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 1: Grid cells in the Corsica River, provided by Richard Tian

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_Chl"             "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 flucation 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)
```

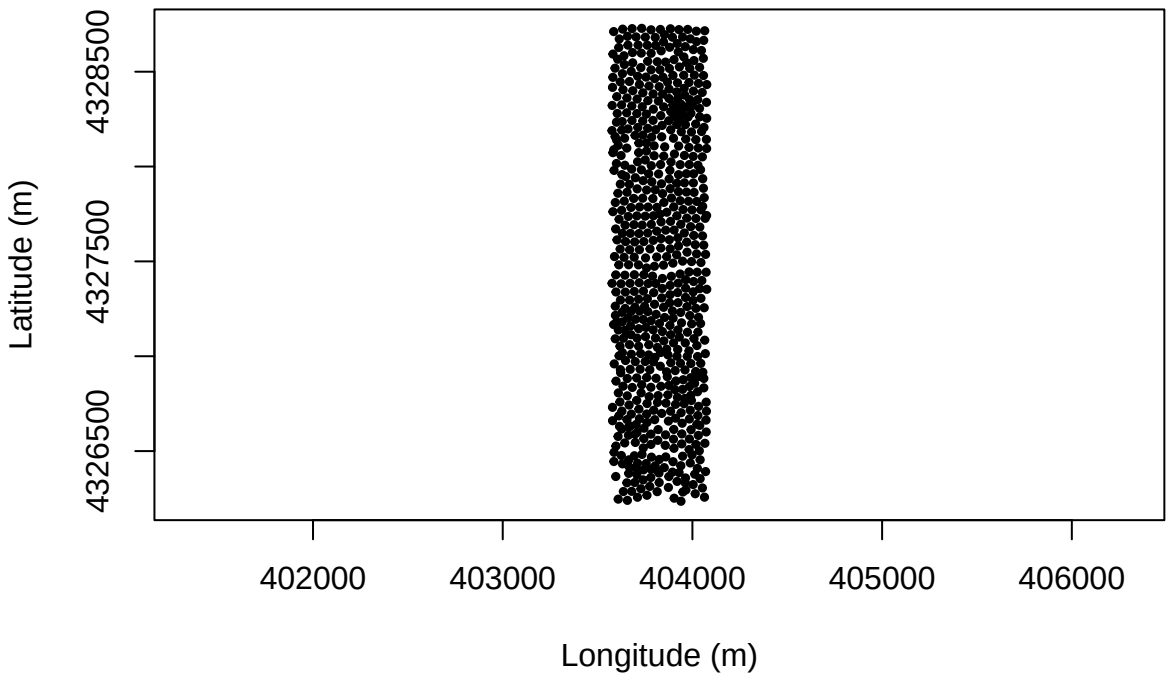

Figure 3: Manipulated grid node positions

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]))
```

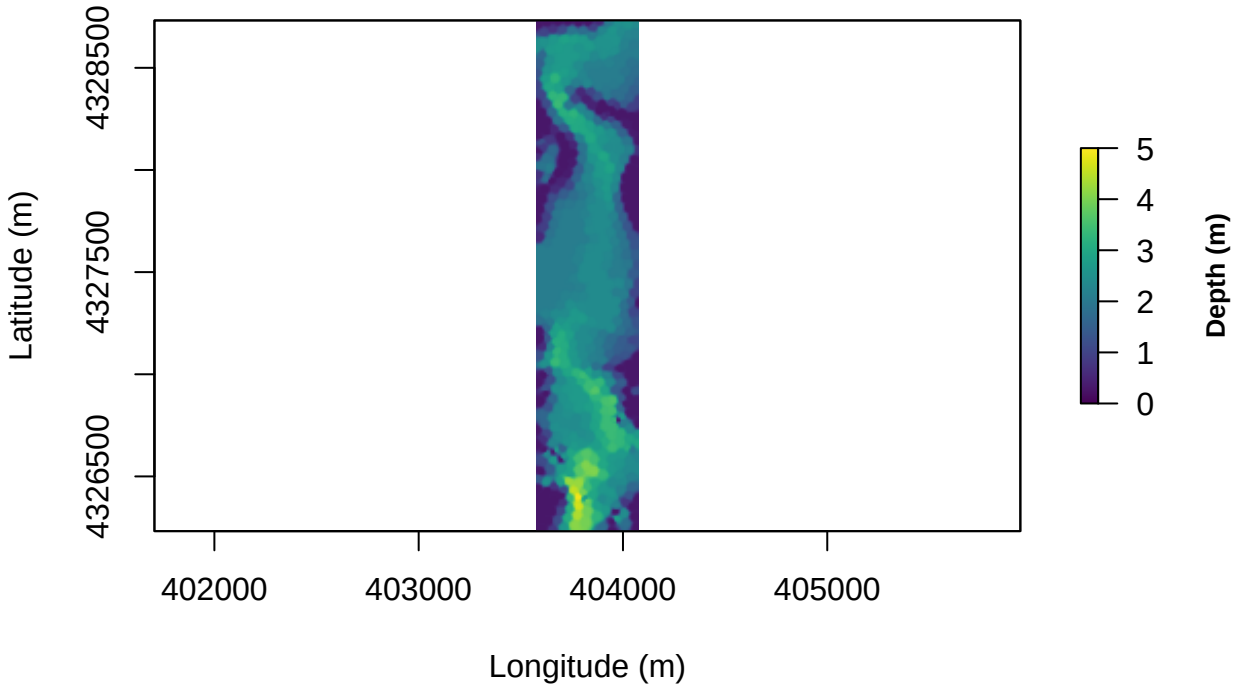

Figure 4: Inverse Distance Weighted Interpolated base depth of raster space

```
## [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.