Population Calculation Methodology

Data Source

I used the 2015 Population Density data at 15 minutes resolution from Gridded Population of the World (GPW), v4 dataset from NASA's Socioeconomic Data and Applications Center (sedac).

Link: http://sedac.ciesin.columbia.edu/data/set/gpw-v4-population-density-rev10/data-download

The dataset contained population densities for longitudes (-180° to 180°) and latitudes (-60° to 85°) with the value -9999 for regions where the data was unavailable. It contained 1440 columns and 580 rows the lower left most block representing -60° latitude and -180° longitude.

Population Density Grid Creation

I performed the following manipulations on the original population density grid to simplify the required population calculation.

- I added the rows for the missing latitudes (-60 to -90) and (-85 to 90)
- In order to make the grid square to make the radius calculation simpler, I repeated the rows 2 times i.e. changing the granularity of latitude from 15 minutes to 7.5 minutes while assuming the density to remain the same.
- I changed the not available density values to zero hence making the assumption that there is no human population in those regions.

Code:

```
def get_density_grid(pop_density_path):

    np_pop_density = np.loadtxt(pop_density_path, skiprows=6)
    n_rows, n_cols = np_pop_density.shape

    # insert the rows for the omitted latitudes i.e (-60 to -90) and (85 to 90)
    np_pop_density_adj = np.vstack([np.zeros((20, n_cols)), np_pop_density, np.zeros((120, n_cols))])

    # to simplify calculations make the scale of latitudes (-90 to 90) and longitudes(-180 to 180) the same
    np_pop_density_adj = np.repeat(np_pop_density_adj, 2, axis=0)

    # replace no data value with 0 - making the assumption that there is no human population there
    np_pop_density_adj[np_pop_density_adj == -9999] = 0

    return np_pop_density_adj
```

Convert Densities to Population

I am assuming the globe to be a perfect sphere with surface area = 510.1 Million sq.km

Calculations:

surface area of a grid block = surface_area_globe / number of grid blocks
= 510.1 * 10^6 / (1440 * 1440)
= 246 sq.km

grid length = sqrt(surface are of a grid block)
= 15.68

grid block population = grid block density * surface area of a grid block

Code:

```
def convert_density_to_population(np_pop_density):

    latitude_grid_count, longitude_grid_count = np_pop_density.shape

    # calculate area of a single grid block
    surface_area_globe = 510.1e6
    surface_area_block = surface_area_globe / (latitude_grid_count * longitude_grid_count)
    grid_length = np.sqrt(surface_area_block)

    # convert population densities to population
    np_population = np.round(np_pop_density * surface_area_block)

    return np_population, grid_length
```

Convert coordinates to row and column index of the population matrix

Given the resolution of the longitude and latitude is 15 minutes and 7.5 minutes (after the adjustment), the relation between the coordinates and row column index is straightforward

Calculations:

row_index = (8 x latitude) + 720
column_index = (4 x longitude) + 720

Code

```
def coord_to_rc(latitude, longitude):

    # latitude should be between -90 and 90
    row_idx = np.round(8 * latitude + 720)

    # longitude should be between -180 and 180
    col_idx = np.round(4 * longitude + 720)

    return row_idx, col_idx
```

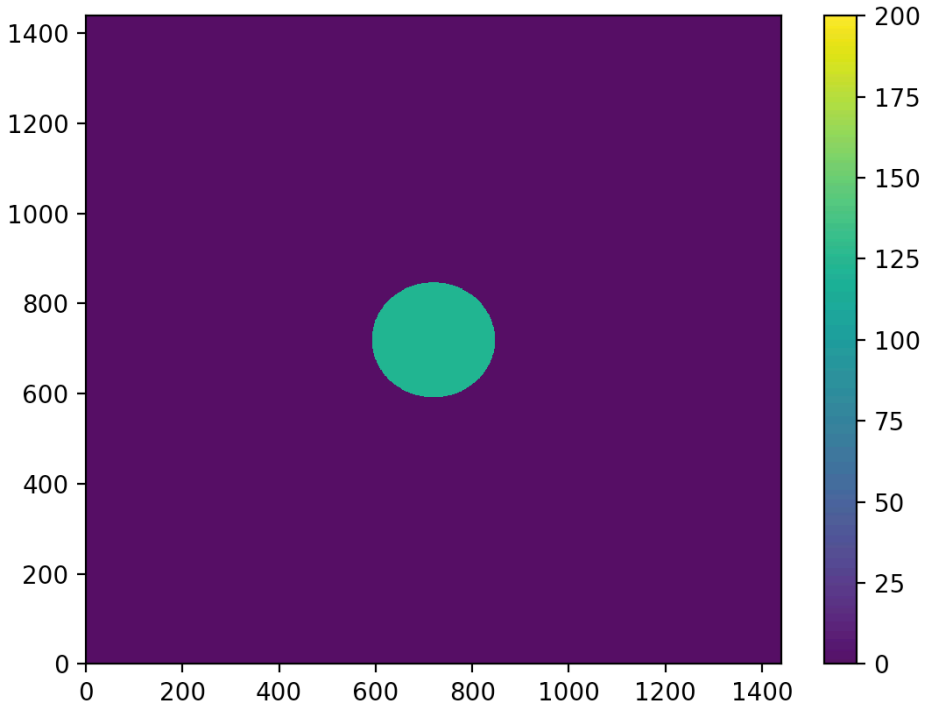
Identifying the region within the radius of the coordinates provided

In order to identify the grid blocks that lie within the radius of the coordinates, the equation of circle is used.

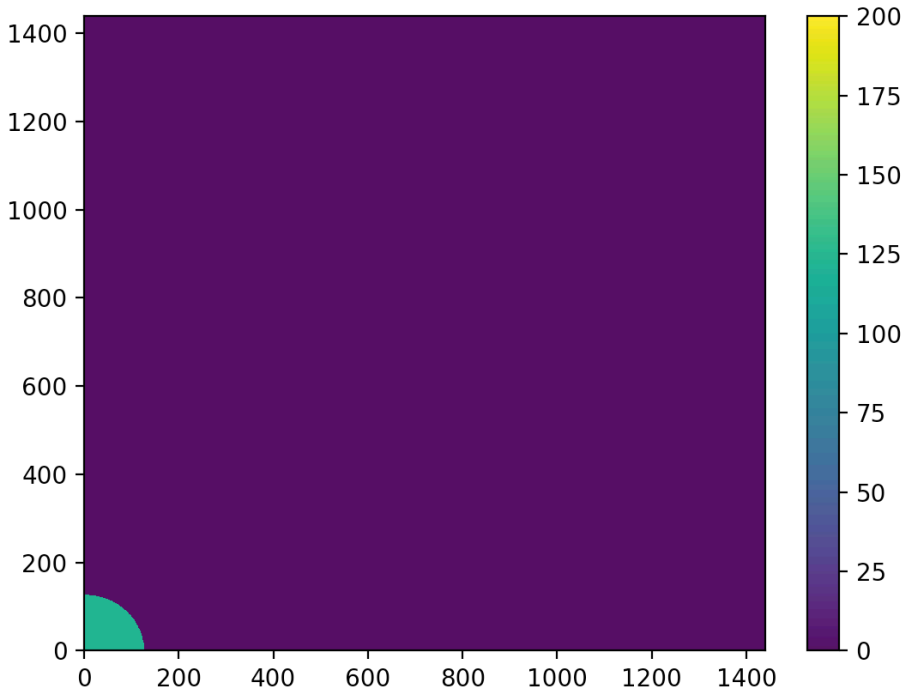
$$X^2 + Y^2 = R^2$$

A mask matrix is created with true values for the (row_idx, col_idx) pairs that lie within the radius of the provided coordinated,

The plot below shows the mask matrix for latitude value 0, longitude value 0, and radius value 500 km. Green indicates True values, and purple indicates False values



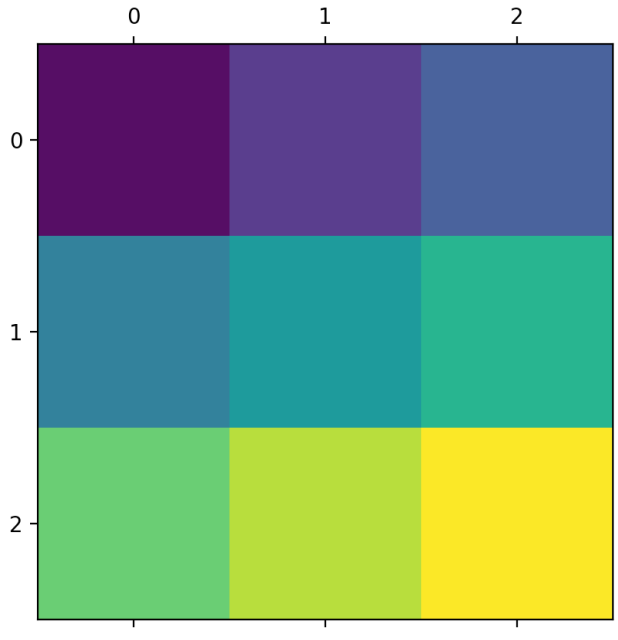
However, we need to account for the fact that the globe is spherical and there are no dead ends. For instance, the plot below shows the mask matrix for latitude value -90, longitude value -180, and radius value 500 km.



To adjust for this effect, we can create a simple 3 by 3 grids of the population matrix. The effect can be visualized with a simpler example.

