Spectrum Spatial Python Package

This notebook describes the **`spectrumspatialpy`** python library through examples.

Setup and Prerequisites ¶

Setup and prerequisites are desicribed in the `spectrumpy` notebook.

In [1]:

```
pip install --quiet git+https://github.com/carypeebles/spectrumpy
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
import spectrumpy
myServer=spectrumpy.Servers.getServer('localhost')
```

About the Spectrum Spatial package

The ***spectrumspatialpy*** package provides Python integration for the Spectrum Spatial services such as the Feature Service for querying spatial data.

Installing the spectrumpy package

In [3]:

```
pip install --quiet git+https://github.com/carypeebles/spectrumpy
```

```
Note: you may need to restart the kernel to use updated packages.
```

Instantiating a Spectrum Spatial service

A Spectrum Spatial service is instantiated using an established Spectrum server object. For example,

In [4]:

```
import spectrumspatialpy
mySpectrumSpatial=spectrumspatialpy.SpatialServer(myServer)
```

There are several service objects that are accessible off the main Spectrum Spatial object (`mySpectrumSpatial`).

- mySpectrumSpatial.FeatureService() : Returns the **Feature Service** for this server.
- mySpectrumSpatial.GeometryOperations() : Returns the **Geometry Service** for this server. This does not correspond to the LIM Geometry service; rather, it exposes a method for converting a GeoJSON FeatureCollection to a GeoPandas GeoDataFrame.
- mySpectrumSpatial.NamedResourceService() : Returns the **Named Resource Service** for this server.
- mySpectrumSpatial.Thematics() : Returns the **Thematics Service** for this server. This does not correspond to a LIM service, it was created to contain some methods that are specifically

designed to output a theme from Python into the repository. There will be an example below.

Feature Service

The `FeatureService` exposes several methods represented by the LIM FeatureService (http://support.pb.com/help/spectrum/18.2/en/webhelp/Spatial/index.html#Spatial/source/Developme

- listTables() : Prints to the output the available named tables at the server
- describeTable(tablePath) : Prints to the output a description of the table
- query() : Accepts an MISQL query and returns a GeoJSON FeatureCollection
- get() : Exposes a way to issue an arbitrary request against the Feature Service

The code below lists the tables at `mySpectrumSpatial` and describes the USA sample table.

In [5]:

```
ftrService = mySpectrumSpatial.FeatureService()
ftrService.listTables()
```

Out[5]:

```
['/Jupyter/NamedTables/StatesQuery',
 '/Samples/NamedTables/CountriesShapeTable',
 '/Samples/NamedTables/FlightsTable',
 '/Samples/NamedTables/Grid15Table',
 '/Samples/NamedTables/Interstates',
 '/Samples/NamedTables/LANDMRKS',
 '/Samples/NamedTables/Lakes',
 '/Samples/NamedTables/LineTable',
 '/Samples/NamedTables/MRRWorldTable',
 '/Samples/NamedTables/MississippiRiver',
 '/Samples/NamedTables/NamedViewTable',
 '/Samples/NamedTables/NamedViewTable_NamedTables',
 '/Samples/NamedTables/OceanTable',
 '/Samples/NamedTables/SavingsNLoan',
 '/Samples/NamedTables/Secondary_Rds',
 '/Samples/NamedTables/Streams_Rivers',
 '/Samples/NamedTables/UKCTY215',
 '/Samples/NamedTables/UK_REGNS',
 '/Samples/NamedTables/USA',
 '/Samples/NamedTables/USAViewTable',
 '/Samples/NamedTables/USA_CAPS',
 '/Samples/NamedTables/USA_OutLine',
 '/Samples/NamedTables/USCTY153',
 '/Samples/NamedTables/USCTY_1K',
 '/Samples/NamedTables/USCTY_8K',
 '/Samples/NamedTables/US_Ele_Grid_Table',
 '/Samples/NamedTables/US_HIWAY',
 '/Samples/NamedTables/Urban_Areas',
 '/Samples/NamedTables/Urban_CitiesPop10K_plus',
 '/Samples/NamedTables/Us_Int_Shields1',
 '/Samples/NamedTables/Us_Int_Shields2',
 '/Samples/NamedTables/Us_Int_Shields3',
 '/Samples/NamedTables/Wilderness_Areas',
 '/Samples/NamedTables/WorldGeoPackageTable',
 '/Samples/NamedTables/WorldModifiableNativeTable',
 '/Samples/NamedTables/WorldModifiableTable',
 '/Samples/NamedTables/WorldOracleDBQueryTable',
 '/Samples/NamedTables/WorldTable',
 '/Samples/NamedTables/WorldcapTable',
 '/Samples/NamedTables/airportswithtimefieldsTable',
 '/Samples/NamedTables/dcwashcities',
 '/Samples/NamedTables/dcwashcounties',
 '/Samples/NamedTables/dcwashexpressways',
 '/Samples/NamedTables/dcwashgazetteer1',
 '/Samples/NamedTables/dcwashgazetteer2',
 '/Samples/NamedTables/dcwashgazetteer3',
 '/Samples/NamedTables/dcwashgazetteer4',
 '/Samples/NamedTables/dcwashgazetteer5',
 '/Samples/NamedTables/dcwashgazetteer6',
 '/Samples/NamedTables/dcwashgazetteer7',
 '/Samples/NamedTables/dcwashlandmarks',
 '/Samples/NamedTables/dcwashlanduse',
 '/Samples/NamedTables/dcwashlocalhwys_med',
 '/Samples/NamedTables/dcwashlocalrtes',
 '/Samples/NamedTables/dcwashoneways',
 '/Samples/NamedTables/dcwashprimaryhwys',
 '/Samples/NamedTables/dcwashprimaryhwys_med',
 '/Samples/NamedTables/dcwashrailroads',
 '/Samples/NamedTables/dcwashregionalhwys',
 '/Samples/NamedTables/dcwashrivers',
 '/Samples/NamedTables/dcwashsecondaryhwys',
 '/Samples/NamedTables/dcwashshldinter_0to5',
 '/Samples/NamedTables/dcwashshldinter_15to50',
 '/Samples/NamedTables/dcwashshldinter_5to15',
 '/Samples/NamedTables/dcwashshldstate_0to5',
 '/Samples/NamedTables/dcwashshldstate_15to50',
 '/Samples/NamedTables/dcwashshldstate_5to15',
 '/Samples/NamedTables/dcwashshldus_0to5',
 '/Samples/NamedTables/dcwashshldus_15to50',
 '/Samples/NamedTables/dcwashshldus_5to15',
 '/Samples/NamedTables/dcwashsignposts',
 '/Samples/NamedTables/dcwashstreets',
 '/Samples/NamedTables/dcwashtowns',
 '/Samples/NamedTables/dcwashurbanareas',
 '/Samples/NamedTables/dcwashwaterbodies']
```

In [6]:

```
ftrService.describeTable("/Samples/NamedTables/USA")
```

```
TABLE:/Samples/NamedTables/USA
------------------------------------------------------------------------
-----------
Obj                            Geometry
MI_Style                       Style
State_Name                     String
State                          String
Fips_Code                      String
Pop_1990                       Decimal  (10,0)
Pop_2000                       Decimal  (10,0)
Num_Hh_1990                    Decimal  (10,0)
Num_Hh_2000                    Integer
Med_Inc_1990                   Decimal  (10,0)
Med_Inc_2000                   Double
Pop_Urban_2000                 Integer
Pop_Rural_2000                 Integer
Pop_Male                       Decimal  (10,0)
Pop_Female                     Decimal  (10,0)
Pop_Cauc                       Decimal  (10,0)
```

MISQL Query

The query method accepts an MISQL (http://support.pb.com/help/spectrum/18.2/en/webhelp/Spatial/index.html#Spatial/source/misql/misql query and returns a GeoJSON FeatureCollection. The following example returns all features from the USA sample dataset whose state name begins with 'N' and prints out some results. Note we return only the centroid of the state geometry only for the purposes of showing a geometry without generating too much output.

In [7]:

```
query = "select State_Name, State, Fips_Code, Pop_1990, Pop_2000, MI_Centroid
    "from \"/Samples/NamedTables/USA\" " \
    "where State_Name LIKE 'N%'"
states = ftrService.query(query)
print(states)
```

```
{'type': 'FeatureCollection', 'features': [{'type': 'Feature', 'propertie
s': {'State_Name': 'Nebraska', 'State': 'NE', 'Fips_Code': '31', 'Pop_199
0': 1578385.0, 'Pop_2000': 1711263.0}, 'geometry': {'type': 'Point', 'coord
inates': [-99.680521, 41.50087]}, 'id': 28}, {'type': 'Feature', 'propertie
s': {'State_Name': 'Nevada', 'State': 'NV', 'Fips_Code': '32', 'Pop_1990':
1201833.0, 'Pop_2000': 1998257.0}, 'geometry': {'type': 'Point', 'coordinat
es': [-117.021761, 38.502190999999996]}, 'id': 29}, {'type': 'Feature', 'pr
operties': {'State_Name': 'New Hampshire', 'State': 'NH', 'Fips_Code': '3
3', 'Pop_1990': 1109252.0, 'Pop_2000': 1235786.0}, 'geometry': {'type': 'Po
int', 'coordinates': [-71.63089099999999, 44.001070999999996]}, 'id': 30},
{'type': 'Feature', 'properties': {'State_Name': 'New Jersey', 'State': 'N
J', 'Fips_Code': '34', 'Pop_1990': 7730188.0, 'Pop_2000': 8414350.0}, 'geom
etry': {'type': 'Point', 'coordinates': [-74.7271, 40.142868]}, 'id': 31},
{'type': 'Feature', 'properties': {'State_Name': 'New Mexico', 'State': 'N
M', 'Fips_Code': '35', 'Pop_1990': 1515069.0, 'Pop_2000': 1819046.0}, 'geom
etry': {'type': 'Point', 'coordinates': [-106.02552, 34.16617]}, 'id': 32},
{'type': 'Feature', 'properties': {'State_Name': 'New York', 'State': 'NY',
'Fips_Code': '36', 'Pop_1990': 17990455.0, 'Pop_2000': 18976457.0}, 'geomet
ry': {'type': 'Point', 'coordinates': [-76.502057, 42.856215999999996]}, 'i
d': 33}, {'type': 'Feature', 'properties': {'State_Name': 'North Carolina',
'State': 'NC', 'Fips_Code': '37', 'Pop_1990': 6628637.0, 'Pop_2000': 804931
3.0}, 'geometry': {'type': 'Point', 'coordinates': [-80.018692, 35.21381
7]}, 'id': 34}, {'type': 'Feature', 'properties': {'State_Name': 'North Dak
ota', 'State': 'ND', 'Fips_Code': '38', 'Pop_1990': 638800.0, 'Pop_2000': 6
42200.0}, 'geometry': {'type': 'Point', 'coordinates': [-100.3012909999999
9, 47.46788]}, 'id': 35}], 'Metadata': [{'name': 'State_Name', 'type': 'Str
ing'}, {'name': 'State', 'type': 'String'}, {'name': 'Fips_Code', 'type':
'String'}, {'name': 'Pop_1990', 'type': 'Decimal', 'fractionalDigits': 0,
'totalDigits': 10}, {'name': 'Pop_2000', 'type': 'Decimal', 'fractionalDigi
ts': 0, 'totalDigits': 10}, {'name': 'MI_Centroid_Obj_', 'type': 'Geometr
y', 'crs': {'type': 'name', 'properties': {'name': 'epsg:4267'}}, 'bbox':
[-117.021761, 34.16617, -71.63089099999999, 47.46788]}], 'bbox': [-117.0217
61, 34.16617, -71.63089099999999, 47.46788], 'crs': {'type': 'name', 'prope
rties': {'name': 'epsg:4267'}}}
```

In [8]:

```
# Iterate through the individual features and properties to display some outp
features = states["features"]
for i in range(len(features)):
    properties = features[i]["properties"]
    print (properties["State_Name"], end='')
    print ("\t", end='')
    print (properties["State"], end='')
    print ("\t", end='')
    print (properties["Fips_Code"], end='')
    print ("\t", end='')
    print (str(properties["Pop_1990"]), end='')
    print ("\t", end='')
    print (str(properties["Pop_2000"]), end='')
    print ("\t", end='')
    print (str(features[i]["geometry"]['coordinates'][0]), end='')
    print (",", end='')
    print (str(features[i]["geometry"]['coordinates'][1]), end='')
    print ("")
```

```
Nebraska        NE      31      1578385.0       1711263.0       -99.680521,
41.50087
Nevada  NV      32      1201833.0       1998257.0       -117.021761,38.5021
90999999996
New Hampshire   NH      33      1109252.0       1235786.0       -71.6308909
9999999,44.001070999999996
New Jersey      NJ      34      7730188.0       8414350.0       -74.7271,4
0.142868
New Mexico      NM      35      1515069.0       1819046.0       -106.02552,
34.16617
New York        NY      36      17990455.0      18976457.0      -76.502057,
42.856215999999996
North Carolina  NC      37      6628637.0       8049313.0       -80.018692,
35.213817
North Dakota    ND      38      638800.0        642200.0        -100.301290
99999999,47.46788
```

Combining Geocoding, Routing and Spatial

The `spectrumpy` and `spectrumspatialpy` packages allow integrated use of any Spectrum capabilities exposed by the server. This example below will combine **Geocoding**, **Routing**, and **Spatial** to produce an elevation profile plot of a route between two addresses.

The example below will perform these steps:

- geocode two addresses
- Invoke a custom data flow that accepts two lon/lat pairs, calls the Route stage and returns the route geometry
- Determine the elevation of node in the route (max of 1000)
- Create a plot using matplotlib of the elevation values

In [9]:

```
start_address = "4750 Walnut St, Boulder, CO 80301"
end_address = "1 Market St, San Francisco, CA 94105"
```

In [10]:

```
start_geocode = myServer.SpectrumServices().GeocodeUSAddress(Data_AddressLine
                                                 Option_Dataset="us",
                                                 Option_OutputRecordType="Lat
end_geocode = myServer.SpectrumServices().GeocodeUSAddress(Data_AddressLine1=
                                                 Option_Dataset="us",
                                                 Option_OutputRecordType="Lat
```

In [11]:

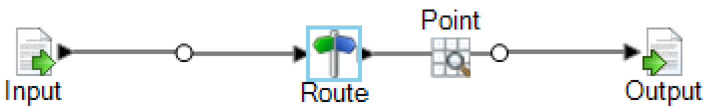
```
print(start_geocode)
print(end_geocode)
```

```
{
  "output_port" : [ {
    "Latitude" : "40.018297",
    "Longitude" : "-105.240967",
    "StreetSide" : "L",
    "FirmName" : "",
    "AddressLine1" : "4750 Walnut St",
    "AddressLine2" : "",
    "LastLine" : "Boulder, CO  80301-2532",
    "StreetName" : "Walnut",
    "CrossStreetName" : "",
    "LeadingDirectional" : "",
    "CrossStreetLeadingDirectional" : "",
    "HouseNumber" : "4750",
    "HouseNumber2" : "",
    "TrailingDirectional" : "",
    "CrossStreetTrailingDirectional" : "",
    "StreetSuffix" : "St",
    "CrossStreetSuffix" : "",
    "ApartmentLabel" : "",
    "ApartmentLabel2" : "",
    "ApartmentNumber" : "",
    "ApartmentNumber2" : "",
    "AdditionalInputData" : "",
    "City" : "Boulder",
    "StateProvince" : "CO",
    "PostalCode.Base" : "80301",
    "PostalCode.AddOn" : "2532",
    "PostalCode" : "80301-2532",
    "PrivateMailbox.Designator" : "",
    "PrivateMailbox" : "",
    "USUrbanName" : "",
    "Country" : "United States of America",
    "RRHC" : "",
    "LocationCode" : "AP05",
    "MatchCode" : "S80",
    "StreetDataType" : "MASTER LOCATION",
    "Confidence" : "100",
    "ProcessedBy" : "KGL",
    "StreetSegmentPoints" : [ ],
    "PBKey" : "P00003PZZOIE",
    "Status" : "",
    "Status.Code" : "",
    "Status.Description" : "",
    "user_fields" : [ ]
  } ]
}
{
  "output_port" : [ {
    "Latitude" : "37.793872",
    "Longitude" : "-122.394865",
    "StreetSide" : "L",
    "FirmName" : "Steuart Tower",
    "AddressLine1" : "1 Market St",
    "AddressLine2" : "",
    "LastLine" : "San Francisco, CA  94105-1420",
    "StreetName" : "Market",
    "CrossStreetName" : "",
    "LeadingDirectional" : "",
    "CrossStreetLeadingDirectional" : "",
    "HouseNumber" : "1",
    "HouseNumber2" : "",
    "TrailingDirectional" : "",
    "CrossStreetTrailingDirectional" : "",
    "StreetSuffix" : "St",
    "CrossStreetSuffix" : "",
    "ApartmentLabel" : "",
    "ApartmentLabel2" : "",
    "ApartmentNumber" : "",
    "ApartmentNumber2" : "",
    "AdditionalInputData" : "",
    "City" : "San Francisco",
    "StateProvince" : "CA",
    "PostalCode.Base" : "94105",
    "PostalCode.AddOn" : "1420",
    "PostalCode" : "94105-1420",
    "PrivateMailbox.Designator" : "",
    "PrivateMailbox" : "",
    "USUrbanName" : "",
    "Country" : "United States of America",
    "RRHC" : "",
    "LocationCode" : "AP05",
    "MatchCode" : "S80",
    "StreetDataType" : "MASTER LOCATION",
    "Confidence" : "100",
    "ProcessedBy" : "KGL",
    "StreetSegmentPoints" : [ ],
    "PBKey" : "P00002T4SV3T",
    "Status" : "",
    "Status.Code" : "",
    "Status.Description" : "",
    "user_fields" : [ ]
  } ]
}
```

In [12]:

```python
import json

json_start_geocode = json.loads(start_geocode)
json_end_geocode = json.loads(end_geocode)
latitude1 = json_start_geocode['output_port'][0]["Latitude"]
longitude1 = json_start_geocode['output_port'][0]["Longitude"]
latitude2 = json_end_geocode['output_port'][0]["Latitude"]
longitude2 = json_end_geocode['output_port'][0]["Longitude"]
```

This notebook includes a dataflow named `spectrumspatialpy_route` under the `dataflows` folder which must be imported into your Spectrum for this notebook to run. The dataflow is defined as follows:



In [13]:

```
# NOTE: This a long running service and needs the notebook started with the c
#       --NotebookApp.iopub_data_rate_limit=1000000000.0
#
sroute = myServer.SpectrumServices().spectrumspatialpy_route(
    Data_latitude1=latitude1,
    Data_longitude1=longitude1,
    Data_latitude2=latitude2,
    Data_longitude2=longitude2)
jroute = json.loads(sroute)
nodes = jroute['Output'][0]['RouteGeometry']['Pos']
print(nodes)
```

```
[{'X': -105.240967, 'Y': 40.018297, 'Z': 0.0}, {'X': -105.24096, 'Y': 40.
018349, 'Z': 0.0}, {'X': -105.24096, 'Y': 40.018349, 'Z': 0.0}, {'X': -10
5.24096, 'Y': 40.018349, 'Z': 0.0}, {'X': -105.240988, 'Y': 40.018363,
'Z': 0.0}, {'X': -105.241085, 'Y': 40.018432, 'Z': 0.0}, {'X': -105.24116
3, 'Y': 40.018483, 'Z': 0.0}, {'X': -105.24127, 'Y': 40.018506, 'Z': 0.
0}, {'X': -105.241704, 'Y': 40.018571, 'Z': 0.0}, {'X': -105.241704, 'Y':
40.018571, 'Z': 0.0}, {'X': -105.241622, 'Y': 40.018993, 'Z': 0.0}, {'X':
-105.241622, 'Y': 40.018993, 'Z': 0.0}, {'X': -105.241595, 'Y': 40.01921
3, 'Z': 0.0}, {'X': -105.241595, 'Y': 40.019213, 'Z': 0.0}, {'X': -105.24
2084, 'Y': 40.019265, 'Z': 0.0}, {'X': -105.242153, 'Y': 40.019283, 'Z':
0.0}, {'X': -105.242153, 'Y': 40.019283, 'Z': 0.0}, {'X': -105.242195,
'Y': 40.019294, 'Z': 0.0}, {'X': -105.242318, 'Y': 40.019356, 'Z': 0.0},
{'X': -105.242318, 'Y': 40.019356, 'Z': 0.0}, {'X': -105.242326, 'Y': 40.
01936, 'Z': 0.0}, {'X': -105.242503, 'Y': 40.019508, 'Z': 0.0}, {'X': -10
5.242605, 'Y': 40.019561, 'Z': 0.0}, {'X': -105.24272, 'Y': 40.019597,
'Z': 0.0}, {'X': -105.243247, 'Y': 40.019655, 'Z': 0.0}, {'X': -105.24336
5, 'Y': 40.019655, 'Z': 0.0}, {'X': -105.243365, 'Y': 40.019655, 'Z': 0.
0}, {'X': -105.243374, 'Y': 40.019655, 'Z': 0.0}, {'X': -105.243504, 'Y':
40.019635, 'Z': 0.0}, {'X': -105.243858, 'Y': 40.019521, 'Z': 0.0}, {'X':
```

Now we will convert each coordinate in the route to an elevation. Spectrum Spatial includes an elevation grid file at `/Samples/NamedTables/MRRWorldTable`. The elevation for a specific coordinate can be determined using the MI_GridValueAt (http://support.pb.com/help/spectrum/18.2/en/webhelp/Spatial/index.html#Spatial/source/misql/misql function. A route could contain many intermediate nodes so this logic will bundle multiple nodes into a single MISQL query. It does this by breaking the total number of nodes in the route into an outer set (by dividing by 100) and then within each set it will split it into 10 nodes and generate an MISQL query for each of the 10.

In [14]:

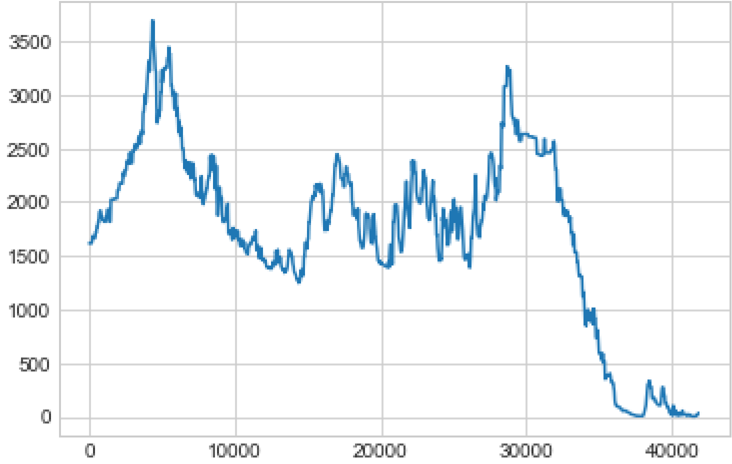
```
outer_step = int(len(nodes) / 100)
inner_step = int(outer_step / 10)

plot_x = []
plot_y = []

for iouter in range(0, len(nodes), outer_step):
    query = "select "
    first=True
    idx = 1
    for iinner in range(iouter, iouter + outer_step, inner_step):
        if iinner <= len(nodes):
            node = nodes[iinner]
            x = node["X"]
            y = node["Y"]
            if not first:
                query += ","
            first=False
            query += "MI_GridValueAt(MI_RASTER, MI_POINT(" + str(x) + ", " +
            idx += 1
    query += " from \"/Samples/NamedTables/MRRWorldTable\""
    fc = ftrService.query(query)
    idx = 1
    for iinner in range(iouter, iouter + outer_step, inner_step):
        if iinner <= len(nodes):
            node = nodes[iinner]
            x = node["X"]
            y = node["Y"]
            elevation = fc['features'][0]['properties']['VAL_' + str(idx)]
            plot_x.append(iinner)
            plot_y.append(elevation)
            idx += 1
```

In [15]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

In [16]:

```
fig = plt.figure()
ax = plt.axes()
ax.plot(plot_x, plot_y);
```



In [ ]:

```
```