Zadanie 3 - Przeniesienie współrzednych geodezyjnych na powierzchnię elipsoidy obrotowej

- Autor: Adrian Fabisiewicz
- Numer indeksu: 328935
- Numer: 4
- φ_1 = 51°00′00.00000″
- λ_1 = 19°00′00.00000″

Cel ćwiczenia

Celem ćwiczenia jest przeniesienie współrzędnych geodezyjnych na powierzchni elipsoidy obrotowej. Na podstawie danej szerokości i długości geograficznej punktu pierwszego, a także danych azymutów oraz długości trzech linii geodezyjnych wyliczyć współrzędne punktów 2, 3 i 4 na powierzchni elipsoidy obrotowej. Do obliczeń wykorzystano algorytm Kivioja. Należało stwierdzić, czy figura po obliczeniu wszystkich współrzędnych się zamknie oraz porównać różnice współrzędnych po ich przeniesieniu. Ostatnim etapem było przedstawienie figury, stworzonej przez obliczone punkty, na mapie, a także obliczenie jej pola powierzchni z wykorzystaniem biblioteki *pyproj*.

Dane do zadania

Dany były współrzędne geodezyjne punktu początkowego P_1, wynoszące:

- φ_1 = 51°00′00.00000″
- λ_1 = 19°00′00.00000″

Podano również azymuty oraz długości trzech lini geodezyjnych:

	długość *s* [km]	**azymut *A* [°]**
1-2	40	0°00′00.000
2-3	100	90°00′00.000
3-4	40	180°00′00.000
4-1*	100	270°00′00.000

Kolejność wykonywania obliczeń

Zapisanie danych używanych w zadaniu

In [ ]:
```
import numpy as np

nr = 4
```

```
point_1_phi_rad = np.deg2rad(50 + nr * 15/60)
point_1_lam_rad = np.deg2rad(18 + nr * 15/60)

s = [40000, 100000, 40000, 100000] #odległości w metrach kolejno 1-2, 2-3, 3-4, 4-1
az = np.deg2rad([0, 90, 180, 270]) #azymuty w radianach kolejno 1-2, 2-3, 3-4, 4-1*
```

Implementacja funkcji, realizującej algorytm Kivioja

Funkcja jako argumenty przyjmuje szerokość i długość geograficzną pierwszego punktu oraz długość i azymut linii geodezyjnej. Akceptuje dane w radianach oraz metrach. Zwraca natomiast szerokość i długość geograficzną kolejnego punktu, a także azymut odwrotny linii w radianach.

In [ ]:
```
a = 6378137
e2 = 0.00669438002290

def Np(phi):
    N = a/np.sqrt(1-e2*np.sin(phi)**2)
    return N

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

def kivioj(phi, lam, s, az):
    n = round(s / 1000)
    ds = s/n

    for i in range(n):
        N_i = Np(phi)
        M_i = Mp(phi)

        dphi_i = ds*np.cos(az)/M_i
        dA_i = ds*np.sin(az)*np.tan(phi)/N_i

        phi_im = phi + dphi_i/2
        az_im = az + dA_i/2

        N_i = Np(phi_im)
        M_i = Mp(phi_im)

        dphi_i = ds*np.cos(az_im)/M_i
        dA_i = ds*np.sin(az_im)*np.tan(phi_im)/N_i
        dlam_i = ds*np.sin(az_im)/N_i/np.cos(phi_im)

        phi = phi + dphi_i
        lam = lam + dlam_i
        az = az + dA_i

    az_odw = (az + np.pi) % (2*np.pi)
    return phi, lam, az_odw
```

Obliczenie kolejnych wierzchołków, z wykorzystaniem utworzonej funkcji

Poniższa pętla przelicza współrzędne każdego punktu na elipsoidzie oraz zapisuje je w tablicy.

In [ ]:
```
# s = [40000, 100000, 40000, 100000]
# az = np.deg2rad([0, 90, 180, 270])

current_phi = point_1_phi_rad
current_lam = point_1_lam_rad

new_points_rad = []

for p in range(len(s)):
    current_phi, current_lam, az_odw = kivioj(current_phi, current_lam, s[p], az[p]
    new_points_rad.append([current_phi, current_lam, az_odw])

new_points_rad = np.array(new_points_rad)
new_points_deg = np.rad2deg(new_points_rad)

print (*new_points_deg, sep='\n')
```

```
[ 51.35954501  19.          180.        ]
[ 51.35075017  20.43548958 271.12118602]
[50.99120462 20.43548958   0.        ]
[50.98252402 19.01138737 88.89344757]
```

In [ ]:
```
def rad2dms(rad):
    dd = np.rad2deg(rad)
    dd = dd
    deg = int(np.trunc(dd))
    mnt = int(np.trunc((dd-deg) * 60))
    sec = ((dd-deg) * 60 - mnt) * 60
    dms = [deg, abs(mnt), abs(sec)]
    return dms

def angle_formatter(degrees, minutes, seconds):
    return f'{degrees}°{minutes:02d}\'{seconds:0.5f}\"'

point_2_phi_d, point_2_phi_m, point_2_phi_s = rad2dms(new_points_rad[0][0])
point_2_lam_d, point_2_lam_m, point_2_lam_s = rad2dms(new_points_rad[0][1])

print(f'Współrzędne punktu 2 wynoszą {angle_formatter(point_2_phi_d, point_2_phi_m,

point_3_phi_d, point_3_phi_m, point_3_phi_s = rad2dms(new_points_rad[1][0])
point_3_lam_d, point_3_lam_m, point_3_lam_s = rad2dms(new_points_rad[1][1])

print(f'Współrzędne punktu 3 wynoszą {angle_formatter(point_3_phi_d, point_3_phi_m,

point_4_phi_d, point_4_phi_m, point_4_phi_s = rad2dms(new_points_rad[2][0])
point_4_lam_d, point_4_lam_m, point_4_lam_s = rad2dms(new_points_rad[2][1])

print(f'Współrzędne punktu 4 wynoszą {angle_formatter(point_4_phi_d, point_4_phi_m,

point_1g_phi_d, point_1g_phi_m, point_1g_phi_s = rad2dms(new_points_rad[3][0])
point_1g_lam_d, point_1g_lam_m, point_1g_lam_s = rad2dms(new_points_rad[3][1])

print(f'Współrzędne punktu 1* wynoszą {angle_formatter(point_1g_phi_d, point_1g_phi
```

```
Współrzędne punktu 2 wynoszą 51°21'34.36205" 19°00'0.00000".
Współrzędne punktu 3 wynoszą 51°21'2.70063" 20°26'7.76249".
Współrzędne punktu 4 wynoszą 50°59'28.33662" 20°26'7.76249".
Współrzędne punktu 1* wynoszą 50°58'57.08648" 19°00'40.99454".
```

punkt	φ	λ
2	51°21'34.36205"	19°00'0.00000"
3	51°21'2.70063"	20°26'7.76249"

punkt	φ	λ
4	50°59'28.33662"	20°26'7.76249"
1*	50°58'57.08648"	19°00'40.99454"

Obliczenie różnicy położenia między punktem 1 i 1*

Na podstawie otrzymanych wyników da się zauważyć, że zwrócone przez funkcję *kivioj* współrzędne punktu 1* różnią się od współrzędnych, danych na początku w zadaniu. Różnicę tę można obliczyć z wykorzystaniem algorytmu Vincentego, którego implementacja została dołączona wraz z instrukcją do zadania.

In [ ]:
```
def vincenty(BA,LA,BB,LB):
    '''
    Parameters
    ----------
    BA : szerokosc geodezyjna punktu A [RADIAN]
    LA : dlugosc geodezyjna punktu A [RADIAN]
    BB : szerokosc geodezyjna punktu B [RADIAN]
    LB : dlugosc geodezyjna punktu B [RADIAN]

    Returns
    -------
    sAB : dlugosc linii geodezyjnej AB [METR]
    A_AB : azymut linii geodezyjnej AB [RADIAN]
    A_BA : azymut odwrotny linii geodezyjne [RADIAN]
    '''
    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))
        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_si
        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)*
    A_BA = np.arctan2((np.cos(UA) * np.sin(L)),(-np.sin(UA)*np.cos(UB) + np.cos(UA)
    return sAB, A_AB, A_BA
```

In [ ]:
```
s_1_1g, az_1_1g, az_1g_1 = vincenty(point_1_phi_rad, point_1_lam_rad, new_points_ra
```

```
print(f'Odległość pomiędzy punktami 1 i 1* wynosi {s_1_1g:.3f}m.')
```

```
Odległość pomiędzy punktami 1 i 1* wynosi 2102.147m.
```