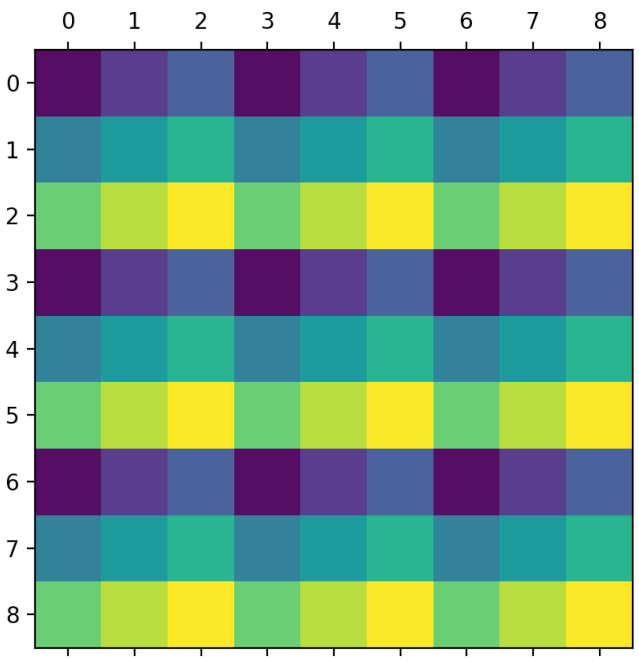
Original Matrix:

1	2	3
4	5	6
7	8	9

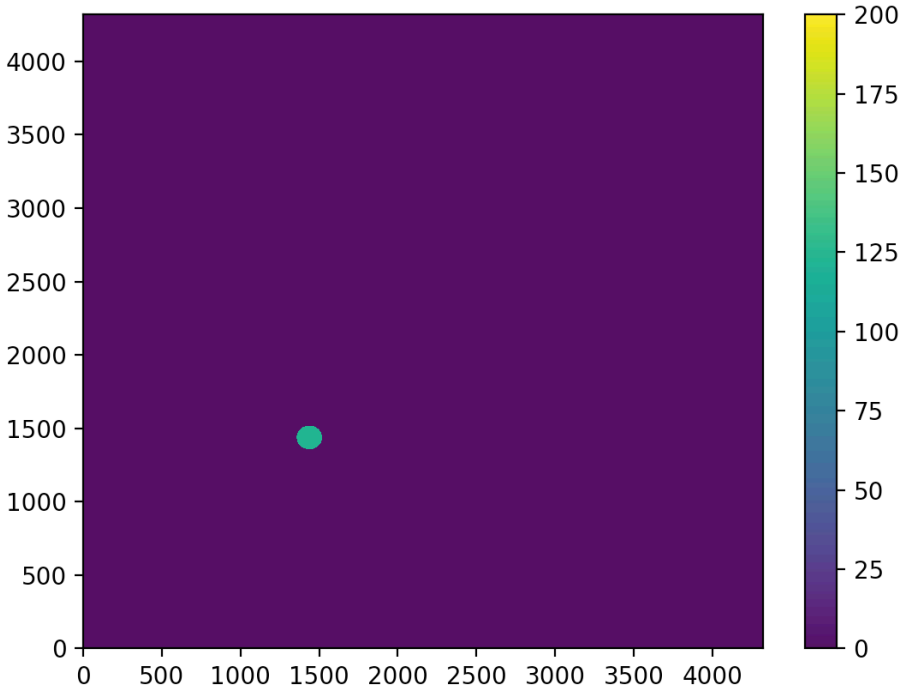


3 x 3 Grid of Matrices:

```
| 1  2  3 | 1  2  3 | 1  2  3 |
| 4  5  6 | 4  5  6 | 4  5  6 |
| 7  8  9 | 7  8  9 | 7  8  9 |
-----------------------------------
| 1  2  3 | 1  2  3 | 1  2  3 |
| 4  5  6 | 4  5  6 | 4  5  6 |
| 7  8  9 | 7  8  9 | 7  8  9 |
-----------------------------------
| 1  2  3 | 1  2  3 | 1  2  3 |
| 4  5  6 | 4  5  6 | 4  5  6 |
| 7  8  9 | 7  8  9 | 7  8  9 |
```



The net effect can be seen in the plot below which shows the mask matrix for latitude value -90, longitude value -180, and radius value 500 km after the spherical adjustment



The population can them be simply taken by summing the population matrix after applying the radius mask

Code:

```
def spherical_adjustment(np_pop):

    np_pop = np.hstack([np_pop] * 3)
    np_pop = np.vstack([np_pop] * 3)

    return np_pop


path_pop_density = 'GPWv4/gpw_v4_population_density_rev10_2015_15_min.asc'

np_population_density = get_density_grid(path_pop_density)
np_population, grid_length = convert_density_to_population(np_population_density)

latitude_grid_count, longitude_grid_count = np_population.shape
total_population = np.sum(np_population)

np_population_adj = spherical_adjustment(np_population)

del np_population
del np_population_density

x = np.arange(0, longitude_grid_count * 3)
y = np.arange(0, latitude_grid_count * 3)


def get_population(latitude, longitude, radius):

    global np_population_adj, total_population, x, y, longitude_grid_count, latitude_grid_count, grid_length

    row_idx, col_idx = coord_to_rc(latitude, longitude)

    cx = col_idx + longitude_grid_count
    cy = row_idx + latitude_grid_count

    r_grid = radius / grid_length

    radius_mask = (x[np.newaxis, :] - cx) ** 2 + (y[:, np.newaxis] - cy) ** 2 < r_grid ** 2
    population = np.sum(np_population_adj[radius_mask])

    return int(min(total_population, population))
```

API Screenshot: