Interpolation Methods

This notebook will compare 4 Interpolation methods in Arc Pro. First, I will use code from Jeffey Bishop to extract the data. Then, I will reshape the table and export it to points. Finally, I will perform inverse distance weighting and kriging to compare the results.

In [1]:

```
import pandas as pd
import requests

from datetime import date
from io import StringIO

class ndawn_request:

    def __init__(self, startDate='YYYY-MM-DD', endDate='YYYY-MM-DD', ontology = None, location = None, save = False):

        self.start = startDate

        self.end = endDate

        # List of ontology terms, and their URL codes to build request URL
        self.ontology = {
            'Air Temperature': ['variable=hdt', 'variable=hdt9'],
            'Relative Humidity': ['variable=hdrh', 'variable=hdrh9'],
            'Soil Temperature': ['variable=hdbst', 'variable=hdtst'],
            'Wind Speed': ['variable=hdws', 'variable=hdmxws', 'variable=hdws10', 'variable=hdmxws10'],
            'Wind Direction': ['variable=hdwd', 'variable=hdsdwd', '&variable=hdwd10', 'variable=hdsdwd10'],
            'Solar Radiation': ['variable=hdsr'],
            'Rainfall': ['variable=hdr'],
            'Air Pressure': ['variable=hdbp'],
            'Dew Point': ['variable=hddp'],
            'Wind Chill': ['variable=hdwc']
        }
        # Concatenate the ontology keys into a list for exception print out later
        ontologiesErrorMessage = '\n'.join(list(self.ontology.keys()))

        # List of stations, and URL codes to build request URL
        self.stations = {
            'Ada': 78,
            'Becker': 118,
            'Campbell': 87,
            'Clarissa': 124,
            'Eldred': 2,
            'Fox': 93,
            'Greenbush': 70,
            'Hubbard': 119,
            'Humboldt': 4,
            'Kennedy': 82,
            'Little Falls': 120,
            'Mavie': 71,
            'Ottertail': 103,
            'Parkers Prairie': 116,
            'Perham': 114,
            'Perley': 3,
            'Pine Point': 115,
            'Rice': 121,
            'Roseau': 61,
            'Sabin': 60,
            'Staples': 122,
            'Stephen': 5,
            'Ulen': 91,
            'Wadena': 117,
            'Warren': 6,
            'Waukon': 92,
            'Westport': 123,
            'Williams': 95
        }
        # Concatenate station names into a list for exception printout later
        stationsErrorMessage = '\n'.join(list(self.stations.keys()))

        self.save = save

        # This checks the start and end dates supplied to make sure they are valid
        # Start by converting dates into iso format
        startDateCheck = date.fromisoformat(startDate)
        endDateCheck = date.fromisoformat(endDate)
        # If start date is after end date, raise exception
        if startDateCheck > endDateCheck:
            raise Exception('End date cannot be before start date')

        # Create empty list to hold URL codes for ontology terms
        self.activeMeasures = []
        # If user supplies ontology terms
        if ontology is not None:
            for item in ontology:
                # If user-supplied term is not in the dictionary, raise exception
                if item not in self.ontology.keys():
                    raise Exception('Ontology term [' + str(item) + '] not recognized. Available ontology terms include: ' + '\n' + ontologiesErrorMessage)
                # Otherwise, append URL codes for ontology terms into the list of measurements to be requested
                else:
                    for code in self.ontology[item]:
                        self.activeMeasures.append(code)
        # If user does not supply ontology terms, add all URL codes in dictionary to the list of measurements to be requested
        else:
            for key in self.ontology:
                for code in self.ontology[key]:
                    self.activeMeasures.append(code)

        # Create empty list to hold URL codes for stations
        self.activeStations = []
        # If user supplies station names
        if location is not None:
            for name in location:
                # If user-supplied name is not in the dictionary, raise exception
                if name not in self.stations.keys():
                    raise Exception('Station [' + str(name) + '] not recognized. Available stations include: ' + '\n' + stationsErrorMessage)
                # Otherwise, append URL codes for stations into the list of stations to be requested
                else:
                    self.activeStations.append('station=' + str(self.stations[name]))
        # If user does not supply station names, add all station URL codes in dictionary to the list of stations to be requested
        else:
            for key in self.stations:
                self.activeStations.append('station=' + str(self.stations[key]))

    def get_data(self):

        # Construct API call for the request
        baseURL = 'https://ndawn.ndsu.nodak.edu/table.csv?'
        stations = '&'.join(self.activeStations)
        measurements = '&'.join(self.activeMeasures)
        options = '&ttype=hourly&quick_pick=&begin_date=' + self.start + '&end_date=' + self.end
        finalURL = str(baseURL + stations + '&' + measurements + options)

        # Request page
        page = requests.get(finalURL)
        # If status code not 200, raise exception
        if page.status_code != 200:
            raise Exception('URL request status not 200. Status code = ' + page.status_code)

        print('Request successful')

        # Convert csv data to string
        content = str(page.content)
        # Remove large, unnecessary header
        trimContent = content[content.find('Station'):len(content)]
        # Replace newline/return with string literal newline
        formatContent = trimContent.replace('\\r\\n', '\n')
        # Convert content to file object
        contentFile = StringIO(formatContent)

        # Read content into pandas dataframe. Second header row contains units
        ndawnData = pd.read_csv(contentFile, header = [0, 1])

        # Concatenate headers to include units
        # Assign column list to object
        columnHeaders = list(ndawnData.columns)
        # List of new headers
        newHeaderList = []
        # Iterate through column names
        for number in range(0, len(columnHeaders)):
            # If no unit, keep header unchanged, pass into new list
            if 'Unnamed' in columnHeaders[number][1]:
                newHeaderList.append(columnHeaders[number][0])
            # If unit exists, concatenate header and unit, pass into ne
w list
            else:
                newHeader = columnHeaders[number][0] + ' (' + columnHea
ders[number][1] + ') '
                newHeaderList.append(newHeader)
        # Assign new column names
        ndawnData.columns = newHeaderList

        # Create single column for datetime
        ndawnData['Date'] = pd.to_datetime(ndawnData[['Year', 'Month',
'Day']])

        # Save to csv if save option selected
        if self.save:
            ndawnData.to_csv('ndawnData.csv', index=False)

        return ndawnData

'''
# Example syntax:
exampleRequest = ndawn_request(startDate='2020-06-23', endDate='2020-06
-28', ontology=['Air Pressure', 'Relative Humidity', 'Soil Temperature
', 'Wind Direction', 'Wind Speed'], location=['Mavie', 'Ottertail', 'Pe
rham', 'Perley'])
ndawnDF = exampleRequest.get_data()
```

Out[1]:

```
"\n# Example syntax:\nexampleRequest = ndawn_request(startDate='2020-
06-23', endDate='2020-06-28', ontology=['Air Pressure', 'Relative Hum
idity', 'Soil Temperature', 'Wind Direction', 'Wind Speed'], location
=['Mavie', 'Ottertail', 'Perham', 'Perley'])\nndawnDF = exampleReques
t.get_data()\n"
```

In [2]:

```
from datetime import timedelta ## importing time deltal for the start a
nd end date
```

In [5]:

```
data = ndawn_request(startDate= str(date.today() - timedelta(30)), endD
ate= str(date.today()), ontology=['Air Temperature'], location=['Ada','
Becker','Campbell','Clarissa','Eldred','Fox','Greenbush','Hubbard', 'Hu
mboldt', 'Kennedy','Little Falls','Mavie','Ottertail','Parkers Prairie
','Perham','Perley','Pine Point','Rice', 'Roseau','Sabin','Staples', 'S
tephen', 'Ulen','Wadena','Warren', 'Waukon', 'Westport','Williams'])
ndawnDF = data.get_data()
```

```
Request successful
```

In [6]:

```
ndawnDF ##Dataframe
```

Out[6]:

	Station Name	Latitude (deg)	Longitude (deg)	Elevation (ft)	Year	Month	Day	Hour (CST)	Avg Air Temp (Degrees F)	Avg Ai Temp Flag
0	Ada	47.321100	-96.513900	910.0	2021.0	3.0	18.0	100.0	21.916	NaN
1	Ada	47.321100	-96.513900	910.0	2021.0	3.0	18.0	200.0	20.800	NaN
2	Ada	47.321100	-96.513900	910.0	2021.0	3.0	18.0	300.0	19.891	NaN
3	Ada	47.321100	-96.513900	910.0	2021.0	3.0	18.0	400.0	17.884	NaN
4	Ada	47.321100	-96.513900	910.0	2021.0	3.0	18.0	500.0	16.232	NaN
...	...	...	...	...	...	...	...	...	...	...
20156	Williams	48.858454	-94.980897	1093.0	2021.0	4.0	16.0	2100.0	40.876	NaN
20157	Williams	48.858454	-94.980897	1093.0	2021.0	4.0	16.0	2200.0	38.500	NaN
20158	Williams	48.858454	-94.980897	1093.0	2021.0	4.0	16.0	2300.0	36.037	NaN
20159	Williams	48.858454	-94.980897	1093.0	2021.0	4.0	16.0	2400.0	35.893	NaN
20160	'	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

20161 rows × 13 columns

In [7]:

```
for i in ndawnDF:
    print(i)
```

```
Station Name
Latitude (deg)
Longitude (deg)
Elevation (ft)
Year
Month
Day
Hour (CST)
Avg Air Temp (Degrees F)
Avg Air Temp Flag
Avg Air Temp at 9 m (Degrees F)
Avg Air Temp at 9 m Flag
Date
```

In [8]:

```
avetemp = ndawnDF.groupby(["Station Name"])[['Latitude (deg) ', 'Longit
ude (deg) ', 'Avg Air Temp (Degrees F) ']].mean()
```

```
In [9]: avetemp ##Reshaped so that it just has station, lat, long, and average 30 day temp
```

Out[9]:

	Latitude (deg)	Longitude (deg)	Avg Air Temp (Degrees F)
Station Name			
'	NaN	NaN	NaN
Ada	47.321100	-96.513900	39.498304
Becker	45.344300	-93.850000	43.157939
Campbell	46.064932	-96.370165	40.354461
Clarissa	46.111560	-94.905800	40.578671
Eldred	47.688000	-96.822000	38.602726
Fox	48.877738	-95.850160	37.903894
Greenbush	48.704000	-96.325000	37.413475
Hubbard	46.820590	-94.995800	39.480175
Humboldt	48.884000	-97.150000	36.644451
Kennedy	48.636709	-97.041117	38.092289
Little Falls	45.932130	-94.251400	41.386361
Mavie	48.121000	-95.971000	38.488503
Ottertail	46.426430	-95.573500	40.046131
Parkers Prairie	46.169400	-95.356000	39.957750
Perham	46.610477	-95.601876	39.854543
Perley	47.179000	-96.680000	39.514340
Pine Point	47.012860	-95.371700	38.849354
Rice	45.793830	-94.261800	42.127231
Roseau	48.685000	-95.734000	38.243338
Sabin	46.794389	-96.611683	40.048608
Staples	46.387730	-94.808700	40.522642
Stephen	48.456750	-96.853953	38.116468
Ulen	47.050473	-96.108432	38.723399
Wadena	46.448300	-95.214200	39.936493
Warren	48.137000	-96.839000	37.574749
Waukon	47.325859	-96.132504	39.043386
Westport	45.715080	-95.171800	40.371417
Williams	48.858454	-94.980897	36.895550

```
In [12]: avetemp.to_csv("d:/git/GIS5572/Lab4/Data/Final_data.csv") ##dataframe to csv
```

In [1]:

```
import arcpy
```

In [2]:

```
arcpy.env.workspace = "D:/Users/Owner/Documents/ArcGIS/Projects/Arc_Lab4/Arc_Lab4.gdb"
arcpy.env.overwriteOutput = True
```

In [21]:

```
arcpy.conversion.TableToTable("d:/git/GIS5572/Lab4/Data/Final_data.csv", "", "Data") #imports table to database
```

Out[21]:

Output

D:/Users/Owner/Documents/ArcGIS/Projects/Arc_Lab4/Arc_Lab4.gdb\Data

Messages

Start Time: Saturday, April 17, 2021 4:20:58 PM

Succeeded at Saturday, April 17, 2021 4:20:58 PM (Elapsed Time: 0.55 seconds)

In [22]:

```
arcpy.management.XYTableToPoint("Data", r"D:\Users\Owner\Documents\ArcGIS\Projects\Arc_Lab4\Arc_Lab4.gdb\Data_Points", "Longitude__deg_", "Latitude__deg_", None, "GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,298.257223563]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]];-400 -400 1000000000;-100000 10000;-100000 10000;8.98315284119521E-09;0.001;0.001;IsHighPrecision") ##Table to xy points
```

Out[22]:

Output

D:\Users\Owner\Documents\ArcGIS\Projects\Arc_Lab4\Arc_Lab4.gdb\Data_Points

Messages

Start Time: Saturday, April 17, 2021 4:21:00 PM

WARNING 100160: Some of the features have invalid geometry and have been removed from the result

WARNING 000192: Invalid value for rows: 1

Succeeded at Saturday, April 17, 2021 4:21:02 PM (Elapsed Time: 1.76 seconds)

In [26]:

```
arcpy.ga.IDW("Data_Points", "Avg_Air_Temp__Degrees_F_", "IDW_Stats") ## Inverse distance Weighting
arcpy.env.addOutputsToMap = True
```

In [31]:

```
arcpy.Kriging_3d("Data_Points", "Avg_Air_Temp__Degrees_F_") ##Ordinary Kriging
```

Out[31]:

Output

id	value
0	D:/Users/Owner/Documents/ArcGIS/Projects/Arc_Lab4/Arc_Lab4.gdb\Kriging_Data2
1	

Messages

```
Start Time: Saturday, April 17, 2021 4:49:53 PM
SPHERICAL
Lag size = nan(snan)
Partial sill = nan(snan)
Nugget = nan(snan)
Major range = nan(snan)
Succeeded at Saturday, April 17, 2021 4:49:54 PM (Elapsed Time: 1.45 seconds)
```

In [3]:

```
out_surface_raster = arcpy.sa.Kriging("Data_Points", "Avg_Air_Temp__Degrees_F_", "LinearDrift 0.013200 # # #", 0.0132, "VARIABLE 12", None); out_surface_raster.save(r"D:\Users\Owner\Documents\ArcGIS\Projects\Arc_Lab4\Arc_Lab4.gdb\Universal_ras")

##Universal Kriging
##This code works but it does not produce the same results as the statistical analysis
```

In [4]:

```
arcpy.ga.EmpiricalBayesianKriging("Data_Points", "Avg_Air_Temp__Degrees_F_", "EBK_STAT", r"D:\Users\Owner\Documents\ArcGIS\Projects\Arc_Lab4\Arc_Lab4.gdb\EBK_Rast", 0.0132, "NONE", 100, 1, 100, "NBRTYPE=StandardCircular RADIUS=1.20984183082956 ANGLE=0 NBR_MAX=15 NBR_MIN=10 SECTOR_TYPE=ONE_SECTOR", "PREDICTION", 0.5, "EXCEED", None, "POWER") ##Empirical Bayesian Kriging.
```

Out[4]:

Output

id	value
0	a Layer object
1	D:\Users\Owner\Documents\ArcGIS\Projects\Arc_Lab4\Arc_Lab4.gdb\EBK_Rast

Messages

```
Start Time: Sunday, April 18, 2021 10:09:49 PM

Warning(s) for dataset: Length of the radius of the search circle = 1.3468e+05 meters.
Succeeded at Sunday, April 18, 2021 10:09:59 PM (Elapsed Time: 9.99 seconds)
```

In [ ]: