HDR | ONE COMPANY *Many Solutions*℠

Memo

To:	Jesse Bradley, Nebraska DNR Integrated Water Management Division Head Brandi Flyr, Nebraska DNR Integrated Water Management Coordinator
From:	HDR Project Team
Project:	Depletion Estimates for the Lower Platte River Basin
CC:	
Date:	December 2013
Job No:	

I. Introduction

HDR provided technical support to the Department by computing surface water depletions from groundwater irrigation pumping within the hydrologically connected area for the Lower Platte River basin. This memo is intended to provide a brief description of raw and prepared datasets used in the analyses, a description of the GIS Python programming script used in the preparation of depletion estimates, and post-processing of results.

II. Raw Datasets

The following raw datasets were utilized in this analysis:

National Hydrography Datasets (NHD)

The NHD datasets for the Lower Platte River basin area was obtained from http://nhd.usgs.gov. Subset of NHDFlowline dataset queried; FTYPE= StreamRiver AND FCODE = 46006 OR FCODE = 46007 OR FCODE=55800.

During the analysis, it was discovered that portions of the main stem streams of the Platte River, Elkhorn River, and Maple Creek were listed in the NHD dataset as 'artificial path' rather than 'perennial stream'. Thus, the NHD subset used in this analysis was expanded to include both 'perennial stream' and 'artificial path'.

Transmissivity Data

The basin hydrographic datasets from the 2013 HDR report, "Hydrogeologic Assessment for Potential Development of Groundwater Modeling Tools in the Lower Platte River and Missouri River Tributary Basins" were sourced for data. This report is available from the DNR website at the following address: http://www.dnr.state.ne.us/Publications_Studies/LowerPlatte_MoTribsAssessment.pdf . Supporting electronic files are available upon request from the Department.

Specific Yield Data

The Specific Yield (SY) dataset was provided by Les Howard of University of Nebraska-Lincoln (UNL) Conservation and Survey Division in the form of a vector line GIS shapefile. The raw dataset provided SY values in the form of % values.

Principal Aquifer/Hydrologically Connected Area

The principal aquifer definition was used to approximate the hydrologically connected area. The definition of the principal aquifer was obtained from the Department and was based upon the 2005 Conservation and Survey Division's *Mapping of Aquifer Properties-Transmissivity and Specific Yield- for Selected River Basins in Central and Eastern Nebraska*.

III. Prepared Datasets

Historic Pumping Estimates

Historic pumping estimates were provided in the form of a grid file. The computational grid file used in this analysis is common to that used in the statewide land use coverage and both the computational grid file and historic pumping estimates are available upon request from the Department. Each grid cell has a unique cell ID number. Text files containing pumping estimates by year (from 1950 to 2012) by cell ID number were provided with the grid file. These text files were joined to the shapefile in GIS.

Development of Grid Cell Centroid Shapefile

Points were assigned to represent each grid cell and were placed at the center of each grid using the Feature to Point tool. This is referred to as the grid cell centroid shapefile and provides a single spatial reference point for the grid cell for use in the computations.

```
arcpy.FeatureToPoint_management("Grid Index Layer", "output feature class", "CENTROID")
```

Specific Yield, Transmissivity, and Near Distance

Attributes including SY, Trans, and NEAR_DIST were added to the grid cell centroid shapefile.

- A conversion was included in the script to convert the specific yield values to decimal format for use in the depletion calculations. In order to assign the specific yield value to the representative grid cell, a spatial join was performed that connected the value from the specific yield dataset described above to the grid centroid shapefile.

- Transmissivity data from the HDR Transmissivity raster were added to the grid cell centroid well file using the Extract Values to Points tool that requires the Spatial Analyst extension to run.

```
arcpy.gp.ExtractValuesToPoints_sa("hypothetical wells", "Transmissivity raster", "output
feature class NONE","VALUE_ONLY")
```

- Near distance added using the perennial stream data sourced from the NHD database using the Near Analyst tool. This tool requires ArcGIS for Desktop Advanced to run.

```
arcpy.Near_analysis("Hypothetical Wells", "perennial streams", "#","NO_LOCATION","NO_ANGLE")
```

IV. Python Script Description

A Python script file was written to perform the computations with an annotated description of the script provided below. The full Python script is included in Section VI of this document.

