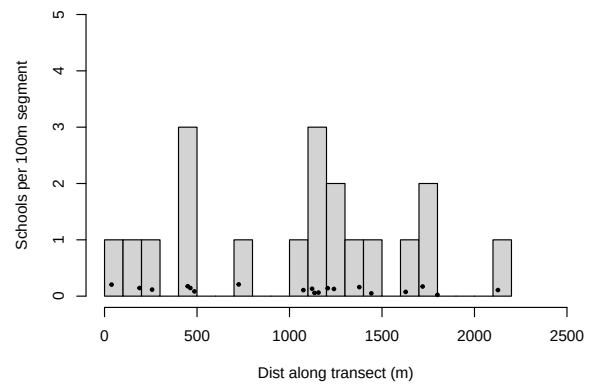
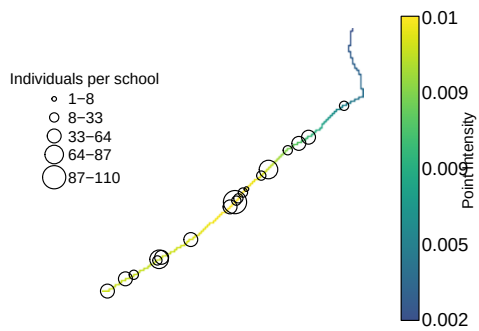
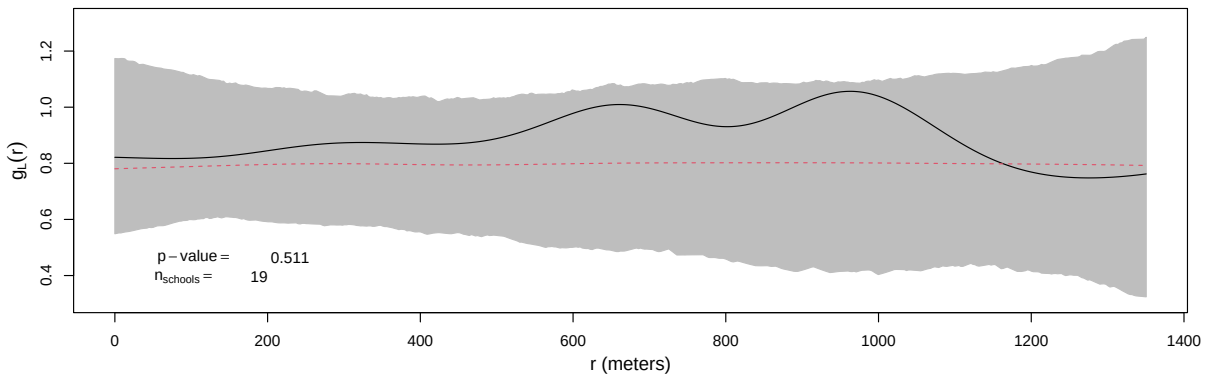
South Patuxent – 21 Jul 2016



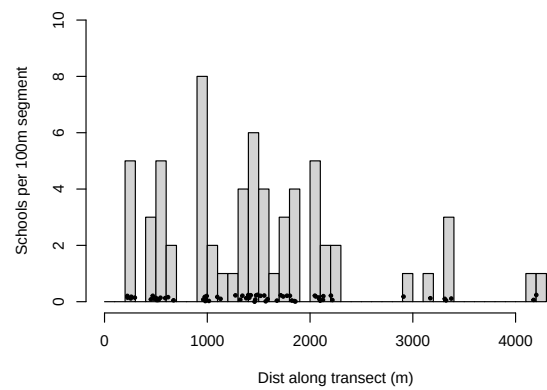
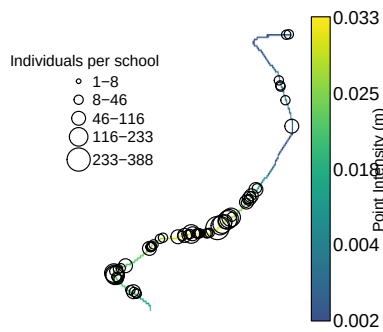
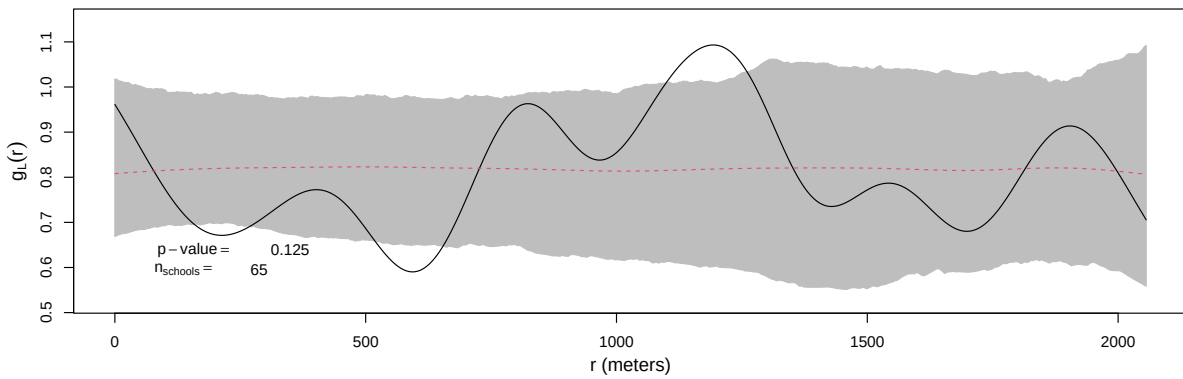
St. Leonard Creek – 21 Jul 2016



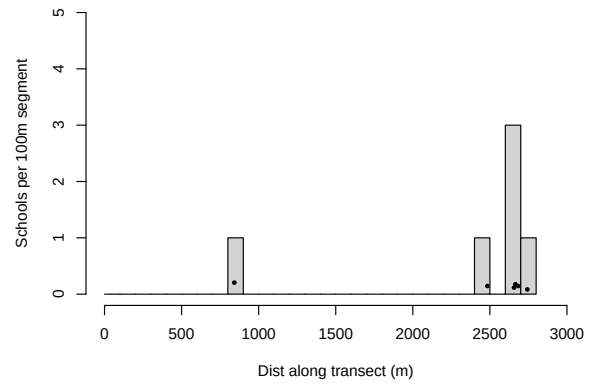
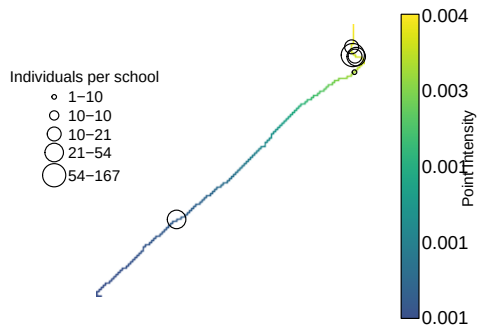
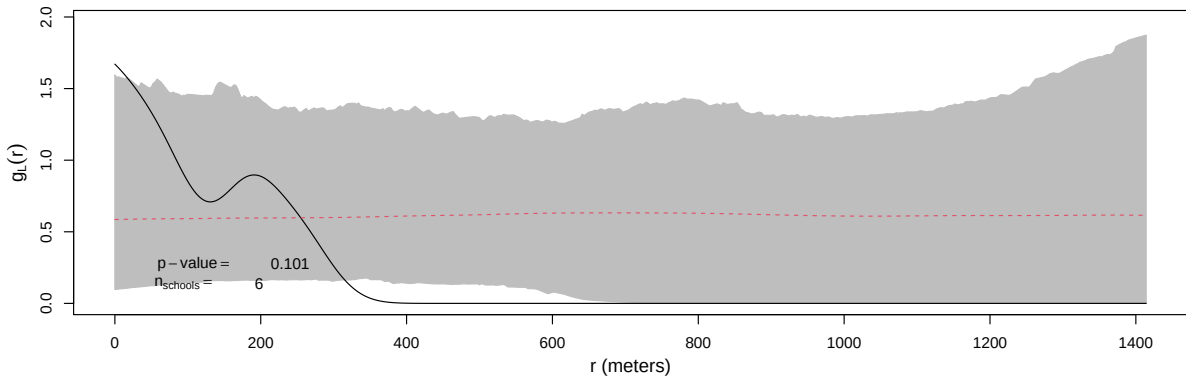
North Patuxent – 21 Jul 2016



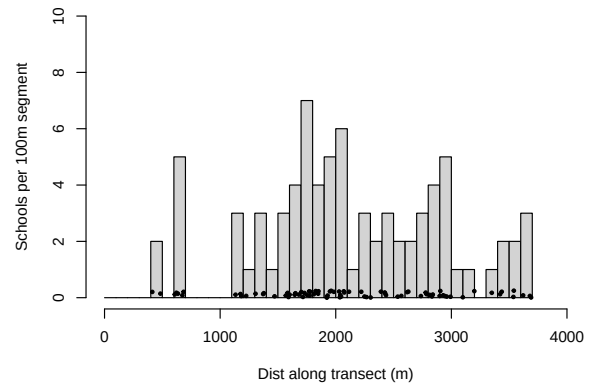
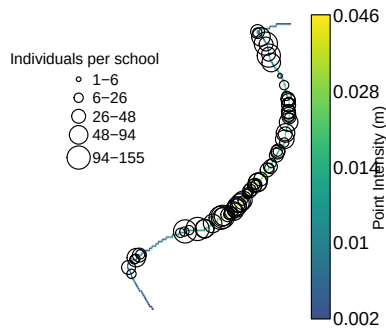
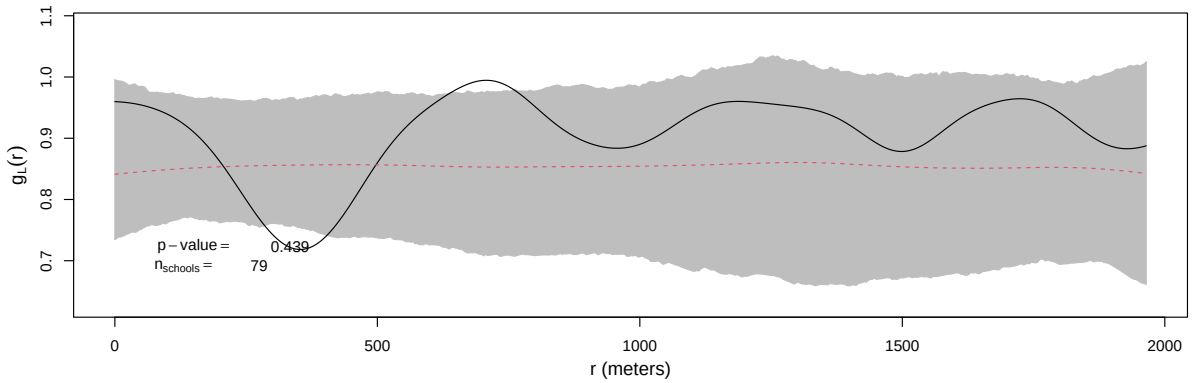
St. Leonard Creek – 28 Jul 2016



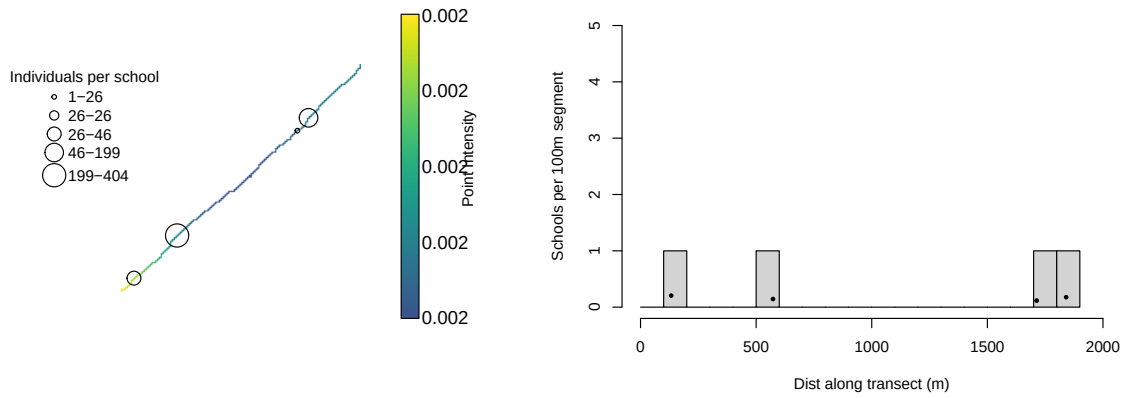
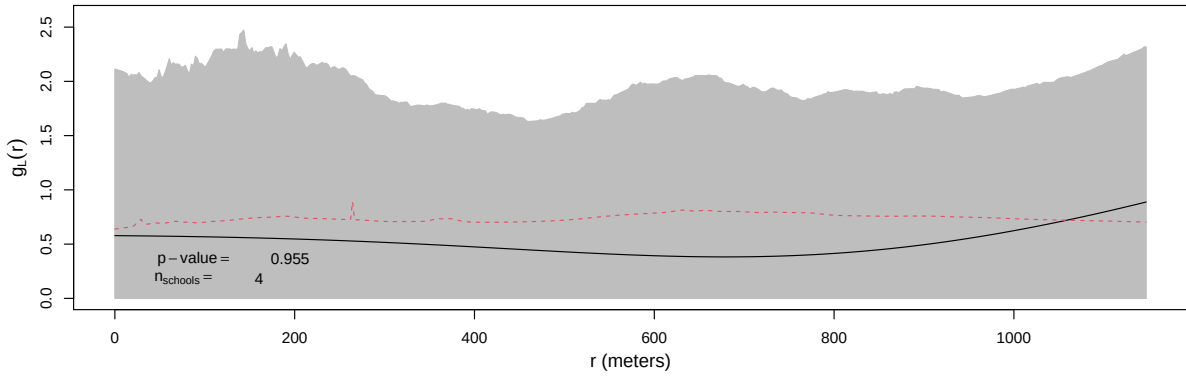
North Patuxent – 28 Jul 2016



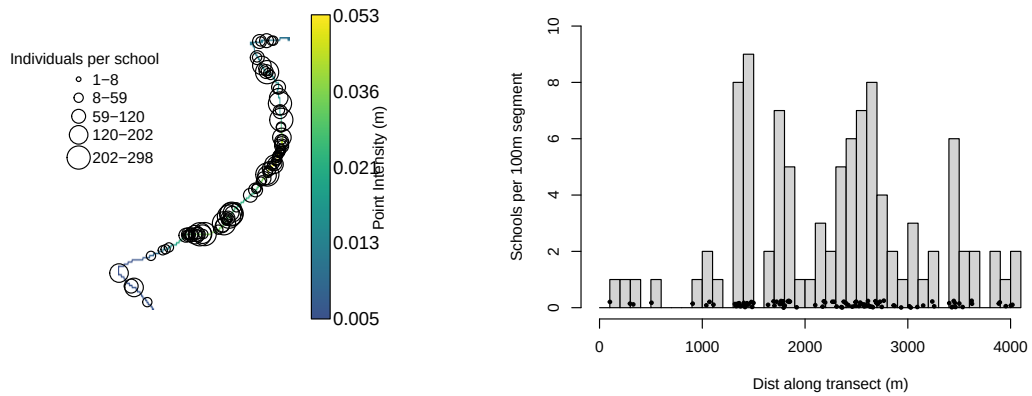
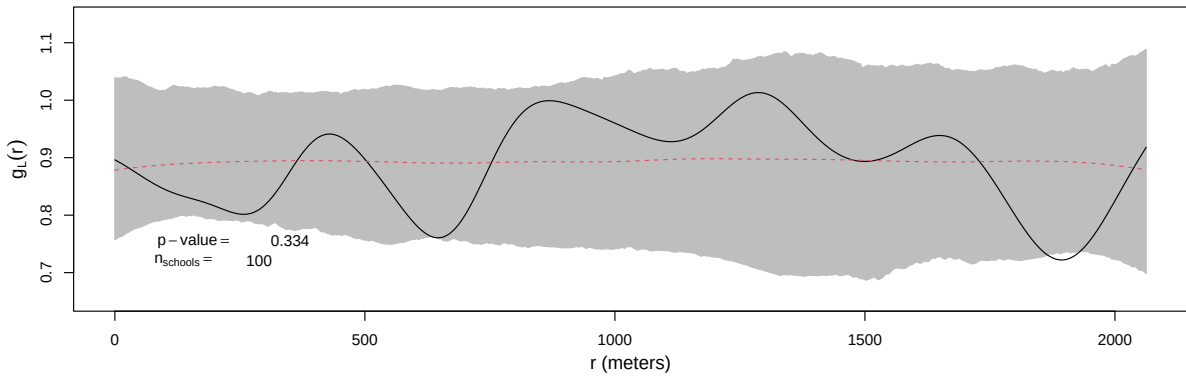
St. Leonard Creek – 11 Aug 2016



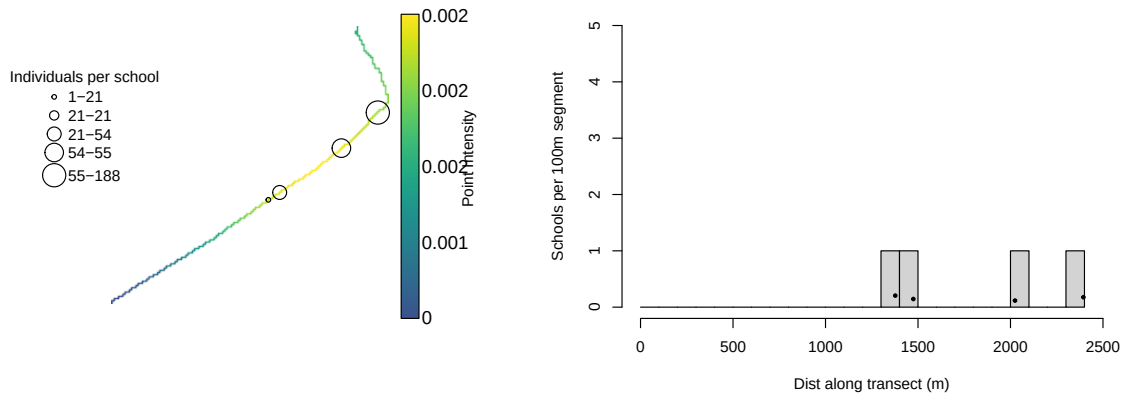
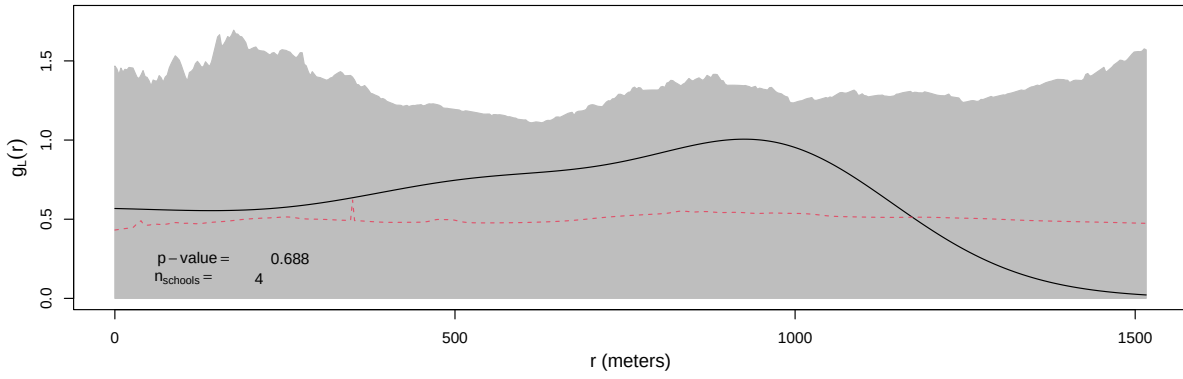
North Patuxent – 11 Aug 2016



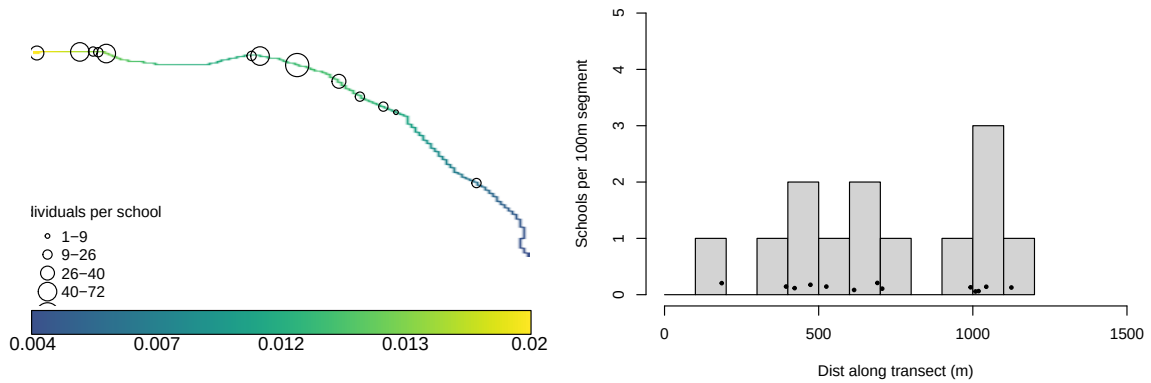
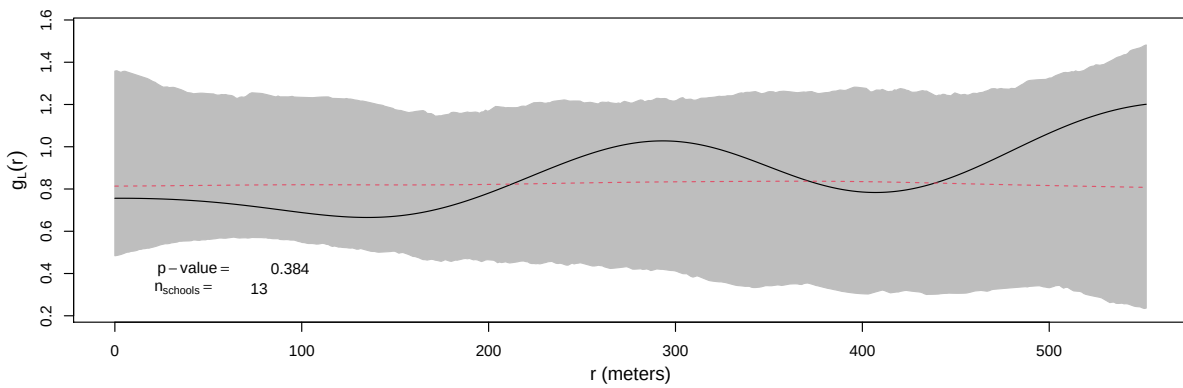
St. Leonard Creek – 26 Aug 2016



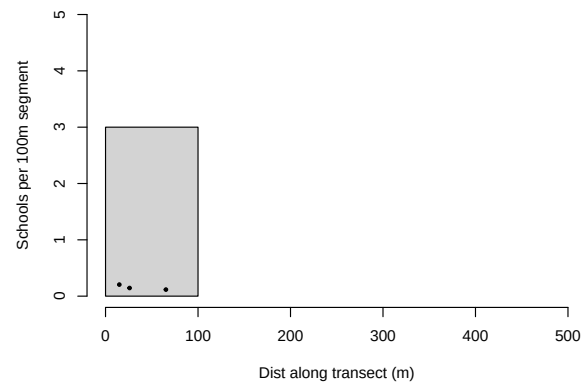
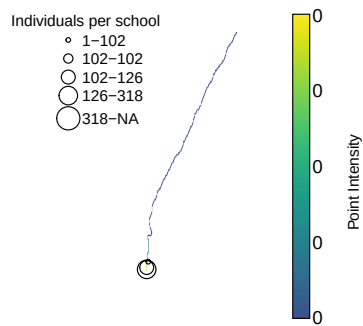
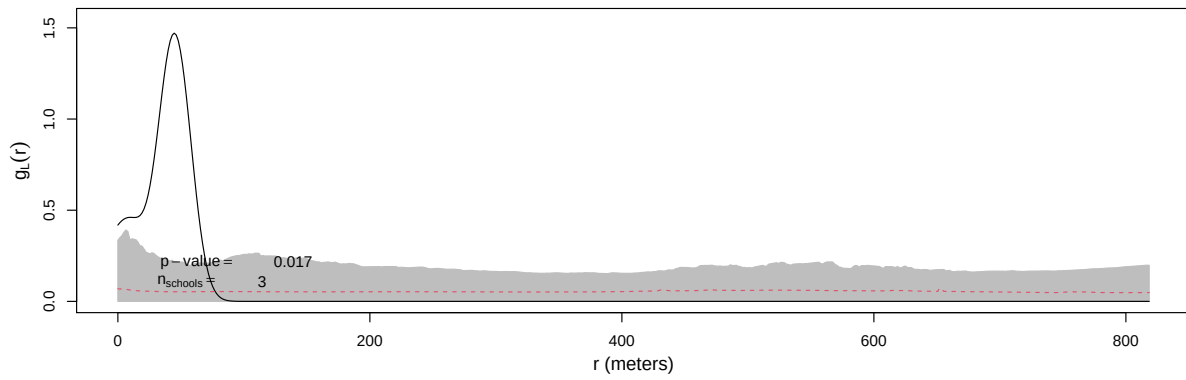
North Patuxent – 26 Aug 2016



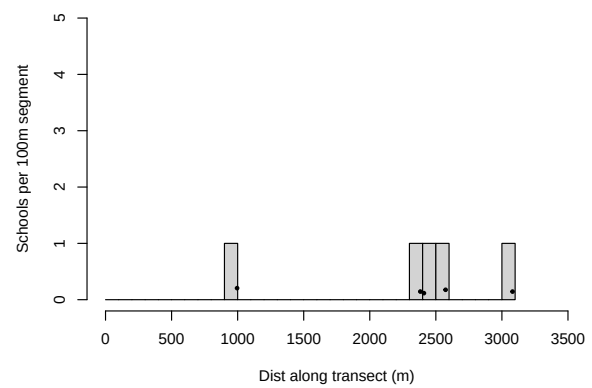
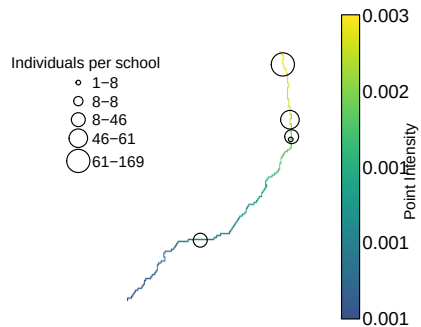
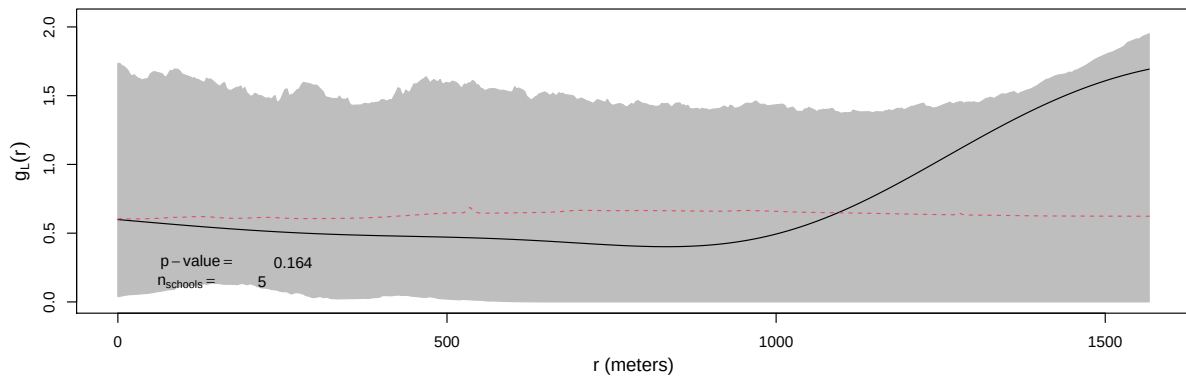
Cuckold Creek – 08 Jun 2017



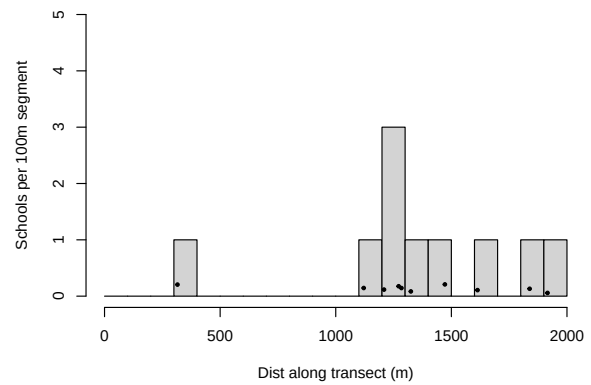
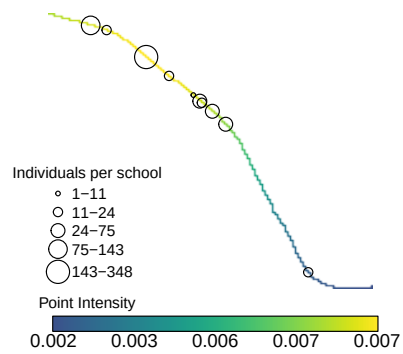
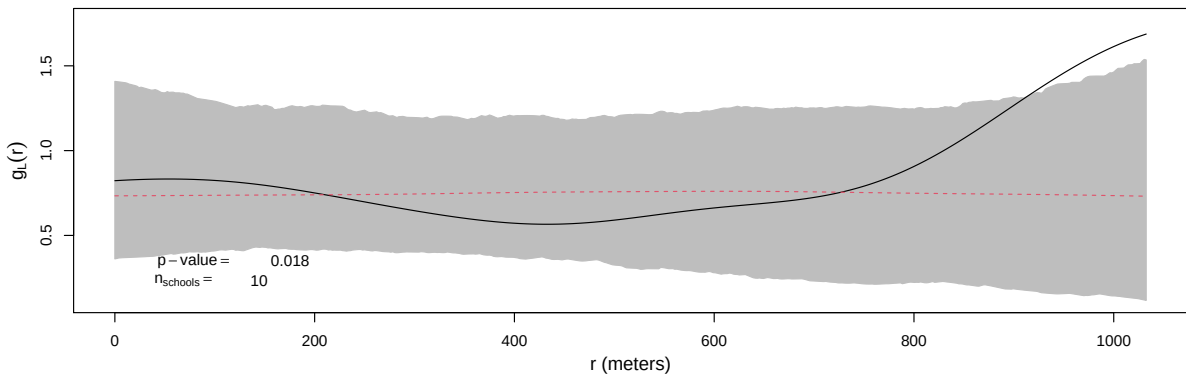
Hellen Creek – 08 Jun 2017



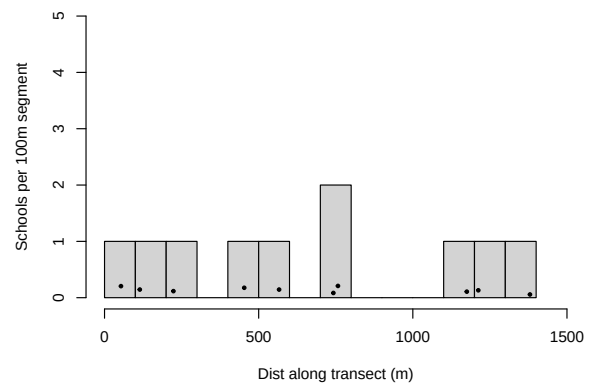
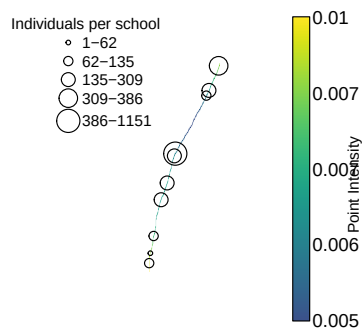
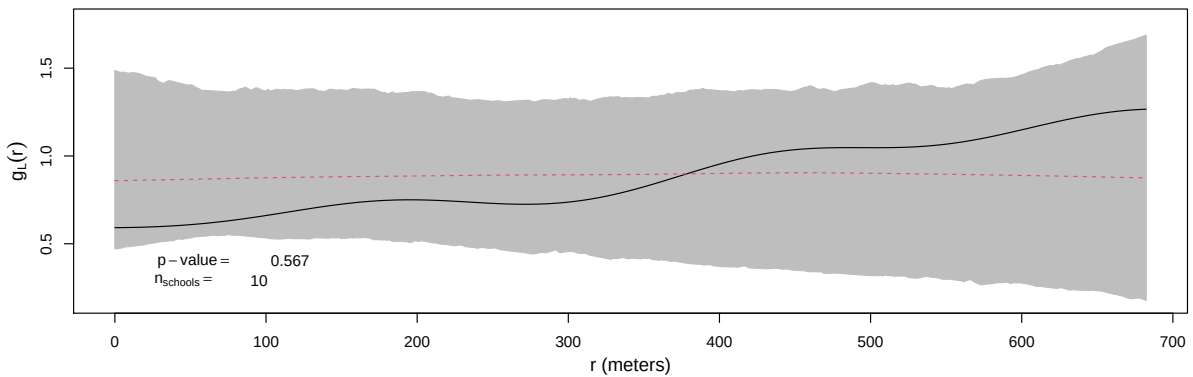
St. Leonard Creek – 08 Jun 2017



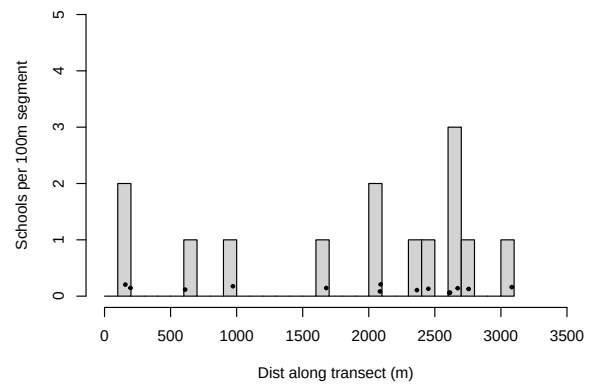
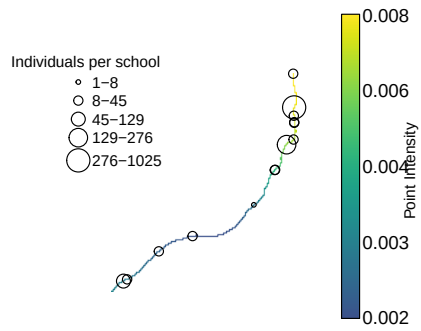
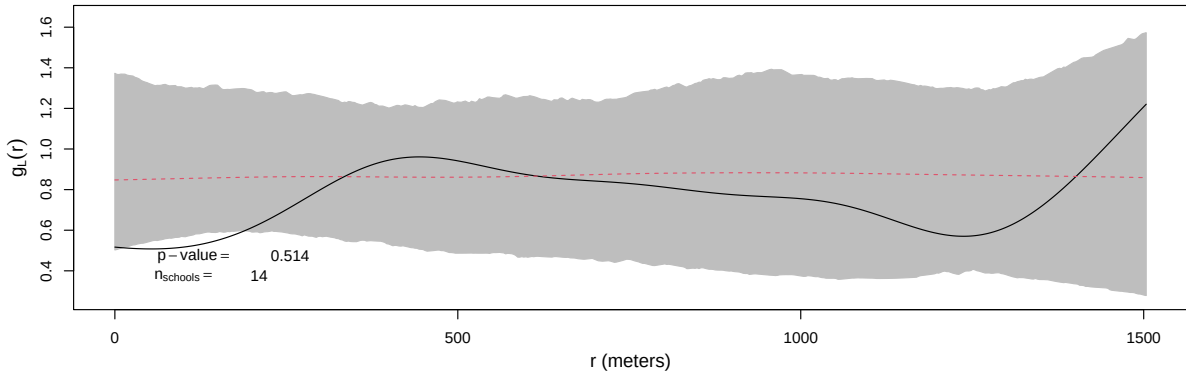
Cuckold Creek – 16 Jun 2017



