

Appendix G

GIS PMP Tool Documentation

Table of Contents

Table of Contents	2
1. PMP Tools Description and Usage	3
1.1 File Structure	3
1.2 PMP Tool Usage	4
1.3 PMP Tool Output	6
1.4 Known Issues and Troubleshooting	8

1. PMP Tools Description and Usage

The PMP Evaluation Tool employed in this study is based on a Python script designed to run within the ArcGIS environment. ESRI's ArcGIS Desktop software is required to run the tool. The tool is compatible with ArcMap, ArcCatalog, or ArcGIS Pro. It is recommended that the most current version of the software is used. The PMP tool provides gridded output at a spatial resolution of 90 arc-seconds (equivalent to .025 x .025 decimal degrees) for a user-designated drainage basin or area at user-specified durations. Standard outputs include basin average PMP depths and temporally distributed accumulations.

1.1 File Structure

The PMP tool, source script, and the storm databases are stored within the 'PMP_Evaluation_Tool' project folder. The file and directory structure within the 'PMP_Evaluation_Tool' folder should be maintained as provided, as the script will locate various data based on its relative location within the project folder. If the subfolders or geodatabases within are relocated or renamed, then the script must be updated to account for these changes.

The file structure consists of three subfolders: Input, Output, and Script. The 'Input' folder contains all input GIS files (Figure 1.1). There are three ArcGIS file geodatabase containers within the 'Input' folder: DAD_Tables.gdb, Non_Storm_Data.gdb, and Storm_Adj_Factors.gdb. The DAD_Tables.gdb contains the DAD tables (in file geodatabase table format) for each of the SPAS-analyzed storm DAD zones included in the storm database. The Storm_Adj_Factors.gdb contains a feature class for each storm center and stores the adjustment factors for each grid point as a separate feature. These feature classes are organized into feature datasets, according to storm type (General, Local, and Tropical). The storm adjustment factor feature classes share their name with their DAD Table counterpart. The naming convention is SPAS_XXXX_Y, where XXXX is the SPAS storm ID number and Y is the DAD zone number. In the case of a hybrid storm (i.e., a storm that is run as both a general and local storm type), there will be a suffix "_gen" or "_loc" to differentiate the storm type specific to the adjustment factors in the feature class. The Non_Storm_Data.gdb contains spatial data not relating to the input rainfall depth or adjustment factors such as the grid network vector files.



Figure 1.1: PMP tool file structure

The 'Script' folder contains an ArcToolbox called NJ_Final_PMP_Tool.tbx. The toolbox contains a script tool called 'Gridded PMP Tool' that is used to calculate basin PMP for a basin or location.

ArcGIS should be used for viewing the GIS tools file structure and interacting with the input and output geospatial data. A typical operating system's file browser does not allow access to the geodatabase containers and cannot be used to directly run the tool.

The tools are stored within the NJ_Final_PMP_Tool.tbx. ArcToolbox opens and runs the scripts within the ArcGIS environment and can be run from ArcCatalog, ArcMap, or ArcPro map session. In addition to running as a standalone tool, the tool can be incorporated into Model Builder or be called as a sub-function of another script.

To run the tools, the user navigates to the NJ_Final_PMP_Tool.tbx toolbox, expands it, and opens the tool. The dialogue window opens, and the user populates input parameters and clicks the 'OK' button. The tool will run in the foreground and display text output in the Messages window. Processing time can vary greatly depending on area of interest (AOI) size, the number of durations selected, and computer hardware. Most basins generally take 10 to 20 minutes to analyze all three storm types. The tool produces PMP output described in Section 1.3.

1.2 PMP Tool Usage

The tool requires several parameters as input to define the area and durations to be analyzed. The first parameter required by the tool dialogue is a feature layer, such as a basin shapefile or feature class, designed to outline the AOI for the PMP analysis. If the AOI dataset does not have a surface projection, the tool will apply the Albers Equal Area projection for the purpose of calculating the AOI area size. If the feature layer has multiple features (or polygons), the tool will use the combined area as the analysis region. Only the selected polygons will be used if the tool is run from the ArcMap environment with selected features highlighted. If the AOI shapefile extends beyond the project analysis domain, PMP will only be calculated for grid cells inside the project domain. The AOI shapefile or feature class should not have any spaces or symbol characters in the filename.

The second parameter requires the path of the 'PMP_Evaluation_Tool' folder. The default location of the folder is set within the tool parameters, but it can be changed if the user wishes to link the tool to another set of input datasets. The 'PMP_Evaluation_Tool' project folder should be stored locally at a location that can be accessed (both read/write permissions) by ArcGIS. The user then will need to set the 'Output Folder' path which provides the tool with the location to create the output PMP files. The user must have read/write privileges for this folder location. Note, the tool will overwrite the previous output if all input parameters are the same. The user then selects the durations to be run for each storm type. Individual durations can be run by checking each individual box or all durations can be run by clicking the "Select All" option (Figure 1.2).

Gridded PMP Tool

Input basin outline shapefile or feature class
D:\GIS\NewJersey\Shapefiles\BeattiesMill.shp

Location of "PMP_Evaluation_Tool" Folder
D:\GIS\NewJersey\Tool_Testing\PMP_Evaluation_Tool

Output Folder
D:\GIS\NewJersey\Tool_Testing\PMP_Evaluation_Tool\Output

Local storm durations ***Basin area should be 100-sqmi or smaller for local storm PMP*** (optional)

- ☒ 01
- ☒ 02
- ☒ 03
- ☒ 04
- ☒ 05
- ☒ 06
- ☒ 12
- ☒ 24

Select All Unselect All Add Value

General storm durations (optional)

- ☒ 01
- ☒ 06
- ☒ 12
- ☒ 24
- ☒ 48
- ☒ 72

Select All Unselect All Add Value

Tropical storm durations (optional)

- ☒ 01
- ☒ 06
- ☒ 12
- ☒ 24
- ☒ 48
- ☒ 72

Select All Unselect All Add Value

Figure 1.2: PMP tool input/output parameters with all durations set to run for the Beatties Mill drainage basin in northern New Jersey

The next parameter allows the user to either use the basins calculated area size or override the default to enter a custom area (in square miles) for areal-average PMP calculations. The user then has the option to have the tool perform a weighted analysis on the grid cells underlying the AOI boundary. If this option is checked each grid cell along the basin's boundary will be weighted by the portion of the cell's area inside the basin for the purpose of the basin average PMP table calculations. It is checked by default. If this option is disabled, the tool will output a basin average of all grid cells equally that intersect the basin boundary. There is an option to include sub-basin averages. This will calculate an average PMP depth for each feature in the input basin feature class from the overall basin PMP. The average sub-basin depths will be based on the area-size of the overall basin. If the 'weighted' option was selected above it will also be applied to the sub-basin averages. The user must select a field within the AOI to be used to identify each sub-basin. The field can be of numeric or text data type but must have a unique ID for each polygon. This option is disabled by default. The user can also choose to include a depth-duration chart .png image in the output folder for each storm type. Finally, the user can select the option to apply the appropriate temporal distribution patterns to the basin average PMP for each storm type. This function needs all durations of PMP to be calculated, so if this option is selected the tool will automatically run all durations for all storm types regardless of what durations were selected by the user in the previous steps (Figure 1.3).

The screenshot shows a software dialog box for the PMP tool. It features a light gray background and a standard Windows-style title bar. At the top, there are three buttons: 'Select All', 'Unselect All', and 'Add Value'. Below these are several options, each with a checkbox. The first checkbox, 'Use basin area size for areal average', is checked. Below it is a text input field labeled 'Area-size to use (sqmi) (optional)'. The second checkbox, 'Apply weighted average to border grid cells', is also checked. The third checkbox, 'Include sub-basin averages (optional)', is unchecked. Below this is a dropdown menu labeled 'Sub-basin field (optional)'. The fourth checkbox, 'Include depth-duration chart output', is checked. The fifth checkbox, 'Apply temporal distributions', is unchecked. At the bottom of the dialog are four buttons: 'OK', 'Cancel', 'Environments...', and '<< Hide Help'.

Figure 1.3: PMP tool input/output parameters with default options enabled.

1.3 PMP Tool Output

Once the tool has been run, the output file geodatabases will be populated with the model results. The GIS files can then be brought into an ArcMap, or other compatible GIS environments, for mapping and analysis. The tool is set to have overwrite capabilities; if output data exists, it will be overwritten the next time the tool is run, if the same output folder and same parameters are used.

A separate output folder is created for each storm type and the output is organized within file geodatabases and named according to the input basin feature name and analyzed PMP area. Each

output file geodatabase contains a feature class which stores each grid point centroid within the basin as a separate feature. Each feature has a field for the grid ID, latitude, longitude, analysis zone, elevation, PMP (for each duration), and the contributing storm ID. The PMP raster files are also stored within the file geodatabase. The naming convention for the raster files is the storm type and duration (L for Local, G for General, and T for Tropical), followed by the input basin feature name, and ending with the basin area (in square miles). If temporal patterns were applied, the output tables will also be in the geodatabase. A folder named CSV is also created and all the geodatabase tables are exported to csv files. An example of the output file structure is shown in Figure 1.4.

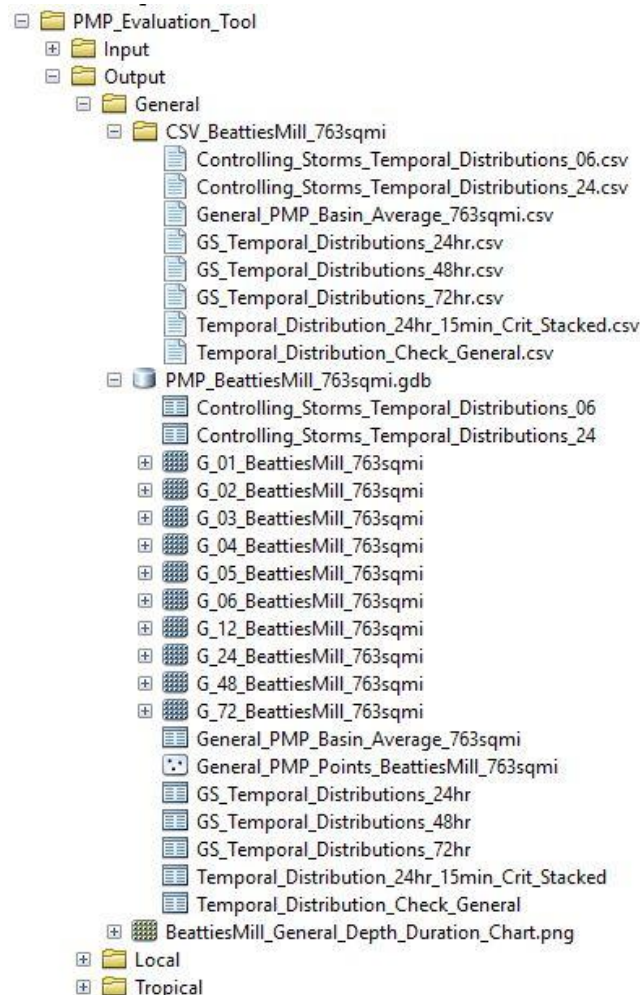


Figure 1.4: Example of the PMP tool output file structure

If the temporal patterns were applied, you will see a series of tables with the word “Check” as a prefix. These are important as it evaluates the temporally distributed PMP values for each duration against the PMP value for that duration. The table has a pass or fail. If the temporally distributed PMP value exceeds the PMP at a given duration, the table will have fail for that duration and this temporal pattern should not be applied. An example of this is shown in figure 1.5.

Table

Check Controlling Storms Temporal Distributions_06

	PATTERN	PMP_01	MAX_01	CHECK_01	PMP_02	MAX_02	CHECK_02	PMP_03	MAX_03	CHECK_03	PMP_04	MAX_04	CHECK_04
	SPAS_1336_1	2.74	2.66	PASS	4.4	5.21	FAIL	7.6	7.67	FAIL	7.6	8.23	FAIL

<

<

Figure 1.5: Example of the temporal check results

In the example above (Figure 1.5), the basin average 1-hour PMP is 2.74. Using the temporal distribution for one of the controlling storms, the maximum 1-hour value is 2.66. This passes the check. However, for the 2-hour PMP the maximum temporally distributed value of 5.21 is exceeding the 4.4 PMP values. This fails the check, and this pattern should not be applied to the PMP values.

1.4 Known Issues and Troubleshooting

The GIS PMP tool has undergone a beta testing program during development. One goal of the beta testing program was to identify possible issues with the GIS tool. The following guidelines may prevent issues with running the GIS tool.

- Ensure ArcGIS Desktop is up-to-date with the most recent version release and maintenance is current.
- Ensure all file and path names do not have spaces or non-alphanumeric symbols (e.g. #, \$, %). Underscores are acceptable and a good alternative to using spaces.
- Close any other applications or instances of ArcMap that may interfere with the current session, files, or file paths that will be used by the tool.
- Ensure that all file paths, input and output files, and ArcGIS Environment settings (including the Default.gdb and Scratch.gdb) are local and not set to a network location.

If the points above have been verified and issues persist, the user may try the following actions to address the issue:

- Close out all ArcMap sessions and all ArcGIS applications and restart session.
- Restart computer. This may be required to completely clear any locks on files or memory.
- Run the Repair Geometry tool on the AOI shapefile or feature class to correct any geometry issues within the file.
- Rename AOI file. Change tool and/or output folder paths.
- If issues persist it may be necessary to contact ESRI support or perform a clean ArcGIS installation or upgrade.

Appendix H

GIS Tool Python Script

Name: Gridded PMP Tool Python Script

Script Version: 2

Python Version: 2.7

ArcGIS Version: ArcGIS Desktop 10.7.1

Author: Applied Weather Associates

Usage: The tool is designed to be executed within an the ArcMap environment with an open MXD session.

Description:

This tool calculates PMP depths for a given drainage basin for the specified durations. PMP point values are calculated (in inches) for each grid point (spaced at 90 arc-second intervals) over the project domain. The points are converted to gridded PMP datasets for each duration.

```
-----'''
#####
## import Python modules

import sys
import arcpy
import os
import traceback
from arcpy import env
import arcpy.analysis as an
import arcpy.management as dm
import arcpy.conversion as con
import numpy as np
import pandas as pd
from pandas import ExcelFile
import matplotlib.pyplot as plt
from heapq import nlargest

env.overwriteOutput = True                # Set overwrite option
env.addOutputsToMap = False

#####
## get input parameters
```

```

basin = arcpy.GetParameter(0)                                # get AOI Basin Shapefile
home = arcpy.GetParameterAsText(1)                          # get location of 'PMP'
Project Folder
outLocation = arcpy.GetParameterAsText(2)
if arcpy.GetParameter(12) == False:
    locDurations = arcpy.GetParameter(3)                    # get local storm durations
    (string)
    genDurations = arcpy.GetParameter(4)                    # get general storm
    durations (string)
    tropDurations = arcpy.GetParameter(5)                    # get tropical storm
    durations (string)
else:
    locDurations = ('01','02','03','04','05','06','12','24')
    genDurations = ('01','02','03','04','05','06','12','24','48','72')
    tropDurations = ('01','02','03','04','05','06','12','24','48','72')

weightedAve = arcpy.GetParameter(8)                          # get option to
apply weighted average (boolean)
#outputTable = arcpy.GetParameter(9)                         # get file path
for basin average summary table
includeSubbasin = arcpy.GetParameter(9)                      # get option add
subbasin averages (boolean)
subbasinIDfield = arcpy.GetParameterAsText(10)               # Subbasin ID
field from AOI Basin Shapefile
ddChart = arcpy.GetParameter(11)                             # get option to create
depth-duration chart(boolean)
runTemporal = arcpy.GetParameter(12)                         # get option to
run temporal distributions (boolean)

dadGDB = home + "\\Input\\DAD_Tables.gdb"                    # location of DAD tables
adjFactGDB = home + "\\Input\\Storm_Adj_Factors.gdb"         # location of feature
datasets containing total adjustment factors
arcpy.AddMessage("\nDAD Tables geodatabase path: " + dadGDB)
arcpy.AddMessage("Storm Adjustment Factor geodatabase path: " + adjFactGDB)

#mxd = arcpy.mapping.MapDocument("CURRENT")
#df = arcpy.mapping.ListDataFrames(mxd)[0]
basAveTables = []                                             # global list of Basin Average
Summary tables

def pmpAnalysis(aoiBasin, stormType, durList):

#####
## Create PMP Point Feature Class from points within AOI basin and add fields
def createPMPfc():

```

```

        arcpy.AddMessage("\nCreating feature class: 'PMP_Points' in Scratch.gdb...")
        dm.MakeFeatureLayer(home + "\\Input\\Non_Storm_Data.gdb\\Vector_Grid", "vgLayer")
# make a feature layer of vector grid cells
        dm.SelectLayerByLocation("vgLayer", "INTERSECT", aoiBasin) #
select the vector grid cells that intersect the aoiBasin polygon
        dm.MakeFeatureLayer(home + "\\Input\\Non_Storm_Data.gdb\\Grid_Points", "gpLayer")
# make a feature layer of grid points
        dm.SelectLayerByLocation("gpLayer", "HAVE_THEIR_CENTER_IN", "vgLayer")
# select the grid points within the vector grid selection
        con.FeatureClassToFeatureClass("gpLayer", env.scratchGDB, "PMP_Points")
# save feature layer as "PMP_Points" feature class
        arcpy.AddMessage("(" + str(dm.GetCount("gpLayer")) + " grid points will be analyzed)\n")

# Add PMP Fields
for dur in durList:
    arcpy.AddMessage("\t...adding field: PMP_" + str(dur))
    dm.AddField(env.scratchGDB + "\\PMP_Points", "PMP_" + dur, "DOUBLE")

# Add STORM Fields (this string values identifies the driving storm by SPAS ID number)
for dur in durList:
    arcpy.AddMessage("\t...adding field: STORM_" + str(dur))
    dm.AddField(env.scratchGDB + "\\PMP_Points", "STORM_" + dur, "TEXT", "", "", 16,
"Storm ID " + dur + "-hour")

# Add STNAME Fields (this string values identifies the driving storm by SPAS ID number)
#     for dur in durList:
#         arcpy.AddMessage("\t...adding field: STNAME_" + str(dur))
#         dm.AddField(env.scratchGDB + "\\PMP_Points", "STNAME_" + dur, "TEXT", "", "",
50, "Storm Name " + dur + "-hour")

return

#####
## Define getAOIarea() function:
## getAOIarea() calculates the area of AOI (basin outline) input shapefile/
## featureclass. The basin outline shapefile must be projected. The area
## is square miles, converted from the basin layers projected units (feet
## or meters). The aoiBasin feature class should only have a single feature
## (the basin outline). If there are multiple features, the area will be stored
## for the final feature only.

def getAOIarea():
    sr = arcpy.Describe(aoiBasin).SpatialReference # Determine
aoiBasin spatial reference system
    srname = sr.name
    srtype = sr.type

```

```

srunitname = sr.linearUnitName                                # Units
arcpy.AddMessage("\nAOI basin spatial reference: " + srname + "\nUnit type: " +
srunitname + "\nSpatial reference type: " + srtype)

aoiArea = 0.0
rows = arcpy.SearchCursor(aoiBasin)
for row in rows:
    feat = row.getValue("Shape")
    aoiArea += feat.area
    if srtype == 'Geographic':                                # Must have a surface projection. If one
doesn't exist it projects a temporary file and uses that.
        arcpy.AddMessage("\n***The basin shapefile's spatial reference 'Geographic' is not
supported. Projecting temporary shapefile for AOI.***")
        arcpy.Project_management(aoiBasin,env.scratchGDB + "\\TempBasin",102039)    #
Projects AOI Basin (102039 = USA_Contiguous_Albers_Equal_Area_Conic_USGS_version)
        TempBasin = env.scratchGDB + "\\TempBasin"                                # Path to
temporary basin created in scratch geodatabase
        sr = arcpy.Describe(TempBasin).SpatialReference                                # Determine
Spatial Reference of temporary basin
        aoiArea = 0.0
        rows = arcpy.SearchCursor(TempBasin)                                #
Assign area size in square meters
        for row in rows:
            feat = row.getValue("Shape")
            aoiArea += feat.area
            aoiArea = aoiArea * 0.000000386102                                # Converts square
meters to square miles
        elif srtype == 'Projected':
            if srunitname == "Meter":
                aoiArea = aoiArea * 0.000000386102                                # Converts square
meters to square miles
            elif srunitname == "Foot" or "Foot_US":
                aoiArea = aoiArea * 0.00000003587                                # Converts square
feet to square miles
            else:
                arcpy.AddMessage("\nThe basin shapefile's unit type '" + srunitname + "' is not
supported.")
                sys.exit("Invalid linear units")                                # Units must be meters or
feet

aoiArea = round(aoiArea, 3)
arcpy.AddMessage("\nArea of interest: " + str(aoiArea) + " square miles.")

if arcpy.GetParameter(6) == False:
    aoiArea = arcpy.GetParameter(7)                                # Enable a constant
area size

```

```

aoiArea = round(aoiArea, 1)
arcpy.AddMessage("\n***Area used for PMP analysis: " + str(aoiArea) + " sqmi***")
return aoiArea

#####
## Define dadLookup() function:
## The dadLookup() function determines the DAD value for the current storm
## and duration according to the basin area size. The DAD depth is interpolated
## linearly between the two nearest areal values within the DAD table.
def dadLookup(stormLayer, duration, area):          # dadLookup() accepts the current
storm layer name (string), the current duration (string), and AOI area size (float)
    #arcpy.AddMessage("\t\tfunction dadLookup() called.")
    durField = "H_" + duration                    # defines the name of the duration field (eg.,
"H_06" for 6-hour)
    dadTable = dadGDB + "\\ " + stormLayer
    rows = arcpy.SearchCursor(dadTable)

    try:
        row = rows.next()                        # Sets DAD area x1 to the value in the first
row of the DAD table.
        x1 = row.AREASQMI
        y1 = row.getValue(durField)
        xFlag = "FALSE"                          # xFlag will remain false for basins that are
larger than the largest DAD area.
        except RuntimeError:                      # return if duration does not exist in DAD
table
            return

    row = rows.next()
    i = 0
    while row:                                    # iterates through the DAD table - assigning the
bounding values directly above and below the basin area size
        i += 1
        if row.AREASQMI < area:
            x1 = row.AREASQMI
            y1 = row.getValue(durField)
        else:
            xFlag = "TRUE"                        # xFlag is switched to "TRUE" indicating area
is within DAD range
            x2 = row.AREASQMI
            y2 = row.getValue(durField)
            break

    row = rows.next()
    del row, rows, i

```

```

    if xFlag == "FALSE":
        x2 = area                                # If x2 is equal to the basin area, this means that the
largest DAD area is smaller than the basin and the resulting DAD value must be extrapolated.
        arcpy.AddMessage("\t\tThe basin area size: " + str(area) + " sqmi is greater than the
largest DAD area: " + str(x1) + " sqmi.\n\t\tDAD value is estimated by extrapolation.")
        y = x1 / x2 * y1                        # y (the DAD depth) is estimated by extrapolating
the DAD area to the basin area size.
        return y                                # The extrapolated DAD depth (in inches) is returned.

    # arcpy.AddMessage("\nArea = " + str(area) + "\nx1 = " + str(x1) + "\nx2 = " + str(x2) +
"\ny1 = " + str(y1) + "\ny2 = " + str(y2))

    x = area                                    # If the basin area size is within the DAD table area
range, the DAD depth is interpolated
    deltax = x2 - x1                            # to determine the DAD value (y) at area (x) based
on next lower (x1) and next higher (x2) areas.
    deltay = y2 - y1
    diffx = x - x1

    y = y1 + diffx * deltay / deltax

    if x < x1:
        arcpy.AddMessage("\t\tThe basin area size: " + str(area) + " sqmi is less than the smallest
DAD table area: " + str(x1) + " sqmi.\n\t\tDAD value is estimated by extrapolation.")

    return y                                    # The interpolated DAD depth (in inches) is returned.

#####
## Define updatePMP() function:
## This function updates the 'PMP_XX_' and 'STORM_XX' fields of the PMP_Points
## feature class with the largest value from all analyzed storms stored in the
## pmpValues list.
def updatePMP(pmpValues, stormID, duration):    # Accepts
four arguments: pmpValues - largest adjusted rainfall for current duration (float list); stormID -
driver storm ID for each PMP value (text list); and duration (string)
    pmpfield = "PMP_" + duration
    stormfield = "STORM_" + duration
    stormTextField = "STNAME_" + duration

    gridRows = env.scratchGDB + "\\PMP_Points"    # iterates
through PMP_Points rows
    i = 0
    with arcpy.da.UpdateCursor(gridRows, (pmpfield, stormfield)) as cursor:
        for row in cursor:
            row[0] = pmpValues[i]                # Sets the PMP field
value equal to the Max Adj. Rainfall value (if larger than existing value).

```

```

        row[1] = stormID[i]                                # Sets the storm ID field
to indicate the driving storm event
        cursor.updateRow(row)
        i += 1
    del row, gridRows, pmpfield, stormfield, i
    arcpy.AddMessage("\n\t" + duration + "-hour PMP values update complete. \n")
    return

#####
## The outputPMP() function produces raster GRID files for each of the PMP durations.
## Aslo, a space-delimited PMP_Distribution.txt file is created in the 'Text_Output' folder.
def outputPMP(type, area, outPath):
    desc = arcpy.Describe(basin)
    basinName = desc.baseName
    pmpPoints = env.scratchGDB + "\\PMP_Points"            # Location of
'PMP_Points' feature class which will provide data for output

    outType = type[:1]
    outArea = str(int(round(area,0))) + "sqmi"
    outGDB = "PMP_" + basinName + "_" + outArea + ".gdb"
    if not arcpy.Exists(outPath + "\\" + outGDB):          # Check to see if
PMP_XXXXX.gdb already exists
        arcpy.AddMessage("\nCreating output geodatabase '" + outGDB + "'")
        dm.CreateFileGDB(outPath, outGDB)
        arcpy.AddMessage("\nCopying PMP_Points feature class to " + outGDB + "...")
        con.FeatureClassToFeatureClass(pmpPoints, outPath + "\\" + outGDB, type +
"_PMP_Points_" + basinName + "_" + outArea)
        pointFC = outPath + "\\" + outGDB + "\\" + type + "_PMP_Points_" + basinName + "_" +
outArea
        # addLayerMXD(pointFC) # calls addLayerMDX function to add output to ArcMap
session

    arcpy.AddMessage("\nBeginning PMP Raster Creation...")

    for dur in durList:                                    # This code creates a raster GRID from
the current PMP point layer
        durField = "PMP_" + dur
        outLoc = outPath + outGDB + "\\" + outType + "_" + dur + "_" + basinName + "_" +
outArea
        arcpy.AddMessage("\n\tInput Path: " + pmpPoints)
        arcpy.AddMessage("\tOutput raster path: " + outLoc)
        arcpy.AddMessage("\tField name: " + durField)
        con.FeatureToRaster(pmpPoints, durField, outLoc, "0.025")
        arcpy.AddMessage("\tOutput raster created...")
    del durField, outLoc, dur

```

```

arcpy.AddMessage("\nPMP Raster Creation complete.")

if includeSubbasin:                                # Begin subbasin average calculations
    subbasinID = []
    with arcpy.da.SearchCursor(basin, subbasinIDfield) as cursor: # Create list of subbasin
ID names
        for row in cursor:
            subbasinID.append(row[0])

    subIDtype = arcpy.ListFields(basin, subbasinIDfield)[0].type # Define the datatype of
the subbasin ID field

    if subIDtype != "String":                        # Convert subbasin IDs to a string, if
they are not already
        subbasinID = [str(i) for i in subbasinID]

    subNameLen = max(map(len, subbasinID))           # Define the length of the
longest subbasin ID

    # arcpy.AddMessage("\nList of subbasins...\n" + "\n".join(subbasinID))

    arcpy.AddMessage("\nCreating Subbasin Summary Table...")
    tableName = type + "_PMP_Subbasin_Average" + "_" + outArea
    tablePath = outPath + "\\ " + outGDB + "\\ " + tableName
    dm.CreateTable(outPath + "\\ " + outGDB, tableName) # Create blank table

    dm.AddField(tablePath, "STORM_TYPE", "TEXT", "", "", 10, "Storm Type") #
Create "Storm Type" field
    dm.AddField(tablePath, "SUBBASIN", "TEXT", "", "", subNameLen, "Subbasin") #
Create "Subbasin" field

    cursor = arcpy.da.InsertCursor(tablePath, "SUBBASIN") # Create Insert cursor and
add a blank row to the table for each subbasin
    for sub in subbasinID:
        cursor.insertRow([sub])
    del cursor, sub

    dm.CalculateField(tablePath, "STORM_TYPE", "" + type + "", "PYTHON_9.3") #
populate storm type field

    i = 0
    for field in arcpy.ListFields(pmpPoints, "PMP_*"): # Add fields for each PMP
duration and calculate the subbasin averages
        fieldName = field.name
        arcpy.AddMessage("\n\tCalculating subbasin average for " + fieldName + "
(weighted)...\n")

```

```

        dm.AddField(tablePath, fieldName, "DOUBLE", "", 2)      # Add duration field
        subAveList = []
        for subbasin in subbasinID:                             # Loop through each subbasin
            if subIDtype != "String":                             # Define an SQL expression that
specifies the current subbasin
                sql_exp = """"{0} = {1}"""".format(arcpy.AddFieldDelimiters(basin,
subbasinIDfield), subbasin)
            else:
                sql_exp = """"{0} = '{1}'"""".format(arcpy.AddFieldDelimiters(basin,
subbasinIDfield), subbasin)
            dm.MakeFeatureLayer(basin, "subbasinLayer", sql_exp)
            outLayer = outputPath + "\\\" + outGDB + "\\subbasin_" + str(subbasin)
            subBasAve = basinAve("subbasinLayer", fieldName)      # Call the basinAve()
function passing the subbasin and duration field
            arcpy.AddMessage("\tSubbasin average for " + str(subbasin) + ": " +
str(subBasAve) + "")
            subAveList.append(subBasAve)                          # Add subbasin average to list
        p = 0
        with arcpy.da.UpdateCursor(tablePath, fieldName) as cursor: # Update the subbasin
average summary table with the subbasin averages
            for row in cursor:
                row = subAveList[p]
                cursor.updateRow([row])
                p += 1

    ##            dm.CalculateField(tablePath, fieldName, fieldAve, "PYTHON_9.3")    # Assigns
the basin average
    ##            dur = durList[i]                                                  # following lines add alias field
names to basin average table (ArcGIS 10.2.1 or later)
    ##            if dur[0] == "0":
    ##                dur = dur[1:]
    ##            fieldAlias = dur + "-hour PMP"
    ##            dm.AlterField(tablePath, fieldName, "#", fieldAlias)
    i += 1
    arcpy.AddMessage("\nSubbasin summary table complete.")

    arcpy.AddMessage("\nCreating Basin Summary Table...")
    tableName = type + "_PMP_Basin_Average" + "_" + outArea
    tablePath = outputPath + "\\\" + outGDB + "\\\" + tableName
    dm.CreateTable(outPath + "\\\" + outGDB, tableName)      # Create blank table
    cursor = arcpy.da.InsertCursor(tablePath, "*")           # Create Insert cursor and add a
blank row to the table
    cursor.insertRow([0])
    del cursor

```

```

        dm.AddField(tablePath, "STORM_TYPE", "TEXT", "", "", 30, "Storm Type")      #
Create "Storm Type" field
        dm.CalculateField(tablePath, "STORM_TYPE", "" + type + "", "PYTHON_9.3")    #
populate storm type field

        i = 0
        for field in arcpy.ListFields(pmpPoints, "PMP_*"):      # Add fields for each PMP
duration and calculate the basin average
            fieldName = field.name
            fieldAve = basinAve(basin, fieldName)                # Calls the basinAve() function -
returns the average (weighted or not)
            dm.AddField(tablePath, fieldName, "DOUBLE", "", 2)    # Add duration field
            dm.CalculateField(tablePath, fieldName, fieldAve, "PYTHON_9.3")    # Assigns the
basin average
##            dur = durList[i]                                # following lines add alias field names to basin
average table (ArcGIS 10.2.1 or later)
##            if dur[0] == "0":
##                dur = dur[1:]
##            fieldAlias = dur + "-hour PMP"
##            dm.AlterField(tablePath, fieldName, "#", fieldAlias)
            i += 1
        arcpy.AddMessage("\nSummary table complete.")
        basAveTables.append(tablePath)

##            The following lines export a .png image depth duration chart and PMP summary
excel file to the output folder
        if ddChart:
            xValues = durList                                #Get list of durations for chart
            xValues = [int(i) for i in xValues]                #Convert duration list to integers
            ax1 = plt.subplot2grid((1,1), (0,0))    #Create variable for subplot in chart
            yValues = []
            pmpFields = [field.name for field in arcpy.ListFields(tablePath, "PMP_*")] # Selects
PMP fields for yValues
            with arcpy.da.SearchCursor(tablePath, pmpFields) as cursor:      # Adds PMP
depths to yValues
                yValues = next(cursor)
            del cursor, pmpFields

            stormFields = [field.name for field in arcpy.ListFields(pmpPoints, "Storm_*")] # Selects
Controlling Storm fields
            contStorms = []                                # List of controlling storms for a single duration
            listOfContStorms = []                            # List of controlling storms for all durations (list of
lists)
            i = 0                                            # iterator (for "Storm_*" fields)
            while i < len(stormFields):                        # iterates through controlling storm fields

```

```

        with arcpy.da.SearchCursor(pmpPoints, stormFields) as cursor: # Search cursor
returns list of unique controlling storms
            contStorms = sorted({row[i] for row in cursor})
            listOfContStorms.append(contStorms) # Add unique storms for current
duration to list of controlling stomrs
            i += 1
        del cursor

        plt.plot(xValues,yValues) #Creates chart
        plt.xlabel('Storm Duration in Hours')
        plt.ylabel('Rainfall Depth in Inches')
        plt.title(basinName + " (" + outArea + ") " + type + ' Storm Basin Average PMP\nDepth
Duration Chart')
        ax1.grid(True) #Creates grid lines in chart
        yTop = max(yValues) + 1
        ax1.set_ylim(top = yTop) #Sets y axis values to match depths +1 1
        ax1.set_xticks(xValues) #Sets x axis values to match durations
##         i = 0
##         xy = zip(xValues, yValues)
##         while i < len(stormFields): # iterates through controlling storm fields
##             pointXY = xy[i]
##             yLabel = '{0:.1f}'.format(yValues[i]) # round PMP depth to 1 decimal and
convert to string
##             stormLabel = str(listOfContStorms[i]) # convert controlling storm ID(s) to string
##             stormLabel = stormLabel.replace("u", "") # remove unicode "u"
##             stormLabel = stormLabel.replace("'", "") # remove unicode "'"
##             stormLabel = stormLabel.replace("[", "") # remove unicode "["
##             stormLabel = stormLabel.replace("]", "") # remove unicode "]"
##             #ax1.annotate(yLabel + "\n" + stormLabel, xy=xy[i], textcoords='offset points',
size=8, annotation_clip=True)
##             ax1.annotate(yLabel + "\n" + stormLabel, xy=xy[i], textcoords='data', size=8,
annotation_clip=True)
##             i += 1
##         del xy

        plt.savefig(outPath + "\\ " + basinName + "_" + type + "_Depth_Duration_Chart.png")
        #Save image
        plt.close() #Close chart to remove from memory
        arcpy.AddMessage("\nDepth Duration Chart exported to output folder.")
        del xValues, yValues, #df, dfLimited
        return
    return

#####
## The basin() returns the basin average PMP value for a given duration field.
## If the option for a weighted average is checked in the tool parameter the script

```

```

## will weight the grid point values based on proportion of area inside the basin.
def basinAve(aoiBasin, pmpField):
    pmpPoints = env.scratchGDB + "\\PMP_Points"                # Path
of 'PMP_Points' scratch feature class
    if weightedAve:
        #arcpy.AddMessage("\tCalculating sub-basin average for " + pmpField + "(weighted)...")
        vectorGridClip = env.scratchGDB + "\\VectorGridClip"    # Path
of 'VectorGridClip' scratch feature class

        dm.MakeFeatureLayer(home + "\\Input\\Non_Storm_Data.gdb\\Vector_Grid",
"vgLayer")                # make a feature layer of vector grid cells
        dm.SelectLayerByLocation("vgLayer", "INTERSECT", aoiBasin)
# select the vector grid cells that intersect the aoiBasin polygon

        an.Clip("vgLayer", aoiBasin, vectorGridClip)            #
clips aoi vector grid to basin
        dm.AddField(pmpPoints, "WEIGHT", "DOUBLE")              #
adds 'WEIGHT' field to PMP_Points scratch feature class
        dm.MakeFeatureLayer(vectorGridClip, "vgClipLayer")      #
make a feature layer of basin clipped vector grid cells
        dm.MakeFeatureLayer(pmpPoints, "pmpPointsLayer")        #
make a feature layer of PMP_Points feature class

        dm.AddJoin("pmpPointsLayer", "ID", "vgClipLayer", "ID")
# joins PMP_Points and vectorGridBasin tables
        dm.CalculateField("pmpPointsLayer", "WEIGHT", "!vectorGridClip.Shape_Area!",
"PYTHON_9.3")            # Calculates basin area proportion to use as weight for each grid cell.
        dm.RemoveJoin("pmpPointsLayer", "vectorGridClip")

        dm.SelectLayerByLocation("pmpPointsLayer", "INTERSECT", "vgLayer")

        na = arcpy.da.TableToNumPyArray("pmpPointsLayer", (pmpField, 'WEIGHT'))
# Assign pmpPoints values and weights to Numpy array (na)
        wgtAve = np.average(na[pmpField], weights=na['WEIGHT'])    #
Calculate weighted average with Numpy average
        del na
        return round(wgtAve, 2)

    else:
        if includeSubbasin:
            #arcpy.AddMessage("\tCalculating sub-basin average for " + pmpField + "(non-
weighted)...")
            vectorGridClip = env.scratchGDB + "\\VectorGridClip"    #
Path of 'VectorGridClip' scratch feature class

```

```

        dm.MakeFeatureLayer(home + "\\Input\\Non_Storm_Data.gdb\\Vector_Grid",
"vgLayer")          # make a feature layer of vector grid cells
        dm.SelectLayerByLocation("vgLayer", "INTERSECT", aoiBasin)
# select the vector grid cells that intersect the aoiBasin polygon

        dm.MakeFeatureLayer(pmpPoints, "pmpPointsLayer")          #
make a feature layer of PMP_Points feature class

        dm.SelectLayerByLocation("pmpPointsLayer", "INTERSECT", "vgLayer")

        na = arcpy.da.TableToNumPyArray("pmpPointsLayer", pmpField)
# Assign pmpPoints values and weights to Numpy array (na)
        fieldAve = np.average(na[pmpField])          #
Calculates arithmetic mean
        del na
        return round(fieldAve, 2)

    else:
        arcpy.AddMessage("\tCalculating basin average for " + pmpField + "(not
weighted)...")
        na = arcpy.da.TableToNumPyArray(pmpPoints, pmpField)
# Assign pmpPoints values to Numpy array (na)
        fieldAve = np.average(na[pmpField])          #
Calculates arithmetic mean
        del na
        return round(fieldAve, 2)

#####
## This basinZone() function returns a list containing transposition zone ID
## (as an integer)

def basinZone(bas):    ## This function returns the basin location transposition zone
    tempBasin = env.scratchGDB + "\\tempBasin"
    tempCentroid = env.scratchGDB + "\\tempCentroid"
    joinFeat = home + "\\Input\\Non_Storm_Data.gdb\\Vector_Grid"
    joinOutput = env.scratchGDB + "\\joinOut"
    dm.Dissolve(bas, tempBasin)
    desc = arcpy.Describe(tempBasin)
    sr = desc.spatialReference
    #dm.FeatureToPoint(tempBasin, tempCentroid, "INSIDE")

    dm.CreateFeatureclass(env.scratchGDB, "tempCentroid", "POINT", spatial_reference = sr)
    with arcpy.da.InsertCursor(tempCentroid, "SHAPE@XY") as iCur:
        with arcpy.da.SearchCursor(tempBasin, "SHAPE@") as sCur:
            for sRow in sCur:

```

```

        cent = sRow[0].centroid          # get the centroid
        iCur.insertRow([(cent.X,cent.Y)])# write it to the new feature class

an.SpatialJoin(tempCentroid, joinFeat, joinOutput)
centZone = arcpy.da.SearchCursor(joinOutput, ("ZONE",)).next()[0]
del tempBasin, tempCentroid, joinFeat, joinOutput, desc, sr
return (centZone)

#####
## The temporalDist() functions applies the temporal distributions scenarios
## to PMP.

def temporalDistControlStorm_06hr(stormType, outPath, location, areaSize, basinName):
# Local Storm 6-hr Temporal Distributions Function

    basinPMP = outPath + "\\" + stormType + "_PMP_Basin_Average_" + areaSize
    basinPMPPoints = outPath + "\\" + stormType + "_PMP_Points_" + basinName + "_" +
areaSize                # Location of basin average PMP table
    controlStormTable = home +
"\Input\Non_Storm_Data.gdb\CONTROLLING_STORM_TEMPORAL_DISTRIBUTIONS_0
6"

    arcpy.AddMessage(stormType + " Storm - " + dur + "-hour Controlling Storm PMP
Temporal Distributions***")
    outTable = outPath + "\\Controlling_Storms_Temporal_Distributions_" + dur
    pointsArray = arcpy.da.TableToNumPyArray(basinPMPPoints, "Storm_" + dur)
    arrayList = []
    for r in pointsArray:
        arrayList.append(r[0])
    distributionList = np.unique(arrayList).tolist()
    controlPatterns = [f.name for f in arcpy.ListFields(controlStormTable)]
    TF = any(item in distributionList for item in controlPatterns)
    if TF == True:
        map = arcpy.FieldMappings()
        fm = arcpy.FieldMap()
        fm.addInputField(controlStormTable, "TIMESTEP")
        map.addFieldMap(fm)
        fm2 = arcpy.FieldMap()
        fm2.addInputField(controlStormTable, "MINUTE")
        map.addFieldMap(fm2)
        for field in distributionList:
            if (field in controlPatterns):
                fm3 = arcpy.FieldMap()
                fm3.addInputField(controlStormTable, field)
                map.addFieldMap(fm3)
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")

```

```

        arcpy.TableToTable_conversion(controlStormTable, outPath,
"Controlling_Storms_Temporal_Distributions_" + dur, "", map)                # Copy 6-
hour temporal dist. factors table to output location
        sixHour = arcpy.da.SearchCursor(basinPMP, ("PMP_06",)).next()[0]    # Gets
6-hour PMP depth
        for distribution in distributionList:                                # Loops through each 6-hour
temporal distribution
            arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
            with arcpy.da.UpdateCursor(outTable, distribution) as cursor:    # Cursor to
apply temporal factor to 6-hour PMP
                for row in cursor:
                    row[0] = row[0] * sixHour
                    cursor.updateRow(row)
                del row, cursor

        dists6hr = []    # add suffix to distribution pattern name
        for dist in distributionList:
            dists6hr.append(dist + " (6-hr)")

        checkTemporal(stormType, outPath, outTable, dists6hr, dur, areaSize)
    else:
        arcpy.AddMessage("****Controlling Storm does not have any temporal distributions for
this duration****")

    return

```

```

def temporalDistControlStorm_12hr(stormType, outPath, location, areaSize, basinName):
# Local Storm 12-hr Temporal Distributions Function

    basinPMP = outPath + "\\ " + stormType + "_PMP_Basin_Average_" + areaSize
    basinPMPPoints = outPath + "\\ " + stormType + "_PMP_Points_" + basinName + "_" +
areaSize    # Location of basin average PMP table
    controlStormTable = home +
"\\Input\\Non_Storm_Data.gdb\\CONTROLLING_STORM_TEMPORAL_DISTRIBUTIONS_1
2"

    arcpy.AddMessage(stormType + " Storm - " + dur + "-hour Controlling Storm PMP
Temporal Distributions****")
    outTable = outPath + "\\Controlling_Storms_Temporal_Distributions_" + dur
    pointsArray = arcpy.da.TableToNumPyArray(basinPMPPoints, "Storm_" + dur)
    arrayList = []
    for r in pointsArray:
        arrayList.append(r[0])
    distributionList = np.unique(arrayList).tolist()
    controlPatterns = [f.name for f in arcpy.ListFields(controlStormTable)]
    TF = any(item in distributionList for item in controlPatterns)

```

```

if TF == True:
    map = arcpy.FieldMappings()
    fm = arcpy.FieldMap()
    fm.addInputField(controlStormTable, "TIMESTEP")
    map.addFieldMap(fm)
    fm2 = arcpy.FieldMap()
    fm2.addInputField(controlStormTable, "MINUTE")
    map.addFieldMap(fm2)
    for field in distributionList:
        fm3 = arcpy.FieldMap()
        fm3.addInputField(controlStormTable, field)
        map.addFieldMap(fm3)
    arcpy.AddMessage("\n\tCreating temporal distribution table:...")
    arcpy.TableToTable_conversion(controlStormTable, outPath,
"Controlling_Storms_Temporal_Distributions_" + dur, "", map)          # Copy 12-
hour temporal dist. factors table to output location
    twelveHour = arcpy.da.SearchCursor(basinPMP, ("PMP_12",)).next()[0]      #
Gets 12-hour PMP depth
    for distribution in distributionList:          # Loops through each 12-hour
temporal distribution
        arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
        with arcpy.da.UpdateCursor(outTable, distribution) as cursor:          # Cursor to
apply temporal factor to 12-hour PMP
            for row in cursor:
                row[0] = row[0] * twelveHour
                cursor.updateRow(row)
            del row, cursor

    dists12hr = []      # add suffix to distribution pattern name
    for dist in distributionList:
        dists12hr.append(dist + " (12-hr)")

    checkTemporal(stormType, outPath, outTable, dists12hr, dur, areaSize)
else:
    arcpy.AddMessage("****Controlling Storm does not have any temporal distributions for
this duration****")

return

def temporalDistControlStorm_24hr(stormType, outPath, location, areaSize, basinName):
# 24-hr Temporal Distributions Function
    basinPMP = outPath + "\\ " + stormType + "_PMP_Basin_Average_" + areaSize
    basinPMPPoints = outPath + "\\ " + stormType + "_PMP_Points_" + basinName + "_" +
areaSize          # Location of basin average PMP table

```

```

controlStormTable = home +
"\\Input\\Non_Storm_Data.gdb\\CONTROLLING_STORM_TEMPORAL_DISTRIBUTIONS_2
4"

arcpy.AddMessage(stormType + " Storm - " + dur + "-hour Controlling Storm PMP
Temporal Distributions****")
outTable = outPath + "\\Controlling_Storms_Temporal_Distributions_" + dur
pointsArray = arcpy.da.TableToNumPyArray(basinPMPPoints, "Storm_" + dur)
arrayList = []
for r in pointsArray:
    arrayList.append(r[0])
distributionList = np.unique(arrayList).tolist()
controlPatterns = [f.name for f in arcpy.ListFields(controlStormTable)]
TF = any(item in distributionList for item in controlPatterns)
if TF == True:
    map = arcpy.FieldMappings()
    fm = arcpy.FieldMap()
    fm.addInputField(controlStormTable, "TIMESTEP")
    map.addFieldMap(fm)
    fm2 = arcpy.FieldMap()
    fm2.addInputField(controlStormTable, "MINUTE")
    map.addFieldMap(fm2)
    for field in distributionList:
        fm3 = arcpy.FieldMap()
        fm3.addInputField(controlStormTable, field)
        map.addFieldMap(fm3)
    arcpy.AddMessage("\n\tCreating temporal distribution table:...")
    arcpy.TableToTable_conversion(controlStormTable, outPath,
"Controlling_Storms_Temporal_Distributions_" + dur, "", map)          # Copy 24-
hour temporal dist. factors table to output location
    twentyfourHour = arcpy.da.SearchCursor(basinPMP, ("PMP_24",)).next()[0]
# Gets 24-hour PMP depth
    for distribution in distributionList:          # Loops through each 24-hour
temporal distribution
        arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
        with arcpy.da.UpdateCursor(outTable, distribution) as cursor:          # Cursor to
apply temporal factor to 24-hour PMP
            for row in cursor:
                row[0] = row[0] * twentyfourHour
                cursor.updateRow(row)
            del row, cursor

dists24hr = []          # add suffix to distribution pattern name
for dist in distributionList:
    dists24hr.append(dist + " (24-hr)")

```

```

        checkTemporal(stormType, outPath, outTable, dists24hr, dur, areaSize)
    else:
        arcpy.AddMessage("****Controlling Storm does not have any temporal distributions for
this duration****")

    return

def temporalDistControlStorm_72hr(stormType, outPath, location, areaSize, basinName):
# Local Storm 72-hr Temporal Distributions Function

    basinPMP = outPath + "\\\" + stormType + "_PMP_Basin_Average_" + areaSize
    basinPMPPoints = outPath + "\\\" + stormType + "_PMP_Points_" + basinName + "_" +
areaSize
                                # Location of basin average PMP table
    controlStormTable = home +
"\\Input\\Non_Storm_Data.gdb\\CONTROLLING_STORM_TEMPORAL_DISTRIBUTIONS_7
2"
    arcpy.AddMessage(stormType + " Storm - " + dur + "-hour Controlling Storm PMP
Temporal Distributions****")
    outTable = outPath + "\\Controlling_Storms_Temporal_Distributions_" + dur
    pointsArray = arcpy.da.TableToNumPyArray(basinPMPPoints, "Storm_" + dur)
    arrayList = []
    for r in pointsArray:
        arrayList.append(r[0])
    distributionList = np.unique(arrayList).tolist()
    controlPatterns = [f.name for f in arcpy.ListFields(controlStormTable)]
    TF = any(item in distributionList for item in controlPatterns)
    if TF == True:
        map = arcpy.FieldMappings()
        fm = arcpy.FieldMap()
        fm.addInputField(controlStormTable, "TIMESTEP")
        map.addFieldMap(fm)
        fm2 = arcpy.FieldMap()
        fm2.addInputField(controlStormTable, "MINUTE")
        map.addFieldMap(fm2)
        for field in distributionList:
            fm3 = arcpy.FieldMap()
            fm3.addInputField(controlStormTable, field)
            map.addFieldMap(fm3)
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")
        arcpy.TableToTable_conversion(controlStormTable, outPath,
"Controlling_Storms_Temporal_Distributions_" + dur, "", map)
# Copy 72-
hour temporal dist. factors table to output location
        seventyTwoHour = arcpy.da.SearchCursor(basinPMP, ("PMP_72",)).next()[0]
# Gets 72-hour PMP depth

```

```

        for distribution in distributionList:                                # Loops through each 72-hour
temporal distribution
            arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
            with arcpy.da.UpdateCursor(outTable, distribution) as cursor:    # Cursor to
apply temporal factor to 72-hour PMP
                for row in cursor:
                    row[0] = row[0] * seventyTwoHour
                    cursor.updateRow(row)
                del row, cursor

        dists72hr = []    # add suffix to distribution pattern name
        for dist in distributionList:
            dists72hr.append(dist + " (72-hr)")

        checkTemporal(stormType, outPath, outTable, dists72hr, dur, areaSize)
    else:
        arcpy.AddMessage("***Controlling Storm does not have any temporal distributions for
this duration***")

    return

def temporalDistLS2(stormType, outPath, location, areaSize):                # Local Storm 2-
hr Temporal Distribution Function
    basinPMP = outPath + "\\\" + stormType + "_PMP_Basin_Average_" + areaSize
# Location of basin average PMP table

    if stormType == "Local":
        arcpy.AddMessage("\n***Local Storm - 2-hour PMP Temporal Distribution***")

        temporalDistTable_2hr = home +
"\Input\Non_Storm_Data.gdb\LS_TEMPORAL_DISTRIBUTIONS_02HR" # 2-hour
Temporal distribution factors table
        outTable = outPath + "\\LS_Temporal_Distributions_02hr"
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")
        dm.Copy(temporalDistTable_2hr, outTable)    # Copy 2-hour temporal
dist. factors table to output location
        distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_2hr,
"LS*")]    # Create a list of 2-hour distribution field names
        arcpy.AddMessage("\n\tDistribution Field Names: " + str(distributionList))

        oneHour = arcpy.da.SearchCursor(basinPMP, ("PMP_01",)).next()[0]    # Gets
1-hour PMP depth
        twoHour = arcpy.da.SearchCursor(basinPMP, ("PMP_02",)).next()[0]    # Gets
2-hour PMP depth

```

```

        for distribution in distributionList:                                # Loops through each 2-hour
temporal distribution
        arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
        accumPMP = 0
        with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to apply temporal factor to 2-hour PMP
            for row in cursor:
                if row[1] <= 6:                                            # Leave loop once a specified row is
reached
                    accumPMP += (twoHour - oneHour) / 12
                    row[0] = accumPMP
                    cursor.updateRow(row)
                if row[1] > 6 and row[1] <= 18:                            # Constrain update to rows 7-
18
                    accumPMP += oneHour * row[0]
                    row[0] = accumPMP
                    cursor.updateRow(row)
                if row[1] > 18 and row[1] <= 24:                            # Constrain update to rows
19-24
                    accumPMP += (twoHour - oneHour) / 12
                    row[0] = accumPMP
                    cursor.updateRow(row)
            del row, cursor
        checkTemporal(stormType, outPath, outTable, distributionList, dur, areaSize) # Calls
checkTemporal function
        return

def temporalDistLS(stormType, outPath, location, areaSize):                # Local Storm 6-
hr Temporal Distributions Function
    basinPMP = outPath + "\\" + stormType + "_PMP_Basin_Average_" + areaSize
# Location of basin average PMP table

    if stormType == "Local":
        arcpy.AddMessage("\n***Local Storm - 6-hour PMP Temporal Distributions***")

        temporalDistTable_6hr = home +
"\Input\Non_Storm_Data.gdb\LS_TEMPORAL_DISTRIBUTIONS_06HR" # 6-hour
Temporal distribution factors table
        outTable = outPath + "\\LS_Temporal_Distributions_6hr"
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")
        dm.Copy(temporalDistTable_6hr, outTable)                        # Copy 6-hour temporal
dist. factors table to output location
        distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_6hr,
"LS*")] # Create a list of 6-hour distribution field names

```

```

        arcpy.AddMessage("\n\tDistribution Field Names: " + str(distributionList))
        sixHour = arcpy.da.SearchCursor(basinPMP, ("PMP_06",)).next()[0]          # Gets
6-hour PMP depth
        for distribution in distributionList:                                # Loops through each 6-hour
temporal distribution
            arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
            with arcpy.da.UpdateCursor(outTable, distribution) as cursor:      # Cursor to
apply temporal factor to 6-hour PMP
                for row in cursor:
                    row[0] = row[0] * sixHour
                    cursor.updateRow(row)
                del row, cursor
            checkTemporal(stormType, outPath, outTable, distributionList, dur, areaSize)
        return

def temporalDist_LS12hr(stormType, outPath, location, areaSize):                # Local
Storm 12-hr Temporal Distributions Function
    basinPMP = outPath + "\\ " + stormType + "_PMP_Basin_Average_" + areaSize
# Location of basin average PMP table

    if stormType == "Local":
        arcpy.AddMessage("\n****" + stormType + " Storm - 12hr PMP Temporal
Distributions****")

        temporalDistTable_12hr = home +
"\\Input\\Non_Storm_Data.gdb\\LS_TEMPORAL_DISTRIBUTIONS_12HR"  # Local Storm
Temporal distribution factors table
        outTable = outPath + "\\LS_Temporal_Distributions_12hr"
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")
        dm.Copy(temporalDistTable_12hr, outTable)                        # Copy temporal dist.
factors table to output location
        distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_12hr,
"LS*")]    # Create a list of 12-hour distribution field names
        arcpy.AddMessage("\n\tDistribution Field Names: " + str(distributionList))

        largest6 = arcpy.da.SearchCursor(basinPMP, ("PMP_06",)).next()[0]      #
Calculate largest 6-hour period PMP
        second6 = (arcpy.da.SearchCursor(basinPMP, ("PMP_12",)).next()[0] - largest6)/2
# Calculate the next largest 6-hr period PMP and divide by 2

        arcpy.AddMessage("\n\tLargest 6-hour Period: " + str(largest6))
        arcpy.AddMessage("\tFirst 3-hour: " + str(second6))
        arcpy.AddMessage("\tLast 3-hour: " + str(second6))

```

```

        for distribution in distributionList:                                # Loops through each 12-hour
temporal distribution
        arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
        arcpy.AddMessage("\t\tFirst 3-hour Period...")
        accumPMP = 0
        with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to evenly distribute half of 2nd largest 6-hr into first 3 hours
            for row in cursor:
                if row[1] <= 36:                                           # Leave loop once a row containing a
temporal dist. factor (ie, first 3h period) is reached
                    accumPMP += second6 / 36
                    row[0] = accumPMP
                    cursor.updateRow(row)
                del row, cursor

            arcpy.AddMessage("\t\tLargest 6-hour Period...")
            with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to apply temporal factors to largest 6-hour PMP
                for row in cursor:
                    if row[1] > 36 and row[1] <= 108:                     # Constrain update to rows
36-108 (middle 6hr period)
                        accumPMP = (largest6 * row[0]) + second6
                        row[0] = accumPMP
                        cursor.updateRow(row)
                    del row, cursor

            arcpy.AddMessage("\t\tLast 3-hour Period...")
            whereClause = distribution + " IS NULL"
            with arcpy.da.UpdateCursor(outTable, distribution, whereClause) as cursor:      #
Cursor to evenly distribute half of 2nd largest 6-hr into last 3 hours
                for row in cursor:
                    accumPMP += second6 / 36
                    row[0] = accumPMP
                    cursor.updateRow(row)
                del row, cursor, accumPMP, whereClause
            checkTemporal(stormType, outPath, outTable, distributionList, dur, areaSize)
        return

def temporalDist_24hr(stormType, outPath, location, areaSize):              #
General/Tropical Storm 24-hr Temporal Distributions Function
    basinPMP = outPath + "\\\" + stormType + "_PMP_Basin_Average_" + areaSize
# Location of basin average PMP table

    if stormType == "General":

```

```

    arcpy.AddMessage("\n***" + stormType + " Storm - 24hr PMP Temporal
Distributions***")

    temporalDistTable_24hr = home +
    "\\Input\\Non_Storm_Data.gdb\\GS_TEMPORAL_DISTRIBUTIONS_24HR" # General
Storm Temporal distribution factors table
    outTable = outPath + "\\GS_Temporal_Distributions_24hr"
    arcpy.AddMessage("\n\tCreating temporal distribution table:...")
    dm.Copy(temporalDistTable_24hr, outTable) # Copy temporal dist.
factors table to output location
    distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_24hr,
"GS*")] # Create a list of 24-hour distribution field names
    arcpy.AddMessage("\n\tDistribution Field Names: " + str(distributionList))
    twentyfourHour = arcpy.da.SearchCursor(basinPMP, ("PMP_24",)).next()[0] #
Gets 24-hour PMP depth
    for distribution in distributionList: # Loops through each 24-hour
temporal distribution
        arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
        with arcpy.da.UpdateCursor(outTable, distribution) as cursor: # Cursor to
apply temporal factor to 24-hour PMP
            for row in cursor:
                row[0] = row[0] * twentyfourHour
                cursor.updateRow(row)
            del row, cursor
    checkTemporal(stormType, outPath, outTable, distributionList, dur, areaSize)

    if stormType == "Tropical":
        arcpy.AddMessage("\n***" + stormType + " Storm - 24hr PMP Temporal
Distributions***")

        temporalDistTable_24hr = home +
        "\\Input\\Non_Storm_Data.gdb\\TS_TEMPORAL_DISTRIBUTIONS_24HR" # Tropical
Storm Temporal distribution factors table
        outTable = outPath + "\\TS_Temporal_Distributions_24hr"
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")
        dm.Copy(temporalDistTable_24hr, outTable) # Copy temporal dist.
factors table to output location
        distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_24hr,
"TS*")] # Create a list of 24-hour distribution field names
        arcpy.AddMessage("\n\tDistribution Field Names: " + str(distributionList))
        twentyfourHour = arcpy.da.SearchCursor(basinPMP, ("PMP_24",)).next()[0] #
Gets 24-hour PMP depth
        for distribution in distributionList: # Loops through each 24-hour
temporal distribution
            arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)

```

```

        with arcpy.da.UpdateCursor(outTable, distribution) as cursor:            # Cursor to
apply temporal factor to 24-hour PMP
        for row in cursor:
            row[0] = row[0] * twentyfourHour
            cursor.updateRow(row)
        del row, cursor
        checkTemporal(stormType, outputPath, outTable, distributionList, dur, areaSize)
    return

def temporalDist_48hr(stormType, outputPath, location, areaSize):                #
General/Tropical Storm 48-hr Temporal Distributions Function
    basinPMP = outputPath + "\\\" + stormType + \"_PMP_Basin_Average_\" + areaSize
# Location of basin average PMP table

    if stormType == \"General\":
        arcpy.AddMessage(\"\\n***\" + stormType + \" Storm - 48hr PMP Temporal
Distributions***\")

        temporalDistTable_48hr = home +
\"\\Input\\Non_Storm_Data.gdb\\GS_TEMPORAL_DISTRIBUTIONS_48HR\" # General
Storm Temporal distribution factors table
        outTable = outputPath + \"\\GS_Temporal_Distributions_48hr\"
        arcpy.AddMessage(\"\\n\\tCreating temporal distribution table:...\")
        dm.Copy(temporalDistTable_48hr, outTable)                            # Copy temporal dist.
factors table to output location
        distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_48hr,
\"GS*\")] # Create a list of 48-hour distribution field names
        arcpy.AddMessage(\"\\n\\tDistribution Field Names: \" + str(distributionList))

        largest24 = arcpy.da.SearchCursor(basinPMP, (\"PMP_24\",)).next()[0]      #
Calculate largest 24-hour period PMP
        second24 = (arcpy.da.SearchCursor(basinPMP, (\"PMP_48\",)).next()[0] - largest24)/2
# Calculate the next largest 24-hr period PMP and divide by 2

        arcpy.AddMessage(\"\\n\\tLargest 24-hour Period: \" + str(largest24))
        arcpy.AddMessage(\"\\tFirst 12-hour: \" + str(second24))
        arcpy.AddMessage(\"\\tLast 12-hour: \" + str(second24))

        for distribution in distributionList:                                # Loops through each 24-hour
temporal distribution
            arcpy.AddMessage(\"\\n\\tApplying temporal distribution for: \" + distribution)
            arcpy.AddMessage(\"\\t\\tFirst 12-hour Period...\")
            accumPMP = 0

```

```

        with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to evenly distribute half of 2nd largest 24-hr into first 12 hours
        for row in cursor:
            if row[1] <= 48:                                     # Leave loop once a row containing a
temporal dist. factor (ie, first 12h period) is reached
                accumPMP += second24 / 48
                row[0] = accumPMP
                cursor.updateRow(row)
            del row, cursor

        arcpy.AddMessage("\t\tLargest 24-hour Period...")
        with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to apply temporal factors to largest 24-hour PMP
        for row in cursor:
            if row[1] > 48 and row[1] <= 144:                   # Constrain update to rows
49-144 (second 24hr period)
                accumPMP = (largest24 * row[0]) + second24
                row[0] = accumPMP
                cursor.updateRow(row)
            del row, cursor

        arcpy.AddMessage("\t\tLast 12-hour Period...")
        whereClause = distribution + " IS NULL"
        with arcpy.da.UpdateCursor(outTable, distribution, whereClause) as cursor:           #
Cursor to evenly distribute half of 2nd largest 24-hr into last 12 hours
        for row in cursor:
            accumPMP += second24 / 48
            row[0] = accumPMP
            cursor.updateRow(row)
        del row, cursor, accumPMP, whereClause
        checkTemporal(stormType, outputPath, outTable, distributionList, dur, areaSize)

    if stormType == "Tropical":
        arcpy.AddMessage("\n***" + stormType + " Storm - 48hr PMP Temporal
Distributions***")

        temporalDistTable_48hr = home +
"\Input\Non_Storm_Data.gdb\TS_TEMPORAL_DISTRIBUTIONS_48HR" # Tropical
Storm Temporal distribution factors table
        outTable = outputPath + "\\TS_Temporal_Distributions_48hr"
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")
        dm.Copy(temporalDistTable_48hr, outTable) # Copy temporal dist.
factors table to output location
        distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_48hr,
"TS*")] # Create a list of 48-hour distribution field names
        arcpy.AddMessage("\n\tDistribution Field Names: " + str(distributionList))

```

```

        largest24 = arcpy.da.SearchCursor(basinPMP, ("PMP_24",)).next()[0] #
Calculate largest 24-hour period PMP
        second24 = (arcpy.da.SearchCursor(basinPMP, ("PMP_48",)).next()[0] - largest24)/2
# Calculate the third largest 12-hr period PMP and divide by 2

arcpy.AddMessage("\n\tLargest 24-hour Period: " + str(largest24))
arcpy.AddMessage("\tFirst 12-hour: " + str(second24))
arcpy.AddMessage("\tLast 12-hour: " + str(second24))

        for distribution in distributionList: # Loops through each 24-hour
temporal distribution
            arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
            arcpy.AddMessage("\t\tFirst 12-hour Period...")
            accumPMP = 0
            with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to evenly distribute half of 2nd largest 24-hour PMP to first 12 hours
                for row in cursor:
                    if row[1] <= 48: # Leave loop once a row containing a
temporal dist. factor (ie, first 12h period) is reached
                        accumPMP += second24 / 48
                        row[0] = accumPMP
                        cursor.updateRow(row)
                    del row, cursor

            arcpy.AddMessage("\t\tLargest 24-hour Period...")
            with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to apply temporal factors to largest 24-hour PMP
                for row in cursor:
                    if row[1] > 48 and row[1] <= 144: # Constrain update to rows
49-144 (second 24hr period)
                        accumPMP = (largest24 * row[0]) + second24
                        row[0] = accumPMP
                        cursor.updateRow(row)
                    del row, cursor

            arcpy.AddMessage("\t\tLast 12-hour Period...")
            whereClause = distribution + " IS NULL"
            with arcpy.da.UpdateCursor(outTable, distribution, whereClause) as cursor: #
Cursor to evenly distribute half of 2nd largest 24-hr into last 12 hours
                for row in cursor:
                    accumPMP += second24 / 48
                    row[0] = accumPMP
                    cursor.updateRow(row)

```

```

del row, cursor, accumPMP, whereClause

checkTemporal(stormType, outPath, outTable, distributionList, dur, areaSize)
return

def temporalDist_72hr(stormType, outPath, location, areaSize):          #
General/Tropical Storm 72-hr Temporal Distributions Function
    basinPMP = outPath + "\\ " + stormType + "_PMP_Basin_Average_" + areaSize
# Location of basin average PMP table

    if stormType == "General":
        arcpy.AddMessage("\n****" + stormType + " Storm - 72hr PMP Temporal
Distributions****")
        temporalDistTable_72hr = home +
"\\Input\\Non_Storm_Data.gdb\\GS_TEMPORAL_DISTRIBUTIONS_72HR" # General
Storm Temporal distribution factors table
        outTable = outPath + "\\GS_Temporal_Distributions_72hr"
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")
        dm.Copy(temporalDistTable_72hr, outTable)                # Copy temporal dist.
factors table to output location
        distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_72hr,
"GS*")] # Create a list of 72-hour distribution field names
        arcpy.AddMessage("\n\tDistribution Field Names: " + str(distributionList))

        largest24 = arcpy.da.SearchCursor(basinPMP, ("PMP_24",)).next()[0]
# Calculate largest 24-hour period PMP
        second24 = arcpy.da.SearchCursor(basinPMP, ("PMP_48",)).next()[0] - largest24
# Calculate 2nd-largest 24-hour period PMP
        third24 = arcpy.da.SearchCursor(basinPMP, ("PMP_72",)).next()[0] -
arcpy.da.SearchCursor(basinPMP, ("PMP_48",)).next()[0] # Calculate 3rd-largest 24-hour
period PMP

        arcpy.AddMessage("\n\tLargest 24-hour: " + str(largest24))
        arcpy.AddMessage("\tSecond largest 24-hour: " + str(second24))
        arcpy.AddMessage("\tThird largest 24-hour: " + str(third24))

        for distribution in distributionList:                # Loops through each 72-hour
temporal distribution
            arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
            arcpy.AddMessage("\t\tFirst 24-hour Period...")
            accumPMP = 0
            with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to evenly distribute 2nd largest 24-hour
                for row in cursor:

```

```

        if row[1] <= 96:                                # Leave loop once a row containing a
temporal dist. factor (ie, second 24h period) is reached
            accumPMP += second24 / 96
            row[0] = accumPMP
            cursor.updateRow(row)
        del row, cursor

    arcpy.AddMessage("\t\tSecond 24-hour Period...")
    with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to apply temporal factors to largest 24-hour PMP
        for row in cursor:
            if row[1] > 96 and row[1] <= 192:            # Constrain update to rows
97-192 (second 24hr period)
                accumPMP = (largest24 * row[0]) + second24
                row[0] = accumPMP
                cursor.updateRow(row)
            del row, cursor

    arcpy.AddMessage("\t\tThird 24-hour Period...")
    whereClause = distribution + " IS NULL"
    with arcpy.da.UpdateCursor(outTable, distribution, whereClause) as cursor:    #
Cursor to evenly distribute 3rd largest hour over remaining empty rows
        for row in cursor:
            accumPMP += third24 / 96
            row[0] = accumPMP
            cursor.updateRow(row)
        del row, cursor, accumPMP, whereClause
    checkTemporal(stormType, outputPath, outTable, distributionList, dur, areaSize)

    if stormType == "Tropical":
        arcpy.AddMessage("\n***" + stormType + " Storm - 72hr PMP Temporal
Distributions***")
        temporalDistTable_72hr = home +
"\Input\Non_Storm_Data.gdb\TS_TEMPORAL_DISTRIBUTIONS_72HR"    # Tropical
Storm Temporal distribution factors table
        outTable = outputPath + "\TS_Temporal_Distributions_72hr"
        arcpy.AddMessage("\n\tCreating temporal distribution table:...")
        dm.Copy(temporalDistTable_72hr, outTable)                # Copy temporal dist.
factors table to output location
        distributionList = [field.name for field in arcpy.ListFields(temporalDistTable_72hr,
"TS*")]    # Create a list of 72-hour distribution field names
        arcpy.AddMessage("\n\tDistribution Field Names: " + str(distributionList))

    largest24 = arcpy.da.SearchCursor(basinPMP, ("PMP_24",)).next()[0]
# Calculate largest 24-hour period PMP

```

```

second24 = arcpy.da.SearchCursor(basinPMP, ("PMP_48",)).next()[0] - largest24
# Calculate 2nd-largest 24-hour period PMP
third24 = arcpy.da.SearchCursor(basinPMP, ("PMP_72",)).next()[0] -
arcpy.da.SearchCursor(basinPMP, ("PMP_48",)).next()[0] # Calculate 3rd-largest 24-hour
period PMP

arcpy.AddMessage("\n\tLargest 24-hour: " + str(largest24))
arcpy.AddMessage("\tSecond largest 24-hour: " + str(second24))
arcpy.AddMessage("\tThird largest 24-hour: " + str(third24))

for distribution in distributionList:                                # Loops through each 24-hour
temporal distribution
    arcpy.AddMessage("\n\tApplying temporal distribution for: " + distribution)
    arcpy.AddMessage("\t\tFirst 24-hour Period...")
    accumPMP = 0
    with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to evenly distribute 2nd largest hour
        for row in cursor:
            if row[1] <= 96:                                # Leave loop once a row containing a
temporal dist. factor (ie, second 24h period) is reached
                accumPMP += second24 / 96
                row[0] = accumPMP
                cursor.updateRow(row)
            del row, cursor

    arcpy.AddMessage("\t\tSecond 24-hour Period...")
    with arcpy.da.UpdateCursor(outTable, [distribution, "TIMESTEP"]) as cursor:
# Cursor to apply temporal factors to largest 24-hour PMP
        for row in cursor:
            if row[1] > 96 and row[1] <= 192:                # Constrain update to rows
97-192 (second 24hr period)
                accumPMP = (largest24 * row[0]) + second24
                row[0] = accumPMP
                cursor.updateRow(row)
            del row, cursor

    arcpy.AddMessage("\t\tThird 24-hour Period...")
    whereClause = distribution + " IS NULL"
    with arcpy.da.UpdateCursor(outTable, distribution, whereClause) as cursor:                #
Cursor to evenly distribute 3rd largest hour over remaining empty rows
        for row in cursor:
            accumPMP += third24 / 96
            row[0] = accumPMP
            cursor.updateRow(row)
        del row, cursor, accumPMP, whereClause

```

```

        checkTemporal(stormType, outPath, outTable, distributionList, dur, areaSize)
    return

    ## This portion of the code checks to make sure none of the temporal distributions
    ## are exceeding the PMP values for any durations. It adds a table to the output
    ## folder called CheckTemporal.
    ##~~~~~
    ~~~~~

def checkTemporal(stormType, outPath, TemporalTable, distributionFields, dur, areaSize):
    basinPMP = outPath + "\\\" + stormType + \"_PMP_Basin_Average_\" + areaSize
# Location of basin average PMP table
    pmpFields = [field.name for field in arcpy.ListFields(basinPMP, \"PMP_*)\"]
# PMP duration run
    temporalFields = [field.name for field in arcpy.ListFields(TemporalTable)]
    table = arcpy.Describe(TemporalTable)
    tableName = table.name

    pmp = []
    updates with PMP values for each duration run
    i = 0
    while i < len(pmpFields):
        with arcpy.da.SearchCursor(basinPMP,pmpFields) as cursor:
            for row in cursor:
                pmp.append(row[i])
                i += 1
    del i, cursor

    checkTable = outPath + \"\\Temporal_Distribution_Check_\" + stormType
    arcpy.AddMessage(\"\\nCheckTable: \" + checkTable)
    maxFields = []
    for each duration
        checkFields = []
    for each duration
        if arcpy.Exists(checkTable):
            with arcpy.da.InsertCursor(checkTable, \"PATTERN\") as cursor:
                for val in distributionFields:
                    cursor.insertRow([val])
            i = 0
            for pmpField in pmpFields:
                with arcpy.da.UpdateCursor(checkTable, pmpField) as cursor:
                    for row in cursor:
                        row = pmp[i]

```

```

        cursor.updateRow([row])
        i += 1
    del i, cursor
else:
    checkTable = dm.CreateTable(outPath, "Temporal_Distribution_Check_" + stormType)
#Creates table in output folder, adds field, and populates field with distributions
    dm.AddField(checkTable, "PATTERN", "TEXT", "", "", 50)
    with arcpy.da.InsertCursor(checkTable, "PATTERN") as cursor:
        for val in distributionFields:
            cursor.insertRow([val])
    for maxField in pmpFields:
        newField = maxField.replace("PMP", "MAX")
        maxFields.append(newField)
    del newField
    for checkField in pmpFields:
        newField = checkField.replace("PMP", "CHECK")
        checkFields.append(newField)
    del newField
    i = 0 #Populate fields
    for pmpField in pmpFields:
        dm.AddField(checkTable, pmpField, "DOUBLE", "", "", 50)
        dm.AddField(checkTable, maxFields[i], "DOUBLE", "", "", 50)
        dm.AddField(checkTable, checkFields[i], "TEXT", "", "", 50)
        with arcpy.da.UpdateCursor(checkTable, pmpField) as cursor:
            for row in cursor:
                row = pmp[i]
                cursor.updateRow([row])
            i += 1
    del i, cursor

step = arcpy.da.SearchCursor(TemporalTable, ("MINUTE",)).next()[0]
if step == 15:
    dic = {"01": 4, "02": 8, "03": 12, "04": 16, "05": 20, "06": 24, "12": 48, "24": 96, "48":
192, "72": 288, "96": 384, "120": 480} # Dictionary to convert durations into 15-minute
timesteps
elif step == 5:
    dic = {"01": 12, "02": 24, "03": 36, "04": 48, "05": 60, "06": 72, "12": 144, "24": 288,
"48": 576, "72": 864, "96": 1152, "120": 1440}
elif step == 60:
    dic = {"01": 1, "02": 2, "03": 3, "04": 4, "05": 5, "06": 6, "12": 12, "24": 24, "48": 48,
"72": 72, "96": 96, "120": 120}
    # arcpy.AddMessage(str(step) + " Minute distribution Pattern.....")

maxFields = [field.name for field in arcpy.ListFields(checkTable, "MAX*")]
i = 0 # Calculates incremental PMP depths from temporal
distribution and gets maximum rainfall for each duration run

```

```

d = durList.index(dur) + 1
for dur in durList[:d]:
    k = dic[dur]
    p = 3 # Skip first 3 fields in
temporaltable (objectID, Timesteps, minutes)
    for distribution in distributionFields:
        incPMP = []
        previousRow = 0
        with arcpy.da.SearchCursor(TemporalTable, temporalFields) as cursor:
            for row in cursor:
                increment = row[p] - previousRow
                previousRow = row[p]
                incPMP.append(increment)
        na = np.array(incPMP)
        sumList = np.convolve(na,np.ones(k))
        maxPMP = max(sumList)
        maximumPMP = math.trunc(maxPMP * 10 ** 2.0) / 10 ** 2.0
        p += 1

        x = 0
        with arcpy.da.UpdateCursor(checkTable, ["PATTERN", maxFields[i]]) as cursor:
# Updates table with max values
            for row in cursor:
                if row[0] == distribution:
                    row[1] = maximumPMP
                    x += 1
                    cursor.updateRow(row)

            i += 1
            del i, k, cursor, x
            with arcpy.da.UpdateCursor(checkTable, '*') as cursor: # Compares PMP
values to max values for each duration. If PMP values are larger update check field with PASS
if not FAIL
                for row in cursor:
                    rec = dict(zip(cursor.fields, row))
                    arcpy.AddMessage("\n\n\tChecking temporally distributed depth-durations against
PMP: " + rec['PATTERN'] + "\n")
                    for k, v in rec.items():
                        if not k.startswith('PMP_'):
                            continue
                        _, n = k.split('_')
                        try:
                            # This try/except skips comparisons for additional durations not
present in current temporal pattern
                            mx = rec['MAX_{}'.format(n)]
                            rec['CHECK_{}'.format(n)] = 'FAIL' if v < mx else 'PASS'
                        except:

```

```

        arcpy.AddMessage("\n\tDuration not present...")
        continue
    if rec['CHECK_{}'.format(n)] == 'PASS':
        arcpy.AddMessage("\t" + str(n) + "-hour \n\t\tPMP value is... " + str(v) + "
\n\t\tmax rainfall value is..." + str(mx) + "\n\t\tThis distribution.... " +
rec['CHECK_{}'.format(n)])
    else:
        arcpy.AddMessage("\t" + str(n) + "-hour \n\t\tPMP value is... " + str(v) + "
\n\t\tmax rainfall value is..." + str(mx) + "\n\t\tThis distribution.... " +
rec['CHECK_{}'.format(n)] + "\n\t\t**Max values for duration are exceeding PMP values. Use of
this temporal distribution not recommended.")
        cursor.updateRow([rec[k] for k in cursor.fields])
        del cursor, k, v, rec
    return

```

```

#####
## The temporalCritStacked() function applies the critically stacked
## temporal distributions scenarios. The function accepts the storm type,
## output .gdb path, AOI area size, PMP duration string (hours), and
## integer timestep duration (minutes). The function outputs a gdb table.

