Documentație Proiect - Estimarea Ariilor Geospațiale prin Simularea Monte Carlo: Aplicare la Orașul New York

251: Grigorașcu Andrei Antonio, Bîrsan Gheorghe-Daniel, Șerbănescu George Florin

Contents

1 Introducere

Problema: Estimarea ariei orașului New York (NYC) și a unei regiuni selectate din acesta folosind simularea Monte Carlo. Aceasta implică aproximarea ariilor definite de forme neregulate (poligoane geospațiale). Calculul ariilor pentru geometria complexă reprezintă o problemă frecvent întâlnită în geospațialitate.

Metoda Monte Carlo oferă o abordare simplă, mai ales atunci când metodele tradiționale de integrare sau calcul direct sunt greu de aplicat.

2 Algoritm

1. **Rasterizarea geometriei:**
   - Rasterizarea este procesul de conversie a unei geometrii vectoriale (cum ar fi puncte, linii, poligoane sau alte forme geometrice definite matematic) într-o reprezentare discretizată, sub forma unei grile bidimensionale (sau matrice) de celule, denumite pixeli sau pixeli raster. Fiecare pixel din grilă are o valoare asociată, care indică dacă aparține unei zone de interes sau fundalului, ori conține alte informații relevante (cum ar fi valori numerice reprezentând altitudinea, densitatea, temperatura etc.).
   - Harta folosită pentru calcularea suprafeței este convertită într-o matrice bidimensională (raster).
   - Celulele raster primesc valoarea 1 dacă aparțin ariei de interes și 0 în caz contrar.
2. **Generarea punctelor aleatoare:**
   - Se generează un număr mare de puncte aleatoare (x, y) în limitele matricii generate din hartă.
   - Coordonatele punctelor sunt generate uniform:
$$x \sim \mathcal{U}(\text{minx}, \text{maxx}),$$
$$y \sim \mathcal{U}(\text{miny}, \text{maxy}).$$
3. **Verificarea apartenenței punctelor:**
   - Pentru fiecare punct (x, y), se determină indicele corespunzător în matricea rasterizată:
$$i = \left\lfloor \frac{x - \text{minx}}{\text{pixel\_width}} \right\rfloor,$$
$$j = \left\lfloor \frac{\text{maxy} - y}{\text{pixel\_height}} \right\rfloor.$$
   - Se verifică dacă raster$[j, i] = 1$. Dacă da, punctul este considerat în interiorul regiunii, iar contorul count este incrementat.
4. **Calcularea proporției:**
   - Proporția punctelor aflate în interiorul regiunii este calculată ca:
$$\text{Proporție} = \frac{\text{count}}{\text{num\_samples}}.$$

5. **Estimarea ariei:**

   - Se calculează dimensiunile dreptunghiului delimitator:

$$\text{width} = \text{maxx} - \text{minx},$$
$$\text{height} = \text{maxy} - \text{miny}.$$

   - Aria dreptunghiului este:

$$\text{Arie dreptunghi} = \text{width} \times \text{height}.$$

   - Aria regiunii este estimată prin:

$$\text{Arie regiune} = \text{Proporție} \times \text{Arie dreptunghi}.$$

3 Demonstrație matematică

1. Contextul problemei

Metoda Monte Carlo estimează o valoare A (aria unei regiuni) folosind un eșantion de puncte aleatorii. Fiecare punct este considerat în interiorul regiunii de interes cu probabilitatea $p = \frac{A}{R}$, unde R este aria totală a gridului.

Definim o variabilă indicator I_i astfel:

$$I_i = \begin{cases} 1, & \text{dacă punctul cade în interiorul regiunii;} \\ 0, & \text{altfel.} \end{cases}$$

Variabilele $I_1, I_2, \ldots, I_n$ sunt independente și identic distribuite, cu:

$$\mathbb{E}[I_i] = p, \quad \text{Var}(I_i) = p(1-p).$$

Estimarea Monte Carlo pentru aria regiunii este:

$$\hat{A} = \frac{\text{Numărul de puncte în interior}}{\text{Numărul total de puncte}} \cdot R = \frac{1}{n} \sum_{i=1}^{n} I_i \cdot R.$$

2. Valoarea așteptată a estimării

Valoarea așteptată a estimării $\hat{A}$ este:

$$\mathbb{E}[\hat{A}] = \mathbb{E}\left[\frac{R}{n} \sum_{i=1}^{n} I_i\right] = \frac{R}{n} \cdot n \cdot \mathbb{E}[I_i] = R \cdot p = A.$$

Astfel, $\hat{A}$ este un estimator **nebiased** al valorii reale A.

