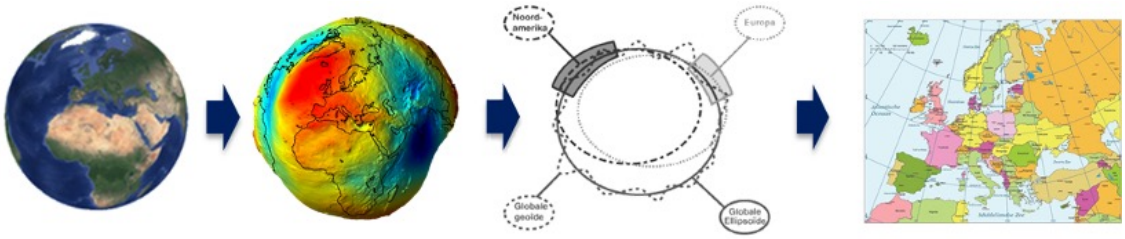
02_02_PyProj

December 21, 2021

1 Projectiesystemen en coördinaatstransformaties

De bedoeling van projectiesystemen is om (een gedeelte van) de aarde op een plat vlak voor te kunnen stellen. In feite dus de vertaling van het datum met geodetische coördinaten naar vlakke coördinaten.



Het vraagstuk van projectiesystemen situeert zich rond de modellering van onze complexe aarde via een zwaartekrachtmodel of geoïde. Op basis van dit onregelmatige oppervlak wordt een globale of lokale ellipsoïde geprojecteerd. Deze wordt vervolgens in 3D gepositioneerd en georiënteerd, wat resulteerd in een (wiskudig te beschrijven) datum. Op dit datum worden coördinaten uitgedrukt door lengte- en breedteliggingen in graden (*geographic coordinate system*). Tot slot worden deze sferische coördinaten geprojecteerd of afgebeeld op een plat vlak volgens bepaalde transformatieformules. Zowel de resulterende cartesiaanse coördinaten (in een *projected coordinate system*) als de sherische coördinaten laten ondubbelzinnige plaatsbepaling toe.

De ondubbelzinnige definiëring van geometrische eigenschappen in absolute termen is direct ook de hoofddoelstelling van projectiesystemen. Om gegeven coordinaten in systeem A uit te drukken in systeem B gebruiken we coördinaattransformaties.

Deze tutorial is een vertaling van de getting started guide op https://pyproj4.github.io

1.1 Coördinaattransformaties met PyProj

Het doel van `PROJ4` is om coördinaten te transformeren van het ene coördinaatreferentiesysteem naar het andere. Standaard zijn dergelijke operaties mogelijk met behulp van de C API, ofwel door gebruik te maken van enkele programma's die vanuit de opdrachtprompt uitgevoerd kunnen worden, en gebaseerd zijn op deze API.

Er bestaat echter ook een Python-*binder* die het toe laat om coördinaattransformaties uit te voeren met dezelfde code. In deze notebook zal het gebruik van deze binder, genaamd `PyProj`, geïlustreerd worden.

Voor meer informatie over PROJ4 verwijzen we door naar de website. Specifiek voor PyProj kan meer informatie op deze website teruggevonden worden.

1.2 Een coördinaatreferentiesysteem als CRS-object

Een coördinaatreferentiesysteem wordt omschreven door een CRS-object, dat aangemaakt worden door de pyproj.crs.CRS-klasse aan te spreken. Het Belgische Lambert '08 CRS (EPSG:3812) kunnen we op verschillende manieren initialiseren:

[ ]:
```
from pyproj import CRS
crs = CRS.from_epsg(3812)
crs = CRS.from_string('epsg:3812')
crs = CRS.from_proj4('+proj=lcc +lat_1=49.83333333333334 +lat_2=51.
 ↪16666666666666 \
                     +lat_0=50.797815 +lon_0=4.359215833333333 +x_0=649328 \
                     +y_0=665262 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m␣
 ↪+no_defs')
crs = CRS.from_user_input(3812)
```

1.2.1 CRS-objecten converteren naar verschillende formaten

Dankzij de EPSG-code beschikken we over de juiste parameters van een bepaald CRS. Deze parameters kunnen op verschillende manieren geëxporteerd worden, wat relevant kan zijn als een bepaalde softeware hier bijvoorbeeld om vraagt. De verschillende parameters kunnen met behulp van deze EPSG-codes opgevraagd worden in verschillende software waarin gewerkt kan worden met geografische objecten. Een handige website voor het achterhalen van deze parameters is https://epsg.io. We kunnen de verwijzingen naar een bepaald CRS eenvoudig opvragen:

[ ]:
```
from pyproj import CRS
crs = CRS.from_epsg(3812)
print("crs.to_epsg(): %s" % (crs.to_epsg()))
print("crs.to_authority(): %s" % (str(crs.to_authority())))
crs = CRS.from_proj4('+proj=lcc +lat_1=49.83333333333334 +lat_2=51.
 ↪16666666666666 \
                     +lat_0=50.797815 +lon_0=4.359215833333333 +x_0=649328 \
                     +y_0=665262 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m␣
 ↪+no_defs')
print(crs)
```

Ook kunnen we de parameters van een CRS printen in het *Well-known text*-formaat:

[ ]:
```
# Methode 1
print(crs.to_wkt(pretty=True))

# Methode 2
from pyproj.enums import WktVersion
print(crs.to_wkt(WktVersion.WKT1_GDAL, pretty=True))
```

```
# Methode 3
from pprint import pprint
pprint(crs.to_cf())
```

Bij het converteren van de parameters van een CRS van het ene naar het andere formaat dienen we bedachtzaam te zijn voor de mogelijkheid dat er nuttige informatie verloren kan gaan.

1.2.2 Attributen uit een CRS ophalen

Uit een `pyproj.crs.CRS`-object kunnen we een groot aantal attributen halen. We illustreren een aantal zaken aan de hand van het Belgische Lambert '08 CRS met inbegrip van de Tweede Algemene Waterpassing (TAW, Ostend Height):

[ ]:
```
crs = CRS("urn:ogc:def:crs,crs:EPSG::3812,crs:EPSG::5710")
crs
```

Met de `sub_crs_list`-attribuut kunnen we bekijken uit welke sub-systemen dit CRS is opgebouwd:

[ ]:
```
crs.sub_crs_list
```

Tot slot bestuderen we de transformatieparameters:

[ ]:
```
cop = crs.sub_crs_list[0].coordinate_operation
print('Code van de transformatie: %s' % cop.method_code)
print('Naam van de transformatie: %s' % cop.method_name)
print('-----------')
print('Parameters van de transformatie: %s' % str(cop.params))
print('-----------')
print(cop.to_wkt(pretty=True))
```

1.3 De juiste UTM-zone zoeken op basis van coördinaten

Een handige tool binnen `PyProj` is de mogelijkheid om voor een gegeven datum en `AreaOfInterest` de bijbehorende UTM-zone te berekenen. Uiteraard kunnen we dit zelf berekenen, maar we kunnen ook gebruik maken van de `query_utm_crs_info`-methode van het `pyproj.database`-object. Meer informatie over dit object is hier terug te vinden.

[ ]:
```
from pyproj import CRS
from pyproj.aoi import AreaOfInterest
from pyproj.database import query_utm_crs_info

utm_crs_list = query_utm_crs_info(
    datum_name=str("ETRS 89"),
    area_of_interest=AreaOfInterest(
        west_lon_degree=3,
        south_lat_degree=50,
        east_lon_degree=4,
        north_lat_degree=51,
    ),
```

```
)
utm_crs = CRS.from_epsg(utm_crs_list[0].code)
utm_crs
```

1.4 Transformaties van het ene naar het andere CRS

In de volgende sectie van deze *notebook* zullen we demonstreren hoe we coördinaten om kunnen zetten van WGS'84 (EPSG:4326) naar het Belgische Lambert '08 CRS (EPSG:3812).

1.4.1 Stap 1: Inspecteer de CRS definitie

We moeten er in eerste instantie zeker van zijn dat de *area of use* en de volgorde van de assen correct zijn. Hiervoor verwijzen we terug naar de vorige secties.

[ ]:
```
from pyproj import CRS
print('--------- WGS84 ---------')
crs_4326 = CRS.from_epsg(4326)
pprint(crs_4326)
print('--------- LB08 ---------')
crs_3812 = CRS.from_epsg(3812)
pprint(crs_3812)
```

Merk op dat bij `crs_4326` eerst de as voor de latitude (N-Z) vermeld wordt, terwijl bij `crs_3812` eerst de as voor de *easting* (W-O) gegeven wordt. Dit betekent dat we bij de transformatie eerst de latitude (breedteligging) en dan de longitude (lengteligging) moeten geven. Merk ook op dat het Belgische Lambert '08 CRS begrenst is door `(2.5, 49.5, 6.4, 51.51)`, wat correspondeerd met `(min_x, min_y, max_x, max_y)`. Deze begrenzing impliceert dat onze te converteren coördinaten binnen dit gebied moeten liggen. Indien niet hebben we ofwel een slecht resultaat, ofwel zullen we een foutmelding krijgen.

1.4.2 Step 2: `Transformer`-object aanmaken om de conversie uit te voeren

We initialiseren een `pyproj.transformer.Transformer`-object waarmee de transformatie uitgevoerd zal worden. De attributen die we aan deze transformatie meegeven kunnen dezelfde vormen hebben als we zojuist beproken hebben. Meer informatie vinden we hier terug.

[ ]:
```
from pyproj import Transformer
transformer = Transformer.from_crs(crs_4326, crs_3812)
transformer = Transformer.from_crs(4326, 3812)
transformer = Transformer.from_crs("EPSG:4326", "EPSG:3812")
transformer
```

De uiteindelijke transformatie van een punt, bijvoorbeeld de coördinaten van Gent `(lat: 51.053581; lon: 3.722969)`, voegen we als volgt uit:

[ ]:
```
transformer.transform(51.053581, 3.722969)
```

Indien gewenst kunnen we de `Transformer` forceren om voor de volgorde van de coordinaten altijd `(x, y)` of `(lon, lat)`aan te houden. Hiervoor voegen we de `always_xy`-optie toe aan de `Transformer`:

```
[ ]: from pyproj import Transformer
transformer = Transformer.from_crs("EPSG:4326", "EPSG:3812", always_xy=True)
transformer.transform(3.722969, 51.053581)
```

1.5 Transformaties van geprojecteerde- naar geografische- coordinaten binnen hetzelfde datum

We kunnen geprojecteerde coördianten eenvoudig omzetten naar geografische coördinaten (en andersom). Dit wordt hieronder geïllustreerd aan de hand van de conversie van coördinaten in het Belgische Lambert '08 CRS naar ETRS'89.

1.5.1 Stap 1: Geodetisch CRS ophalen uit het originele CRS

We starten met de extractie van het geodetisch CRS uit het originele CRS:

```
[ ]: from pyproj import CRS
from pyproj import Transformer
print('--------- LB08 ---------')
crs = CRS.from_epsg(3812)
pprint(crs_3812)
print('--------- ETRS89 ---------')
pprint(crs.geodetic_crs)
```

1.5.2 Stap 2: `Transformer`-object aanmaken voor de transformatie

Gelijkaardig aan wat we hierboven gedeminstreerd hebben, maken we opnieuw een `Transformer`-object aan. In dit geval geven we echter het geodetische CRS mee als tweede parameter:

```
[ ]: proj = Transformer.from_crs(crs, crs.geodetic_crs)
proj
```

Tot slot voeren we de transformatie uit voor een gegeven cartesiaanse coördinaat:

```
[ ]: proj.transform(604717.90, 693905.17)
```

1.6 4D transformaties met tijd

De `PyProj`-bibliotheek laat toe om tijdsafhankelijke coördinaattransformaties uit te voeren. Dit is vooral interessant om oude coordinaten terug te rekenen, bijvoorbeeld van ITRF2014 (EPSG:7789) naar ETRF2014 (EPSG:8401). Deze systemen worden geregeld geüpdate, onder meer omwille van veranderingen veroorzaakt worden door plaattektoniek.

```
[ ]: from pyproj import Transformer
transformer = Transformer.from_crs(7789, 8401)
print(transformer.transform(51.053581, 3.722969, tt=1985.0))
print(transformer.transform(51.053581, 3.722969, tt=1990.0))
print(transformer.transform(51.053581, 3.722969, tt=1995.0))
print(transformer.transform(51.053581, 3.722969, tt=2000.0))
print(transformer.transform(51.053581, 3.722969, tt=2019.0))
```

1.6.1 Geodetische berekeningen

Om afstanden tussen twee punten te berekenen of de oppervlakte van een polygoon te berekenen maken we doorgaans gebruik van geprojecteerde coördinaten volgens een gegeven CRS. Afhankelijk van het gekozen CRS zal er een projecttiefout optreden, die zeker over grote afstanden of voor grote oppervlakten onacceptabele proporties aan kan nemen. Om dit probleem het hoofd te bieden, kunnen we gebruik maken van geodetische berekenen, waarbij lengten en oppervlakten berekend worden volgens de kromming van het oppervlak van de aarde. In onderstaande voorbeeld zullen we dit concept illustreren aan de hand van de afstand tussen Oostende en Luik, alsook de oppervlakte die gevormd wordt door de driehoek Oostende-Luik-Arlon. De berekende afstanden en oppervlakten zullen we vergelijken met de afstanden verkregen via cartesiaanse coördinaten volgens het Belgische Lambert '08 CRS.

Plaats	breedte (graden)	lengte (graden)
Oostende	51.215	2.851
Luik	50.625	5.529
Arlon	49.675	5.725

Meer informatie over het gebruik van het `pyproj.Geod`-object, alsook de bijbehorende berekeningen en methodes, kan terug gevonden worden in de handleiding.

1.6.2 Een `Geod`-object aanmaken

Een `Geod`-object wordt aangemaakt met een verwijzing naar de referentie-ellipsoïde, en bevat enkel de grote as (a) en afplattingscoëfficient (f):

[ ]:
```
from pyproj import CRS, Geod
geod_grs80 = Geod(ellps='GRS80')
geod_lb08 = CRS("epsg:3812").get_geod()
geod_lb08
```

1.6.3 Geodetische lengte

De geodetische lengte tussen Oostende en Luik berekenen we als volgt:

[ ]:
```
from pyproj import Geod
lats = [51.215, 50.625]
lons = [2.851, 5.529]
geoLength = geod_lb08.line_length(lons, lats)
print('%.3f m' % geoLength)
```

Dezelfde lengte vinden we eveneens terug wanneer we de `Shapely`-bibliotheek gebruiken om deze punten eerst om te zetten naar `Points` en vervolgens naar een `LineString`:

[ ]:
```
from pyproj import Geod
from shapely.geometry import Point, LineString
pOostende_geo = Point(2.851, 51.215)
pLuik_geo = Point(5.529, 50.625)
```

```
line_string = LineString([pOostende_geo, pLuik_geo])
geoLength = geod_lb08.geometry_length(line_string)
print('%.3f m' % geoLength)
```

Wanneer we het eerder besproken `Transformer`-object nog eens toepassen op bovenstaande coördinaten, en deze omzetten van geografische WGS'84 coördinaten naar geprojecteerde Belgische Lambert '08 coördinaten, krijgen we een afwijkende lengte:

[ ]:
```
from pyproj import Transformer
import math
transformer = Transformer.from_crs("EPSG:4326", "EPSG:3812", always_xy=True)
pOostende_proj = Point(transformer.transform(pOostende_geo.x, pOostende_geo.y))
pLuik_proj = Point(transformer.transform(pLuik_geo.x, pLuik_geo.y))
line_string = LineString([pOostende_proj, pLuik_proj])
print('%.3f m' % line_string.length)
```

1.6.4 Geodetische oppervlakte en omtrek

Het berekenen van de oppervlakte en omstrek van een vlak, beschreven door 3 of meerdere punten verloopt volgens een gelijkaardige methode. Vanuit het `Geod`-object gebruiken we de `polygon_area_perimeter`-methode:

Merk op dat het teken van de oppervlakte afhankelijk is van de volgorde waarin de coördinaten aangeboden worden. Oostende-Luik-Arlon (wijzerzin) zal een negatieve oppervlakte geven.

[ ]:
```
from pyproj import Geod
geod_lb08 = CRS("epsg:3812").get_geod()
lats = [51.215, 49.675, 50.625]
lons = [2.851, 5.725, 5.529]
area, perimeter = geod_lb08.polygon_area_perimeter(lons, lats)
print('Oppervlakte: %.3f km² ; Omtrek: %.3f km' % (area / 1000000, perimeter /␣
↪1000))
```

Opnieuw kunnen we deze berekeningen ook uitvoeren op een `Shapely`-object. In dit geval definiëren we een `Polygon`-object:

[ ]:
```
from pyproj import Geod
from shapely.geometry import LineString, Point, Polygon
geod_lb08 = CRS("epsg:3812").get_geod()
pOostende_geo = Point(2.851, 51.215)
pLuik_geo = Point(5.529, 50.625)
pArlon_geo = Point(5.725, 49.675)
area, perimeter = geod_lb08.geometry_area_perimeter(
    Polygon(LineString([pOostende_geo, pArlon_geo, pLuik_geo]))
)
print('Oppervlakte: %.3f km² ; Omtrek: %.3f km' % (area / 1000000, perimeter /␣
↪1000))
```

1.7 Coördinaten omzetten van Lambert '72 (EPSG:31370) naar Lambert '08 (EPSG:3812)

De conversie van het Belgische Lambert '72 CRS (EPSG:31370) naar het Belgische Lambert '08 CRS (EPSG:3812) is een specifieke conversie die de Belgische landmeter meer en meer zal moeten toepassen. Verschillende software, zoals CConvert van het NGI of PCTrans van de Nederlandse Marine laten dergelijke transformaties toe. Deze tools maken beide gebruik van transformatiegrids, waarmee de nauwkeurigheid van de transformatie verbeterd zal worden. Deze transformatiegrids worden standaard niet geleverd bij de installatie van `PyProj`, maar zijn we degelijk afzonderlijk toe te voegen volgens deze procedure. We gaan hier niet verder in op deze toevoeging, maar zullen de transformatie tussen beide systemen hier nog wel een keer herhalen. We vertrekken hierbij van een officieel meetpunt van het NGI (14B07T1). Dit en andere punten zijn terug te vinden op de website van het NGI. Het te transformeren punt heeft de volgende coördinaten:

CRS	x (m)	y (m)
LB72	111061.41	210162.32
LB08	611056.63	710157.85

[ ]:
```
from pyproj import Proj, Transformer
transformer = Transformer.from_crs("EPSG:31370", "EPSG:3812", always_xy=True)
xLB72, yLB72 = (111061.41, 210162.32)
xLB08, yLB08 = (611056.63, 710157.85)
xLB08_calc, yLB08_calc = transformer.transform(xLB72, yLB72)
print('%9.3f %11.3f' % (xLB08_calc,yLB08_calc))
print('%9.3f %11.3f' % (xLB08-xLB08_calc,yLB08-yLB08_calc))
```

Opdracht: de coördinaten van het HOGENT P-gebouw op de campus Schoonmeersen kunnen we met de volgende code ophalen van de WFS van het Vlaamse GRB:

```
import requests
url = 'https://geoservices.informatievlaanderen.be/overdrachtdiensten/GRB/wfs'
payload = {'SERVICE':'WFS','REQUEST':'GetFeature','VERSION':'2.0.0',
           'TYPENAMES':'GRB:GBG','featureID':'GBG.1929586',
           'outputFormat':'application/json','srsName':'<CRS_CODE/>'}
r = requests.get(url, params=payload)
r.json()['features'][0]['geometry']['coordinates'][0]
```

Haal de coördinaten van het gebouw eerst op in het Belgische Lambert '72 CRS, en converteer de coördinaten naar het Belgische Lambert '08 CRSmet behulp van PyProj. Haal vervolgens de coöordinaten nog eens op, maar dan in het Belgische Lambert '08 CRS, en vergelijk deze coördinaten met de door PyProj getransformeerde coördinaten. Geef het gemiddelde verschil in 2D weer, alsook de standaardafwijking van deze verschillen.

[ ]:
```
## UW CODE HIER ##
```