geomcompare Documentation

Release 0.3.0

Mathieu Tachon

Mar 22, 2022

CONTENTS

1 Contents 1

2 Indices and tables 29

Python Module Index 31

Index 33

CHAPTER

ONE

CONTENTS

1.1 GeomCompare

Compare two sets of geometrical features.

GeomCompare provides multiple tools for comparing two independant sets of geometrical features. It can be used to identify features with similar geometry (based on pre-defined similarity functions) found in both sets, as well features with geometry that are found in only one of the sets. *GeomCompare* defines a few similarity functions, but it possible for the user to define its own customized similarity functions.

1.1.1 Installation

Requirements

GeomCompare requires `Python >= 3.9`.

In addition, for a fully fledged installation of *GeomCompare* and to have access to all functionalities provided by the library, the user need to install the following:

- `shapely`
- `numpy`
- `psycopg2`
- `rtree`
- `pyproj`
- `gdal` (core libraries and Python bindings)
- `spatialite`

> **Note:** `mod_spatialite` must be installed and accessible from `sqlite3`:

```
import sqlite3
conn = sqlite3.connect(":memory:")
conn.enable_load_extension(True)
conn.load_extension("mod_spatialite")
```

PIP

If you use `pip`, you can install *GeomCompare* with:

```
pip install geomcompare
```

1.1.2 Docker

A *Docker* image for *GeomCompare* is also available on *DockerHub*:

- Run the geomcompare image and start an iPython session inside the container:

  ```
  docker run -it mtachon/geomcompare
  ```

- Run the geomcompare image, and mount the current directory into the *data* folder of the container:

  ```
  docker run --entrypoint bash -v `pwd`:/data -w /data -it mtachon/geomcompare
  ```

For more information on *Docker* and command-line arguments, see: https://docs.docker.com/ and https://docs.docker.com/engine/reference/run/ .

1.2 Getting started

If you have not installed *GeomCompare* yet, you can follow the *installation instructions*.

1.2.1 Input/Output

Load a geometry dataset from disk

Load geometrical features from a Shapefile:

```
from geomcompare.io import extract_geoms_from_file

filename = "/path/to/my/file.shp"

# The names of supported OGR/GDAL drivers for opening files with
# geometrical features are listed in
# https://gdal.org/drivers/vector/index.html
driver_name = "ESRI Shapefile" # driver name for opening shapefiles

# Get an iterator of the geometries
geoms = extract_geoms_from_file(filename, driver_name)

# Note: the file will only be closed after the
# "extract_geoms_from_file" has yielded the last geometry.
# If you intend to iterate through "geoms" multiple times, you can
# store the geometries in a list instead
geoms_list = list(geoms) # now the file is closed
```

Filtering the extracted geometrical features:

```
from geomcompare.io import extract_geoms_from_file, LayerFilter

filename = "/path/to/my/file.json"
driver_name = "GeoJSON"

# Extract only geometrical features from the layer "my_lyr"
geoms_my_lyr = extract_geoms_from_file(
    filename=filename,
    driver_name=driver_name,
    layers=["my_lyr"],
)

# Extract only the first 10 geometrical features from the layer
# "my_lyr"
lyr_filter = LayerFilter(fids=list(range(10)))
geoms_my_lyr_10 = extract_geoms_from_file(
    filename=filename,
    driver_name=driver_name,
    layers=["my_lyr"],
    layer_filters=[lyr_filter],
)

# Area of interest
aoi = list(
    extract_geoms_from_file("/path/to/aoi_poly.shp", "ESRI Shapefile")
)[0]

# Extract features from the layer "large_lyr" that are within the
# aoi polygon, as well as all features from other layers
lyr_filter = LayerFilter(layer_id="large_lyr", aoi=aoi)
filtered_geoms = extract_geoms_from_file(
    filename=filename,
    driver_name=driver_name,
    layer_filters=[lyr_filter],
)

# Extract only features which "distance" attribute is inferior to
# 1000
lyr_filter = LayerFilter(attr_filter="distance < 1000")
geoms_dist_inf_1000 = extract_geoms_from_file(
    filename=filename,
    driver_name=driver_name,
    layer_filters=[lyr_filter],
)
```

Load a geometry dataset from a PostGIS database

```
from geomcompare.io import fetch_geoms_from_pg, ConnectionParameters, SchemaTableColumn

# Pass the correct values to keyword parameters
conn_params = ConnectionParameters(
    host="host_name",
    dbname="db_name",
    user="my_user",
    password="my_pwd",
    port=5432,
)

# Using some fictive database layout
geoms_location = SchemaTableColumn(
    schema="building",
    table="public",
    column="geom",
)

# Open a connection to the database and get an iterator of the
# geometries. The connection stays opened until the function has
# yielded the last geometry at that location in the database.
geoms = fetch_geoms_from_pg(
    conn_params=conn_params, geoms_col_loc=geoms_location,
)

# Store the geometries in a list and close the connection.
geoms_list = list(geoms)

# Get the same geometries, but this time using the "sql_query"
# parameter instead of the "geoms_col_loc" parameter. Any SQL query
# which return geometrical features can be passed as argument.
geoms_list = list(fetch_geoms_from_pg(
    conn_params=conn_params,
    sql_query="SELECT geom FROM building.public;",
))

# Area of interest
aoi = list(
    extract_geoms_from_file("/path/to/aoi_poly.shp", "ESRI Shapefile")
)[0]
# Get an iterator of the geometries from the same geometry column,
# but only those which lie within the aoi polygon. The
# "output_epsg" parameter can be use to reproject the geometries to
# the wanted spatial reference system.
geoms = fetch_geoms_from_pg(
    conn_params=conn_params,
    geoms_col_loc=geoms_location,
    aoi=aoi,
    output_epsg=25833,
)
```

Write a geometry dataset to disk

Warning: When writing to disk, *GeomCompare* assumes that all geometrical features have the same geometry type. *write_geoms_to_file()* will not check for geometry type homogeneity and will instead throw an error if the features have different geometry types. If the features have different geometry types, you can still group them into multiple datasets of homogeneous geometry type, and write these datasets to the same file on different layers, if the data format supports it, as shown below.

Write a list of geometrical features to Shapefile:

```
from geomcompare.io import write_geoms_to_file

filename = "/path/to/output/file.shp"
driver_name = "ESRI Shapefile"

# "geoms_list" is our list of geometrical features
write_geoms_to_file(
    filename=filename,
    driver_name=driver_name,
    geoms_iter=geoms_list,
    geoms_epsg=4326, # not required, but good practice if available
)
```

Write two datasets with different geometry types to the same GeoPackage file:

```
from geomcompare.io import write_geoms_to_file

filename = "/path/to/output/file.gpkg"
driver_name = "GPKG"

write_geoms_to_file(
    filename=filename,
    driver_name=driver_name,
    geoms_iter=points_list,
    geoms_epsg=25833,
    layer="my_point_layer",
)

write_geoms_to_file(
    filename=filename,
    driver_name=driver_name,
    geoms_iter=polygons_list,
    geoms_epsg=4326,
    layer="my_polygon_layer",
    mode="update",
)
```

Note: If the `geoms_epsg` parameter is given, and the layer where the geometrical features are to be written on has a different Spatial Reference System, the geometries' coordinates will be re-projected on-the-fly.

1.2.2 Comparing datasets

GeomCompare provides three main classes that can be used to compare two datasets of geometrical features:

- *SQLiteGeomRefDB*
- *PostGISGeomRefDB*
- *RtreeGeomRefDB*

These classes present an interface to store or give access to a *reference* dataset/database of geometrical features, to which a *test* dataset can be compared. Instances of these classes present a similar API, but they all have pros and cons when compared against each others. Presently, the class *SQLiteGeomRefDB* gives more flexibility to the user and will therefore be used in the following examples.

Comparison of two geometries

The main classes provided by *GeomCompare* delegates the comparison of two geometrical features to an external function. This can be a user-defined function, or one of the few comparison functions provided by *GeomCompare*. These functions' signature must match the following template:

```
comparison_function(gtest, gref) -> bool
```

where the first positional argument `gtest` is the *test* geometry (shapely geometrical object), and where the second positional argument `gref` is the *reference* geometry. If the comparison function finds that the input geometries are similar, it must return `True`. It must return `False` for different geometries.

```
from shapely.geometry import Polygon

from geomcompare.comparefunc import polygons_area_match

# The dispatch function "polygons_area_match" is not itself a
# comparison function, but it returns comparison functions with the
# right signature instead. The way the returned function compare
# two geometries depends on the values passed as arguments to
# "polygons_area_match".

# Use Intersection over Union metric for comparison
strategy = "IoU"
threshold = 0.7
comparison_f = polygons_area_match(strategy, threshold)

# Reference polygon
poly_ref = Polygon(((0, 0), (1, 0), (1, 1), (0, 1)))

# Test polygons
poly_test1 = Polygon(((0, 0), (0.5, 0), (0.5, 1), (0, 1)))
poly_test2 = Polygon(((0, 0), (0.75, 0), (0.75, 1), (0, 1)))

comparison_f(poly_test1, poly_ref) # returns False: IoU < 0.7
comparison_f(poly_test1, poly_ref) # returns True: IoU >= 0.7
```

The comparison function will be passed as argument to one of the methods of the main classes' instances to compare *test* and *reference* geometries.

Managing a reference dataset

The following code examples shows how to manage a *reference* geometry dataset using the *SQLiteGeomRefDB* class. Internally, instances of this class uses a SQLite database (with spatialite extension) to store the *reference* geometries.

Initialize a SQLiteGeomRefDB instance and populate it with geometries:

```
from geomcompare import SQLiteGeomRefDB

# Start with an empty instance.
geomref = SQLiteGeomRefDB()

filename = "/path/to/reference/dataset.shp"
driver_name = "ESRI Shapefile" # driver name for opening shapefiles

# Get an iterator of the reference geometries, let us assume that
# the geometries are polygons.
ref_polys = extract_geoms_from_file(filename, driver_name)

# Add the geometries to the SQLiteGeomRefDB instance.
geomref.add_geometries(
    ref_polys,
    geom_type="Polygon",
    geoms_epsg=25833,
    geoms_tab_name="my_ref_polys",
)
```

The code above instantiates a *SQLiteGeomRefDB* object, which internally creates a SQLite database in *RAM*, and adds *reference* polygons from a *shapefile* to a table named "my_ref_polys". As we have created a new database, the `geom_type` and parameter of *add_geometries()* must be passed an argument, since the geometry type is required by *spatialite* when creating a new geometry column in a table of the database. If the *default_epsg* was not set (either when creating the instance or at least before adding the new geometries), `geoms_epsg` (identifying the spatial reference system of the input *reference* geometries) must also be given when calling *add_geometries()*.

1.3 License

1.3.1 GNU General Public License

Version 3, 29 June 2007 Copyright © 2007 Free Software Foundation, Inc <http://fsf.org>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you

receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: **(1)** assert copyright on the software, and **(2)** offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that **(1)** displays an appropriate copyright notice, and **(2)** tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License,

and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that **(a)** is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and **(b)** serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- **a)** The work must carry prominent notices stating that you modified it, and giving a relevant date.
- **b)** The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- **c)** You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- **d)** If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- **a)** Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- **b)** Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either **(1)** a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or **(2)** access to copy the Corresponding Source from a network server at no charge.
- **c)** Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- **d)** Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- **e)** Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either **(1)** a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or **(2)** anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in

which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- **a)** Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- **b)** Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- **c)** Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- **d)** Limiting the use for publicity purposes of names of licensors or authors of the material; or
- **e)** Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- **f)** Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated **(a)** provisionally, unless and until the copyright holder explicitly and finally terminates your license, and **(b)** permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either **(1)** cause the Corresponding Source to be so available, or **(2)** arrange to deprive yourself of the benefit of the patent license for this particular work, or **(3)** arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license **(a)** in connection with copies of the covered work conveyed by you (or copies made from those copies), or **(b)** primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

1.4 Contributors

- Mathieu Tachon <tachon.mathieu@protonmail.com> - Main author

1.5 Changelog

1.6 Changelog

1.6.1 v0.3.0 (2022-03-22)

New Features

- (SQLiteGeomRefDB): add the get_geometries method #### Refactorings
- (comparefunc): rename geoms_always_match -> _geoms_always_match #### Docs
- change homepage link in README.rst
- add usage section with link to homepage for GitHub repo #### Others
- remove commitizen section from pyproject.toml
- fix setup.cfg so that version can be found
- import tag and metadata from master

Full set of changes: ``v0.2.2...v0.3.0` <https://github.com/kartverket/GeomCompare/compare/v0.2.2...v0.3.0>`_

1.6.2 v0.2.2 (2022-03-18)

Refactorings

- prefix a few private variables/functions with an underscore
- Change LayerID from TypeVar to Union.
- Type for shapely geometrical object and reformatting of docstrings #### Docs
- update CHANGELOG.md
- Update the README.rst file
- Add content to getting started page.
- Update auto-generated CHANGELOG.md to fix bad formatted commit msg.
- Add shapely to intersphinx_mapping
- “geometrical/geographical” -> “geometrical”
- Fixed table of contents #### Others
- Add the Dockerfile to the repo.

1.7 geomcompare

1.7.1 geomcompare package

GeomCompare

The *Geomcompare* package provides multiple tools for comparing two independant sets of geometrical features.

Documentation for *GeomCompare* is available in the form of docstrings provided with the code, as well as on the project's homepage https://geomcompare.readthedocs.io/en/latest/.

Available submodules

geomrefdb Defines the main classes of the library used for comparing geometry datasets.

io Provides a set of tools for I/O operations, extracting geometrical features from disk or from a PostGIS database, as well as writing a dataset of geometries to disk.

comparefunc Defines a few comparison functions to use with the geomrefdb's main classes.

geomutils Defines a few functions and types to work with shapely geometrical objects.

stats Defines functions for computing classifier metrics (e.g. when comparing a result dataset from a machine learning model with a reference dataset).

Submodules

geomcompare.comparefunc module

geomcompare.geomrefdb module

class geomcompare.geomrefdb.**PostGISGeomRefDB**(*PG_params*, *PG_schema*, *PG_table*, *PG_geoms_column*)
Bases: geomcompare._geomrefdb_abc.GeomRefDB

true_positives(*geoms_iter*, *geoms_EPSG*, *same_geoms_func*)
Return an iterable of input geometries that are matching geometries of the GeomRefDB instance.

false_positives(*geoms_iter*, *geoms_EPSG*, *same_geoms_func*)
Return an iterable of input geometries that are not matching any geometry of the GeomRefDB instance.

missing_geometries(*geoms_iter*, *AOI_geom*, *geoms_EPSG*, *same_geoms_func*)
Return an iterable of geometries of the GeomRefDB that are not matching any of the input geometries.

class geomcompare.geomrefdb.**RtreeGeomRefDB**(*geoms_iter*, *geoms_EPSG*)
Bases: geomcompare._geomrefdb_abc.GeomRefDB

true_positives(*geoms_iter*, *geoms_EPSG*, *same_geoms_func*)
Return an iterable of input geometries that are matching geometries of the GeomRefDB instance.

false_positives(*geoms_iter*, *geoms_EPSG*, *same_geoms_func*)
Return an iterable of input geometries that are not matching any geometry of the GeomRefDB instance.

missing_geometries(*geoms_iter*, *AOI_geom*, *geoms_EPSG*, *same_geoms_func*)
Return an iterable of geometries of the GeomRefDB that are not matching any of the input geometries.

geomcompare.geomrefdb.**SpatialiteGeomType = typing.Literal['Point', 'LineString', 'Polygon', 'MultiPoint', 'MultiLineString', 'MultiPolygon', 'GeometryCollection']**

Type: _LiteralGenericAlias

Geometry types supported by the *SQLiteGeomRefDB* class.

class geomcompare.geomrefdb.**SQLiteGeomRefDB**(*filename=None*, *default_epsg=None*, *geoms_iter=None*, *geoms_tab_name=None*, *geom_type=None*, *geoms_epsg=None*, *in_ram=True*, *logger=None*, *logger_name=None*, *logging_level=20*)

Bases: geomcompare._geomrefdb_abc.GeomRefDB

Concrete implementation of the GeomRefDB ABC using SQLite.

SQLiteGeomRefDB is a concrete implementation of the interface defined by the GeomRefDB abstract base class. It enables to load an existing (or create a new) SQLite database, where geometry datasets can be stored and can be compared (based on geometry similarity functions) with other geometrical features from an external dataset. Instances of this class can handle simultaneously multiple reference datasets, with various geometry types (see *supported_geom_types*) and spatial reference systems.

Parameters

- **filename** (str, optional) – Path to an existing spatialite database.
- **default_epsg** (int, optional) – Default EPSG code of the geometrical features that will be added to the database. If specified, the EPSG code will be default value of the geoms_epsg parameter for any subsequent call of the *add_geometries()* method.
- **geoms_iter** (iterable of *GeomObject*, optional) – Iterable of the geometrical features to add to this *SQLiteGeomRefDB* instance. Such features can also be added later to the class instance with the *add_geometries()* method.
- **geoms_tab_name** (str, optional) – Name of the table where the geometrical features are to be stored. If the geoms_iter parameter is not given, geoms_tab_name will be ignored.
- **geom_type** (*SpatialiteGeomType*, optional) – Geometry type of the geometrical features passed as argument to the geoms_iter parameter.
- **geoms_epsg** (int, optional) – EPSG code of the geometrical features passed as argument to the geoms_iter parameter. If specified, it overrides the default_epsg parameter during the instance construction.
- **in_ram** (bool, default: True) – Set to True to create/load the database in RAM for faster access. Set to False for larger-than-RAM databases.
- **logger** (logging.Logger, optional) – Logger instance to use for logging outputs.
- **logger_name** (str, optional) – Name of the logging.Logger object to create for logging outputs. This parameter will be ignored if a Logger instance is passed to the logger parameter.
- **logging_level** (int, default: logging.INFO) – Logging level of the logging output. For mor information, please see the documentation of the logging module.

Raises ValueError – If in_ram=False and filename=None.

Notes

This class makes use of the spatialite extension of SQLite, and as such, spatialite must be installed and available in order to work with instances of this class.

class property supported_geom_types
Types supported by *SQLiteGeomRefDB*.

Type list of supported geometry types

property filename
Path of the opened database file. The attribute is set to None if a new database was created in RAM for this instance.

property in_ram
True if the database is created/loaded in RAM. False if the instance is connected to database file on disk.

property default_epsg
Default EPSG code of the geometrical features that are added to the database.

property logger
Ready configured Logger instance used for logging outputs.

save_db(*filename*, *overwrite=True*)
Save the internal SQLite database to disk.

The function saves the internal SQLite database, together with all the geometrical features added with *add_geometries()*, to disk. The path of the resulting output file can later be passed to the filename argument of the *SQLiteGeomRefDB* class' constructor to load the saved database with all its features. This function is useful only to save loaded-in-RAM databases, as the geometrical features added to a *SQLiteGeomRefDB* instance, with an opened connections to databases that reside on disk, will be saved automatically even after the instance destruction.

Parameters

- **filename** (str) – Path of the output database file.
- **overwrite** (bool, default: True) – True if the output file should overwrite any existing file at path filename, else False.

Return type None

add_geometries(*geoms_iter*, *geom_type=None*, *geoms_epsg=None*, *geoms_tab_name=None*)
Add geometrical features to the internal SQLite database.

The function adds geometrical features to the internal SQLite database, which can then be used as a "reference dataset" when running other public methods of the *SQLiteGeomRefDB* instance.

Parameters

- **geoms_iter** (iterable of *GeomObject*) – Iterable of the geometrical features to add to this *SQLiteGeomRefDB* instance.
- **geom_type** (*SpatialiteGeomType*, optional) – Geometry type of the input geometrical features. If the geom_type is not specified by the user, the function will assume that the input features have the same geometry type as the features already stored in the destination table.
- **geoms_epsg** (int, optional) – EPSG code of the input geometrical features. If the geoms_epsg is not specified by the user, the function will assume that the input features are in the same spatial reference system as the features already stored in the destination table. Also, if the input features are to be stored in a new table of the database and the

geoms_epsg is omitted, the SSQLiteGeomRefDB instance will use the EPSG code stored in the *default_epsg* attribute (if set).

- **geoms_tab_name** (str, optional) – Name of the table where the input geometrical features are to be stored in the internal SQLite database. If no argument is passed to the geoms_tab_name parameter, the function will try to store the input geometrical features into a table named *default_table*. The *default_table* table will be created if it does not already exist in the database.

Raises

- **ValueError** – If geom_type is not specified, in the case of a new database/table.
- **ValueError** – If geoms_epsg is not specified, in the case of a new database/table and if the *default_epsg* attribute is not set.
- **ValueError** – If the argument passed to the geom_type parameter does not match the geometry type of the features already stored in the destination table.

Warning: The *geometry type* must be the same for all input features as they are to be stored in the same *geometry column* of the same table, and spatialite does not allow *geometry columns* to have mixed *geometry types*.

Return type None

get_geometries(*aoi_geom=None*, *aoi_epsg=None*, *geoms_tab_name=None*, *output_epsg=None*)

Get geometrical features from the internal SQLite database.

Generator function which yields geometrical features stored in the internal database. The user can specify the table, or define a limited area to yield the features from. In addition, the spatial reference system of the output geometries can also be specified.

Parameters

- **aoi_geom** (*GeomObject*, optional) – *Area of interest*, where the geometrical features lies.
- **aoi_epsg** (int, optional) – EPSG code of the *area of interest* geometry/ies.
- **geoms_tab_name** (str, optional) – Name of the table where the geometrical features are stored in the internal SQLite database. If no argument is passed to the geoms_tab_name parameter, the function will try to yield geometrical features from a table named *default_table*.
- **output_epsg** (int, optional) – EPSG code of the yielded geometrical features. This parameter can be used to transform the yielded geometries to a different Spatial Reference System from the one used in the internal database.

Yields *GeomObject* – Geometrical features from the internal SQLite database.

Raises **ValueError** – If geoms_tab_name is not specified and no table named *default_table* exist in the database.

db_geom_info(*to_stdout=False*, *count_features=False*)

Get information on features stored in the internal SQLite database.

Get information on the geometrical features such as the name of the table(s) where they are stored, their geometry type(s), spatial reference system(s) and the number of features per table. This information can be returned as dict instance, or printed to *stdout*.

Parameters

- **to_stdout** (bool, default: `False`) – If set to `False`, the information is returned as a `dict`. If set to `True`, the information is written to *stdout*.
- **count_features** (bool, default: `False`) – If set to `True`, the function will also return the number of features/rows per table. If set to `False`, the features will not be counted.

Returns If `to_stdout=False`, returns a `dict` which keys are the table name(s), and which values are information on the individual table(s). This information is itself structured as a `dict`, which key/value pairs indicate for each table the geometry type (key: *geom_type*), the spatial reference system (key: *srid*), and optionally (if `count_features=True`) the features count (key: *count*). The function returns `None` if `to_stdout=True`.

Return type `dict` or `None`

true_positives(*geoms_iter*, *aoi_geom=None*, *geoms_epsg=None*, *geoms_tab_name=None*, *geoms_match=None*, *get_search_frame=None*, *ncores=None*)

Identidy *matching* **input** geometries.

The function takes as input geometrical features, and searches for *reference features* in one table of the internal database which geometries are considered to *match* that of the *input features*. All *input features* that have a geometry that *matches* the geometry of at least one of the *reference features* will be yielded back by the function.

Parameters

- **geoms_iter** (iterable of *GeomObject*) – Iterable of input geometrical features to compare to the features of the internal SQLite database.
- **aoi_geom** (*GeomObject*, optional) – *Area of interest*, within which the database's features must lie.
- **geoms_epsg** (`int`, optional) – EPSG code of the input geometrical features (including `aoi_geom` if specified). If the `geoms_epsg` is not specified by the user, the function will assume that the *input features* are in the same spatial reference system as the *reference features*.
- **geoms_tab_name** (`str`, optional) – Name of the table where database's features that will be used as reference are stored. If no argument is passed to the `geoms_tab_name` parameter, the function will search for *reference features* in a table named *default_table*.
- **geoms_match** (`callable`, optional) – Comparison function that takes two positional arguments:
  - `gtest`: *input* geometry (*GeomObject*)
  - `gref`: *reference* geometry (*GeomObject*)

  The function returns `True` if it finds that both geometries *match*, else returns `False`. If this parameter is omitted, the *input* geometrical feature will always be considered as a *match* in the case where its *search frame* (see `get_search_frame` parameter) interesects with one of the feature from the database's table.
- **get_search_frame** (`callable`, optional) – Function that takes as single argument an *input* geometry (*GeomObject*) and returns its *search frame* (*GeomObject*). If this parameter is omitted, the *search frame* will be the same as the *input* geometry.
- **ncores** (`int`, optional) – Number of cores to use for running the function. If unspecified, the function will run in a single process

Yields *GeomObject* – *Matching input* geometrical features.

Notes

If the *spatial reference system* of the *input* geometrical features is different from that of the database's features, the *input features*' coordinates are reprojected on-the-fly, before being compared to features stored in the database. If an *input feature* is considered to be a *match*, it is yielded back unchanged (its coordinates in the original *spatial reference system*).

false_positives(*geoms_iter*, *aoi_geom=None*, *geoms_epsg=None*, *geoms_tab_name=None*, *geoms_match=None*, *get_search_frame=None*, *ncores=None*)

Identify *non-matching* **input** geometries.

The function takes as input geometrical features, and searches for *reference features* in one table of the internal database which geometries are considered to *match* that of the *input features*. All *input features* that **DO NOT** have a geometry that matches the geometry of any *reference features* will be yielded back by the function.

Parameters

- **geoms_iter** (iterable of *GeomObject*) – Iterable of input geometrical features to compare to the features of the internal SQLite database.
- **aoi_geom** (*GeomObject*, optional) – *Area of interest*, within which the database's features must lie.
- **geoms_epsg** (int, optional) – EPSG code of the input geometrical features (including `aoi_geom` if specified). If the `geoms_epsg` is not specified by the user, the function will assume that the *input features* are in the same spatial reference system as the *reference features*.
- **geoms_tab_name** (str, optional) – Name of the table where database's features that will be used as reference are stored. If no argument is passed to the `geoms_tab_name` parameter, the function will search for *reference features* in a table named *default_table*.
- **geoms_match** (callable, optional) – Comparison function that takes two positional arguments:
  - `gtest`: *input* geometry (*GeomObject*)
  - `gref`: *reference* geometry (*GeomObject*)

  The function returns `True` if it finds that both geometries *match*, else returns `False`. If this parameter is omitted, the *input* geometrical feature will always be considered as a *match* in the case where its *search frame* (see `get_search_frame` parameter) interesects with one of the features from the database's table.
- **get_search_frame** (callable, optional) – Function that takes as single argument an *input* geometry (*GeomObject*) and returns its *search frame* (*GeomObject*). If this parameter is omitted, the *search frame* will be the same as the *input* geometry.
- **ncores** (int, optional) – