Przeniesienie współrzędnych geodezyjnych na powierzchni elipsoidy obrotowej

Autor: Julia Zapała
Numer indeksu: 325710
Grupa: GI II

1. Dane do zadania

Numer	Szerokość geograficzna φ	Długość geograficzna λ
11	51° 15’ 00”	17° 45’ 00”

2. Cel zadania

Celem zadania jest przeniesienie współrzędnych geograficznych punktu początkowego na elipsoidę obrotową, obliczenie współrzędnych następnych punktów i sprawdzenie czy obliczone punkty stworzą figurę zamkniętą.

3. Wstęp teoretyczny

W geodezji wykorzystuje się wiele modeli matematycznych do opisu kształtu ziemi. Jednym z nich jest elipsoida obrotowa. W przeniesieniu punktów na tym modelu szczególnie pomocny może się okazać algorytm Kivioja i Vincentego. Wykorzystuje się je ze względu na swoją precyzję i wykorzystywanie bardziej złożonych czynników. Pierwszy z algorytmów rozwiązuje zadania wprost, czyli poszukiwanie współrzędnych geodezyjnych i azymutu odwrotnego na podstawie punktu początkowego, długości linii geodezyjnej oraz azymutu wprost. Z kolei drugi algorytm skupia się na zadaniu odwrotnym, czyli znalezieniu długości linii geodezyjnej oraz azymutów wprost i odwrotnego na podstawie współrzędnych punktów.

4. Realizacja zadania

Zadanie zostało wykonane za pomocą programu napisanego w python, z wykorzystaniem algorytmów Vincentego i Kivioja. Do programu zostały wprowadzone dane: punkt początkowy, odległości między punktami i azymuty. Na ich podstawie za pomocą algorytmu Kivioja zostały obliczone pozostałe punkty. Następnie za pomocą biblioteki folium przedstawiono je na mapie. Dodatkowo za pomocą algorytmu Vincentego obliczono odległość i azymut między punktem 4 i 1, a funkcja biblioteki pyproj (inv) umożliwiła obliczenie odległości między punktami 1 i 1*. Na koniec funkcją geometry_area_perimeter() obliczono pole powstałej figury.

5. Wyniki zadania i wnioski

a) *Oblicz współrzędne geodezyjne punktów: 2, 3, 4, z wykorzystaniem algorytmu Kivioja lub z wykorzystaniem bibliotek języka python.*

Współrzędne kolejnych punktów zostały obliczone za pomocą algorytmu Kivioja, a następnie przedstawione w tabeli.

Tabela współrzędnych punktów

Numer punktu	Szerokość geograficzna [° ' "]	Długość geograficzna [° ' "]
1	51 15 0.0	17 45 0.0
2	51 36 34.30649	17 45 0.0
3	51 36 2.36215	19 11 36.08424
4	51 14 28.0537	19 11 36.08424
1*	51 13 56.52521	17 45 41.59599

