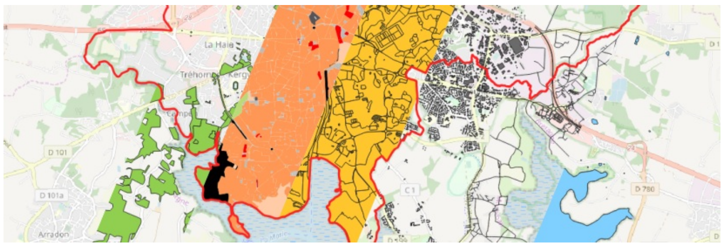


GeoClimate 0.0.1 documentation

Table of contents

Home

Welcome to the GeoClimate wiki documentation!

GeoClimate is an opensource geospatial toolbox to compute a set of climate related parameters describing a territory (morphological indicators such as Sky View Factor, urban classifications such as Local Climate Zones, etc.).

GeoClimate uses vector-based inputs. Specific workflows have been developed to automatically use OpenStreetMap and the French BD Topo 2.2 version databases but the algorithms are data independent, thus allowing the user to connect any vector-based dataset.

GeoClimate is developed in Groovy language.

Open-source philosophy

It can be freely used either for research and education, as well as by experts in a professional use.

GeoClimate is distributed under LGPL 3 license by the DECIDE GIS team of the Lab-STICC (CNRS).

You are welcome:

- to contribute to GeoClimate please visit the code source repository : https://github.com/orbisgis/geoclimate
- to contact members of the team, use the email info@orbisgis.org or let an issue : https://github.com/orbisgis/geoclimate/issues

Fundings

The GeoClimate library has been originally developed within the two following research projects:

- URCLIM (2017 -2021), part of ERA4CS, a project initiated by JPI Climate and co-funded by the European Union under grant agreement No 690462,
- PAENDORA (2017 -2021), funded by ADEME
- CENSE (2017 -2021), funded by ANR
- Chaire GeoTERA (2020), funded by SNCF Réseau on behalf of the foundation Université Bretagne Sud (https://www.univ-ubs.fr)

```
Note

    The official documentation is available in English only.

    If you observe some mistakes or errors, please contact us at info@orbisgis.org or let an issue here.

    You are welcome to contribute, improve the documentation
```

Introduction

Local climate is affected by key factors such as the type of land surface (vegetation, water, building, impervious, etc.) or the size, the shape, the use and the distribution of the buildings. Thus it is necessary to describe accurately the land fabric in order to better understand climate processes.

Geoclimate computes geospatial indicators which can be currently used:

- for modeling purpose: to create the input data needed by parametric urban climate models such as Town Energy Balance (TEB),
- for planning purpose: to qualify urban tissues according to **climate-related classifications** such as the *urban typology* presented in Bocher et al. (2017) or the Local Climate Zones (LCZ).

Geoclimate performs indicator computation at three spatial unit scales, a spatial unit being a POLYGON or MULTIPOLYGON geometry:

1. **building** scale, defined as a collection of features that represent 3D objects with walls and a roof,
2. **block** scale, defined as a set of buildings touching each other (at least one point in common) or as an isolated building,
3. Reference Spatial Unit (**RSU**) scale, being the elementary unit to characterize all the characteristics of a piece of land (not only related to buildings but also to vegetation, water, etc.). It can be defined in different ways:
    - Topographical Unit (TU): it is a continuous and homogeneous way to divide the space using topographic constraints: road and railway center lines, vegetation, impervious and water areas, administrative boundaries.

Overall, more than **100 urban indicators** are currently calculated (e.g. compactness and road distance at building scale, volume at block scale, building fraction, mean sky view factor at RSU scale, …). Note that Geoclimate has first been developed for climate studies but many indicators could also be useful to analyze landscape ecology, land use, habitat conservation planning or any environmental or territory applications.

Processing steps

To compute the whole indicators and classifications, GeoClimate uses the concept of Workflow to chain a set of spatial analysis and statistical processes. By default, the indicators are calculated at the TU scale (cf. subsection Default TU scale calculation) but optionally the user may aggregate indicators spatially at the scale of a grid of rectangular meshes (cf. subsection Spatial aggregation using a regular grid).

Default TU scale calculation

The Workflow is organized in 3 steps (Figure 1). Note that each step or each process within a step can be run individually (cf. section Coding implementation). Note also that you can use the workflow even though you do not provide all input data. Partitioning and indicators calculations will be limited to the supplied data.

1. Building new spatial units

The first step of the GeoClimate chain concerns the construction of two new spatial units (block and RSU).

- a block is defined as an aggregation of buildings that are in contact,
- a RSU (Reference Spatial Unit), being the elementary unit to characterize all the characteristics of a piece of land (not only related to buildings but also to vegetation, water, etc.). In the default case described here, Topographical Spatial Units (TSU) are used and they are defined as a continuous and homogeneous way to divide the space using topographic constraints based on road and railway center lines, vegetation and water surface boundaries, administrative boundaries. The construction of the TSU is a key process in GeoClimate. First, a planar graph is built using all input geometries. The planar graph is then traversed to generate new polygons. Only 2D is considered for partitioning, therefore underground elements (such as tunnels), or overground (such as bridges) are excluded from the input. Water and vegetation surfaces are also excluded from the input data when they are smaller than a certain threshold, set by default to 2'500 m² for water and 10'000 m² for vegetation.

2. Compute spatial indicators

The spatial indicators are computed at three scales : building, block and RSU. Buildings are characterized by their location in a geographical space (e.g distance to the nearest road, average distance to other buildings, number of building neighbor...), building and blocks are characterized by morphological indicators (e.g. a form factor), RSU are characterized by fraction of land type (e.g. vegetation, water, impervious fractions...) and specific climate-oriented indicators (e.g. aspect ratio, mean sky view factor...). Some of the building indicators are also aggregated at block scale (e.g. mean block height) and some of the building and block indicators are aggregated at RSU scale (e.g. mean number of neighbors per building, mean building height...). At the end, more than 100 indicators are calculated.

3. Apply classifications

Classifications use the spatial indicators at the three scales and specific statistical model / algorithms to set:

- the typology of a building based on an architectural approach (Urban Typology by Random Forest - UTRF) such as defined in Bocher et al. (2017) which is then aggregated at RSU scale
- the typology of a RSU based on the Local Climate Zones definition (Stewart et Oke, 2012)

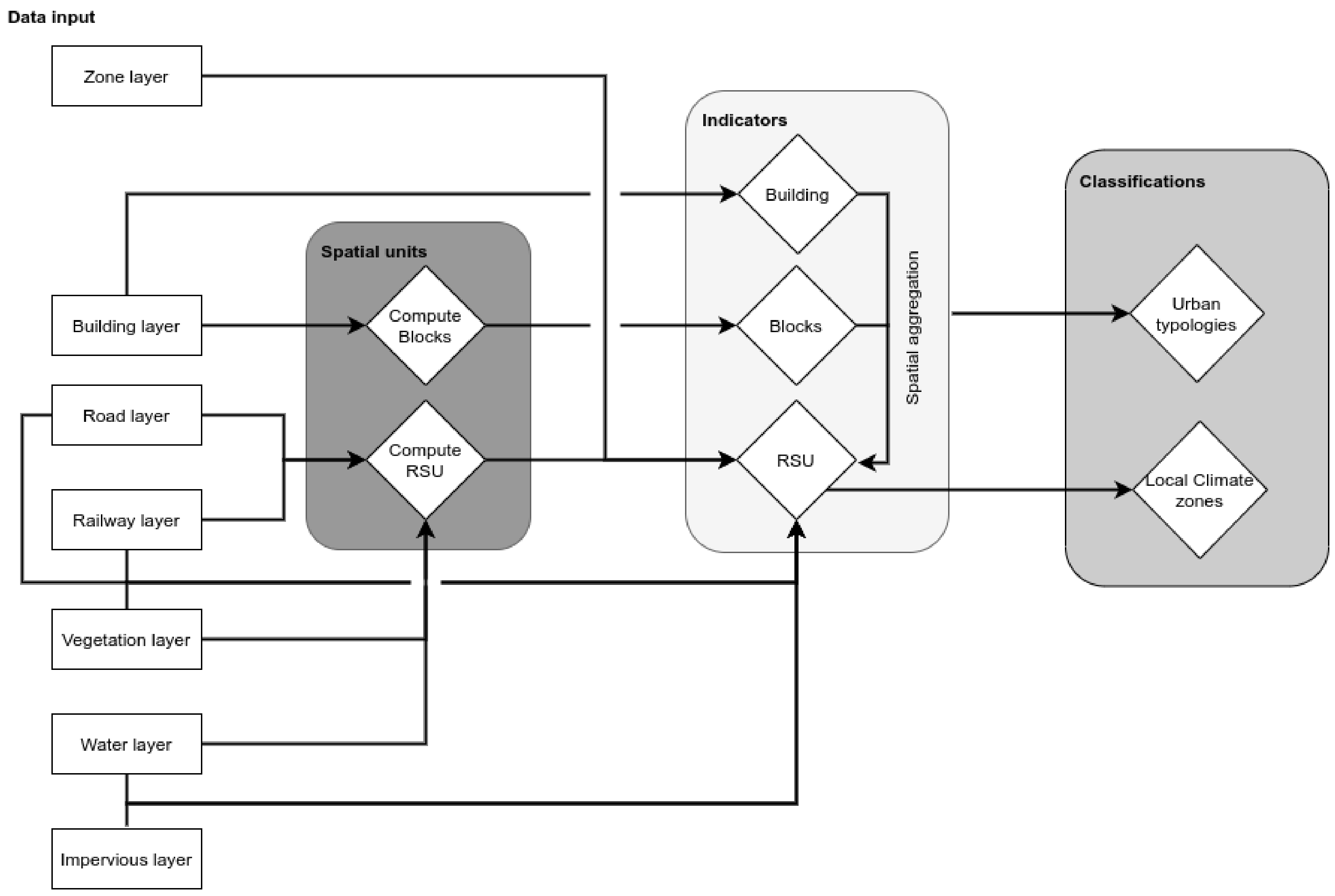


Figure 1. Main GeoClimate processing steps

Spatial aggregation using a regular grid

GeoClimate integrates a "rasterization" of the indicators and of the classifications which can be done directly from the input data sources (if the LCZ classification is not needed) or after applications of the chain defined above (if the LCZ classification is needed). The rasterization process creates a grid based on the bounding box of the zone layer. The grid is intersected with a set of input data given to a spatial aggregating function that computes the square cell fraction of each input data and the building height which is a footprint area weighted mean height (Figure 2).

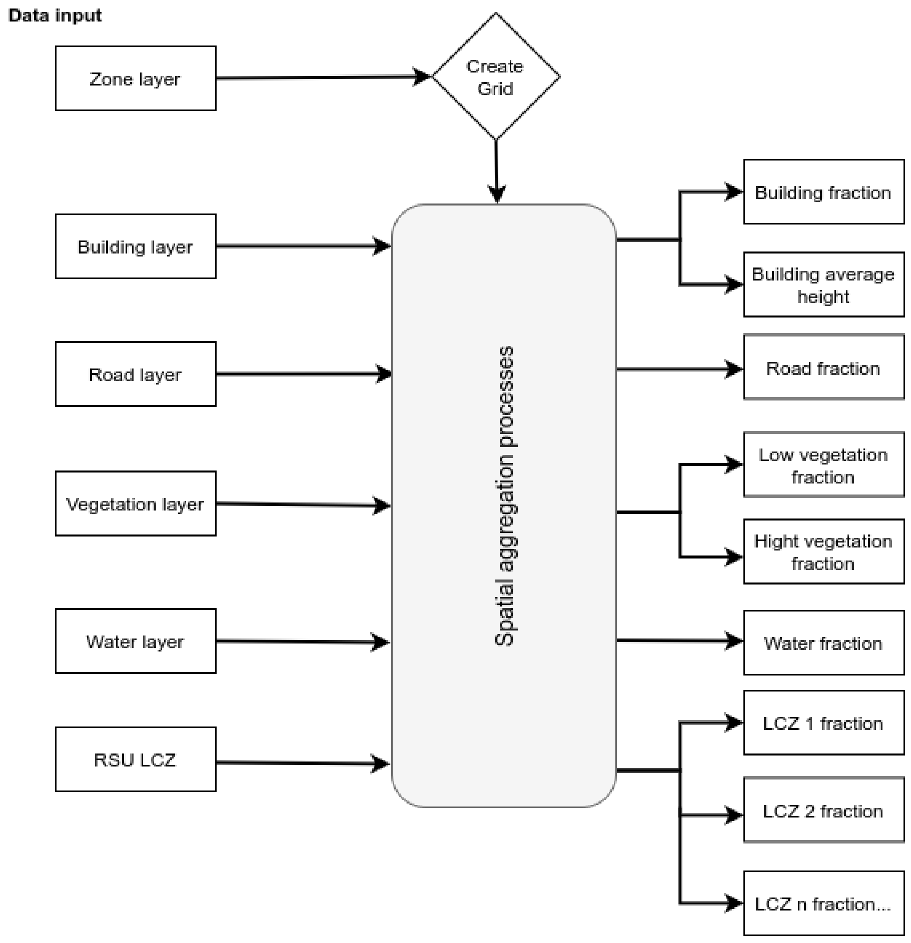


Figure 2. Processing steps to aggregate indicators at grid scale

Input data

The indicators in GeoClimate are calculated from vector GIS layers that represent the main topographic features. To guarantee the use of the algorithms and their outputs, the GIS layers must follow a set of specifications. These specifications are defined for each layer. They include the name of the columns, the values used by the attributes, the dimension of the geometry...

Depending on the use of GeoClimate, the number of input GIS layers differs.

Note that a GIS layer is an abstraction of reality specified by a geographic data model (geometry + attributes). It represents a single geographic subject. It consists in a set of data staged in a tabular way (rows, columns).

The GIS layer have to use a metric reference spatial system. Lat/Long coordinates are not supported by the algorithms. Thus if you have Lat/Long coordinates data, you first need to reproject in a local metric system.

Zone layer

The zone layer represents the studied area. Only one geometry is expected.

Name	Type	Constraints	Definition
the_geom	POLYGON	X Y dimension	Geometry
id_zone	VARCHAR	*not null*	Identifier of the zone area

Building layer

The building layer represents the footprint of the building as a set of Polygons or MultiPolygons in 2D coordinates.

Name	Type	Constaints	Definition
the_geom	POLYGON	X Y dimensions	Geometry
id_build	INTEGER	Primary Key	Unique Identifier
id_source	VARCHAR	*not null*	Identifier of the feature from the input data source
id_zone	VARCHAR	*not null*	Studied zone identifier
height_wall	FLOAT	*not null; > 0*	The (corrected) height of the building in meters. Height of the building measured between the ground and the gutter (maximum altitude of the polyline describing the building). *(expressed in meters)*
height_roof	FLOAT	*not null ; > 0 ; >= height_wall*	The maximum height of a building is the distance between the top edge of the building (including the roof, but excluding antennas, spires and other equipment mounted on the roof) and the lowest point at the bottom where the building meets the ground. *(expressed in meters)*
nb_lev	INTEGER	*not null; > 0*	Number of levels (have to be greater than 0)
type	VARCHAR	*not null ; in type list*	Value allowing to distinguish the type of building according to its architecture. These values are listed in the BUILDING_use_and_type section.
main_use	VARCHAR	*in type list*	Main use of the building. The use of a building corresponds to a de facto element, relating to what it is used for. These values are listed in the BUILDING_use_and_type section.
zindex	INTEGER	*not null ; >- 4 ; 4<*	Defines the position with respect to the ground. 0 indicates that the object is on the ground. 1 to 4 indicates that the objects above the ground surface. -4 to -1 value indicates that the object is underground.

type and main_use column values

List of all possible values for the `type` and the `main_use` attributes, in the `BUILDING` layer. We consider that a same value can be used to qualify a `type` or a `main_use`.

Term	Definition	Source
building	Used to qualify any kind of feature that is a building	1
house	A single dwelling unit usually inhabited by one family	2
detached	A free-standing residential building usually housing a single-family.	3
residential	A building used primarily for residential purposes	4
apartments	A building arranged into individual dwellings, often on separate floors. May also have retail outlets on the ground floor.	5
bungalow	A small, single-storey detached house in the form of a bungalow	6
historic	Any buildings of historical interest	7
monument	A memorial object, which is especially large, built to remember, show respect to a person or group of people or to commemorate an event.	8
ruins	House that is an abandoned (but still a building)	9
castle	Various kinds of structures, most of which were originally built as fortified residences of a lord or noble	10

Term	Definition	Source
agricultural	A building, machinery, facilities, related to agricultural production.	11
farm	A farmhouse is the main building of a farm	12
farm_auxiliary	A building on a farm that is not a dwelling	13
barn	An agricultural building used for storage and as a covered workplace	14
greenhouse	A greenhouse (also called a glasshouse) is a building in which plants are grown. It typically has a roof and walls made of clear glass or plastic to allow sunlight to enter.	15
silo	A storage container for bulk material, often grains such as corn or wheat	16
commercial	A building where non-specific commercial activities take place	17
industrial	A building where some industrial process takes place	18
sport	Buildings, constructions, installations, organized areas and equipment for indoor and outdoor sport activities.	19
sports_centre	Building that is designed for sports, e.g. for school sports, university or club sports	20
grandstand	Building for the main stand, usually roofed, commanding the best view for spectators at racecourses or sports grounds	21
transportation	Buildings, constructions, installations, organized areas and equipment for transportation	22
train_station	A train station building	23
toll_booth	Toll roads charge money for some or all traffic	24
terminal	An airport passenger building	25
healthcare	All places that provide healthcare	26
education	All places that provide education	27
entertainment_arts_culture	All places that provide entertainment, arts and culture	28
sustenance	Buildings, constructions, installations, organized areas and equipment of any food commodity or related food products. ex : bar, pub...	29 , 30
military	Buildings, constructions, installations necessary to the performance of military activities, either combat or noncombat.	31 , 32 , 33
religious	Unspecific religious building	34
chapel	Religious building, often pretty small. One can enter in it to pray or meditate	35
church	A building that was built as a church	36
government	Building built to house government offices	37
townhall	Building that may serve as an administrative center, or may be merely a community meeting place	38
office	Office block typically houses companies, but offices may be also rented by any other kind of organization like charities, government, any NGO etc.	39

Road layer

The road layer represents any kind of roadways.

Name	Type	Constraints	Definition
the_geom	LINESTRING or MULTILINESTRING	X Y dimensions	Geometry
id_road	INTEGER	Primary Key	Unique Identifier
id_source	VARCHAR	*not null*	Identifier of the feature from the input datasource
width	FLOAT	*not null*	Width of the road *(expressed in meters)*
type	VARCHAR	*not null*; in type list	Type of road
surface	VARCHAR	in surface list	The surface value is used to provide additional information about the physical surface of roads/footpaths and some other features, particularly regarding material composition and/or structure.
sidewalk	VARCHAR	*not null*; in (`one`, `two`, `no`)	Specify if the road has one, two or no sidewalk(s). Default value should be `no`.
zindex	VARCHAR	*not null* ; ≥ -4 ; ≤ 4	Defines the position with respect to the ground. 0 indicates that the object is on the ground. 1 to 4 indicates that the object is above the ground surface. -4 to -1 value indicates that the object is underground.
crossing	VARCHAR	`bridge` or `null`	Indicates whether the road is located on a `bridge` or not (`null`). A bridge is defined as an artificial construction that spans features such as roads, railways, waterways or valleys and carries a road, railway or other feature
maxspeed	INTEGER	-1 if unknown	Indicates the maximum legal speed limit for general traffic
direction	INTEGER	-1 if unknown	Indicates the direction of the road. 1 = one way road section and the traffic goes in the same way that the slope definition you have used, 2 = one way road section and the traffic goes in the inverse way that the slope definition you have used, 3 = bi-directional traffic flow, the flow is split into two components and correct half for uphill and half for downhill

List of type column values

The possible values for the `type` column are the following:

Term	Definition	Source
cycleway	Separated way for the use of cyclists.	1
ferry	A ferry route used to transport things or people from one bank of a watercourse or inlet to the other, or as a permanent or seasonal local maritime link, and a link to a foreign country.	2
footway	For designated footpaths, i.e. mainly/exclusively for pedestrians.	3
highway	Any kind of street or way.	4
highway_link	Connecting ramp to/from a highway.	5
motorway	Highest-performance highway within a territory that deserve main towns. Usually have a reglemented access.	6
path	A generic multi-use path open to non-motorized vehicles.	7
primary	Important highway linking large towns. Usually have two lanes but not separated by a central barrier.	8

Term	Definition	Source
residential	Highway generally used for local traffic within settlement. Usually highway accessing or around residential areas.	9
roundabout	Generally a circular (self-intersecting) highway junction where the traffic on the roundabout has right of way.	10
secondary	Highway linking large towns. Usually have two lanes but not separated by a central barrier.	11
steps	For flights of steps on footways and paths.	12
tertiary	Highway linking small settlements, or the local centers of a large town or city.	13
track	Highway for mostly agricultural use, forest tracks etc.; usually unpaved (unsealed) but may apply to paved tracks as well, that are suitable for two-track vehicles, such as tractors or jeeps.	14
trunk	Important high-performance highway that are not motorways. Deserving main towns.	15
unclassified	Minor public highway typically at the lowest level of the interconnecting grid network. Have lower importance in the highway network than tertiary and are not residential streets or agricultural tracks.	16

List of surface column values

Possible values for the `surface` column are the following:

Term	Definition	Source
asphalt	Any asphalt surface.	1
cobblestone	Any cobbled surface.	2
compacted	A mixture of larger (e.g., gravel) and smaller (e.g., sand) parts, compacted.	3
concrete	Cement based concrete surface.	4
grass	Grass covered ground.	5
gravel	Surface composed of broken/crushed rock larger than sand grains and thinner than pebblestone.	6
ground	Surface of the ground itself with no specific fraction of rock.	7
metal	Metallic surface.	8
mud	Wet unpaved surface.	9
paved	Surface with coating. Generic term for a highway with a stabilized and hard surface.	10
pebblestone	Surface made of rounded rock as pebblestone findable alongside body of water.	11
sand	Small to very small fractions of rock as findable alongside body of water.	12
unpaved	Generic term to qualify the surface of a highway that is predominantly unsealed along its length; i.e., it has a loose covering ranging from compacted stone chippings to ground.	13
water	Used to qualify the surface of ferry route that uses water (waterbodies, watercourses, seas,...) as a traffic surface.	14
wood	Highway made of wooden surface.	15

Railway layer

The Railway layer represents any kind of runways for wheeled equipment. Geometries must be LINESTRING or MULTILINESTRING.

Name	Type	Constraints	Definition
the_geom	LINESTRING or MULTILINESTRING	X Y dimensions	Geometry
id_railway	INTEGER	Primary Key	Unique Identifier
id_source	VARCHAR	*not null*	Identifier of the feature from the input datasource
type	VARCHAR	*not null*; *in type list*	Type of rail
zindex	INTEGER	*not null*; *>= -4*; *<= 4*	Defines the position with respect to the ground. 0 indicates that the object is on the ground. 1 to 4 indicates that the object is above the ground surface. -4 to -1 value indicates that the object is underground.
crossing	VARCHAR	`bridge` or `null`	Indicates whether the rail is located on a `bridge` or not (`null`). A bridge is defined as an artificial construction that spans features such as roads, railways, waterways or valleys and carries a road, railway or other feature

type column values

List of all possible values for the `type` values.

Term	Definition	Source
highspeed	Railway track for highspeed rail.	1
rail	Railway track for full sized passenger or freight trains in the standard gauge for the country or state.	2
service track	Railway track mainly used for sorting or temporary parking of freight trains.	3
disused	A section of railway which is no longer used but where the track and infrastructure remain in place.	4
funicular	Cable railway in which a cable attached to a pair of tram-like vehicles on rails moves them up and down a steep slope, the ascending and descending vehicles counterbalancing each other.	5
subway	Rails used for city public transport that are always completely separated from other traffic, often underground	6
tram	Railway track which is mainly or exclusively used for trams, or where tram tracks are laid within a normal road open to all traffic, often called street running.	7

Vegetation layer

The vegetation layer represents any kind of land areas that qualify a natural feature.

Name	Type	Constraints	Definition
the_geom	POLYGON	X Y dimensions	Geometry
id_vegetation	INTEGER	Primary Key	Unique Identifier
id_source	VARCHAR	*not null*	Identifier of the feature from the input datasource
type	VARCHAR	*not null*	Type of vegetation.
height_class	VARCHAR	*not null*	Height class (`low` or `high`)

type column values

List of all possible values for `type` column.

Term	Definition	Source
tree	A single tree	1
wood	Tree-covered area (a 'forest' or 'wood') not managed for economic purposes	2
forest	Managed woodland or woodland plantation. Wooded area maintained by human to obtain forest products	3
scrub	Uncultivated land covered with bushes or stunted trees	4
grassland	Natural areas where the vegetation is dominated by grasses (Poaceae) and other herbaceous (non-woody) plants	5
heath	A dwarf-shrub habitat, characterized by open, low growing woody vegetation, often dominated by plants of the Ericaceae	6
tree_row	A line of trees	7
hedge	A line of closely spaced shrubs and tree species, which form a barrier or mark the boundary of an area	8
mangrove	It is formed by forests of salt tolerant mangrove trees in the tidal zone of tropical coasts with water temperatures above 20° C	9
orchard	Intentional planting of trees or shrubs maintained for food production	10
vineyard	A piece of land where grapes are grown	11
banana_plants	A banana plantation	12
sugar_cane	A piece of land where sugar cane are grown	13

Water layer

The water layer represents any kind of surface (river, sea, lake...)

Name	Type	Constraints	Definition
the_geom	POLYGON	X Y dimensions	Geometry
id_water	INTEGER	Primary Key	Unique Identifier
id_source	VARCHAR	*not null*	Identifier of the feature from the input datasource

Impervious layer

The impervious layer means any kind of artificial surfaces which obstructs the percolation of water.

Name	Type	Constraints	Definition
the_geom	POLYGON	X Y dimensions	Geometry
id_impervious	INTEGER	Primary Key	Unique Identifier
id_source	VARCHAR	*not null*	Identifier of the feature from the input datasource

Output data

GeoClimate output data consists both in a set of indicators and classifications.

Indicators

GeoClimate indicators are used to:

- measure morphological properties (e.g the form factor),
- describe spatial organizations (e.g. distance measurements, patch metrics, shape index, spatial density, etc).

They quantify the shape and pattern of urban and landscape structures.

Geoclimate indicators are calculated at 3 different scales:

1. building scale
2. block of buildings (aggregation of buildings that are in contact)
3. chosen Reference Spatial Unit (RSU)

Note that scale 2 (resp. 3) indicators need scale 1 (resp. 2) indicators to be computed.

Each indicator will be shortly described and its computation method roughly/simply defined.

Classifications

Two classifications are available:

1. Local Climate Zones (LCZ) classification, at the RSU scale
2. Urban Typology by Random Forest (UTRF) classification, computed at the building scale and aggregated at the RSU scale

Building indicators

The table `building_indicators` contains the initial informations being in the input table 'building' (id_build, id_source, height_wall, height_roof, etc. - cf. building input table), the identifier of the block and of the RSU it belongs to (id_rsu, id_block) and a certain number of indicators described below.

`AREA`

Description: Building's area.

Method: `Area of the building footprint`

`AREA_CONCAVITY`

Description: Calculates a degree of convexity of a building (according to the building surface).

Method: `Area / Convex Hull area`

Range of values: [`0`, `1`] - the closer the result from 1, the more convex the building.

`COMMON_WALL_FRACTION`

Description: Fraction of linear of facade (also called “party walls”) shared with other buildings.

Method: `Shared facade length / total facade length`

`CONTIGUITY`

Description: Fraction of wall shared with other buildings

Method: `Shared wall area / total wall area`

`FLOOR_AREA`

Description: Building's floor area.

Method: `Area * Number of level`

`FORM_FACTOR`

Description: Ratio between the building’s area and the square of the external building’s perimeter

Method: `Area / (perimeter)^2`

`LIKELIHOOD_LARGE_BUILDING`

Description: Building closeness to a 50 m wide isolated building (where `NUMBER_BUILDING_NEIGHBOR` = 0).

Method: The step 9 of the decision tree used for the MaPUCE project manual building typology classification consists of checking whether a building has a horizontal extent larger than 50 m. We therefore introduce an indicator which measures the horizontal extent of buildings. This indicator is based on the largest side of the building minimum rectangle. We then use a logistic function to avoid threshold effects (e.g. totally different result for building sizes of 49 m and 51 m). The gamma and *x0* parameters in the logistic function are specified after analysis of the training data to identify the real size of the buildings classified as larger than 50 m in the subjective training process.

`MINIMUM_BUILDING_SPACING`

Description: Building closest distance *(expressed in meter)* to an other building.

Method: `Min(distance(building, other buildings within bufferDist))` , where the buffer size of search is defined in the `bufferDist` parameter *(default value = 100 m)*

Warning:

- If the building touches an other building, the result is 0.
- If there is no building in a 100m circle around the building, the result is set to 100m *(this value may be different if the `bufferDist` default value is modified)*.

`NUMBER_BUILDING_NEIGHBOR`

Description: Number of neighboring buildings, in contact with the building.

Method: Count the number of buildings touching (at least one point) the building of interest.

`PERIMETER`

Description: Building's perimeter (external perimeter, do not consider courtyard).

Method: `External building perimeter`

`PERIMETER_CONVEXITY`

Description: Calculates a degree of convexity of a building (according to the building perimeter).

Method: `Convex Hull perimeter / Perimeter`

Range of values: [`0`, `1`] - the closer the result from 1, the more convex the building.

`RAW_COMPACTNESS`

Description: Ratio between building external surfaces (walls and roof) and the building volume at the power 2/3.

Method: `(External walls area + courtyard walls area + roof area) / (volume^(2/3))`

Warning: For the calculation, the roof is supposed to have a gable and the roof surface is calculated considering that the building is square (otherwise, the choice related to the gable direction - which is not known - would affect the result).

`ROAD_DISTANCE`

Description: Building closest distance *(expressed in meter)* to a road,

Method: The search is made within a buffer area around the building whose size is defined in the `bufferDist` parameter *(default value = 100 m)*.

→ `Min(distance(building, roads within bufferDist))`

Warning:

- If the building touches a road, the result is 0.
- If the roads are further than 100m from the building, the result is set to 100m *(this value may be different if the* `bufferDist` *default value is modified)*.

`TOTAL_FACADE_LENGTH`

Description: Total length of external facade.

Method: `Building perimeter + Courtyard perimeter`

`VOLUME`

Description: Building's volume.

Method: The building volume is calculated considering that all buildings have either horizontal or gable roofs. In this case, the building volume can be calculated using the following equation:

→ `Area * ((Wall height + Roof height)/2)`

Block indicators

The table `block_indicators` contains the block identifier (id_block), the identifier of the RSU it belongs to (id_rsu) and a certain number of indicators described below.

`AREA`

Description: Area of the block footprint.

Method: `Sum(building area)`

`AVG_HEIGHT_ROOF_AREA_WEIGHTED`

Description: Mean building's roof height within a block (the building height being weighted by the building areas).

Method: `SUM(Bu_Wall_Height * Bu_Area) / SUM(Bu_Area)`

`BUILDING_DIRECTION_EQUALITY`

Description: Indicates how equal is the block building direction distribution (having `nb_direction` directions of analysis).

Method: From the building direction distribution created in the `MAIN_BUILDING_DIRECTION` indicator calculation, an indicator of equality of the distribution is calculated:

→ `Sum(Min(1/nb_direction, length_dir_i/length_all_dir))`

Range of values: [`nb_direction`, `1`] - the higher the value the most equal is the distribution

`BUILDING_DIRECTION_UNIQUENESS`

Description: Indicates how unique is the RSU main building direction.

Method: From the building direction distribution created in the `MAIN_BUILDING_DIRECTION` indicator calculation, an indicator of uniqueness of the main direction is calculated:

→ `| Length_First_Dir - Length_Second_Dir | / (Length_Second_Dir + Length_First_Dir)`

Range of values: [`0`, `1`] - the closer the value from 1, the more unique is the main building direction

`CLOSINGNESS`

Description: This calculation indicates if a block has a large closed courtyard. This information could be useful for the urban fabric classification proposed in Thornay et al. (2017) and also described in Bocher et al. (2018). It answers to the Step 11 of the manual decision tree which consists in checking whether the block is closed (continuous buildings the aligned along road).

Method: In order to identify the RSU with closed blocks, the difference between the `st_holes(bloc scale)` and `SUM(st_holes(building scale))` indicators is calculated.

Warning: this method will not be able to identify blocks that are nearly closed (e.g. 99 % of the RSU perimeter) while it would be interesting to know how much the block is closed (for ventilation purpose for example).

References:

- Bocher, E., Petit, G., Bernard, J., & Palominos, S. (2018). A geoprocessing framework to compute urban indicators: The MApUCE tools chain. Urban climate, 24, 153-174.
- Tornay, Nathalie, Robert Schoetter, Marion Bonhomme, Serge Faraut, and Valéry Masson. "GENIUS: A methodology to define a detailed description of buildings for urban climate and building energy consumption simulations." Urban Climate 20 (2017): 75-93.

`FLOOR_AREA`

Description: Total floor area within the block.

Method: `Sum(building floor area)`

`HOLE_AREA_DENSITY`

Description: Density of hole within a block.

Method: `Block courtyard area / block area`

`MAIN_BUILDING_DIRECTION`

Description: Main orientation of the buildings within the blocks (from the North, clockwise).

Method: The smallest rectangle enclosing each building is calculated. Then each side of the rectangles are attributed to a direction range (by default every 15° within the [0, 180°[interval - from North, clockwise). For each direction range, the length of the sides of all rectangles contained in a block are summed. Finally, the mode of this distribution is set as the main building direction within the block.

`NET_COMPACTNESS`

Description: Net block's compactness, defined as the ratio between the area of its free external facade and its building volume.

Method:

→ `SUM((Bu_Contiguity * Bu_Perimeter + Bu_Hole_Perimeter) * Bu_Wall_Height)/Sum(Bu_Volume)`

`STD_HEIGHT_ROOF_AREA_WEIGHTED`

Description: Variability of the building's roof height within a block (the building heights being weighted by the building areas).

Method: By default, the indicator of variability is the Standard Deviation (STD) defined as :

→ `SUM(Bu_Area*(Bu_Wall_Height - AVG_HEIGHT_ROOF_AREA_WEIGHTED)^2)) / SUM (Bu_Area)`

`VOLUME`

Description: Volume of buildings composing a block.

Method: `Sum(building volume)`

RSU indicators

The table `rsu_indicators` contains the RSU identifier (id_rsu) and a certain number of indicators described below. Note that some indicators are generic and thus are described only once with x, y, w, z replacing a generic information (for example `x_FRACTION` is the fraction of input layer x within a RSU). Thus if you look for WATER_FRACTION, look for x_FRACTION, etc.

`AREA`

Description: RSU's area.

Method: `Area of the RSU footprint`

`AREA_FRACTION_x`

Description: Footprint fraction within the RSU of type x building.

Method: `SUM(Bu_AREA of type X) / RSU_Area`

`ASPECT_RATIO`

Description: aspect ratio such as defined by Stewart et Oke (2012): mean height-to-width ratio of street canyons (LCZs 1-7), building spacing (LCZs 8-10), and tree spacing (LCZs A - G).

Method: A simple approach based on the street canyons assumption is used for the calculation. The sum of facade area within a given RSU area is divided by the area of free surfaces of the given RSU (not covered by buildings).

→ `RSU_free_external_facade_density / (1 - RSU_building_density)`

`AVG_HEIGHT_ROOF_AREA_WEIGHTED`

Description: Mean building’s roof height within the RSU (the building heights being weighted by the building areas)

Method: `SUM(Bu_Wall_Height * Bu_Area) / SUM(Bu_Area)`

`AVG_MINIMUM_BUILDING_SPACING`

Description: RSU average minimum distance between buildings.

Method: `SUM(Minimum_Building_Spacing) / Nb_Building`

`AVG_NB_LEV_AREA_WEIGHTED`

Description: RSU average number of levels per building (the building levels being weighted by the building areas).

Method: `SUM(Number_Building_Level * Bu_Area) / SUM(Bu_Area)`

`AVG_NUMBER_BUILDING_NEIGHBOR`

Description: RSU average number of neighbors per building.

Method: `SUM(Number_Building_Neighbors) / Nb_Building`

`AVG_VOLUME`

Description: RSU average building volume.

Method: `SUM(Bu_Volume) / Nb_Building`

`BUILDING_DIRECTION_EQUALITY`

Description: Indicates how equal is the RSU building direction distribution (having `nb_direction` directions of analysis).

Method: From the building direction distribution created in the `MAIN_BUILDING_DIRECTION` indicator calculation, an indicator of equality of the distribution is calculated:

→ `Sum(Min(1/nb_direction, length_dir_i/length_all_dir))`

Range of values: [`nb_direction`, `1`] - the higher the value the most equal is the distribution

`BUILDING_DIRECTION_UNIQUENESS`

Description: Indicates how unique is the RSU main building direction.

Range of values: [0, 1] - the higher the value, the more unique is the main building direction

Method: `| Length_First_Dir - Length_Second_Dir | / (Length_Second_Dir + Length_First_Dir)`

`BUILDING_FLOOR_AREA_DENSITY`

Description: Density of building floor areas within the RSU.

Method: `SUM(Bu_FLOOR_AREA) / RSU_Area`

`BUILDING_FRACTION_LCZ`

Description: Building fraction used for the LCZ classification (by default, total building fraction).

Method: `SUM(Bu_Area without superimposition + Bu_Area superimposed by high_vegetation) / RSU_Area`

back to top

`BUILDING_NUMBER_DENSITY`

Description: RSU number of building density.

Method: `Nb_Building / Rsu_Area`

`BUILDING_TOTAL_FRACTION`

Description: Total fraction of building within the RSU (covered and not covered by high vegetation).

Method: `SUM(Bu_Area without superimposition + Bu_Area superimposed by high_vegetation) / RSU_Area`

`BUILDING_VOLUME_DENSITY`

Description: Density of building volumes within the RSU.

Method: `SUM(Bu_VOLUME) / NB_Building`

`EFFECTIVE_TERRAIN_ROUGHNESS_CLASS`

Description: Effective terrain class from the effective terrain roughness length (z0). The classes are defined according to the Davenport lookup Table (cf Table 5 in Stewart and Oke, 2012)

Method: The Davenport definition defines a class for a unique z0 value (instead of a range). Then there is no definition of the z0 range corresponding to a certain class. We have arbitrarily defined the boundary between two classes as the arithmetic average between the z0 values of each class.

Warning: The choice for the interval boundaries has been made arbitrarily. A definition of the interval based on a log profile of class = f(z0) could lead to different results (especially for classes 3, 4 and 5).

References:

- Stewart, Ian D., and Tim R. Oke. "Local climate zones for urban temperature studies." Bulletin of the American Meteorological Society 93, no. 12 (2012): 1879-1900.

`EFFECTIVE_TERRAIN_ROUGHNESS_LENGTH`

Description: Effective terrain roughness length (z0).

Method: The method for z0 calculation is based on the Hanna and Britter (2010) procedure (see equation (17) and examples of calculation p. 156 in the corresponding reference). The `rsu_projected_facade_area_distribution_Hx_y_Dw_z` is used to calculate the mean projected facade density (considering all directions) and `z0` is then obtained multiplying the resulting value by the `rsu_geometric_mean_height`.

Warning: the calculation of z0 is only performed for angles included in the range [0, 180[°. To simplify the calculation, z0 is considered as equal for a given orientation independently of the direction. This assumption is right when the RSU do not split buildings but could slightly overestimate the results otherwise (z0 is actually overestimated in one direction but OK in the opposite direction).

References:

- Stewart, Ian D., and Tim R. Oke. "Local climate zones for urban temperature studies." Bulletin of the American Meteorological Society 93, no. 12 (2012): 1879-1900.
- Hanna, Steven R., and Rex E. Britter. Wind flow and vapor cloud dispersion at industrial and urban sites. Vol. 7. John Wiley & Sons, 2010.

`FLOOR_AREA_FRACTION_X`

Description: Floor area fraction within the RSU of type X building.

Method: `SUM(Bu_FLOOR_AREA of type X) / RSU_Area`

`FREE_EXTERNAL_FACADE_DENSITY`

Description: Sum of all building free facades (roofs are excluded) included in a RSU, divided by the RSU area.

Method: `SUM((1 - Bu_Contiguity) * Bu_TotalFacadeLength * HEIGHT_WALL) / RSU_Area`

`GEOM_AVG_HEIGHT_ROOF`

Description: RSU geometric mean of the building roof heights.

Method: `EXP(SUM(LOG(Bu_ROOF_HEIGHT)) / NB_Building)`

back to top

`GROUND_LINEAR_ROAD_DENSITY`

Description: Road linear density, having a ZINDEX = 0, within the RSU.

Method: Linear of road at zindex = 0 within a RSU divided by the RSU area

`GROUND_SKY_VIEW_FACTOR`

Description: RSU ground Sky View Factor such as defined by Stewart et Oke (2012): ratio of the amount of sky hemisphere visible from ground level to that of an unobstructed hemisphere. In our case, only buildings are considered as obstructing the atmosphere.

Method: The calculation is based on the ST_SVF function of H2GIS using only buildings as obstacles and with the following parameters: ray length = 100, number of directions = 60. Using a uniform grid mesh of 10 m resolution, the SVF obtained has a standard deviation of the estimate of 0.03 when compared with the most accurate method (according to Bernard et al. (2018)).

Using a grid of regular points, the density of points used for the calculation actually depends on building density (higher the building density, lowei the density of points). To avoid this phenomenon and have the same density of points per free ground surface, we use an H2GIS function to distribute randomly points within free surfaces (ST_GeneratePoints). This density of points is set by default to 0.008, based on the median of Bernard et al. (2018) dataset.

References:

- Stewart, Ian D., and Tim R. Oke. "Local climate zones for urban temperature studies." Bulletin of the American Meteorological Society 93, no. 12 (2012): 1879-1900.
- Jérémy Bernard, Erwan Bocher, Gwendall Petit, Sylvain Palominos. Sky View Factor Calculation in Urban Context: Computational Performance and Accuracy Analysis of Two Open and Free GIS Tools. Climate, MDPI, 2018, Urban Overheating - Progress on Mitigation Science and Engineering Applications, 6 (3), pp.60.

`HIGH_VEGETATION_FRACTION_LCZ`

Description: High vegetation fraction used for the LCZ classification (by default, total high_vegetation fraction).

Method: `SUM(High_veg_Area without superimposition + High_veg_Area superimposing all other layers) / RSU_Area`

`HIGH_VEGETATION_IMPERVIOUS_FRACTION_URB`

Description: Fraction of high vegetation covering impervious layer such as defined for the UTRF classification.

Method: `SUM(Imperv_Area with and without superimposition + Road_Area with and without superimposition) / RSU_Area`

`HIGH_VEGETATION_PERVIOUS_FRACTION_URB`

Description: Fraction of high vegetation covering pervious layer such as defined for the UTRF classification.

Method: `SUM(Perv_Area with and without superimposition + Road_Area with and without superimposition) / RSU_Area`

back to top

`IMPERVIOUS_FRACTION_LCZ`

Description: Impervious fraction used for the LCZ classification (by default, total impervious fraction).

Method: `SUM(Imperv_Area with and without superimposition + Road_Area with and without superimposition) / RSU_Area`

`IMPERVIOUS_FRACTION_URB`

Description: Impervious fraction used for the UTRF classification.

Method: `SUM(Imperv_Area with and without superimposition + Road_Area with and without superimposition) / RSU_Area`

`LOW_VEGETATION_FRACTION_LCZ`

Description: Low vegetation fraction used for the LCZ classification.

Method: `SUM(Low_veg_Area without superimposition) / RSU_Area`

`LOW_VEGETATION_FRACTION_URB`

Description: Low vegetation fraction used for the UTRF classification.

Method: `SUM(Low_veg_Area without superimposition) / RSU_Area`

`MAIN_BUILDING_DIRECTION`

Description: Main direction of the buildings contained in a RSU.

Method: The building direction distribution is calculated according to the length of the building SMBR sides (width and length). The [0, 180]° angle range is splitted into `nb_directions` angle ranges . Then the length of each SMBR building side is attributed to one of these ranges according to the side direction. Within each angle range, the total length of SMBR sides are summed and then the mode of the distribution is taken as the main building direction.

`NON_VERT_ROOF_AREA_Hx_y`

Description: The non-vertical (horizontal and tilted) roofs area is calculated within each vertical layer of a RSU (the bottom of the layer being at `x` meters from the ground while the top is at `y` meters).

Method: The calculation is based on the assumption that all buildings having a roof height higher than the wall height have a gable roof (the other being horizontal). Since the direction of the gable is not taken into account for the moment, we consider that buildings are square in order to limit the potential calculation error (otherwise a choice should have been made to locate the line corresponding to the top of the roof).

`NON_VERT_ROOF_DENSITY`

Description: RSU surface density of non-vertical roofs (horizontal and tilted roofs).

Method: `SUM(Non_vert_roof_area_Hx_y) / RSU_Area`

back to top

`PERVIOUS_FRACTION_LCZ`

Description: Pervious fraction used for the LCZ classification.

Method: `SUM(Low_veg_Area with and without high vegetation superimposition + Water_Area with and without high vegetation` `superimposition + High_veg_Area without superimposition) / RSU_Area`

`PROJECTED_FACADE_AREA_DISTRIBUTION_Hx_y_Dw_z`

Description: Distribution of projected facade area within a RSU per vertical layer (the height being from `x` to `y`) and per direction of analysis (ie. wind or sun direction - the angle range being from `w` to `z` within the range [0, 180[°).

Method: Each line representing the facades of a building are projected in order to be perpendicular to the median of each angle range of analysis. Only free facades are considered. The projected surfaces are then summed within each layer and direction of analysis. The analysis is only performed within the [0, 180[° range since the projected facade of a building is identical for opposite directions. Thus because we consider all facades of a building in the calculation (facades upwind but also downwind), the final result is divided by 2.

Warning: To simplify the calculation, z0 is considered as equal for a given orientation independently of the direction. This assumption is right when the RSU do not split buildings but could slightly overestimate the results otherwise (the projected facade area is actually overestimated in one direction but OK in the opposite direction).

`ROAD_DIRECTION_DISTRIBUTION_H0_Dw_z`

Description: Distribution of road length within a RSU per direction of analysis (ie. wind or sun direction - the angle range being from `w` to `z` within the range [0, 180[°). Note that by default, only roads located at ground level are considered for the calculation (z_index = 0).

Method: The direction of each segment of road is calculated. The percentage of linear of road in each range of direction is then calculated (a range is defined - default 30°) for directions included in [0, 180[°.

`ROAD_FRACTION_URB`

Description: Road fraction used for the UTRF classification.

Method: `SUM(Road_Area with and without superimposition) / RSU_Area`

back to top

`STD_HEIGHT_ROOF_AREA_WEIGHTED`

Description: Variability of the building's roof height within the RSU (the building heights being weighted by the building areas)

Method: By default, the indicator of variability is the Standard Deviation (STD) defined as :

→ `SUM(Bu_Area*(Bu_Wall_Height - AVG_HEIGHT_ROOF_AREA_WEIGHTED)^2)) / SUM (Bu_Area)`

`VEGETATION_FRACTION_URB`

Description: Road fraction used for the UTRF classification.

Method: `SUM(High_veg_Area without superimposition + High_veg_Area superimposing all other layers + Low_veg_area without superimposition) / RSU_Area`

`VERT_ROOF_AREA_Hxx_xx`

Description: Vertical roofs area is calculated within each vertical layer of a RSU (the bottom of the layer being at `x` meters from the ground while the top is at `y` meters).

Method: The calculation is based on the assumption that all buildings having a roof height higher than the wall height have a gable roof (the other being horizontal). Since the direction of the gable is not taken into account for the moment, we consider that buildings are square in order to limit the potential calculation error (otherwise a choice should have been made to locate the line corresponding to the top of the roof).

`VERT_ROOF_DENSITY`

Description: RSU surface density of vertical roofs.

Method: `SUM(Vert_roof_area_Hx_y) / RSU_Area`

`WATER_FRACTION_LCZ`

Description: Water fraction used for the LCZ classification.

Method: `SUM(Water_Area with and without superimposition) / RSU_Area`

`X_FRACTION`

Description: Fraction of the X input layer within the RSU which is not superimposed with any other Y input layer (note that the vegetation layer is split into a low_vegetation layer and a high_vegetation layer). Superimposed layer fraction are calculated in 'X_Y_FRACTION' when they are physically relevant (e.g. high_vegetation above impervious). When not relevant (e.g. low_vegetation and impervious), only one of the intersected layers is kept for fraction calculation. By default, superimposition is considered only between high_vegetation and all other layers and otherwise intersected layers are kept in the following priority order: "water", "building", "high_vegetation", "low_vegetation", "road", "impervious".

Method: `SUM(X_Area without superimposition) / RSU_Area`

`X_Y_FRACTION`

Description: Fraction of the X input layer within the RSU which superimposed the Y input layer (note that the vegetation layer is split into a low_vegetation layer and a high_vegetation layer). Superimposed layer fraction are calculated when they are physically relevant (e.g. high_vegetation above impervious). By default, superimposition is considered only between high_vegetation and all other layers and otherwise intersected layers.

Method: `SUM(X_Area superimposing Y) / RSU_Area`

LCZ classification

WARNING: The article describing the full method of the LCZ attribution is currently under writing. You will soon find more informations on the References page.

GeoClimate computes the Local Climate Zones (LCZ) at the RSU's scale.

The LCZ, introduced by *Stewart* & *Oke* (2012, 2014), is a classification scheme used to segment the climate area's of cities(and other).

Methodology

A LCZ type is assigned to a RSU. This "assignment" is performed according to the 7 indicators used for LCZ classification (`sky_view_factor`, `aspect_ratio`, `building_surface_fraction`, `impervious_surface_fraction`, `pervious_surface_fraction`, `height_of_roughness_elements` and `terrain_roughness_class`). Each LCZ type has a given range for each of the 7 indicators. Then the method to find the LCZ type that is the most appropriate for a given RSU is based on the minimum distance (`MIN_DISTANCE`) to each LCZ (in the 7 dimensions space). In order to calculate this distance, each dimension is normalized according to the mean and the standard deviation (or median and absolute median deviation) of the interval values. Some of the indicators may be more important (or reliable) than the other for the LCZ identification. In order to manage this point, a map containing weights may be passed and will be used to multiply the distance due to a given indicator.

The distance of each RSU to each of the LCZ types is calculated in the normalized interval. The two LCZ types being the closest to the RSU indicators (`LCZ_PRIMARY` and `LCZ_SECONDARY`) are associated to this RSU. Three indicators are also used to show the degree of certainty of the allocated LCZ class:

- *MIN_DISTANCE*: it is the distance from a RSU point to the closest LCZ type (the lower the more certain the LCZ_PRIMARY value)
- *LCZ_UNIQUENESS_VALUE*: indicates how sure is the LCZ type attributed as primary value (the closest from 1 the more certain the LCZ_PRIMARY value)
- *LCZ_EQUALITY_VALUE*: indicates whether the LCZ type of a RSU could be any LCZ type (the closest from 0 the more certain the LCZ_PRIMARY value).

Note that this method is only valid for most of the built LCZ types. For LCZ types 8, 10 and all land-cover LCZ types, the method is slightly different and will be further described in the article available soon in the References page (the LCZ classification source code is available here for those who can not wait).

Output LCZ layer

Field name	Field type	Definition
ID_RSU	integer	RSU's unique id
LCZ_PRIMARY	integer	Main LCZ type
LCZ_SECONDARY	integer	Secondary LCZ type
MIN_DISTANCE	double precision	Minimum distance to each LCZ
LCZ_UNIQUENESS_VALUE	double precision	Indicates how unique is the attributed LCZ type
LCZ_EQUALITY_VALUE	double precision	indicates whether the LCZ type of a RSU could be any LCZ type

LCZ_PRIMARY and LCZ_SECONDARY column values

Each LCZ value is encoded using the following Type code.

For each of them, we give the LCZ class name and an hexadecimal color code to build map.

Type	Type definition	Hexa Color code
`1`	LCZ 1: Compact high-rise	`#8b0101`
`2`	LCZ 2: Compact mid-rise	`#cc0200`
`3`	LCZ 3: Compact low-rise	`#fc0001`

Type	Type definition	Hexa Color code
`4`	LCZ 4: Open high-rise	`#be4c03`
`5`	LCZ 5: Open mid-rise	`#ff6602`
`6`	LCZ 6: Open low-rise	`#ff9856`
`7`	LCZ 7: Lightweight low-rise	`#fbed08`
`8`	LCZ 8: Large low-rise	`#bcbcba`
`9`	LCZ 9: Sparsely built	`#ffcca7`
`10`	LCZ 10: Heavy industry	`#57555a`
`101`	LCZ A: Dense trees	`#006700`
`102`	LCZ B: Scattered trees	`#05aa05`
`103`	LCZ C: Bush,scrub	`#648423`
`104`	LCZ D: Low plants	`#bbdb7a`
`105`	LCZ E: Bare rock or paved	`#010101`
`106`	LCZ F: Bare soil or sand	`#fdf6ae`
`107`	LCZ G: Water	`#6d67fd`

UTRF classification

Based on (Tornay *et al*. 2017) method, GeoClimate classifies the RSU areas according urban classes.

The Urban Typology by Random Forest (UTRF) is defined in table below with a proposed color to map them.

Type	Type definition	Hexa Color code
`ba`	Industrial building	`#8f8f8f`
`bgh`	High-rise building	`#000d00`
`icif`	Block of buildings on closed urban islet	`#d52623`
`icio`	Block of buildings on open urban islet	`#f07923`
`id`	Detached building	`#eccb27`
`local`	Informal building	`#d728ac`
`pcif`	Residential on closed islet	`#2b6724`
`pcio`	Residential on open islet	`#36884a`
`pd`	Detached house	`#22be2f`
`psc`	Semi-detached house	`#05ff58`
-	Undefined	`#ffffff`

Output urban typologies layer

Field name	Field type	Definition
ID_RSU	INTEGER	RSU's unique id
THE_GEOM	GEOMETRY	RSU'S geometry
TYPO_MAJ	VARCHAR	Main urban typology
UNIQUENESS_VALUE	DOUBLE	The value of the uniqueness main class for the RSU

Road indicators

GeoClimate offers a processing chain to compute road traffic indicators based on the table Tool 2.5 (WG-AEN method) described in Good Practice Guide for Strategic Noise Mapping and the Production of Associated Data on Noise Exposure Version 2 13th January 2006. The chain uses the GeoClimate road layer and a configuration file that contains informations to establish relations between the WG-AEN referential and the road layer features :

- a mapping from WG-AEN road types to the road layer types (WG type)
- the CNOSSOS-EU pavement codes according the surface values available in the road layer (WG pavement)
- the Tool 2.5 flow data by WG-AEN road types for the 3 periods day, night, evening (WG data flow)
- the maximum speed value according the WG-AEN road types (WG maxspeed)

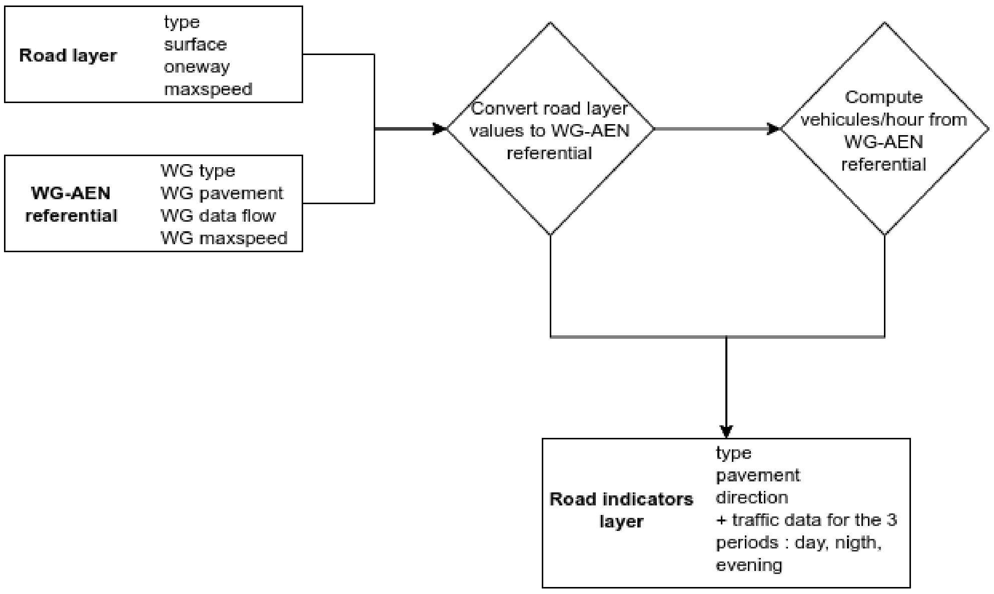

Road layer
type
surface
oneway
maxspeed
WG-AEN referential
WG type
WG pavement
WG data flow
WG maxspeed
Convert road layer values to WG-AEN referential
Compute vehicules/hour from WG-AEN referential
Road indicators layer
type
pavement
direction
+ traffic data for the 3 periods : day, nigth, evening

Figure 3. Processing steps to compute a road traffic flow based on WG-AEN referential

For each road geometry, the road layer generic fields "type", "surface", "oneway" and "maxspeed" are wrapped to the WG type, pavement and direction defined in the WG-AEN referential.Those values are then are intersected with the WG data flow to compute the number of light and heavy vehicles per hour for the 3 time periods : day (06:00-18:00), evening (ev) (18:00-22:00) and night (22:00-06:00).

The 16 resulting indicators are stored in the "road_indicators" table:

- the WG-AEN road type,
- the pavement code,
- the direction of the road section. 1 = one way road section and the traffic goes in the same way that the slope definition you have used, 2 = one way road section and the traffic goes in the inverse way that the slope definition you have used, 3 = bi-directional traffic flow, the flow is split into two components and correct half for uphill and half for downhill,
- the number of light vehicles per hour for day,
- the number of heavy vehicles per hour for day,
- the light vehicles speed for day,
- the heavy vehicles speed for day,
- the number of light vehicles per hour for night,

- the number of heavy vehicles per hour for night,
- the light vehicles speed for night,
- the heavy vehicles speed for night,
- the number of light vehicles per hour for evening,
- the number of heavy vehicles per hour for evening,
- the light vehicles speed for evening,
- the number of heavy vehicles per hour for evening,
- the slope (in %) of the road section

Tutorials for Linux

Specific workflows have been developed to automatically use OpenStreetMap and the French BD Topo 2.2 version databases.

Two options are available : Command Line Interface (CLI) (beginner user) and Groovy (intermediate and advanced user)

Several tutorials are available for linux distributions. Please select a tutorial in the table below.

Actions	OpenStreetMap	BDTopo 2.2 version
Default case : Calculate LCZ, TEB inputs and UTRF	CLI / Groovy	CLI / Groovy

Default case with OSM

This tutorial presents how to create Local Climate Zones with OpenStreetMap data.

Two tools are available to run GeoClimate algorithms: Command Line Interface (beginner user) and Groovy (intermediate and advanced user)

Command Line Interface

Get Geoclimate.jar on your computer

You will run the archive Geoclimate.jar in a Command Line Interface.

First, make sure Java (version 8 minimum) is installed in your computer.

You need to download Geoclimate.jar here.

Rename the downloaded file as "Geoclimate.jar".

Create a folder in your documents (for instance /home/mydirectory/Geoclimate) and place Geoclimate.jar in this folder.

Create and understand the configuration file

In order to run Geoclimate, you need to write a configuration file. This file specifies inputs, methods and outputs of Geoclimate.

An example of configuration file is presented below :

```
{
    "description": "Processing OSM data",
    "input": {
        "osm": [
            "Pont-de-Veyle"
        ]
    },
    "output": {
        "folder": "/tmp"
    },
    "parameters": {
        "rsu_indicators": {
            "indicatorUse": [
                "LCZ",
                "TEB",
                "UTRF"
            ],
            "svfSimplified": true,
            "estimateHeight": true
        },
        "grid_indicators": {
            "x_size": 100,
        "y_size": 100,
        "rowCol": false,
        "output" : "geojson",
        "indicators" :[
                 "BUILDING_FRACTION",
                 "BUILDING_HEIGHT",
                 "WATER_FRACTION",
                 "VEGETATION_FRACTION",
                 "ROAD_FRACTION",
                 "IMPERVIOUS_FRACTION",
                 "LCZ_FRACTION"
             ]
         }
    }
}
```

You can copy this example in a notebook and name it "my_first_config_file_osm.json". Place this configuration file in the same folder than Geoclimate.jar .

Understand the configuration file

The configuration file is structured in four main parts.

- "description" is a text that describes your process. You can name your process here.
- "input" specifies the input data you will use. In this example, we specify "osm" for OpenStreetMap, and we run Geoclimate for a small village

in France called Pont-de-Veyle.

- "output" specifies the format you expect for your output (here "folder") and where you want to create your output files (here in /tmp).
- "parameters" specifies the output you want to calculate based on your reference spatial units ("rsu_indicators") or on a grid ("grid_indicators").
- At RSU scale, we calculate the LCZ, the TEB inputs and the UTRF ("indicatorUse": ["LCZ", "TEB", "UTRF"]). We use the simplified method to calculate the sky view factor ("svfSimplified": true) and the method to estimate the height of buildings in OSM ("estimateHeight" : true).
- With the grid approach, we specify the grid dimensions in meters ("x_size" and "y_size") and the output format ("output" : "geojson"). Then, we specify the indicators we want to calculate for each cell of the grid ("BUILDING_FRACTION", "BUILDING_HEIGHT", "WATER_FRACTION", "VEGETATION_FRACTION", "ROAD_FRACTION", "IMPERVIOUS_FRACTION", "LCZ_FRACTION").

Run Geoclimate

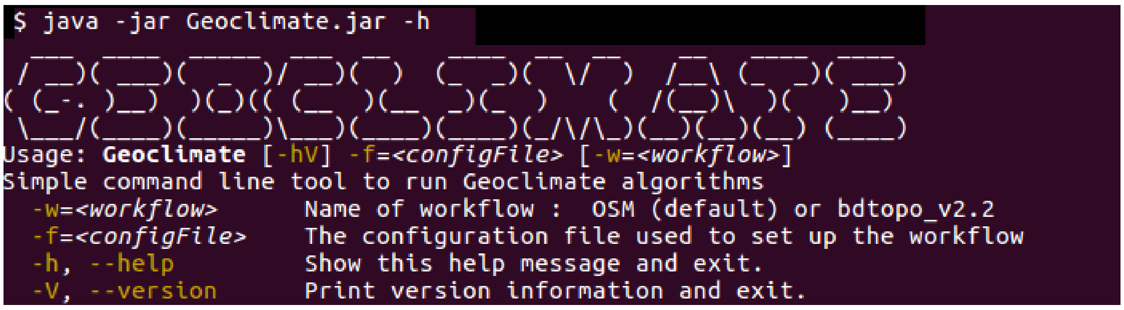
On your machine, open a terminal.

Go to the folder where Geoclimate.jar is located using this command line :

`cd /home/mydirectory/Geoclimate`

Then, you can run this command line which presents you the main options of Geoclimate :

`java -jar Geoclimate.jar -h`



```
$ java -jar Geoclimate.jar -h  [redacted]
  ___  ____  _____  ___  __    ____  __  __    __    ____  ____
 / __)( ___)(  _  )/ __)(  )  (_  _)(  \/  )  /__\  (_  _)( ___)
( (_-. )__)  )(_)(( (__  )(__  _)(_  )    (  /(__)\   )(   )__)
 \___/(____)(_____)\___)(____)(____)(_/\/\_)(__)(__) (__) (____)
Usage: Geoclimate [-hV] -f=<configFile> [-w=<workflow>]
Simple command line tool to run Geoclimate algorithms
  -w=<workflow>      Name of workflow :  OSM (default) or bdtopo_v2.2
  -f=<configFile>    The configuration file used to set up the workflow
  -h, --help         Show this help message and exit.
  -V, --version      Print version information and exit.
```

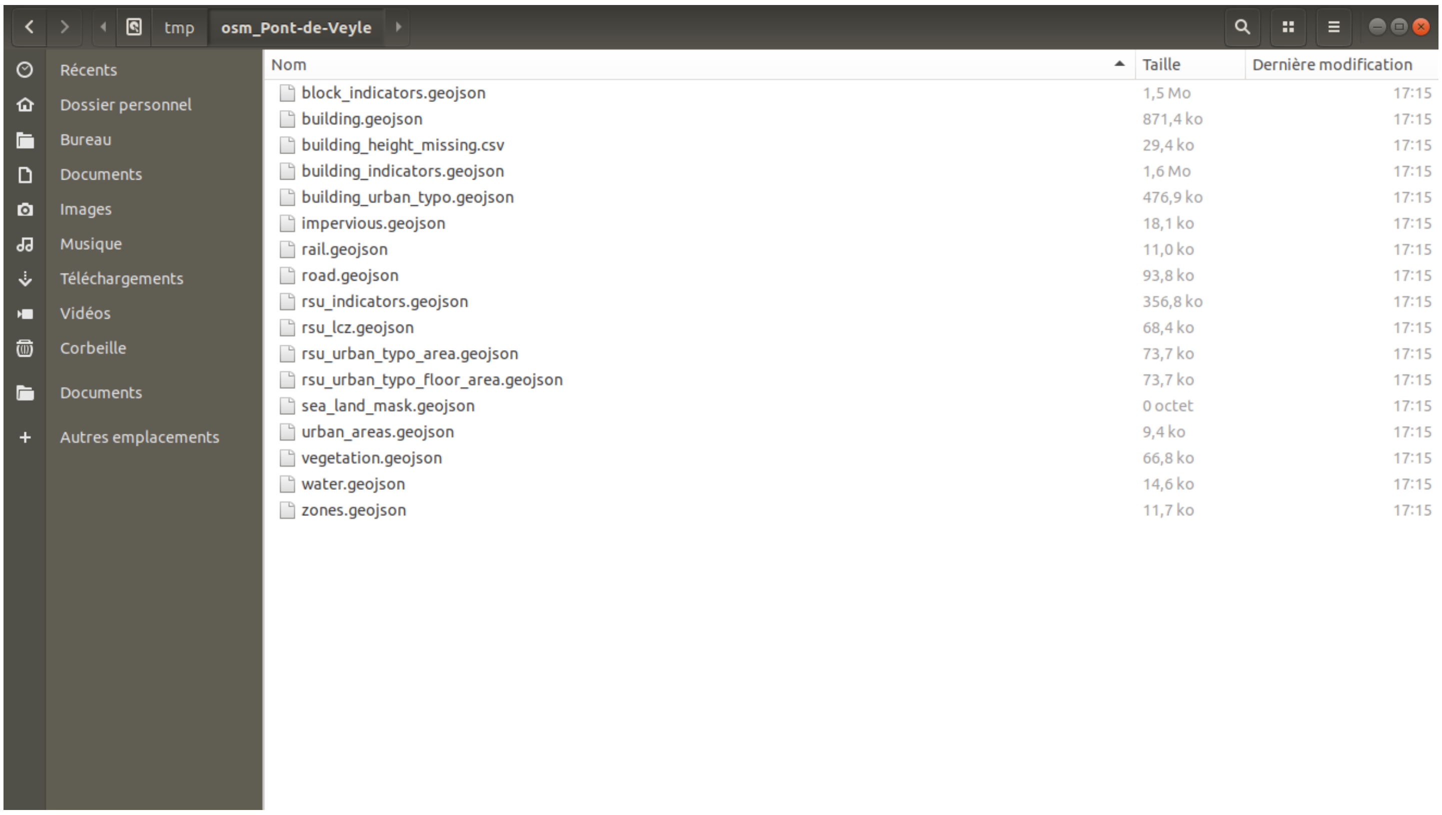
In order to perform your first calculations with the configuration file above, use

`java -jar Geoclimate.jar -f my_first_config_file_osm.json -w OSM`

where the f option is used to set the path of the configuration file.

If everything runs well, you will obtain a message : `The OSM workflow has been successfully executed`

The results of your calculations are located in you `\tmp` folder.


tmp osm_Pont-de-Veyle

Nom	Taille	Dernière modification
block_indicators.geojson	1,5 Mo	17:15
building.geojson	871,4 ko	17:15
building_height_missing.csv	29,4 ko	17:15
building_indicators.geojson	1,6 Mo	17:15
building_urban_typo.geojson	476,9 ko	17:15
impervious.geojson	18,1 ko	17:15
rail.geojson	11,0 ko	17:15
road.geojson	93,8 ko	17:15
rsu_indicators.geojson	356,8 ko	17:15
rsu_lcz.geojson	68,4 ko	17:15
rsu_urban_typo_area.geojson	73,7 ko	17:15
rsu_urban_typo_floor_area.geojson	73,7 ko	17:15
sea_land_mask.geojson	0 octet	17:15
urban_areas.geojson	9,4 ko	17:15
vegetation.geojson	66,8 ko	17:15
water.geojson	14,6 ko	17:15
zones.geojson	11,7 ko	17:15

Groovy console

Get Groovy on your computer

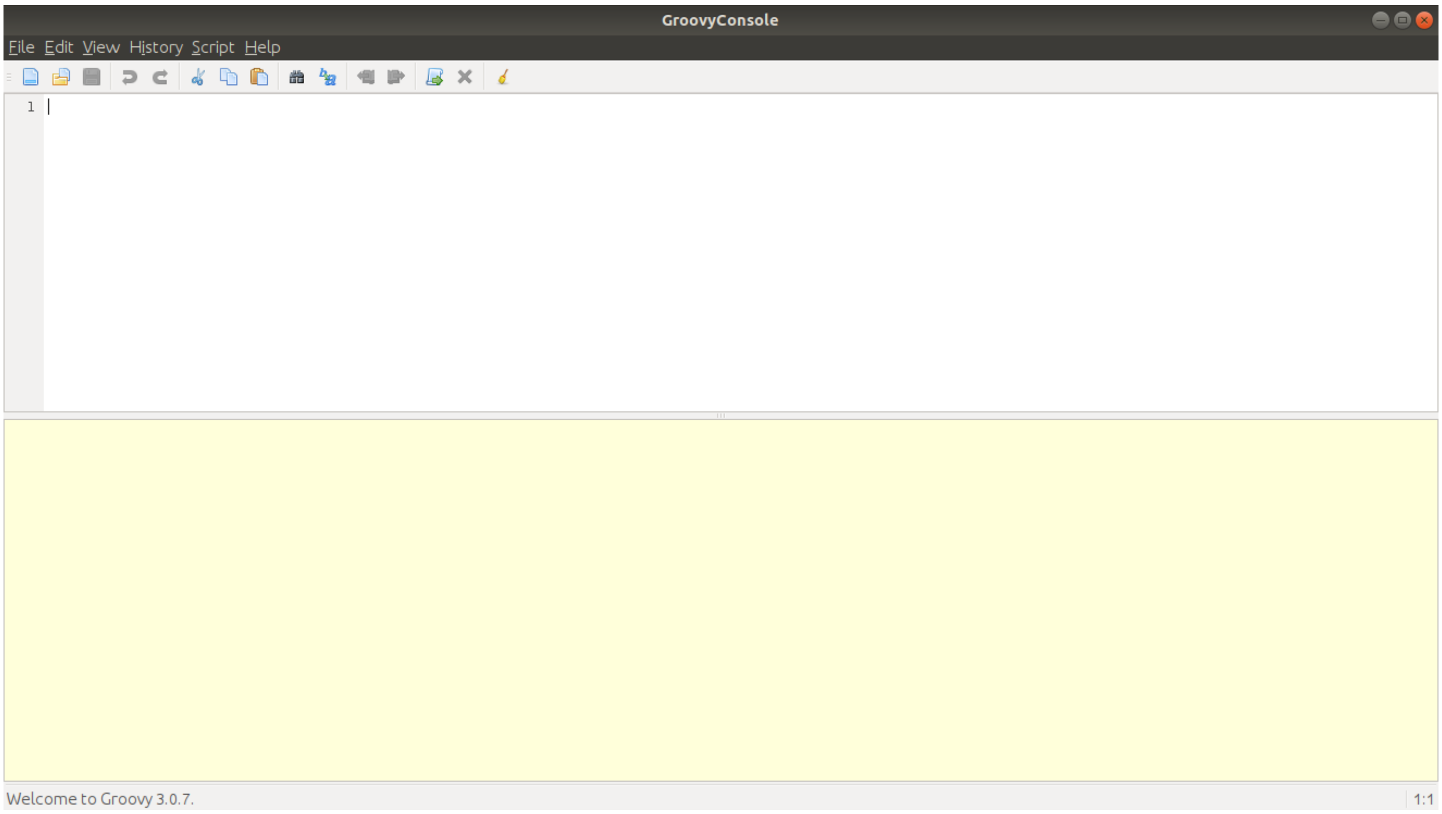
First, make sure that OpenJDK is installed in your computer.

You need to install Groovy 3.0.7. The easiest way is to use the The Software Development Kit Manager (SDKMAN).

Run Geoclimate

When Groovy is installed, open the Groovy console using this command line in a terminal :

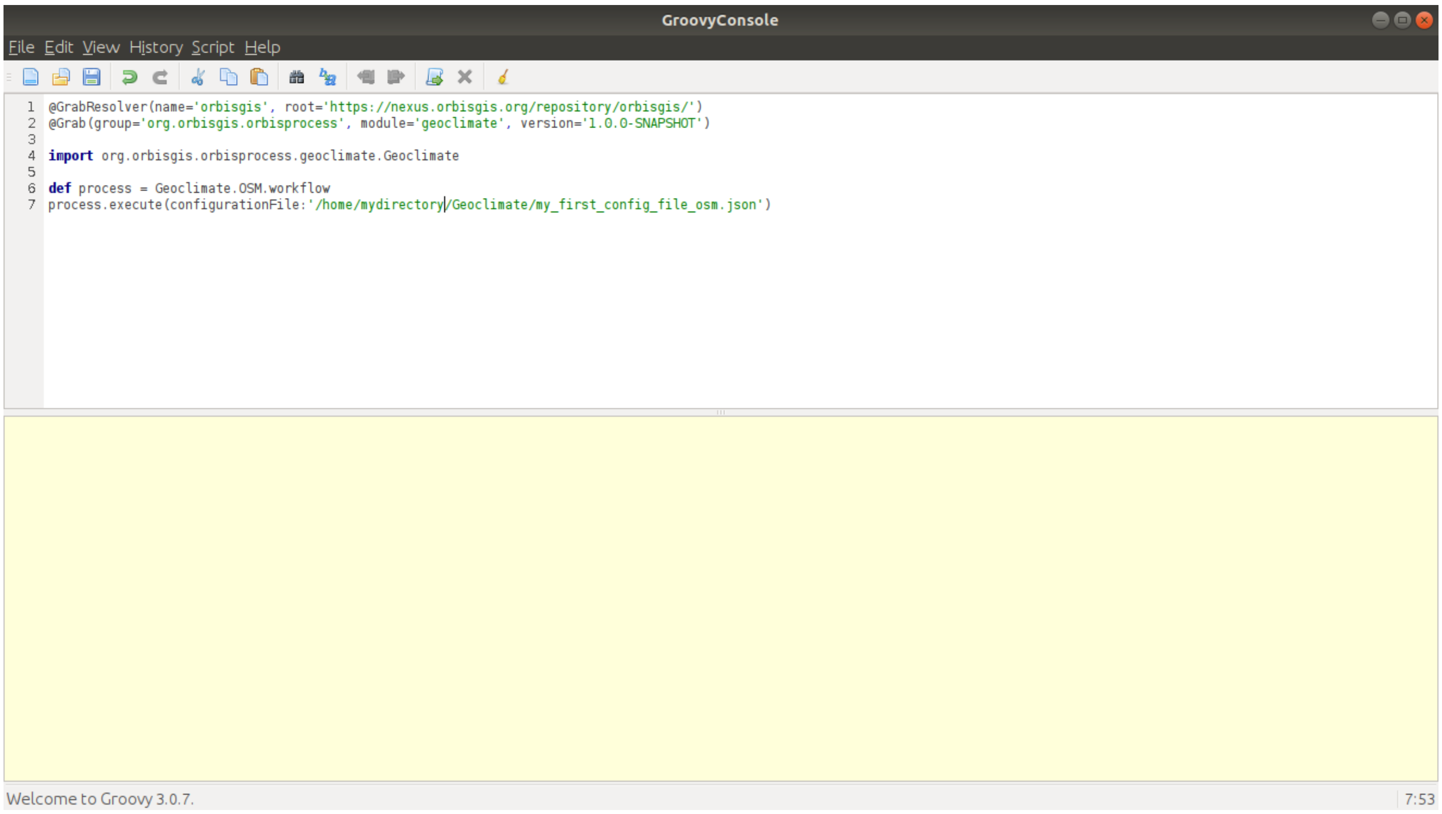
```
groovyConsole
```



In the Groovy console, you can copy/paste the following script :

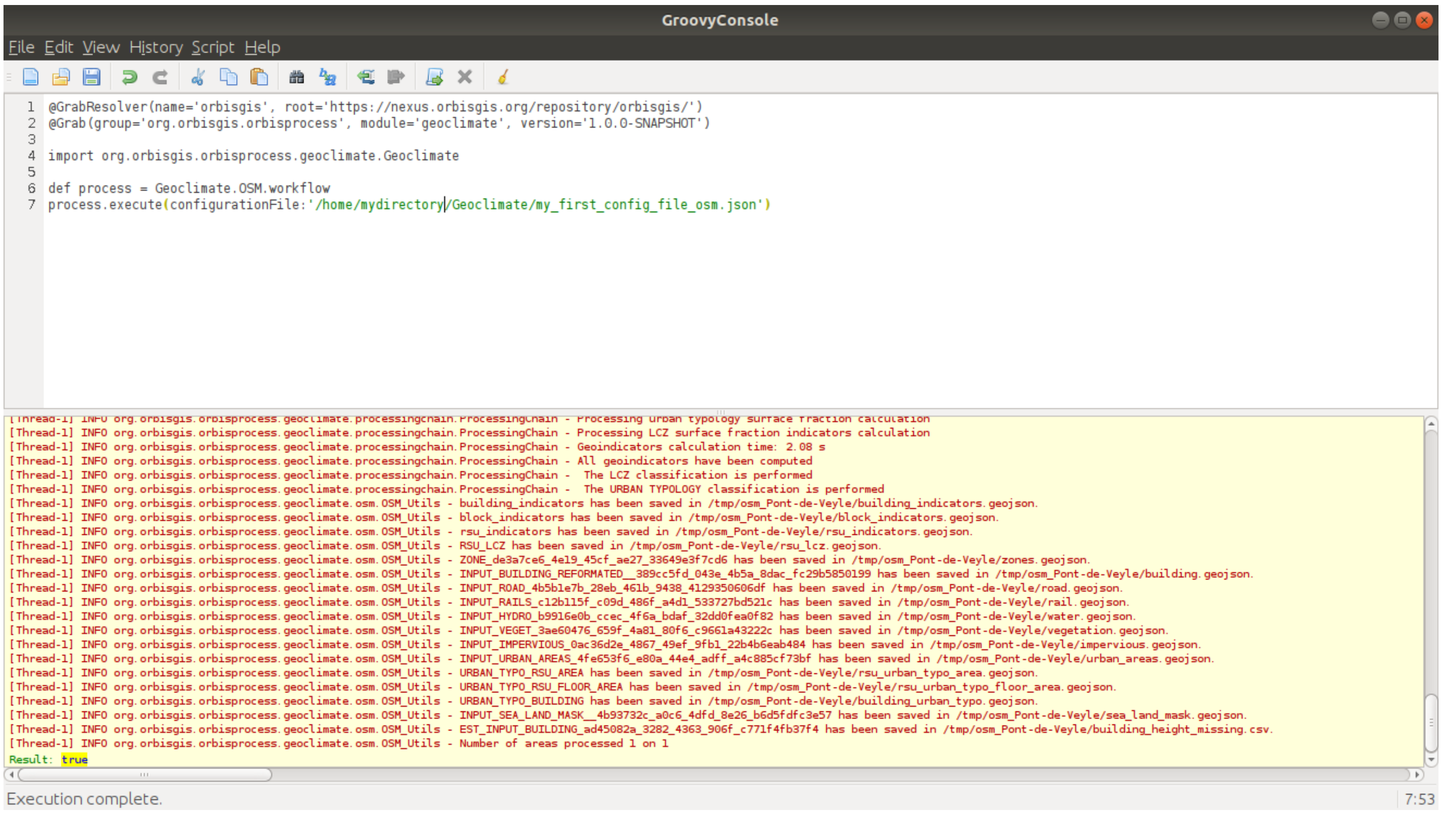
```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.geoclimate', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.geoclimate.Geoclimate

def process = Geoclimate.OSM.workflow
process.execute(configurationFile:'/home/mydirectory/Geoclimate/my_first_config_file_osm.json')
```

This script will run Geoclimate using the configuration file `my_first_config_file_osm.json`. Please see here for more explanations.



```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.orbisprocess', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.orbisprocess.geoclimate.Geoclimate

def process = Geoclimate.OSM.workflow
process.execute(configurationFile:'/home/mydirectory/Geoclimate/my_first_config_file_osm.json')
```

You can run your script by using the shortcut `Ctrl + R` or clicking on .

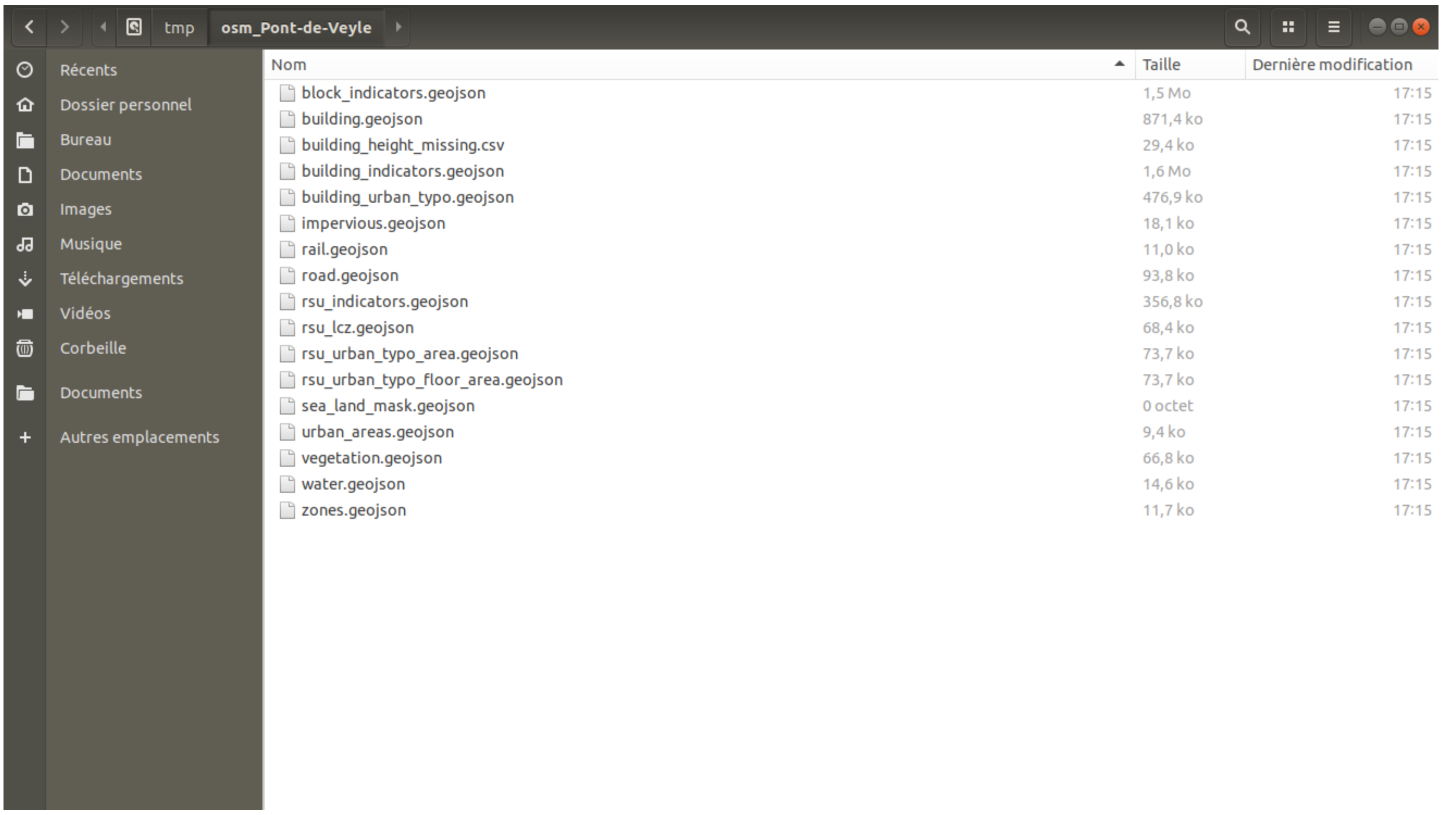


```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.orbisprocess', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.orbisprocess.geoclimate.Geoclimate

def process = Geoclimate.OSM.workflow
process.execute(configurationFile:'/home/mydirectory/Geoclimate/my_first_config_file_osm.json')
```

```
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Processing urban typology surface fraction calculation
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Processing LCZ surface fraction indicators calculation
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Geoindicators calculation time: 2.08 s
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - All geoindicators have been computed
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain -  The LCZ classification is performed
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain -  The URBAN TYPOLOGY classification is performed
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - building_indicators has been saved in /tmp/osm_Pont-de-Veyle/building_indicators.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - block_indicators has been saved in /tmp/osm_Pont-de-Veyle/block_indicators.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - rsu_indicators has been saved in /tmp/osm_Pont-de-Veyle/rsu_indicators.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - RSU_LCZ has been saved in /tmp/osm_Pont-de-Veyle/rsu_lcz.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - ZONE_de3a7ce6_4e19_45cf_ae27_33649e3f7cd6 has been saved in /tmp/osm_Pont-de-Veyle/zones.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - INPUT_BUILDING_REFORMATED__389cc5fd_043e_4b5a_8dac_fc29b5850199 has been saved in /tmp/osm_Pont-de-Veyle/building.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - INPUT_ROAD_4b5b1e7b_28eb_461b_9438_4129350606df has been saved in /tmp/osm_Pont-de-Veyle/road.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - INPUT_RAILS_c12b115f_c09d_486f_a4d1_533727bd521c has been saved in /tmp/osm_Pont-de-Veyle/rail.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - INPUT_HYDRO_b9916e0b_ccec_4f6a_bdaf_32dd0fea0f82 has been saved in /tmp/osm_Pont-de-Veyle/water.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - INPUT_VEGET_3ae60476_659f_4a81_80f6_c9661a43222c has been saved in /tmp/osm_Pont-de-Veyle/vegetation.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - INPUT_IMPERVIOUS_0ac36d2e_4867_49ef_9fb1_22b4b6eab484 has been saved in /tmp/osm_Pont-de-Veyle/impervious.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - INPUT_URBAN_AREAS_4fe653f6_e80a_44e4_adff_a4c885cf73bf has been saved in /tmp/osm_Pont-de-Veyle/urban_areas.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - URBAN_TYPO_RSU_AREA has been saved in /tmp/osm_Pont-de-Veyle/rsu_urban_typo_area.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - URBAN_TYPO_RSU_FLOOR_AREA has been saved in /tmp/osm_Pont-de-Veyle/rsu_urban_typo_floor_area.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - URBAN_TYPO_BUILDING has been saved in /tmp/osm_Pont-de-Veyle/building_urban_typo.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - INPUT_SEA_LAND_MASK__4b93732c_a0c6_4dfd_8e26_b6d5fdfc3e57 has been saved in /tmp/osm_Pont-de-Veyle/sea_land_mask.geojson.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - EST_INPUT_BUILDING_ad45082a_3282_4363_906f_c771f4fb37f4 has been saved in /tmp/osm_Pont-de-Veyle/building_height_missing.csv.
[Thread-1] INFO org.orbisgis.orbisprocess.geoclimate.osm.OSM_Utils - Number of areas processed 1 on 1
Result: true
```

Execution complete. 7:53

The results of your calculations are located in you `\tmp` folder.



Nom	Taille	Dernière modification
block_indicators.geojson	1,5 Mo	17:15
building.geojson	871,4 ko	17:15
building_height_missing.csv	29,4 ko	17:15
building_indicators.geojson	1,6 Mo	17:15
building_urban_typo.geojson	476,9 ko	17:15
impervious.geojson	18,1 ko	17:15
rail.geojson	11,0 ko	17:15
road.geojson	93,8 ko	17:15
rsu_indicators.geojson	356,8 ko	17:15
rsu_lcz.geojson	68,4 ko	17:15
rsu_urban_typo_area.geojson	73,7 ko	17:15
rsu_urban_typo_floor_area.geojson	73,7 ko	17:15
sea_land_mask.geojson	0 octet	17:15
urban_areas.geojson	9,4 ko	17:15
vegetation.geojson	66,8 ko	17:15
water.geojson	14,6 ko	17:15
zones.geojson	11,7 ko	17:15

Default case with BDTopo 2.2

This tutorial presents how to create Local Climate Zones with BD Topo 2.2 version.

Two tools are available : Command Line Interface (beginner user) and Groovy (intermediate and advanced user)

Command Line Interface

Get Geoclimate.jar on your computer

You will run the archive Geoclimate.jar in a Command Line Interface.

First, make sure Java (version 8 minimum) is installed in your computer.

You need to download Geoclimate.jar here.

Rename the downloaded file as "Geoclimate.jar".

Create a folder in your documents (for instance /home/mydirectory/Geoclimate) and place Geoclimate.jar in this folder.

Get data from BD Topo 2.2 on your computer

Collect all your BD Topo 2.2 data in one single folder. The required files are listed here.

Create a subfolder (for instance /home/mydirectory/Geoclimate/BD_TOPO_v2) and place your BD Topo 2.2 data in this folder.

Create and understand a configuration file

In order to run Geoclimate, you need to write a configuration file. This file specifies inputs, methods and outputs of Geoclimate.

An example of configuration file is presented below :

```
{
    "description": "Processing BD Topo v2 data",
    "input": {"bdtopo_v2":    {
        "folder": {"path": "/home/mydirectory/Geoclimate/BD_TOPO_v2/"}
                }
    },
    "output": {
        "folder": "/tmp"
    },
    "parameters": {
        "rsu_indicators": {
            "indicatorUse": [
                "LCZ",
                "TEB",
                "UTRF"
            ],
            "svfSimplified": true
        },
        "grid_indicators": {
            "x_size": 100,
        "y_size": 100,
        "rowCol": false,
        "output" : "geojson",
        "indicators" :[
                 "BUILDING_FRACTION",
                 "BUILDING_HEIGHT",
                 "WATER_FRACTION",
                 "VEGETATION_FRACTION",
                 "ROAD_FRACTION",
                 "IMPERVIOUS_FRACTION",
                 "LCZ_FRACTION"
             ]
         }
    }
}
```

You can copy this example in a notebook and name it "my_first_config_file_bdtopov2.json" . Place this configuration file in the same folder than Geoclimate.jar .

Understand the configuration file

The configuration file is structured in four main parts.

- "description" is a character string that describes your process. You can name your process here.
- "input" specifies the input data you will use. In this example, we specify "folder" for BDTopo 2.2 version, and we specify where the BD Topo 2.2 files are located on the computer ("/home/mydirectory/Geoclimate/BD_TOPO_v2/").
- "output" specifies the format you expect for your output (here "folder") and where you want to create your output files (here in /tmp).
- "parameters" specifies the output you want to calculate based on your reference spatial units ("rsu_indicators") or on a grid ("grid_indicators").
- At RSU scale, we calculate the LCZ, the TEB inputs and the UTRF ("indicatorUse": ["LCZ", "TEB", "UTRF"]). We use the simplified method to calculate the sky view factor ("svfSimplified": true).
- With the grid approach, we specify the grid dimensions in meters ("x_size" and "y_size") and the output format ("output" : "geojson"). Then, we specify the indicators we want to calculate for each cell of the grid ("BUILDING_FRACTION", "BUILDING_HEIGHT", "WATER_FRACTION", "VEGETATION_FRACTION", "ROAD_FRACTION", "IMPERVIOUS_FRACTION", "LCZ_FRACTION").

Run Geoclimate

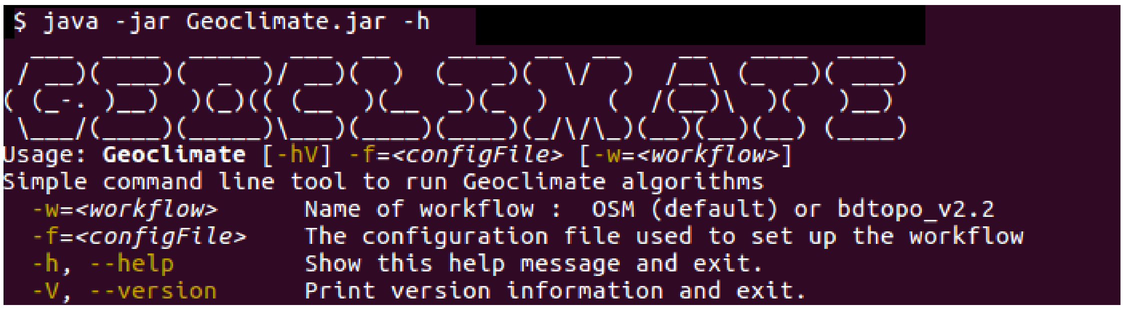
Open a Command Line Interface.

Go to the folder where Geoclimate.jar is located using this command line :

```
cd /home/mydirectory/Geoclimate
```

Then, you can run this command line which presents you the main options of Geoclimate :

```
java -jar Geoclimate.jar -h
```



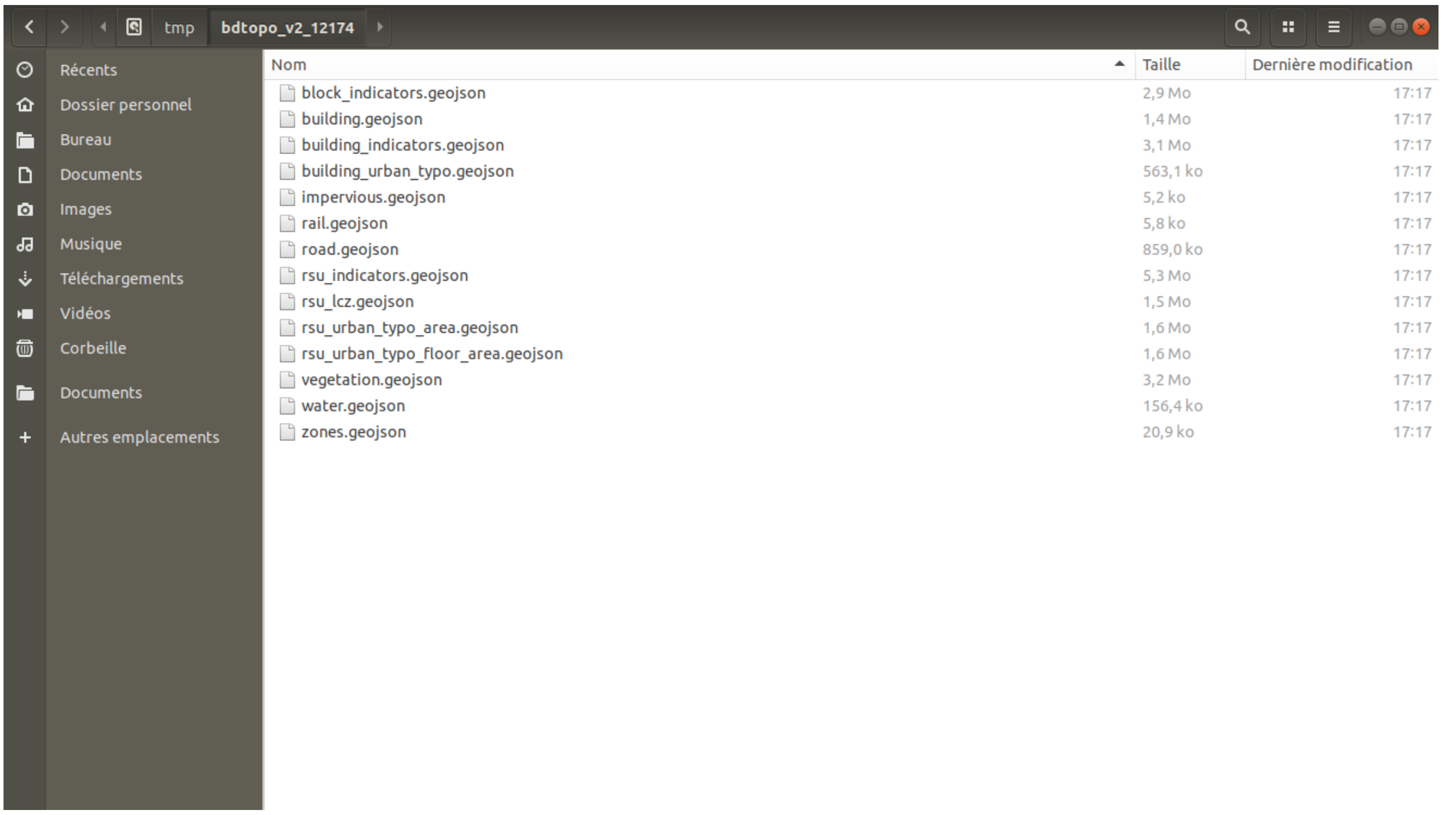
```
$ java -jar Geoclimate.jar -h    [redacted]
  ___  ____  _____  ___  __    ____  __  __    __   ____  ____
 / __)( ___)(  _  )/ __)(  )  (_  _)(  \/  )  /__\ (_  _)( ___)
( (_-. )__)  )(_)(( (__  )(__  _)(_  )    (  /(__)\  )(   )__)
 \___/(____)(_____)\___)(____)(____)(_/\/\_)(__)(__)(__) (____)
Usage: Geoclimate [-hV] -f=<configFile> [-w=<workflow>]
Simple command line tool to run Geoclimate algorithms
  -w=<workflow>      Name of workflow :  OSM (default) or bdtopo_v2.2
  -f=<configFile>    The configuration file used to set up the workflow
  -h, --help         Show this help message and exit.
  -V, --version      Print version information and exit.
```

In order to perform your first calculations with the configuration file above, use

```
java -jar Geoclimate.jar -f my_first_config_file_bdtopov2.json -w BDTOPO_V2.2
```

If everything runs well, you will obtain a message : `The BDTOPO_V2.2 workflow has been successfully executed`

The results of your calculations are located in you `\tmp` folder.



Nom	Taille	Dernière modification
block_indicators.geojson	2,9 Mo	17:17
building.geojson	1,4 Mo	17:17
building_indicators.geojson	3,1 Mo	17:17
building_urban_typo.geojson	563,1 ko	17:17
impervious.geojson	5,2 ko	17:17
rail.geojson	5,8 ko	17:17
road.geojson	859,0 ko	17:17
rsu_indicators.geojson	5,3 Mo	17:17
rsu_lcz.geojson	1,5 Mo	17:17
rsu_urban_typo_area.geojson	1,6 Mo	17:17
rsu_urban_typo_floor_area.geojson	1,6 Mo	17:17
vegetation.geojson	3,2 Mo	17:17
water.geojson	156,4 ko	17:17
zones.geojson	20,9 ko	17:17

Groovy console

Get Groovy on your computer

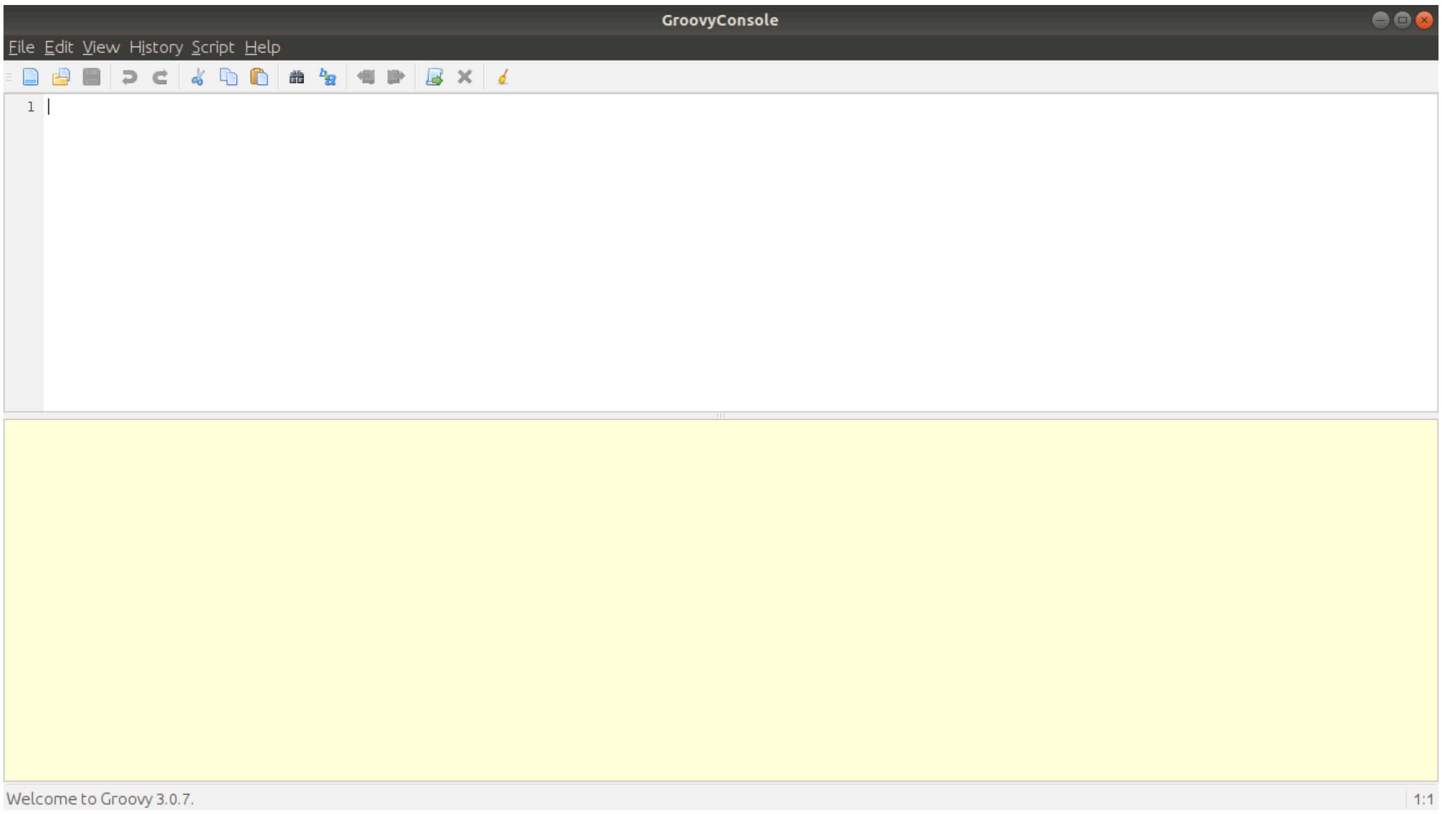
First, make sure that OpenJDK is installed in your computer.

You need to install Groovy 3.0.7. The easiest way is to use the The Software Development Kit Manager (SDKMAN).

Run Geoclimate

When Groovy is installed, open the Groovy console using this command line in a terminal :

```
groovyConsole
```

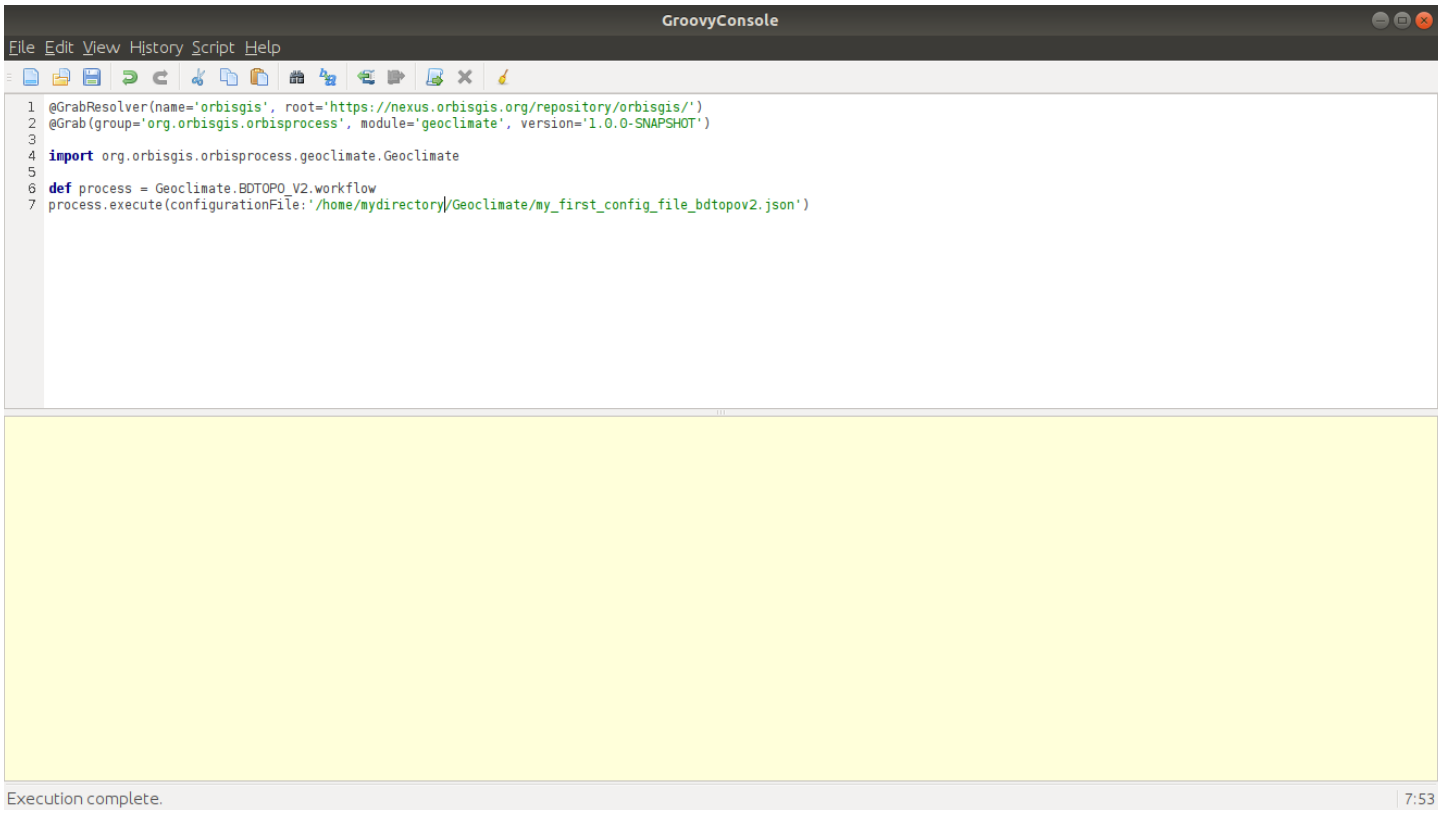


```
Welcome to Groovy 3.0.7.                                                   1:1
```

In the Groovy console, you can copy/paste the following script :

```groovy
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.geoclimate', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.geoclimate.Geoclimate

def process = Geoclimate.BDTopo_V2.workflow
process.execute(configurationFile:'/home/mydirectory/Geoclimate/my_first_config_file_bdtopov2.json')
```

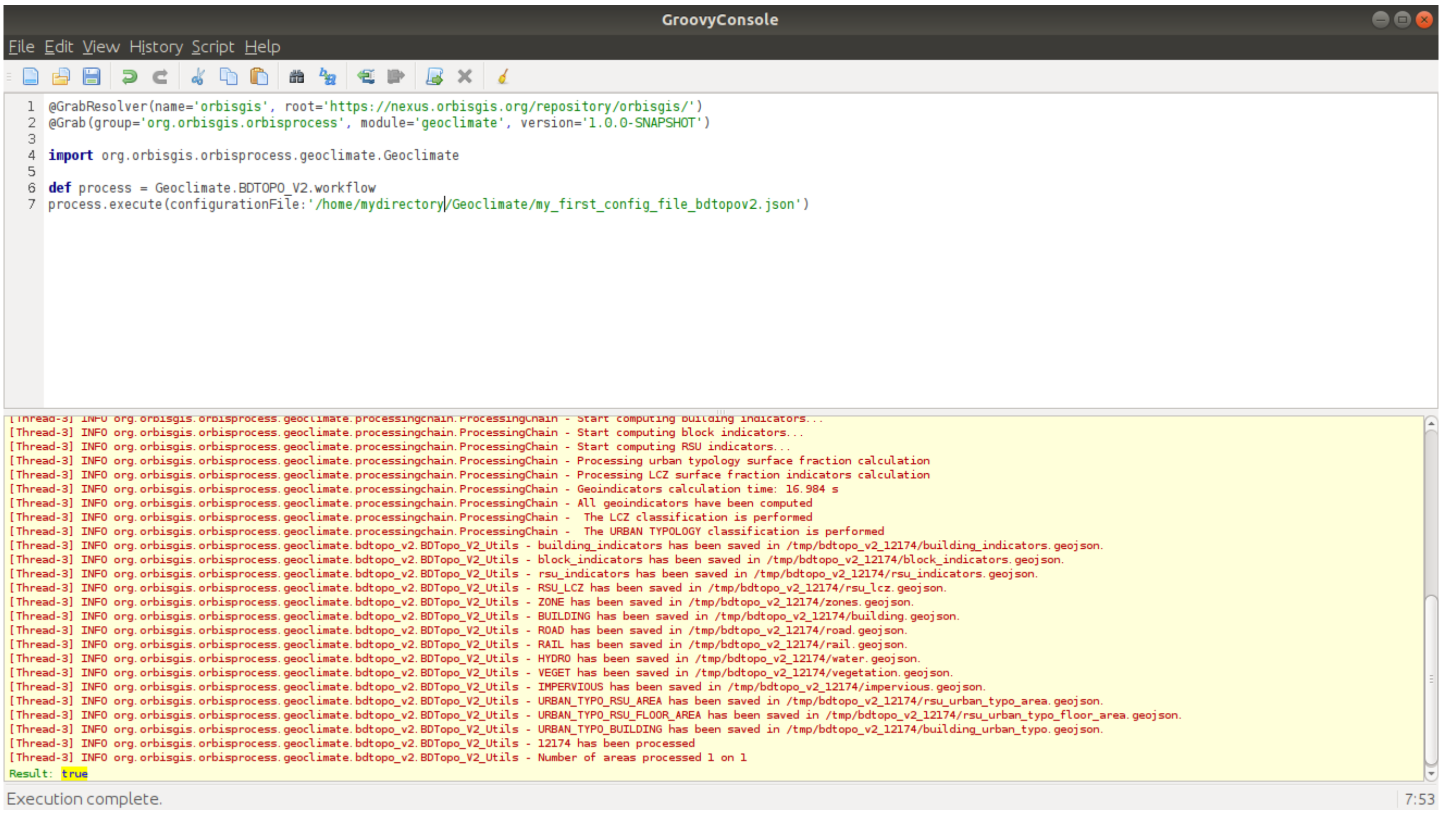
This script will run Geoclimate using the configuration file `my_first_config_file_bdtopov2.json`. Please see here for more explanations.



```groovy
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.orbisprocess', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.orbisprocess.geoclimate.Geoclimate

def process = Geoclimate.BDTOPO_V2.workflow
process.execute(configurationFile:'/home/mydirectory/Geoclimate/my_first_config_file_bdtopov2.json')
```

```
Execution complete.                                                        7:53
```

You can run your script by using the shortcut `Ctrl + R` or clicking on .

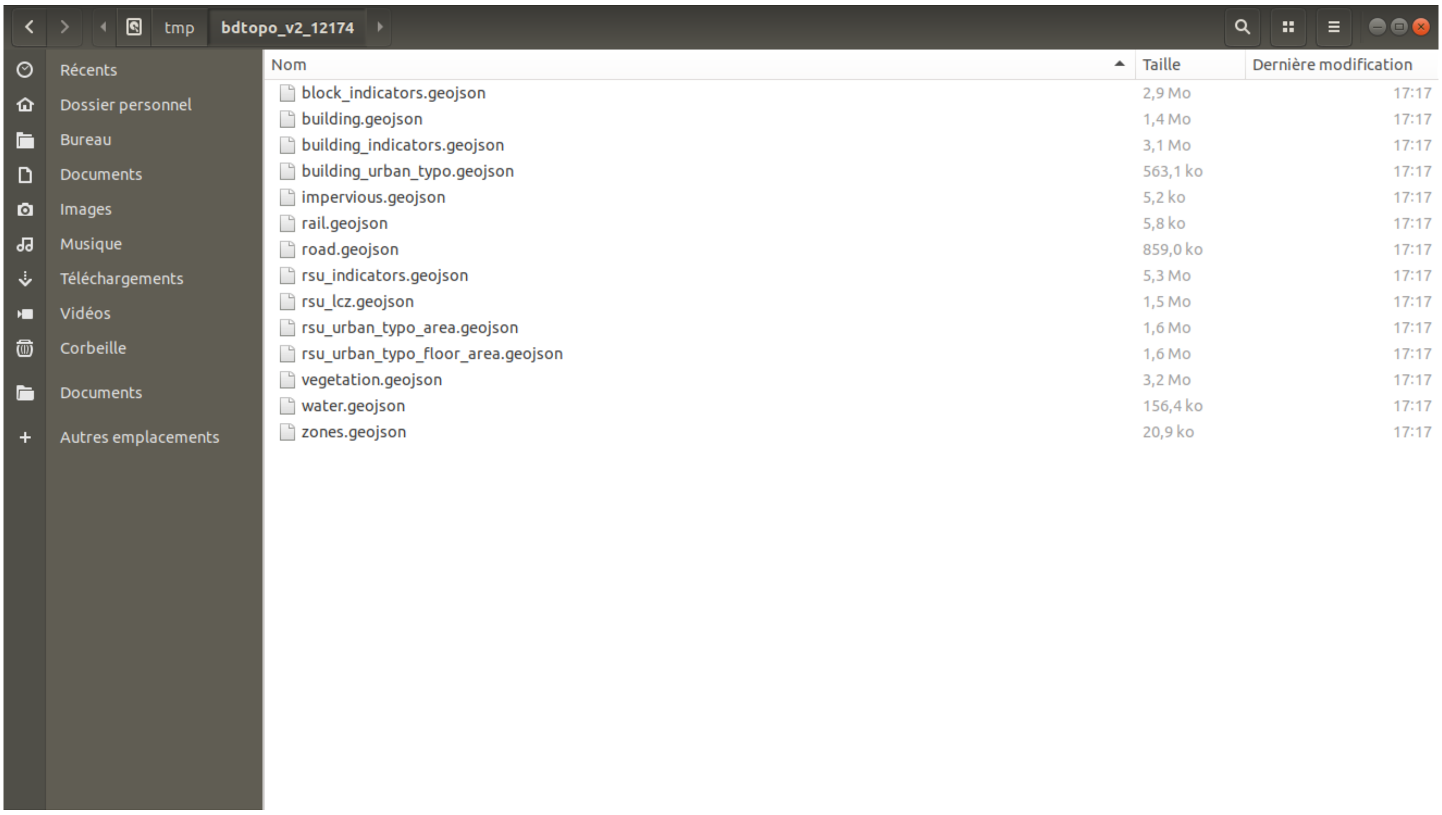


```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.orbisprocess', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.orbisprocess.geoclimate.Geoclimate

def process = Geoclimate.BDTOPO_V2.workflow
process.execute(configurationFile:'/home/mydirectory/Geoclimate/my_first_config_file_bdtopov2.json')
```

```
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Start computing building indicators...
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Start computing block indicators...
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Start computing RSU indicators...
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Processing urban typology surface fraction calculation
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Processing LCZ surface fraction indicators calculation
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - Geoindicators calculation time: 16.984 s
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain - All geoindicators have been computed
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain -  The LCZ classification is performed
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.processingchain.ProcessingChain -  The URBAN TYPOLOGY classification is performed
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - building_indicators has been saved in /tmp/bdtopo_v2_12174/building_indicators.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - block_indicators has been saved in /tmp/bdtopo_v2_12174/block_indicators.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - rsu_indicators has been saved in /tmp/bdtopo_v2_12174/rsu_indicators.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - RSU_LCZ has been saved in /tmp/bdtopo_v2_12174/rsu_lcz.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - ZONE has been saved in /tmp/bdtopo_v2_12174/zones.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - BUILDING has been saved in /tmp/bdtopo_v2_12174/building.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - ROAD has been saved in /tmp/bdtopo_v2_12174/road.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - RAIL has been saved in /tmp/bdtopo_v2_12174/rail.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - HYDRO has been saved in /tmp/bdtopo_v2_12174/water.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - VEGET has been saved in /tmp/bdtopo_v2_12174/vegetation.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - IMPERVIOUS has been saved in /tmp/bdtopo_v2_12174/impervious.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - URBAN_TYPO_RSU_AREA has been saved in /tmp/bdtopo_v2_12174/rsu_urban_typo_area.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - URBAN_TYPO_RSU_FLOOR_AREA has been saved in /tmp/bdtopo_v2_12174/rsu_urban_typo_floor_area.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - URBAN_TYPO_BUILDING has been saved in /tmp/bdtopo_v2_12174/building_urban_typo.geojson.
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - 12174 has been processed
[Thread-3] INFO org.orbisgis.orbisprocess.geoclimate.bdtopo_v2.BDTopo_V2_Utils - Number of areas processed 1 on 1
Result: true
```

Execution complete. 7:53

The results of your calculations are located in you `\tmp` folder.



Nom	Taille	Dernière modification
block_indicators.geojson	2,9 Mo	17:17
building.geojson	1,4 Mo	17:17
building_indicators.geojson	3,1 Mo	17:17
building_urban_typo.geojson	563,1 ko	17:17
impervious.geojson	5,2 ko	17:17
rail.geojson	5,8 ko	17:17
road.geojson	859,0 ko	17:17
rsu_indicators.geojson	5,3 Mo	17:17
rsu_lcz.geojson	1,5 Mo	17:17
rsu_urban_typo_area.geojson	1,6 Mo	17:17
rsu_urban_typo_floor_area.geojson	1,6 Mo	17:17
vegetation.geojson	3,2 Mo	17:17
water.geojson	156,4 ko	17:17
zones.geojson	20,9 ko	17:17

Bounding box case

The default cases for OSM and BDTOPO v2.2 are presented in other pages. Please make sure that the default cases are running on your computer before using the bounding box method.

In the default case, the area of interest is specified using a city name. You also have the possibility to calculate all the GeoClimate outputs inside a given rectangle (i.e. a bounding box).

The only difference with the default case is that geographic coordinates replace the city name. All the other elements of the configuration file remain unchanged.

The coordinates of the bounding box are expressed as [minY, minX, maxY, maxX].

Bounding box method with OSM

A bounding box has been determined for the city of Pont-de-Veyle, with the following coordinates : 46.257330,4.870033,46.269970,4.905224

The configuration file below uses this bounding box method with OSM.

```
{
    "description": "Processing OSM data",
    "input": {
        "osm": [
            [46.257330,4.870033,46.269970,4.905224]
        ]
    },
    "output": {
        "folder": "/tmp"
    },
    "parameters": {
        "rsu_indicators": {
            "indicatorUse": [
                "LCZ",
                "TEB",
                "UTRF"
            ],
            "svfSimplified": true,
            "estimateHeight": true
        },
        "grid_indicators": {
            "x_size": 100,
        "y_size": 100,
        "rowCol": false,
        "output" : "geojson",
        "indicators" :[
                 "BUILDING_FRACTION",
                 "BUILDING_HEIGHT",
                 "WATER_FRACTION",
                 "VEGETATION_FRACTION",
                 "ROAD_FRACTION",
                 "IMPERVIOUS_FRACTION",
                 "LCZ_FRACTION"
             ]
         }
    }
}
```

In order to determine the coordinates of your bounding box, you can use the bboxfinder website. Make sure your coordinates are in latitude / longitude ("Lat / Lon") format. You can choose the coordinates options on the bottom right of the bboxfinder website.

Bounding box method with BDTOPO v2.2

The configuration file below uses a bounding box method with BDTOPO v2.2.

This bounding box represents the envelope of the city of Redon.

Note that the EPSG code for the projection system here is 2154 and not "Lat / Lon" anymore.

```
{
    "description": "Processing BDTopo v 2.2 data",
    "input": {
        "bdtopo_v2": {
            "folder": {
                "path": "/home/mydirectory/Geoclimate/BD_TOPO_v2/",
                "id_zones": [
                    [
                        6737756.724564202,
                        316124.01010211144,
                        6743486.0484706545,
                        321921.09550058335
                    ]
                ]
            }
        }
    },
    "output": {
        "folder": "/tmp"
    },
    "parameters": {
        "rsu_indicators": {
            "indicatorUse": [
                "LCZ",
                "TEB",
                "UTRF"
            ],
            "svfSimplified": true
        },
        "grid_indicators": {
            "x_size": 100,
        "y_size": 100,
        "rowCol": false,
        "output" : "geojson",
        "indicators" :[
                 "BUILDING_FRACTION",
                 "BUILDING_HEIGHT",
                 "WATER_FRACTION",
                 "VEGETATION_FRACTION",
                 "ROAD_FRACTION",
                 "IMPERVIOUS_FRACTION",
                 "LCZ_FRACTION"
             ]
         }
    }
}
```

Tutorials for Windows

Specific workflows have been developed to automatically use OpenStreetMap and the French BD Topo 2.2 version databases.

Two options are available : Command Line Interface (CLI) (beginner user) and Groovy (intermediate and advanced user)

Several tutorials are **under construction** for Windows 10. Please select a tutorial in the table below.

Actions	OpenStreetMap	BDTopo 2.2 version
Default case : Calculate LCZ, TEB inputs and UTRF	CLI / Groovy	CLI / Groovy

Default case with OSM

This tutorial presents how to create Local Climate Zones with OpenStreetMap data.

Two tools are available to run GeoClimate algorithms: Command Line Interface (beginner user) and Groovy (intermediate and advanced user)

Command Line Interface

Get Geoclimate.jar on your computer

You will run the archive Geoclimate.jar in a Command Line Interface.

First, make sure Java (version 8 minimum) is installed in your computer.

You need to download Geoclimate.jar here.

Rename the downloaded file as "Geoclimate.jar".

Create a folder in your documents (for instance C:\mydirectory\Geoclimate) and place Geoclimate.jar in this folder.

Create and understand the configuration file

In order to run Geoclimate, you need to write a configuration file. This file specifies inputs, methods and outputs of Geoclimate.

An example of configuration file is presented below :

```
{
    "description": "Processing OSM data",
    "input": {
        "osm": [
            "Pont-de-Veyle"
        ]
    },
    "output": {
        "folder": "C:\\temp"
    },
    "parameters": {
        "rsu_indicators": {
            "indicatorUse": [
                "LCZ",
                "TEB",
                "UTRF"
            ],
            "svfSimplified": true,
            "estimateHeight": true
        },
        "grid_indicators": {
            "x_size": 100,
        "y_size": 100,
        "rowCol": false,
        "output" : "geojson",
        "indicators" :[
                 "BUILDING_FRACTION",
                 "BUILDING_HEIGHT",
                 "WATER_FRACTION",
                 "VEGETATION_FRACTION",
                 "ROAD_FRACTION",
                 "IMPERVIOUS_FRACTION",
                 "LCZ_FRACTION"
             ]
         }
    }
}
```

You can copy this example in a notebook and name it "my_first_config_file_osm.json". Place this configuration file in the same folder than Geoclimate.jar .

Understand the configuration file

The configuration file is structured in four main parts.

- "description" is a text that describes your process. You can name your process here.
- "input" specifies the input data you will use. In this example, we specify "osm" for OpenStreetMap, and we run Geoclimate for a small village

in France called Pont-de-Veyle.

- "output" specifies the format you expect for your output (here "folder") and where you want to create your output files (here in C:\temp).
- "parameters" specifies the output you want to calculate based on your reference spatial units ("rsu_indicators") or on a grid ("grid_indicators").
- At RSU scale, we calculate the LCZ, the TEB inputs and the UTRF ("indicatorUse": ["LCZ", "TEB", "UTRF"]). We use the simplified method to calculate the sky view factor ("svfSimplified": true) and the method to estimate the height of buildings in OSM ("estimateHeight" : true).
- With the grid approach, we specify the grid dimensions in meters ("x_size" and "y_size") and the output format ("output" : "geojson"). Then, we specify the indicators we want to calculate for each cell of the grid ("BUILDING_FRACTION", "BUILDING_HEIGHT", "WATER_FRACTION", "VEGETATION_FRACTION", "ROAD_FRACTION", "IMPERVIOUS_FRACTION", "LCZ_FRACTION").

Run Geoclimate

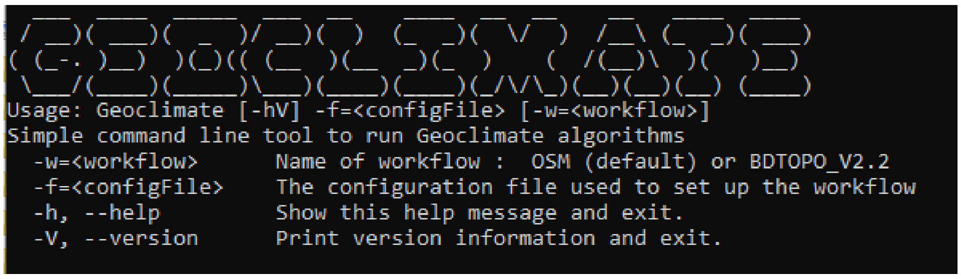
On your machine, open a command prompt.

Go to the folder where Geoclimate.jar is located using this command line :

`cd mydirectory\Geoclimate`

Then, you can run this command line which presents you the main options of Geoclimate :

`java -jar Geoclimate.jar -h`



```
  ___  ____  _____  ___  __    ____  __  __    __   ____  ____
 / __)( ___)(  _  )/ __)(  )  (_  _)(  \/  )  /__\ (_  _)( ___)
( (_-. )__)  )(_)(( (__  )(__  _)(_  )    (  /(__)\  )(   )__)
 \___/(____)(_____)\___)(____)(____)(_/\/\_)(__)(__)(__) (____)
Usage: Geoclimate [-hV] -f=<configFile> [-w=<workflow>]
Simple command line tool to run Geoclimate algorithms
  -w=<workflow>      Name of workflow :  OSM (default) or BDTOPO_V2.2
  -f=<configFile>    The configuration file used to set up the workflow
  -h, --help         Show this help message and exit.
  -V, --version      Print version information and exit.
```

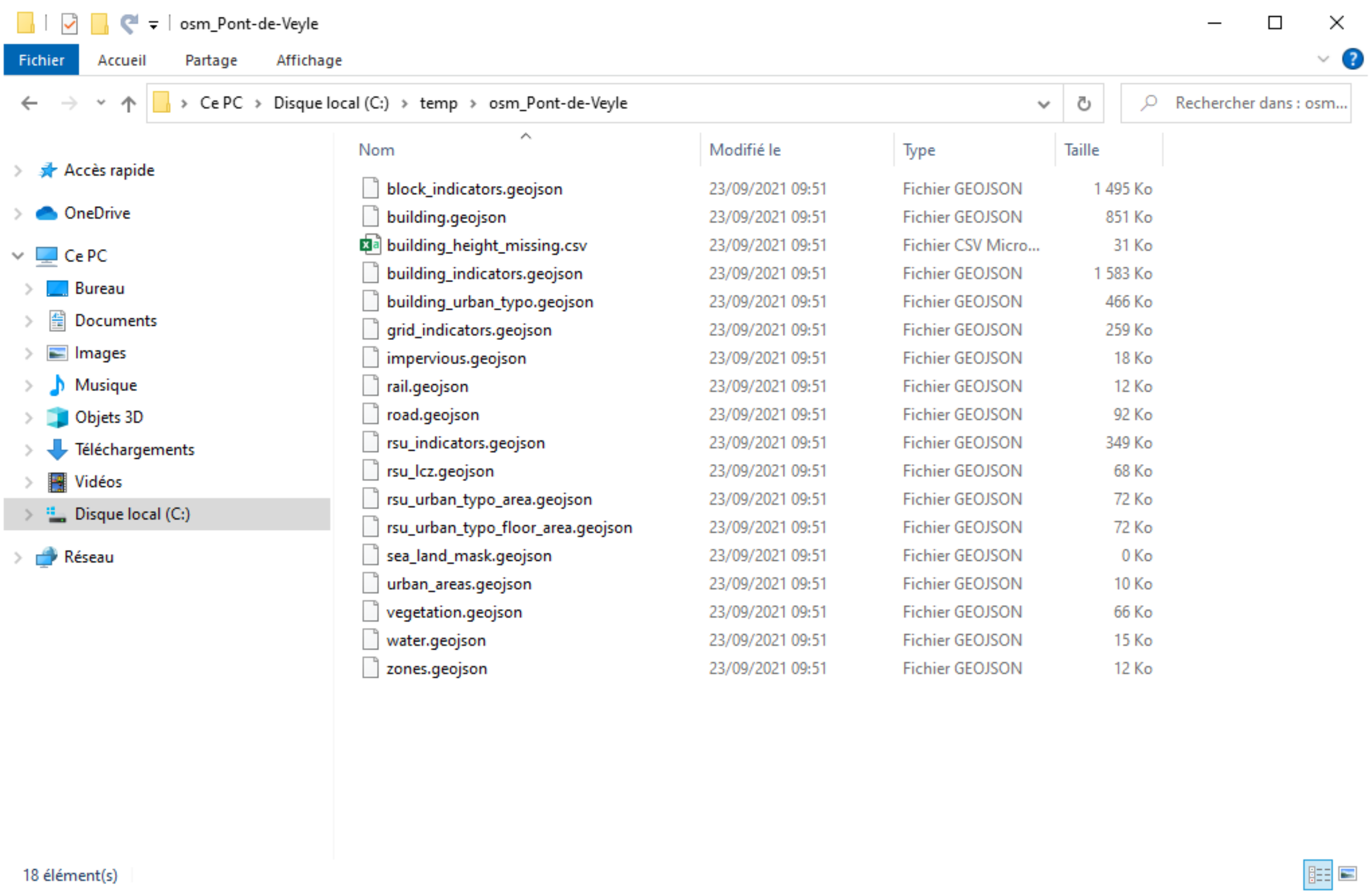
In order to perform your first calculations with the configuration file above, use

`java -jar Geoclimate.jar -f my_first_config_file_osm.json -w OSM`

where the f option is used to set the path of the configuration file.

If everything runs well, you will obtain a message : `The OSM workflow has been successfully executed`

The results of your calculations are located in you `C:\temp` folder.


osm_Pont-de-Veyle

Ce PC > Disque local (C:) > temp > osm_Pont-de-Veyle

Nom	Modifié le	Type	Taille
block_indicators.geojson	23/09/2021 09:51	Fichier GEOJSON	1 495 Ko
building.geojson	23/09/2021 09:51	Fichier GEOJSON	851 Ko
building_height_missing.csv	23/09/2021 09:51	Fichier CSV Micro...	31 Ko
building_indicators.geojson	23/09/2021 09:51	Fichier GEOJSON	1 583 Ko
building_urban_typo.geojson	23/09/2021 09:51	Fichier GEOJSON	466 Ko
grid_indicators.geojson	23/09/2021 09:51	Fichier GEOJSON	259 Ko
impervious.geojson	23/09/2021 09:51	Fichier GEOJSON	18 Ko
rail.geojson	23/09/2021 09:51	Fichier GEOJSON	12 Ko
road.geojson	23/09/2021 09:51	Fichier GEOJSON	92 Ko
rsu_indicators.geojson	23/09/2021 09:51	Fichier GEOJSON	349 Ko
rsu_lcz.geojson	23/09/2021 09:51	Fichier GEOJSON	68 Ko
rsu_urban_typo_area.geojson	23/09/2021 09:51	Fichier GEOJSON	72 Ko
rsu_urban_typo_floor_area.geojson	23/09/2021 09:51	Fichier GEOJSON	72 Ko
sea_land_mask.geojson	23/09/2021 09:51	Fichier GEOJSON	0 Ko
urban_areas.geojson	23/09/2021 09:51	Fichier GEOJSON	10 Ko
vegetation.geojson	23/09/2021 09:51	Fichier GEOJSON	66 Ko
water.geojson	23/09/2021 09:51	Fichier GEOJSON	15 Ko
zones.geojson	23/09/2021 09:51	Fichier GEOJSON	12 Ko

18 élément(s)

Groovy console

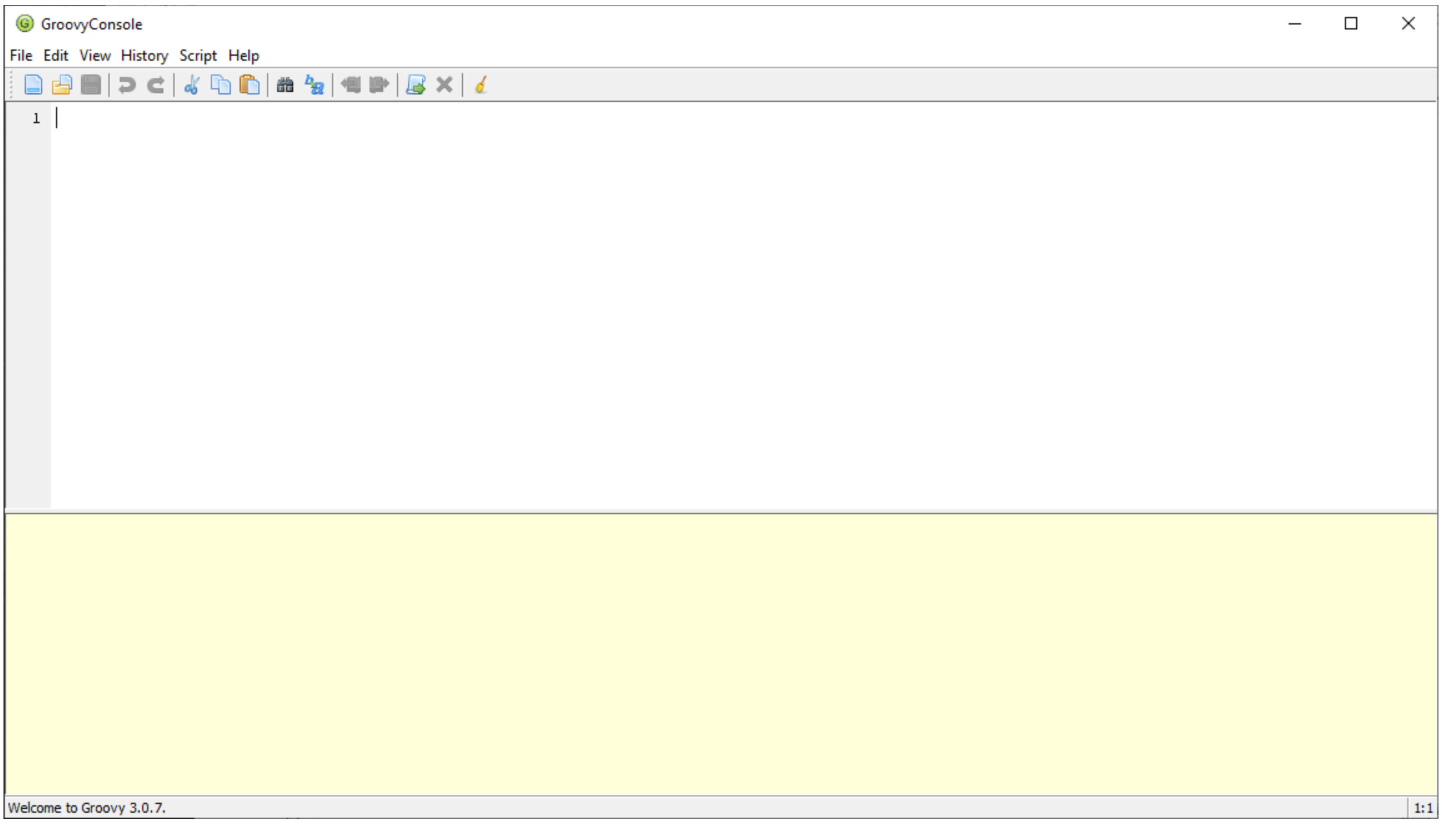
Get Groovy on your computer

First, make sure that Java Development Kit (JDK) and Java Runtime Environment (JRE) are installed in your computer.

You need to install Groovy 3.0.7, using for instance this link.

Run Geoclimate

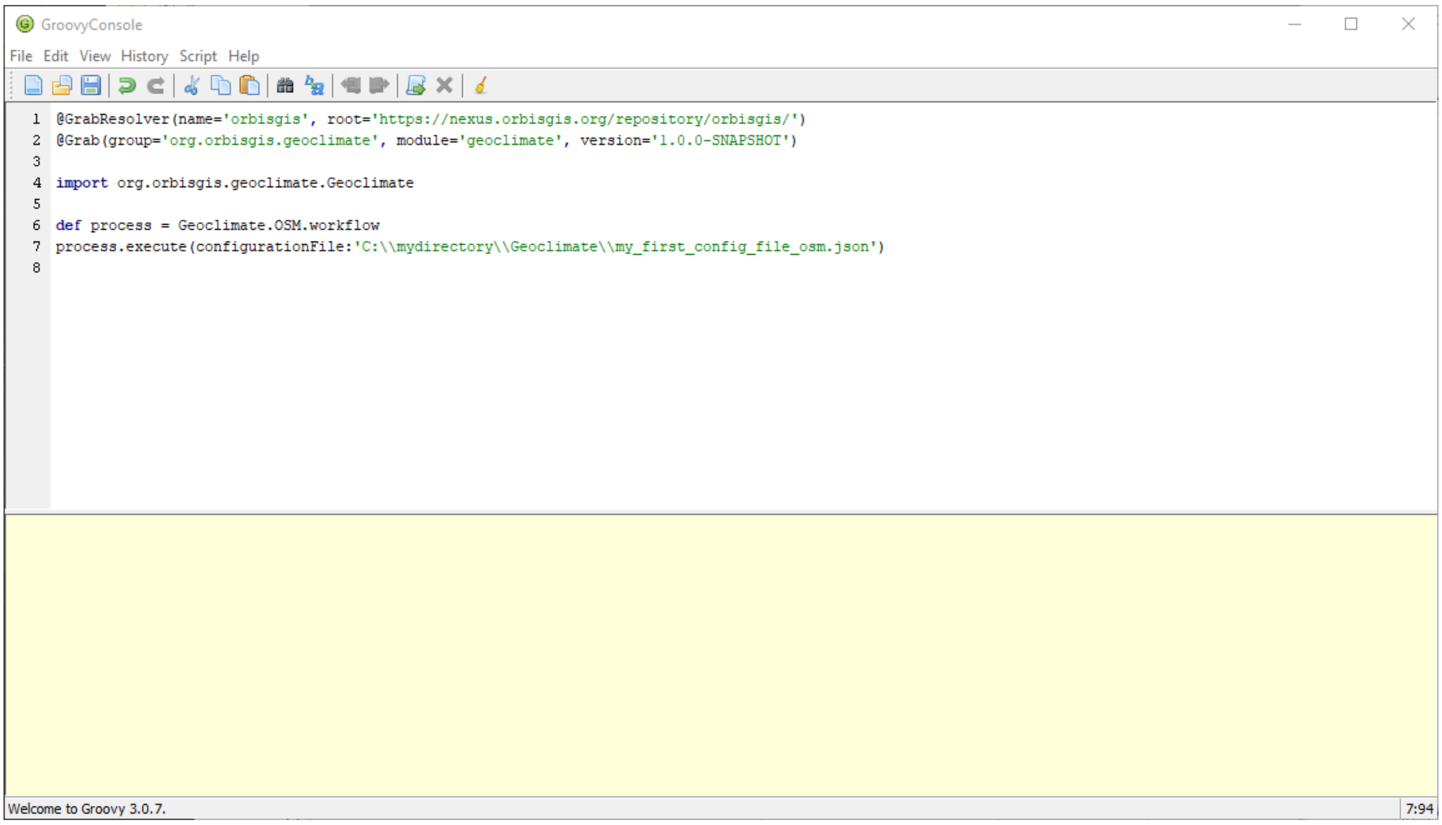
When Groovy is installed, open the Groovy console using the research tool of windows.



In the Groovy console, you can copy/paste the following script :

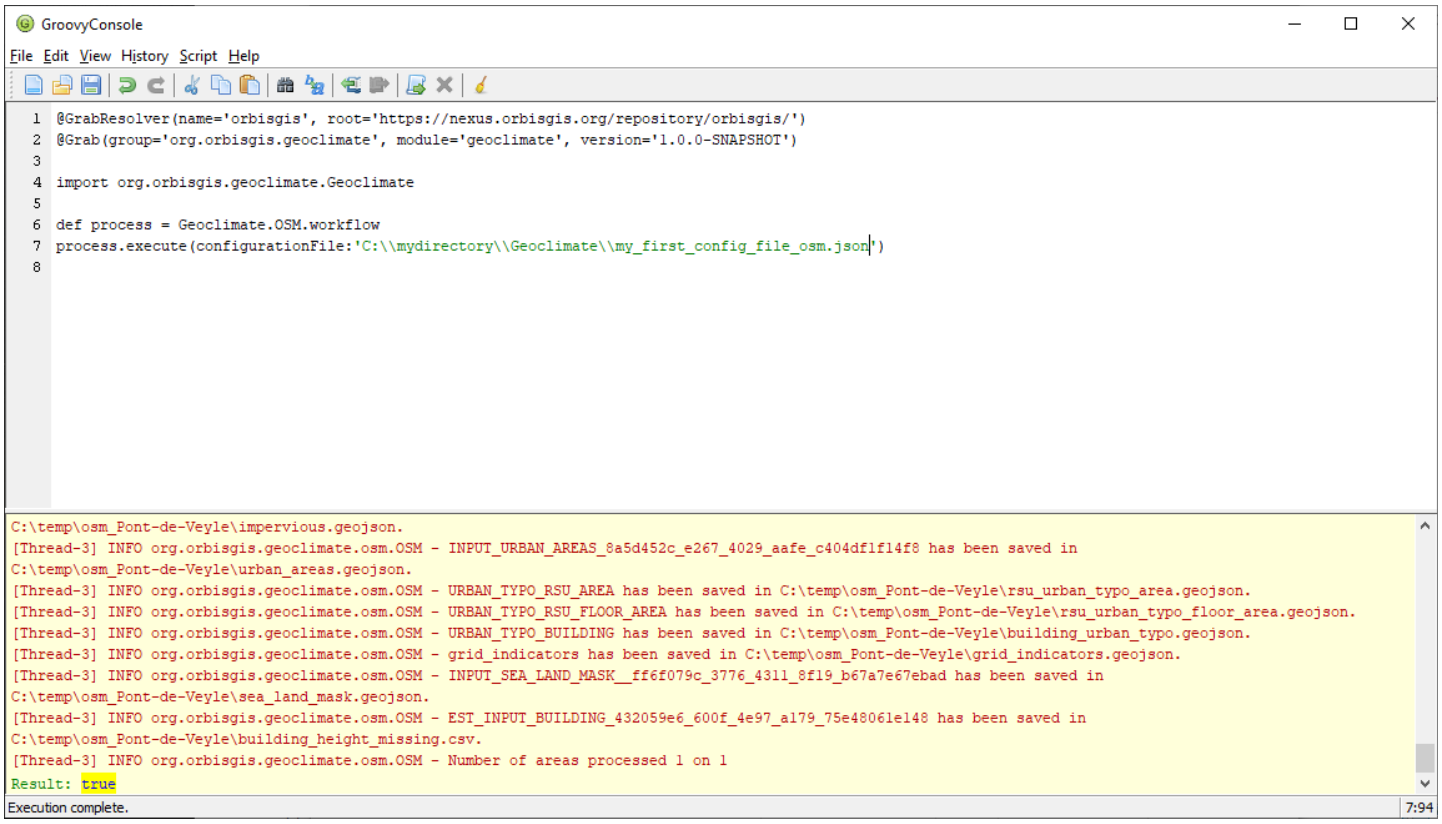
```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.geoclimate', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.geoclimate.Geoclimate

def process = Geoclimate.OSM.workflow
process.execute(configurationFile:'C:\\mydirectory\\Geoclimate\\my_first_config_file_osm.json')
```

This script will run Geoclimate using the configuration file `my_first_config_file_osm.json`. Please see here for more explanations.



```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.geoclimate', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.geoclimate.Geoclimate

def process = Geoclimate.OSM.workflow
process.execute(configurationFile:'C:\\mydirectory\\Geoclimate\\my_first_config_file_osm.json')

```

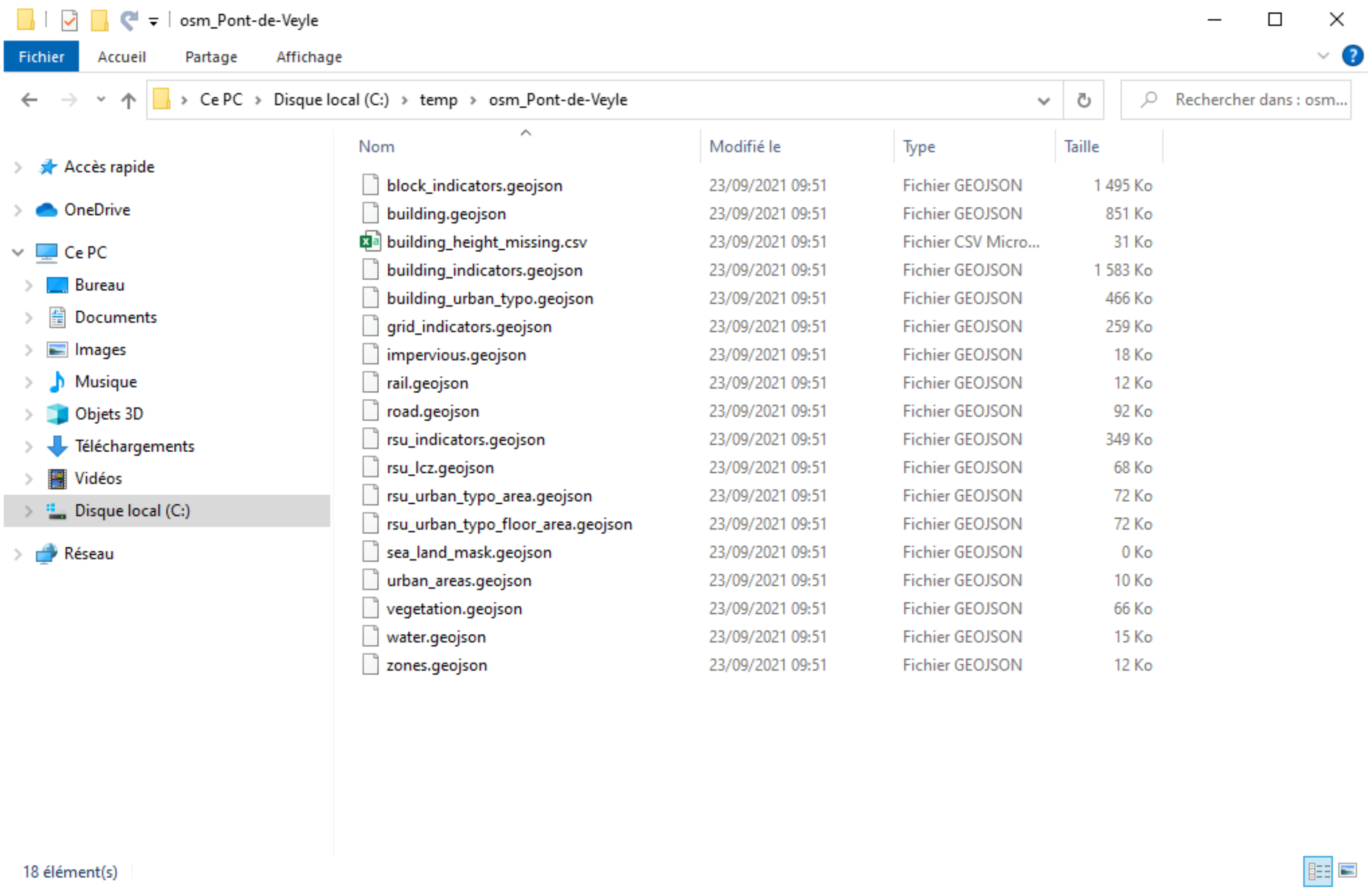
You can run your script by using the shortcut `Ctrl + R` or clicking on .



```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.geoclimate', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.geoclimate.Geoclimate

def process = Geoclimate.OSM.workflow
process.execute(configurationFile:'C:\\mydirectory\\Geoclimate\\my_first_config_file_osm.json')

```

```
C:\temp\osm_Pont-de-Veyle\impervious.geojson.
[Thread-3] INFO org.orbisgis.geoclimate.osm.OSM - INPUT_URBAN_AREAS_8a5d452c_e267_4029_aafe_c404df1f14f8 has been saved in
C:\temp\osm_Pont-de-Veyle\urban_areas.geojson.
[Thread-3] INFO org.orbisgis.geoclimate.osm.OSM - URBAN_TYPO_RSU_AREA has been saved in C:\temp\osm_Pont-de-Veyle\rsu_urban_typo_area.geojson.
[Thread-3] INFO org.orbisgis.geoclimate.osm.OSM - URBAN_TYPO_RSU_FLOOR_AREA has been saved in C:\temp\osm_Pont-de-Veyle\rsu_urban_typo_floor_area.geojson.
[Thread-3] INFO org.orbisgis.geoclimate.osm.OSM - URBAN_TYPO_BUILDING has been saved in C:\temp\osm_Pont-de-Veyle\building_urban_typo.geojson.
[Thread-3] INFO org.orbisgis.geoclimate.osm.OSM - grid_indicators has been saved in C:\temp\osm_Pont-de-Veyle\grid_indicators.geojson.
[Thread-3] INFO org.orbisgis.geoclimate.osm.OSM - INPUT_SEA_LAND_MASK__ff6f079c_3776_4311_8f19_b67a7e67ebad has been saved in
C:\temp\osm_Pont-de-Veyle\sea_land_mask.geojson.
[Thread-3] INFO org.orbisgis.geoclimate.osm.OSM - EST_INPUT_BUILDING_432059e6_600f_4e97_a179_75e48061e148 has been saved in
C:\temp\osm_Pont-de-Veyle\building_height_missing.csv.
[Thread-3] INFO org.orbisgis.geoclimate.osm.OSM - Number of areas processed 1 on 1
Result: true
```

The results of your calculations are located in you `C:\temp` folder.



Nom	Modifié le	Type	Taille
block_indicators.geojson	23/09/2021 09:51	Fichier GEOJSON	1 495 Ko
building.geojson	23/09/2021 09:51	Fichier GEOJSON	851 Ko
building_height_missing.csv	23/09/2021 09:51	Fichier CSV Micro...	31 Ko
building_indicators.geojson	23/09/2021 09:51	Fichier GEOJSON	1 583 Ko
building_urban_typo.geojson	23/09/2021 09:51	Fichier GEOJSON	466 Ko
grid_indicators.geojson	23/09/2021 09:51	Fichier GEOJSON	259 Ko
impervious.geojson	23/09/2021 09:51	Fichier GEOJSON	18 Ko
rail.geojson	23/09/2021 09:51	Fichier GEOJSON	12 Ko
road.geojson	23/09/2021 09:51	Fichier GEOJSON	92 Ko
rsu_indicators.geojson	23/09/2021 09:51	Fichier GEOJSON	349 Ko
rsu_lcz.geojson	23/09/2021 09:51	Fichier GEOJSON	68 Ko
rsu_urban_typo_area.geojson	23/09/2021 09:51	Fichier GEOJSON	72 Ko
rsu_urban_typo_floor_area.geojson	23/09/2021 09:51	Fichier GEOJSON	72 Ko
sea_land_mask.geojson	23/09/2021 09:51	Fichier GEOJSON	0 Ko
urban_areas.geojson	23/09/2021 09:51	Fichier GEOJSON	10 Ko
vegetation.geojson	23/09/2021 09:51	Fichier GEOJSON	66 Ko
water.geojson	23/09/2021 09:51	Fichier GEOJSON	15 Ko
zones.geojson	23/09/2021 09:51	Fichier GEOJSON	12 Ko

Default case with BDTopo 2.2

This tutorial presents how to create Local Climate Zones with BD Topo 2.2 version.

Two tools are available : Command Line Interface (beginner user) and Groovy (intermediate and advanced user)

Command Line Interface

Get Geoclimate.jar on your computer

You will run the archive Geoclimate.jar in a Command Line Interface.

First, make sure Java (version 8 minimum) is installed in your computer.

You need to download Geoclimate.jar here.

Rename the downloaded file as "Geoclimate.jar".

Create a folder in your documents (for instance C:\mydirectory\Geoclimate) and place Geoclimate.jar in this folder.

Get data from BD Topo 2.2 on your computer

Collect all your BD Topo 2.2 data in one single folder. The required folder are listed here.

Create a subfolder (for instance C:\mydirectory\Geoclimate\BD_TOPO_v2) and place your BD Topo 2.2 data in this folder.

Create and understand a configuration file

In order to run Geoclimate, you need to write a configuration file. This file specifies inputs, methods and outputs of Geoclimate.

An example of configuration file is presented below :

```
{
    "description": "Processing BD Topo v2 data",
    "input": {"bdtopo_v2":    {
        "folder": {"path": "C:\\mydirectory\\Geoclimate\\BD_TOPO_v2"}
                }
    },
    "output": {
        "folder": "C:\\temp"
    },
    "parameters": {
        "rsu_indicators": {
            "indicatorUse": [
                "LCZ",
                "TEB",
                "URBAN_TYPOLOGY"
            ],
            "svfSimplified": true
        },
        "grid_indicators": {
            "x_size": 100,
        "y_size": 100,
        "rowCol": false,
        "output" : "geojson",
        "indicators" :[
                 "BUILDING_FRACTION",
                 "BUILDING_HEIGHT",
                 "WATER_FRACTION",
                 "VEGETATION_FRACTION",
                 "ROAD_FRACTION",
                 "IMPERVIOUS_FRACTION",
                 "LCZ_FRACTION"
             ]
         }
    }
}
```

You can copy this example in a notebook and name it "my_first_config_file_bdtopov2.json" . Place this configuration file in the same folder than Geoclimate.jar .

Understand the configuration file

The configuration file is structured in four main parts.

- "description" is a character string that describes your process. You can name your process here.
- "input" specifies the input data you will use. In this example, we specify "folder" for BD Topo 2.2 version, and we specify where the BD Topo 2.2 files are located on the computer ("C:\mydirectory\Geoclimate\BD_TOPO_v2").
- "output" specifies the format you expect for your output (here "folder") and where you want to create your output files (here in C:\temp).
- "parameters" specifies the output you want to calculate based on your reference spatial units ("rsu_indicators") or on a grid ("grid_indicators").
- At RSU scale, we calculate the LCZ, the TEB inputs and the UTRF ("indicatorUse": ["LCZ", "TEB", "URBAN_TYPOLOGY"]). We use the simplified method to calculate the sky view factor ("svfSimplified": true).
- With the grid approach, we specify the grid dimensions in meters ("x_size" and "y_size") and the output format ("output" : "geojson"). Then, we specify the indicators we want to calculate for each cell of the grid ("BUILDING_FRACTION", "BUILDING_HEIGHT", "WATER_FRACTION", "VEGETATION_FRACTION", "ROAD_FRACTION", "IMPERVIOUS_FRACTION", "LCZ_FRACTION").

Run Geoclimate

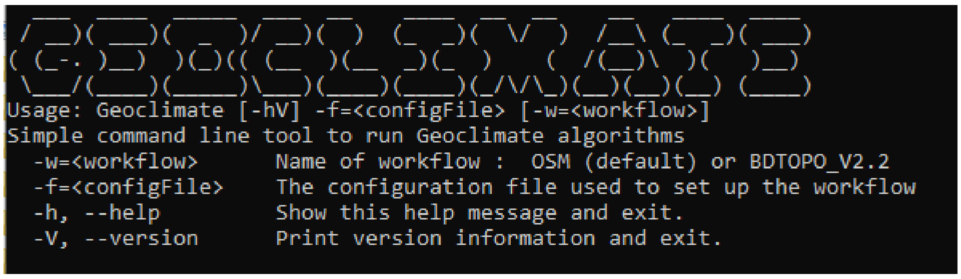
On your machine, open a command prompt.

Go to the folder where Geoclimate.jar is located using this command line :

```
cd mydirectory\Geoclimate
```

Then, you can run this command line which presents you the main options of Geoclimate :

```
java -jar Geoclimate.jar -h
```



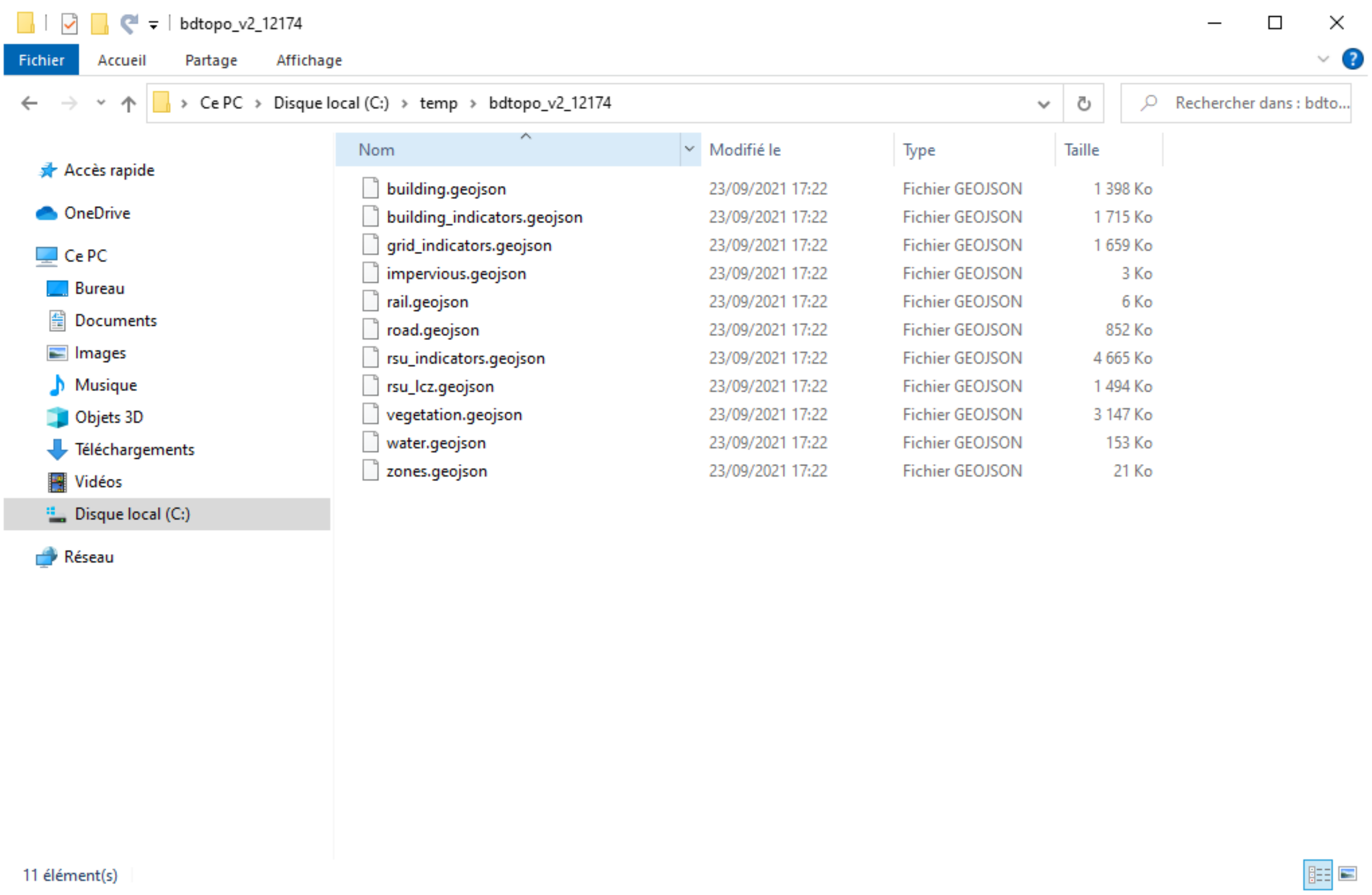
```
  ___  ____  _____  ___  __    ____  __  __    __   ____  ____
 / __)( ___)(  _  )/ __)(  )  (_  _)(  \/  )  /__\ (_  _)( ___)
( (_-. )__)  )(_)(( (__  )(__  _)(_  )    (  /(__)\  )(   )__)
 \___/(____)(_____)\___)(____)(____)(_/\/\_)(__)(__)(__) (____)
Usage: Geoclimate [-hV] -f=<configFile> [-w=<workflow>]
Simple command line tool to run Geoclimate algorithms
  -w=<workflow>      Name of workflow :  OSM (default) or BDTOPO_V2.2
  -f=<configFile>    The configuration file used to set up the workflow
  -h, --help         Show this help message and exit.
  -V, --version      Print version information and exit.
```

In order to perform your first calculations with the configuration file above, use

```
java -jar Geoclimate.jar -f my_first_config_file_bdtopov2.json -w BDTOPO_V2.2
```

If everything runs well, you will obtain a message : `The BDTOPO_V2.2 workflow has been successfully executed`

The results of your calculations are located in you `C:\temp` folder.


bdtopo_v2_12174

Fichier Accueil Partage Affichage

Ce PC > Disque local (C:) > temp > bdtopo_v2_12174

Rechercher dans : bdto...

Nom	Modifié le	Type	Taille
building.geojson	23/09/2021 17:22	Fichier GEOJSON	1 398 Ko
building_indicators.geojson	23/09/2021 17:22	Fichier GEOJSON	1 715 Ko
grid_indicators.geojson	23/09/2021 17:22	Fichier GEOJSON	1 659 Ko
impervious.geojson	23/09/2021 17:22	Fichier GEOJSON	3 Ko
rail.geojson	23/09/2021 17:22	Fichier GEOJSON	6 Ko
road.geojson	23/09/2021 17:22	Fichier GEOJSON	852 Ko
rsu_indicators.geojson	23/09/2021 17:22	Fichier GEOJSON	4 665 Ko
rsu_lcz.geojson	23/09/2021 17:22	Fichier GEOJSON	1 494 Ko
vegetation.geojson	23/09/2021 17:22	Fichier GEOJSON	3 147 Ko
water.geojson	23/09/2021 17:22	Fichier GEOJSON	153 Ko
zones.geojson	23/09/2021 17:22	Fichier GEOJSON	21 Ko

11 élément(s)

Groovy console

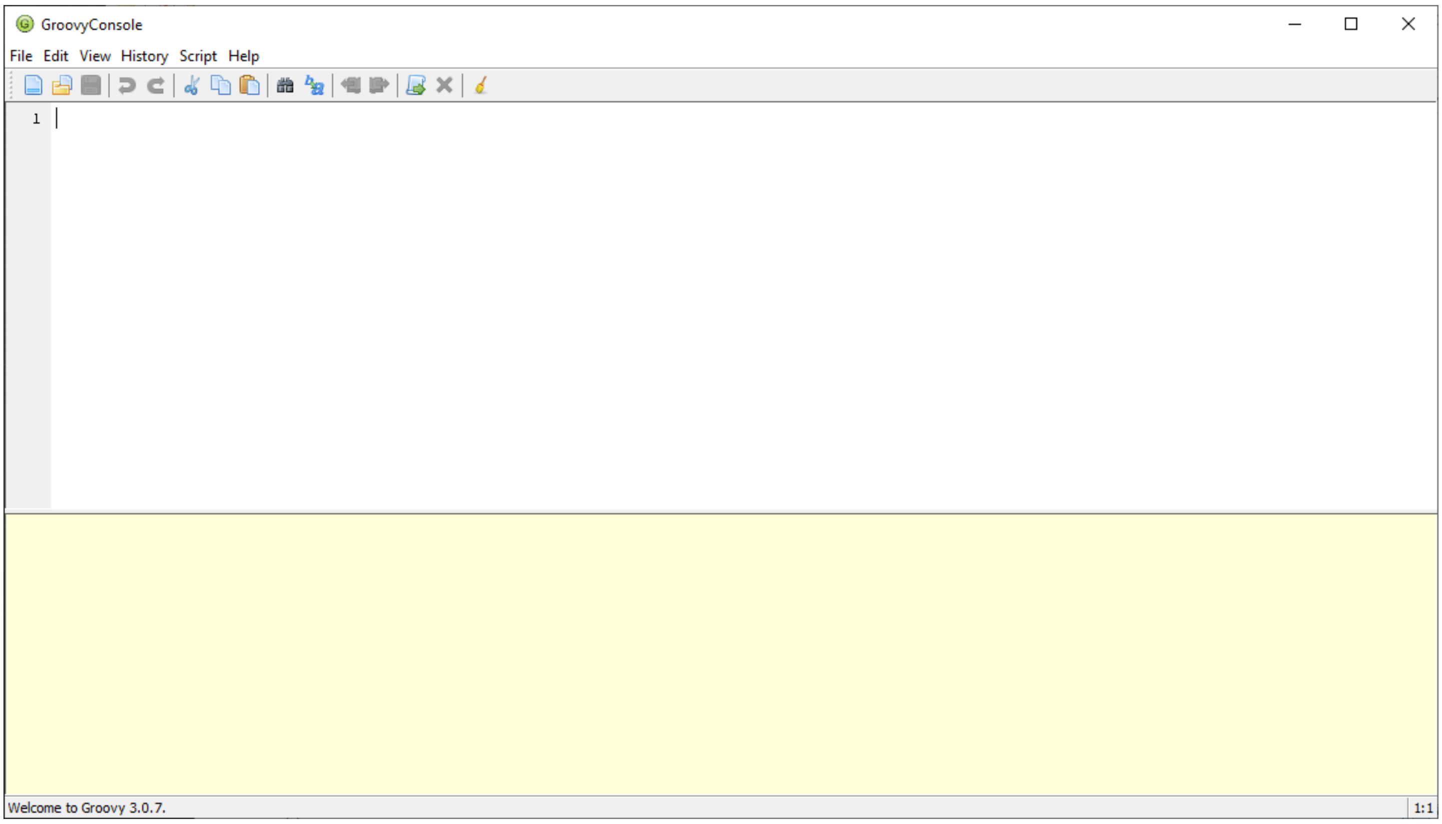
Get Groovy on your computer

First, make sure that Java Development Kit (JDK) and Java Runtime Environment (JRE) are installed in your computer.

You need to install Groovy 3.0.7, using for instance this link.

Run Geoclimate

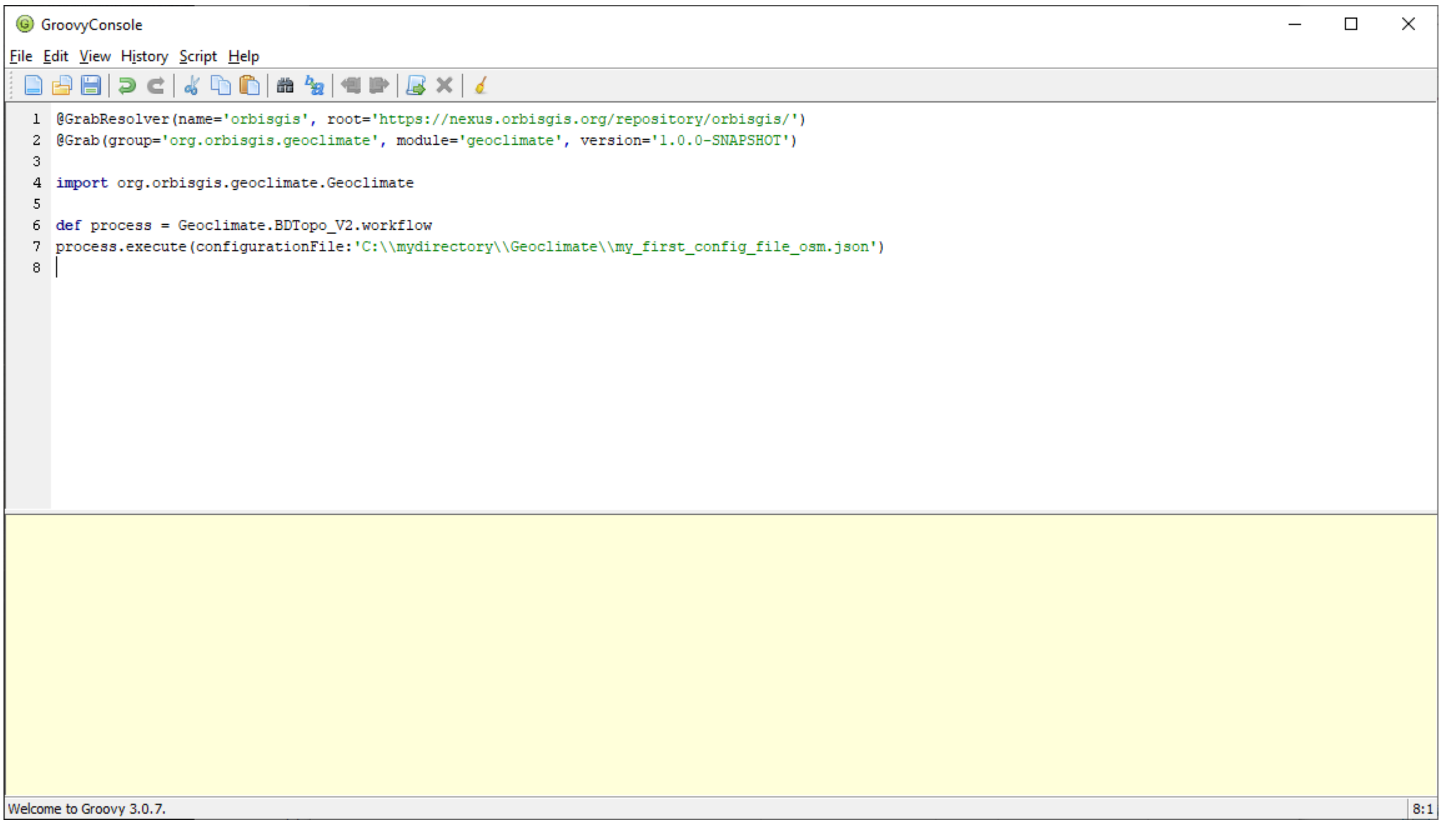
When Groovy is installed, open the Groovy console using the research tool of windows :



In the Groovy console, you can copy/paste the following script :

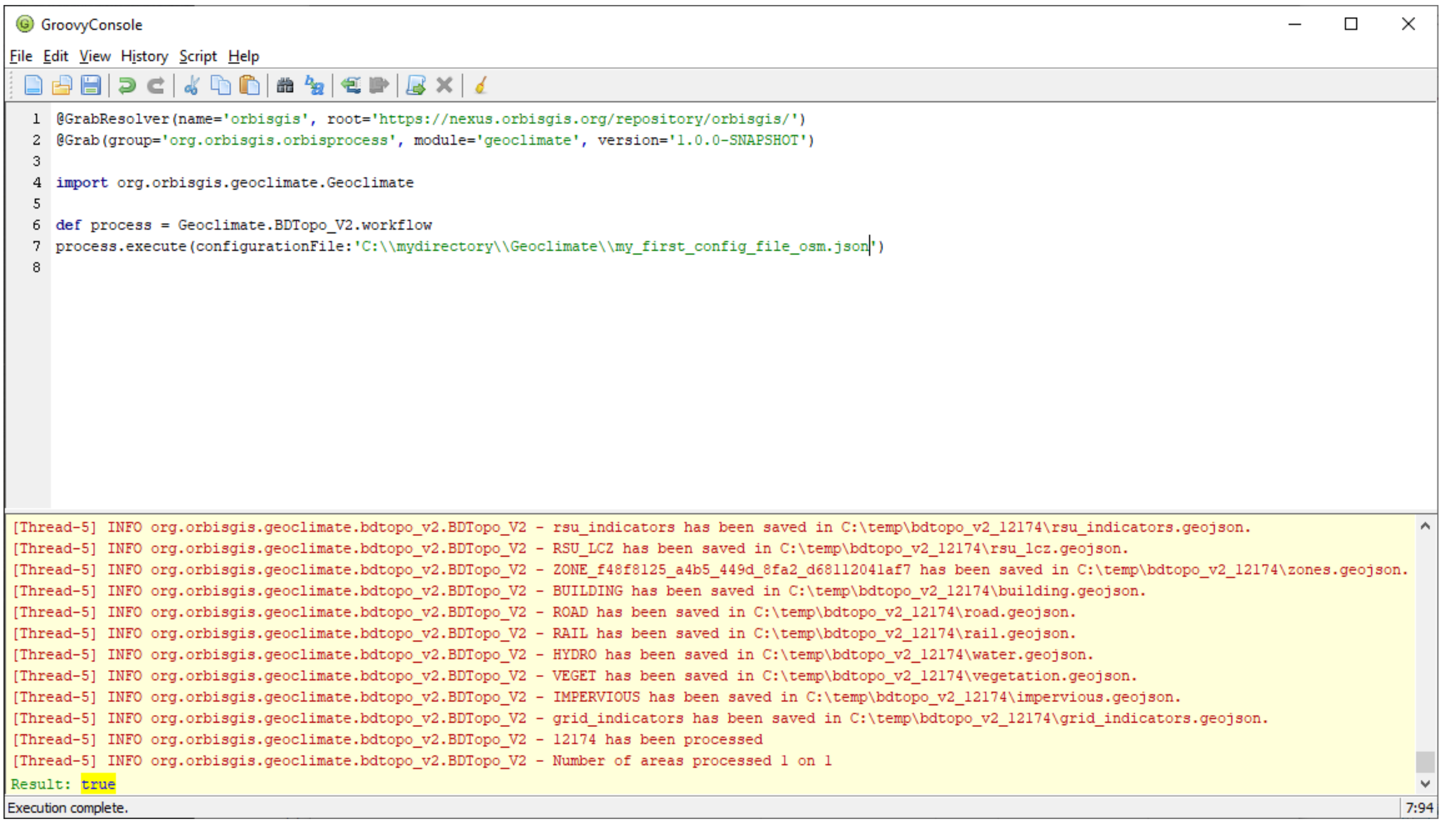
```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.orbisprocess', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.geoclimate.Geoclimate

def process = Geoclimate.BDTopo_V2.workflow
process.execute(configurationFile:'C:\\mydirectory\\Geoclimate\\my_first_config_file_bdtopov2.json')
```

This script will run Geoclimate using the configuration file `my_first_config_file_bdtopov2.json`. Please see here for more explanations.



```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.geoclimate', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.geoclimate.Geoclimate

def process = Geoclimate.BDTopo_V2.workflow
process.execute(configurationFile:'C:\\mydirectory\\Geoclimate\\my_first_config_file_osm.json')
```

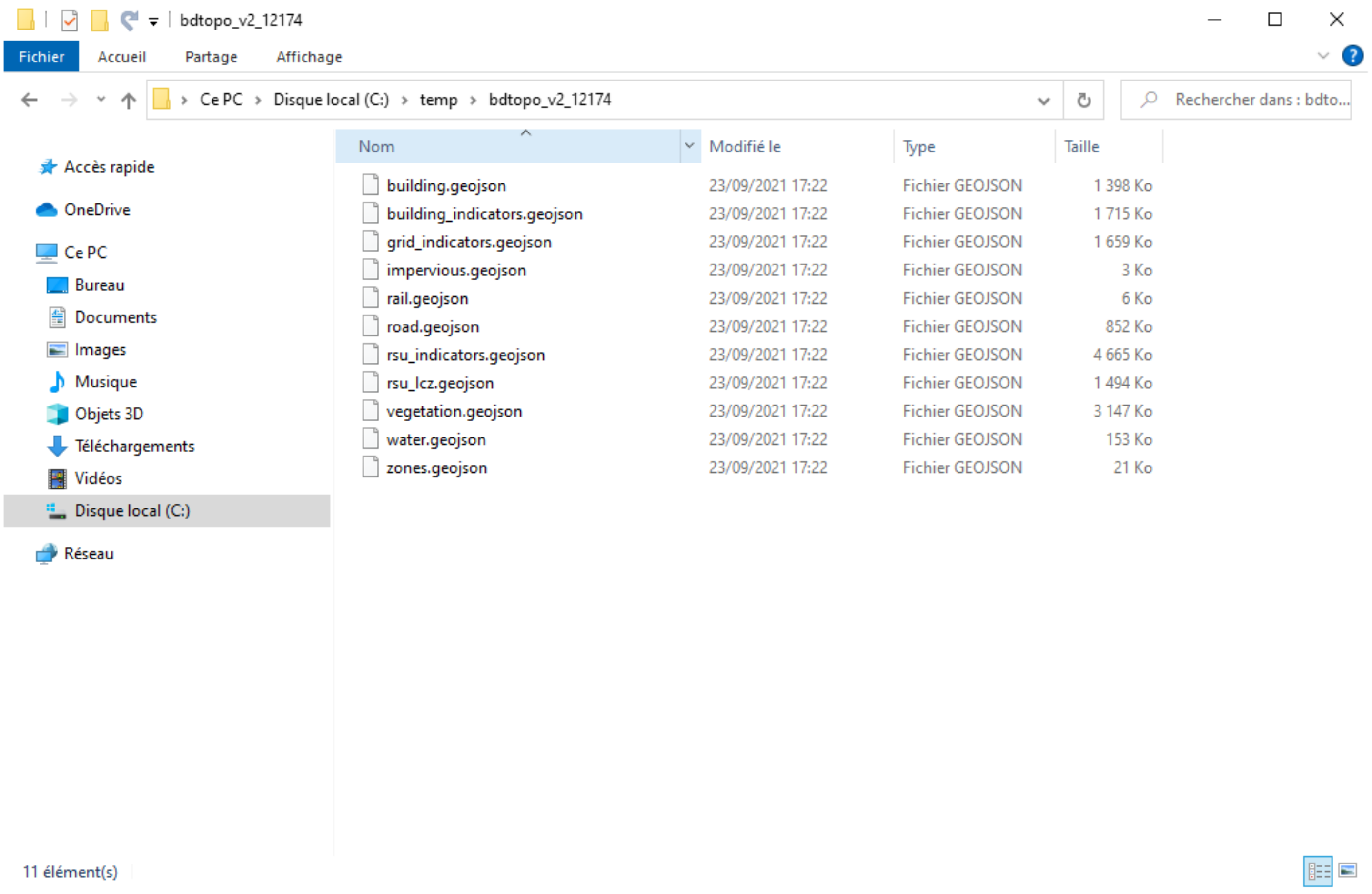
You can run your script by using the shortcut `Ctrl + R` or clicking on .



```
@GrabResolver(name='orbisgis', root='https://nexus.orbisgis.org/repository/orbisgis/')
@Grab(group='org.orbisgis.orbisprocess', module='geoclimate', version='1.0.0-SNAPSHOT')

import org.orbisgis.geoclimate.Geoclimate

def process = Geoclimate.BDTopo_V2.workflow
process.execute(configurationFile:'C:\\mydirectory\\Geoclimate\\my_first_config_file_osm.json')
```

```
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - rsu_indicators has been saved in C:\temp\bdtopo_v2_12174\rsu_indicators.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - RSU_LCZ has been saved in C:\temp\bdtopo_v2_12174\rsu_lcz.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - ZONE_f48f8125_a4b5_449d_8fa2_d68112041af7 has been saved in C:\temp\bdtopo_v2_12174\zones.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - BUILDING has been saved in C:\temp\bdtopo_v2_12174\building.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - ROAD has been saved in C:\temp\bdtopo_v2_12174\road.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - RAIL has been saved in C:\temp\bdtopo_v2_12174\rail.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - HYDRO has been saved in C:\temp\bdtopo_v2_12174\water.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - VEGET has been saved in C:\temp\bdtopo_v2_12174\vegetation.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - IMPERVIOUS has been saved in C:\temp\bdtopo_v2_12174\impervious.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - grid_indicators has been saved in C:\temp\bdtopo_v2_12174\grid_indicators.geojson.
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - 12174 has been processed
[Thread-5] INFO org.orbisgis.geoclimate.bdtopo_v2.BDTopo_V2 - Number of areas processed 1 on 1
Result: true
Execution complete.                                                                                  7:94
```

The results of your calculations are located in you `C:\temp` folder.



Nom	Modifié le	Type	Taille
building.geojson	23/09/2021 17:22	Fichier GEOJSON	1 398 Ko
building_indicators.geojson	23/09/2021 17:22	Fichier GEOJSON	1 715 Ko
grid_indicators.geojson	23/09/2021 17:22	Fichier GEOJSON	1 659 Ko
impervious.geojson	23/09/2021 17:22	Fichier GEOJSON	3 Ko
rail.geojson	23/09/2021 17:22	Fichier GEOJSON	6 Ko
road.geojson	23/09/2021 17:22	Fichier GEOJSON	852 Ko
rsu_indicators.geojson	23/09/2021 17:22	Fichier GEOJSON	4 665 Ko
rsu_lcz.geojson	23/09/2021 17:22	Fichier GEOJSON	1 494 Ko
vegetation.geojson	23/09/2021 17:22	Fichier GEOJSON	3 147 Ko
water.geojson	23/09/2021 17:22	Fichier GEOJSON	153 Ko
zones.geojson	23/09/2021 17:22	Fichier GEOJSON	21 Ko

Bounding box case

The default cases for OSM and BDTOPO v2.2 are presented in other pages. Please make sure that the default cases are running on your computer before using the bounding box method.

In the default case, the area of interest is specified using a city name. You also have the possibility to calculate all the GeoClimate outputs inside a given rectangle (i.e. a bounding box).

The only difference with the default case is that geographic coordinates replace the city name. All the other elements of the configuration file remain unchanged.

The coordinates of the bounding box are expressed as [minY, minX, maxY, maxX].

Bounding box method with OSM

A bounding box has been determined for the city of Pont-de-Veyle, with the following coordinates : 46.257330,4.870033,46.269970,4.905224

The configuration file below uses this bounding box method with OSM.

```
{
    "description": "Processing OSM data",
    "input": {
        "osm": [
            [46.257330,4.870033,46.269970,4.905224]
        ]
    },
    "output": {
        "folder": "C:\\temp"
    },
    "parameters": {
        "rsu_indicators": {
            "indicatorUse": [
                "LCZ",
                "TEB",
                "UTRF"
            ],
            "svfSimplified": true,
            "estimateHeight": true
        },
        "grid_indicators": {
            "x_size": 100,
        "y_size": 100,
        "rowCol": false,
        "output" : "geojson",
        "indicators" :[
                 "BUILDING_FRACTION",
                 "BUILDING_HEIGHT",
                 "WATER_FRACTION",
                 "VEGETATION_FRACTION",
                 "ROAD_FRACTION",
                 "IMPERVIOUS_FRACTION",
                 "LCZ_FRACTION"
             ]
         }
    }
}
```

In order to determine the coordinates of your bounding box, you can use the bboxfinder website. Make sure your coordinates are in latitude / longitude ("Lat / Lon") format. You can choose the coordinates options on the bottom right of the bboxfinder website.

Bounding box method with BDTOPO v2.2

The configuration file below uses a bounding box method with BDTOPO v2.2.

This bounding box represents the envelope of the city of Redon.

Note that the EPSG code for the projection system here is 2154 and not "Lat / Lon" anymore.

```
{
    "description": "Processing BDTopo v 2.2 data",
    "input": {
        "bdtopo_v2": {
            "folder": {
                "path": "C:\\home\\mydirectory\\Geoclimate\\BD_TOPO_v2\\",
                "id_zones": [
                    [
                        6737756.724564202,
                        316124.01010211144,
                        6743486.0484706545,
                        321921.09550058335
                    ]
                ]
            }
        }
    },
    "output": {
        "folder": "C:\\temp"
    },
    "parameters": {
        "rsu_indicators": {
            "indicatorUse": [
                "LCZ",
                "TEB",
                "UTRF"
            ],
            "svfSimplified": true
        },
        "grid_indicators": {
            "x_size": 100,
        "y_size": 100,
        "rowCol": false,
        "output" : "geojson",
        "indicators" :[
                 "BUILDING_FRACTION",
                 "BUILDING_HEIGHT",
                 "WATER_FRACTION",
                 "VEGETATION_FRACTION",
                 "ROAD_FRACTION",
                 "IMPERVIOUS_FRACTION",
                 "LCZ_FRACTION"
             ]
         }
    }
}
```

Frequently Asked Questions

GeoClimate reports "java.lang.OutOfMemoryError: Java heap space"

As the error message suggests, you have run out of memory on your GeoClimate instance. Increase it with the following arguments :

```
java -Xmx1024m -jar geoclimate.jar -f myconfigFile.json
```

See https://docs.oracle.com/cd/E15523_01/web.1111/e13814/jvm_tuning.htm#PERFM160

How to use GeoClimate with proxy configuration

Depending on networking environments, particularly corporate ones, you must have to deal with proxy configuration.

If you run Geoclimate with a Groovy script, tune the proxy just like that

```
System.getProperties().put("proxySet", true);
System.getProperties().put("proxyHost", "proxyUrl");
System.getProperties().put("proxyPort", "proxyPort");
```

If you use the Geoclimate CLI try this :

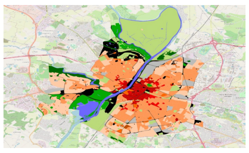
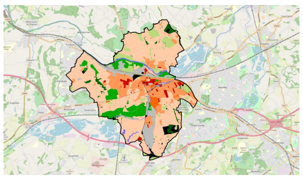
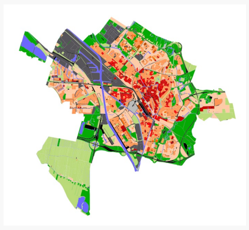
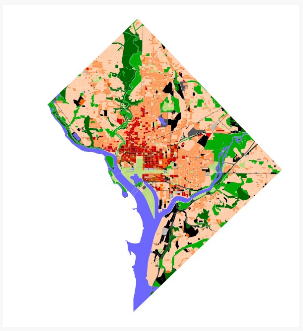
```
java -Djava.net.useSystemProxies=true -Dhttp.proxyHost=10.10.10.10 -Dhttp.proxyPort=8080  -jar  Geoclimate.jar -f osm_geoclimate.jso
n
```

if nothing works, please contact your system administrator ;-)

Gallery

This page shares some maps build from GeoClimate output.

Local Climate Zones

	
Angers city - FR (EPSG 4326)	Reading city - UK (EPSG 3857)
	
Utrecht city - NL (EPSG 3857)	Washington D.C - USA (EPSG 3857)

Legend

- 🟥 LCZ 1: Compact high-rise
- 🟥 LCZ 2: Compact mid-rise
- 🟥 LCZ 3: Compact low-rise
- 🟫 LCZ 4: Open high-rise
- 🟧 LCZ 5: Open mid-rise
- 🟧 LCZ 6: Open low-rise
- 🟨 LCZ 7: Lightweight low-rise
- ⬜ LCZ 8: Large low-rise
- 🟧 LCZ 9: Sparsely built
- ⬛ LCZ 10: Heavy industry
- 🟩 LCZ 101: Dense trees
- 🟩 LCZ 102: Scattered trees
- 🟩 LCZ 103: Bush,scrub
- 🟩 LCZ 104: Low plants
- ⬛ LCZ 105: Bare rock or paved
- 🟨 LCZ 106: Bare soil or sand
- 🟦 LCZ 107: Water

Fractions at grid scale

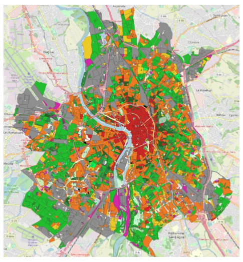
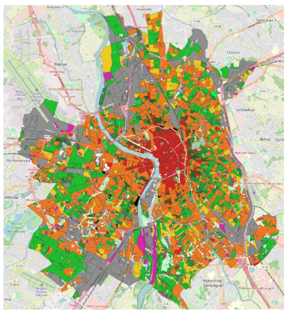
Topographic feature fractions on a 10 x 10 km domain and 1 x 1 km square cell for the city of Angers

Local Climate zone fractions on a 10 x 10 km domain and 1 x 1 km square cell for the city of Angers

Urban typologies

Toulouse city - FR (EPSG 3857)

With building area indicator	With building floor area indicator	Legend

With building area indicator	With building floor area indicator	Legend
		⬛ Industrial building ⬛ High rise building 🟥 Linear building on closed urban island 🟧 Linear building on open urban island 🟨 Detached building 🟪 Informal building 🟩 Row house on closed island 🟩 Row house on open island 🟩 Detached house 🟩 Semi-detached house ⬜ Undefined

Download

GeoClimate library uses a set of dependencies to run its algorithms. A GeoClimate package containing all these dependencies is compiled and publish after every change on the code source on our Jenkins build.

Go to GeoClimate releases page to download the last jar package. This version is ready to run the command line interface.

Integrate GeoClimate in your lib

For advanced users or developers, GeoClimate and its dependencies can be grabbed from our Nexus repository using Maven.

Set in your pom the following information

```
<dependencies>
<dependency>
  <groupId>org.orbisgis.orbisprocess</groupId>
  <artifactId>geoclimate</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</dependency
</dependencies>

<repositories>
        <repository>
            <id>orbisgis-release</id>
            <url>http://nexus.orbisgis.org/repository/orbisgis-release</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
            <releases>
                <enabled>true</enabled>
            </releases>
        </repository>
        <repository>
            <id>orbisgis-snapshot</id>
            <url>http://nexus.orbisgis.org/repository/orbisgis-snapshot</url>
            <snapshots>
                <enabled>true</enabled>
                <updatePolicy>always</updatePolicy>
            </snapshots>
            <releases>
                <enabled>false</enabled>
            </releases>
        </repository>
</repositories>
```

Coding implementation

GeoClimate algorithms are implemented as functions in Groovy Scripts.

GeoClimate is organized in 3 modules (Figure 1).

- GeoIndicators is the main module. It contains all the algorithms to build the units of analysis, compute the corresponding indicators and classify urban fabric by type. The *SpatialUnits* script creates the units of analysis (currently blocks and TSU). The *BuildingIndicators*, *BlockIndicators*, *RoadIndicators*, *RSUIndicators* calculates morphological and topographical indicators respectively at building, block, road and RSU scales. The *GenericIndicators* script calculates indicators which can be applied to any scale (e.g. the area of a unit - building, block, RSU - or the aggregation of indicator from one scale to an other - mean building height within a block or a RSU). The *TypologyClassification* script classifies units to a certain type (currently building to UTRF and TSU to LCZ) based on indicators value. The *DataUtil* script facilitates data handling (e.g. join several tables). All functions contained in the previous scripts may be called individually. To run several of them in a row, workflows are available in the *WorkflowGeoIndicators* script. The main one perform all the analysis (green arrows Figure 1): it produces the units of analysis, compute the indicators at the base scales (building and road), computes indicators at block scale, aggregate indicators from lower to upper scale, compute indicators at RSU scale and then classify urban fabric.
- OSM module extracts and transforms the OSM data to the GeoClimate abstract model. Those data processing are specified in the two scripts InputDataLoading and InputDataFormating. The *WorkflowOSM* script chains algorithms (blue arrow Figure 1): it triggers the 2 scripts dedicated to the OSM data preparation and then the *WorkflowGeoIndicators* script. It is the main entry to specify the area to be processed, the indicators and the classifications to compute.
- BDTopo_V2 module follows the same logic as the OSM module except that it is dedicated to version 2.2 of the IGN BDTopo database

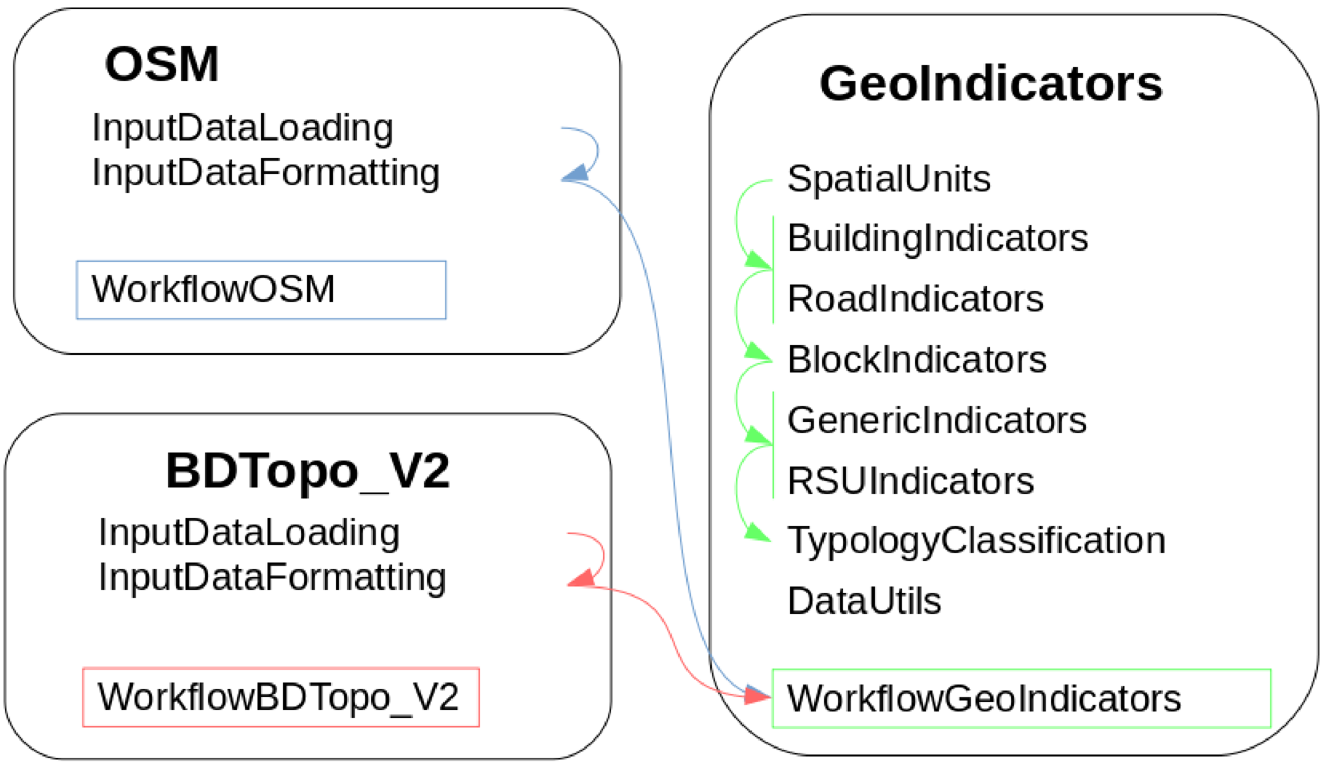


Figure 1. The GeoClimate modules

References

General presentation of the GeoClimate tool:

> *Bocher, Erwan and Bernard, Jérémy and Le Saux Wiederhold, Elisabeth and Leconte, François and Petit, Gwendall and Palominos, Sylvain and Noûs, Camille. "GeoClimate : a Geospatial processing toolbox for environmental and climate studies" Journal of Open Source Software (under review).*

For specific features in GeoClimate

The Sky View Factor calculation at Reference Spatial Unit (RSU):

> *Bernard, J., Bocher,E., Petit,G., Palominos.S. (2018). Sky View Factor Calculation in Urban Context: Computational Performance and Accuracy Analysis of Two Open and Free GIS Tools. Climate, MDPI, Urban Overheating - Progress on Mitigation Science and Engineering Applications, 6 (3), pp.60. https://www.mdpi.com/2225-1154/6/3/60*

There are currently 2 other publications pending:

- one corresponding to the presentation and evaluation of the method used to estimate the missing building height in OSM data
- one corresponding to the presentation of the method used to calculate the Local Climate Zone using OSM and BDTopo V2