Wyznaczenie odległości oraz azymutu z punktu 4 do punktu 1

In [ ]:

```
def rad2dms(rad):
    dd = np.rad2deg(rad)
    dd = dd
    deg = int(np.trunc(dd))
    mnt = int(np.trunc((dd-deg) * 60))
    sec = ((dd-deg) * 60 - mnt) * 60
    dms = [deg, abs(mnt), abs(sec)]
    return dms

s_4_1, az_4_1, az_1_4 = vincenty(new_points_rad[2][0], new_points_rad[2][1], point_
az_4_1 = (az_4_1) % (2*np.pi)
d, m, s = rad2dms(az_4_1)

print(f'Odległość z punktu 4 do punktu 1 wynosi {s_4_1:0.3f} m, czyli {s_4_1/1000:0
print(f'Azymut z punktu 4 do punktu 1 wynosi {d}°{m:02d}\'{s:0.5f}\".')
```

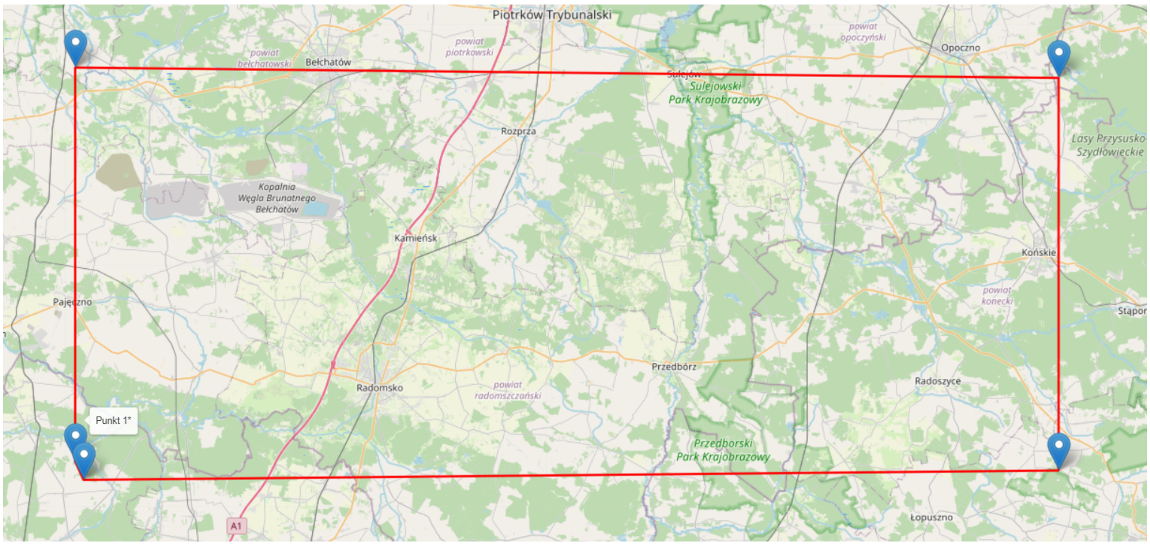
```
Odległość z punktu 4 do punktu 1 wynosi 100780.718 m, czyli 100.780718 km.
Azymut z punktu 4 do punktu 1 wynosi 271°06'50.47059".
```

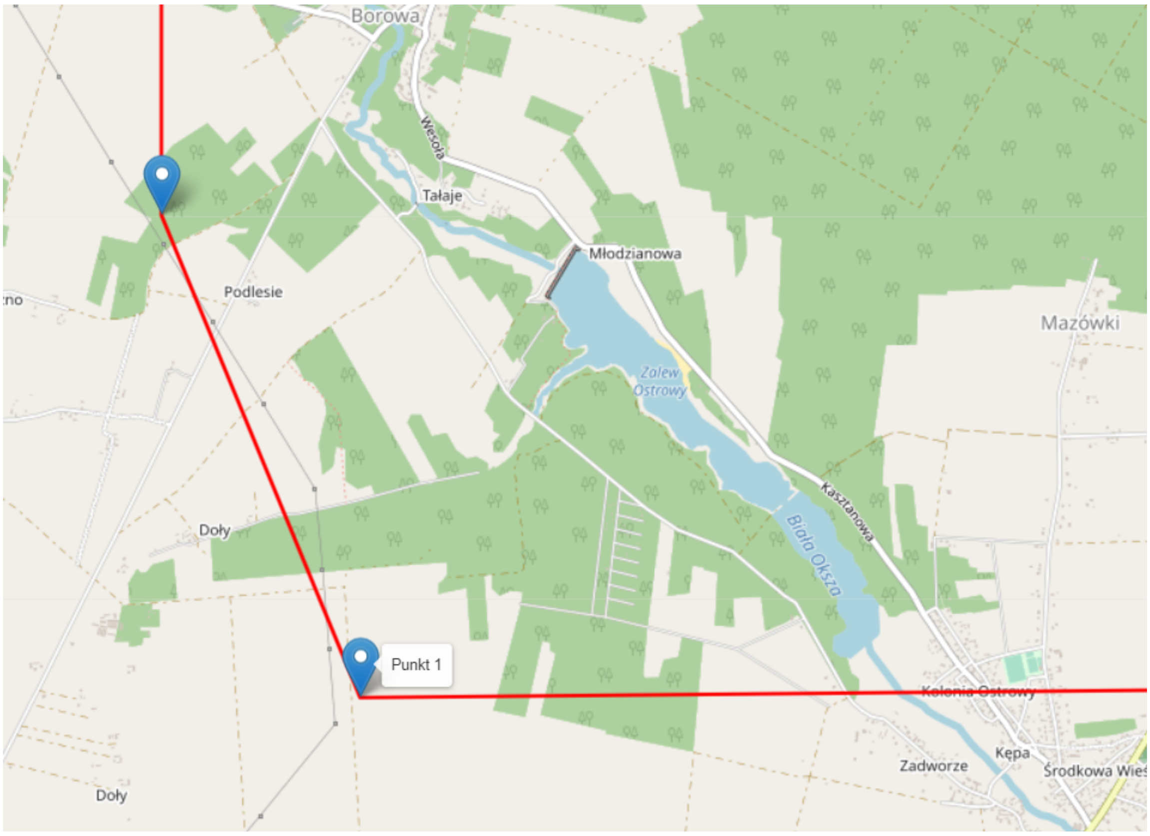
Wizualizacja położenia wszystkich punktów

Z wykorzystaniem biblioteki folium zwizualizowano położenie punktów 1*, 1, 2, 3 oraz 4. Na utworzonej mapie widać różnicę w położeniu punktów 1 oraz 1*.

In [ ]:

```
import folium

m = folium.Map(location=[51, 19], zoom_start=9)

points = []

for i in range(len(new_points_deg)):
    point_label = (i + 1) % len(new_points_deg) + 1
    points.append([new_points_deg[i][0], new_points_deg[i][1]])
    folium.Marker(points[i], tooltip=f'Punkt {point_label} {new_points_deg[i][0]} {

points.append([np.rad2deg(point_1_phi_rad), np.rad2deg(point_1_lam_rad)])
folium.Marker(points[-1], tooltip=f'Punkt 1* {points[-1][0]} {points[-1][1]}').add_

folium.Polygon(points, color='red', weight=2.5).add_to(m)

m.save('mapa.html')
```





Obliczenie pola powierzchni powstałej figury

Z wykorzystaniem biblioteki *pyproj*, a dokładniej modułu Geod, obliczono pole powierzchni figury, powstałej z połączenia wszystkich punktów. Użyto do tego funkcji *geometry_area_perimeter*, wcześniej tworząc polygon z punktów z użyciem biblioteki *shapely*.

In [ ]:
```
from pyproj import Geod
from shapely.geometry import Polygon

geod = Geod(ellps='GRS80')
poly = Polygon(points)

lons = points[0][1], points[1][1], points[2][1], points[3][1], points[4][1]
lats = points[0][0], points[1][0], points[2][0], points[3][0], points[4][0]

area = -geod.polygon_area_perimeter(lons, lats)[0]
```

```
print(f'Pole powierzchni wynosi {area:.6f} m^2, czyli {area/1000000:.12f} km^2.')
```

```
Pole powierzchni wynosi 4113295698.406223 m^2, czyli 4113.295698406223 km^2.
```

Wniosek

Figura utworzona przez obliczone współrzędne punktów na elipsoidzie nie zamyka się. Współrzędne 1 i 1* są różne. Jest między nimi ponad 2km różnicy. Wynika to z tego, że elipsoida obrotowa nie jest dokładnym modelem Ziemi.