3. Variabilitatea estimării (Var$(\hat{A})$)

Folosind proprietățile variației, avem:

$$\text{Var}(\hat{A}) = \text{Var}\left(\frac{R}{n} \sum_{i=1}^{n} I_i\right) = \frac{R^2}{n^2} \cdot \text{Var}\left(\sum_{i=1}^{n} I_i\right).$$

Deoarece variabilele I_i sunt independente, varianța sumei este suma variațelor:

$$\text{Var}\left(\sum_{i=1}^{n} I_i\right) = \sum_{i=1}^{n} \text{Var}(I_i) = n \cdot p(1-p).$$

Astfel:

$$\text{Var}(\hat{A}) = \frac{R^2}{n^2} \cdot n \cdot p(1-p) = \frac{R^2 \cdot p(1-p)}{n}.$$

4. Eroarea standard (SE($\hat{A}$))

Eroarea standard este rădăcina pătrată a variației:

$$\text{SE}(\hat{A}) = \sqrt{\text{Var}(\hat{A})} = \sqrt{\frac{R^2 \cdot p(1-p)}{n}}.$$

5. Deducerea numărului de simulări

Pentru o marjă de eroare ϵ și un nivel de încredere $1 - \alpha$, folosim:

$$\epsilon \geq z_{\alpha/2} \cdot \text{SE}(\hat{A}),$$

unde $z_{\alpha/2}$ este valoarea critică a distribuției normale standard.

Înlocuind SE($\hat{A}$), obținem:

$$\epsilon \geq z_{\alpha/2} \cdot \sqrt{\frac{R^2 \cdot p(1-p)}{n}}.$$

Ridicând la pătrat și izolând n, obținem:

$$n \geq \frac{z_{\alpha/2}^2 \cdot R^2 \cdot p(1-p)}{\epsilon^2}.$$

6. Exemplu numeric actualizat

Condiții inițiale

Folosim următoarele date:

- Aria grilei totale: $R = 1000^2$ unități,
- Aria aproximată a New York-ului (rasterizată): $A = 358536$ unități,
- Probabilitatea $p = \frac{A}{R} = \frac{358536}{1000^2} \approx 3.58536 \cdot 10^{-1}$,
- Marja de eroare $\epsilon = 0.01 \cdot A = 3585.36$,
- Nivel de încredere 95%, deci $z_{\alpha/2} = 1.96$.

Calculul numărului minim de simulări

Formula numărului minim de simulări este:

$$n \geq \frac{z_{\alpha/2}^2 \cdot R^2 \cdot p(1-p)}{\epsilon^2}.$$

Înlocuim valorile:

$$p = \frac{358536}{1000^2}$$

Așadar:

$$n \geq \frac{(1.96)^2 \cdot (1000^2)^2 \cdot \frac{358536}{1000^2} \cdot (1 - \frac{358536}{1000^2})}{(3585.36)^2}.$$

Folosind calculele:

$$n \geq \frac{3080310128}{44817} \approx 68730,8416$$

Concluzie

Numărul minim de simulări necesare pentru o marjă de eroare de 1% și un nivel de încredere de 95% este:

$$n \approx 68731.$$

4 Implementare

```
    import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
from rasterio.features import rasterize
from rasterio.transform import from_origin
import geodatasets
from scipy.ndimage import zoom

world = gpd.read_file(geodatasets.get_path('geoda nyc'))


def calculate_square_unit():
    # Calculate how many square kilometers are in a square unit
    nyc_area = 778.18  # Square kilometers (source: Wikipedia)
    nyc_area_units = world.area.sum()
    square_unit_conversion = nyc_area / nyc_area_units
    return square_unit_conversion


def get_area_of_interest():
    print(world['name'])

    input_index = int(input("Enter the index of the area of interest: "))
    if input_index < 0 or input_index >= len(world):
        raise IndexError("Index out of range")
    area_of_interest = world.iloc[[input_index]]
    print(f"Selected area of interest: {area_of_interest['name'].values[0]}")
    return area_of_interest



# Define grid resolution
grid_size = 1000  # 1000x1000 Grid

# Select the northernmost, southernmost, easternmost, and westernmost points
minx, miny, maxx, maxy = world.total_bounds


# Define the raster transform
pixel_width = (maxx - minx) / grid_size
pixel_height = (maxy - miny) / grid_size
# This function creates an affine transformation matrix for a raster dataset.
# This transformation matrix is used to map pixel coordinates to geographic coordinates.
```

```
transform = from_origin(minx, maxy, pixel_width, pixel_height)


def calculate_area(rasterized_matrix, num_samples=68000):
    # Generate random points
    x_samples = np.random.uniform(minx, maxx, num_samples)
    y_samples = np.random.uniform(miny, maxy, num_samples)

    # Count the number of points inside the polygon
    count = 0
    for x, y in zip(x_samples, y_samples):
        i = int((x - minx) / pixel_width)    # Convert x to column index
        j = int((maxy - y) / pixel_height)   # Convert y to row index
        if rasterized_matrix[j, i] == 1:
            count += 1

    # Calculate the area
    width = maxx - minx
    height = maxy - miny

    area = count / num_samples * width * height
    return area


# Map of NYC

# Rasterize: Convert geometries to a 2D matrix
shapes = [(geom, 1) for geom in world.geometry]  # Assign value 1 to all geometries
rasterized_matrix = rasterize(
    shapes=shapes,
    out_shape=(grid_size, grid_size),  # Define output grid size
    transform=transform,
    fill=0  # Background value
    # (0: background, 1: inside the polygon)
)

# Plot the original map
world.plot(cmap="jet", edgecolor="black", column="name")
plt.title('Dataset Map')
plt.show()

# Plot the rasterized matrix
plt.imshow(rasterized_matrix, cmap='Greys', extent=(minx, maxx, miny, maxy))
plt.title("Rasterized Map")
plt.show()

# Run Monte Carlo simulation to calculate the area of NYC
approximated_area = calculate_area(rasterized_matrix)
print()
print(f"Real area of NYC: {world.area.sum()} square units")
```

```
print(f"Approximated Area NYC: {approximated_area:.2f} square units")

print()
print(f"Real area of NYC in square kilometers:
{world.area.sum() * calculate_square_unit():.2f} square kilometers")
print(f"Aproximated Area in square kilometers:
{approximated_area * calculate_square_unit():.2f} square kilometers")


# Area of Interest
area_of_interest = get_area_of_interest()

# Plot the area of interest
area_of_interest.plot()
plt.title('Area of Interest')
plt.show()

# Rasterize the area of interest
shapes = [(geom, 1) for geom in area_of_interest.geometry]
rasterized_matrix_interest = rasterize(
    shapes=shapes,
    out_shape=(grid_size, grid_size),
    transform=transform,
    fill=0
)

# Plot the rasterized matrix
plt.imshow(rasterized_matrix_interest, cmap='Greys', extent=(minx, maxx, miny, maxy))
plt.title("Rasterized Map Interest Area")
plt.show()



# # Run Monte Carlo simulation to calculate the area of the area of interest
approximated_area_of_interest = calculate_area(rasterized_matrix_interest)

print()
print(f"Real area of Interest:
    {area_of_interest.area.sum()} square units")
print(f"Aproximated Area of Interest:
    {approximated_area_of_interest:.2f} square units")

accepted_range = [
    area_of_interest.area.sum() - area_of_interest.area.sum() * 0.01,
    area_of_interest.area.sum() + area_of_interest.area.sum() * 0.01]

if accepted_range[0] <= approximated_area_of_interest <= accepted_range[1]:
    print(f"The approximated area of interest is in the accepted range")
else:
    print(f"The approximated area of interest is not in the accepted range")
```

```
print()
print(f"Real area of Interest in square kilometers:
{area_of_interest.area.sum() * calculate_square_unit():.2f} square kilometers")
print(f"Aproximated Area of Interest in square kilometers:
{approximated_area_of_interest * calculate_square_unit():.2f} square kilometers")
```

5 Concluzii

În urma testării simulării de un număr mai mare de ori (1000), se poate observa că simularea realizată cu numărul de puncte generate ales respectă nivelul de încredere și marja de eroare menționate.