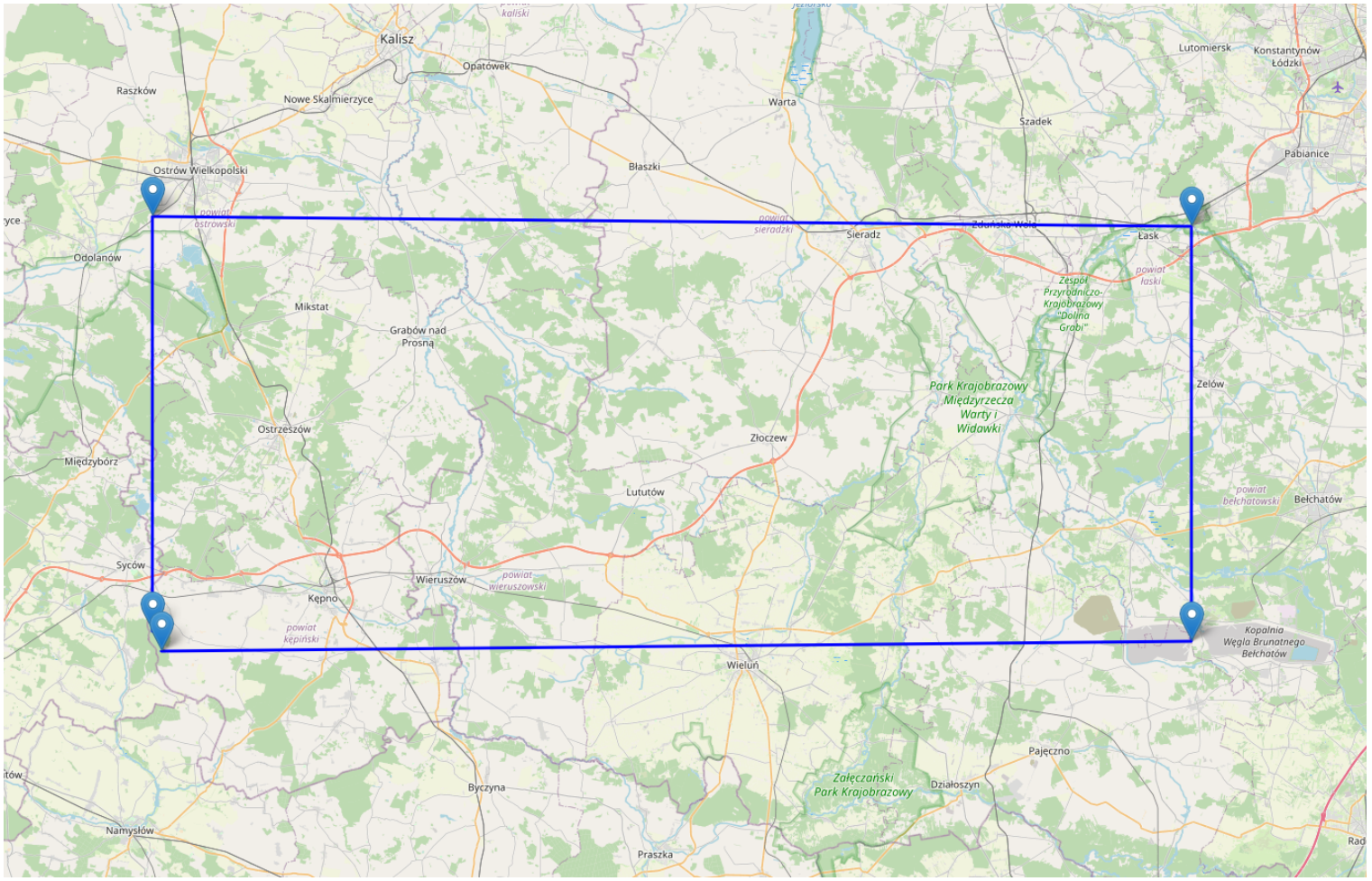
b) Czy po obliczeniu kolejnych wierzchołków 'trapezu', na podstawie podanych obserwacji, otrzymamy figurę zamkniętą? Jaka będzie różnica położenia punktów 1 i 1? Czy spowodowana będzie otrzymana różnica?*

Po obliczeniu kolejnych wierzchołków nie otrzymano figury zamkniętej, położenie punktu 1 i 1* różni się. Odległość między tymi punktami policzono za pomocą funkcji biblioteki pyproj, wyniosła ona ***2,121068 km***. Punkt 1* jest położony bardziej na południowy wschód w porównaniu do punktu 1. Różnica w położeniu tych punktów wynika z zakrzywienia elipsoidy obrotowej.

c) Wyznacz właściwe obserwacje: odległość oraz azymut z punktu 4 do punktu 1

Odległość i azymut z punktu 4 do punktu 1 obliczono za pomocą algorytmu Vincentego. Odległość wynosi **100,787744 km**, a azymut **271° 07' 26.42735"**.

d) Przedstaw na mapie położenie wszystkich punktów



e) Oblicz pole powierzchni powstałej figury

Pole zostało obliczone za pomocą funkcji z biblioteki pyproj, geometry_area_perimeter. Pole powstałej figury, uwzględniającej 5 punktów wyniosło **4114,314455 km²**, a pole czworokąta bez punktu 1* wyniosło **4015,461980 km²**. Przesunięcie punktu 1 widocznie zwiększyło pole powstałej figury.

