

02_Measure_morphometric_characters

March 27, 2020

1 Measure morphometric characters

Computational notebook 02 for *NAME* Dal Cin, Fleischmann.. - ADD Reference.

Contact: martin@martinfleischmann.net

Date: 27/03/2020

This notebook generates additional morphometric elements (morphological tessellation and tessellation-based blocks) and measures primary morphometric characters using *momepy* v0.1.1.

The network data obtained using *01_Retrieve_network_data.ipynb* were manually cleaned in the meantime to represent topologically correct morphological network. Moreover, the layer **name_case** containing a single polygon representing case study area for each case was manually created based on street network (captures blocks with any buildings).

Structure of GeoPackages:

```
./data/
  atlantic.gpkg
    name_blg - Polygon layers
    name_str - LineString layers
    name_case - Polygon layers
    ...
  preat1.gpkg
    name_blg
    name_str
    name_case
    ...
  premed.gpkg
    name_blg
    name_str
    name_case
    ...
  med.gpkg
    name_blg
    name_str
    name_case
    ...
```

CRS of the original data is EPSG:3763.

```
<Projected CRS: EPSG:3763>
Name: ETRS89 / Portugal TM06
Axis Info [cartesian]:
- X[east]: Easting (metre)
- Y[north]: Northing (metre)
Area of Use:
- name: Portugal - mainland - onshore
- bounds: (-9.56, 36.95, -6.19, 42.16)
Coordinate Operation:
- name: Portugual TM06
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

```
[1]: import fiona
import geopandas as gpd
import momepy as mm
import libpysal
import numpy as np
```

```
[2]: fiona.__version__, gpd.__version__, mm.__version__, libpysal.__version__, np.
    ↪ __version__
```

```
[2]: ('1.8.13', '0.7.0', '0.1.1', '4.2.2', '1.18.1')
```

```
[ ]: folder = 'data/'
```

```
[ ]: parts = ['atlantic', 'preatl', 'premed', 'med']

# Iterate through parts and layers
for part in parts:
    path = folder + part + '.gpkg'
    layers = [x for x in fiona.listlayers(path) if 'blg' in x]
    for l in layers:
        print(l)
        buildings = gpd.read_file(path, layer=l)
        buildings = buildings.explode().reset_index(drop=True) # avoid ↵
        ↪ MultiPolygons
        buildings['uID'] = mm.unique_id(buildings)
        try:
            buildings = buildings.drop(columns=['Buildings', 'id'])
        except:
            buildings = buildings[['uID', 'geometry']]

# Generate morphological tessellation
```

```

limit = gpd.read_file(path, layer=1[:-3] + 'case').geometry[0]
tess = mm.Tessellation(buildings, 'uID', limit=limit)
tessellation = tess.tessellation

# Measure individual characters
buildings['sdbAre'] = mm.Area(buildings).series
buildings['sdbPer'] = mm.Perimeter(buildings).series
buildings['ssbCCo'] = mm.CircularCompactness(buildings).series
buildings['ssbCor'] = mm.Corners(buildings).series
buildings['ssbSqu'] = mm.Squareness(buildings).series
buildings['ssbERI'] = mm.EquivalentRectangularIndex(buildings).series
buildings['ssbElo'] = mm.Elongation(buildings).series
buildings['ssbCCD'] = mm.CentroidCorners(buildings).mean
buildings['stbCeA'] = mm.CellAlignment(buildings, tessellation,
                                       mm.Orientation(buildings).series,
                                       mm.Orientation(tessellation).
↳series, 'uID', 'uID').series
    buildings['mtbSWR'] = mm.SharedWallsRatio(buildings, 'uID').series
    blg_sw1 = mm.sw_high(k=1, gdf=tessellation, ids='uID')
    buildings['mtbAli'] = mm.Alignment(buildings, blg_sw1, 'uID', mm.
↳Orientation(buildings).series).series
    buildings['mtbNDi'] = mm.NeighborDistance(buildings, blg_sw1, 'uID').
↳series

    tessellation['sdcLAL'] = mm.LongestAxisLength(tessellation).series
    tessellation['sdcAre'] = mm.Area(tessellation).series
    tessellation['sscERI'] = mm.EquivalentRectangularIndex(tessellation).
↳series
    tessellation['sicCAR'] = mm.AreaRatio(tessellation, buildings,
↳'sdcAre', 'sdbAre', 'uID').series

    buildings['ldbPWL'] = mm.PerimeterWall(buildings).series

    edges = gpd.read_file(path, layer=1[:-3] + 'str')

    edges = edges.loc[~(edges.geom_type != "LineString")].explode().
↳reset_index(drop=True)
    edges = mm.network_false_nodes(edges)
    edges['nID'] = mm.unique_id(edges)

    buildings['nID'] = mm.get_network_id(buildings, edges, 'nID',
↳min_size=100)

# merge and drop unlinked
tessellation = tessellation.drop(columns='nID').merge(buildings[['uID',
↳'nID']], on='uID')

```

```

tessellation = tessellation[~tessellation.isna().any(axis=1)]
buildings = buildings[~buildings.isna().any(axis=1)]

buildings['stbSAI'] = mm.StreetAlignment(buildings, edges, mm.
↳Orientation(buildings).series, network_id='nID').series
tessellation['stcSAI'] = mm.StreetAlignment(tessellation, edges, mm.
↳Orientation(tessellation).series, network_id='nID').series

edges['sdsLen'] = mm.Perimeter(edges).series
edges['sssLin'] = mm.Linearity(edges).series

profile = mm.StreetProfile(edges, buildings, distance=3)
edges['sdsSPW'] = profile.w
edges['stsOpe'] = profile.o
edges['svsSDe'] = profile.wd

edges['sdsAre'] = mm.Reached(edges, tessellation, 'nID', 'nID',
↳mode='sum').series
edges['sdsBAr'] = mm.Reached(edges, buildings, 'nID', 'nID',
↳mode='sum').series

edges['sisBpM'] = mm.Count(edges, buildings, 'nID', 'nID',
↳weighted=True).series

regimes = np.ones(len(buildings))
block_w = libpysal.weights.block_weights(regimes, ids=buildings.uID.
↳values)

buildings['ltcBuA'] = mm.BuildingAdjacency(buildings, block_w, 'uID').
↳series

G = mm.gdf_to_nx(edges)

G = mm.meshedness(G, radius=5, name='meshedness')
mm.mean_nodes(G, 'meshedness')

edges = mm.nx_to_gdf(G, points=False)

if 'bID' in buildings.columns:
    buildings = buildings.drop(columns='bID')

# Generate blocks
gen_blocks = mm.Blocks(tessellation, edges, buildings, 'bID', 'uID')
blocks = gen_blocks.blocks
buildings['bID'] = gen_blocks.buildings_id
tessellation['bID'] = gen_blocks.tessellation_id

```

```

blocks['ldkAre'] = mm.Area(blocks).series
blocks['lskElo'] = mm.Elongation(blocks).series
blocks['likGra'] = mm.Count(blocks, buildings, 'bID', 'bID',
↪weighted=True).series

# Save to file
buildings.to_file(path, layer=1, driver='GPKG')
tessellation.to_file(path, layer=1[:-3] + 'tess', driver='GPKG')
edges.to_file(path, layer=1[:-3] + 'str', driver='GPKG')
blocks.to_file(path, layer=1[:-3] + 'blocks', driver='GPKG')

```