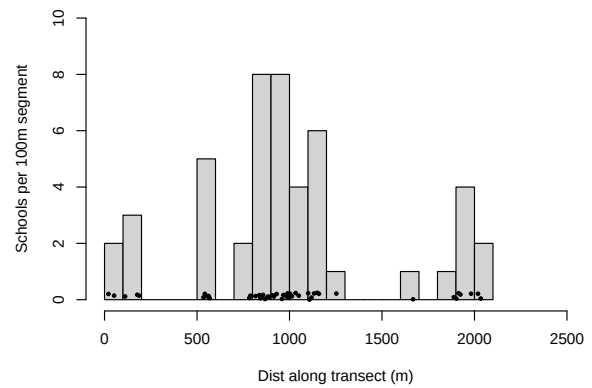
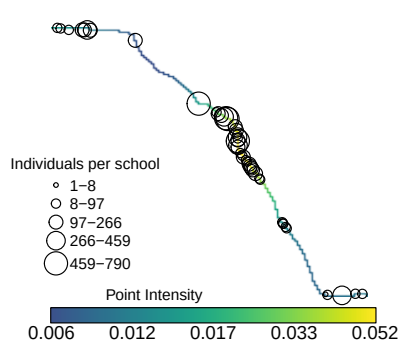
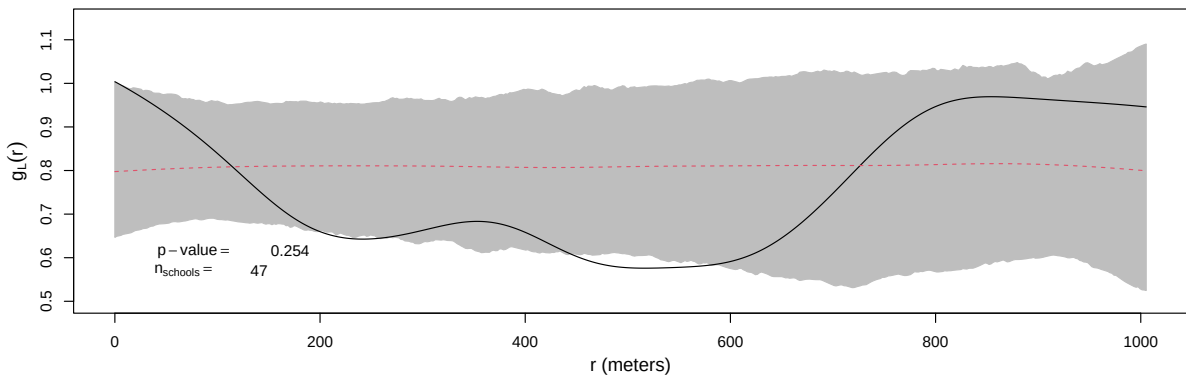
Hellen Creek – 16 Jun 2017



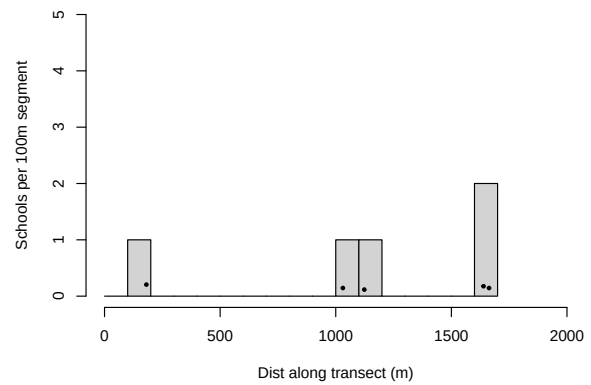
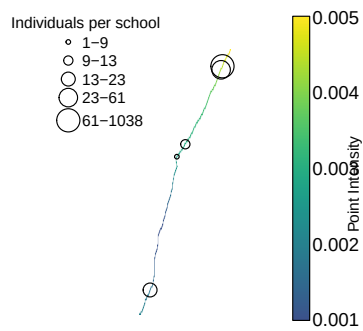
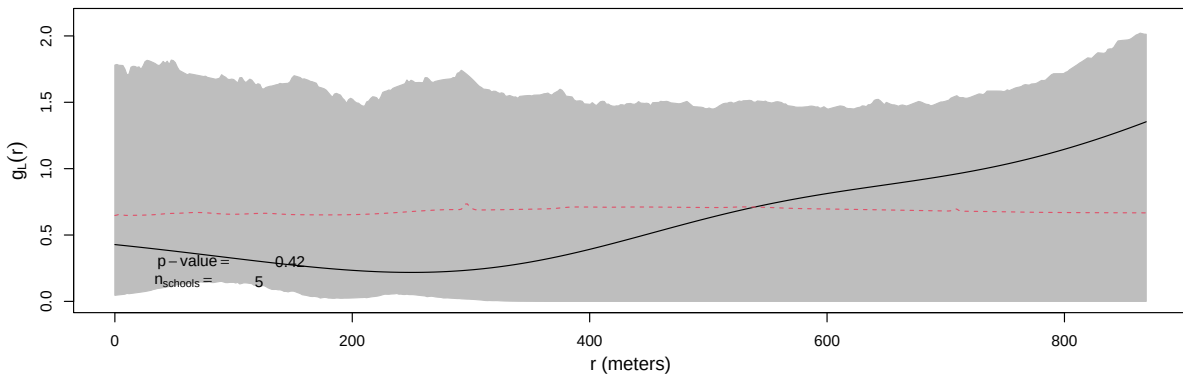
St. Leonard Creek – 16 Jun 2017



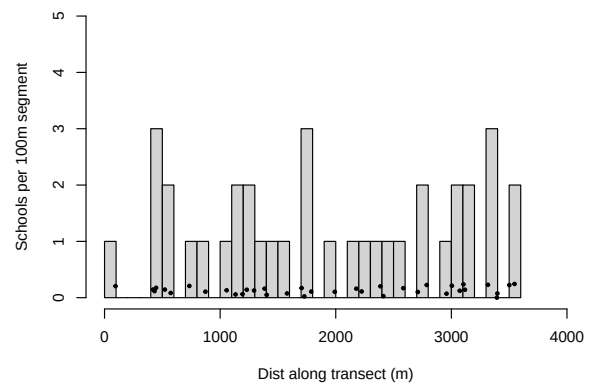
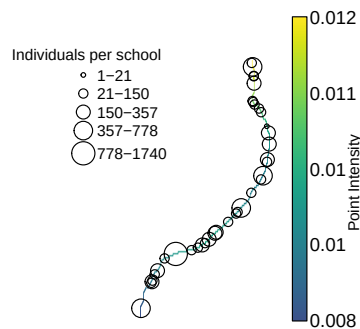
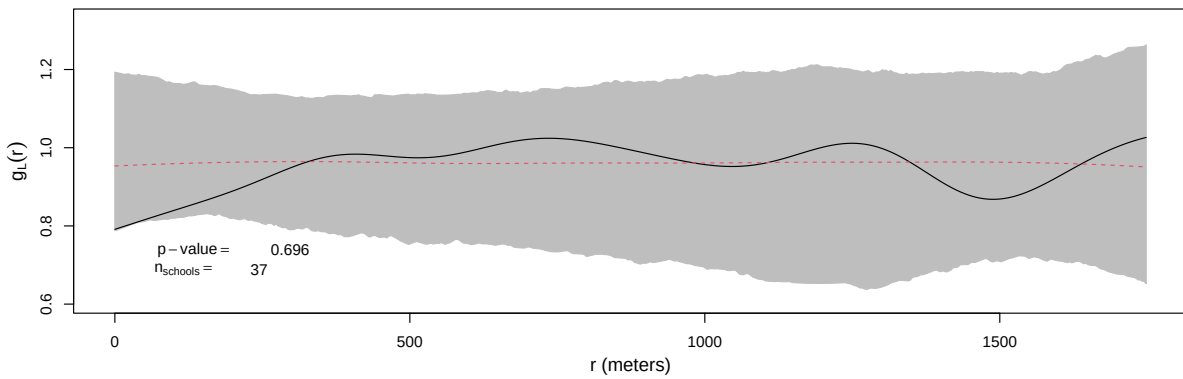
Cuckold Creek – 27 Jun 2017



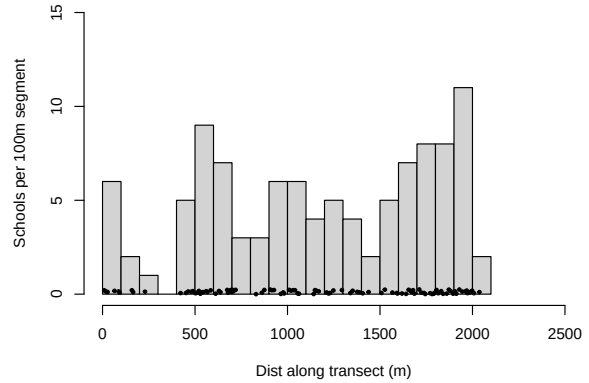
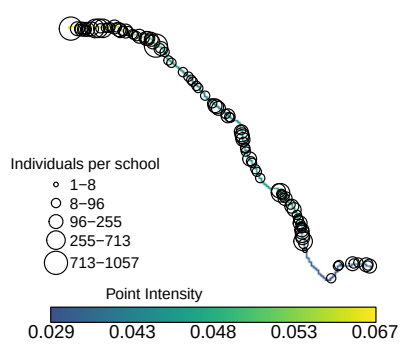
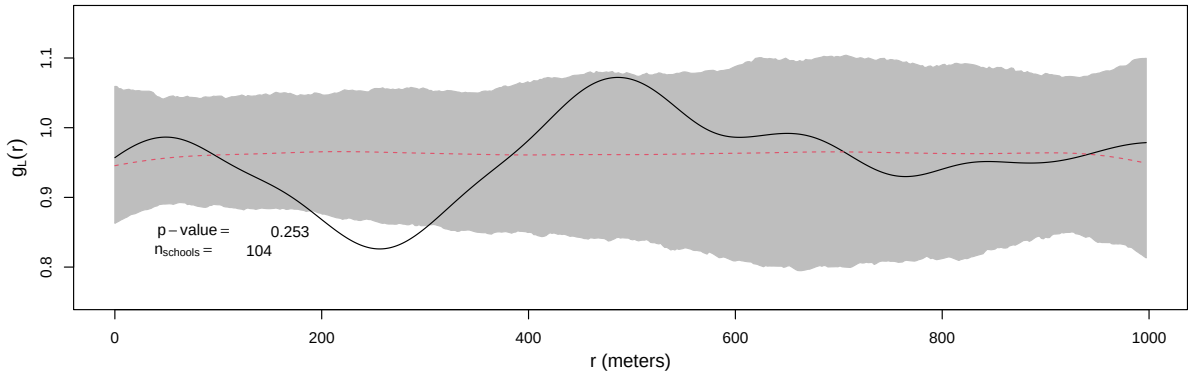
Hellen Creek – 27 Jun 2017



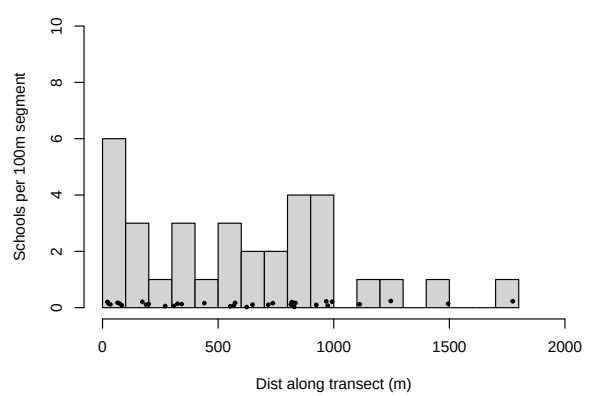
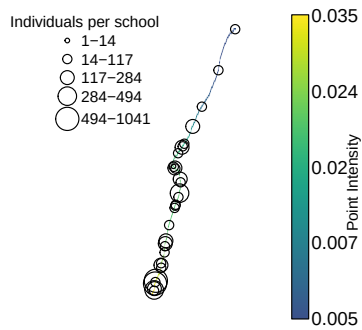
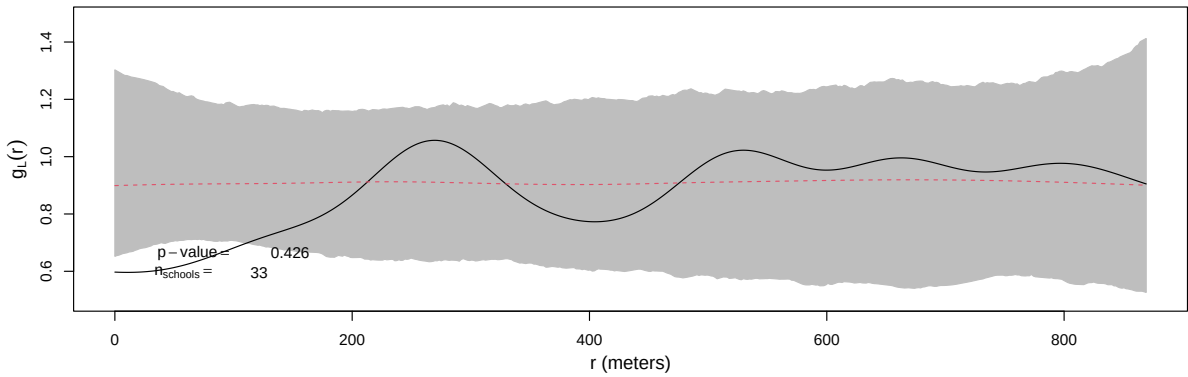
St. Leonard Creek – 27 Jun 2017



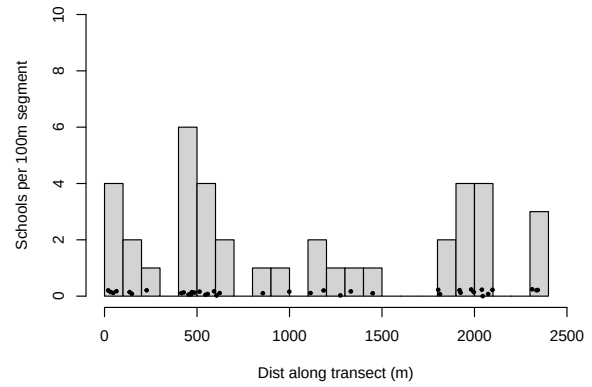
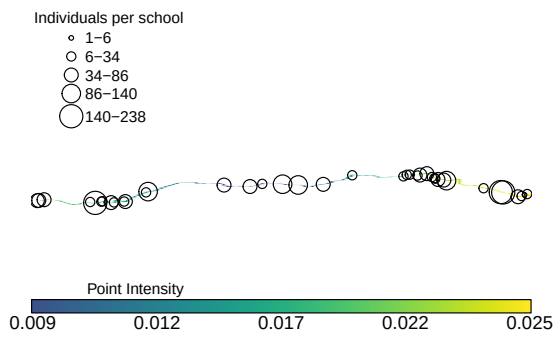
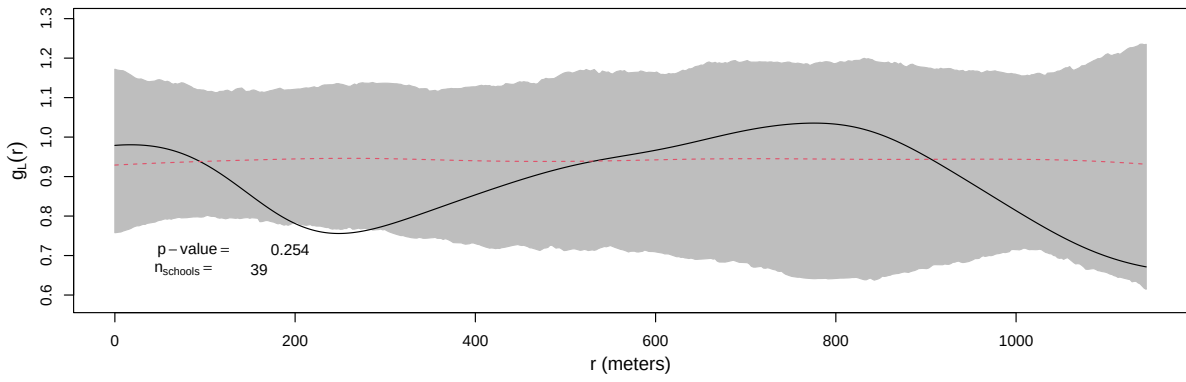
Cuckold Creek – 07 Jul 2017



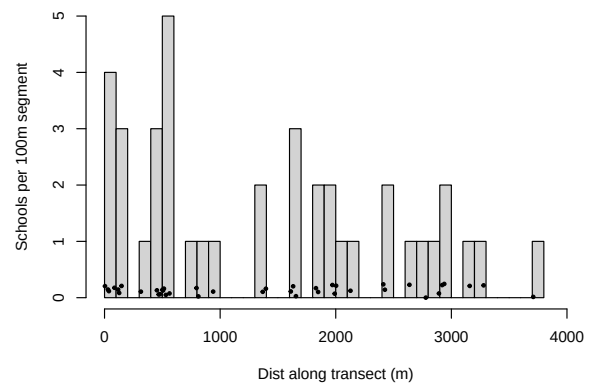
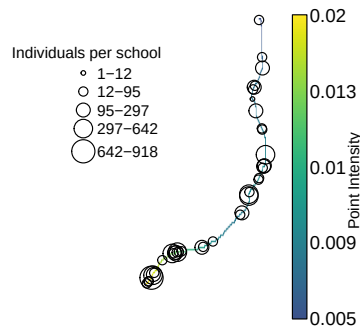
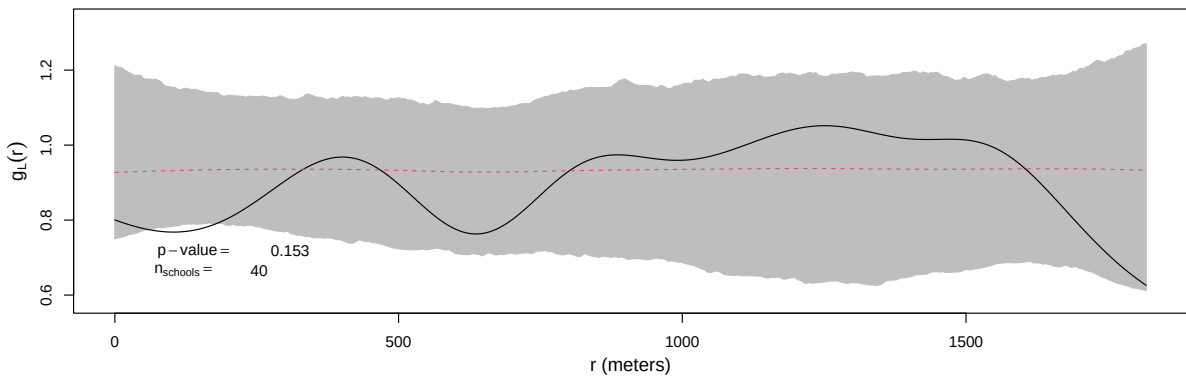
Hellen Creek – 07 Jul 2017



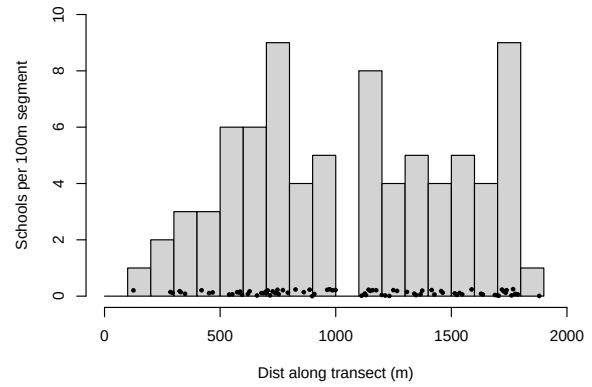
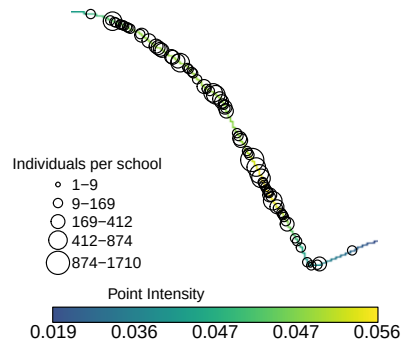
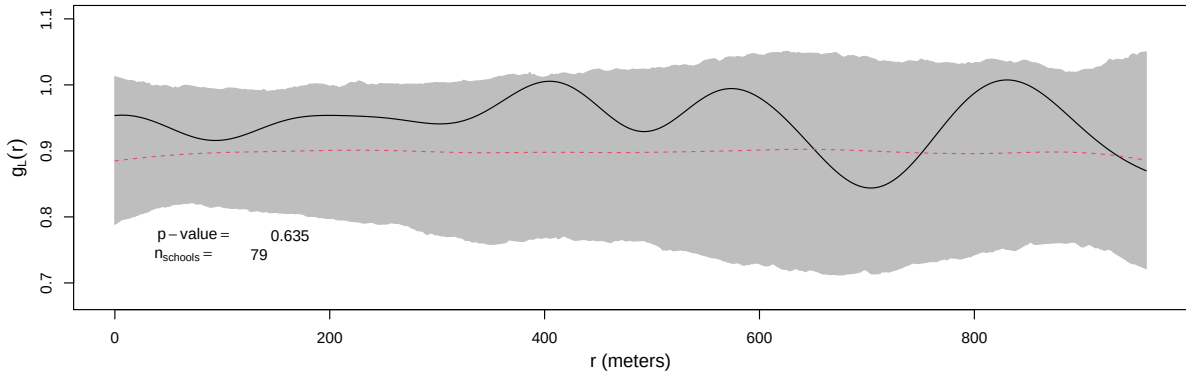
South Patuxent – 07 Jul 2017



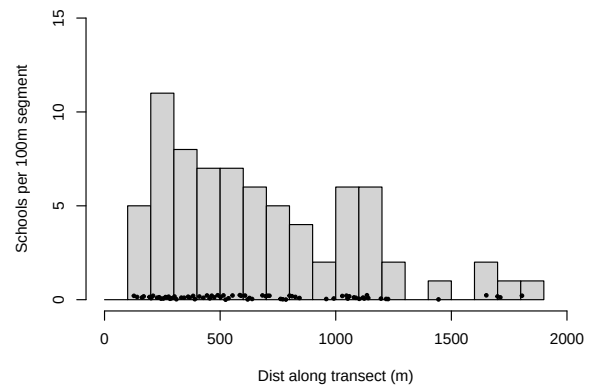
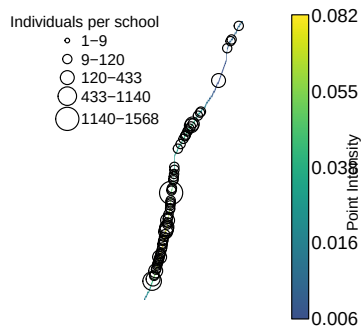
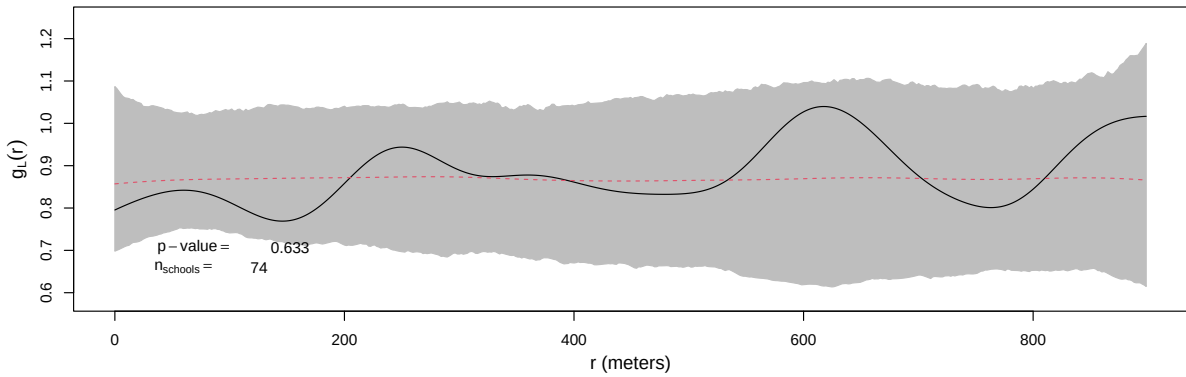
St. Leonard Creek – 07 Jul 2017



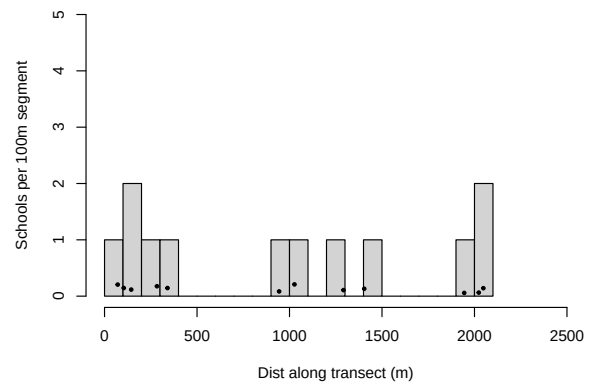
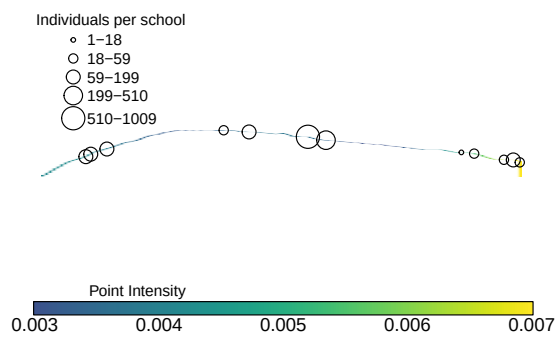
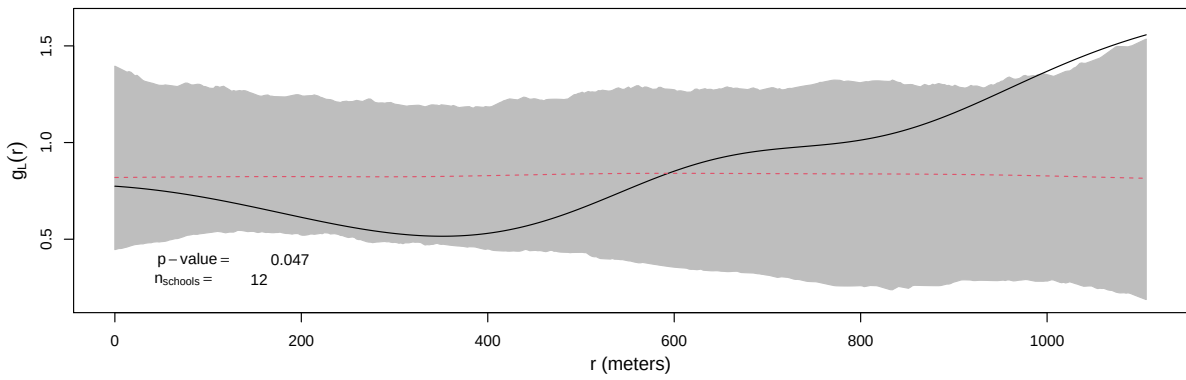
Cuckold Creek – 18 Jul 2017



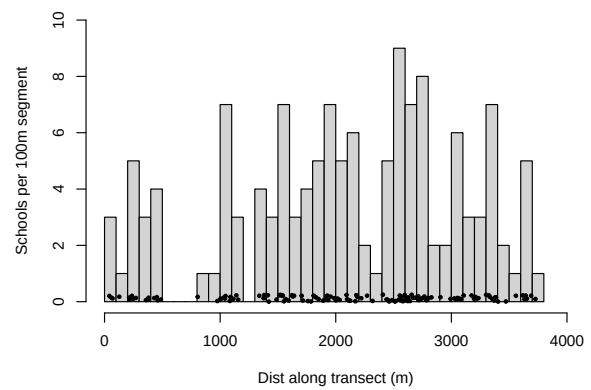
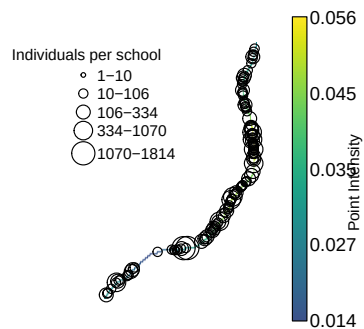
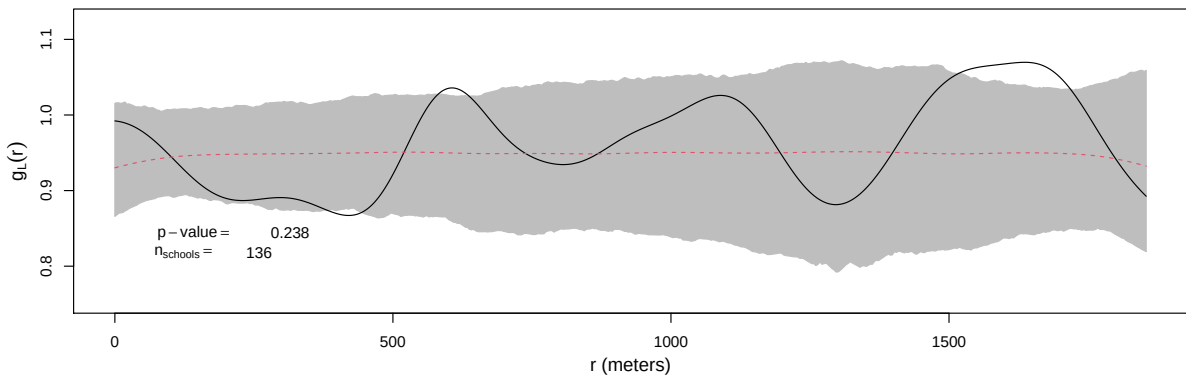
Hellen Creek – 18 Jul 2017



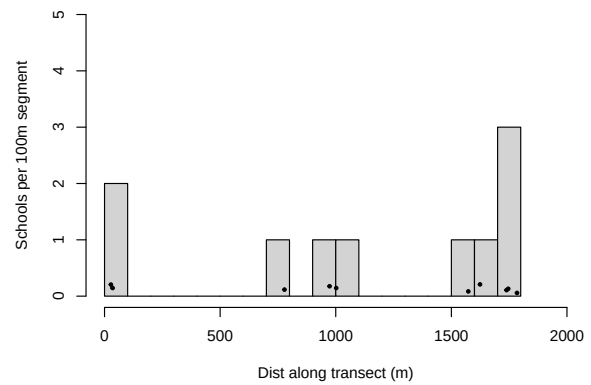
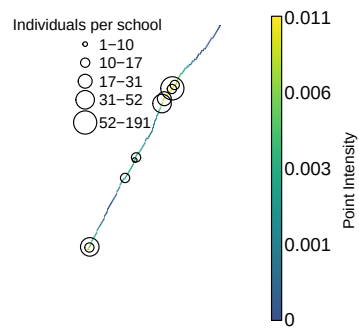
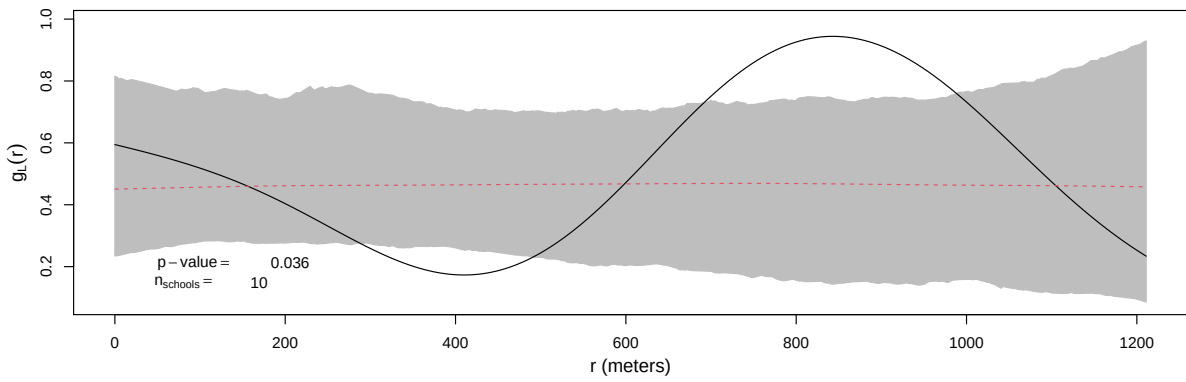
South Patuxent – 18 Jul 2017



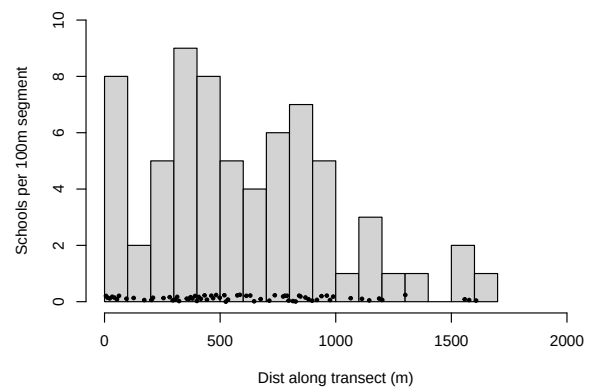
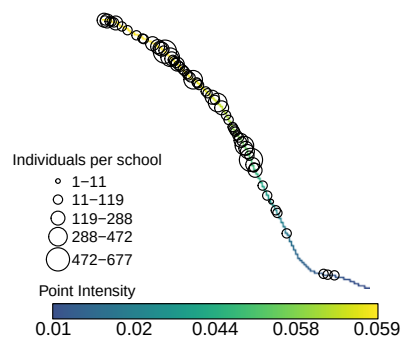
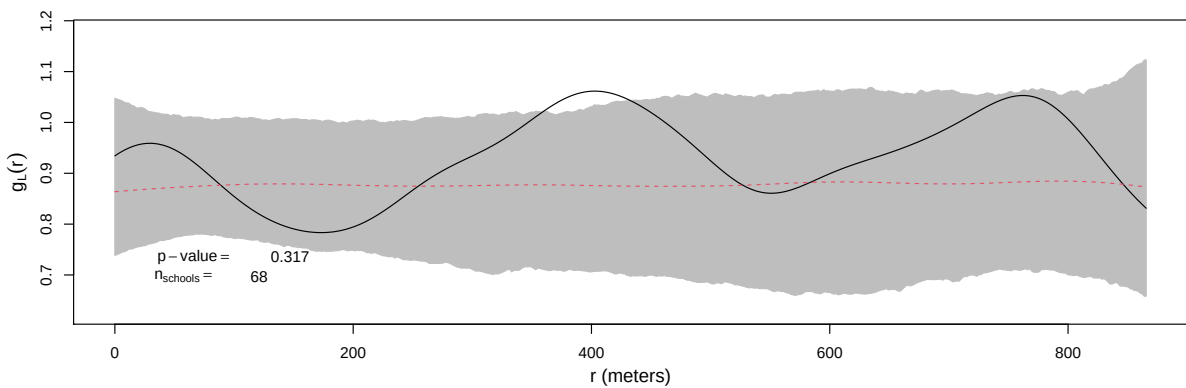
St. Leonard Creek – 18 Jul 2017



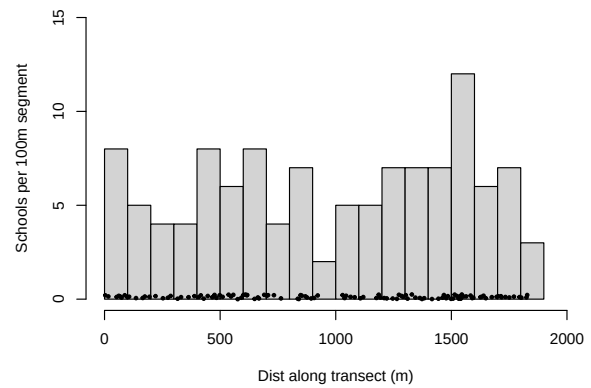
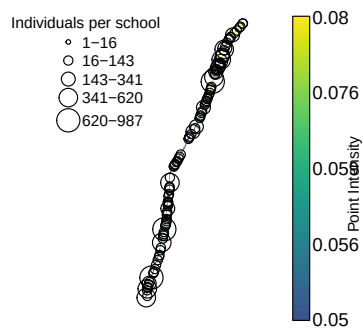
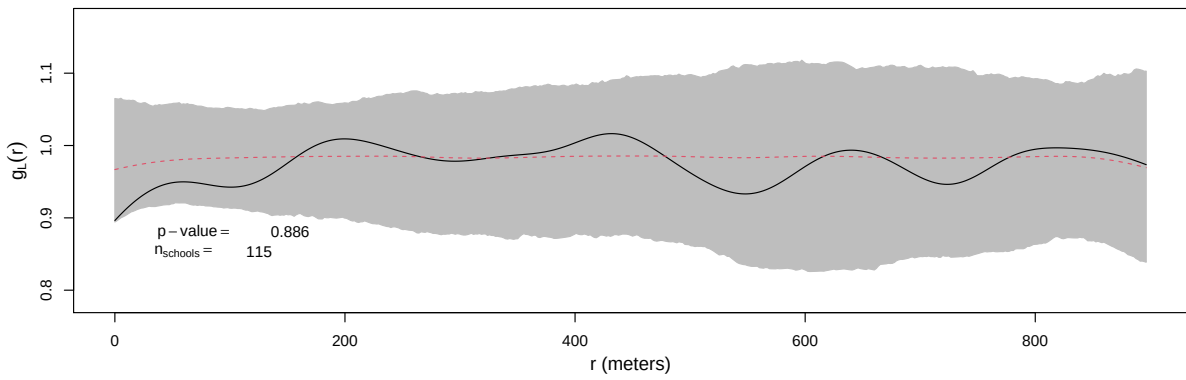
North Patuxent – 18 Jul 2017



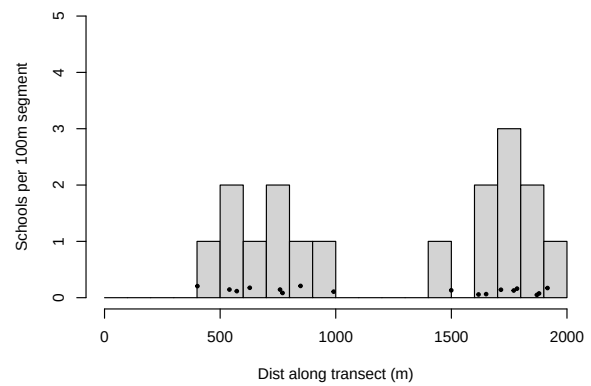
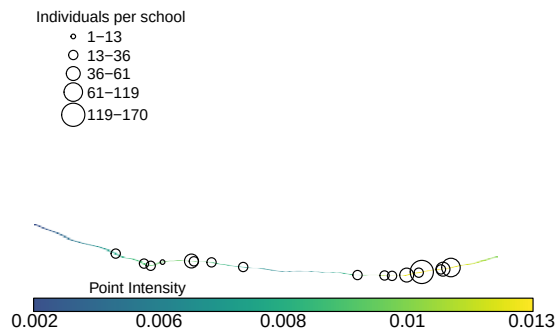
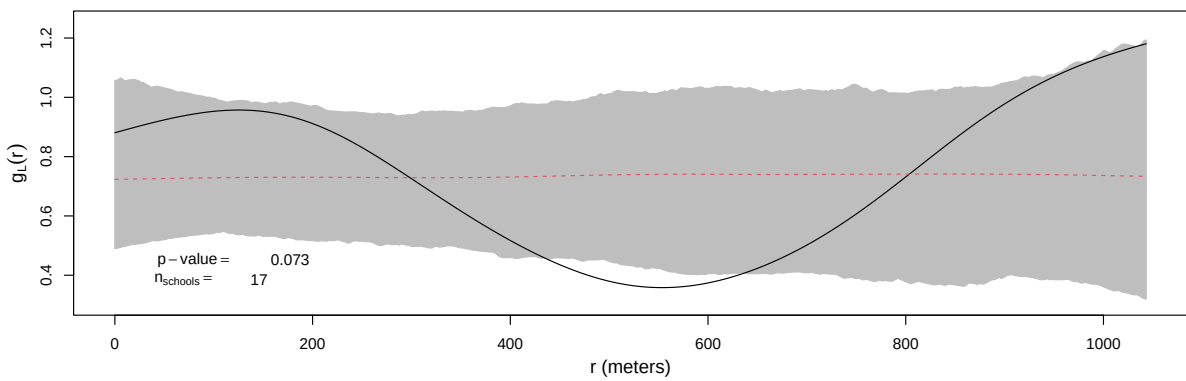
Cuckold Creek – 24 Jul 2017



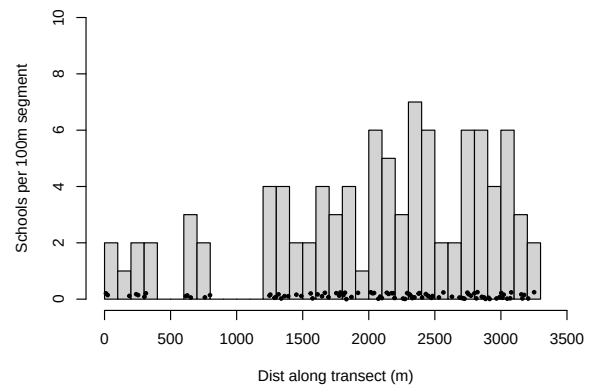
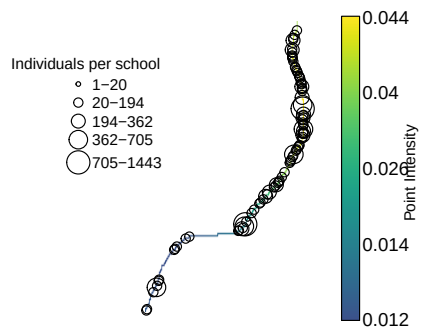
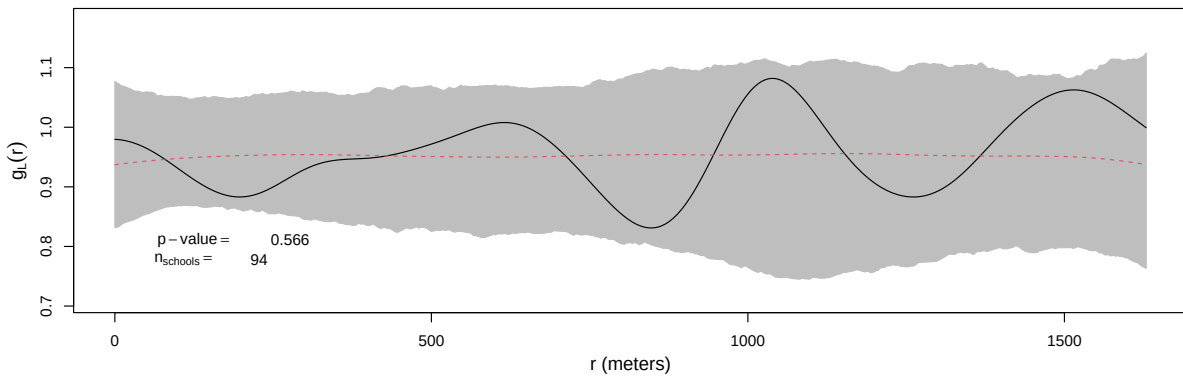
Hellen Creek – 24 Jul 2017



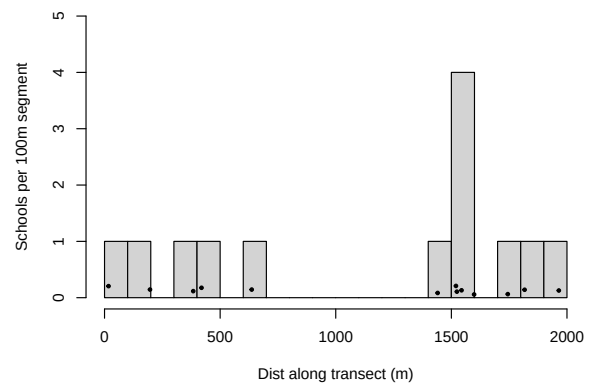
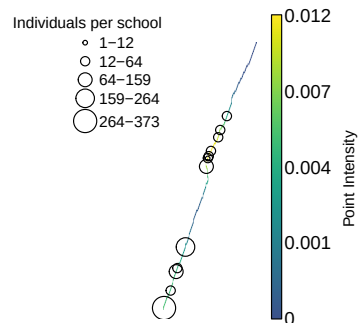
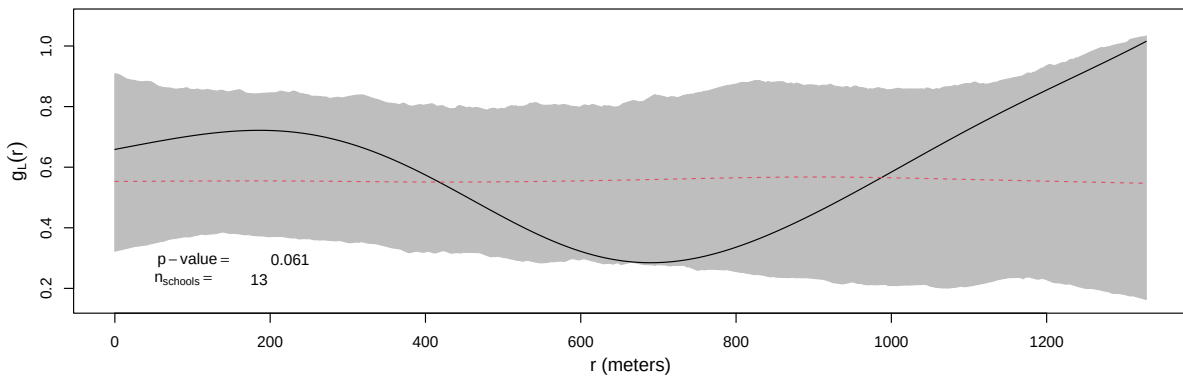
South Patuxent – 24 Jul 2017



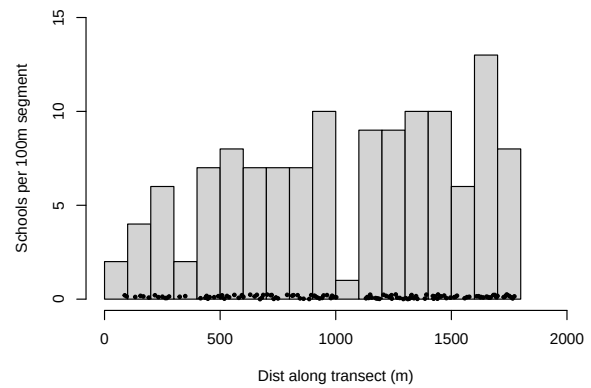
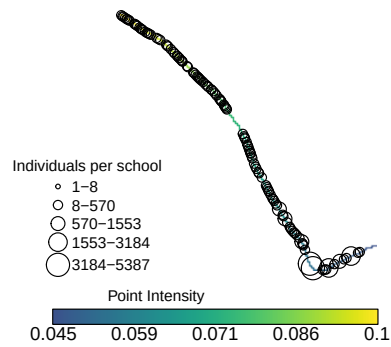
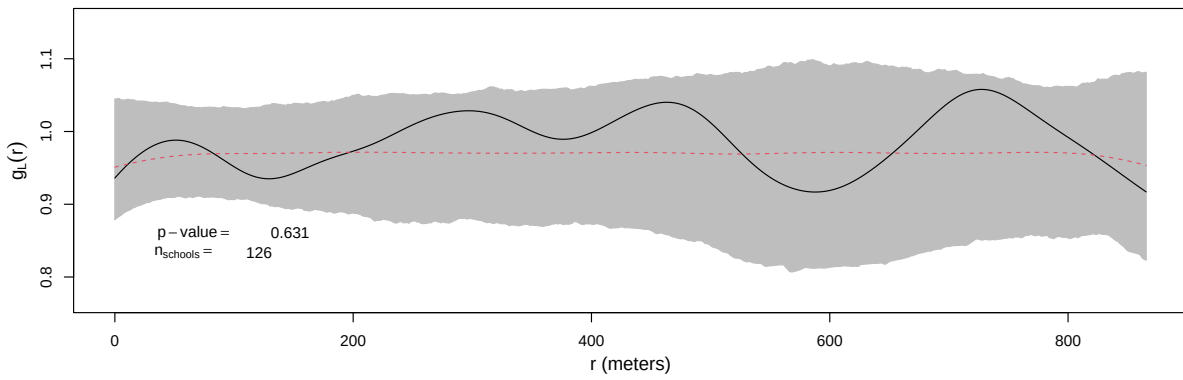
St. Leonard Creek – 24 Jul 2017



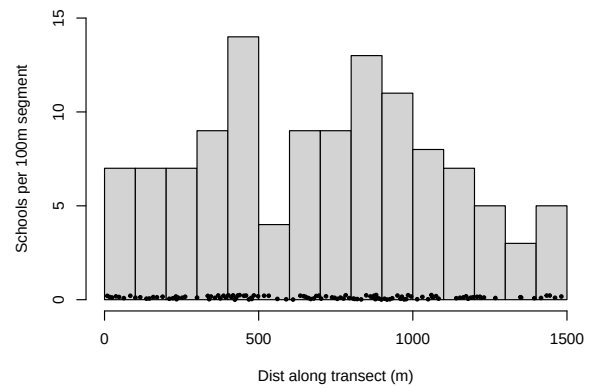
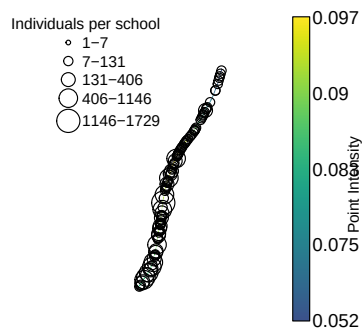
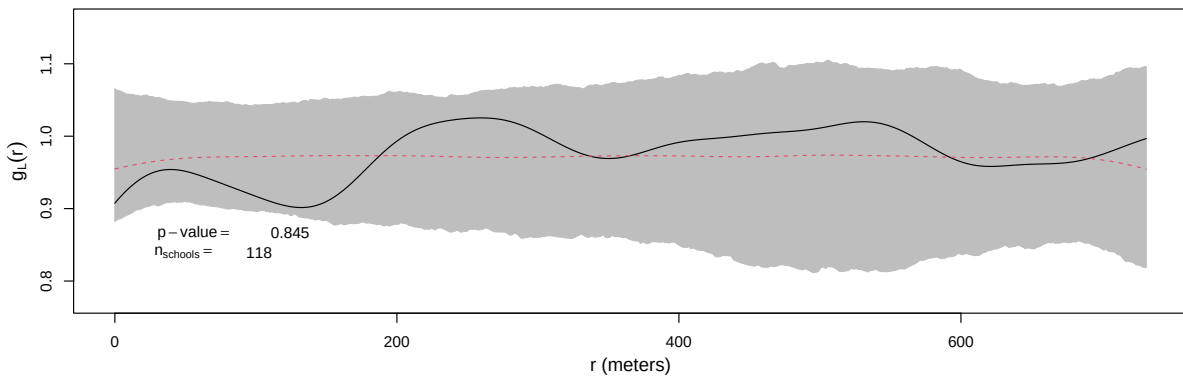
North Patuxent – 24 Jul 2017



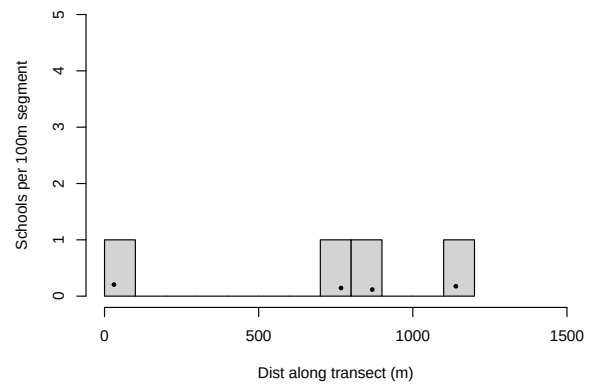
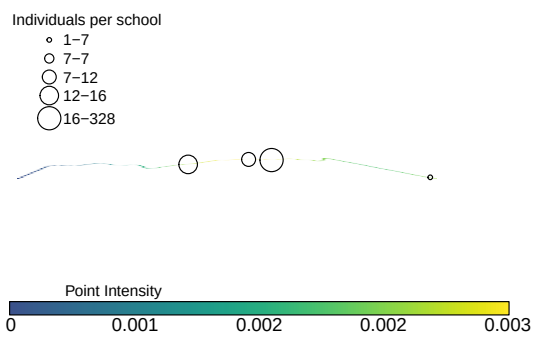
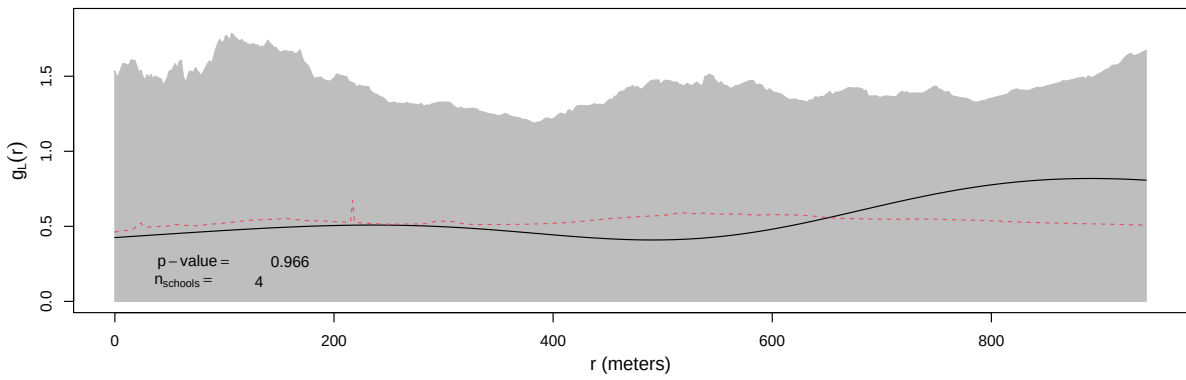
Cuckold Creek – 11 Aug 2017



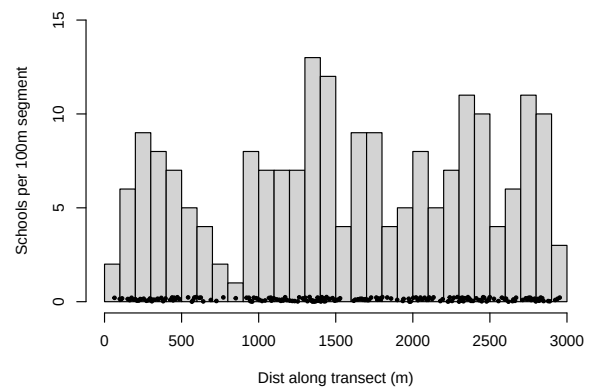
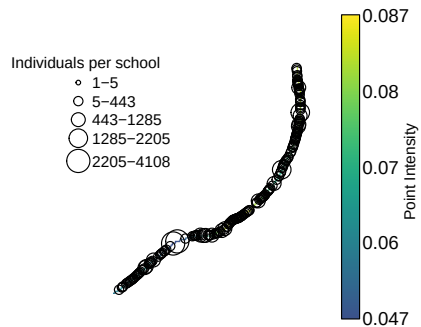
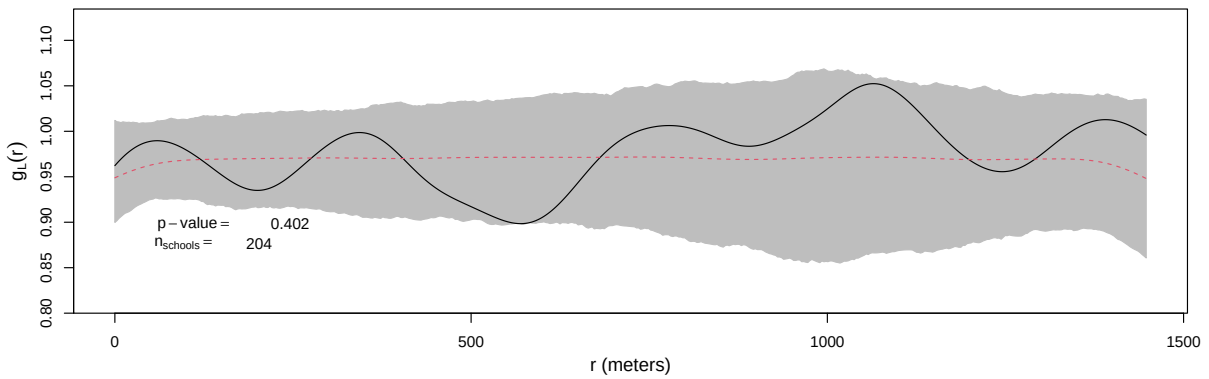
Hellen Creek – 11 Aug 2017



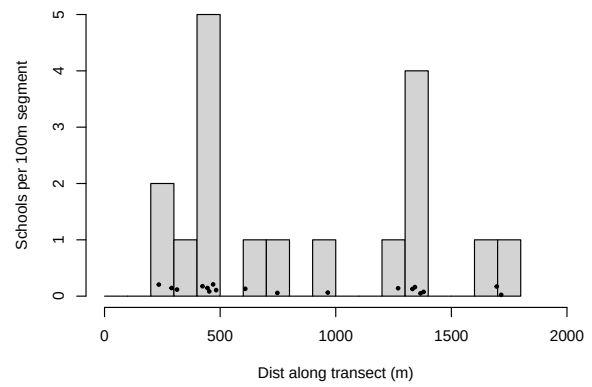
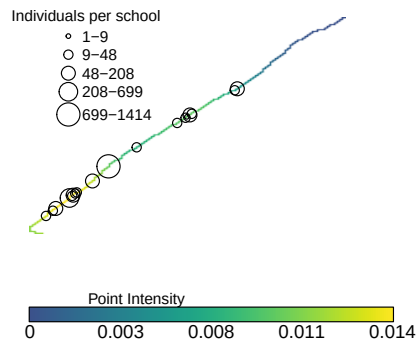
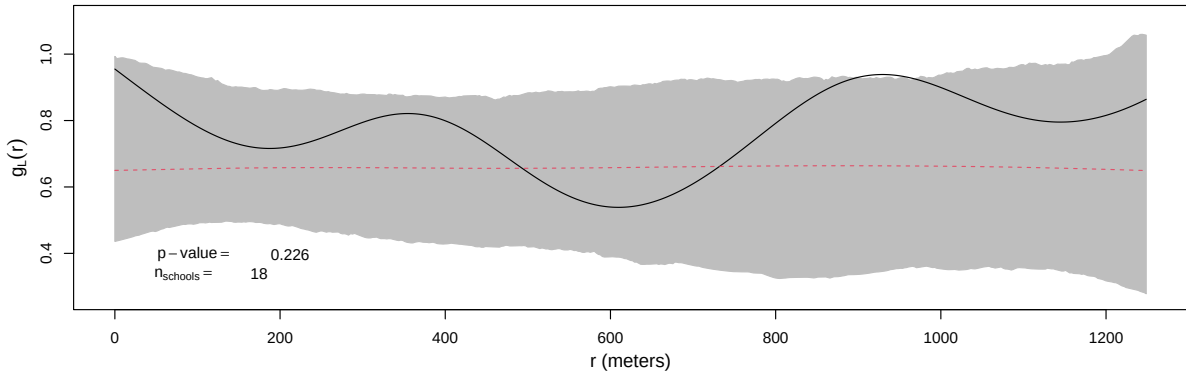
South Patuxent – 11 Aug 2017



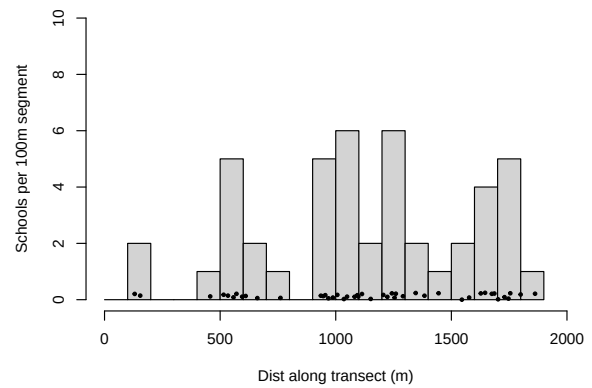
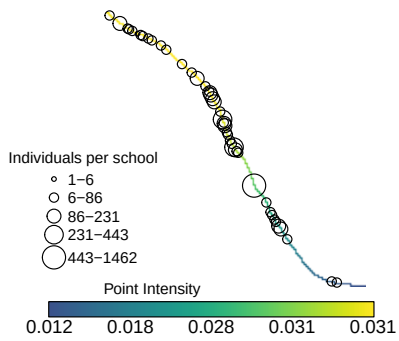
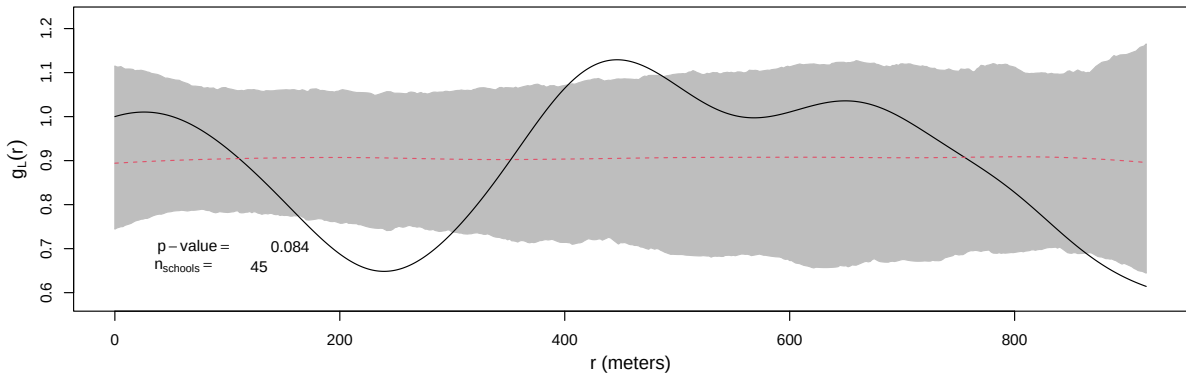
St. Leonard Creek – 11 Aug 2017



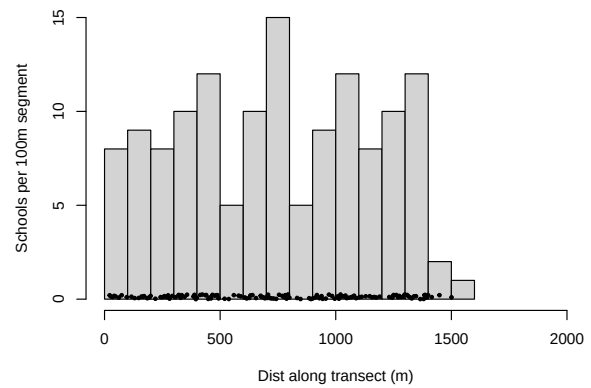
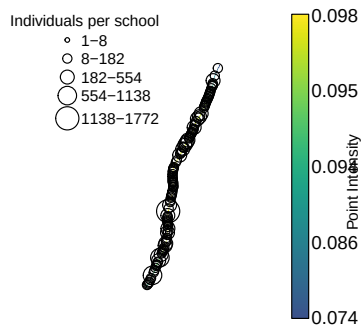
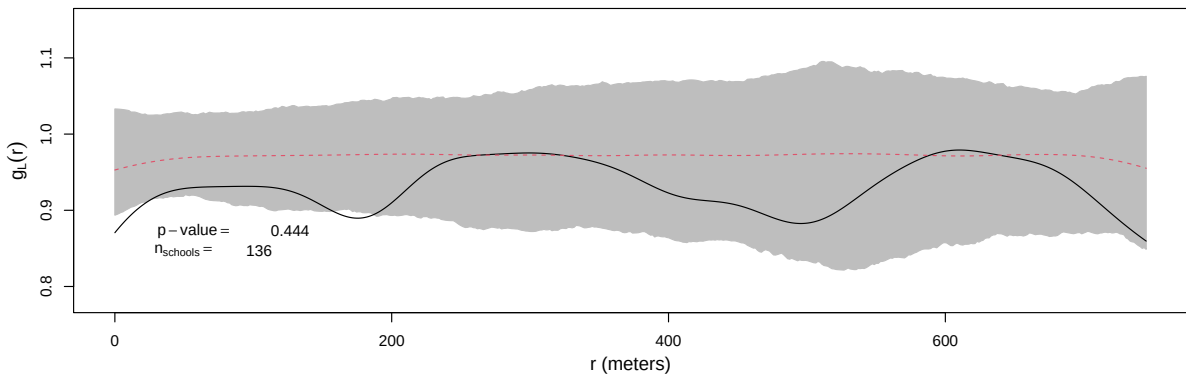
North Patuxent – 11 Aug 2017



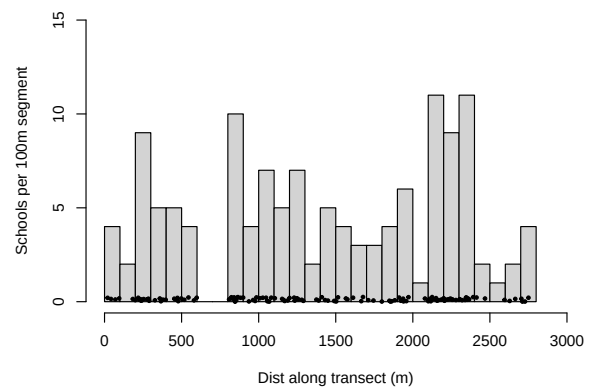
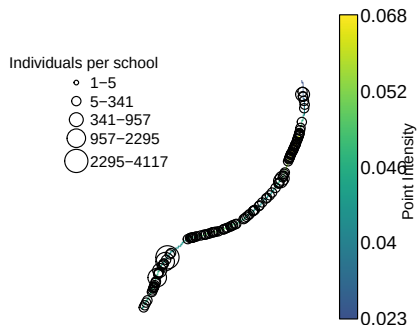
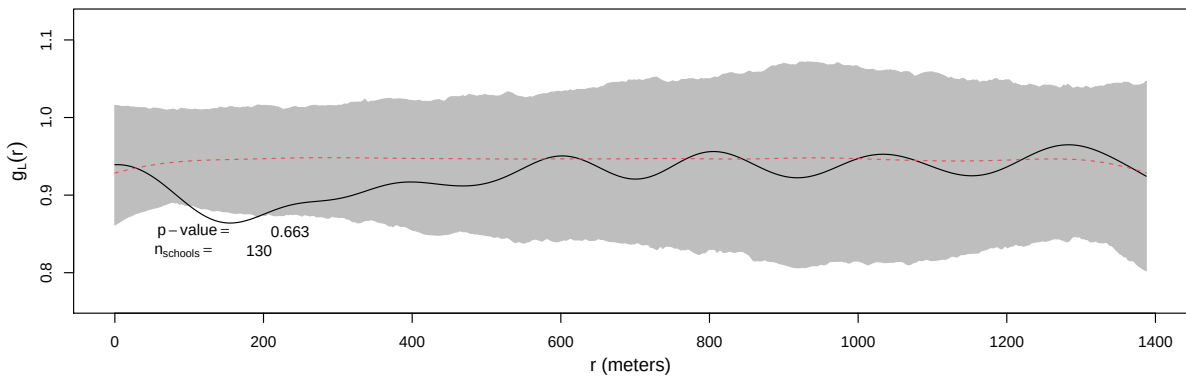
Cuckold Creek – 31 Aug 2017



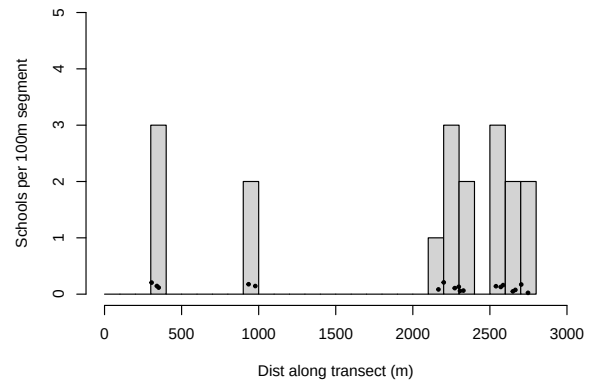
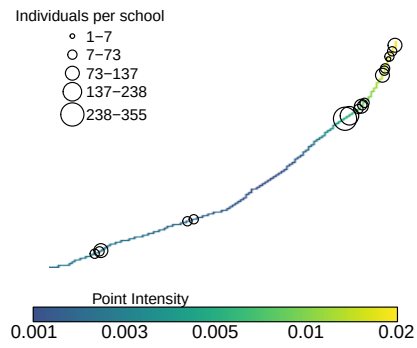
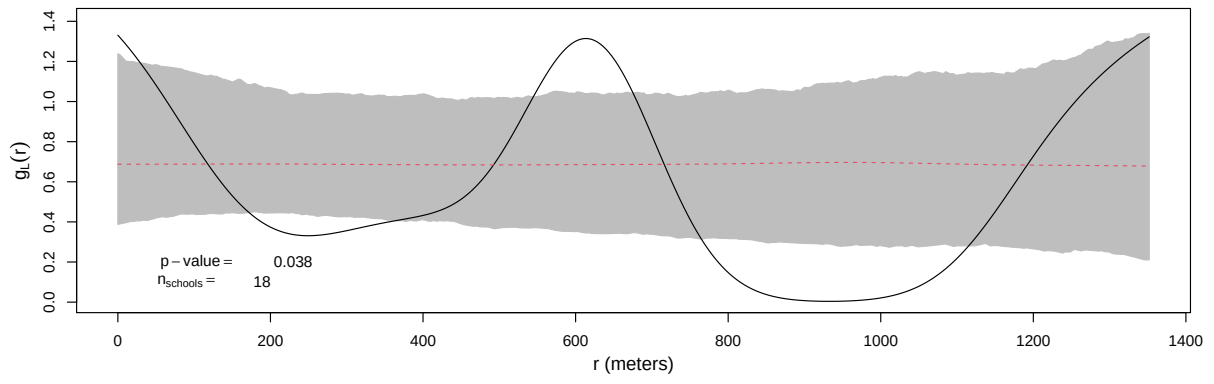
Hellen Creek – 31 Aug 2017



St. Leonard Creek – 31 Aug 2017

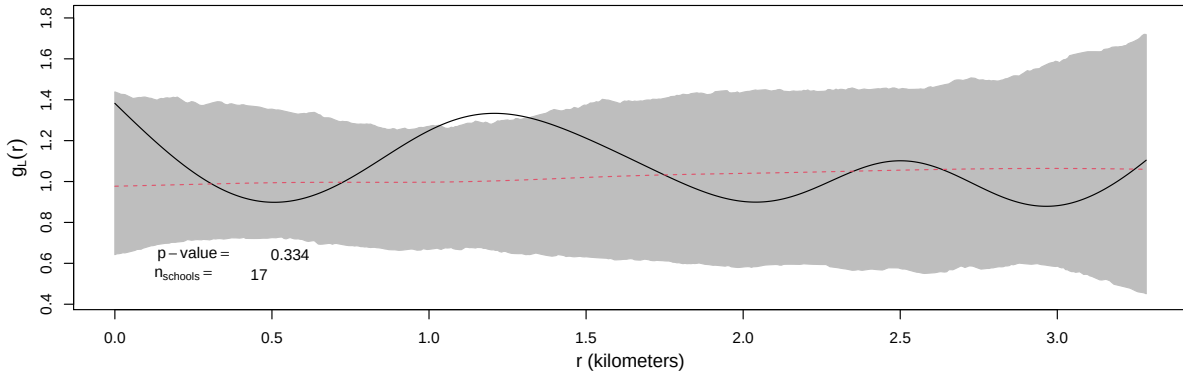


North Patuxent – 31 Aug 2017

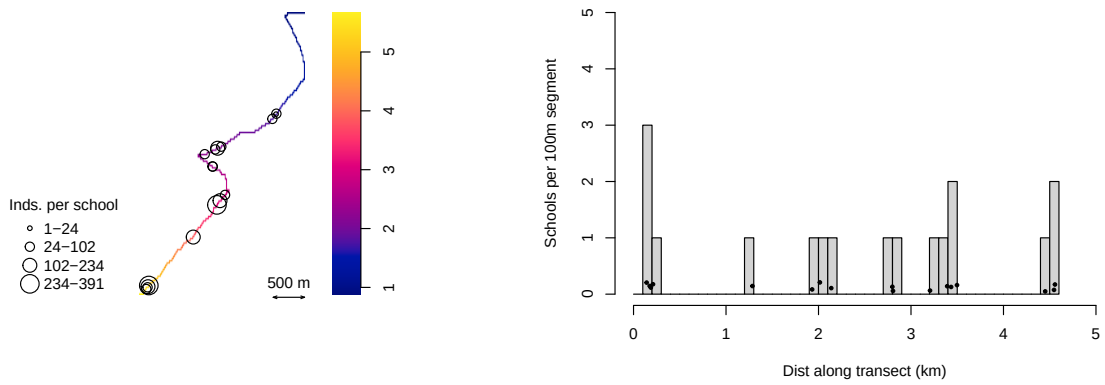


Appendix B: Linear networks and PCFs for all combined-extent transects

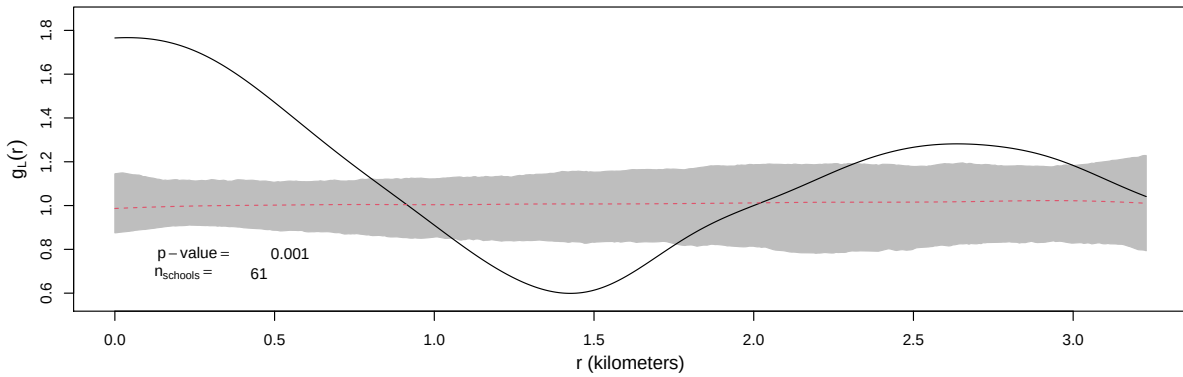
PAXSUR20160714



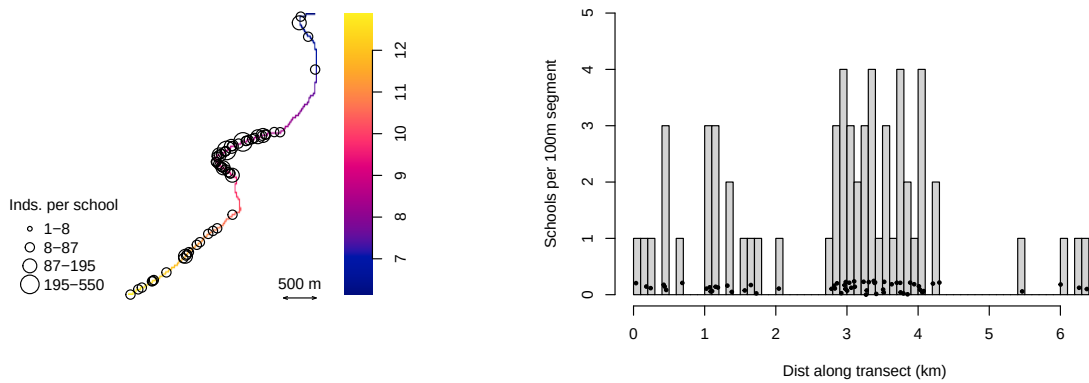
Inhomogeneous Poisson Process: Y-Direction



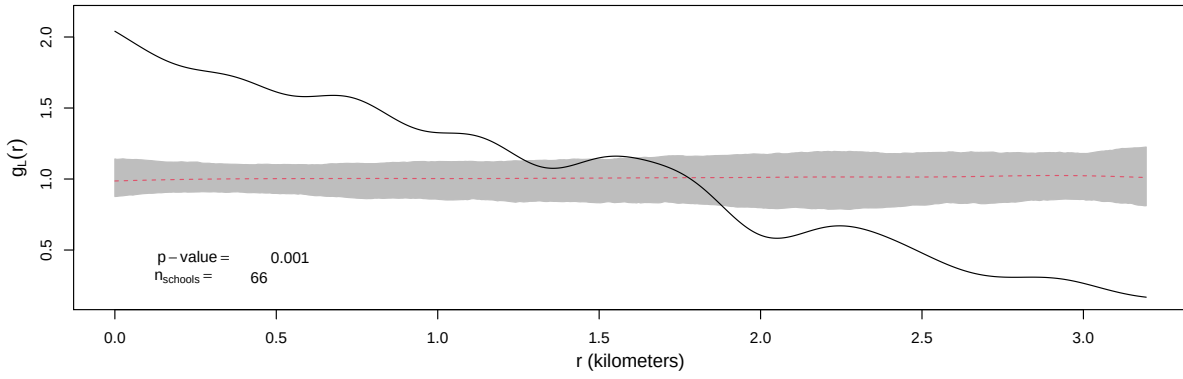
PAXSUR20160721



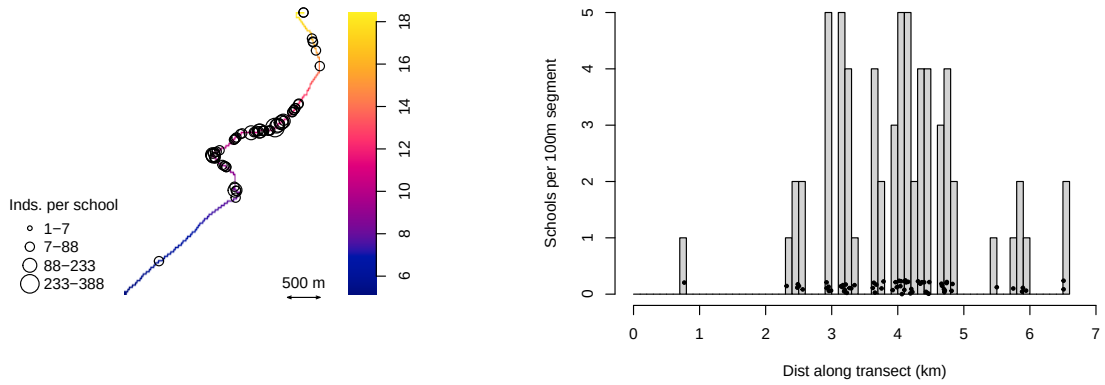
Inhomogeneous Poisson Process: Y-Direction



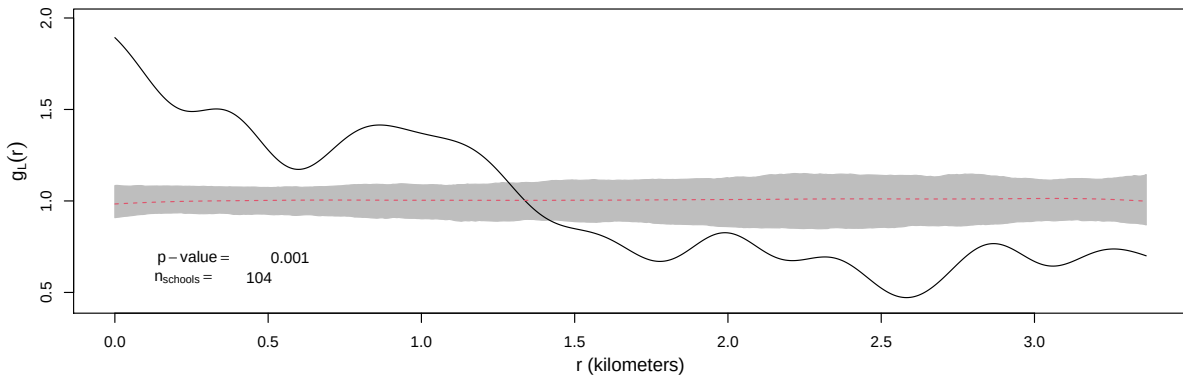
PAXSUR20160728



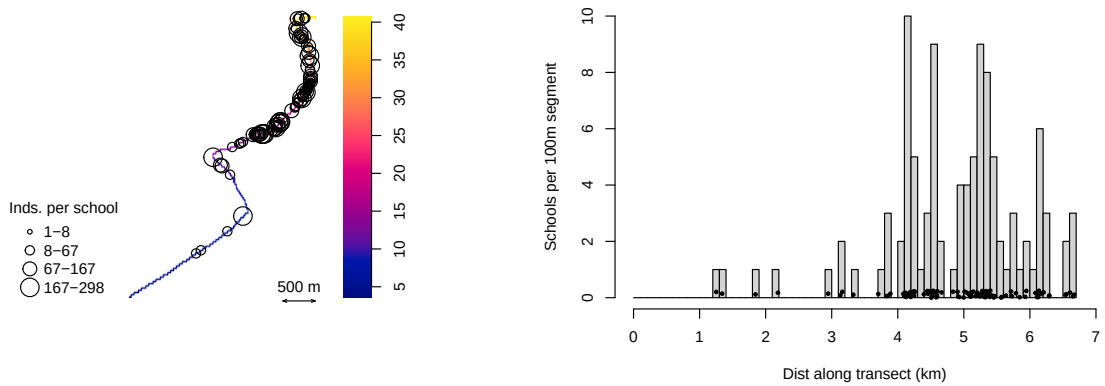
Inhomogeneous Poisson Process: Y-Direction



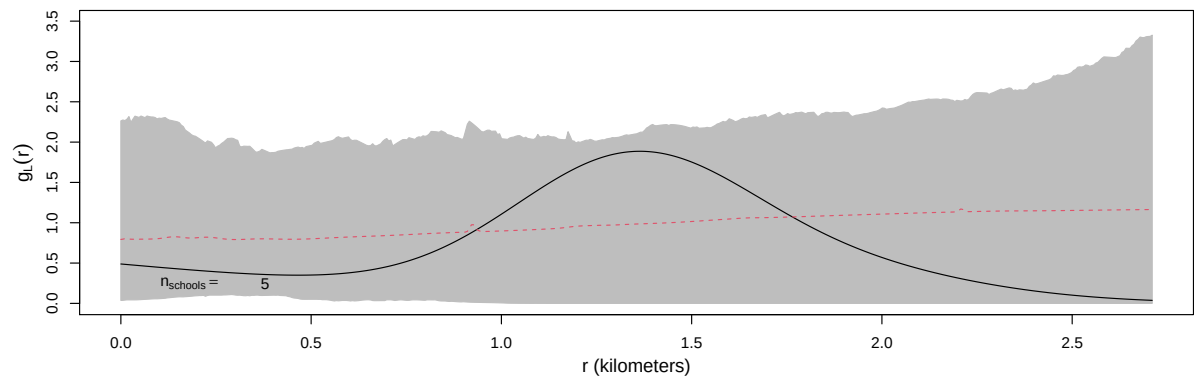
PAXSUR20160826



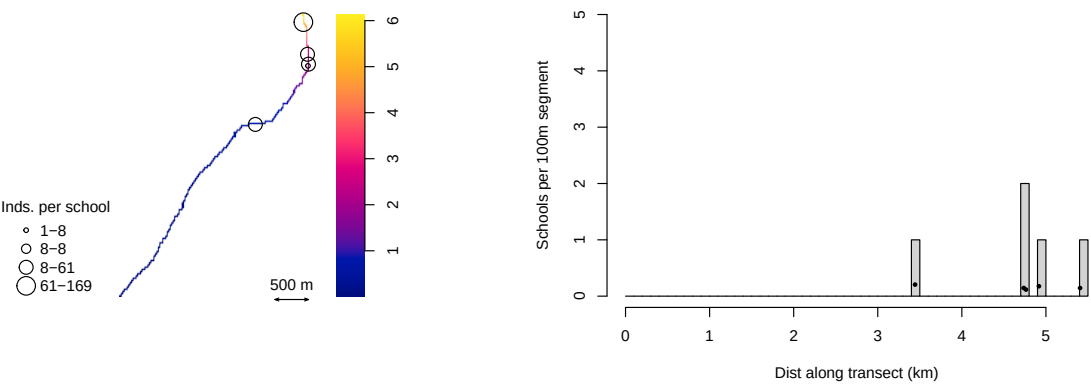
Inhomogeneous Poisson Process: Y-Direction



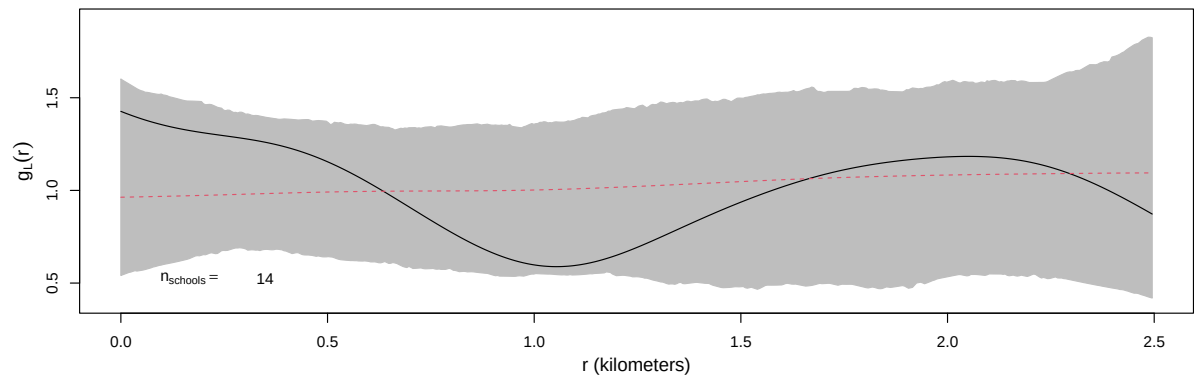
PAXSUR20170608



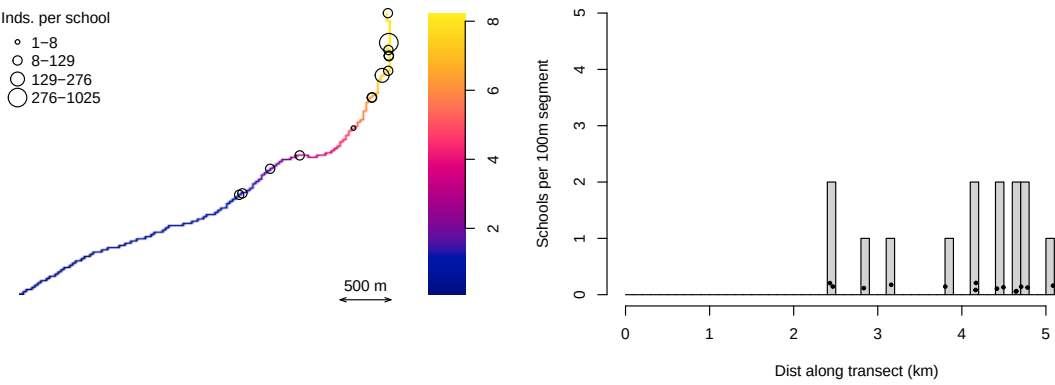
Inhomogeneous Poisson Process: Y-Direction



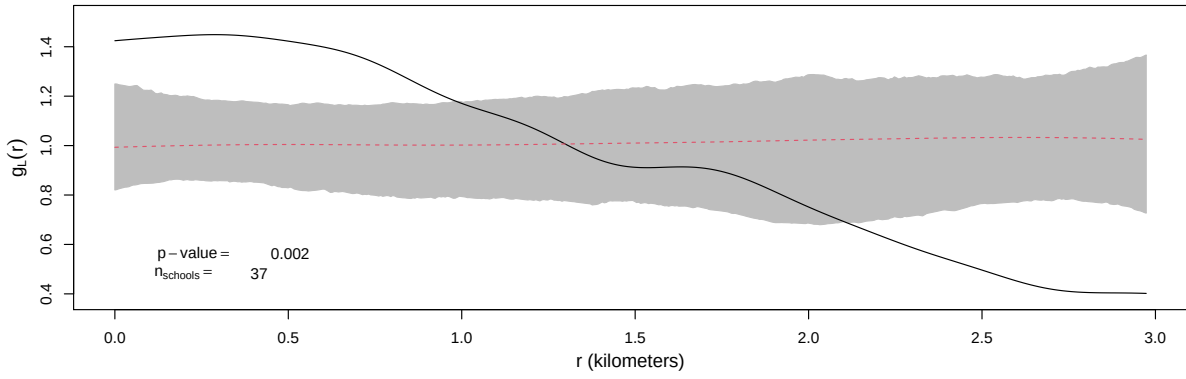
PAXSUR20170616



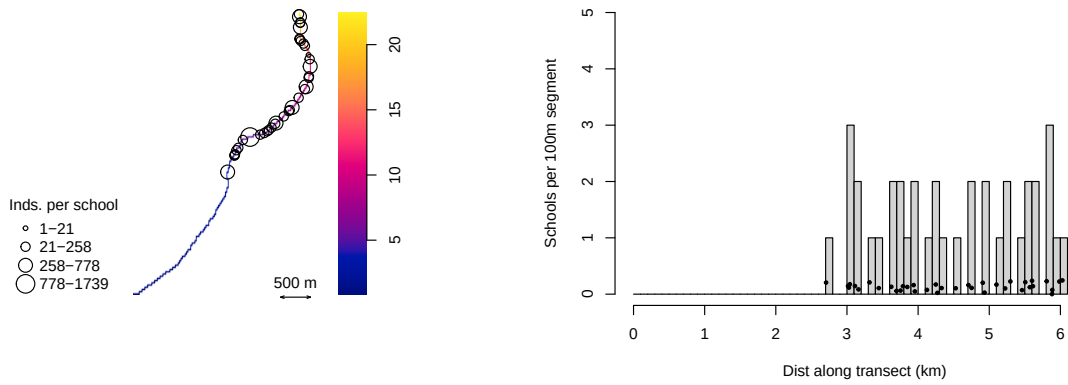
Inhomogeneous Poisson Process: X-Direction



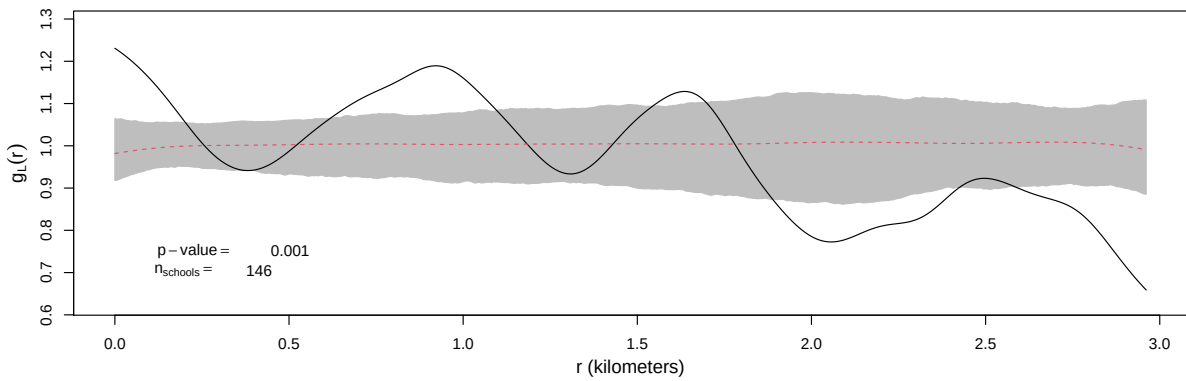
PAXSUR20170627



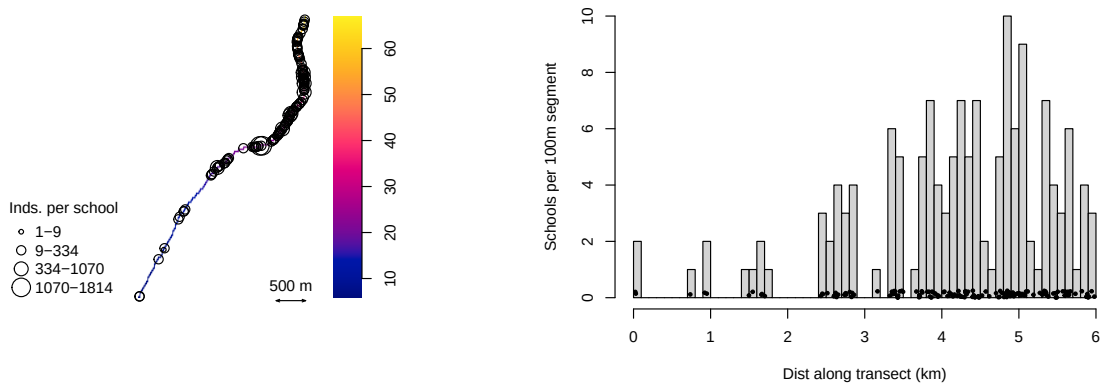
Inhomogeneous Poisson Process: Y-Direction



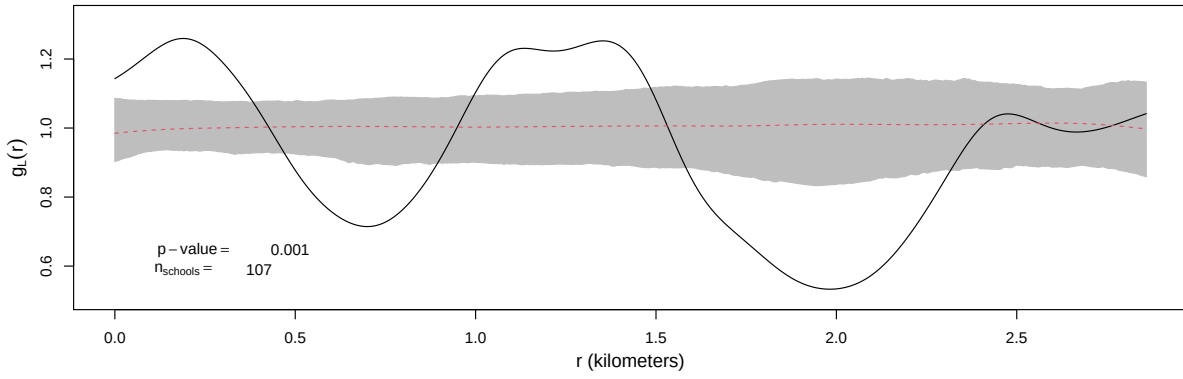
PAXSUR20170718



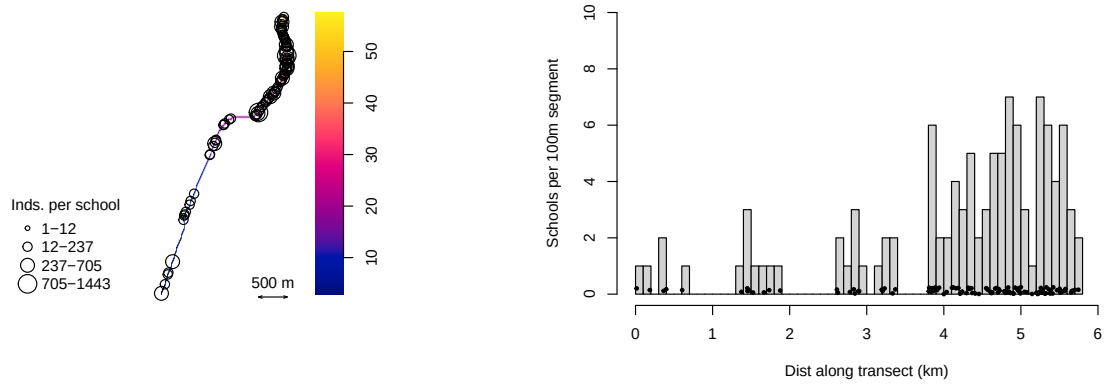
Inhomogeneous Poisson Process: Y-Direction



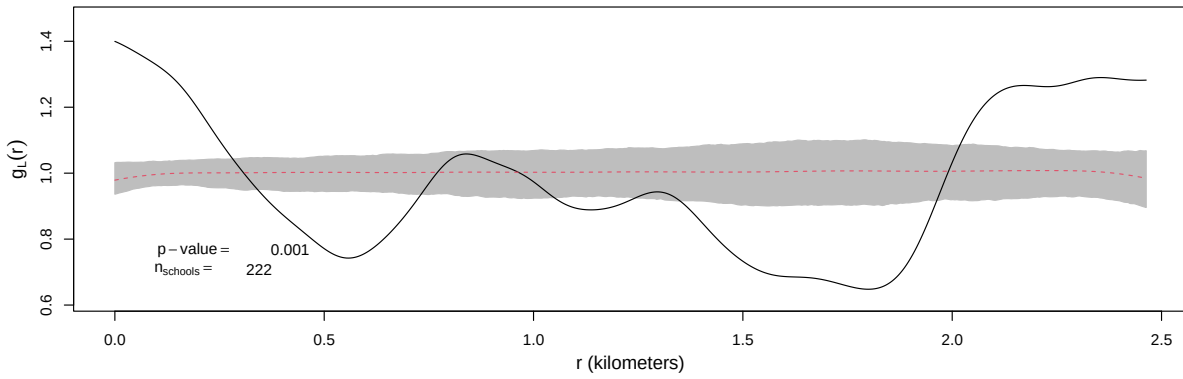
PAXSUR20170724



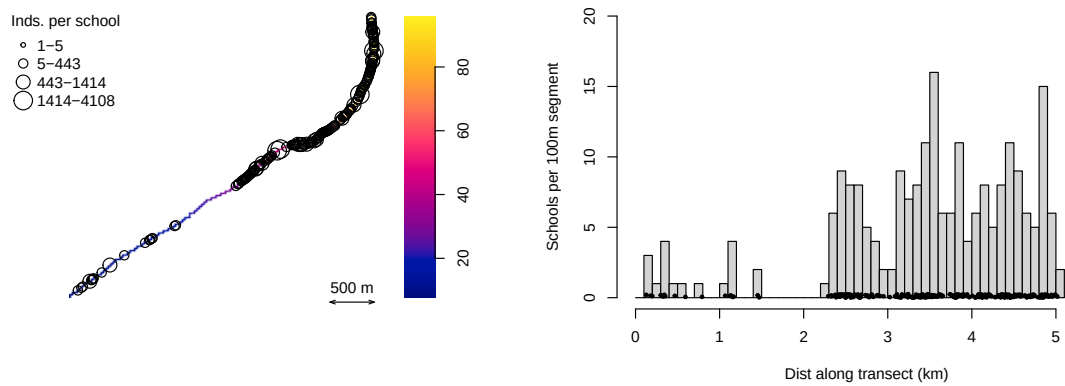
Inhomogeneous Poisson Process: Y-Direction

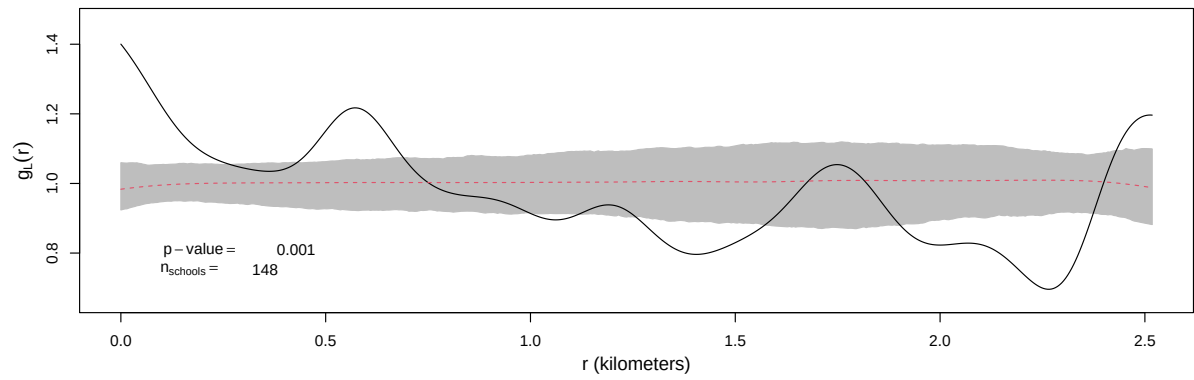


PAXSUR20170811



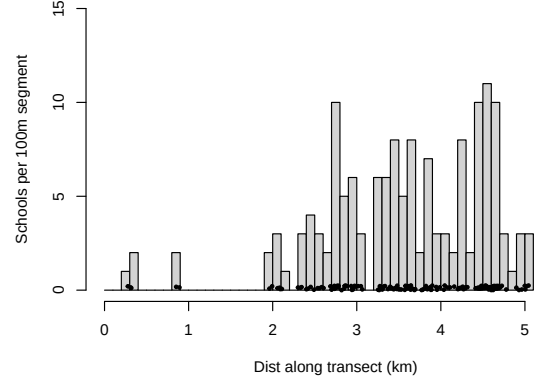
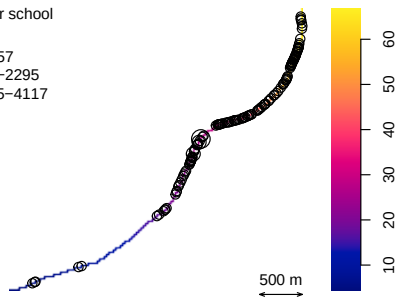
Inhomogeneous Poisson Process: X-Direction





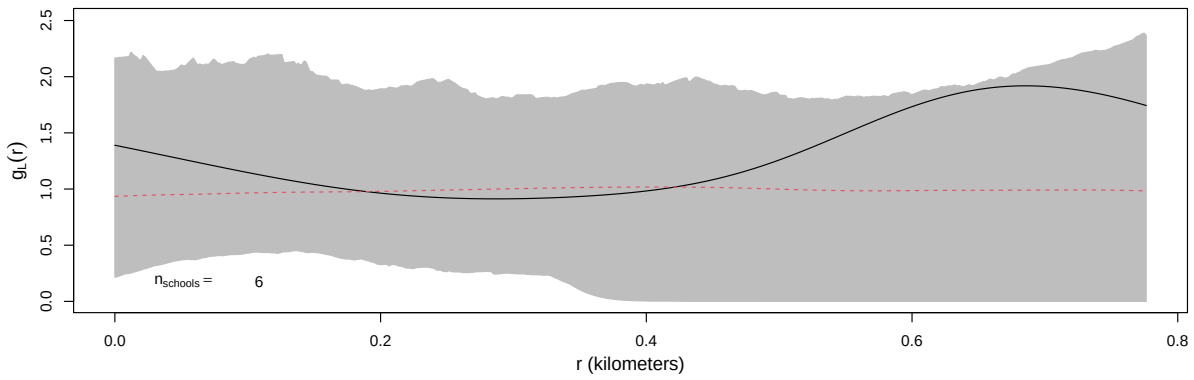
Inhomogeneous Poisson Process: X-Direction

Inds. per school
○ 1-5
○ 5-957
○ 957-2295
○ 2295-4117

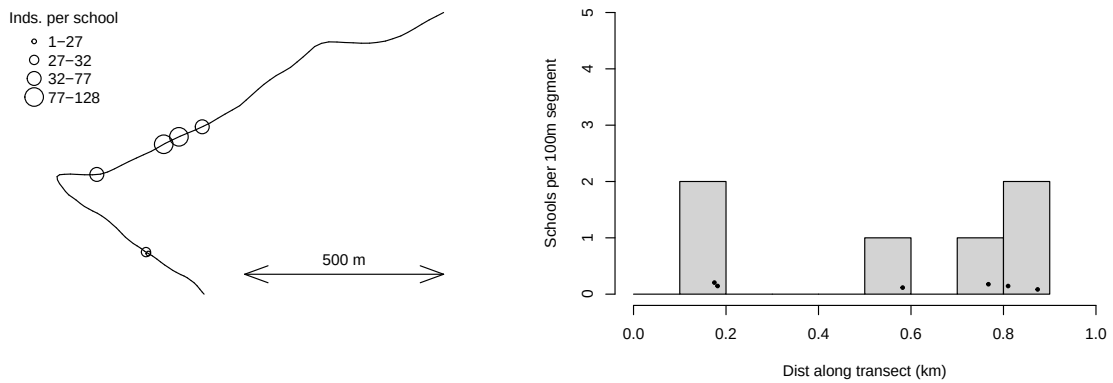


Appendix C: Linear networks and PCFs for all shorter-extent transects

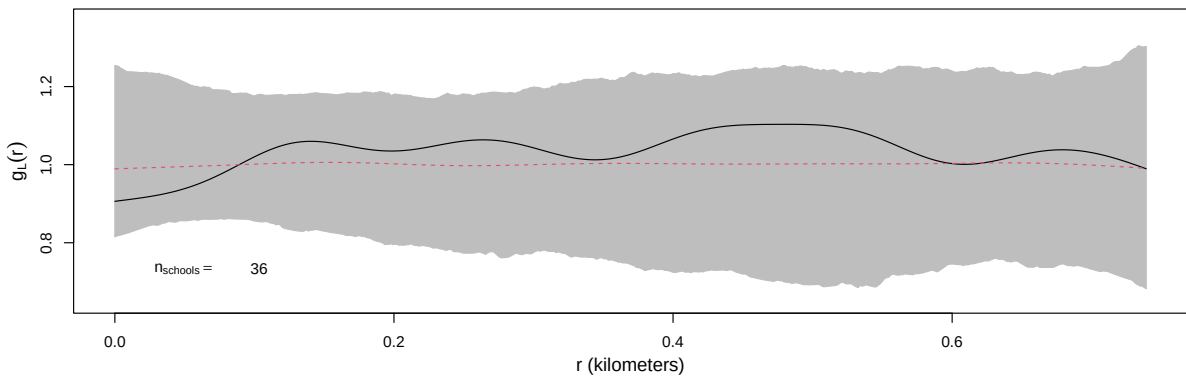
14 Jul 2016



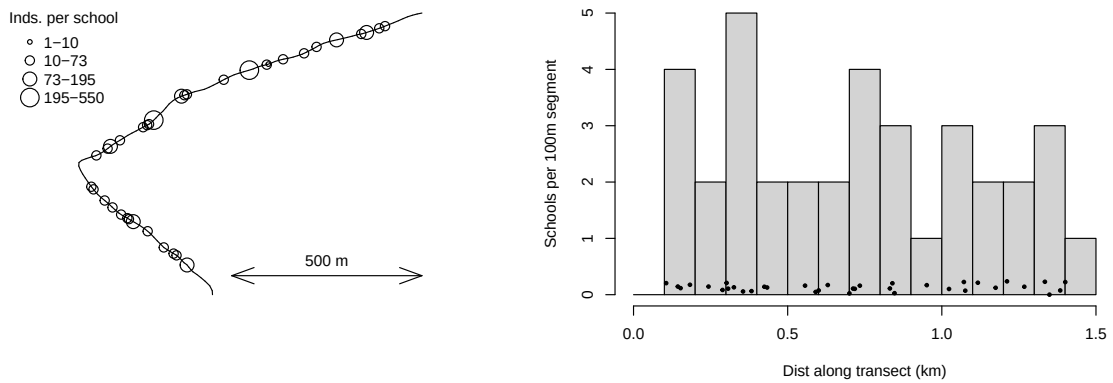
Homogeneous Poisson Process



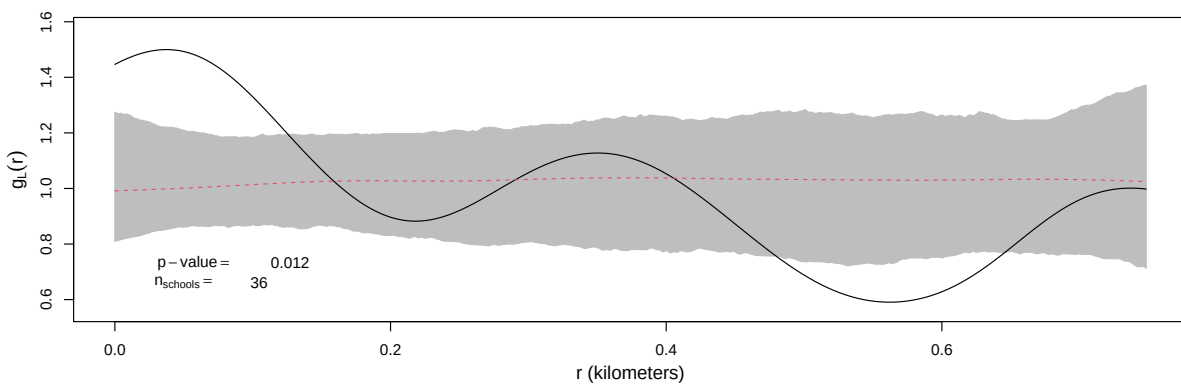
21 Jul 2016



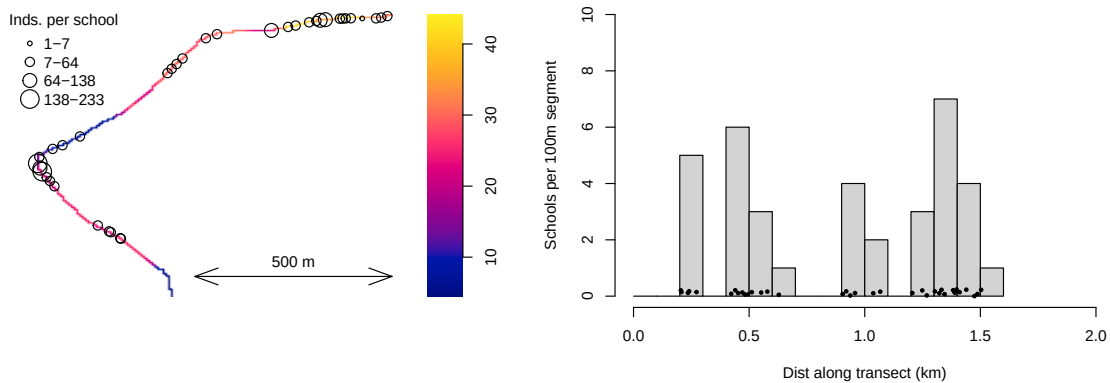
Homogeneous Poisson Process



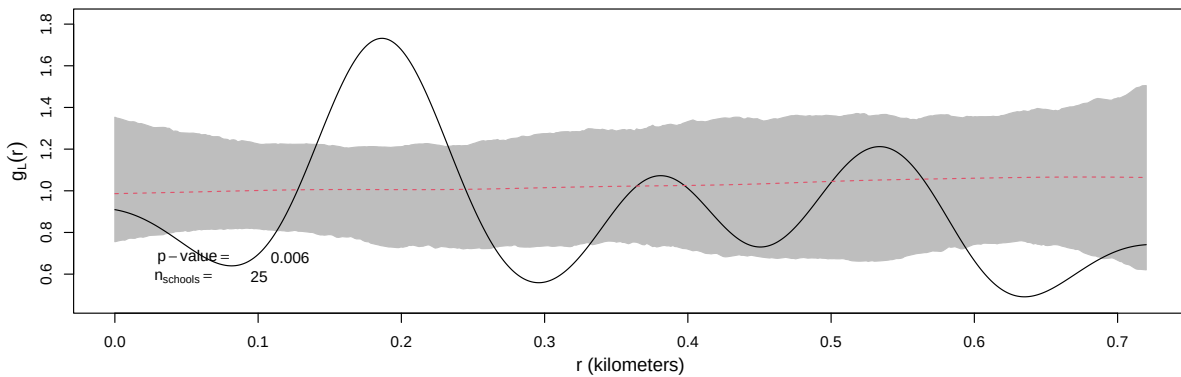
28 Jul 2016



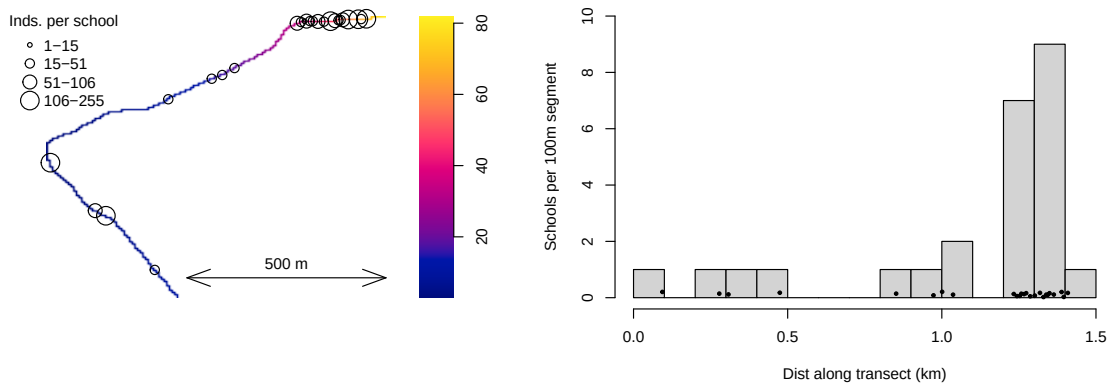
Inhomogeneous Poisson Process: Depth



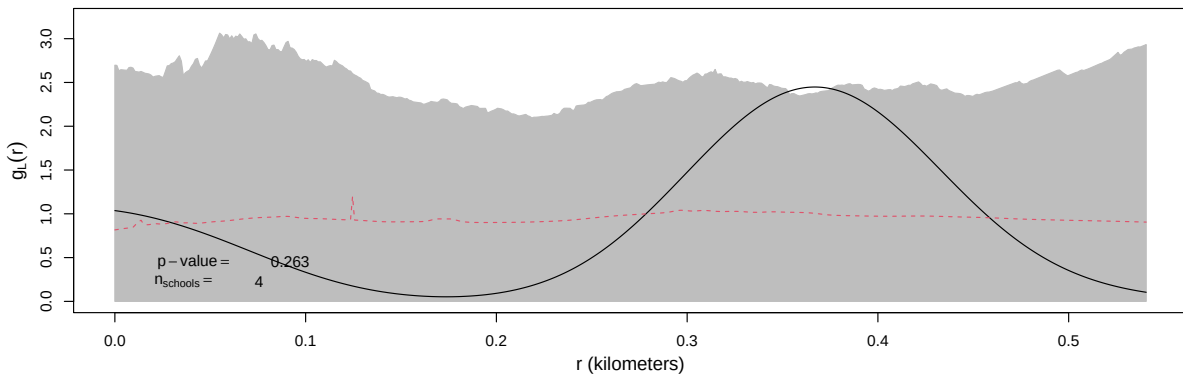
26 Aug 2016



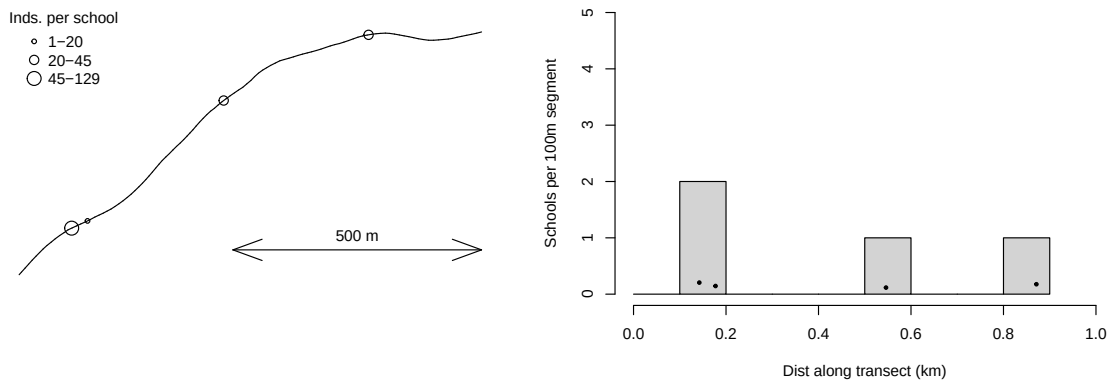
Inhomogeneous Poisson Process: X-Direction



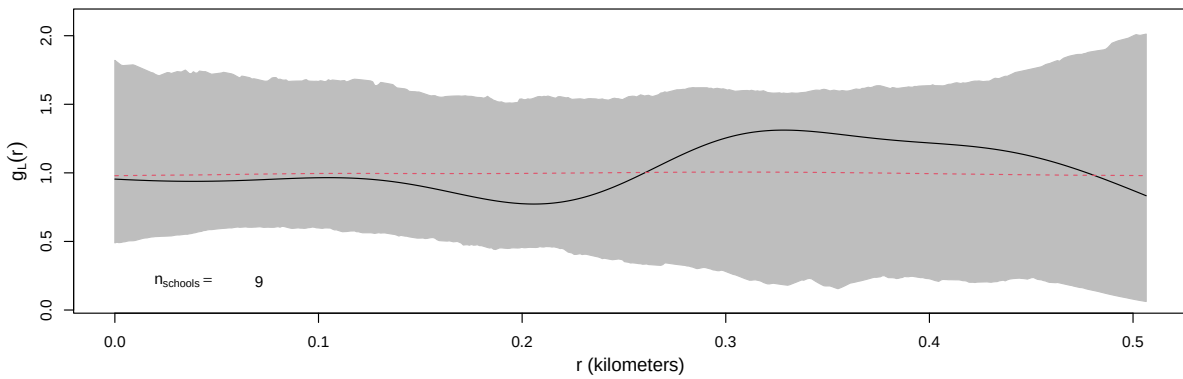
16 Jun 2017



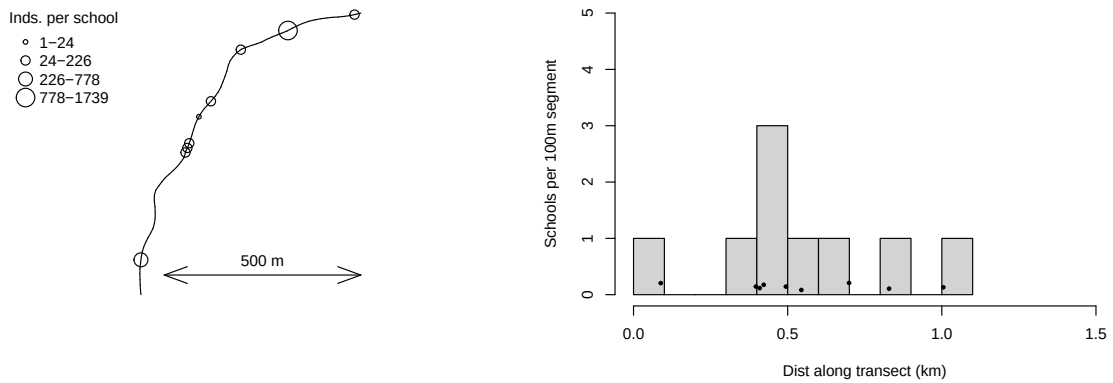
Homogeneous Poisson Process



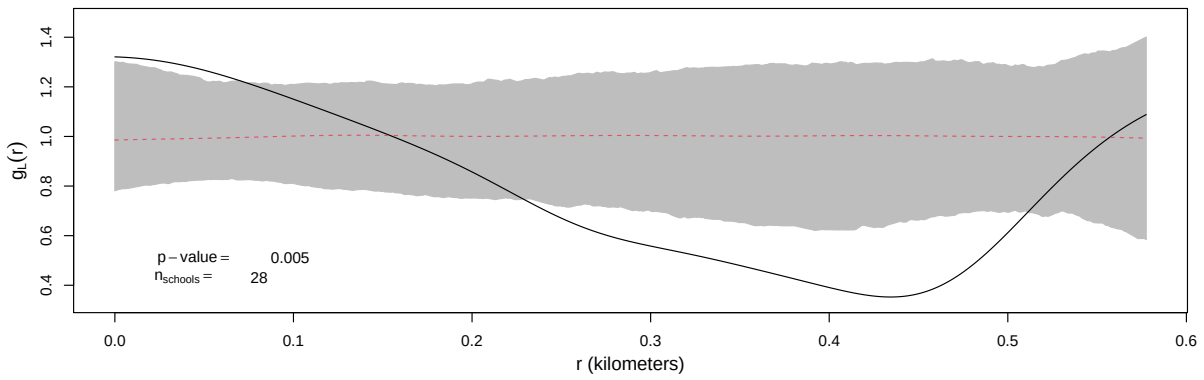
27 Jun 2017



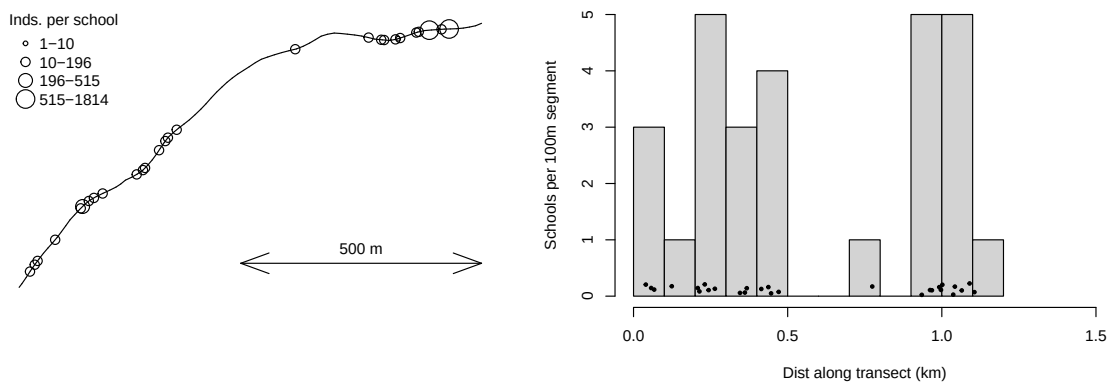
Homogeneous Poisson Process



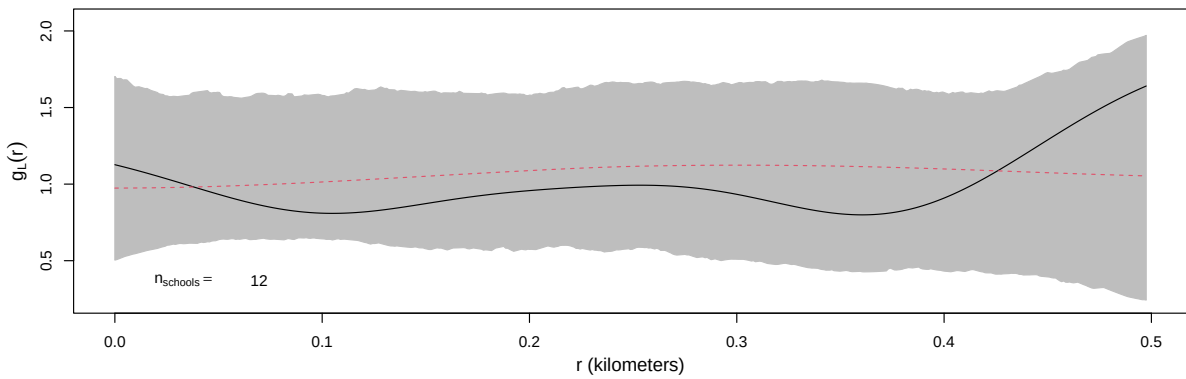
18 Jul 2017



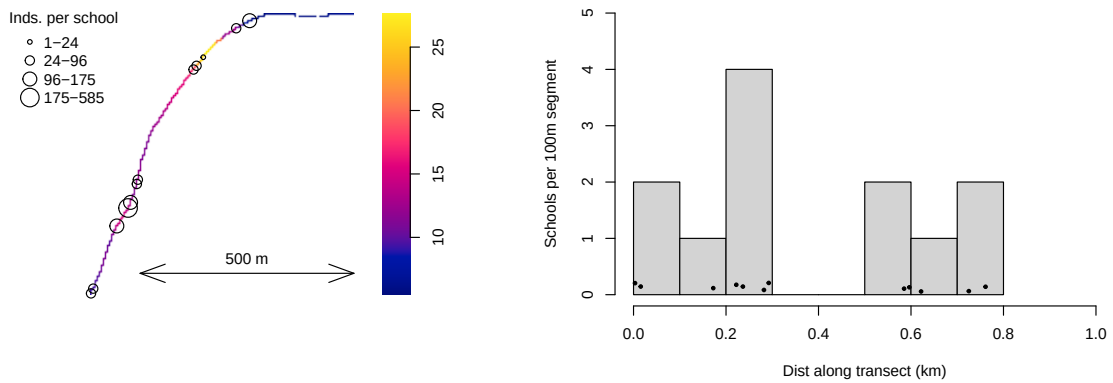
Homogeneous Poisson Process



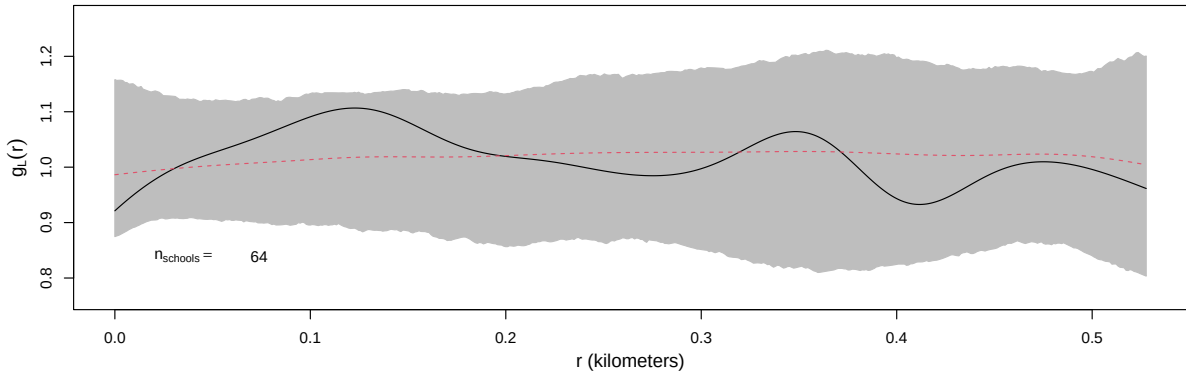
24 Jul 2017



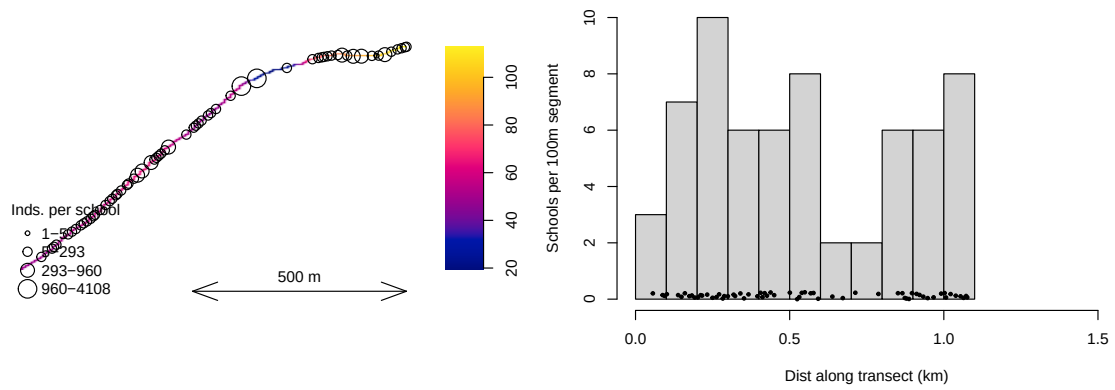
Inhomogeneous Poisson Process: Depth



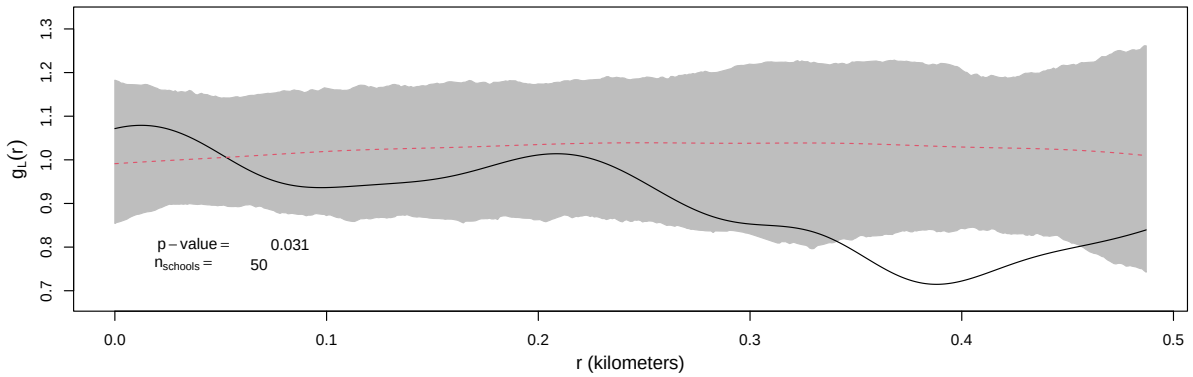
PAXSUR20170811



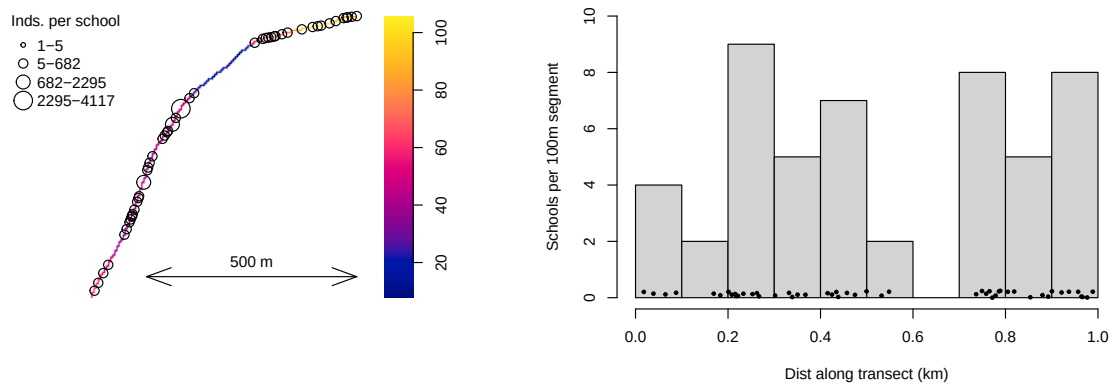
Inhomogeneous Poisson Process: Depth



31 Aug 2017



Inhomogeneous Poisson Process: Depth



Appendix D: Code for IBMs

Preparing SCHISM-ICM netcdf files

Katie Lankowicz

24 May 2022

The Chesapeake Bay Program model working group (headed by Richard Tian) have provided me with output from a Semi-implicit Cross-scale Hydroscience Integrated System Model - Integrated Compartment Model. This is a three-dimensional unstructured grid that simulates environmental variables in the Corsica River in 2006. We will use these data to create dynamic environmental conditions that will be used in an individual-based model (IBM) of menhaden movement. Extensive pre-processing is needed before we can put these data into our IBM, which we hope will closely resemble an idealized shallow estuarine tidal tributary. This document will describe the pre-processing methods used.

Original grid space

The bounding box around the modeled portion of the Corsica is approximately 7400m x 5600m. The Corsica is similar to St. Leonard Creek, which is our field data collection area- the surrounding watershed is not highly developed. It is mostly agricultural land or forested buffers around lots. St. Leonard Creek is mostly forested land with some agricultural and residential area.

Grid cells in the model are triangles of varying shape and size. The cells meet at nodes, and environmental variables are simulated through time at these nodes. The high spatial resolution of the grid cells and high temporal resolution of simulated environmental variables is a good match for our proposed fine-scale IBM.



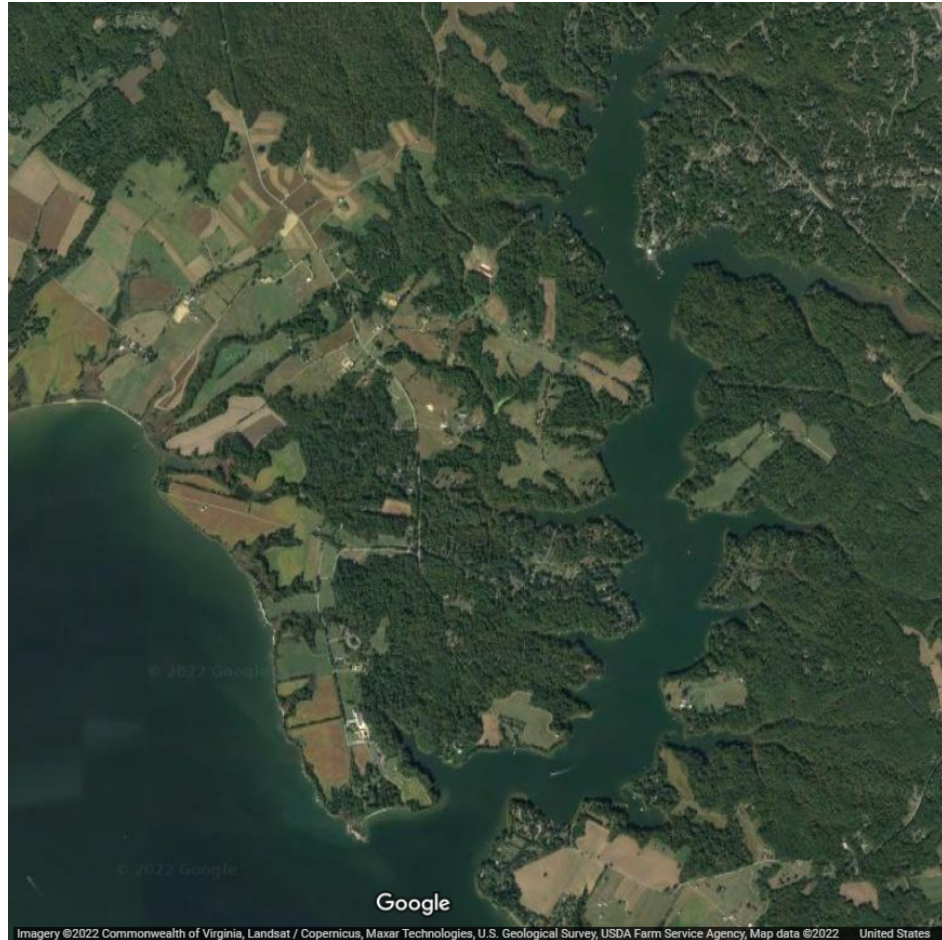


Figure 2: Land use surrounding St. Leonard Creek

Load data

The first step is to open the `netcdf` file and view the variables included.

```
nc_data <- nc_open(paste0('C:/Users/klank/OneDrive/',  
                          'Desktop/CBP_Corsica_SCHISM.nc'))  
names(nc_data$var)  
  
## [1] "ICM_21"          "ICM_3"           "ICM_4"  
## [4] "ICM_5"           "ICM_6"           "ICM_7"  
## [7] "ICM_Ch1"         "SCHISM_hgrid_face_x" "SCHISM_hgrid_face_y"  
## [10] "SCHISM_hgrid_node_x" "SCHISM_hgrid_node_y" "depth"  
## [13] "elev"            "salt"            "sigma_h_c"  
## [16] "sigma_theta_b"    "sigma_theta_f"    "temp"
```

The horizontal space of the grid is reported in the xy coordinates of the grid faces and grid nodes. There are 5029 grid cells (and therefore grid face coordinate values) and 2888 grid nodes.

Each grid cell has a static `depth` value and a temporally dynamic `elev` value. The value of `depth` is the base depth of the water in that cell. The value of `elev` is water depth fluctuation due to tides. To find the true depth of water at any xy point at any time i , we need to add the base depth of the cell with the tidal fluctuation within the cell at time i .

The grid also predicts environmental variables through the water column by using sigma layers. These layers vary in thickness, but always cover the entire water column from surface to the river bottom. We know from prior data exploration and communication with the model's creators that the Corsica only experiences episodic stratification. Therefore, we are not overly concerned by the sigma layers that make up the vertical dimension of the model and can simply pull the surface layer values. In the future, we could consider using other sigma layers or averaging the environmental conditions in a cell through all its sigma layers.

Environmental variables of interest are salinity, temperature, dissolved oxygen (here called `ICM_21`), and chlorophyll. All variables but chlorophyll are simulated at the grid nodes. Chlorophyll is simulated at the grid faces. This is because Chlorophyll is not a state variable, but rather a property of phytoplankton biomass within the cell. We can alternatively calculate the chlorophyll at every node by adding the biomass of all phytoplankton variables (`ICM_3`: Diatom, `ICM_4`: Green Algae, `ICM_5`: Cyanobacteria) and multiplying them by 0.04, which is the C:Chl ratio used by the model. We will compute chlorophyll in this alternative way to match the xy grid the other environmental variables are mapped to. This method was suggested by the model creators.

Subset data

Time

We now need to start loading in our variables and paring them down. We will start with time. The model simulates conditions in the Corisca at an hour time step for the entire year of 2006, but we only need the summer season. We have defined this time period as 1 June through 31 August. Time in the model is recorded as seconds from an origin point, which is 00:00:00 on 1 January 2006. We can convert to a conventional timestamp, identify the seconds values of 00:00:00 1 June 2006 and 23:00:00 31 August 2006, and create a subsetting index from these bounds.

```
# Load time variable
time <- ncvar_get(nc_data, "time")

# Create conventional timestamps
f <- as.data.frame(time)
f$timestamp <- f$time/60/60/24
f$timestamp <- chron(f$timestamp, origin=c(month=1, day=1, year=2006))
f$timestamp <- as.Date(f$timestamp)

# Create vector of appropriate dates
dates <- seq(as.Date("2006-06-01"), as.Date("2006-08-31"), by='days')

# Subset f so only appropriate dates remain
f <- subset(f, f$timestamp %in% dates)
row.names(f) <- NULL

# Pull minimum and maximum timestamps
mintime <- f$time[1]
maxtime <- f$time[nrow(f)]

# Create subsetting index by time
timidx <- which(nc_data$dim$time$vals > mintime &
               nc_data$dim$time$vals < maxtime)

# Convert to data frame
time <- as.data.frame(time)

# Create index of steps
time$sidx <- seq(1:nrow(time))

# Subset so only times of interest are included
time <- time[time$sidx %in% timidx,]

# Create new index of steps, with 1 being first included in model
time$tidx <- seq(1:nrow(time))

# Add a more legible timestamp for later mapping purposes
time$timestamp <- time$time/60/60/24
time$timestamp <- chron(time$timestamp,
                       origin=c(month=1, day=1, year=2006))

# Convert chron to POSIXct (add 3 hours, some kind of timezone issue)
```

```
time$timestamp <- as.POSIXct(time$timestamp)+ hours(3)

# Remove intermediates
rm(f, dates, mintime, maxtime)
```

Horizontal grid

Now we will load the horizontal grid nodes. We want to subset to a portion of the total model to better match our idealized grid. The idealized grid is meant to be a simple representation of the St. Leonard Creek transect, which is typically 2500m long and centered across 500m of water. I have chosen a segment of the model that approximates those proportions. This is the section between Emory Creek (the largest creek on the North shore of the River, closer to mouth) and Alder Branch (the next creek on the North shore moving upstream, closer to head). I will manipulate the nodes in this segment to more closely fit in our idealized rectangular model space.

The nodes here are not at all regularly spaced. That's part of the beauty of the SCHISM setup– it uses irregularly sized and shaped grid cells to vary resolution as needed. The nodes are therefore irregularly placed. The river itself is also not a simple shape, but has many curves and bends. The grid is shaped by these shores. We want to set up our IBM in a rectangular grid space, so we will need to manipulate the nodes to better fit within our chosen extent and shape. The entire feature needs to be rotated so the “mouth” end is at the lower limit of our y-axis and the “head” is at the upper limit of our y-axis. Nodes within the feature need to be moved to give better coverage within the rectangular space. Because this is an idealized space, it is not critical that nodes are exactly the same distance and bearing to each other in the IBM as they are in SCHISM-ICM output. We simply want to make sure that we preserve the cross-channel and down-creek trends.

Moving the nodes in space will change the original xy coordinates, and therefore we need to have another method of relating the environmental variables to their matching nodes in space. This was done by creating an ID for each of the nodes that was preserved throughout the point manipulation. Environmental variables simulated at point x_q, y_q were given the same ID as the grid node that was originally at point x_q, y_q . Therefore, environmental variables were still matched with their original nodes even after the nodes were moved in space.

The manipulation of nodes was best done in ArcMap. I have read the manipulated nodes in below.

```
# Load in manipulated points
gri <- read.csv(paste0('G:/My Drive/Documents_Backup/Modeling/',
                      'GIS/SCHISM_pts3.csv'))

# Create index of points to keep
keep_pts <- gri$idx
# Plot, maintain 1:1 aspect ratio
plot(gri$lon, gri$lat, asp=1, xlab='Longitude (m)', ylab='Latitude (m)',
     pch=19, cex=0.5)
```

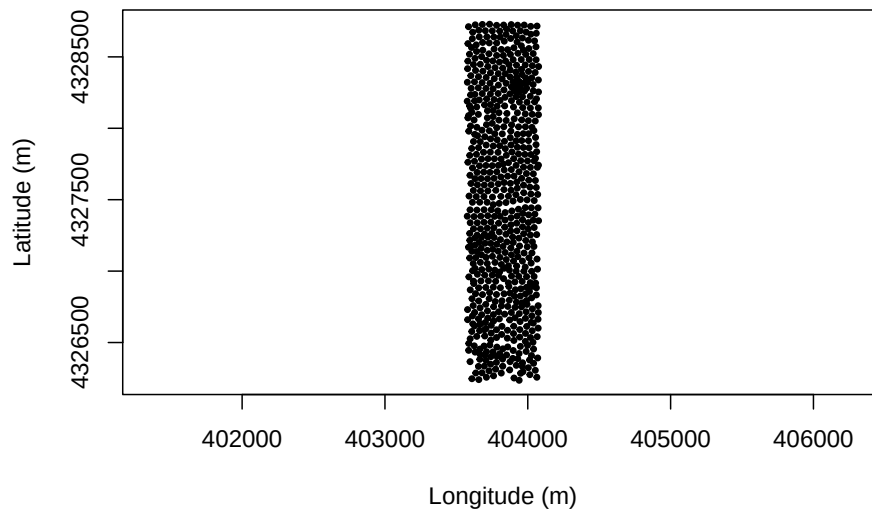


Figure 3: Manipulated grid node positions

Depth and tidal information

Now we have an index of space-time locations that we are interested in. We will pull the static base depth and dynamic tidal fluctuations using these indexes.

```
# Read tidal info
ele <- ncvar_get(nc_data, "elev")

# Convert to dataframe
ele <- as.data.frame(ele)

# Subset to times of interest: removes columns for other times
ele <- ele[,timidx]

# Create location index (still have all locations)
ele$idx <- seq(1:nrow(ele))

# Subset location index to only include points of interest
ele <- ele[ele$idx %in% keep_pts,]
```

Environmental variables

Next, we'll read in the other environmental variables. These will be subset to only include the locations and times that we are interested in. They will also only include the surface measurements—sigma layer 5.

The chlorophyll variable will need to be calculated by summing the phytoplankton inputs at each node and multiplying by the C:Chl ratio.

We will also calculate the true depth of water in space-time by adding the static base depth with the dynamic changes in elevation due to tides.

```
# Identify variables of interest
data <- c('salt', 'temp', 'ICM_21', 'ICM_3', 'ICM_4', 'ICM_5')

# Loop to read in environmental variables
for(i in 1:length(data)){
  # Read in variable
  varo <- ncvar_get(nc_data, paste0(data[i]))[5,,]
  # Convert to dataframe
  varo <- as.data.frame(varo)
  # Subset to times of interest: removes columns for other times
  varo <- varo[,timidx]
  # Create location index (still have all locations)
  varo$idx <- seq(1:nrow(varo))
  # Subset location index to only include points of interest
  varo <- varo[varo$idx %in% keep_pts,]
  # Identify variable name
  char <- paste0(data[i])
  # Rename variable to variable name
  varo <- assign(char, varo)
  # Remove intermediates
  rm(varo, char)
}

# Calculate chlorophyll
# Create duplicates of phytoplankton variables without index
ICM_32 <- ICM_3
ICM_32$idx <- NULL
ICM_42 <- ICM_4
ICM_42$idx <- NULL
ICM_52 <- ICM_5
ICM_52$idx <- NULL

# Combine data for all phytoplankton variables
chl <- cbind(ICM_32, ICM_42, ICM_52)
# Add columns with matching names across all dataframes
chl <- t(rowsum(t(chl), group=colnames(chl), na.rm=T))
# Convert to dataframe
chl <- as.data.frame(chl)
# Divide by C:Chl ratio (0.04)
chl <- chl / 0.04
# Copy over preserved index variable
chl$idx <- ICM_3$idx
# Move index variable to last column
```

```

chl <- chl %>% dplyr::select(-idx, idx)

# Rename dataframes to desired variable names
sal <- salt
dox <- ICM_21
tem <- temp

# Calculate true depth -- depth plus elevation
# Create duplicate of elevation without index
tdep <- ele
tdep$idx <- NULL
# Add depth
for(i in 1:ncol(tdep)){
  tdep[,i] <- tdep[,i] + gri$dep
}
tdep$idx <- ele$idx

# Close netcdf file
nc_close(nc_data)

# Remove intermediates
rm(ICM_21, ICM_3, ICM_4, ICM_5,
   ICM_32, ICM_42, ICM_52,
   salt, temp, ele,
   nc_data, keep_pts, timidx, i, data)

```

Checking

All variables should have the same dimensions. There are 624 nodes of interest, 1 sigma layer of interest, and 2206 time steps of interest. Let's check that we have the correct dimensions. Variables in both space and time (eg. salt) will have an extra column: the index variable we will keep to preserve mapping ability.

```

dim(time)

## [1] 2206    4
dim(gri)

## [1] 624    4
dim(sal);dim(tem);dim(dox);dim(chl); dim(tdep)

## [1] 624 2207
## [1] 624 2207
## [1] 624 2207
## [1] 624 2207
## [1] 624 2207

```

Combine

We already have a dataframe of the node positions and static base depth at those positions. This defines the grid space we will use. We want to make sure that we keep things together, so we will combine all environmental variables into a list. Time will remain a separate dataframe.

```
# Gather environmental variables into list
env <- list(sal, tem, dox, chl, tdep)

# Name list items
names(env) <- c('sal', 'tem', 'dox', 'chl', 'tdep')

# Remove intermediates
rm(sal, tem, dox, chl, tdep)
```

Creation of model space raster

Next we'll make a blank raster of the desired proportions. We want a 2500m by 500m grid. Desired grid cell resolution is 10m by 10m. This translates to a grid that is 250 cells long and 50 cells wide.

```
# Set number of rows and columns
n.row <- 250; n.col <- 50

# Create matrix of desired dimensions, rasterize
r <- raster(matrix(0, n.row, n.col))

# Set extent to match points retained from SCHISM manipulation
# Make sure extent is exactly 2500m by 500m so cell resolution is 10m
extent(r) <- c(403575.9, 404075.9,
              4326232 , 4328732 )

# Set projection
# NAD83 HARN StatePlane Maryland FIPS 1900 (m)
r@crs <- CRS("ESRI:102285")
```

Test of raster by plotting depth

We'll test the success of our raster by plotting the static base depth. Base depth at each cell will be interpolated using inverse distance weighting interpolation. The four closest neighbors by Euclidean distance will be considered. The inverse distance power needed to complete IDP will be estimated using k-fold cross-validation methods. K chosen for this interpolation is 10. Sample size is the number of points, which is 624.

```
# Convert raster to SpatialPixels
rpix <- as(r, "SpatialPixels")

# Points to spatial points data frame
spdat <- sp::SpatialPointsDataFrame(
  # Coordinates set to lon and lat (will be same for all timesteps)
  coords=gri[,c("lon", "lat")],
  # Data set to variable at timestep i
  data =gri[,c('idx', 'dep')],
  # CRS set to NAD83 HARN StatePlane Maryland FIPS 1900 (m)
  proj4string = CRS("ESRI:102285")
)

# Rename dep to value, artifact of intamap package
names(spdat@data) <- c('idx', 'value')

# Suppress warning associated with CRS comments
options(warn=-1)

# Set up idwObject
idwObject <- createIntamapObject(
  # Set observations to SpatialPointsDataFrame
  observations = spdat,
  # Set formula to universal kriging
  formulaString = as.formula(value~1),
  # Set prediction locations to blank raster of model space
  predictionLocations = rpix,
  # Set method to inverse distance weighting
  class = "idw"
)

# k-fold brute-force Cross-validation selection of IDP
# Set k, typically 5 or 10
k <- 10
# Estimate best IDP for IDW interpolation
idwObject <- estimateParameters(
  idwObject,
  # Set range of possible inverse distance power values
  # min=0, max=5, step=0.05. All steps will be evaluated.
  idpRange = seq(0.00,5.00,0.05),
  # Call k-fold value
  nfold=k
)

# Interpolate using IDP chosen above
fitmax <- gstat::gstat(
```

```

# Again, universal kriging
formula=value ~ 1,
# Call spatial points dataframe with
data=spdat,
# Set number of neighbors to consider. Bigger=smoother
nmax=4,
# Set IDW estimated above
set=list(idp=idwObject$inverseDistancePower)
)

# Rasterize interpolation based on previous steps
maxint <- raster::interpolate(r, model=fitmax, ext=r)

# Save minimum and maximum values for plotting (maximizes color stretch)
mindep <- floor(min(maxint@data@values))
maxdep <- ceiling(max(maxint@data@values))

# Set color scheme
cols <- viridis(256)

# Plot raster
raster::plot(maxint, asp=1, zlim=c(mindep,maxdep), col=cols,
             legend=T, ylab='Latitude (m)',
             xlab='Longitude (m)',
             legend.args=list(text='Depth (m)',side=4,
                              font=2, line=2.5, cex=0.8),
             xlim=c(maxint@extent[1], maxint@extent[2]))

```

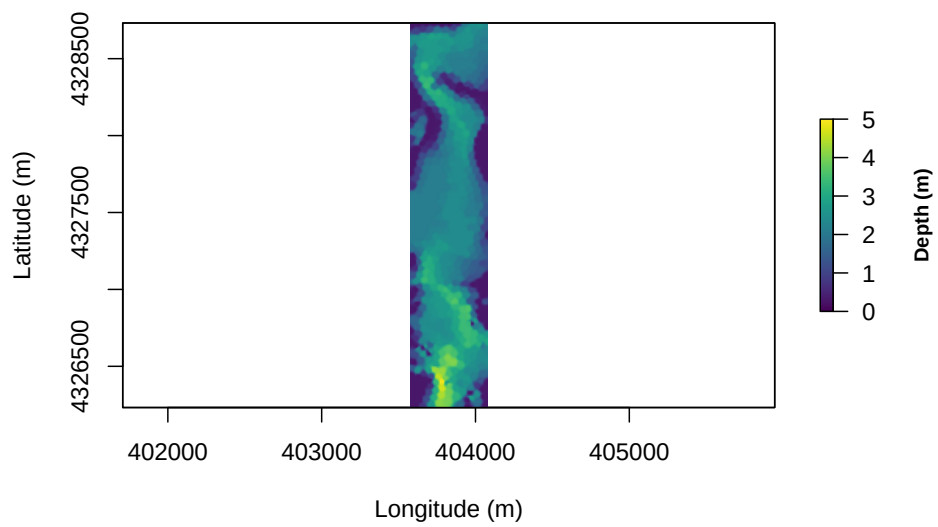


Figure 4: Inverse Distance Weighted Interpolated base depth of raster space

```
## [1] "best idp value found is 4.95 rmse 0.529657973103509"
## [inverse distance weighted interpolation]
```

Creation of hourly rasters of environmental variables

This is a successful method of creating hourly rasters of environmental conditions. We will now run this method to create rasters for every timestep and every location in our model. Obviously, that won't be run in an RMD. This is a recreation of the code used to generate the rasters and save them as data files to reference later.

```
# Set color scheme
cols <- viridis(256)

# Convert raster to SpatialPixels
rpix <- as(r, "SpatialPixels")

# Set number of steps
steps <- nrow(time)

for(q in 1:length(env)){
  # Allocate blank lists for raw data and rasters
  val_list <- vector(mode="list", length=steps)
  val_rast <- vector(mode="list", length=steps)

  # Loop and create IDW interpolated rasters of variable
  for(i in 1:steps){
    # Create dataframe of variable at timestep i for all locations
    ts_dat <- as.data.frame(cbind(env[[q]][,i],
                                  env[[q]][,2207]))
    names(ts_dat) <- c('value', 'idx')

    # Merge location and variable by index name of location points
    toplot <- merge(gri,
                    ts_dat,
                    by='idx')

    # Save dataframe for this timestep
    val_list[[i]] <- toplot

    # Points to spatial points data frame
    spdat <- sp::SpatialPointsDataFrame(
      # Coordinates set to lon and lat (will be same for all timesteps)
      coords=val_list[[i]][,c("lon", "lat")],
      # Data set to variable at timestep i
      data =val_list[[i]][,c('idx', 'value')],
      # CRS set to NAD83 HARN StatePlane Maryland FIPS 1900 (m)
      proj4string = CRS("ESRI:102285")
    )

    # Set up idwObject
    idwObject <- createIntamapObject(
      # Set observations to SpatialPointsDataFrame
      observations = spdat,
      # Set formula to universal kriging
      formulaString = as.formula(value~1),
      # Set prediction locations to blank raster of model space
```

```

    predictionLocations = rpix,
    # Set method to inverse distance weighting
    class               = "idw"
  )

  # k-fold Cross-validation selection of IDP
  # Set k, typically 5 or 10
  k <- 10

  # Estimate best IDP for IDW interpolation
  idwObject <- estimateParameters(
    idwObject,
    # Set range of possible inverse distance power values
    idpRange = seq(0.00,5.00,0.05),
    # Call k-fold value
    nfold=k
  )

  # Interpolate using IDP chosen above
  fitmax <- gstat::gstat(
    # Again, universal kriging
    formula=value ~ 1,
    # Call spatial points dataframe with
    data=spdat,
    # Set number of neighbors to consider. Bigger=smoother
    nmax=4,
    # Set IDW estimated above
    set=list(idp=idwObject$inverseDistancePower)
  )

  # Rasterize interpolation based on previous steps
  maxint <- raster::interpolate(r, model=fitmax, ext=r)

  # Save raster to list
  val_rast[[i]] <- maxint

  # Call variable name
  char <- names(env[q])

  # Rename as needed
  names(val_rast[[i]]) <- char
  rm(maxint, spdat, toplot, ts_dat, fitmax, char)
}

# Call variable name
char <- names(env[q])
setwd("G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/Raster_Data")
saveRDS(val_rast, file=paste0(char, '_rast.rds'))
rm(char, val_list, val_rast)
}

```

Conclusion

The next step of modeling will be to create suitability indices for each of these environmental variables. The end goal is to create an overall habitat suitability index based on the variables we have. The fish will then move in the modeled space by sensing suitability within a sensing neighborhood and moving towards the best sensed cell. This document is followed by “HSI Modeling,” which explains the process of converting these idealized environmental rasters to a habitat suitability index raster.

Habitat Suitability Index Modeling

Katie Lankowicz

19 April 2022

This document will preserve the workflow and explain the code for building suitability indices (SIs) and an overall habitat suitability index (HSI) for the modeled space that will serve as the backdrop for our individual-based model of juvenile menhaden movement.

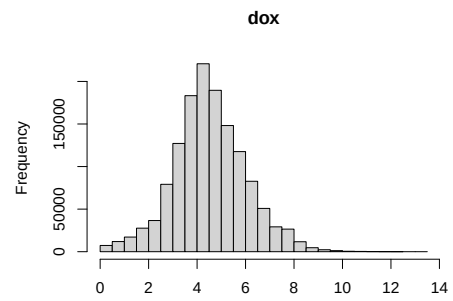
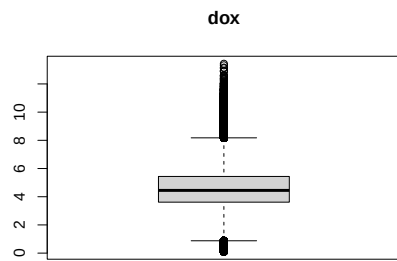
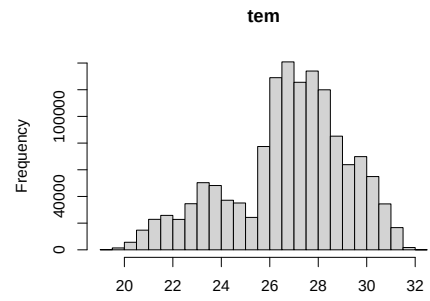
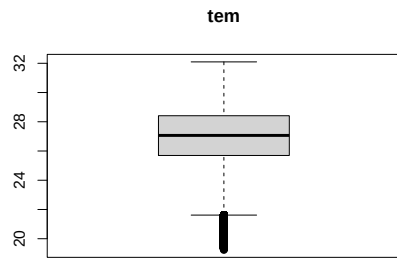
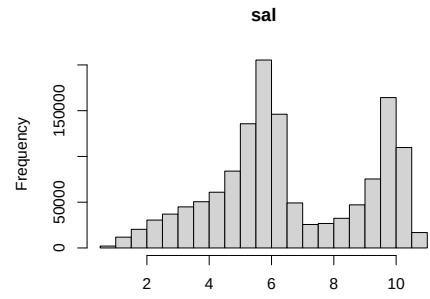
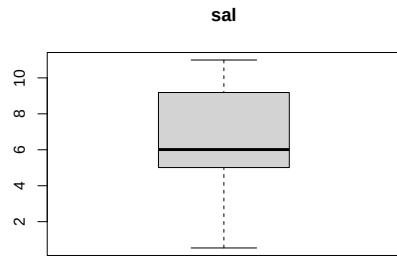
Loading the environmental rasters

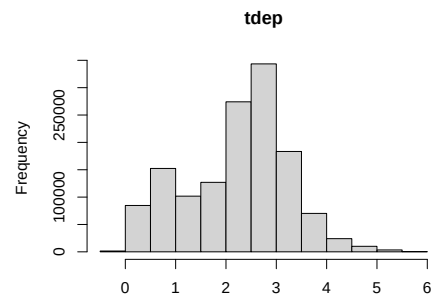
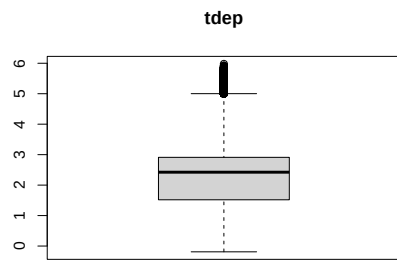
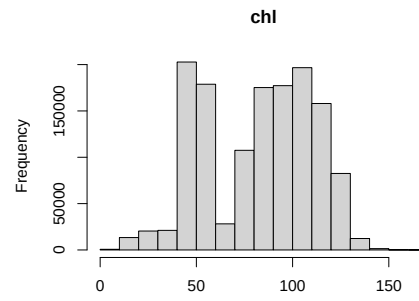
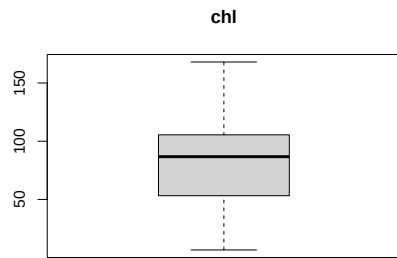
A previous document (“Preparing SCHISM-ICM netcdf files”) has explained the preparation of SCHISM-ICM data for inclusion in the model. Here, we will load the prepared environmental data and rasterized model space. We will also explore the range of these variables. Please keep in mind that these boxplots and histograms will show the range of the variables through the entire space and time of the model.

Variables available to us are salinity (ppt), temperature (°C), dissolved oxygen (mg/L), chlorophyll (ug/L), and depth (m). The ranges of these values are comparable to those the authors have recorded (albiet intermittently) in St. Leonard Creek, where our ARIS menhaden schooling information comes from.

```
# Load data
load(paste0('G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/',
            'Raster_Data/ncdf_output.RData'))

# Set plotting arrangement
par(mfrow=c(1,2))
# Create boxplot and histogram for each variable
for(i in 1:length(env)){
  char <- names(env[i])
  val <- env[[i]]
  val$idx <- NULL
  val <- unlist(val, use.names = F)
  boxplot(val, main=paste(char))
  hist(val, main=paste(char), xlab=' ')
  rm(val, char)
}
```





Relating habitat suitability to environmental variables

We now need to make some decisions about what makes the most suitable habitat for juvenile menhaden schools. We will need to generate suitability functions for each of our environmental variables. The shape of these functions is not likely to be linear. Values from the literature will provide priors from which we will build these functions.

Dissolved oxygen

We will start with dissolved oxygen. It is obvious that deoxygenated waters ($< 2\text{mg/L}$) are poor habitat and highly-oxygenated waters are more suitable. The shape of the suitability curve is loosely based on a 1982 report on habitat suitability indexes for estuarine gulf menhaden, which are closely-related clupeids. This report proposes that weekly DO concentrations of 3-5 mg/L have a linear increase in suitability, and increases in DO beyond 5 mg/L offer little additional benefit.

```
# Call dissolved oxygen
val <- env[['dox']]
# Remove indexing variable
val$idix <- NULL
# Convert from matrix to vector
val <- unlist(val, use.names = F)
# Save min/ max and rounded range
minval <- min(val)
maxval <- max(val)
rangeval <- ceiling(maxval) - floor(minval)

# Calculate number of points to have an estimate every 0.0001 x units
nest <- rangeval/0.0001

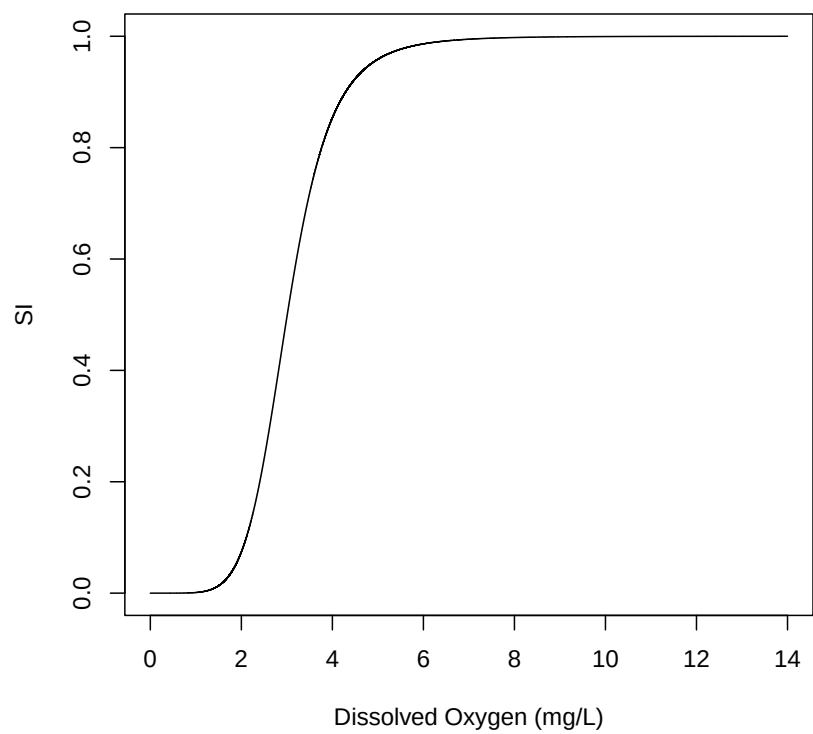
# Fit curve
dox_curve <- curve(0.9822161 + (-0.0309863 - 0.9822161)/(1 + (x/3.008742)^6.191692),
  from=floor(minval),
  to=ceiling(maxval),
  xlab='Dissolved Oxygen (mg/L)',
  ylab='SI',
  n=nest
)

# Unlist from curve item to dataframe
dox_curve <- as.data.frame(t(do.call(rbind, dox_curve)))

# Rescale so there are no negative values
dox_curve$y <- range01(dox_curve$y)

#Plot
plot(dox_curve$x, dox_curve$y, type='l',
  xlab='Dissolved Oxygen (mg/L)', ylab='SI')

rm(minval, maxval, nest, rangeval, val)
```



Depth

The tidal fluctuations in the SCHISM simulation result in “dry cells”, or times when near-coast areas are above the low water mark. Obviously, “dry cells” will have a suitability of 0. From empirical ARIS data, juvenile menhaden prefer shallow areas. In St. Leonard Creek, GAM output of density of menhaden decreases almost linearly from 1m to 5m, at which point density is less than the mean for the entire dataset. Areas deeper than 5m are still suitable habitat, so we will not scale the suitability of areas > 5m in depth to 0. We will represent this relationship with the following function.

```
# Call depth
val <- env[['tdep']]
# Remove indexing variable
val$idix <- NULL
# Convert from matrix to vector
val <- unlist(val, use.names = F)
# Save min/ max and rounded range
minval <- min(val)
maxval <- max(val)
rangeval <- ceiling(maxval) - floor(minval)

# Calculate number of points to have an estimate every 0.0001 x units
nest <- rangeval/0.0001

# Fit curve
tdep_curve1 <- as.data.frame(cbind(seq(-1,-0.0001,by=0.0001),
                                   rep(0,1000)))

# Section 2: 0 to 1
tdep_curve2 <- curve(-0.793*x^2 + 1.8279*x - 0.0313,
                    from=0,
                    to=1,
                    n=(1/0.0001))

# Section 3: 1+
tdep_curve3 <- curve(0.0019*x^3 - 0.0266*x^2 - 0.0142*x + 1.0333,
                    from=1,
                    to=ceiling(maxval),
                    xlab='Depth (m)',
                    ylab='SI',
                    n=((ceiling(maxval)-1) / 0.0001)
                    )

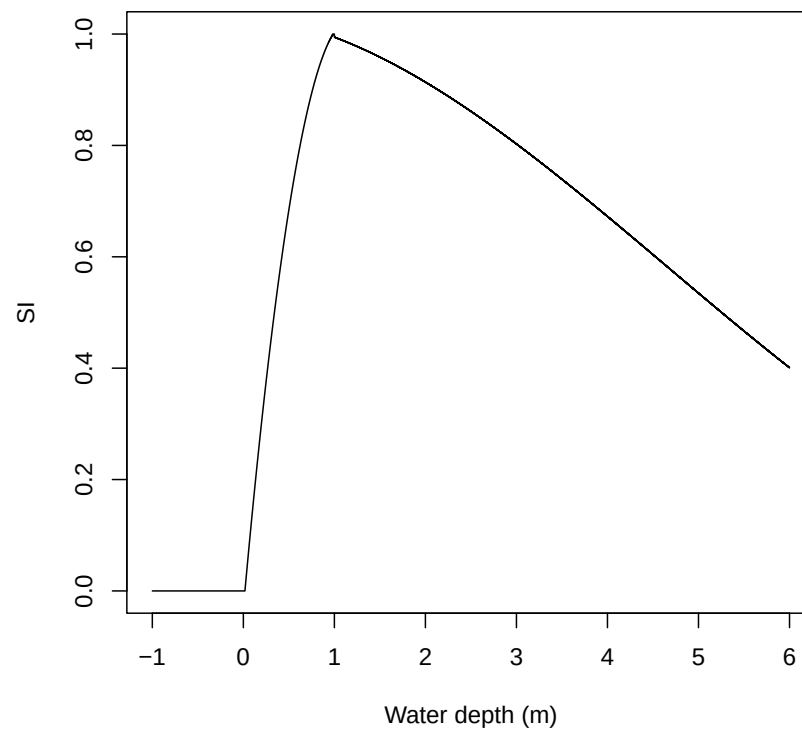
# Combine
tdep_curve <- tdep_curve3
tdep_curve$x <- c(tdep_curve1$V1, tdep_curve2$x, tdep_curve3$x)
tdep_curve$y <- c(tdep_curve1$V2, tdep_curve2$y, tdep_curve3$y)

# Convert to dataframe
tdep_curve <- as.data.frame(t(do.call(rbind, tdep_curve)))
row.names(tdep_curve) <- NULL

# Replace slightly over/ underestimated values
tdep_curve['y'][tdep_curve['y'] < 0] <- 0
```

```
tdep_curve['y'][tdep_curve['y']>1] <- 1

#Plot
plot(tdep_curve$x, tdep_curve$y, type='l',
      xlab='Water depth (m)', ylab='SI')
rm(tdep_curve1, tdep_curve2, tdep_curve3, minval, maxval, nest, rangeval, val)
```



Temperature

Priors for temperature come from SEDAR 69, which is ASMFC's January 2020 release on the stock assessment for menhaden. The relationship is for age-1+ menhaden in the coastal Atlantic, and so this is only a loose approximation. Further, estuarine environments in summer do not experience the full range of temperatures modeled by the ASMFC (0-35 °C). We will subset their temperature model to only include temperatures that we see in our model and scale the suitability index over our range of temperatures.

```
# Call temperature
val <- env[['tem']]
# Remove indexing variable
val$idx <- NULL
# Convert from matrix to vector
val <- unlist(val, use.names = F)
# Save min/ max and rounded range
minval <- min(val)
maxval <- max(val)
rangeval <- ceiling(maxval) - floor(minval)

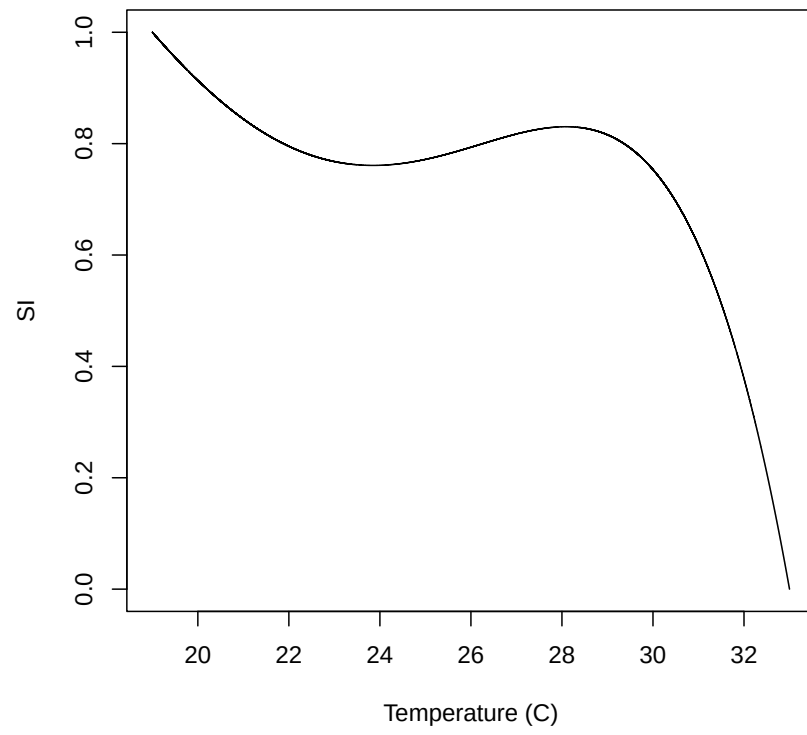
# Calculate number of points to have an estimate every 0.0001 x units
nest <- rangeval/0.0001

# Fit curve
tem_curve <- curve(0.2594068 - 0.04318046*x + 0.03738088*x^2 -
                    0.003929789*x^3 + 0.000146598*x^4 -
                    0.000001847609*x^5,
                    from=floor(minval),
                    to=ceiling(maxval),
                    xlab='Temperature (C)',
                    ylab='SI',
                    n=nest
                    )

# Convert to dataframe
tem_curve <- as.data.frame(t(do.call(rbind, tem_curve)))
row.names(tem_curve) <- NULL

# Rescale values
tem_curve$y <- range01(tem_curve$y)

#Plot
plot(tem_curve$x, tem_curve$y, type='l',
      xlab='Temperature (C)', ylab='SI')
rm(minval, maxval, nest, rangeval, val)
```



Salinity

Similar to temperature, priors for salinity come from SEDAR 69. Also similar to temperature, our summer estuarine environment has a comparatively limited range of salinity. We will subset their salinity model and rescale the suitability index to cover our range of salinity values.

```
# Call salinity
val <- env[['sal']]
# Remove indexing variable
val$idx <- NULL
# Convert from matrix to vector
val <- unlist(val, use.names = F)
# Save min/ max and rounded range
minval <- min(val)
maxval <- max(val)
rangeval <- ceiling(maxval) - floor(minval)

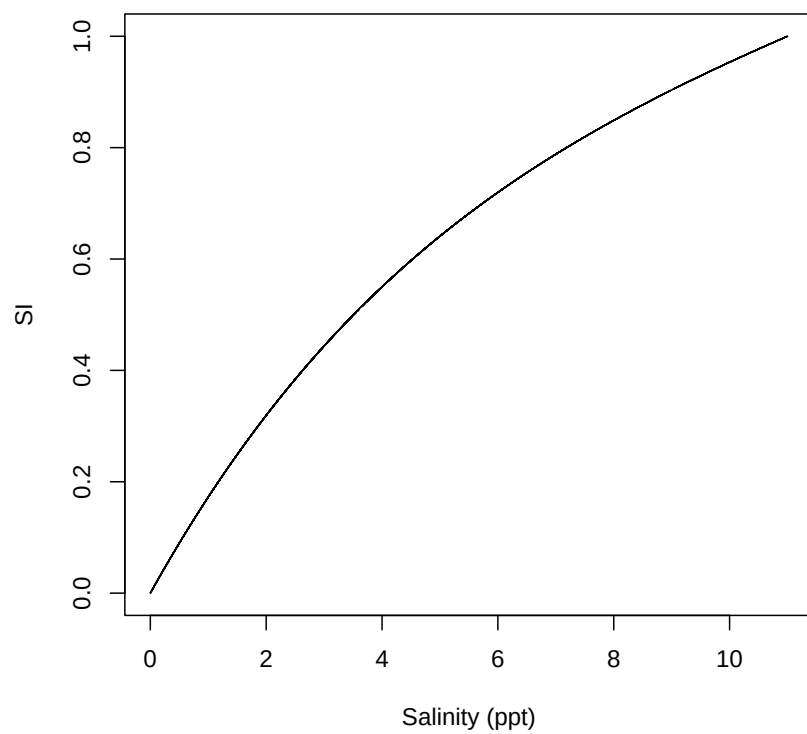
# Calculate number of points to have an estimate every 0.0001 x units
nest <- rangeval/0.0001

# Fit curve
sal_curve <- curve(8.217366 + 1.203565*x - 0.09766958*x^2 +
                   0.004988636*x^3 - 0.0001227273*x^4 +
                   0.000001071795*x^5,
                   from=floor(minval),
                   to=ceiling(maxval),
                   xlab='Salinity (ppt)',
                   ylab='SI',
                   n=nest
                   )

# Convert to dataframe
sal_curve <- as.data.frame(t(do.call(rbind, sal_curve)))
row.names(sal_curve) <- NULL

# Rescale values
sal_curve$y <- range01(sal_curve$y)

#Plot
plot(sal_curve$x, sal_curve$y, type='l',
      xlab='Salinity (ppt)', ylab='SI')
rm(minval, maxval, nest, rangeval, val)
```



Chlorophyll

Finally, chlorophyll. There are several papers that discuss juvenile menhaden affinity for chlorophyll. Dalyander and Cerco (2010) suggest that the feeding threshold exists at 0.004 mg/L, or 4 ug/L. Lynch et al. (2010) suggest that grazing rates increase until around 200 ug/L, at which point it begins to level off. We will model the suitability of chlorophyll concentrations in the model as logarithmic relationship that increases past 4 ug/L and levels off at around 200 ug/L .

```
# Call chlorophyll
val <- env[['chl']]
# Remove indexing variable
val$idx <- NULL
# Convert from matrix to vector
val <- unlist(val, use.names = F)
# Save min/ max and rounded range
minval <- min(val)
maxval <- max(val)
rangeval <- ceiling(maxval) - floor(minval)

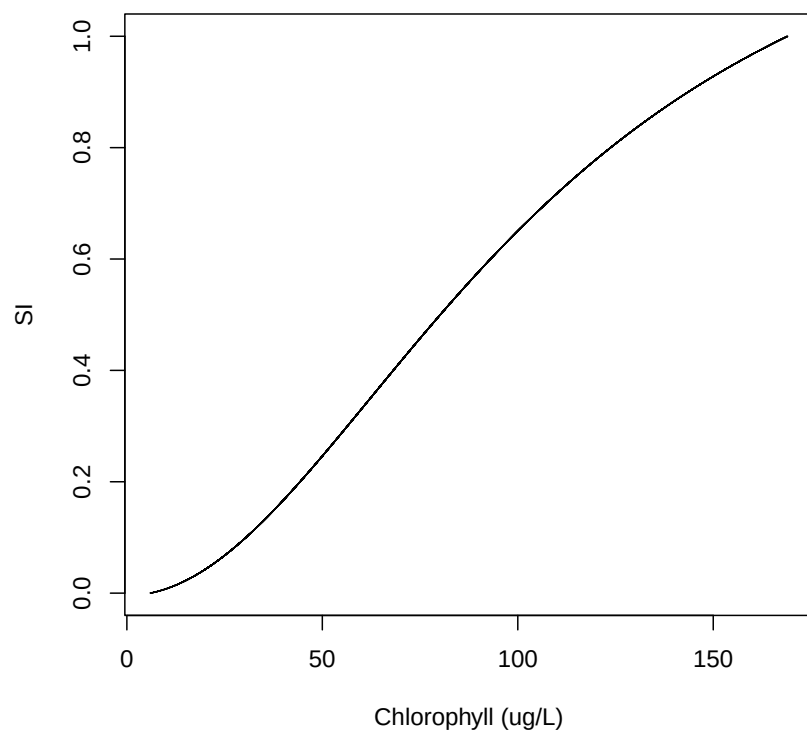
# Calculate number of points to have an estimate every 0.0001 x units
nest <- rangeval/0.0001

# Fit curve
chl_curve <- curve(1.285553 + (-0.01257163
                      - 1.285553)/(1 + (x/107.1234)^2.007354),
                  from=floor(minval),
                  to=ceiling(maxval),
                  xlab='Salinity (ppt)',
                  ylab='SI',
                  n=nest
                  )

# Convert to dataframe
chl_curve <- as.data.frame(t(do.call(rbind, chl_curve)))
row.names(chl_curve) <- NULL

# Rescale values
chl_curve$y <- range01(chl_curve$y)

#Plot
plot(chl_curve$x, chl_curve$y, type='l',
      xlab='Chlorophyll (ug/L)', ylab='SI')
rm(minval, maxval, nest, rangeval, val)
```



Gather all curves into a list

We will collate all the curves into a single list for safekeeping.

```
# Gather into list and name
curv_list <- list(sal_curve, tem_curve, dox_curve, chl_curve,
                 tdep_curve)
names(curv_list) <- names(env)

# Remove intermediates
rm(sal_curve, tem_curve, dox_curve, chl_curve, tdep_curve)
```

Loading environmental rasters

In the previous document, I went through the process of creating inverse distance weighted interpolated rasters of these environmental conditions through the model space at each time step. Rasters for the month of June will be loaded below.

```
# Load environmental raster lists
for(i in 1:length(env)){
  char <- names(env[i])
  char_rast <- readRDS(paste0("G:/My Drive/Documents_Backup/Modeling/",
                             "Modeling_RCode/Raster_Data/shortcut/",
                             char, "_rast_short.RDS"))
  char_rast <- assign(paste0(char, "_rast"), char_rast)
}

# Gather lists into one big list, rename
rast_list <- list(sal_rast, tem_rast, dox_rast, chl_rast, tdep_rast)
names(rast_list) <- names(env)

# Remove intermediates
rm(char_rast, char, sal_rast, tem_rast, dox_rast, chl_rast, tdep_rast)
```

Substitution of suitability indices for environmental variables

We now have a collection of environmental rasters collated into a two-layer list. The outer layer of the list has five levels, one for each environmental variable of interest. Inside each of those levels of the outer layer are n rasters, where n represents the number of modeled time steps. Now that we have our SI curves, we need to replace the simulated values of the environmental conditions in each raster with the matching suitability index value for that environmental condition value. This code is intensive, and so will only be reproduced below without running.

```
# Set names of variables
env <- names(env)

# Set steps
steps <- length(rast_list[[1]])

for(q in 1:length(env)){
  # Call variable name
  char <- names(rast_list[q])

  # Call original raster list
  val_rast <- rast_list[[q]]

  # Call curve, round to 4 decimals
  val_curv <- curv_list[[q]]
  val_curv <- round(val_curv,4)

  # Loop and create SI for each raster
  for(i in 1:steps){
    # Call data, round to 4 decimals
    tempdata <- round(val_rast[[i]]@data@values,4)

    # Replace raster data values with matched SI values
    for(m in 1:length(tempdata)){
      newval <- tempdata[m]
      newval <- val_curv$y[val_curv$x==newval]
      # If (rarely) a single x-value is skipped, find average of neighboring values
      if(length(newval)==0){
        oneless <- tempdata[m]- 0.0001
        onemore <- tempdata[m]+ 0.0001
        valless <- val_curv$y[val_curv$x==oneless]
        valmore <- val_curv$y[val_curv$x==onemore]
        newval <- mean(valless, valmore)
        # Remove intermediates
        rm(oneless, onemore, valless, valmore)
      }
      tempdata[m] <- newval
    }

    # Replace values in raster with HSI values
    val_rast[[i]]@data@values <- tempdata

    # Rename as needed
  }
}
```

```
names(val_rast[[i]]) <- char

# Remove intermediate
rm(tempdata, newval)

}

# Save output
setwd("G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/Raster_Data/HSI")
saveRDS(val_rast, file=paste0(char, '_hsi_rast.rds'))
rm(char, val_rast, val_curv)
}
```

Overall Habitat suitability index

Ideas from this section came from Fabrizio et al. 2021, where habitat suitability for several species was modeled in the entire Chesapeake Bay given data from an unTRIM three-dimensional hydrodynamic model. There are two methods to consider for calculating the overall habitat suitability index. The first is a simple arithmetic mean:

$$HSI_{am} = \frac{EV_1 + EV_2 + \dots + EV_p}{p}$$

where EV_1 is the suitability for environmental variable 1, etc, and p is the total number of environmental variables considered.

| The second method is a geometric mean, which applies a limiting factor so that a single poor environmental variable can result in a low overall habitat suitability. This is given by:

$$HSI_{gm} = \sqrt[p]{EV_1 \times EV_2 \times \dots \times EV_p}$$

The geometric mean method may be more appropriate for us- extremely poor SI values for a single environmental variable, like water depth, would realistically “tank” the overall habitat suitability index in that raster cell.

We will now calculate the habitat suitability index for each raster cell using the geometric mean method. This will need to be done at each time step. Again, the code is simply reproduced here due to computational strain.

```
# Set names of variables
env <- names(env)

# Load SI rasters
for(i in 1:length(env)){
  char <- env[i]
  char_rast <- readRDS(paste0("G:/My Drive/Documents_Backup/Modeling/",
                             "Modeling_RCode/Raster_Data/HSI/short/",
                             char, "_hsi_rast_short.RDS"))
  char_rast <- assign(paste0(char, "_si"), char_rast)
}

# Gather SI rasters into list
SI_list <- list(sal_si, tem_si, dox_si, chl_si, tdep_si)
names(SI_list) <- env

# Remove intermediates
rm(sal_si, tem_si, dox_si, chl_si, tdep_si)

# Allocate blank list
HSI_rast <- vector(mode="list", length=length(SI_list[[1]]))

# Loop through SI rasters to create HSI
for(i in 1:length(SI_list[[1]])){
  # Overlay all rasters and take geometric mean HSI per cell
  HSI_rast[[i]] <- overlay(SI_list[['sal']][[i]],
                          SI_list[['tem']][[i]],
```

```
        SI_list[['dox']][[i]],
        SI_list[['chl']][[i]],
        SI_list[['tdep']][[i]],
        fun=psych::geometric.mean)

    print(i)
}

# Save results
setwd("G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/Raster_Data/HSI/results")
saveRDS(HSI_rast, file='HSI_rast_short_gm.rds')
```

Conclusion

These rasters are the final result of the pre-processing. We can now feed these results into the individual-based model of juvenile menhaden movement in which the menhaden sense and respond to their surrounding environment. A selection of HSI rasters will be animated in a GIF to visualize spatio-temporal suitability trends.

Habitat Suitability Correlation

Katie Lankowicz

20 April 2022

We will now do a subset of simple visual checks for correlation between the environmental inputs of our habitat suitability model. This will be completed by simply leaving one environmental variable out of the overall HSI calculation at a time. If there is a strong correlation between two of the environmental variables, leaving one out will look exactly the same as leaving the other out. For example, if temperature and salinity are highly correlated (which we expect), removing salinity from the HSI model will result in a raster that looks very similar to the resulting raster from removing temperature.

Loading data

```
# Load data
load(paste0('G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/',
            'Raster_Data/ncdf_output.RData'))

# Fix time
time$timestamp <- time$timestamp + lubridate::hours(3)

# Create subset of times to test
timesubs <- seq(1, 2206, by=240)

# Set names of variables
env <- names(env)

# Load SI rasters
for(i in 1:length(env)){
  char <- env[i]
  char_rast <- readRDS(paste0("G:/My Drive/Documents_Backup/Modeling/",
                              "Modeling_RCode/Raster_Data/HSI/",
                              char, "_hsi_rast.RDS"))
  char_rast <- assign(paste0(char, "_si"), char_rast)
}

# Gather SI rasters into list
SI_list <- list(sal_si, tem_si, dox_si, chl_si, tdep_si)
names(SI_list) <- env
```

Remove items sequentially

```
no_sal <- SI_list[-1]
no_tem <- SI_list[-2]
no_dox <- SI_list[-3]
no_chl <- SI_list[-4]
no_tdep <- SI_list[-5]

# Remove intermediates
rm(sal_si, tem_si, dox_si, chl_si, tdep_si)
```

Calculate leave-one-out HSI results

```
# Allocate blank list - all rasters
HSI_rast <- vector(mode="list", length=length(timesubs))
# Loop through SI rasters to create HSI
for(i in 1:length(timesubs)){
  j <- timesubs[i]
  # Overlay all rasters and take geometric mean HSI per cell
  HSI_rast[[i]] <- raster::overlay(SI_list[['sal']][[j]],
                                   SI_list[['tem']][[j]],
                                   SI_list[['dox']][[j]],
                                   SI_list[['chl']][[j]],
                                   SI_list[['tdep']][[j]],
                                   fun=psych::geometric.mean)
}

# Allocate blank list - no salinity
nosal_rast <- vector(mode="list", length=length(timesubs))
# Loop through SI rasters to create HSI
for(i in 1:length(timesubs)){
  j <- timesubs[i]
  # Overlay all rasters and take geometric mean HSI per cell
  nosal_rast[[i]] <- raster::overlay(#no_sal[['sal']][[j]],
                                     no_sal[['tem']][[j]],
                                     no_sal[['dox']][[j]],
                                     no_sal[['chl']][[j]],
                                     no_sal[['tdep']][[j]],
                                     fun=psych::geometric.mean)
}

# Allocate blank list - no temperature
notem_rast <- vector(mode="list", length=length(timesubs))
# Loop through SI rasters to create HSI
for(i in 1:length(timesubs)){
  j <- timesubs[i]
  # Overlay all rasters and take geometric mean HSI per cell
  notem_rast[[i]] <- raster::overlay(no_tem[['sal']][[j]],
                                     #no_sal[['tem']][[j]],
                                     no_tem[['dox']][[j]],
                                     no_tem[['chl']][[j]],
                                     no_tem[['tdep']][[j]],
                                     fun=psych::geometric.mean)
```

```

}

# Allocate blank list - no dissolved oxygen
nodox_rast <- vector(mode="list", length=length(timesubs))
# Loop through SI rasters to create HSI
for(i in 1:length(timesubs)){
  j <- timesubs[i]
  # Overlay all rasters and take geometric mean HSI per cell
  nodox_rast[[i]] <- raster::overlay(no_dox[['sal']][[j]],
                                     no_dox[['tem']][[j]],
                                     #no_tem[['dox']][[j]],
                                     no_dox[['chl']][[j]],
                                     no_dox[['tdep']][[j]],
                                     fun=psych::geometric.mean)
}

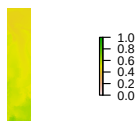
# Allocate blank list - no chlorophyll
nochl_rast <- vector(mode="list", length=length(timesubs))
# Loop through SI rasters to create HSI
for(i in 1:length(timesubs)){
  j <- timesubs[i]
  # Overlay all rasters and take geometric mean HSI per cell
  nochl_rast[[i]] <- raster::overlay(no_chl[['sal']][[j]],
                                     no_chl[['tem']][[j]],
                                     no_chl[['dox']][[j]],
                                     #no_dox[['chl']][[j]],
                                     no_chl[['tdep']][[j]],
                                     fun=psych::geometric.mean)
}

# Allocate blank list - no depth
notdep_rast <- vector(mode="list", length=length(timesubs))
# Loop through SI rasters to create HSI
for(i in 1:length(timesubs)){
  j <- timesubs[i]
  # Overlay all rasters and take geometric mean HSI per cell
  notdep_rast[[i]] <- raster::overlay(no_tdep[['sal']][[j]],
                                     no_tdep[['tem']][[j]],
                                     no_tdep[['dox']][[j]],
                                     no_tdep[['chl']][[j]],
                                     #no_chl[['tdep']][[j]],
                                     fun=psych::geometric.mean)
}

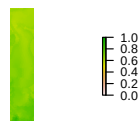
# Combine all leave-one-out results into list
rastcombine <- list(HSI_rast, nochl_rast, nodox_rast, nosal_rast,
                    notdep_rast, notem_rast)
names(rastcombine) <- c("All values", "No chl", "No dox", "No sal",
                       "No dep", "No temp")

```

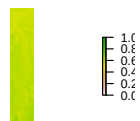
All values



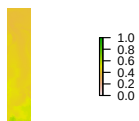
No chl



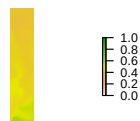
No dox



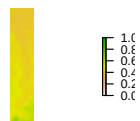
No sal



No dep



No temp



All values



No chl



No dox



No sal



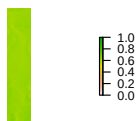
No dep



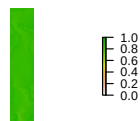
No temp



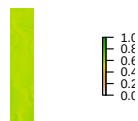
All values



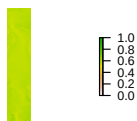
No chl



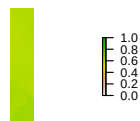
No dox



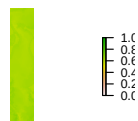
No sal



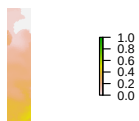
No dep



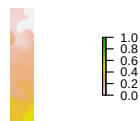
No temp



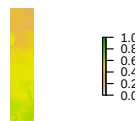
All values



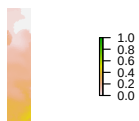
No chl



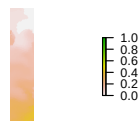
No dox



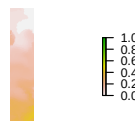
No sal



No dep



No temp



All values



No chl



No dox



No sal



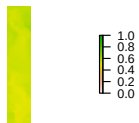
No dep



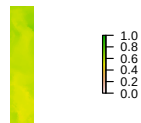
No temp



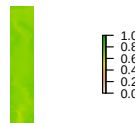
All values



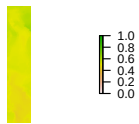
No chl



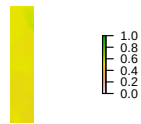
No dox



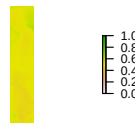
No sal



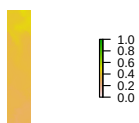
No dep



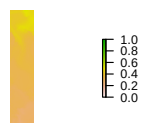
No temp



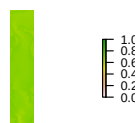
All values



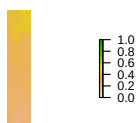
No chl



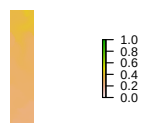
No dox



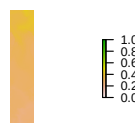
No sal



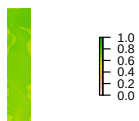
No dep



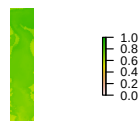
No temp



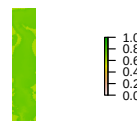
All values



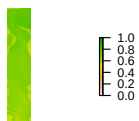
No chl



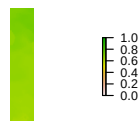
No dox



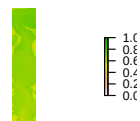
No sal



No dep



No temp



All values



No chl



No dox



No sal



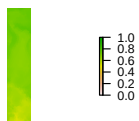
No dep



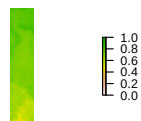
No temp



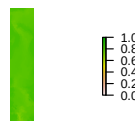
All values



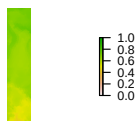
No chl



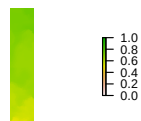
No dox



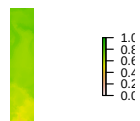
No sal



No dep



No temp



Results

This subset of ten timestamps shows that dissolved oxygen is often the “limiting factor” in habitat suitability. The raster that has no dissolved oxygen input usually has the highest average HSI. Chlorophyll is also a frequent limiting factor, though low chlorophyll values have less of an ability to “tank” the overall HSI as compared to dissolved oxygen. Depth is an important limitation at low tides, and in all other rasters you can see a faint outline of the “shoulders” of the river where the water rapidly shallows out to less than 1m and therefore becomes highly unsuitable.

Temperature and salinity have much less of an impact on HSI. This shouldn’t be surprising– they’re usually a reflection of large-scale processes like tides and seasonal warming. The overall range of temperature and salinity both within a single timestep and over the entire sampling season are limited compared to the other environmental inputs. From this subset, it also appears as though they are highly correlated. One should therefore be removed from the HSI model, as we are essentially “double-counting” the importance of the tidal process in structuring menhaden movement and schooling. Removing salinity would make more sense than removing temperature. The range of salinity in this modeled time period is not likely wide enough to cause noticeable physiological or behavioral changes in menhaden. The range of temperature, on the other hand, is wide enough to cause these changes.

We will save the HSI model without salinity and use that moving forward.

Conclusion

These rasters are the final result of the pre-processing. We can now feed these results into the individual-based model of juvenile menhaden movement in which the menhaden sense and respond to their surrounding environment. A selection of HSI rasters will be animated in a GIF to visualize spatio-temporal suitability trends.

Environmental sensing and movement

Katie Lankowicz

10 March 2022

This document preserves the code used to create an individual-based model of juvenile menhaden movement when the fish sense and respond to their environment, but not each other. A second individual-based model will simulate juvenile menhaden movement when the fish sense and respond to each other, but not to the environment. It is recognized that neither of these two models is likely to be a good representation of actual menhaden movement and aggregation. The end goal is to compare the properties of aggregations resulting from these two simulations to each other AND to empirical data on juvenile menhaden schooling collected from St. Leonard Creek.

The model space, interpolated environmental conditions, and habitat suitability index were created in previous documents. This model is dependent on those data.

Load data

```
# Load original grid and environmental data
load(paste0("G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/",
            "Raster_Data/", 'ncdf_output.RData'))

# Load HSI
hsi <- readRDS(paste0("G:/My Drive/Documents_Backup/Modeling/",
                    "Modeling_RCode/Raster_Data/HSI/results/",
                    "HSI_rast_short.rds"))

# Determine universal spacetime HSI range
newlist <- vector(mode="list", length=length(hsi))
for(i in 1:length(hsi)){
  newlist[[i]] <- hsi[[i]]@data@values
}
hsivals <- do.call(rbind, newlist)
minhsi <- round(min(hsivals),2)
maxhsi <- round(max(hsivals),2)

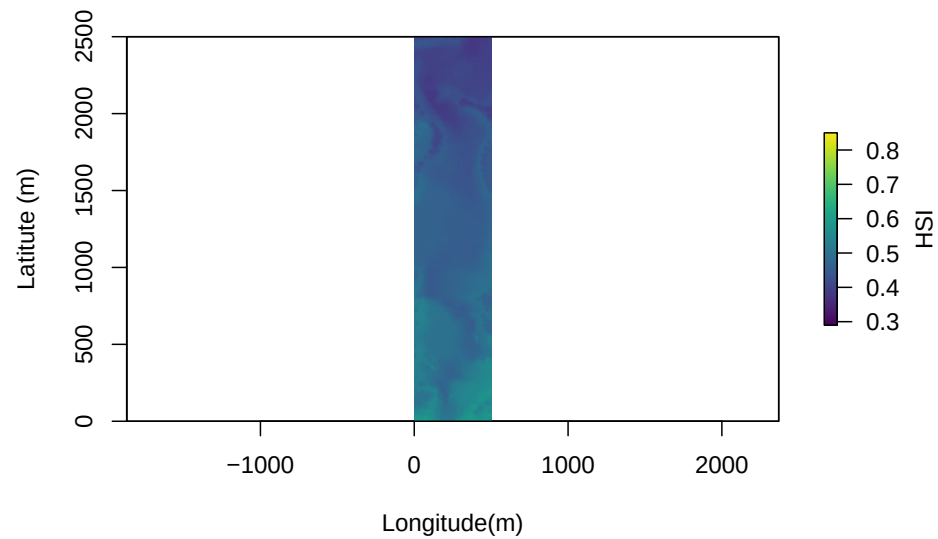
# Change HSI coordinates from projected m lat-lon to simple m
for(i in 1:length(hsi)){
  hsi[[i]]@extent[1] <- 0
  hsi[[i]]@extent[2] <- 500
  hsi[[i]]@extent[3] <- 0
  hsi[[i]]@extent[4] <- 2500
}
```

```

# Set color palette
cols <- viridis(256)

# Check extent change
plot(hsi[[i]],
     col=cols,
     xlim=c(minhsi, maxhsi),
     xlab='Longitude(m)',
     ylab='Latitude (m)',
     legend.args=list(text="HSI", side=4, line=2.5)
)

```



Define raster size

```

n.row <- 250
n.col <- 50

```

Initialize fish

We will begin by initializing 1000 individual fish. These fish will have their own identities, and the environmental conditions they pass through will be preserved for every time step in separate arrays. Many of the initialized parameters are based on values pulled from the literature or assumptions.

We will start with 1000 fish, which is likely to be a little low for a creek similar to the idealized version in the model. We typically identified around 1000 fish per 2.5km survey in the St. Leonard Creek in June 2017, which means there are likely thousands more outside our field of view. We will play around with this number in future versions of the model, and may be able to use it to get a more accurate population index later.

Each fish will be a length between 4 and 6 cm long to start, which is consistent with juvenile menhaden length in the Chesapeake Bay in early May. The exact length of each fish will come from a truncated normal distribution between these values.

Fish will be able to move >0 to 4 body lengths per second. They must move and cannot stay still. The literature states that undisturbed, menhaden typically move up to 2.5 body lengths per second. When placed in a flow-through chamber, they are able to sustain up to 23 body lengths per second. We will set an arbitrary maximum at 4 body lengths per second, and may change it if necessary. The movement speed of each fish will come from a truncated normal distribution between the bounds.

Finally, fish will be able to sense HSI gradients between 5 and 30 meters distance around them in a “visual arc.” The fish cannot see or respond to environments directly behind them, and so can only sense the environment 135 degrees to the left and right of their initial bearing. This makes a 90 degree blind spot directly behind them. The “visual arc” is more suited to the schooling model than the environmental model, but at this stage I have kept it in. It may be removed from this model in the future. The fish must be able to sense outside of the cell they are in. Cells are 10m by 10m resolution, so it is likely that a 5m minimum sensing distance will allow a fish to sense neighboring cells within its “vision arc,” unless it is at the very edge of a cell and facing directly towards the center of the cell. This is unlikely, but future versions of the model may compensate for it by increasing the minimum sensing distance or removing the “vision arc” as discussed earlier. The maximum sensing distance, 30m, is an arbitrary decision. The sense distance for each individual will come from a truncated normal distribution between the minimum and maximum bounds.

The initial x and y locations of each fish are randomly sampled from within the grid space. The initial bearing of the fish is randomly initialized. Keep in mind that R selects angles according to the unit circle, so that a bearing of 0 degrees is parallel to the x-axis of a Cartesian plot and towards the positive direction. This is fine for us. No turn angle is given at initialization, as this is a variable that is calculated by having the fish sense and move towards suitable habitat.

The HSI value for the cell they are, recorded as `score` in is pulled from the HSI raster of the associated hour (recall that the environmental variables were simulated hourly by the SCHISM-ICM).

The model also keeps empty variables that are related to schooling. These are not used in the environmental variable, but instead are important to the schooling model. They have been kept in this model out of blind optimism that one day, I will be able to combine the behavioral and environmental cues into a more comprehensive IBM. These variables are: `nn.idx`, which will record the ID of the nearest neighboring fish, `nn.dist`, which is the distance to that nearest neighbor, `ang`, which is the angle between the bearings of the two fish, and `tru`, which is a flag used to record whether the nearest neighbor is within the “vision arc” and recalculate neighbor identities if it is not.

```

# Number of individuals
n <- 1000

# List the things that we will use to identify individuals
listofthings <- c("x_loc", "y_loc",
                  "body_len",
                  "swimspd",
                  "score",
                  "dist",
                  "bearing",
                  "cellnum",
                  "individual",
                  "prevbear",
                  "turn",
                  "sensedist",
                  "nn.idx",
                  "nn.dist",
                  "ang",
                  "tru")

# Blank array
inds <- array(data = 0, dim = c(n, length(listofthings)))

# Set row names
rn <- as.numeric(1:n)
rownames(inds) <- rn;

# Set variable names
colnames(inds) <- listofthings

# Initiate ID
inds[, "individual"] <- seq(1:n)

# Body length for May expected 0.042-0.055 m
inds[, "body_len"] <- round(rtruncnorm(n=n,
                                     mean=0.0485, sd=0.0065,
                                     a=0.032, b=0.065), 3)

#summary(inds[, "body_len"])

# Initiate positions on xy grid of size 500m, 2500m
inds[, "x_loc"] <- round(runif(n, 0, n.col*10), 1)
inds[, "y_loc"] <- round(runif(n, 0, n.row*10), 1)

# Find cell name in rasters
inds[, "cellnum"] <- cellFromXY(hsi[[1]],
                               cbind(inds[, "x_loc"],
                                       inds[, "y_loc"]))

# Pull conditions at cell
inds[, "score"] <- extract(hsi[[1]],
                           inds[, "cellnum"])

```

```

# Initiate bearing
inds[, "prevbear"] <- round(runif(n=n, min=0, max=359),0)
inds[, "turn"] <- round(rtruncnorm(n=n,
                                mean=0, sd=67.5,
                                a=-135, b=135),0)
inds[, "bearing"] <- inds[, "prevbear"] + inds[, "turn"]
inds[, "turn"] <- NA

# Set sense distance
inds[, "sensedist"] <- round(rtruncnorm(n=n, mean=15, sd=7.5,
                                       a=5, b=30),2)

# Randomly determine the bodylengths to move per individ in xy directs.
# Literature states this should be around 0.5-2.5 bl/sec
# Max short burst swim spd is 23 bl/sec but is only seen in lab
# We will set an arbitrary max at 4 bl/sec for now
inds[, "swimspd"] <- (round(rtruncnorm(n=n, mean=1, sd=1,
                                       a=0, b=4),1))

for(i in 1:n){
  while(inds[i, 'swimspd'] == 0){
    inds[i, "swimspd"] <- (round(rtruncnorm(n=1, mean=1, sd=1,
                                           a=0, b=4),1))
  }
}

# Find coterminal angle within unit circle
for(i in 1:n){
  if(inds[i, "bearing"] < 0){
    inds[i, "bearing"] <- inds[i, "bearing"] + 360
  }
  if(inds[i, "bearing"] > 360){
    inds[i, "bearing"] <- inds[i, "bearing"] - 360
  }
}
rm(i)
#summary(inds[, "bearing"])

# Set sequence of angles that a fish can turn
# This is 135 deg left or right relative to current bearing
bearseq <- seq(0, 135, by=1)

```

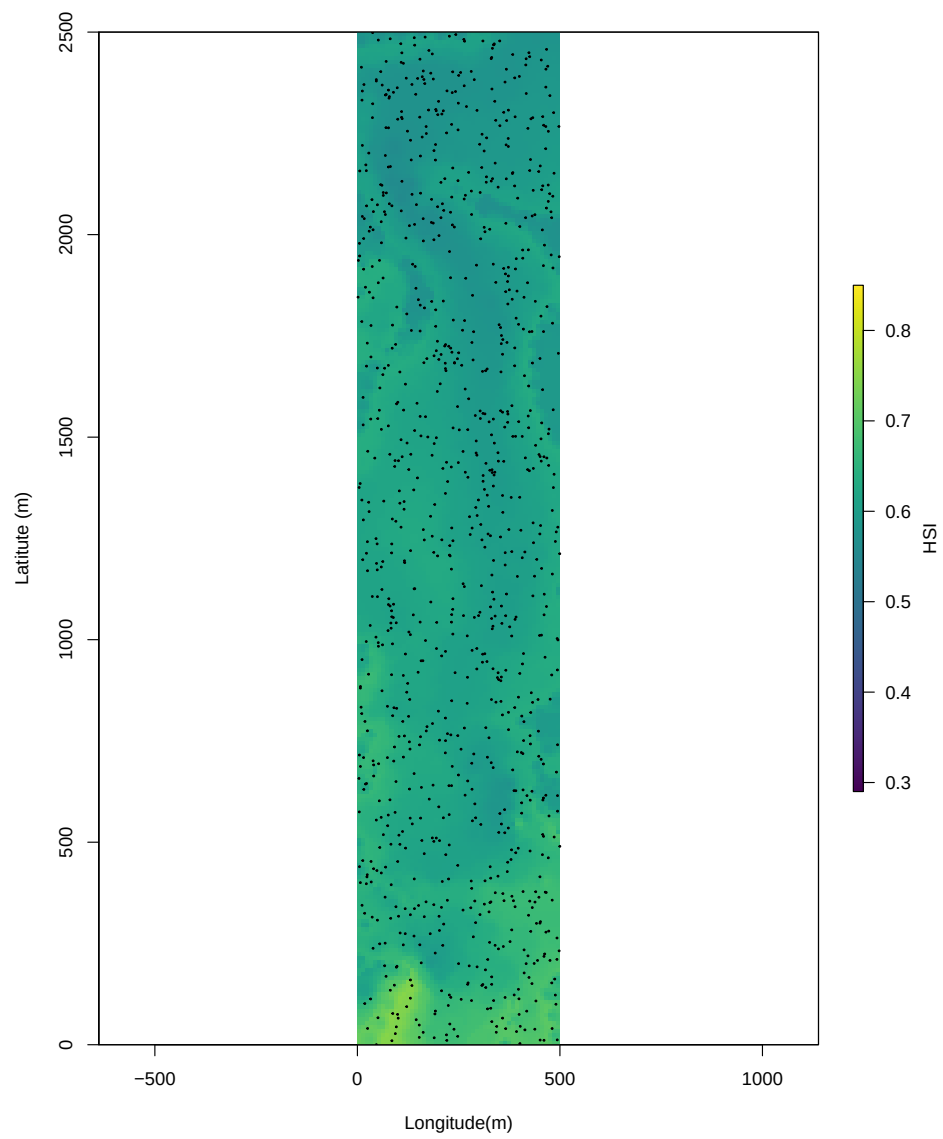
```
summary(inds)
```

```
##      x_loc      y_loc      body_len      swimspd
## Min.   : 1.8   Min.   : 0.2   Min.   :0.03200   Min.   :0.100
## 1st Qu.:118.7 1st Qu.: 615.7 1st Qu.:0.04400 1st Qu.:0.700
## Median :249.8 Median :1263.5 Median :0.04900 Median :1.200
## Mean   :250.9 Mean   :1260.8 Mean   :0.04857 Mean   :1.302
## 3rd Qu.:373.1 3rd Qu.:1919.8 3rd Qu.:0.05300 3rd Qu.:1.800
## Max.   :499.8 Max.   :2498.7 Max.   :0.06500 Max.   :3.900
##
##      score      dist      bearing      cellnum      individual
## Min.   :0.5593   Min.   :0   Min.   : 0.00   Min.   : 4   Min.   : 1.0
## 1st Qu.:0.5949   1st Qu.:0   1st Qu.: 88.75   1st Qu.: 2900   1st Qu.: 250.8
## Median :0.6143   Median :0   Median :183.00   Median : 6190   Median : 500.5
## Mean   :0.6174   Mean   :0   Mean   :179.05   Mean   : 6196   Mean   : 500.5
## 3rd Qu.:0.6318   3rd Qu.:0   3rd Qu.:260.00   3rd Qu.: 9443   3rd Qu.: 750.2
## Max.   :0.7472   Max.   :0   Max.   :360.00   Max.   :12491   Max.   :1000.0
##
##      prevbear      turn      sensedist      nn.idx      nn.dist
## Min.   : 0.0   Min.   : NA   Min.   : 5.11   Min.   :0   Min.   :0
## 1st Qu.: 91.5   1st Qu.: NA   1st Qu.:11.29   1st Qu.:0   1st Qu.:0
## Median :175.0   Median : NA   Median :15.74   Median :0   Median :0
## Mean   :176.0   Mean   :NaN   Mean   :15.90   Mean   :0   Mean   :0
## 3rd Qu.:263.0   3rd Qu.: NA   3rd Qu.:19.95   3rd Qu.:0   3rd Qu.:0
## Max.   :359.0   Max.   : NA   Max.   :29.95   Max.   :0   Max.   :0
##
##      NA's      :1000
##
##      ang      tru
## Min.   :0   Min.   :0
## 1st Qu.:0   1st Qu.:0
## Median :0   Median :0
## Mean   :0   Mean   :0
## 3rd Qu.:0   3rd Qu.:0
## Max.   :0   Max.   :0
##
```

Plot position of fish

The initial positions of the fish can be plotted on the HSI raster for the first time step. Menhaden are schooling fish, so it is not ecologically sensible that so many are off on their own. Both this model and the schooling model will likely need a significant “spool up” period to get to more ecologically realistic aggregations. We may consider using the entire month of May as a spool up period, since we have environmental data and one field sample taken in that month.

```
# Plot original positions of fish
plot(hsi[[1]],
     asp=1,
     col=cols,
     zlim=c(minhsi, maxhsi),
     xlab='Longitude(m)',
     ylab='Latitude (m)',
     legend.args=list(text="HSI", side=4, line=2.5)
)
points(x = inds["x_loc"], y = inds["y_loc"], pch = 20, cex = 0.3)
```



Sensing function

This function contains all the rules for sensing the environment. The next function will actually move the fish according to the results of this function. Comments on the specifics of the function are made throughout the code.

```
sense <- function(inds){  
  # Set sequence of angles that a fish can turn  
  # This is 135 deg left or right relative to current bearing  
  # Fish cannot "see" directly behind themselves  
  bearseq <- seq(0, 135, by=1)  
  
  # Loop through individuals  
  for(i in 1:n){  
  
    # Record initial position, bearing, sense distance  
    og_xloc <- inds[i,"x_loc"]  
    og_yloc <- inds[i,"y_loc"]  
    og_bear <- inds[i,"bearing"]  
    sensedist <- inds[i,"sensedist"]  
  
    # Create vectors to store info for adjacent cells looking right  
    r_x_loc <- vector(mode="numeric", length=length(bearseq))  
    r_y_loc <- vector(mode="numeric", length=length(bearseq))  
    r_cellnum <- vector(mode="numeric", length=length(bearseq))  
    r_bearing <- bearseq  
  
    # Create vectors to store info for adjacent cells looking left  
    l_x_loc <- vector(mode="numeric", length=length(bearseq))  
    l_y_loc <- vector(mode="numeric", length=length(bearseq))  
    l_cellnum <- vector(mode="numeric", length=length(bearseq))  
    l_bearing <- -1*bearseq  
  
    # Loop through all possible turning angles  
    for(k in 1:length(bearseq)){  
      # Sense looking right, x location  
      r_x_loc[k] <- og_xloc + sensedist *  
        cos(deg2rad(og_bear - bearseq[k]))  
      # Sense looking right, y location  
      r_y_loc[k] <- og_yloc + sensedist *  
        sin(deg2rad(og_bear - bearseq[k]))  
  
      # If near the edge of the grid, sense around torus shape  
      # Sense below 0 in x direction  
      if(r_x_loc[k]<0){  
        r_x_loc[k] <- r_x_loc[k]+500  
      }  
  
      # Sense below 0 in y direction  
      if(r_y_loc[k]<0){  
        r_y_loc[k] <- r_y_loc[k]+2500  
      }  
      # Sense above x limit (500)
```

```

if(r_x_loc[k]>500){
  r_x_loc[k] <- r_x_loc[k]-500
}
# Sense above y limit (2500)
if(r_y_loc[k]>2500){
  r_y_loc[k] <- r_y_loc[k]-2500
}

# Pull cell numbers from xy location of sensing radius
r_cellnum[k] <- cellFromXY(estov,
                           cbind(r_x_loc[k],
                                   r_y_loc[k]))

# Repeat process looking left
# Sense looking right, x location
l_x_loc[k] <- og_xloc + sensedist *
  cos(deg2rad(og_bear - (-1*bearseq[k])))
# Sense looking right, y location
l_y_loc[k] <- og_yloc + sensedist *
  sin(deg2rad(og_bear - (-1*bearseq[k])))

# If near the edge of the grid, sense around torus shape
# Sense below 0 in x direction
if(l_x_loc[k]<0){
  l_x_loc[k] <- l_x_loc[k]+500
}
# Sense below 0 in y direction
if(l_y_loc[k]<0){
  l_y_loc[k] <- l_y_loc[k]+2500
}
# Sense above x limit (500)
if(l_x_loc[k]>500){
  l_x_loc[k] <- l_x_loc[k]-500
}
# Sense above y limit (2500)
if(l_y_loc[k]>2500){
  l_y_loc[k] <- l_y_loc[k]-2500
}

# Pull cell numbers from xy location of sensing radius
l_cellnum[k] <- cellFromXY(estov,
                           cbind(l_x_loc[k],
                                   l_y_loc[k]))
}

# Combine cellnums and bearings
pick <- unique(cbind(c(r_cellnum, l_cellnum),
                     c(bearseq, (-1*bearseq))))

# Combine into single vector
cellnums <- c(r_cellnum, l_cellnum)

# Pull HSI values for "sensed" cells

```

```

scorenums <- estov[cellnums]

# Sense cell fish is currently in
homecell <- inds[i,"cellnum"]
homescore <- inds[i,"score"]

# Combine sensed cells and home cell in a single array
allcells <- c(cellnums, homecell)
allscores <- c(scorenums, homescore)
cells <- cbind(allcells, allscores)

# Remove duplicates
cells <- unique(cells)

# Determine the cell number of the "best" nearby cell by HSI
best <- cells[(cells[, "allscores"])==max(cells[, "allscores"]),][[1]]

# If current cell is better than surrounding cells, turn randomly
if(best==homecell){
  inds[i,"turn"] <- round(rtruncnorm(n=1,
                                     mean=0, sd=67.5,
                                     a=-135, b=135),0)
}

# If another cell is best, pick random bearing towards best cell
if(best!=homecell){
  inds[i,"turn"] <- sample(pick[,2][pick[,1]==best],1)
}
}

# Append results to array of individuals
return(inds);
}

```

Movement function

Now that we have determined the direction that each fish should move in, we must make them move that distance. This function is less complicated, and simply adds the distance that the fish can cover in the timestep given its body length and swim speed (in body lengths per second) to the fish's original position, along the calculated bearing.

```
movement <- function(inds){  
  # Get the number of individuals in array of individuals  
  total_inds <- dim(inds)[1]  
  
  # Calculate physical distance to move  
  inds[, "dist"] <- inds[, "body_len"] * (inds[, "swimspd"] * s)  
  
  # Copy current bearing (from last time step) to previous column  
  inds[, "prevbear"] <- inds[, "bearing"]  
  
  # Calculate new bearing given turning angle from sense function  
  inds[, "bearing"] <- inds[, "prevbear"] + inds[, "turn"]  
  
  # Calculate new xy locations for the fish given bearing and swim distance  
  inds[, "x_loc"] <- inds[, "x_loc"] +  
    inds[, "dist"] * cos(deg2rad(inds[, "bearing"]))  
  inds[, "y_loc"] <- inds[, "y_loc"] +  
    inds[, "dist"] * sin(deg2rad(inds[, "bearing"]))  
  
  # ===== Torus setup =====  
  for(i in 1:n){  
    # Wrap around if move off left edge  
    if(inds[i, "x_loc"] < 0){  
      inds[i, "x_loc"] <- inds[i, "x_loc"] + 500  
    }  
    # Wrap around if move off bottom edge  
    if(inds[i, "y_loc"] < 0){  
      inds[i, "y_loc"] <- inds[i, "y_loc"] + 2500  
    }  
    # Wrap around if move off right edge  
    if(inds[i, "x_loc"] > 500){  
      inds[i, "x_loc"] <- inds[i, "x_loc"] - 500  
    }  
    # Wrap around if move off top edge  
    if(inds[i, "y_loc"] > 2500){  
      inds[i, "y_loc"] <- inds[i, "y_loc"] - 2500  
    }  
  }  
  
  # Append results to array of individuals  
  return(inds);  
}
```

Simulate movement

Both functions have been set up. Now, we can use them to run the model for a desired number of time steps. Whereas the schooling model needs a very fine time step (around 2 seconds) for the fish to be able to rapidly respond to each other. The environmental rasters are static for an hour, and therefore having them sense and respond to the environment less frequently should be fine. The choice of a 30 second timestep was arbitrary, and it may be adjusted to optimize the model or produce more ecologically realistic results.

Though the code has been optimized for computational strain, the model is still fairly complicated. It takes around 11 seconds to run a loop (which represents 30 seconds) of the model for 1000 individuals. We will start by running for a week (or approximately 20,000 time steps of 30 seconds each) and check the results. This should take around 2.5 days of computing time. The end goal is to have three months of modeled time—June, July, and August, which matches the temporal extent of our empirical data.

The model begins by making a large blank array which will hold the results for each time step for each individual. Next, the HSI raster is pulled for the hour in which the time step occurs. Then, the sense and move functions are used to send the individuals forward in time and space. The results are stored in the large array. New sense distances and swim speeds are drawn for the individual fish from the same range of values as the initialization. Re-drawing these values adds some stochasticity to the model and prevents the fish from moving the same distance every time step, which is unrealistic. Then, the loop for the current time step is ended and the model is advanced by a time step.

```
# Frequency (in seconds) to update movement
s <- 30

# Set timesteps
desired_hours <- 7*24
time_steps <- (60/s)*60*desired_hours
ts <- 1

# Initialize blank history list by individual
fakeinds <- as.data.frame(inds)
inds_hist <- split(fakeinds, f=fakeinds$individual)

# Initialize progress bar
pb <- txtProgressBar(min=0, max=time_steps, initial=0, char='=', style=3)

# Simulate time passing
while(ts < time_steps){
  estov <- hsi[[ceiling(ts/60/(60/s))]]
  inds <- sense(inds)
  inds <- movement(inds);
  ts <- ts + 1;

  for(i in 1:n){
    # Append stepwise results to history list
    inds_hist[[i]][ts,] <- inds[i,]
  }

  # Generate new swim speed and sense distance
  inds[, "sensedist"] <- round(rtruncnorm(n=n, mean=15, sd=7.5,
                                         a=5, b=30), 2)
```

```

inds[, "swimspd"] <- (round(rtruncnorm(n=n, mean=1, sd=1,
                                     a=0, b=4), 1))

for(i in 1:n){
  while(inds[i, 'swimspd'] == 0){
    inds[i, "swimspd"] <- (round(rtruncnorm(n=1, mean=1, sd=1,
                                             a=0, b=4), 1))
  }
}

# Update progress bar
setTxtProgressBar(pb, ts)
getTxtProgressBar(pb)
# Close segment of progress bar every loop
close(pb)
}

```

Conclusions and predictions

I hypothesize that the model will fail to produce aggregations, and certainly will fail to produce polarized-swimming schools akin to the juvenile menhaden schools we have observed in St. Leonard Creek. This is partially due to issues in the modeling parameters— we do not have a spool-up period, and so the fish are unreasonably started at random. Some of our assumptions along the way may be ecologically unreasonable, although we have tried to reduce this possibility as much as possible through literature review.

Even if fish were started in clusters, they would not likely retain that spatial structure. The fish sense the environment and move independently of each other, and with variance built in to their “physical” sensing and moving capabilities. The presence of any variance at all is likely realistic, although we may not have the values correct. However, one of the known benefits of schooling is rapid information transfer. While menhaden are not “social” creatures in the way we think of other humans, dolphins, or even dogs, they do still recognize and respond to each other. Without that capability built into the model, the probability of menhaden converging on the same area and orienting themselves in the same direction is greatly reduced.

Bonus: saving to animation

```
setwd(paste0("G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/",
             "GIFs/env_mov/subset"))

for(i in 1:time_steps){
  timest <- '2006-06-01 00:00:00'
  timest <- as.POSIXct(timest,
                       format="%Y-%m-%d %H:%M:%S",
                       tz="America/New_York")
  timest <- timest + seconds(i*s)
  timest <- gsub(":", "-", timest)
  png(paste0(timest, '.png'))
  raster::plot(hsi[[ceiling(i/60/(60/s))]],
               asp=1,
               col=cols,
               xlab='Longitude (m)',
               ylab='Latitude (m)',
               zlim=c(minhsi, maxhsi),
               legend.args=list(text="HSI", side=4, line=2.5),
               main=paste0(timest))
  for(k in 1:n){
    points(inds_hist[[k]]$x_loc[i],
           inds_hist[[k]]$y_loc[i],
           pch=19,
           cex=0.5)
  }
  dev.off()
}
files <- list.files()
files <- paste0(getwd(), '/', files)

setwd("G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/GIFs")
gifski(files, "env_mov.gif", loop = FALSE, delay = 0.01)
```

Schooling Movement

Katie Lankowicz

11 March 2022

This document preserves the code used to create an individual-based model of juvenile menhaden movement when the fish sense and respond to each other, but not to their environment. A second individual-based model will simulate juvenile menhaden movement when the fish sense and respond to the environment, but not to each other. It is recognized that neither of these two models is likely to be a good representation of actual menhaden movement and aggregation. The end goal is to compare the properties of aggregations resulting from these two simulations to each other AND to empirical data on juvenile menhaden schooling collected from St. Leonard Creek.

The model space and the initial parameters for the set of fish were created in a different document. This model is dependent on those data.

Load model space

```
load(paste0('G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/',
            'Raster_Data/ncdf_output.RData'))

# Change raster coordinates from projected m lat-lon to simple m
r@extent[1] <- 0
r@extent[2] <- 500
r@extent[3] <- 0
r@extent[4] <- 2500

# Set rows and columns
n.row <- r@nrows
n.col <- r@ncols
```

Initialize fish

We will begin by initializing 1000 individual fish. These fish will have their own identities, and their positions relative to each other will be preserved for every time step. Many of the initialized parameters are based on values pulled from the literature or assumptions.

We will start with 1000 fish, which is likely to be a little low for a creek similar to the idealized version in the model. We typically identified around 1000 fish per 2.5km survey in the St. Leonard Creek in June 2017, which means there are likely thousands more outside our field of view. We will play around with this number in future versions of the model, and may be able to use it to get a more accurate population index later.

Each fish will be a length between 4 and 6 cm long to start, which is consistent with juvenile menhaden length in the Chesapeake Bay in early May. The exact length of each fish will come from a truncated normal distribution between these values.

Fish will be able to move >0 to 4 body lengths per second. They must move and cannot stay still. The literature states that undisturbed, menhaden typically move up to 2.5 body lengths per second. When placed in a flow-through chamber, they are able to sustain up to 23 body lengths per second. We will set an arbitrary maximum at 4 body lengths per second, and may change it if necessary. The movement speed of each fish will come from a truncated normal distribution between the bounds.

Finally, fish will be able to sense each other between 5 and 30 meters distance around them in a “visual arc.” The fish cannot see or respond to other fish directly behind them, and so can only sense the environment 135 degrees to the left and right of their initial bearing. This makes a 90 degree blind spot directly behind them. The fish must be able to sense outside of the cell they are in. Cells are 10m by 10m resolution, so it is likely that a 5m minimum sensing distance will allow a fish to sense neighboring cells within its “vision arc,” unless it is at the very edge of a cell and facing directly towards the center of the cell. This is unlikely, but future versions of the model may compensate for it by increasing the minimum sensing distance or removing the “vision arc” as discussed earlier. The maximum sensing distance, 30m, is an arbitrary decision. The sense distance for each individual will come from a truncated normal distribution between the minimum and maximum bounds.

The initial x and y locations of each fish are randomly sampled from within the grid space. The initial bearing of the fish is randomly initialized. Keep in mind that R selects angles according to the unit circle, so that a bearing of 0 degrees is parallel to the x-axis of a Cartesian plot and towards the positive direction. This is fine for us. No turn angle is given at initialization, as this is a variable that is calculated by having the fish sense and move towards suitable habitat.

The model retains an empty space for HSI value of the raster cell that the fish are in, recorded as **score**. It pulled from the HSI raster of the associated hour (recall that the environmental variables were simulated hourly by the SCHISM-ICM). This model does not have fish sense the environment, so it will remain blank. It's kept in the model out of pure optimism that one day, I'll be able to combine behavioral and environmental cues into a more comprehensive IBM.

The model also specifies variables that are related to schooling. These variables are: **nn.idx**, which will record the ID of the nearest neighboring fish, **nn.dist**, which is the distance to that nearest neighbor, **ang**, which is the angle between the bearings of the two fish, and **tru**, which is a flag used to record whether the nearest neighbor is within the “vision arc” and recalculate neighbor identities if it is not.

```

# Number of individuals
n <- 1000

# List the things that we will use to identify individuals
listofthings <- c("x_loc", "y_loc",
                  "body_len",
                  "swimspd",
                  "score",
                  "dist",
                  "bearing",
                  "cellnum",
                  "individual",
                  "prevbear",
                  "turn",
                  "sensedist",
                  "nn.idx",
                  "nn.dist",
                  "ang",
                  "tru")

# Blank array
inds <- array(data = 0, dim = c(n, length(listofthings)))

# Set row names
rn <- as.numeric(1:n)
rownames(inds) <- rn;

# Set variable names
colnames(inds) <- listofthings

# Initiate ID
inds[, "individual"] <- seq(1:n)

# Body length for May expected 0.042-0.055 m
inds[, "body_len"] <- round(rtruncnorm(n=n,
                                     mean=0.0485, sd=0.0065,
                                     a=0.032, b=0.065), 3)

#summary(inds[, "body_len"])

# Initiate positions on xy grid of size 500m, 2500m
inds[, "x_loc"] <- round(runif(n, 0, n.col*10), 1)
inds[, "y_loc"] <- round(runif(n, 0, n.row*10), 1)

# Find cell name in rasters
inds[, "cellnum"] <- cellFromXY(r,
                               cbind(inds[, "x_loc"],
                                       inds[, "y_loc"]))

```

```

# Initiate bearing
inds[, "prevbear"] <- round(runif(n=n, min=0, max=359),0)
inds[, "turn"] <- round(rtruncnorm(n=n,
                                mean=0, sd=67.5,
                                a=-135, b=135),0)

inds[, "bearing"] <- inds[, "prevbear"] + inds[, "turn"]
inds[, "turn"] <- NA

# Set sense distance
inds[, "sensedist"] <- round(rtruncnorm(n=n, mean=15, sd=7.5,
                                       a=5, b=30),2)

# Randomly determine the bodylengths to move per individ in xy directs.
# Literature states this should be around 0.5-2.5 bl/sec
# Max short burst swim spd is 23 bl/sec but is only seen in lab
# We will set an arbitrary max at 4 bl/sec for now
inds[, "swimspd"] <- (round(rtruncnorm(n=n, mean=1, sd=1,
                                       a=0, b=4),1))

for(i in 1:n){
  while(inds[i, "swimspd"]==0){
    inds[i, "swimspd"]<- (round(rtruncnorm(n=1, mean=1, sd=1,
                                       a=0, b=4),1))
  }
}

# Find coterminal angle within unit circle
for(i in 1:n){
  if(inds[i, "bearing"]<0){
    inds[i, "bearing"] <- inds[i, "bearing"]+360
  }
  if(inds[i, "bearing"]>360){
    inds[i, "bearing"] <- inds[i, "bearing"]-360
  }
}
rm(i)

# Set sequence of angles that a fish can turn
# This is 135 deg left or right relative to current bearing
bearseq <- seq(0, 135, by=1)

# Initialize blank schooling variables
inds[, "nn.idx"] <- 0
inds[, "nn.dist"] <- 0
inds[, "ang"] <- 0
inds[, "tru"] <- 0

```

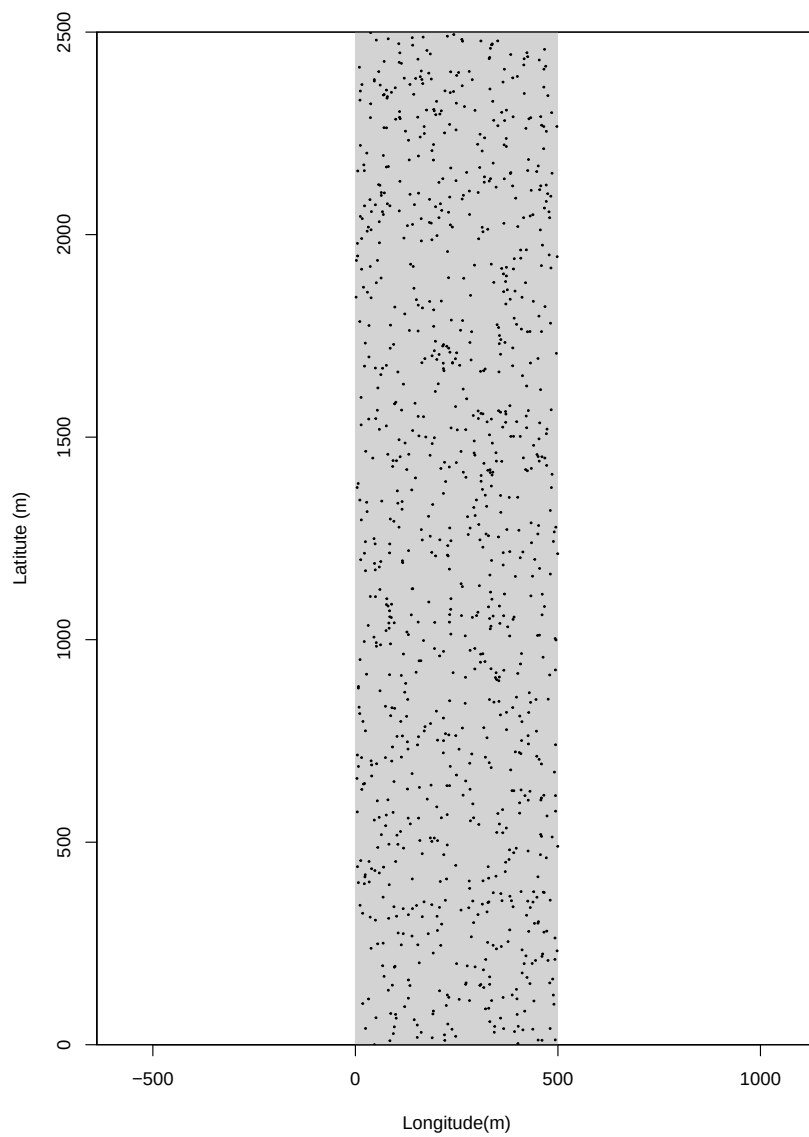
```
summary(inds)
```

```
##      x_loc      y_loc      body_len      swimspd      score
## Min.   : 1.8   Min.   : 0.2   Min.   :0.03200   Min.   :0.100   Min.   :0
## 1st Qu.:118.7 1st Qu.: 615.7 1st Qu.:0.04400 1st Qu.:0.700 1st Qu.:0
## Median :249.8 Median :1263.5 Median :0.04900 Median :1.200 Median :0
## Mean   :250.9 Mean   :1260.8 Mean   :0.04857 Mean   :1.302 Mean   :0
## 3rd Qu.:373.1 3rd Qu.:1919.8 3rd Qu.:0.05300 3rd Qu.:1.800 3rd Qu.:0
## Max.   :499.8 Max.   :2498.7 Max.   :0.06500 Max.   :3.900 Max.   :0
##
##      dist      bearing      cellnum      individual      prevbear
## Min.   :0   Min.   : 0.00   Min.   : 4   Min.   : 1.0   Min.   : 0.0
## 1st Qu.:0   1st Qu.: 88.75   1st Qu.: 2900   1st Qu.: 250.8   1st Qu.: 91.5
## Median :0   Median :183.00   Median : 6190   Median : 500.5   Median :175.0
## Mean   :0   Mean   :179.05   Mean   : 6196   Mean   : 500.5   Mean   :176.0
## 3rd Qu.:0   3rd Qu.:260.00   3rd Qu.: 9443   3rd Qu.: 750.2   3rd Qu.:263.0
## Max.   :0   Max.   :360.00   Max.   :12491   Max.   :1000.0   Max.   :359.0
##
##      turn      sensedist      nn.idx      nn.dist      ang      tru
## Min.   : NA   Min.   : 5.11   Min.   :0   Min.   :0   Min.   :0   Min.   :0
## 1st Qu.: NA   1st Qu.:11.29   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0
## Median : NA   Median :15.74   Median :0   Median :0   Median :0   Median :0
## Mean   :NaN   Mean   :15.90   Mean   :0   Mean   :0   Mean   :0   Mean   :0
## 3rd Qu.: NA   3rd Qu.:19.95   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0
## Max.   : NA   Max.   :29.95   Max.   :0   Max.   :0   Max.   :0   Max.   :0
## NA's    :1000
```

Plot position of fish

The initial positions of the fish can be plotted on the HSI raster for the first time step. Menhaden are schooling fish, so it is not ecologically sensible that so many are off on their own. Both this model and the schooling model will likely need a significant “spool up” period to get to more ecologically realistic aggregations. We may consider using the entire month of May as a spool up period, since we have environmental data and one field sample taken in that month.

```
# Plot original positions of fish
plot(r,
      asp=1,
      col="lightgray",
      xlab='Longitude(m)',
      ylab='Latitude (m)',
      legend=FALSE
    )
points(x = inds["x_loc"], y = inds["y_loc"], pch = 20, cex = 0.3)
```



Sensing function

This model is a torus, which means the left and right edges are connected and the top and bottom edges are connected. In the real world, this would look like a doughnut. This is a common modeling method to reduce edge effects and prevent emigration out of the model space. Because we do not “sense” the environment in this model, this is fine. Other methods include reflecting fish off the edges, creating highly unfavorable buffers to reduce the likelihood of wrapping at the edges, or simply throwing the fish back in the model space at a random point if they move off. We want to preserve schooling behavior as much as possible, so we have chosen the least disruptive way to treat the model space.

This function contains all the rules for sensing the environment. The next function will actually move the fish according to the results of this function. Comments on the specifics of the function are made throughout the code.

```
sense <- function(inds){
  # Create data frame of fish positions
  inds.df <- data.frame(inds[, "x_loc"], inds[, "y_loc"],
                        inds[, "individual"])
  names(inds.df) <- c('x', 'y', 'ID')

  # Create ID key to merge by
  inds.df <- inds.df %>% mutate(k = 1)

  # Matrix of all possible pointwise distances
  # NOTE: THIS DOES NOT WRAP AROUND TORUS. FIX LATER.
  m <- inds.df %>%
    full_join(inds.df, by = "k") %>%
    filter(ID.x != ID.y) %>%
    mutate(dist = sqrt((x.x - x.y)^2 + (y.x - y.y)^2))

  # Reduce size by deleting pointwise pairs further away than sensedist
  m <- m %>%
    filter(dist < max(inds[, "sensedist"]))

  # Set torus ranges
  ranges <- rbind(c(r@extent[1], r@extent[2]),
                  c(r@extent[3], r@extent[4]))

  # Use nnt (donut pkg) and knn (nabor pkg) to find nn within sensedist
  # Loop through all fish
  for(i in 1:n){
    # Call neighbors within fish's sensedist
    neighbs <- m %>%
      filter(ID.x == i) %>%
      filter(dist < inds[i, "sensedist"])

    # If no nearby fish, move to next individual
    if(nrow(neighbs) == 0){
      next()
    }

    # Set number of neighbors to those within sensedist
  }
}
```

```

# Add an additional 10 fish to sense for: maybe they're around torus?
k=nrow(neighbs) + 10

# Call position of fish
query <- inds.df[i,1:2]

# Find ID of and distance to nearest fish
res2 <- nnt(inds.df[,1:2],
           query, k=k, fn=nabor::knn, torus=1:2, ranges=ranges,
           radius=inds[i,"sensedist"])

# Store IDs
nnb <- unlist(res2$nn.idx[2:length(res2$nn.idx)])
nnb <- nnb[nnb!=0]

# Store distances
nnd <- unlist(res2$nn.dists[2:length(res2$nn.idx)])
nnd <- nnd[is.infinite(nnd)==FALSE]

# Cycle through nearest neighbors to find closest in vision arc
# If nearest neighbor in vision arc is not yet identified
if(inds[i,"tru"]!=1){
  # Cycle through list of identified neighbors
  for(z in 1:length(nnb)){
    # Calculate bearing to neighbor z
    b <- bearing(inds[i,"x_loc"],
                 inds[i,"y_loc"],
                 inds[nnb[z],"x_loc"],
                 inds[nnb[z],"y_loc"])

    # Calculate angular difference to neighbor z
    ang <- round(360-(90-b),0)
    # Find coterminal angle in unit circle
    if(as.numeric(ang)>360){
      ang <- as.numeric(ang)-360
    }
    # Maximum distance fish can see to left and right
    sl <- as.numeric(inds[i,"bearing"])+135
    sr <- as.numeric(inds[i,"bearing"])-135

    # All possible bearings the fish can turn
    sll <- seq(as.numeric(inds[i,"bearing"]),sl, by=1)
    srr <- seq(sr, as.numeric(inds[i,"bearing"]), by=1)

    # Find coterminal angles in unit circle
    for(q in 1:length(sll)){
      if(sll[q]>360){
        sll[q] <- sll[q]-360
      }
    }
    for(j in 1:length(srr)){
      if(srr[j]<0){
        srr[j] <- srr[j]+360
      }
    }
  }
}

```

```

    }

    # Create vector of "good" angles that coincide with vision arc
    goodangs <- c(sll, srr)
    goodangs <- goodangs[order(goodangs)]
    goodangs <- unique(goodangs)

    # If angle to attract/ align with nearest neighbor is within arc
    if(ang %in% goodangs){
      # Nearest neighbor ID copied
      inds[i,"nn.idx"] <- nnb[z]
      # Distance to nearest neighbor copied
      inds[i,"nn.dist"] <- nnd[z]
      # Angle to attract/ align copied
      inds[i,"ang"] <- ang
      # Flag set to 1
      inds[i,"tru"] <- 1
    }

    # If angle to attract/ align with nearest neighbor isn't within arc
    if(ang %in% goodangs == FALSE){
      # Flag remains at 0
      inds[i,"tru"] <- 0
    }

    # If there are no neighbors within vision arc
    if(ang %in% goodangs == FALSE & z==length(nnb)){
      # Move to next loop iteration
      next()
    }
  }
}
}
# Append results to array of individuals
return(inds)
}

```

Movement function

Now that we have determined the direction that each fish should move in, we must make them move that distance. This function is less complicated, and simply adds the distance that the fish can cover in the timestep given its body length and swim speed (in body lengths per second) to the fish's original position, along the calculated bearing.

```
movement <- function(inds){  
  # Get the number of individuals in inds  
  total_inds <- dim(inds)[1]  
  # Calculate physical distance in xy directs  
  inds[, "dist"] <- inds[, "body_len"] * (inds[, "swimspd"] * s)  
  
  # Set prevbear to bear  
  inds[, "prevbear"] <- inds[, "bearing"]  
  
  # Set attract distance  
  # If nearest neighbor is further away than this, they'll come together  
  q3 <- 1  
  
  # Loop through fish  
  for(i in 1:n){  
    # If distance to nearest neighbor is within alignment distance  
    if(inds[i, "nn.dist"] >= q3){  
      # Calculate bearing between the two  
      b <- bearing(inds[i, "x_loc"],  
                   inds[i, "y_loc"],  
                   inds[inds[i, "nn.idx"], "x_loc"],  
                   inds[inds[i, "nn.idx"], "y_loc"])  
  
      # Assign bearing  
      inds[i, "bearing"] <- round(360-(90-b), 0)  
  
      # Find coterminal angle on unit circle  
      if(as.numeric(inds[i, "bearing"]) > 360){  
        inds[i, "bearing"] <- as.numeric(inds[i, "bearing"])-360  
      }  
      if(as.numeric(inds[i, "bearing"]) < 0){  
        inds[i, "bearing"] <- as.numeric(inds[i, "bearing"])+360  
      }  
  
      # Set turning angle the fish needs to move  
      inds[i, "turn"] <- inds[i, "bearing"] - inds[i, "prevbear"]  
    }  
  }  
  rm(i)  
  
  # Repel  
  # If fish are too close (within a centimeter), they'll be repelled  
  # Loop through fish  
  for(i in 1:n){  
    # Set repel distance  
    if(inds[i, "nn.dist"] != 0 & inds[i, "nn.dist"] <= 0.01){
```

```

# Initialize coin: determines if they move right or left
coin <- c(1, -1)
# Flip the coin
cf <- sample(coin,size=1)
# List of potential turns away from too-close neighbor
p_turn <- seq(45, 135,1)
# Randomly select one
i_turn <- sample(p_turn,1)
# Calculate turning angle
inds[i,"turn"] <- cf*i_turn
# Calculate new bearing
inds[i,"bearing"] <- inds[i,"prevbear"] +inds[i,"turn"]
# Find coterminal angle on unit circle
if(inds[i,"bearing"]<0){
  inds[i,"bearing"] <- inds[i,"bearing"]+360
}
if(inds[i,"bearing"]>360){
  inds[i,"bearing"] <- inds[i,"bearing"]-360
}
}
}
rm(i)

# Align
# If fish are closer than attraction but further than repulsion,
# They will align their bearings for polarized swimming
# Loop through fish
for(i in 1:n){
  # Set alignment distance
  if(inds[i,"nn.dist"]>0.01 & inds[i,"nn.dist"]<q3){
    # Set ID of nearest neighbor
    nnb <- inds[i,"nn.idx"]
    # Set bearing of nearest neighbor
    alb <- inds[nnb,"bearing"]
    # Call bearing
    inb <- inds[i,"bearing"]

    # NOT CURRENTLY IN USE =====
    # Add slight error to bearing alignment
    #errorseq <- seq(-10, 10, 1)
    #errorseq <- errorseq[-11]
    #errortak <- sample(errorseq, 1)
    #inds[i,"bearing"] <- round(((alb + inb) / 2) + errortak,0)
    # =====

    # Find coterminal angle on unit circle
    inds[i,"bearing"] <- round((alb + inb)/2, 0)
    if(as.numeric(inds[i,"bearing"])>360){
      inds[i,"bearing"] <- as.numeric(inds[i,"bearing"])-360
    }
    if(as.numeric(inds[i,"bearing"])<0){
      inds[i,"bearing"] <- as.numeric(inds[i,"bearing"])+360
    }
  }
}

```

```

    # Calculate turning angle
    inds[i,"turn"] <- inds[i,"bearing"] - inds[i,"prevbear"]
    # Find coterminal angle on unit circle
    if(inds[i,"turn"]<(-135)){
      inds[i,"turn"] <- inds[i,"turn"]+360
    }
    if(inds[i,"turn"]>135){
      inds[i,"turn"] <- inds[i,"turn"]-360
    }
  }
}

# Random turning if no nearest neighbor
# Loop through fish
for(i in 1:n){
  # If no nearest neighbor
  if(inds[i,"nn.dist"]==0){
    # Random turn within vision arc
    inds[i,"turn"] <- round(rtruncnorm(n=1,
                                     mean=0, sd=67.5,
                                     a=-135, b=135),0)

    # Calculate new bearing
    inds[i,"bearing"] <- inds[i,"turn"] + inds[i,"prevbear"]
    # Find coterminal angle on unit circle
    if(inds[i,"bearing"]<0){
      inds[i,"bearing"] <- inds[i,"bearing"]+360
    }
    if(inds[i,"bearing"]>360){
      inds[i,"bearing"] <- inds[i,"bearing"]-360
    }
  }
}
rm(i)

# Calculate new xy locations for the fish
inds[, "x_loc"] <- inds[, "x_loc"] +
  inds[, "dist"]*cos(deg2rad(inds[, "bearing"]))
inds[, "y_loc"] <- inds[, "y_loc"] +
  inds[, "dist"]*sin(deg2rad(inds[, "bearing"]))

# ===== Torus setup =====
for(i in 1:n){
  # Wrap around if move off left edge
  if(inds[i,"x_loc"]<0){
    inds[i,"x_loc"] <- inds[i,"x_loc"]+500
  }
  # Wrap around if move off bottom edge
  if(inds[i,"y_loc"]<0){
    inds[i,"y_loc"] <- inds[i,"y_loc"]+2500
  }
  # Wrap around if move off right edge
  if(inds[i,"x_loc"]>500){
    inds[i,"x_loc"] <- inds[i,"x_loc"]-500
  }
}

```

```
}  
# Wrap around if move off top edge  
if(inds[i,"y_loc"]>2500){  
  inds[i,"y_loc"] <- inds[i,"y_loc"]-2500  
}  
}  
# ===== Torus end  
# Append results to array of individuals  
return(inds);  
}
```

Simulate movement

Both functions have been set up. Now, we can use them to run the model for a desired number of time steps. The schooling model needs a very fine time step for the fish to be able to rapidly respond to each other. Modeling efforts by Kenny and his collaborators indicate this value is under 3 seconds. We will set the timestep to 2 seconds.

The model is extremely complicated and computationally difficult. On my desktop, it takes around 10 seconds of computational time to run a loop (which represents 2 seconds of modeled time) of the model for 1000 individuals. We will start by running for a day of modeled time (or approximately 40,000 time steps of 2 seconds each) and check the results. This should take around 5 days of computing time. The end goal is to have three months of modeled time— June, July, and August, which matches the temporal extent of our empirical data. High performance computing will be needed to reach this goal.

The model begins by making a large blank array which will hold the results for each time step for each individual. Then, the sense and move functions are used to send the individuals forward in time and space. The results are stored in the large array. New sense distances and swim speeds are drawn for the individual fish from the same range of values as the initialization. Re-drawing these values adds some stochasticity to the model and prevents the fish from moving the same distance every time step, which is unrealistic. Then, the loop for the current time step is ended and the model is advanced by a time step.

```
# Frequency (in seconds) to update movement
s <- 2

# Set timesteps
desired_hours <- 7*24
time_steps <- (60/s)*60*desired_hours
ts <- 1

# Initialize blank history list by individual
fakeinds <- as.data.frame(inds)
inds_hist <- split(fakeinds, f=fakeinds$individual)

# Initialize progress bar
pb <- txtProgressBar(min=0, max=time_steps, initial=0, char='=', style=3)

# Simulate time passing
while(ts < time_steps){
  inds <- sense(inds)
  inds <- movement(inds);
  ts <- ts + 1;

  for(i in 1:n){
    # Append stepwise results to history list
    inds_hist[[i]][ts,] <- inds[i,]
  }

  # Generate new swim speed and sense distance
  inds[, "sensedist"] <- round(rtruncnorm(n=n, mean=15, sd=7.5,
                                         a=5, b=30), 2)
  inds[, "swimspd"] <- (round(rtruncnorm(n=n, mean=1, sd=1,
```

```

                                a=0, b=4),1))
for(i in 1:n){
  while(inds[i,'swimspd']==0){
    inds[i,"swimspd"]<- (round(rtruncnorm(n=1, mean=1, sd=1,
                                a=0, b=4),1))
  }
}

# Update progress bar
setTxtProgressBar(pb, ts)
getTxtProgressBar(pb)
# Close segment of progress bar every loop
close(pb)
}

```

Conclusions and predictions

This model will produce schools, because that's what we have coded it to do. Some of the remaining questions are how long it will take to produce schools, and what these schools will look like. Will we see many smaller schools, as in the empirical data? Or will our fish clump together into one large school? The sense distance range is large enough so that the fish should remain aggregated once they hit attraction distance. However, the transition between aggregation (or shoal) and a polarized school is a little more difficult. It could be that our alignment distance is too small, and creating polarized schools will be difficult until it is increased.

We should see complete spatial randomness in the distribution of schools, as we have not modeled any reaction to the environment. We will be able to test whether the spatial point pattern of schools comes from either a homogeneous or inhomogeneous Poisson distribution spatial point process model. We can even test whether the best spatial covariate is the y-coordinate, x-coordinate. If I extract the raster depth at every time step, we could use depth as a spatial covariate too. The spatial point pattern modeling for the empirical data showed no dominant point process model in St. Leonard Creek. Fifteen sampling events in St. Leonard Creek were suitable for modeling the spatial point pattern of the schools (had more than 3 schools). Out of those sampling events:

- schools in 1 exhibited complete spatial randomness
- schools in 2 exhibited clustering or repulsion and were best modeled with a homogeneous Poisson point process
- schools in 6 exhibited were best modeled with an inhomogeneous Poisson point process but had no evidence of clustering or repulsion (the spatial covariate in these being the y-coordinate in 5 and depth in 1)
- schools in 6 exhibited clustering or repulsion and were best modeled with an inhomogeneous Poisson point process (the spatial covariate in these being the y-coordinate in 3 and depth in 3)

Once this model finishes running, I intend to identify the resulting schools. The empirical data gave us a sense of the school size and density, so we can use those values to separate distinct schools. I also want to run “digital transects” through the model space on occasion. This would be a subset of the model space of the dimensions of the ARIS field of view, and only the fish within the “digital transect” would be considered in analyses of schools. I suspect that the subset will not affect the results of point pattern modeling of the digital schools, but that it will affect the number and average size of schools that we “see.” Comparing the results of the full model to the results of the “digital transect” subset could be an interesting way to approximate the population size within St. Leonard Creek given the subset of schools that we sampled with the ARIS.

Bonus: saving to animation

```
setwd(paste0("G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/",
             "GIFs/sch_mov/subset"))

for(i in 1:time_steps){
  timest <- '2006-06-01 00:00:00'
  timest <- as.POSIXct(timest,
                       format="%Y-%m-%d %H:%M:%S",
                       tz="America/New_York")
  timest <- timest + seconds(i*s)
  timest <- gsub(":", "-", timest)
  png(paste0(timest, '.png'))
  raster::plot(r,
               asp=1,
               col="lightgray",
               xlab='Longitude (m)',
               ylab='Latitude (m)',
               zlim=c(minhsi, maxhsi),
               legend=FALSE,
               main=paste0(timest))
  for(k in 1:n){
    points(inds_hist[[k]]$x_loc[i],
           inds_hist[[k]]$y_loc[i],
           pch=19,
           cex=0.5)
  }
  dev.off()
}
files <- list.files()
files <- paste0(getwd(), '/', files)

setwd("G:/My Drive/Documents_Backup/Modeling/Modeling_RCode/GIFs")
gifski(files, "sch_mov.gif", loop = FALSE, delay = 0.01)
```