Geospatial Software Design
“ARCPY”

Andry Rajaoberison,
‘19 MEM

Summary of the tool

WHAT and WHY

Coastal areas are annually exposed to risk of flooding from storm surges.

With the expected increase on storm intensity due to climate change, more and more properties are at risk .

This tool will allow you to identify areas at risk and get insights on the efficiency of their protection.

WHAT you’ll NEED

High resolution elevation data (ideally Lidar)

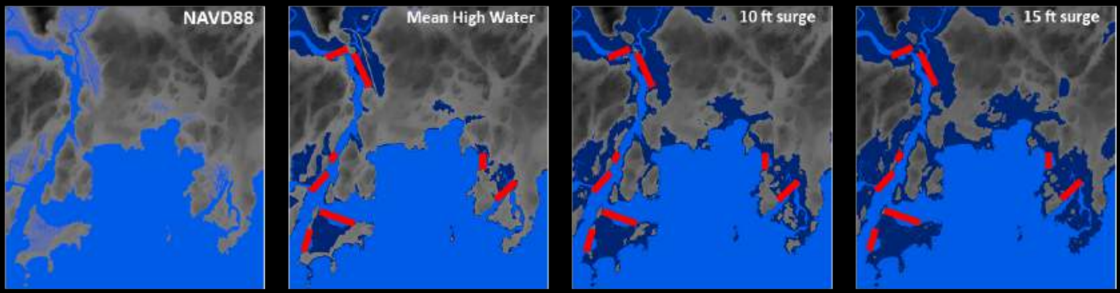
Geospatial data (shapefile) of the properties and their economic values

Expert knowledge of the storm surges on the locality of interest

Here is a short video on how it works: https://youtu.be/8dYU8tFJRV4. The tool and the code are available for download at: http://bit.ly/SeaWallTB and http://bit.ly/SeaWallTBX

Explanation of the tool and the Scripts

- Within a specific coastline there are main entry points where the storm surges can come in.
- Each entry point will affect the decision in the design, nature, and feasibility of the seawall to be established.



- One of the main challenge in coastal defense is the identification of these entry points and consequently the delimitation of the coastal segment for each seawall.

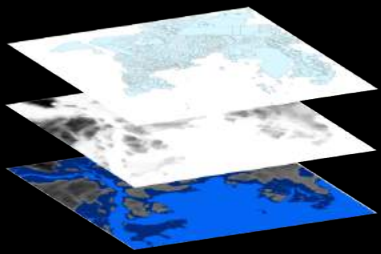
As storm surge intensify, longer walls are needed. The ability to predict identify potential water entry during the design of the wall is therefore crucial as the storm intensity are expected to increase in the future.

Given these challenges, 3 main questions are to be answered prior to a seawall construction:

- How many segments are there within the study area?
- What and how much are the capital in harms' way for each segment?
- Which of the segments are worth protecting economically and ecologically?

Which requires the availability of these information

- Elevation data of the region
- Spatial data of property values (natural, public, and private capital)
- Economic model of damages and cost of seawall construction



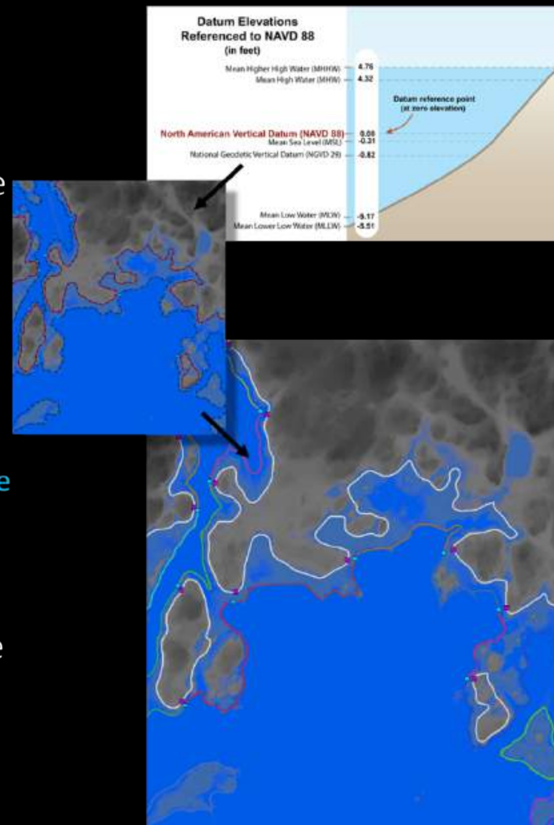
How many segments?

Segments location and length depend on the scenario (defined by the expert) in the design of the seawall.

Two elevation values need to be identified in order to determine the segments.

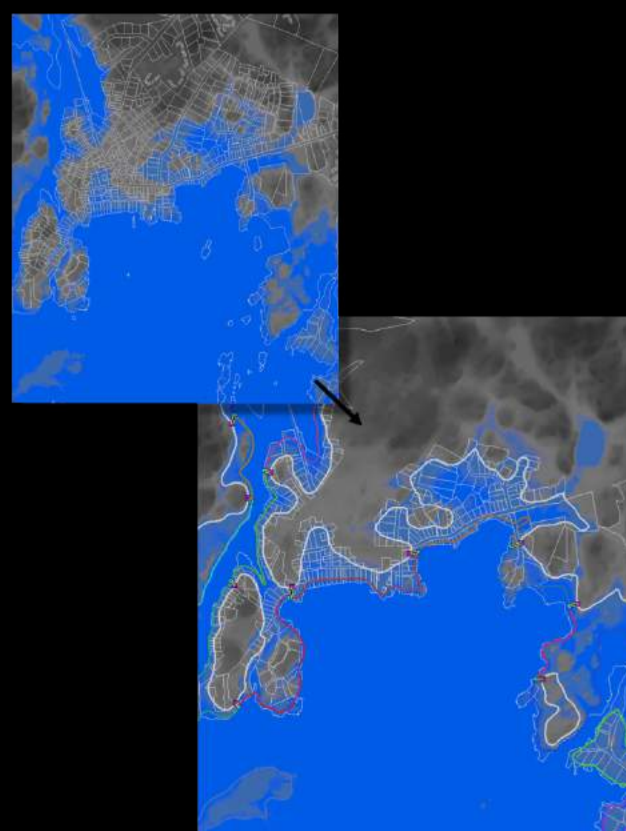
- **Mean High Water (which under US Law is where wall should be built)**
- **Local storm surge level potential within the timeframe**

Then for each of the two elevation values, contour lines will be created and seawall segments are where they are the closest to each other (i.e. where the elevation gradient is highest and sea water split in two directions)



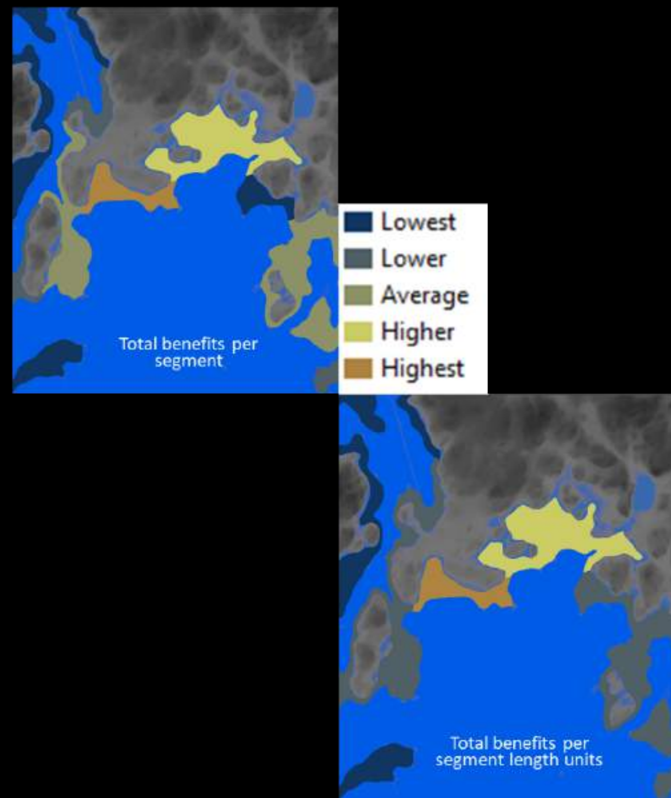
Capital in harms' way

- When the segments is delimited, they will be connected with their corresponding storm surge line ➔ which will produce a polygon, a region of interest for each segment.
- Anything within that region will be in harms' way.
- With the spatial data of the properties available, ArcGIS's Select by Location tool can highlights these at risk capitals.



Damages and costs

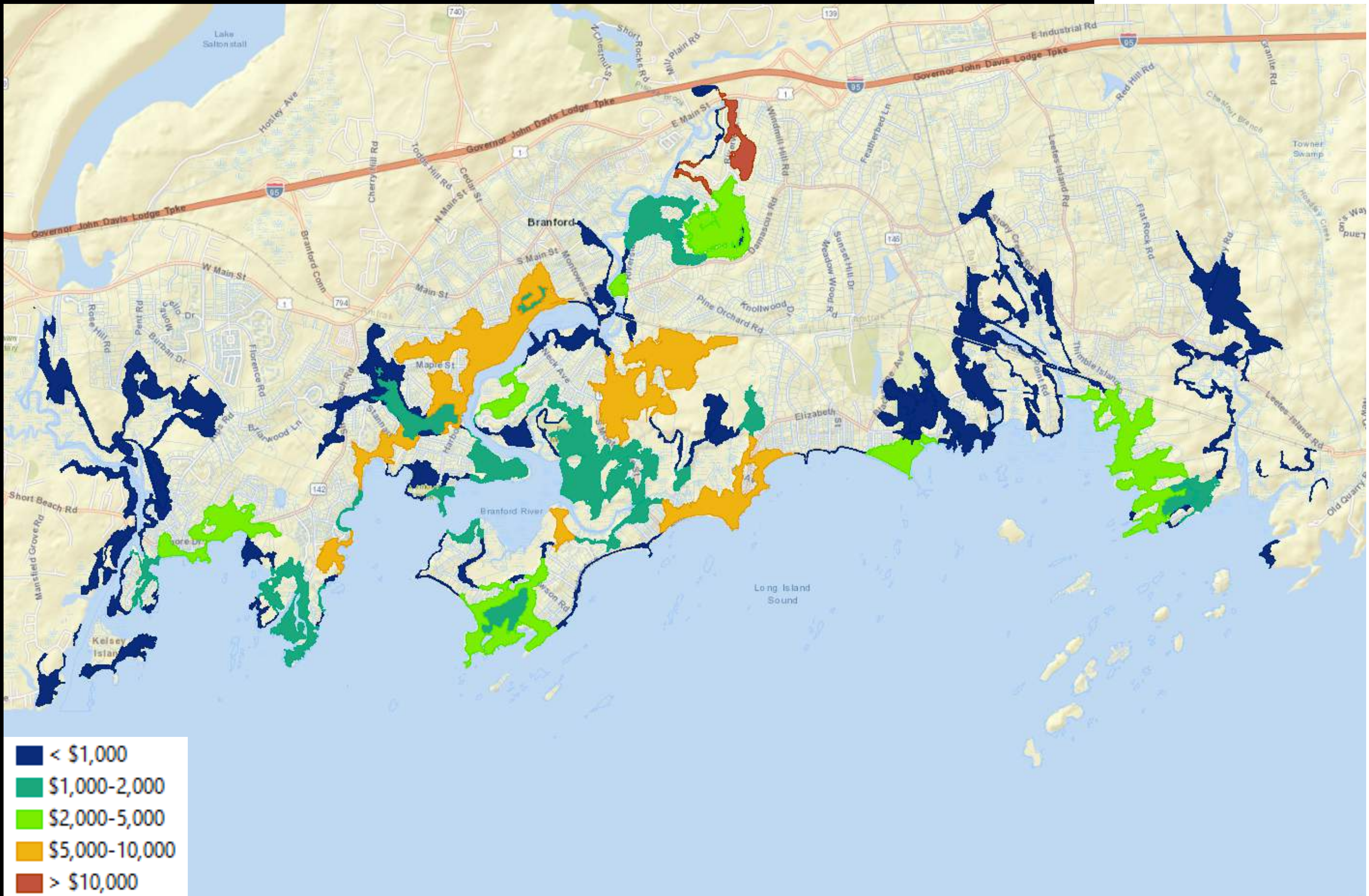
- For the selected properties, average elevation can be calculated using Zonal Statistics as Table, and then damages from storm can be calculated.
  - This tool will use a simple Cost-Benefit Analysis for the wall by calculating total benefits for each segment and the benefits per segment length units
- Wall considered feasible are the ones providing economic value.



The example results are from the Short Beach part of the town of Branford, which was my section in the coastal defense class. Obviously, economic analysis of the design of a seawall has to be more robust than this. However, in terms of the physical length and location of the segments, this tool is very helpful. As shown, above the orange polygons are the ones worth protecting, which was the results of my economic analysis for that section.

When using the tool, the user has to be careful with their spatial data they use and whether their data needs cleaning. Because during the tool does not correct for overlap in individual features during Zonal Statistics of the average elevation of the properties, which is a crucial information for the analysis.

Example output (for the town of Branford): Benefits/Damages per segment length (Feet)



```python
"""
THIS SCRIPT CREATES SEAWALL SEGMENTS WITHIN A CHOSEN COASTAL REGION AND ASSESS THEIR ECONOMIC EFFICIENCY

To create an ArcToolbox tool with which to execute this script, do the following.
1   In  ArcMap > Catalog > Toolboxes > My Toolboxes, either select an existing toolbox
    or right-click on My Toolboxes and use New > Toolbox to create (then rename) a new one.
2   Drag (or use ArcToolbox > Add Toolbox to add) this toolbox to ArcToolbox.
3   Right-click on the toolbox in ArcToolbox, and use Add > Script to open a dialog box.
4   In this Add Script dialog box, use Label to name the tool being created, and press Next.
5   In a new dialog box, browse to the .py file to be invoked by this tool, and press Next.
6   In the next dialog box, specify the following inputs (using dropdown menus wherever possible)
    before pressing OK or Finish.
        DISPLAY NAME                    DATA TYPE           PROPERTY>DIRECTION>VALUE      PROPERTY>DEFAULT>VALUE   PROPERTY>OBTAINED FROM>VALUE
        Raster elevation data           Raster Layer        Input
        Mean High Water                 Long                Input                         4
        Chosen surge level              Long                Input                         15
        Shapefile of the properties     Feature Layer       Input
        Field with building values      Field               Input                                                  Shapefile of the properties
        Unique ID field of buildings    Field               Input                                                  Shapefile of the properties
        Set Your Workspace              Workspace           Input
        Save the final output           Feature Class       Output

   To later revise any of this, right-click to the tool's name and select Properties.
"""


# -*- coding: utf-8 -*-
import sys, os, string, math, arcpy, traceback, time
from datetime import datetime


arcpy.env.overwriteOutput = True


#-----------------------------------------------------------------------------------------------------------------------#
# FUNCTIONS USED IN THE CODE
#-----------------------------------------------------------------------------------------------------------------------#
#-----------------------------------------------------------------------------------------------------------------------#
# FUNCTION TO CALCULATE DISTANCE BETWEEN POINTS
# https://community.esri.com/thread/158038
#-----------------------------------------------------------------------------------------------------------------------#
def calculateDistance(x1,y1,x2,y2):
     dist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
     return dist


#-----------------------------------------------------------------------------------------------------------------------#
# FUNCTION TO CALCULATE DAMAGES FROM STORM
# From FES Coastal Defense class (https://environment.yale.edu)
#-----------------------------------------------------------------------------------------------------------------------#
def stormDamage(value, dem, surge):
    value = float(value)
    dem = float(dem)
    surge = float(surge)
    flooded = surge-dem
    lower = dem-2
    upper = dem+7
    percent = max(min(flooded/(upper-lower),1),0)
    damage = value*percent
    return damage

#-----------------------------------------------------------------------------------------------------------------------#
# FUNCTION FOR SPATIAL JOIN
# http://pro.arcgis.com/en/pro-app/tool-reference/analysis/spatial-join.htm
# https://gis.stackexchange.com/questions/199754/arcpy-field-mapping-for-a-spatial-join-keep-only-specific-columns
#-----------------------------------------------------------------------------------------------------------------------#
def spatialJoin(target_feature, source_feature, in_field, out_field, stats, output):
    fieldmappings = arcpy.FieldMappings()
    fieldmappings.addTable(target_feature)
    fieldmappings.addTable(source_feature)

    # Remove unnecessary fields
    # We'll ultimately use length and ID so we keep it here
    keepers = [in_field, "Length", "Id"]
    for field in fieldmappings.fields:
        if field.name not in keepers:
             fieldmappings.removeFieldMap(fieldmappings.findFieldMapIndex(field.name))

    zonal_field_stats = fieldmappings.findFieldMapIndex(in_field)
    fieldmap = fieldmappings.getFieldMap(zonal_field_stats)
    field = fieldmap.outputField
    field.name = out_field
    field.aliasName = out_field
    fieldmap.outputField = field
    fieldmap.mergeRule = stats
```

```python
    fieldmappings.replaceFieldMap(zonal_field_stats, fieldmap)

    # Now joining. 10 Feet is my assumption of tolerance based on my method
    return arcpy.SpatialJoin_analysis(target_feature, source_feature, output,\
                                           "JOIN_ONE_TO_ONE", "KEEP_ALL", fieldmappings,\
                                           "INTERSECT", "10 Feet")


#------------------------------------------------------------------------------------------------------------------------#
# FUNCTION TO CREATE CONTOUR LINES FROM A SPECIFIC DEM VALUE
#------------------------------------------------------------------------------------------------------------------------#
def createContour(contourLines, demValue):
    # Start a timer
    time1 = time.clock()