Annotated Script Overview

1. The python script extracts the parameters from the historic pumping estimate file.

```
inFC = arcpy.GetParameterAsText(0)
```

2. It then adds the following fields to the attribute table (feature class) which will be populated during the run:

```
try:
        arcpy.AddField_management(inFC, "SDF", "Double", "20", "8")
        arcpy.AddField_management(inFC, "Dimen", "Double", "20", "8")
        arcpy.AddField_management(inFC, "yrOnline", "Double", "20", "8")
        arcpy.AddField_management(inFC, "yrOffline", "Double", "20", "8")
        arcpy.AddField_management(inFC, "sumPump", "Double", "20", "8")
        arcpy.AddField_management(inFC, "CumDepTotal", "Double", "20", "8")
```

3. The script ensures there are no null values in the attribute table.

```
arcpy.CalculateField_management(inFC, "SDF", 0)
```

4. The script is set up to display an error message if there is an error when adding any of the above fields.

```
except:
        print arcpy.GetMessages(2)
```

5. The script places all grid rows into an update cursor.

```
cur = arcpy.UpdateCursor(inFC)
```

6. The script grabs the first row in the dataset.

```
row = cur.next()
```

7. At the beginning of each loop, the script sets pumping, online and offline values to 0 so that previous values are removed.

```
while row:
        PumpTotal = 0
        online = 0
        offline = 0
```

The script is set up to calculate values to “"SDF", "CumPump" and "CumDep" fields based on the Jenkins Method.

8. The script looks at values for Specific Yield. If any values are less than 0, it will replace these with the value of ‘0.15’.

```
if row.SY < 0:
        row.SY = .15
```

9. The script looks at values for Transmissivity. If any values are less than 0, it will replace those with ‘5000’.

```
if row.Trans <=0:
        row.Trans = 5000
```

SDF

10. The SDF field is calculated using the following equation.

SDF = (“*Distance Nearest Perennial Stream”*)^2 * *“specific yield”/ “Transmissivity”*

```
if row.SDF == 0:
        SDF = pow(row.Near_Dist,2) * row.SY / (row.Trans)
else:
        SDF = row.SDF
```

11. The script is set up to run for any time interval specified by the user. For the purpose of this analysis, the script was run on an annual basis. The script will select the first non-zero year as the “online” year and will select the user entered end year (entered as one year after the desired end date) as the “offline” year.

```
for y in range (Begin Year, End Year + 1):
```

12. The script will sum the pumping values for the time range for each row during this loop.

```
CumPumpA = "Ctot" + str(y)
if row.getValue(CumPumpA) is None:
        continue
        else:
                if online is 0:
                        online = y
                        offline = y
                else:
                        offline = y
```

Dimen (dimenVal)

13. To find the Dimen value, the script evaluates the range for each cell and the associated time period is defined as the "offline" year less the "online" year times 365 to convert to days.

```
Time = (offline - online) * 365
dimenVal = Time / SDF
```

This value for each cell is then divided by its associated SDF (explained above). If this value is less than .07 then it is set to be '0', if it is greater than 600 it is set to be '0.96'.

Fraction (delVal)

14. The Dimen value is then compared to values in the Jenkins table (defined as "key list" in the Python script and shown below) to find the correct values to use for the linear interpolation.

```
if dimenVal < 0.07:
        delVal = 0
elif dimenVal > 600:
        delVal = 0.96
else:
        xold = 0.07
        for x1 in keylist:
                if x1 >= dimenVal:
                        x2 = x1
                        x1 = xold
                        break
                xold = x1
        y1 = myDict[x1]
        y2 = myDict[x2]
 # Linear interpolation
        delVal = y1 + (dimenVal - x1)* (y2 - y1)/(x2-x1)
```

15. The values from Jenkins (Table 1) are passed into the linear interpolation process

```
myDict =
{0.07:0.001,0.10:0.006,0.15:0.019,0.20:0.037,0.25:0.057,0.30:0.077,0.35:0.097,0.40:0.115,0.45:0.134,
0.50:0.151,0.55:0.167, 0.60:0.182, 0.65:0.197, 0.70:0.211, 0.75:0.224, 0.80:0.236, 0.85:0.248,
0.90:0.259, 0.95:0.27, 1:0.28, 1.1:0.299, 1.2:0.316, 1.3:0.333, 1.4:0.348, 1.5:0.362, 1.6:0.375,
1.7:0.387, 1.8:0.398, 1.9:0.409, 2:0.419, 2.2:0.438, 2.4:0.455, 2.6:0.470, 2.8:0.484, 3:0.497,
3.5:0.525, 4:0.549, 4.5:0.569, 5:0.587, 5.5:0.603, 6:0.616, 7:0.64, 8:0.659, 9:0.676, 10:0.69,
15:0.74, 20:0.772, 30:0.81, 50:0.85, 100:0.892,600:0.955}

keylist = myDict.keys()
keylist.sort()
```

CumPump

16. The script searches the attribute table of the grid cell centroid well files for the first non-zero pumping value for each cell ID (row). It assigns the year associated with this first non-zero value as the "online" year. The user inputs an end year for the range the script is analyzing (user enters Year +1). The script assigns this end of range as the "offline" year.

```
for x in range (Start Year, End Year + 1):
        CumPump = "Ctot" + str(x)
        if row.getValue(CumPump) is None:
                continue
        else:
                PumpTotal = row.getValue(CumPump) + PumpTotal
```

17. The script then writes the results of the calculation into the attribute table.

```
row.SDF = SDF
# Jenkins dimensionless variable
row.Dimen = dimenVal
row.sumPump = PumpTotal
row.yrOnline = online
row.yrOffline = offline
```

CumDep

18. The cumulative depletion value is determined by taking the CumPump value times the computed delVal value. The script writes this value to the attribute table.

```
row.CumDepTotal = PumpTotal * delVal
```

19. The script moves on to the next row in the feature class and repeats the process.

```
cur.updateRow(row)
row = cur.next()
```

V. Post-Processing of Depletion Estimates

Clip to Hydrologically Connected Area and Trim to Subbasin

Once the estimated depletion were computed for each grid cell, the grid shapefile was then clipped to the hydrologically connected area and further trimmed to the North Bend, Ashland and Louisville subbasins. The definition of these subbasins is common to those used in the Fully Appropriated Evaluation Methodology Development analysis conducted by HDR and is available upon request from the Department. An intersect tool was used to allocate each grid to its respective subbasin post analysis.

Temporal Distribution

The final step was partitioning the estimated total annual depletions between peak and non-peak seasons. Typical monthly depletion curves were developed for a set of grid cells of varying distances from the nearest stream source and the temporal distribution reviewed. Based the review of these temporal distributions, the estimated total annual depletions were split evenly between peak and non-peak seasons.