```

```

def temporalCritStacked(stormType, outPath, area, duration, timestep):
# Function applied Critically Stacked temporal distribution
    basinPMP = outPath + "\\ " + stormType + "_PMP_Basin_Average_" + area
# Location of basin average PMP table
    if stormType == "Local" and duration == "06":
# These conditional statements define the field name based on storm type, PMP duration, and
timestep duration
        csField = "LS_" + duration + "_HOURL_" + str(timestep) + "MIN_CRIT_STACKED"
    elif stormType == "General":
        csField = "GS_" + duration + "_HOURL_" + str(timestep) + "MIN_CRIT_STACKED"
    elif stormType == "Tropical":
        csField = "TS_" + duration + "_HOURL_" + str(timestep) + "MIN_CRIT_STACKED"
    else:
        arcpy.AddMessage("\n***Invalid storm type: " + stormType)
        return
    arcpy.AddMessage("\n***" + duration + "-hour " + str(timestep) + "-min Critically Stacked
Temporal Distribution***")
    tableName = "Temporal_Distribution_" + duration + "hr_" + str(timestep) +
"min_Crit_Stacked" # Output table name
    tablePath = outPath + "\\ " + tableName
# Output table full path
    pmpFields = [field.name for field in arcpy.ListFields(basinPMP, "PMP*")]
# Gets the "PMP_XX" field names from the basin average PMP table

```

```

        if duration == "06":
            #
            These conditional statements define the key durations needed to build the critically stacked
            patterns for the following durations...
            keyDurations = [1, 2, 3, 4, 5, 6]
            elif duration == "12":
                keyDurations = [1, 2, 3, 4, 5, 6, 12]
            elif duration == "24":
                keyDurations = [1, 2, 3, 4, 5, 6, 12, 24]
            elif duration == "48":
                keyDurations = [1, 2, 3, 4, 5, 6, 12, 24, 48]
            elif duration == "72":
                keyDurations = [1, 2, 3, 4, 5, 6, 12, 24, 48, 72]
            elif duration == "96":
                keyDurations = [1, 2, 3, 4, 5, 6, 12, 24, 48, 72, 96]
            elif duration == "120":
                keyDurations = [1, 2, 3, 4, 5, 6, 12, 24, 48, 72, 96, 120]
            else:
                arcpy.AddMessage("\n\t...Critically stacked temporal distribution not available for " +
                duration + "-hour duration.")
                return
            timestepLen = int(duration) * 60 // timestep
# number of rows in output table
            xValues = [0]
            for i in keyDurations:
                #
                defines the known x-values (xp) to be used in the interpolation
                xVal = i * timestepLen // int(duration)
                xValues.append(xVal)
            del i, xVal
            yValues = [0]
            d = 0
            for i in keyDurations:
                #
                defines the known y-values (fp) to be used in the interpolation
                pmpDepth = arcpy.da.SearchCursor(basinPMP, pmpFields).next()[d]
                yValues.append(pmpDepth)
                d += 1
            del d, i, pmpDepth

            x = np.arange(0, timestepLen + 1, 1)
# defines the x points at which to interpolate values
            xp = np.asarray(xValues)
            #
            np.asarray converts lists into numpy arrays
            fp = np.asarray(yValues)
            y = np.interp(x, xp, fp)
            inc = []
            prevDepth = 0

```

```

    i = 0
    for depth in np.nditer(y):
        #
        populates incremental depths list 'inc' with y array
        inc.append(depth - prevDepth)
        prevDepth = depth
        i += 1
    del i, prevDepth
    periods = int(duration)
    #
    defines number of periods (known hours) as the duration
    periodLen = 60 // timestep
    #
    defines number of timesteps (minutes) in each period
    ranks = []
    stackRank = 1
    i = 0
    while i < periods:
        #
        populates list 'ranks' with a rank integer, one entry per period
        ranks.append(stackRank)
        stackRank += periodLen
        i += 1
    del i

    orderRanks = []
    orderRanks.insert(0, ranks.pop(0))
    for i in range(timestepLen // periodLen):
        ## orders the ranks according to critically stacked pattern. Pulls
        if ranks:
            ## (pop())
            the first rank from the ranks list and places it in the orderRanks
            orderRanks.insert(0, ranks.pop(0))
        ## list. Places next two ranks at the beginning of the list
        if ranks:
            ## and the
            following at the end of the list. Repeats until ranks is empty.
            orderRanks.insert(0, ranks.pop(0))
            if ranks:
                orderRanks.append(ranks.pop(0))
        del i
        orderRanks += [orderRanks.pop(0)]
        if orderRanks[0] == max(orderRanks):
            ## Moves last rank to the end of of orderRanks list.
            arcpy.AddMessage("\n*** moving first rank to last")
            orderRanks.append(orderRanks.pop(max))
    orderInc = []
    n = 0
    for i in range(periods):
        # gets
        the nth largest increment where n is the ordered Rank.
        for q in range(periodLen):
            nthLargest = nlargest(orderRanks[n], inc)[-1]

```

```

        orderInc.append(nthLargest)
    n += 1
    del n, i, q
    cumulative = []
    prevInc = 0
    for i in orderInc:
        Converts the incremental depths to cumulative depths and places in cumulative list
        value = round(i + prevInc, 2)
        cumulative.append(value)
        prevInc = i + prevInc
        i += 1
    del i, prevInc
    timesteps = x.tolist()
    Converts the timesteps array (x) to a list then removes the first zero entry
    timesteps.pop(0)
    minutes = []
    minutesInc = timestep
    for i in range(timestepLen):
        Constructs the minutes list to be used in output column based on timestep interval
        minutes.append(minutesInc)
        minutesInc += timestep
    del i
    dm.CreateTable(outPath, tableName)
# Create the output geodatabase table
    dm.AddField(tablePath, "TIMESTEP", "DOUBLE")
# Create "TIMESTEP" field
    dm.AddField(tablePath, "MINUTES", "DOUBLE")
# Create "MINUTES" field
    dm.AddField(tablePath, csField, "DOUBLE")
# Create cumulated rainfall field
    zipped = zip(timesteps, minutes, cumulative)
# Zip up lists of output items.
    fields = ('TIMESTEP', 'MINUTES', csField)
# Output table field names
    arcpy.AddMessage("\n\tApplying temporal distribution for: " + csField)
    with arcpy.da.InsertCursor(tablePath, fields) as cursor:
# Cursor to populate output Critically Stacked table
        for i in zipped:
            cursor.insertRow(i)
        del cursor, i
    return

#####
## This portion of the code iterates through each storm feature class in the
## 'Storm_Adj_Factors' geodatabase (evaluating the feature class only within

```

```

## the Local, Tropical, or general feature dataset). For each duration,
## at each grid point within the aoi basin, the transpositionality is
## confirmed. Then the DAD precip depth is retrieved and applied to the
## total adjustment factor to yield the total adjusted rainfall. This
## value is then sent to the updatePMP() function to update the 'PMP_Points'
## feature class.
##~~~~~
~~~~~
~~~~~##

desc = arcpy.Describe(basin)                                # Check to ensure AOI input
shape is a Polygon. If not - exit.
basinShape = desc.shapeType
if desc.shapeType == "Polygon":
    arcpy.AddMessage("\nBasin shape type: " + desc.shapeType)
else:
    arcpy.AddMessage("\nBasin shape type: " + desc.shapeType)
    arcpy.AddMessage("\nError: Input shapefile must be a polygon!\n")
    sys.exit()

createPMPfc()                                                # Call the createPMPfc() function
to create the PMP_Points feature class.

env.workspace = adjFactGDB                                  # the workspace
environment is set to the 'Storm_Adj_Factors' file geodatabase

aoiSQMI = round(getAOIarea(),2)                               # Calls the getAOIarea()
function to assign area of AOI shapefile to 'aoiSQMI'
if aoiSQMI > 100 and stormType is "Local":
    arcpy.AddMessage("\n***Warning - Local storm PMP depths only valid for basins 100
square miles or smaller***")

stormList = arcpy.ListFeatureClasses("", "Point", stormType) # List all the total
adjustment factor feature classes within the storm type feature dataset.
for dur in durList:

arcpy.AddMessage("\n*****
**\nEvaluating " + dur + "-hour duration...")

pmpList = []
driverList = []
gridRows = arcpy.SearchCursor(env.scratchGDB + "\\PMP_Points")
try:
    for row in gridRows:

```

```

        pmpList.append(0.0)                                # creates pmpList of empty
float values for each grid point to store final PMP values
        driverList.append("STORM")                        # creates driverList of
empty text values for each grid point to store final Driver Storm IDs
        del row, gridRows
except UnboundLocalError:
        arcpy.AddMessage("\n***Error: No data present within basin/AOI area.***\n")
        sys.exit()

env.workspace = adjFactGDB
for storm in stormList[:]:
        arcpy.AddMessage("\n\tEvaluating storm: " + storm + "...")
        dm.MakeFeatureLayer(storm, "stormLayer")          # creates a feature
layer for the current storm
        dm.SelectLayerByLocation("stormLayer", "HAVE_THEIR_CENTER_IN", "vgLayer")
# examines only the grid points that lie within the AOI
        gridRows = arcpy.SearchCursor("stormLayer")
        pmpField = "PMP_" + dur
        i = 0
        try:
                dadPrecip = round(dadLookup(storm, dur, aoisQMI),3)
                arcpy.AddMessage("\t\t" + dur + "-hour DAD value: " + str(dadPrecip) + chr(34))
        except TypeError:                                  # In no duration exists in the DAD
table - move to the next storm
                arcpy.AddMessage("\t***Duration '" + str(dur) + "-hour' is not present for '" +
str(storm) + "'.***\n")
                continue
                arcpy.AddMessage("\t\tComparing " + storm + " adjusted rainfall values against current
driver values...")
                transCounter = 0                            # Counter for number of grid points
transposed to
                for row in gridRows:
                        if row.TRANS == 1:                  # Only continue if grid point is
transpositionable ('1' is transpositionable, '0' is not).
                                try:                        # get total adj. factor if duration exists
                                        transCounter += 1
                                        adjRain = round(dadPrecip * row.TAF,1)
                                        if adjRain > pmpList[i]:
                                                pmpList[i] = adjRain
                                                driverList[i] = storm
                                except RuntimeError:
                                        arcpy.AddMessage("\t\t *Warning* Total Adjusted Raifnall value falied to set
for row " + str(row.CNT))
                                        break
                                        del adjRain
                                i += 1

```

```

        if transCounter == 0:
            arcpy.AddMessage("\t\tStorm not transposable to basin. Removing " + storm + " from
list...\n")
            stormList.remove(storm)
        else:
            arcpy.AddMessage("\t\tTransposed to " + str(transCounter) + "/" + str(i) + " grid
points...\n")
            del row, transCounter
            del storm, gridRows, dadPrecip
            updatePMP(pmpList, driverList, dur)          # calls function to update "PMP Points"
feature class
            del pmpList, stormList

    arcpy.AddMessage("\n'PMP_Points' Feature Class 'PMP_XX' fields update complete for all '"
+ stormType + "' storms.")

    outputPMP(stormType, aoisQMI, outputPath)          # calls outputPMP() function
    outArea = str(int(round(aoisQMI,0))) + "sqmi"
    outGDB = outLocation + "\\" + stormType + "\\PMP_" + desc.baseName + "_" + outArea +
".gdb"
    basinName = desc.baseName

    if runTemporal:                                     #Calls temporal distribution functions
        centroidLocation = basinZone(basin)
        arcpy.AddMessage("\nBasin Centroid Transposition Zone: " + str(centroidLocation))

    for dur in durList:
        if dur == "02":
            temporalDistLS2(stormType, outGDB, centroidLocation, outArea)
        if dur == "06":
            temporalDistLS(stormType, outGDB, centroidLocation, outArea)
            temporalDistControlStorm_06hr(stormType, outGDB, centroidLocation, outArea,
basinName)
        if stormType == "Local" and dur == "06":
            temporalCritStacked(stormType, outGDB, outArea, dur, 5)
        if dur == "12":
            temporalDistControlStorm_12hr(stormType, outGDB, centroidLocation, outArea,
basinName)
        if stormType == "Local" and dur == "12":
            temporalDist_LS12hr(stormType, outGDB, centroidLocation, outArea)
        if dur == "24":
            temporalDist_24hr(stormType, outGDB, centroidLocation, outArea)
            temporalCritStacked(stormType, outGDB, outArea, dur, 15)
            temporalDistControlStorm_24hr(stormType, outGDB, centroidLocation, outArea,
basinName)
        if dur == "48":

```

```

        temporalDist_48hr(stormType, outGDB, centroidLocation, outArea)
    if dur == "72":
        temporalDist_72hr(stormType, outGDB, centroidLocation, outArea)
        temporalDistControlStorm_72hr(stormType, outGDB, centroidLocation, outArea,
basinName)

```

```

    i = 0 #Creates CSV files of all output tables
    csvPath = outLocation + "\\\" + stormType + "\\CSV_" + desc.baseName + "_" + outArea +
    "\\\"
    if not arcpy.Exists(outLocation + "\\\" + stormType + "\\CSV_" + desc.baseName + "_" +
outArea):
        arcpy.CreateFolder_management(outLocation + "\\\" + stormType + "\\\", "CSV_" +
desc.baseName + "_" + outArea)
        arcpy.AddMessage("\n\t...Creating output tables as CSV files.. ")
        env.workspace = outGDB
        outTables = arcpy.ListTables()
        for t in outTables:
            arcpy.TableToTable_conversion(t, csvPath, outTables[i] + ".csv")
            i += 1
        xmlFiles = os.listdir(csvPath)
        for file in xmlFiles:
            if file.endswith(".xml"):
                os.remove(os.path.join(csvPath,file))
        return
##~~~~~
~~~~~
~~~~~##

```

```

def outputBasAveTable():
    arcpy.AddMessage("\n\nCreating basin average summary table.\n")
    tableList = basAveTables
    for table in tableList:
        arcpy.AddMessage("\t\tMerging tables... " + table)

    dm.Merge(basAveTables, outputTable)
    ## addLayerMXD(outputTable) adds output table to ArcMap session

    return
##~~~~~
~~~~~
~~~~~##

```

```

def addLayerMXD(addFC):

```

```

desc = arcpy.Describe(addFC)
layerName = desc.name
arcpy.AddMessage("\nAdding " + layerName + " table to current MXD...")
if desc.dataType == "FeatureClass":
    dm.MakeFeatureLayer(addFC, layerName)
    layer = arcpy.mapping.Layer(layerName)
    arcpy.mapping.AddLayer(df, layer)
    arcpy.AddMessage("\n" + layerName + " added to current map session.\n")
elif desc.dataType == "Table":
    layer = arcpy.mapping.TableView(desc.catalogPath)
    arcpy.mapping.AddTableView(df, layer)
    arcpy.AddMessage("\n" + layerName + " added to current map session.\n")
elif desc.dataType == "ArcInfoTable":
    layer = arcpy.mapping.TableView(desc.catalogPath + ".dbf")
    arcpy.mapping.AddTableView(df, layer)
    arcpy.AddMessage("\n" + layerName + " added to current map session.\n")

del desc, layerName, layer
return

```

```

##~~~~~
~~~~~
~~~~~##

```

```

if locDurations:
    type = "Local"
    durations = locDurations
    dm.CreateFolder(outLocation, type)
    outputPath = outLocation + "\\Local\\"
    arcpy.AddMessage("\nRunning PMP analysis for storm type: " + type)
    pmpAnalysis(basin, type, durations)      # Calls the pmpAnalysis() function to calculate the
local storm PMP
    arcpy.AddMessage("\nLocal storm analysis
complete...\n*****")

```

```

if genDurations:
    type = "General"
    durations = genDurations
    dm.CreateFolder(outLocation, type)
    outputPath = outLocation + "\\General\\"
    arcpy.AddMessage("\nRunning PMP analysis for storm type: " + type)
    pmpAnalysis(basin, type, durations)      # Calls the pmpAnalysis() function to calculate the
general storm PMP

```

```

    arcpy.AddMessage("\nGeneral storm analysis
complete...\n*****
*****")

```

```

if tropDurations:
    type = "Tropical"
    durations = tropDurations
    dm.CreateFolder(outLocation, type)
    outputPath = outLocation + "\\Tropical\\"
    arcpy.AddMessage("\nRunning PMP analysis for storm type: " + type)
    pmpAnalysis(basin, type, durations)      # Calls the pmpAnalysis() function to calculate the
    tropical storm PMP
    arcpy.AddMessage("\nTropical storm analysis
complete...\n*****
*****")

```

```

#if arcpy.Describe(outputTable).name:
#    outputBasAveTable()

```

```

#arcpy.RefreshTOC()
#arcpy.RefreshActiveView()
#del mxd, df

```

Appendix I

PMP Version Log: Changes to Storm Database and Adjustment Factors

New Jersey PMP version Log

Version 1.0 – 5/17/2020

- Previous transposition limits from the Pennsylvania statewide and several site-specific studies in the region (e.g. Catawba, Conowingo, Ashokan) were utilized as a starting point when storms were common to these other studies
- Initial GTF calculations for sensitivities. Set lower 0.5 and upper 1.5 limit
- Created 5 transposition zones
- Added transposition constraints to all storms
- MTF was set to 1 to remove from calculations.
- Used merged NOAA Atlas 14 Volume 2 and Volume 10 precipitation frequency datasets that were created for Pennsylvania Statewide study.

General Storms

- SPAS 1041_1 (Norwalk, CT) – Moved to zones 1,2,3,4
- SPAS 1047_1 (Tamaqua, PA) – Moved to zones 1,2,3,4,5
- SPAS 1201_1 (Halifax, VT) – Moved to zones 2,3,4,5
- SPAS 1255_1 (Pittsfield, MA) - Moved to zones 2,3,4,5
- SPAS 1259_1 (Slide Mountain, NY) - Moved to zones 3,4,5
- SPAS 1339_1 (Wellsboro, NC) - Moved to zones 1,2,3,4,5
- SPAS 1350_1 (New Bern, NC) - Moved to zones 1,2
- SPAS 1514_1 (Vade Mecum, NC) - Moved to zones 1,2
- SPAS 1533_1 (Montebello, VA) - Moved to zones 2,3,4,5
- SPAS 1565_1 (Paterson, NJ) - Moved to zones 1,2
- SPAS 1566_1 (Paterson, NJ) - Moved to zones 1,2

Hybrid Storms

- SPAS 1340_1 (Big Meadows, VA) - Moved to zones 3,4,5
- SPAS 1629_1 (Hector, NY) - Moved to zones 2,3,4,5

Local Storms

- SPAS 1017_1 (Sparta, NJ) - Moved to zones 2,3,4
- SPAS 1040_1 (Tabernacle, NJ) - Moved to zones 1,2
- SPAS 1049_1 (Delaware County, NY) - Moved to zones 2,3,4,5
- SPAS 1406_1 (Rapidan, VA) - Moved to zones 3,4,5
- SPAS 1415_1 (Islip, NY) - Moved to zones 1,2
- SPAS 1489_1 (Jewell, MD) - Moved to zones 1,2
- SPAS 1534_1 (Ewan, NJ) - Moved to zones 1,2
- SPAS 1547_1 (Catskill, NY) - Moved to zones 1,2,3,4,5
- SPAS 1700_1 (Elliot City, MD) - Moved to zones 1,2,3,4

Tropical Storms

- SPAS 1006_1 (West Shokan, NY) - Moved to zones 3,4,5
- SPAS 1224_1 (Maplecrest, NY) - Moved to zones 3,4,5
- SPAS 1243_1 (Westfield, MA) - Moved to zones 1,2,3,4,5
- SPAS 1275_2 (Montgomery Dam, PA) - Moved to zones 1,2,3,4,5
- SPAS 1276_2 (Zerbe, PA) - Moved to zones 1,2,3,4,5
- SPAS 1298_1 (Harrisburg, PA) - Moved to zones 1,2,3,4,5
- SPAS 1299_1 (Alta Pass, NC) - Moved to zones 3,4,5
- SPAS 1341_1 (Buck, CT) - Moved to zones 1,2
- SPAS 1490_1 (Easton, MD) - Moved to zones 1,2
- SPAS 1491_1 (Tyro, VA) - Moved to zones 3,4
- SPAS 1517_2 (Moncure, NC) - Moved to zones 1,2
- SPAS 1535_2 (Upper Sherando, VA) - Moved to zones 3,4,5
- SPAS 1551_1 (Richmond, VA) - Moved to zones 1,2
- SPAS 1552_1 (Southport 5N, NC) - Moved to zones 1,2
- SPAS 1552_2 (Yorktown, VA) - Moved to zones 1,2
- SPAS 1552_3 (Pomton Lake, NJ) - Moved to zones 1,2
- SPAS 1552_4 (Cairo, NY) - Moved to zones 1,2,3,4,5
- SPAS 1567_1 (Tuckerton, NJ) - Moved to zones 1,2
- SPAS 1669_1 (Evergreen, NC) - Moved to zones 1,2
- SPAS 1679_1 (Slide Mountain, NY) - Moved to zones 3,4,5
- SPAS 1720_1 (Wrightsville Beach, NC) - Moved to zones 1,2

Version 1a – 6/29/2020

Local Storms

- SPAS 1547_1 (Catskill, NY) - Lowered the upper GTF cap from 1.5 to 1.2. Set IPMF to 1.13

Tropical Storms

- SPAS 1491_1 (Tyro, VA) – Updated transposition limits to allow storm to go to all of zones 3,4,5.

Version 2 – 6/30/2020 - This version was not used

Used v1a with these changes

Removed storms that were questionable as to whether they should be used in this study.

General Storms

- SPAS 1259_1 (Slide Mountain, NY) – Removed from storm list (cool-season and too small for PMP)
- SPAS 1339_1 (Wellsboro, PA) – Removed from storm list (controlling storm, but smaller than tropical and local)

- SPAS 1533_1 (Montebello, VA) – Removed from storm list

Hybrid Storms

- SPAS 1340_1 (Big Meadows, VA) – Removed from storm list
- SPAS 1629_1 (Hector, NY) – Removed from storm list controlling storm, but smaller than tropical and local)

Local Storms

- SPAS 1049_1 (Delaware County, NY) – Removed from storm list

Tropical Storms

- SPAS 1299_1 (Alta Pass, NC) – Removed from storm list (controlling in zone 5)
- SPAS 1535_2 (Upper Sherando, VA) – Removed from storm list
- SPAS 1679_1 (Slide Mountain, NY) – Removed from storm list

Restrict trans limits for 1339, 1629, 1340, 1533 to PA zones only?

Compare 1a vs 2

1340 tropical or general only-check reanalysis/synoptics-remove from local

Notes:

V1

Tropical

6hr and 24hr-what to do with Tyro SPAS 1491? Move it into zone 5? Move it to piedmont and coast-if so adjust to limit increase? What did we do in VA?

72hr-move SPAS 1720 inland further?

V1A/V2

Tropical

6hr and 24hr-what to do with Tyro SPAS 1491? Move it into zone 5? Move it to piedmont and coast-if so adjust to limit increase? What did we do in VA? What did we do in Ashokan? What did we do in PA?

72hr-move SPAS 1720 inland further?

Need comparison against HMR 51 using the largest of each storm type

From July 9, 2020 teleconference: recommend holding SPAS 1339 and 1629 to within the PA overall study domain.

Version 3 – 9/21/2020 – This version was not used

Used v1a with these changes. Calculated MTF for sensitivity.

Version 4 – 10/5/2020

Used v1a with these changes.

General Storms

- SPAS 1339_1 – Updated Transposition limits to zones 1,2,3,4,5 but only in areas that overlap the PA statewide study
- SPAS 1533_1 - Updated Transposition limits to zones 2,3,4,5 but only in areas that overlap the PA statewide study

Hybrid Storms

- SPAS 1629_1 - Updated Transposition limits to zones 2,3,4,5 but only in areas that overlap the PA statewide study
- SPAS 1340_1 - Updated Transposition limits to zones 3,4,5 but only in areas that overlap the PA statewide study

Local Storms

- SPAS 1049_1 - Updated Transposition limits to zones 2,3,4,5 but only in areas that overlap the PA statewide study

Tropical Storms

- SPAS 1299_1 - Updated Transposition limits to zones 3,4,5 but only in areas that overlap the PA statewide study
- SPAS 1720_1 – Limited to south of 40°N

Discontinuity between 1720-does that mean we just haven't seen that type storm yet up north?

General storm discontinuity at 6hr-doesn't matter as filled in by others? How much time should we worry about those?

PMP maps of all types together?

Keep SPAS 1629 limits at v4?

Apply PA trans limits to SPAS 1491

Version 5 – 2/20/2021

Used v4 with these changes.

- Updated in place max dewpoint and SST values to the latest data available.
- Changed SPAS 1565_1 (Paterson, NJ) from General storm to Tropical storm.
- Smoothed precipitation frequency values in the far north to reduce unreasonably high PMP values. This matches what was done in the PA statewide study.

Tropical Storms

- SPAS 1006_1 – This storm was reanalyzed as SPAS 1680_1
- SPAS 1491_1 Tyro, VA – Allowed this storm to go to all zones. Previous version did not have it going to coastal areas.

General Storms

- Wellsboro, PA – This storm was split into 3 separate DAD zones. Storm rep SST was capped at 77. Average of 6 data points shown on May 31, 1889 SST map to account for uncertainty of inflow moisture region.
- SPAS 1629_1 Hector, NY – The general storm 100-year 24-hour precipitation frequency value was set to 7.13. This is an average of 3 stations. The local storm version still uses NOAA atlas 14 6 hr value.

Tropical

Tyro-Alta Pass-Florence

Local

Rapidan-Ewan

General

72hr 10sqmi Oct 2005

How to adjust/change trans limits

Get DD curves to see, include combination of all types

Don't need maps greater than total area size (5k, 10k, 20k)

Local storm boundary between Ewan and Rapidan GTF is reasonable, so what else can we do?

Only matters at 100sqmi and above-does 100sqmi and above matter for NJ? Move Ewan farther inland so only a small portion is affected by boundary?

Tropical-24hr 200sqmi boundary between 1299 and 1491, why 1491 to coast but 1299 limited?

Florence, no issues until 100sqmi and above at 72hr

Version 6 – 3/1/2021

Used v5 with these changes.

Local Storms

- SPAS 1534_1 Ewan, NJ – Updated transposition limits from just zones 1 & 2 to 1,2,3 & 4.
- Changed Big Meadows from G/L Hybrid to G/T

Tropical Storms

- SPAS 1720_1 Florence – Updated transposition from south of 40° N to zones 1 & 2
- SPAS 1299_1 Alta Pass, NC – Updated transposition from zones 3,4,5 to go to all zones and capped the GTF at .85 to reduce the values in the coastal areas.

Notes for v6

SPAS 1534_1 (Ewan) was only used in zones 1 & 2 in Pennsylvania. For NJ v6 we allowed it to also go up into zones 3 & 4. When comparing Local Storm PMP from PA to NJ, the PMP values are 20-25% higher in zones 3 & 4 because of this. SPAS 1406_1 is the controlling storm if 1534 is not allowed into zones 3 & 4. PMP is also larger in other overlap areas as the In-place max

factor have been updated for this study. SPAS 1534_1 Max SST values increased from 80 to 81.5°, thus increasing the IPMF by 8%. (1.21 to 1.29).

Version 7 – 6/10/2021

Used v6 with these changes.

Local Storms

- SPAS 1534_1 Ewan, NJ – This storm was producing large breaks in PMP values at the borders of the transposition limits and was determined to be too far inland allowing it to go to all of zones 3 & 4. Transposition limits were updated to go to all of New Jersey but starting at the boundary between zones 2 and 3 the GTF factor is reduced by 2.5% every 2 miles. The storm is no longer controlling at 10 miles past the boundary.

Tropical Storms

- SPAS 1720_1 Florence – Removed from storm list

Version 8 – 9/3/2021

Used v7 with these changes.

Local Storms

- SPAS 1818_1 Atlantic City, NJ – Added to storm list.

Version 9 – 12/29/2021

Used v8 with these changes. – **This is the final version**

Tropical Storms

- SPAS 1299_1 Alta Pass, NC – Updated maximization. This changes storm from dew point to SST and lowered the IPMF from 1.28 to 1.13
- Updated SPAS 1275_2 (Montgomery Dam, PA) from a tropical storm to a hybrid general/tropical storm
- Updated SPAS 1298_1 (Harrisburg, PA) from tropical to general storm
- Updated SPAS 1680_1 (West Shokan, NY) from tropical to general storm
- Added SPAS 1891_1 (Downington, PA) to tropical storm list

Appendix J

Precipitable Water Depths

J - 2

	0.0	0.0	60.5	61.0	61.5	62.0	62.5	63.0	63.5	64.0	64	65.0	65.5	66.0	66.5	67.0	67.5	68.0	68.5	69.0	69.5	70.0	70.5	71.0	71.5	72.0	72.5	73.0	73.5	74.0	74.5	75.0	75.5	76.0	76.5	77.0	77.5	78.0	78.5	79.0	79.5		
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		
100	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	
200	0.03	0.03	0.03	0.03	0.03	0.03	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	
300	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	
400	0.06	0.06	0.06	0.06	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.09	0.09	0.09	0.09	0.09	0.09	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.11	0.11	0.11	0.11	0.11	0.11	0.12	0.12	0.12	0.12
500	0.08	0.08	0.08	0.08	0.08	0.09	0.09	0.09	0.09	0.09	0.09	0.10	0.10	0.10	0.10	0.10	0.10	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.12	0.12	0.12	0.12	0.12	0.12	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.15
600	0.09	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.11	0.11	0.11	0.11	0.11	0.12	0.12	0.12	0.12	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.16	0.16	0.16	0.16	0.16	0.17	0.17	0.17	0.18	
700	0.11	0.11	0.11	0.12	0.12	0.12	0.12	0.12	0.12	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.15	0.15	0.15	0.16	0.16	0.16	0.16	0.17	0.17	0.17	0.18	0.18	0.18	0.18	0.18	0.19	0.19	0.19	0.19	0.20	0.20	0.20	0.21
800	0.12	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.15	0.16	0.16	0.16	0.16	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.18	0.18	0.18	0.18	0.19	0.19	0.20	0.20	0.20	0.21	0.21	0.21	0.22	0.22	0.22	0.23	0.23	0.24	
900	0.14	0.14	0.14	0.15	0.15	0.15	0.15	0.16	0.16	0.16	0.16	0.16	0.17	0.17	0.17	0.18	0.18	0.19	0.19	0.19	0.19	0.19	0.20	0.20	0.21	0.21	0.21	0.21	0.21	0.22	0.22	0.23	0.23	0.23	0.23	0.24	0.24	0.25	0.25	0.26	0.26	0.27	
1,000	0.15	0.16	0.16	0.16	0.16	0.17	0.17	0.17	0.17	0.18	0.18	0.19	0.19	0.19	0.20	0.20	0.21	0.21	0.21	0.21	0.21	0.21	0.22	0.22	0.23	0.23	0.23	0.23	0.24	0.24	0.25	0.25	0.26	0.26	0.27	0.27	0.28	0.28	0.28	0.28	0.29	0.29	
1,100	0.17	0.17	0.17	0.18	0.18	0.19	0.19	0.19	0.19	0.20	0.20	0.20	0.20	0.21	0.21	0.22	0.22	0.23	0.23	0.23	0.23	0.23	0.24	0.24	0.25	0.25	0.26	0.26	0.27	0.27	0.28	0.28	0.28	0.28	0.29	0.29	0.30	0.30	0.31	0.31	0.31	0.31	
1,200	0.18	0.19	0.19	0.19	0.19	0.20	0.20	0.21	0.21	0.22	0.22	0.22	0.22	0.23	0.23	0.24	0.24	0.25	0.25	0.25	0.25	0.26	0.26	0.27	0.27	0.28	0.28	0.29	0.29	0.30	0.31	0.31	0.32	0.32	0.33	0.33	0.34	0.34	0.35	0.34	0.34	0.35	
1,300	0.20	0.20	0.20	0.21	0.21	0.22	0.22	0.22	0.22	0.23	0.23	0.24	0.24	0.25	0.25	0.26	0.26	0.27	0.27	0.28	0.28	0.29	0.29	0.30	0.31	0.31	0.32	0.32	0.33	0.34	0.35	0.35	0.36	0.36	0.37	0.37	0.38	0.38	0.39	0.39	0.39	0.39	
1,400	0.21	0.22	0.22	0.22	0.22	0.23	0.23	0.24	0.24	0.25	0.25	0.26	0.26	0.27	0.27	0.28	0.28	0.28	0.28	0.29	0.29	0.30	0.30	0.31	0.31	0.32	0.32	0.33	0.34	0.34	0.35	0.36	0.37	0.37	0.38	0.38	0.39	0.39	0.40	0.40	0.40	0.40	
1,500	0.22	0.23	0.23	0.24	0.24	0.25	0.25	0.26	0.26	0.27	0.27	0.28	0.28	0.29	0.29	0.30	0.31	0.31	0.32	0.32	0.33	0.33	0.34	0.35	0.35	0.36	0.36	0.37	0.38	0.39	0.39	0.40	0.41	0.42	0.43	0.43	0.44	0.44	0.44	0.44	0.44	0.44	
1,600	0.24	0.25	0.25	0.26	0.26	0.26	0.26	0.27	0.28	0.28	0.29	0.29	0.30	0.30	0.31	0.31	0.32	0.32	0.33	0.33	0.34	0.35	0.35	0.36	0.36	0.37	0.37	0.38	0.39	0.39	0.40	0.41	0.42	0.43	0.43	0.44	0.44	0.45	0.45	0.46	0.46		
1,700	0.25	0.26	0.26	0.27	0.27	0.28	0.28	0.29	0.29	0.30	0.30	0.31	0.31	0.32	0.32	0.33	0.33	0.34	0.35	0.35	0.36	0.37	0.37	0.38	0.39	0.39	0.40	0.40	0.41	0.42	0.43	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.48	0.49	0.49		
1,800	0.26	0.27	0.27	0.28	0.28	0.29	0.29	0.30	0.30	0.31	0.32	0.33	0.33	0.34	0.34	0.35	0.35	0.36	0.36	0.37	0.37	0.38	0.39	0.40	0.40	0.41	0.41	0.42	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.48	0.49	0.50	0.51	0.51	0.51		
1,900	0.28	0.29	0.29	0.30	0.30	0.31	0.31	0.32	0.32	0.33	0.33	0.34	0.34	0.35	0.35	0.36	0.37	0.38	0.38	0.39	0.39	0.40	0.41	0.42	0.42	0.43	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.54	0.54		
2,000	0.29	0.30	0.30	0.31	0.31	0.32	0.32	0.33	0.34	0.35	0.35	0.36	0.36	0.37	0.37	0.38	0.38	0.39	0.40	0.41	0.42	0.42	0.43	0.44	0.44	0.45	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.55	0.55	0.55	0.56		
2,100	0.30	0.31	0.32	0.33	0.33	0.34	0.34	0.35	0.35	0.36	0.36	0.37	0.38	0.39	0.39	0.40	0.40	0.41	0.42	0.43	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.59	0.59	0.60		
2,200	0.32	0.33	0.33	0.34	0.34	0.35	0.35	0.36	0.37	0.38	0.38	0.39	0.39	0.40	0.41	0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67		
2,300	0.33	0.34	0.34	0.35	0.36	0.37	0.37	0.38	0.38	0.39	0.39	0.40	0.40	0.41	0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68		
2,400	0.34	0.35	0.36	0.37	0.37	0.38	0.38	0.39	0.40	0.41	0.42	0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.70	0.71		
2,500	0.36	0.37	0.37	0.38	0.38	0.39	0.40	0.41	0.42	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.70	0.72	0.73	0.74		
2,600	0.37	0.38	0.38	0.39	0.40	0.41	0.41	0.42	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.70	0.72	0.73	0.74			
2,700	0.38	0.39	0.40	0.41	0.41	0.42	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.70	0.72	0.73	0.74					
2,800	0.39	0.40	0.41	0.42	0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.70	0.72	0.73	0.74	0.75	0.77	0.78	0.79			
2,900	0.41	0.42	0.43	0.44	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.70	0.72	0.73	0.74	0.75	0.77	0.78	0.79	0.80				
3,000	0.42	0.43	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.70	0.72	0.73	0.74	0.75	0.77	0.78	0.79	0.80	0.82				
3,100	0.43	0.44	0.45	0.46	0.46	0.47	0.48																																				

Elev. FT	80.0	80.5	81.0	81.5	82.0	82.5	83.0	83.5	84.0	84.5	85.0	85.5	86.0	86.5	87.0	87.5	88.0	88.5	89.0	89.5	90.0
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
200	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
300	0.09	0.09	0.09	0.09	0.09	0.09	0.10	0.10	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
400	0.12	0.12	0.12	0.12	0.12	0.12	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
500	0.15	0.15	0.15	0.15	0.15	0.15	0.16	0.17	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.19	0.19
600	0.18	0.18	0.18	0.18	0.18	0.18	0.19	0.20	0.21	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22
700	0.21	0.21	0.21	0.21	0.21	0.21	0.22	0.23	0.24	0.25	0.25	0.25	0.25	0.25	0.26	0.26	0.26	0.26	0.26	0.26	0.27
800	0.24	0.24	0.24	0.24	0.24	0.24	0.25	0.26	0.27	0.28	0.28	0.29	0.29	0.29	0.29	0.30	0.30	0.30	0.31	0.31	0.31
900	0.27	0.27	0.27	0.27	0.27	0.27	0.28	0.29	0.30	0.31	0.32	0.32	0.32	0.33	0.33	0.34	0.34	0.35	0.35	0.36	0.36
1,000	0.29	0.30	0.30	0.30	0.30	0.30	0.31	0.32	0.33	0.34	0.35	0.35	0.36	0.37	0.37	0.38	0.39	0.39	0.40	0.40	0.41
1,100	0.32	0.33	0.33	0.33	0.33	0.33	0.34	0.35	0.36	0.37	0.38	0.39	0.40	0.40	0.41	0.42	0.43	0.43	0.44	0.45	0.46
1,200	0.35	0.36	0.36	0.36	0.36	0.36	0.37	0.38	0.39	0.40	0.41	0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51
1,300	0.38	0.39	0.39	0.39	0.39	0.39	0.40	0.41	0.42	0.44	0.45	0.46	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55
1,400	0.40	0.42	0.42	0.42	0.42	0.42	0.43	0.44	0.45	0.47	0.48	0.49	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.58	0.59
1,500	0.43	0.45	0.45	0.45	0.45	0.45	0.46	0.47	0.48	0.50	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.61	0.62
1,600	0.46	0.48	0.48	0.48	0.48	0.48	0.49	0.50	0.51	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.64	0.65
1,700	0.48	0.50	0.51	0.51	0.51	0.51	0.52	0.53	0.54	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.67	0.68
1,800	0.52	0.53	0.54	0.54	0.54	0.54	0.55	0.56	0.57	0.59	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.70	0.71
1,900	0.54	0.55	0.57	0.57	0.57	0.57	0.58	0.59	0.60	0.62	0.63	0.64	0.65	0.66	0.66	0.67	0.68	0.69	0.70	0.71	0.71
2,000	0.57	0.58	0.60	0.60	0.60	0.60	0.61	0.62	0.63	0.65	0.66	0.67	0.67	0.68	0.68	0.68	0.69	0.70	0.71	0.72	0.72
2,100	0.60	0.61	0.62	0.63	0.63	0.63	0.64	0.65	0.66	0.68	0.69	0.70	0.70	0.70	0.71	0.71	0.72	0.72	0.73	0.73	0.73
2,200	0.62	0.64	0.65	0.65	0.66	0.66	0.67	0.68	0.69	0.71	0.72	0.73	0.73	0.73	0.73	0.74	0.74	0.75	0.75	0.76	0.76
2,300	0.65	0.67	0.67	0.68	0.69	0.69	0.70	0.71	0.72	0.74	0.75	0.76	0.76	0.76	0.76	0.77	0.77	0.78	0.78	0.79	0.79
2,400	0.68	0.69	0.70	0.71	0.71	0.72	0.73	0.74	0.75	0.77	0.78	0.79	0.79	0.79	0.79	0.80	0.80	0.81	0.81	0.82	0.82
2,500	0.70	0.72	0.73	0.73	0.74	0.75	0.76	0.77	0.78	0.80	0.81	0.82	0.82	0.82	0.82	0.83	0.83	0.84	0.84	0.85	0.85
2,600	0.73	0.74	0.75	0.76	0.77	0.78	0.79	0.80	0.81	0.83	0.84	0.85	0.85	0.85	0.85	0.86	0.86	0.87	0.87	0.88	0.88
2,700	0.75	0.76	0.78	0.79	0.80	0.81	0.82	0.83	0.84	0.86	0.87	0.88	0.89	0.90	0.91	0.92	0.92	0.94	0.94	0.95	0.96
2,800	0.78	0.79	0.80	0.82	0.83	0.84	0.85	0.86	0.87	0.89	0.90	0.92	0.93	0.94	0.96	0.97	0.98	1.00	1.01	1.02	1.04
2,900	0.80	0.81	0.83	0.84	0.86	0.87	0.88	0.89	0.90	0.92	0.93	0.95	0.97	0.98	1.00	1.01	1.03	1.04	1.06	1.07	1.09
3,000	0.83	0.84	0.85	0.87	0.89	0.90	0.91	0.92	0.93	0.95	0.97	0.98	1.00	1.02	1.03	1.05	1.07	1.08	1.10	1.12	1.13
3,100	0.85	0.86	0.88	0.90	0.91	0.93	0.94	0.95	0.96	0.98	1.00	1.02	1.04	1.05	1.07	1.09	1.11	1.13	1.15	1.16	1.18
3,200	0.88	0.88	0.90	0.92	0.94	0.96	0.97	0.98	0.99	1.01	1.03	1.05	1.07	1.09	1.11	1.13	1.15	1.17	1.19	1.21	1.23
3,300	0.90	0.91	0.93	0.95	0.97	0.99	1.00	1.01	1.02	1.04	1.06	1.09	1.11	1.13	1.15	1.17	1.19	1.21	1.23	1.25	1.27
3,400	0.93	0.93	0.95	0.97	1.00	1.02	1.03	1.04	1.05	1.07	1.09	1.11	1.13	1.15	1.17	1.19	1.21	1.23	1.25	1.27	1.29
3,500	0.95	0.96	0.98	1.00	1.02	1.04	1.05	1.07	1.08	1.10	1.12	1.14	1.15	1.17	1.19	1.21	1.23	1.24	1.26	1.28	1.30
3,600	0.97	0.98	1.00	1.02	1.04	1.06	1.08	1.10	1.11	1.13	1.15	1.16	1.18	1.20	1.21	1.23	1.25	1.26	1.28	1.29	1.31
3,700	1.00	1.00	1.02	1.05	1.07	1.09	1.11	1.12	1.14	1.16	1.17	1.19	1.20	1.22	1.23	1.25	1.26	1.28	1.29	1.31	1.32
3,800	1.02	1.03	1.05	1.07	1.09	1.11	1.13	1.15	1.17	1.19	1.20	1.21	1.23	1.24	1.25	1.27	1.28	1.30	1.31	1.32	1.34
3,900	1.05	1.05	1.07	1.09	1.12	1.14	1.16	1.18	1.20	1.22	1.23	1.24	1.25	1.26	1.28	1.29	1.30	1.31	1.33	1.34	1.35
4,000	1.07	1.08	1.10	1.12	1.14	1.16	1.18	1.20	1.23	1.24	1.25	1.27	1.28	1.29	1.31	1.32	1.33	1.34	1.36	1.37	1.38
4,100	1.09	1.10	1.12	1.14	1.16	1.19	1.21	1.23	1.25	1.27	1.28	1.30	1.31	1.32	1.34	1.35	1.37	1.38	1.40	1.41	1.42
4,200	1.12	1.12	1.14	1.17	1.19	1.21	1.23	1.25	1.27	1.29	1.31	1.32	1.34	1.36	1.37	1.39	1.40	1.42	1.43	1.45	1.47
4,300	1.14	1.15	1.17	1.19	1.21	1.23	1.26	1.28	1.30	1.32	1.33	1.35	1.37	1.39	1.40	1.42	1.44	1.46	1.47	1.49	1.51
4,400	1.16	1.17	1.19	1.21	1.24	1.26	1.28	1.30	1.32	1.34	1.36	1.38	1.40	1.42	1.44	1.46	1.47	1.49	1.51	1.53	1.55
4,500	1.18	1.19	1.22	1.24	1.26	1.28	1.30	1.32	1.35	1.37	1.39	1.41	1.43	1.45	1.47	1.49	1.51	1.53	1.55	1.57	1.59
4,600	1.21	1.22	1.24	1.26	1.28	1.31	1.33	1.35	1.37	1.39	1.41	1.44	1.46	1.48	1.50	1.52	1.55	1.57	1.59	1.61	1.63
4,700	1.23	1.24	1.26	1.29	1.31	1.33	1.35	1.37	1.39	1.42	1.44	1.46	1.48	1.50	1.53	1.55	1.57	1.59	1.61	1.64	1.66
4,800	1.25	1.27	1.29	1.31	1.33	1.35	1.38	1.40	1.42	1.44	1.46	1.48	1.51	1.53	1.55	1.57	1.59	1.62	1.64	1.66	1.68
4,900	1.27	1.29	1.31	1.33	1.36	1.38	1.40	1.42	1.44	1.46	1.49	1.51	1.53	1.55	1.57	1.60	1.62	1.64	1.66	1.68	1.71
5,000	1.29	1.31	1.34	1.36	1.38	1.40	1.42	1.44	1.47	1.49	1.51	1.53	1.55	1.58	1.60	1.62	1.64	1.66	1.69	1.71	1.73
5,500	1.40	1.43	1.46	1.48	1.50	1.52	1.54	1.56	1.59	1.61	1.64	1.66	1.69	1.71	1.74	1.77	1.79	1.82	1.84	1.87	1.89
6,000	1.50	1.55	1.57	1.59	1.61	1.64	1.66	1.68	1.71	1.74	1.77	1.80	1.83	1.87	1.90	1.93	1.96	1.99	2.03	2.06	2.09
6,500	1.60	1.64	1.66	1.68	1.70	1.73	1.76	1.79	1.82	1.86	1.89	1.92	1.95	1.99	2.02	2.05	2.08	2.12	2.15	2.18	2.22
7,000	1.70	1.73	1.76	1.78	1.81	1.84	1.88	1.91	1.94	1.98	2.01	2.04	2.07	2.11	2.14	2.17	2.20	2.24	2.27	2.30	2.34
7,500	1.79	1.82	1.85	1.88	1.91	1.95	1.98	2.01	2.05	2.08	2.11	2.14	2.18	2.21	2.24	2.28	2.31	2.34	2.37	2.41	2.44
8,000	1.88	1.91	1.94	1.98	2.01	2.04	2.08	2.11	2.14	2.18	2.21	2.24	2.27	2.31	2.34	2.37	2.41	2.44	2.47	2.50	2.54
8,500	1.97	2.00	2.04	2.08	2.12	2.16	2.20	2.23	2.26	2.30	2.33	2.36	2.39	2.43	2.46	2.49	2.53	2.56	2.59	2.62	2.66
9,000	2.05	2.09	2.13	2.17	2.21	2.26	2.29	2.32	2.35	2.39	2.43	2.47	2.51	2.55	2.59	2.63	2.67	2.71	2.75	2.79	2.83
9,500	2.13	2.17	2.22	2.26	2.30	2.35	2.38	2.41	2.44	2.48	2.53	2.57	2.62	2.66							