6. Kod źródłowy

```
import numpy as np
from pyproj import Geod
import folium
from shapely.geometry import Point, Polygon
import tkinter as tk
from tkinter import ttk
import pandas as pd

a = 6378137
e2 = 0.00669438002290

def vincenty(BA,LA,BB,LB):
    b = a * np.sqrt(1-e2)
    f = 1-b/a
    dL = LB - LA
    UA = np.arctan((1-f)*np.tan(BA))
    UB = np.arctan((1-f)*np.tan(BB))
    L = dL
    while True:
        sin_sig = np.sqrt((np.cos(UB)*np.sin(L))**2 +\
                          (np.cos(UA)*np.sin(UB) - np.sin(UA)*np.cos(UB)*np.cos(L))**2)
        cos_sig = np.sin(UA)*np.sin(UB) + np.cos(UA) * np.cos(UB) * np.cos(L)
        sig = np.arctan2(sin_sig,cos_sig)
        sin_al = (np.cos(UA)*np.cos(UB)*np.sin(L))/sin_sig
        cos2_al = 1 - sin_al**2
        cos2_sigm = cos_sig - (2 * np.sin(UA) * np.sin(UB))/cos2_al
        C = (f/16) * cos2_al * (4 + f*(4 - 3 * cos2_al))
        Lst = L
        L = dL + (1-C)*f*sin_al*(sig+C*sin_sig*(cos2_sigm+C*cos_sig*(-1 + 2*cos2_sigm**2)))
        if abs(L-Lst)<(0.000001/206265):
            break
    u2 = (a**2 - b**2)/(b**2) * cos2_al
    A = 1 + (u2/16384) * (4096 + u2*(-768 + u2 * (320 - 175 * u2)))
    B = u2/1024 * (256 + u2 * (-128 + u2 * (74 - 47 * u2)))
    d_sig = B*sin_sig * (cos2_sigm + 1/4*B*(cos_sig*(-1+2*cos2_sigm**2)\
            - 1/6 *B*cos2_sigm * (-3 + 4*sin_sig**2)*(-3+4*cos2_sigm**2)))
    sAB = b*A*(sig-d_sig)
    A_AB = np.arctan2((np.cos(UB) * np.sin(L)),(np.cos(UA)*np.sin(UB) - np.sin(UA)*np.cos(UB)*np.cos(L)))
    A_BA = np.arctan2((np.cos(UA) * np.sin(L)),(-np.sin(UA)*np.cos(UB) + np.cos(UA)*np.sin(UB)*np.cos(L))) + np.pi
    return sAB, A_AB, A_BA

def deg2dms(dd):
    deg = int(np.trunc(dd))
    minutes = int(np.trunc((dd - deg) * 60))
    seconds = round(((dd - deg) * 60 - minutes) * 60, 5)
    dms = [deg, abs(minutes), abs(seconds)]
    return dms
```

```
def Np(B, a=a, e2=e2):
    N = a/(1-e2*(np.sin(B)**2))**0.5
    return N

def Mp(B, a=a, e2=e2):
    M = a * (1 -e2)/((1-e2*np.sin(B)**2)**3)**(0.5)
    return M

distances = [40000, 100000, 40000, 100000]
azimuths = [0, np.deg2rad(90), np.deg2rad(180), np.deg2rad(270)]
phi1 = np.deg2rad(51.25)
lam1 = np.deg2rad(17.75)
```

```
def Kivioj(phi1, lam1, distances, azimuths):
    coordinates = [(np.rad2deg(phi1), np.rad2deg(lam1))]
    for i in range(4):
        s = distances[i]
        Az = azimuths[i]
        n = round(s / 1000)
        ds = s / n
        for j in range(n):
            M_i = Mp(phi1)
            N_i = Np(phi1)
            dphi_i = ds * np.cos(Az) / M_i
            dAz_i = ds * np.sin(Az) * np.tan(phi1) / N_i
            phi_m = phi1 + dphi_i / 2
            Az_m = Az + dAz_i / 2
            M_m = Mp(phi_m)
            N_m = Np(phi_m)
            dphi_i = ds * np.cos(Az_m) / M_m