    arcpy.AddMessage("\nCreating countour line at "+str(demValue)+" Feet. "+str(datetime.now()))

    # First let's make a layer of the contours
    arcpy.MakeFeatureLayer_management(contourLines, 'contourLines_lyr')

    # Select the corresponding contour lines
    contours_at_dem = arcpy.SelectLayerByAttribute_management(\
                         "contourLines_lyr", "ADD_TO_SELECTION", '"Contour" = {0}'.format(demValue))

    raw_contours = arcpy.CopyFeatures_management(contours_at_dem, "raw_contours_"+str(demValue)+".shp")
    smoothed0 = arcpy.cartography.SmoothLine(raw_contours, "smoothed0"+str(demValue)+".shp", "PAEK", "10 Feet")

    # If there are small lines in the selected, remove them
    # Users are not yet given control of this. Values given are based on my visual analysis of Branford case
    if int(demValue) < 5:
        th = 5000
    elif int(demValue) >= 5:
        th = 2000

    with arcpy.da.UpdateCursor(smoothed0, ["SHAPE@LENGTH"]) as lines:
        for line in lines:
            if line[0] < th:
                lines.deleteRow()

    del line, lines

    # Now smooth the lines to remove other noises and for better visualization
    smoothed = arcpy.cartography.SmoothLine(smoothed0, "smoothed"+str(demValue)+".shp", "PAEK", "10 Feet")

    # Dissolve the remaining polyline to fomr only one feature (Necessary for coastal segment delimitation)
    dissolved = arcpy.Dissolve_management(smoothed, 'contours'+str(demValue)+'.shp', ["FID"])

    # Now delete unnecessary files
    arcpy.Delete_management('contourLines_lyr')
    arcpy.Delete_management("raw_contours_"+str(demValue)+".shp")
    arcpy.Delete_management("smoothed"+str(demValue)+".shp")

    # Get the time (Stop the timer). And send success message.
    time2 = time.clock()
    arcpy.AddMessage("Contour line successfully created at "+str(demValue)+" Feet. It took "\
                     +str(time2-time1)+" seconds")

    return dissolved


#------------------------------------------------------------------------------------------------------------------------#
# FUNCTION TO DELIMITE THE SEGMENTS
#------------------------------------------------------------------------------------------------------------------------#
def createSegments(contour_at_mean_high_water, contour_at_surge):
    # Start a timer
    time1 = time.clock()
    arcpy.AddMessage("\nSegmentation of the coastline started at "+str(datetime.now()))

    # Specify a tolerance distance or minimum length of a seawall
    # Users are not yet given control of this
    th = 150

    # Create random points along the lines (mean high water and the surge of choice)
    # The numbers used are just my choice based on iterative observations
    random0 = arcpy.CreateRandomPoints_management(out_path= arcpy.env.workspace, \
                                                 out_name= "random0", \
                                                 constraining_feature_class= contour_at_mean_high_water, \
                                                 number_of_points_or_field= long(1600), \
                                                   minimum_allowed_distance = "{0} Feet".format(th))

    random1 = arcpy.CreateRandomPoints_management(out_path= arcpy.env.workspace, \
                                                     out_name= "random1", \
                                                     constraining_feature_class= contour_at_surge, \
```

```
                                                    number_of_points_or_field= long(1600), \
                                                  minimum_allowed_distance = "{0} Feet".format(th))

    # Perform a proximity analysis with the NEAR tool
    arcpy.Near_analysis(random0, random1)
    # Give each point a fixed unique ID
    # Create the ID field
    arcpy.AddField_management (random0, "UniqueID", "SHORT")
    arcpy.AddField_management (random1, "UniqueID", "SHORT")
    # Add Unique IDs
    arcpy.CalculateField_management(random0, "UniqueID", "[FID]")
    arcpy.CalculateField_management(random1, "UniqueID", "[FID]")

    # Categorize/Separate each feature based on their near feature
    # Crate a table view of random0
    table0 = arcpy.MakeTableView_management(random0, "random0_table")
    #table1 = arcpy.MakeTableView_management(random1, "random1_table")
    # Sort the near feature for each points in random0
    random0_sorted = arcpy.Sort_management(table0, "random0_sorte.dbf", [["NEAR_FID", "ASCENDING"]])


    # Create "long enough" lists for each of the field of interests: ID, NEAR_ID, and NEAR_DIST
    # (distance to closest point). I added [99999] here to extend the list length and avoid IndexError
    list_fid = [r.getValue("UniqueID") for r in arcpy.SearchCursor(random0_sorted, ["UniqueID"])] +[99999]
    list_nearid = [r.getValue("NEAR_FID") for r in arcpy.SearchCursor(random0_sorted, ["NEAR_FID"])]\
                  +[99999]
    list_neardist = [r.getValue("NEAR_DIST") for r in arcpy.SearchCursor(random0_sorted, ["NEAR_DIST"])]\
                    +[99999]

    del r

    # Only take points with near feature within the specified threshold. If it's too far, it's not better
    # than the others for a segment point
    list_fid_filtered = [i for i in list_neardist if i < th]
    # Then initiate a list o contain their Unique ID and Near ID
    first_unique_id = []
    first_near_id = []
    # Get NEAR_ID and Unique ID for each of these points
    for i in list_fid_filtered:
        first_unique_id.append(list_fid[list_neardist.index(i)])