VI. Full Python Script

```
# CalculateDepletions.py
# Author: Jesse Bradley and Shuhai Zheng
# Modified: Amy Sorensen, HDR
# Modified Date: October 2013
# Version: ArcGIS 10.1 arcpy used in updates
# Purpose: Calculate well pumping depletion on river flows using the Jenkins method.
# Customize Script for Task Order #2, Task 230, Fully Appropriated Basin (FAB) Evaluation
# Input Transmissivity need to be in square feet/day
# Inputs: grid cell centroid shapefile
# Results: Updated fields SDF(a**2 S/T), DIMEN (t/SDF), Fraction, Depletion.
#-------------------------------------------------------------------------------

# Import system modules
import arcpy

# Set working space
arcpy.env.workspace = "C:\\TEMP"

# Allow overwrite an existing file
arcpy.env.overwriteOutput = True

# Input wells with the parameters assigned.
inFC = arcpy.GetParameterAsText(0)

# Add fields into the input feature class
try:
    # Add necessary fields
        arcpy.AddField_management(inFC, "SDF", "Double", "20", "8")
        arcpy.AddField_management(inFC, "Dimen", "Double", "20", "8")
```

```
        arcpy.AddField_management(inFC, "yrOnline", "Double", "20", "8")
        arcpy.AddField_management(inFC, "yrOffline", "Double", "20", "8")
        arcpy.AddField_management(inFC, "sumPump", "Double", "20", "8")
        arcpy.AddField_management(inFC, "CumDepTotal", "Double", "20", "8")

        # Calculates so values aren't NULL so SDF math runs later in script
        arcpy.CalculateField_management(inFC, "SDF", 0)


except:
        # If an error occurs when running Addfield, print out the error message.
    print arcpy.GetMessages(2)

################################################################################
# Place all well rows into an update cursor
cur = arcpy.UpdateCursor(inFC)

# Get the first row
row = cur.next()

# the values from Jenkins (Table 1) which are passed into the linear interpolation process
myDict =
{0.07:0.001,0.10:0.006,0.15:0.019,0.20:0.037,0.25:0.057,0.30:0.077,0.35:0.097,0.40:0.115,0.45:
0.134,0.50:0.151,0.55:0.167,

0.60:0.182,0.65:0.197,0.70:0.211,0.75:0.224,0.80:0.236,0.85:0.248,0.90:0.259,0.95:0.27,1:0.28,1.1:0.
299,1.2:0.316,1.3:0.333,

1.4:0.348,1.5:0.362,1.6:0.375,1.7:0.387,1.8:0.398,1.9:0.409,2:0.419,2.2:0.438,2.4:0.455,2.6:0.470,2.
8:0.484,3:0.497,

3.5:0.525,4:0.549,4.5:0.569,5:0.587,5.5:0.603,6:0.616,7:0.64,8:0.659,9:0.676,10:0.69,15:0.74,20:0.77
2,30:0.81,50:0.85,100:0.892,600:0.955}

keylist = myDict.keys()
keylist.sort()


# Loop through each row in the input feature class and calculate values to "SDF", "CumPump" and
"CumDep" fields.
while row:
        PumpTotal = 0
        online = 0
        offline = 0

        if row.SY < 0:
                row.SY = .15

        if row.Trans <=0:
                row.Trans = 5000

        if row.SDF == 0:
                SDF = pow(row.Near_Dist,2) * row.SY / (row.Trans)
        else:
                SDF = row.SDF

        # Determine year well went online and year well went offline. For the end year, add +1 to #
the year you'd like the script to end in
        for y in range (1950, 2012):
                CumPumpA = "Ctot" + str(y)
                if row.getValue(CumPumpA) is None:
                        continue
                else:
                        if online is 0:
                                online = y
                                offline = y
                        else:
                                offline = y

        Time = (offline - online) * 365
```

```
        dimenVal = Time / SDF

        if dimenVal < 0.07:
                delVal = 0
        elif dimenVal > 600:
                delVal = 0.96
        else:
                xold = 0.07
                for x1 in keylist:
                        if x1 >= dimenVal:
                                x2 = x1
                                x1 = xold
                                break
                        xold = x1
                y1 = myDict[x1]
                y2 = myDict[x2]
         # Linear interpolation
                delVal = y1 + (dimenVal - x1)* (y2 - y1)/(x2-x1)

        # This loop will sum up the pumping values for the calculation for the period selected
        for x in range (1950, 2012):
                CumPump = "Ctot" + str(x)
                if row.getValue(CumPump) is None:
                        continue
                else:
                        PumpTotal = row.getValue(CumPump) + PumpTotal


    # Write Values to fields
        row.SDF = SDF
        # Jenkins dimensionless variable
        row.Dimen = dimenVal
        row.sumPump = PumpTotal
        row.yrOnline = online
        row.yrOffline = offline

        #Calculate Depletion Total and write to field
        row.CumDepTotal = PumpTotal * delVal

    #Update the row and move on to next well.
        cur.updateRow(row)
        row = cur.next()

# Delete variables
del arcpy, cur, row, dimenVal, delVal, PumpTotal, online, offline

print "FINISHED"
```