            dAz_i = ds * np.sin(Az_m) * np.tan(phi_m) / N_m
            dlam_i = ds * np.sin(Az_m) / (N_m * np.cos(phi_m))
            phi1 = phi1 + dphi_i
            Az = Az + dAz_i
            lam1 = lam1 + dlam_i
            if j == n - 1:
                coordinates.append((np.rad2deg(phi1), np.rad2deg(lam1), np.rad2deg(Az)))
    return coordinates

result_coordinates = Kivioj(phi1, lam1, distances, azimuths)
phi_1, lam_1 = result_coordinates[0]
phi_2, lam_2, Az_2 = result_coordinates[1]
phi_3, lam_3, Az_3 = result_coordinates[2]
phi_4, lam_4, Az_4 = result_coordinates[3]
phi_1_prim, lam_1_prim, Az_1_prim = result_coordinates[4]
punkty = [(phi_1, lam_1), (phi_2, lam_2), (phi_3, lam_3), (phi_4, lam_4), (phi_1_prim, lam_1_prim)]

g = Geod(ellps = 'WGS84')

def plot_coordinates_on_map(coordinates):
    center = [coordinates[0][0], coordinates[0][1]]
    m = folium.Map(location=center, zoom_start=12)
    for i, coord in enumerate(coordinates, start=1):
        popup_text = '1*' if i == 5 else f'Point {i}'
        folium.Marker([coord[0], coord[1]], popup=popup_text).add_to(m)
    folium.PolyLine(locations=[[coord[0], coord[1]] for coord in coordinates], color='blue').add_to(m)
    return m

map_coordinates = Kivioj(phi1, lam1, distances, azimuths)
map_obj = plot_coordinates_on_map(map_coordinates)
map_obj.save('map1.html')

data = {
    'Numer punktu': ['1', '2', '3', '4', '1*'],
    'Szerokość geograficzna [° \' \"] ': [deg2dms(phi_1), deg2dms(phi_2), deg2dms(phi_3), deg2dms(phi_4), deg2dms(phi_1_prim)],
    'Długość geograficzna [° \' \"]': [deg2dms(lam_1), deg2dms(lam_2), deg2dms(lam_3), deg2dms(lam_4), deg2dms(lam_1_prim)]
}
df = pd.DataFrame(data)
window = tk.Tk()
window.title("Tabela współrzędnych punktów")
tree = ttk.Treeview(window)
tree["columns"] = tuple(df.columns)
tree["show"] = "headings"
for column in df.columns:
    tree.heading(column, text=column)
for i, row in df.iterrows():
    tree.insert("", "end", values=tuple(row))
tree.pack(expand=True, fill="both")
window.mainloop()

odleglosc, azymut, azymut_odwrotny  = vincenty(np.deg2rad(phi_4), np.deg2rad(lam_4), np.deg2rad(phi_1), np.deg2rad(lam_1))
print('Odległość z punktu 4 do punktu 1: {:.6f} km'.format(odleglosc / 1000))
azymut1 = np.deg2rad(360) + azymut
print('Azymut z punkt 4 do punktu 1:', (deg2dms(np.rad2deg(azymut1))))

r = g.inv(np.deg2rad(lam_1), np.deg2rad(phi_1), np.deg2rad(lam_1_prim), np.deg2rad(phi_1_prim), radians=True)
print('Różnica położenia punktów 1 i 1* wynosi: {:.6f} km'.format(r[2]/1000))

punkty_odwrotne = punkty[::-1]
point_objects = [Point(lon, lat) for lat, lon in punkty_odwrotne]
polygon = Polygon(point_objects)
area, perimeter = g.geometry_area_perimeter(polygon)
area = area / 1000000
print('Pole powstałej figury: {:.6f} km²'.format(area))
```