        first_near_id.append(list_nearid[list_neardist.index(i)])

    # Only take the unique values in case there are duplicates. This shoudn't happen. Just to make sure.
    first_unique_id = [i for i in set(first_unique_id)]
    first_near_id = [i for i in set(first_near_id)]


    # Now create a new feature out of these points
    # Frist let's create a Feature Layer
    arcpy.MakeFeatureLayer_management("random0.shp", "random0_lyr")
    # Let's select all points and export them into a new feature
    random0_points = arcpy.SearchCursor(random0, ["UniqueID"])
    point0 = random0_points.next()

    for point0 in random0_points:
        for i in range(len(first_unique_id)):
            if point0.getValue("UniqueID") == first_unique_id[i]:
                selector0 = arcpy.SelectLayerByAttribute_management(\
                     "random0_lyr", "ADD_TO_SELECTION", '"UniqueID" = {0}'.format(first_unique_id[i]))

    del point0, random0_points

    new_random0 = arcpy.CopyFeatures_management(selector0, "new_random0")
    arcpy.Delete_management('random0_lyr')


    # Now for the new point feature, remove clusters of points around them and take only the ones
    # with minimum NEAR_DIST
    # First, get the geometry attributes of the new points
    arcpy.AddGeometryAttributes_management(new_random0, "POINT_X_Y_Z_M", "", "", "")

    # Create long enough list of the field of interest (same as the previous)
    pointx = [r.getValue("POINT_X") for r in arcpy.SearchCursor(new_random0, ["POINT_X"])] +[99999]
    pointy = [r.getValue("POINT_Y") for r in arcpy.SearchCursor(new_random0, ["POINT_Y"])] +[99999]
    new_list_fid = [r.getValue("UniqueID") for r in arcpy.SearchCursor(new_random0, ["UniqueID"])]\
                   +[99999]
    new_list_nearid = [r.getValue("NEAR_FID") for r in arcpy.SearchCursor(new_random0, ["NEAR_FID"])]\
                      +[99999]
    new_list_neardist = [r.getValue("NEAR_DIST") for r in arcpy.SearchCursor(new_random0, ["NEAR_DIST"])]\
                        +[99999]

    del r
```

```python

    # Initiate a list of every points that has already been compared to the near points
    garbage = []
    # Also initiate a list for the new Unique ID and NEAR ID
    new_unique_ID = []
    new_near_ID = []
    # Then, check if the points are right next to them. If so, add them to a temporary list
    # and find the one with closest near ID (or find minimum of their NEAR_DIST)
    for i in range(len(pointx)):
        if i+1 < len(pointx):

            # If not within the th range
            if not calculateDistance(pointx[i], pointy[i], pointx[i+1], pointy[i+1]) < float(th)*1.5:
                # Skip if it's in garbage
                if new_list_nearid[i] in garbage:
                    continue
                else:
                    new_unique_ID.append(new_list_fid[i])
                    new_near_ID.append(new_list_nearid[i])

            # If within the range
            else:
                # Skip if it's in garbage
                if new_list_nearid[i] in garbage:
                    continue
                else:
                    temp_ID = []
                    temp_NEAR = []
                    temp_DIST = []
                    while True:
                        temp_ID.append(new_list_fid[i])
                        temp_NEAR.append(new_list_nearid[i])
                        temp_DIST.append(new_list_neardist[i])
                        garbage.append(new_list_nearid[i])
                        i = i+1
                        # Stop when within the range again. And add the last point within the range
                        if not calculateDistance(pointx[i], pointy[i], pointx[i+1], pointy[i+1]) < 200:
                            temp_ID.append(new_list_fid[i])
                            temp_NEAR.append(new_list_nearid[i])
                            temp_DIST.append(new_list_neardist[i])
                            garbage.append(new_list_nearid[i])

                            # Calculate the minimum and get the Unique ID and Near ID
                            minD = min(temp_DIST)
                            new_unique_ID.append(new_list_fid[new_list_neardist.index(minD)])
                            new_near_ID.append(new_list_nearid[new_list_neardist.index(minD)])

                            del temp_ID, temp_NEAR, temp_DIST
                            break


    # Now select these final points export them into new feature.
    # These are the end points for the segments to be created
    # First, make a layer out of all the random points
    arcpy.MakeFeatureLayer_management("random0.shp", "random0_lyr")
    arcpy.MakeFeatureLayer_management("random1.shp", "random1_lyr")

    # Then select and export the end points into feature0 and feature1
    # Based on new_unique_ID for random0
    random0_points = arcpy.SearchCursor(random0, ["UniqueID"])
    point0 = random0_points.next()
    for point0 in random0_points:
        for i in range(len(new_unique_ID)):
            if point0.getValue("UniqueID") == new_unique_ID[i]:
                selected0 = arcpy.SelectLayerByAttribute_management(\
                     "random0_lyr", "ADD_TO_SELECTION", '"UniqueID" = {0}'.format(new_unique_ID[i]))

    feature0 = arcpy.CopyFeatures_management(selected0, "feature0")

    # Based on new_near_ID for random1
    random1_points = arcpy.SearchCursor(random1, ["UniqueID"])
    point1 = random1_points.next()
    for point1 in random1_points:
        for k in range(len(new_near_ID)):
            if point1.getValue("UniqueID") == new_near_ID[k]:
                selected1 = arcpy.SelectLayerByAttribute_management(\
                     "random1_lyr", "ADD_TO_SELECTION", '"UniqueID" = {0}'.format(new_near_ID[k]))

    feature1 = arcpy.CopyFeatures_management(selected1, "feature1")

    del point0, point1, random0_points, random1_points
    arcpy.Delete_management('random0_lyr')
```

```
arcpy.Delete_management('random1_lyr')


# Now for the actual create of the coastal segments
# Which include creation of polygon and splitting the contours as the corresponding points
# STEPS NECESSARY FOR POLYGON CREATION
# Let's first add geometry attributes to these points
arcpy.AddGeometryAttributes_management(feature0, "POINT_X_Y_Z_M", "", "", "")
arcpy.AddGeometryAttributes_management(feature1, "POINT_X_Y_Z_M", "", "", "")

# Let's create lines that connects points from feature0 to feature1
# Initiate a POLYLINE feature class for these lines
arcpy.CreateFeatureclass_management (arcpy.env.workspace, "connector_lines.shp", "POLYLINE")

# Then for each of the points in feature0, get the correspondingin feature1
# And create a line for each of the two points
with arcpy.da.SearchCursor(feature0, ["NEAR_FID", "POINT_X", "POINT_Y"]) as features0:
    for feat0 in features0:

        with arcpy.da.SearchCursor(feature1, ["UniqueID", "POINT_X", "POINT_Y"]) as features1:
            x=0
            for feat1 in features1:
                x = x+1
                theseTwoPoints = []

                if feat0[0] == feat1[0]:
                    # Get coordinates
                    X0, Y0 = feat0[1], feat0[2]
                    X1, Y1 = feat1[1], feat1[2]
                    # Append coordinates
                    theseTwoPoints.append(arcpy.PointGeometry(arcpy.Point(X0, Y0)))
                    theseTwoPoints.append(arcpy.PointGeometry(arcpy.Point(X1, Y1)))
                    # Create line from the coordinates
                    subline = arcpy.PointsToLine_management(theseTwoPoints, "subline"+str(x)+".shp")
                    # Append all lines into one feature
                    lines = arcpy.Append_management(["subline"+str(x)+".shp"], "connector_lines.shp")
                    # Then delete subline as it's now unnecessary
                    arcpy.Delete_management(subline)

                    continue


del feat0, feat1, features0, features1

# Now that the connectors are created, let's split the segments
# Before splitting contours into segments, let's integrate the points and the segments
# Just in case, there are misalignment
arcpy.Integrate_management([contour_at_mean_high_water, feature0])
arcpy.Integrate_management([contour_at_surge, feature1])
segments0 = arcpy.SplitLineAtPoint_management(contour_at_mean_high_water, feature0, "segments0.shp", "10 Feet")
segments1 = arcpy.SplitLineAtPoint_management(contour_at_surge, feature1, "segments1.shp", "10 Feet")
# And let's give fixed unique ID for each segment
arcpy.CalculateField_management(segments0, "Id", "[FID]")
arcpy.CalculateField_management(segments1, "Id", "[FID]")

# Now with the split segments and connector lines, let's make segment polygon of the segments
almost_segment_polygons = arcpy.FeatureToPolygon_management([segments0, segments1, lines],\
                                                            "almost_segment_polygons.shp")
# Adding unique ID to the segment polygons
arcpy.CalculateField_management(almost_segment_polygons, "Id", "[FID]")

# The Feature to Polygon process also created polygons that are surrounded by polygons
# These are because these areas are surrounded by flooded areas at surge.
# They are above the surge and technically safe. So, let's remove them.
arcpy.MakeFeatureLayer_management(almost_segment_polygons, 'almost_segment_polygons_lyr')
arcpy.MakeFeatureLayer_management(segments0, 'segments0_lyr')
# Only the polygons within the mean_high_water segments are at risk
arcpy.SelectLayerByLocation_management('almost_segment_polygons_lyr', 'INTERSECT', 'segments0_lyr')
final_without_length = arcpy.CopyFeatures_management('almost_segment_polygons_lyr', 'final.shp')

arcpy.Delete_management('segments0_lyr')
arcpy.Delete_management('almost_segment_polygons_lyr')

# For the new polygons, let's add the corresponding seawall length
# Let's add Length field to both first
arcpy.AddField_management(final_without_length, "Length", "SHORT")
arcpy.AddField_management(segments0, "Length", "SHORT")
# Calculation of the length
with arcpy.da.UpdateCursor(segments0, ["SHAPE@LENGTH", "Length"]) as segments_0:
     for segment_0 in segments_0:
          length = segment_0[0]
          segment_0[1] = length
```

```
              segments_0.updateRow(segment_0)
    del segment_0, segments_0

    # With spatial join, let's add these results to the segment polygons
    final = spatialJoin(final_without_length, segments0, "Length", "Length", "max", "joined_segment.shp")

    # Delete the created but now unnecessary files
    arcpy.Delete_management(random0)
    arcpy.Delete_management(random1)

    # Stop the timer
    time2 = time.clock()

    arcpy.AddMessage("Seawall segments and regions successfully created. It took "\
                     +str(time2-time1)+" seconds")

    return final



#----------------------------------------------------------------------------------------------------------------------#
# MAIN CODE
#----------------------------------------------------------------------------------------------------------------------#
# Check to see if Spatial Analyst license is available
if arcpy.CheckExtension("spatial") == "Available":

    try:

        # Activate Spatial Analyst
        arcpy.CheckOutExtension("spatial")

        # Necessary user inputs
        raster_dem = arcpy.GetParameterAsText(0)       # LIDAR DEM
        mean_high_water = arcpy.GetParameterAsText(1)  # In Feet recommended
        surge = arcpy.GetParameterAsText(2)            # In Feet recommended
        properties = arcpy.GetParameterAsText(3)       # Get geosptatial data of the properties
        building = arcpy.GetParameterAsText(4)         # Field of Building value in the properties shapefile
        zoneField = arcpy.GetParameterAsText(5)        # Field for unique ID of buildings the properties shapefile

        # Set Workspace for the results as defined by the user
        arcpy.env.workspace = arcpy.GetParameterAsText(6)

        # Save final Output
        output = arcpy.GetParameterAsText(7)

        # Let's first create a copy of the properties' feature we'll use
        properties_copy = arcpy.CopyFeatures_management(properties, "properties_copy")

        # Create layers for the properties and the raster
        raster_dem_lyr = arcpy.MakeRasterLayer_management(raster_dem, "raster_dem_lyr")
        properties_lyr = arcpy.MakeFeatureLayer_management(properties_copy, 'properties_lyr')

        # Create contours from the raster layer
        raster_to_contours = arcpy.sa.Contour(raster_dem_lyr, "raster_to_contours.shp", 1)

        # Create contour line for the user-specificed mean high water
        contour_mhw = createContour(raster_to_contours, mean_high_water)

        # Create contour line for the user-specificed storm surge level
        contour_surge = createContour(raster_to_contours, surge)

        # Create the coastal segments
        regions = createSegments(contour_mhw, contour_surge)

        # Create Layer from the segment polygons
        regions_lyr = arcpy.MakeFeatureLayer_management(regions, 'regions_lyr')


        # Calculating storm damage for each properties
        # Let's fits add a field
        arcpy.AddField_management(properties_copy, "S_Damage", "LONG")
        # Now let's get the mean elevation of each properties with zonal statistics
        # Do Zonal Statistics as Table
        zonal_stats = arcpy.sa.ZonalStatisticsAsTable(properties_lyr, zoneField, raster_dem_lyr,\
                                                      "zonal_stats", "NODATA", "MEAN")
        # Let's joing the result with the properties' feature
        arcpy.JoinField_management(properties_copy, zoneField, zonal_stats, zoneField)
        # Now the actual calculation, using UpdateCursor
        with arcpy.da.UpdateCursor(properties_copy, [building, "MEAN", "S_Damage"]) as segments:
             for segment in segments:
                  value = float(segment[0])
```

```
                  dem = segment[1]
                  segment[2] = stormDamage(value, dem, surge)
                  segments.updateRow(segment)

        del segment, segments

        # With spatial join, let's add these results to the segment polygons
        joined_r_p = spatialJoin(regions, properties_copy, "S_Damage", "T_Damage", "sum", "joined_r_p.shp")


        # Let's remove the surrounded polygons which are not at risk but automatically
        # created by spatial join
        arcpy.MakeFeatureLayer_management(contour_mhw, 'contour_mhw_lyr')
        arcpy.MakeFeatureLayer_management(joined_r_p, 'joined_r_p_lyr')
        # Only those intersecting segments at mean high water are at risk
        arcpy.SelectLayerByLocation_management('joined_r_p_lyr', 'INTERSECT', 'contour_mhw_lyr')

        # Save results
        before_output = arcpy.CopyFeatures_management('joined_r_p_lyr', 'before_output')

        # Now the calculation of the damage per segment length, using UpdateCursor
        # Add field for per segment damage
        arcpy.AddField_management(before_output, "PS_Damage", "FLOAT")

        with arcpy.da.UpdateCursor(before_output, ["Length", "T_Damage", "PS_Damage"]) as segments:
             for segment in segments:
                  length = float(segment[0])
                  damage = float(segment[1])
                  segment[2] = damage / length
                  segments.updateRow(segment)

        del segment, segments


        # Let's delete all now unnecessary layers
        arcpy.Delete_management('raster_dem_lyr')
        arcpy.Delete_management('properties_lyr')
        arcpy.Delete_management('regions_lyr')
        arcpy.Delete_management('contour_mhw_lyr')
        arcpy.Delete_management('joined_r_p_lyr')
        arcpy.Delete_management('raster_to_contours.shp')


        # Save results
        arcpy.CopyFeatures_management(before_output, output)


        # Adding the results in the dataframe
        mxd = arcpy.mapping.MapDocument("CURRENT")
        dataFrame = arcpy.mapping.ListDataFrames(mxd, "*")[0]
        addLayer0 = arcpy.mapping.Layer(output)
        arcpy.mapping.AddLayer(dataFrame, addLayer0)


    except Exception as e:
        arcpy.AddError('\n' + "Script failed because: \t\t" + e.message)
        exceptionreport = sys.exc_info()[2]
        fullermessage = traceback.format_tb(exceptionreport)[0]
        arcpy.AddError("at this location: \n\n" + fullermessage + "\n")


else:
    # Report error message if Spatial Analyst license is unavailable
    arcpy.AddMessage("Spatial Analyst license is unavailable")
```