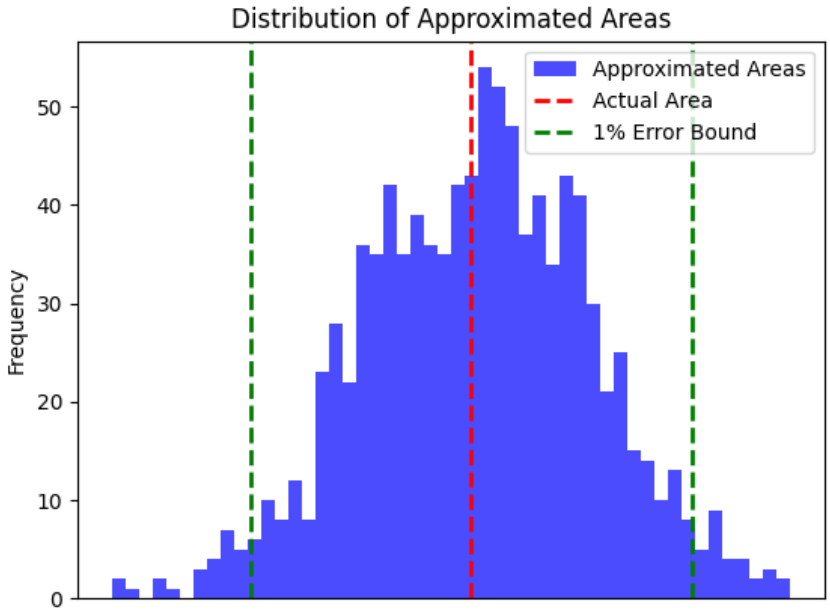

Figure 1: Aproximarea suprafeței orașului New York

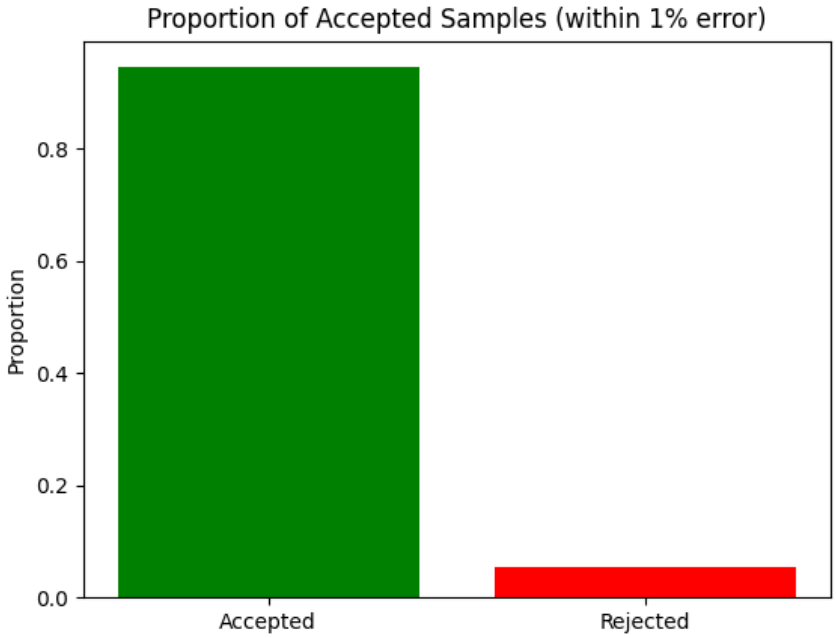

Figure 2: Numărul de simulări care respectă marja de eroare

6 Referințe

- Documentația bibliotecii `geopandas`: `https://geopandas.org/`
- Documentația bibliotecii `rasterio`: `https://rasterio.readthedocs.io/`
- Documentația bibliotecii `scipy`: `https://scipy.org/`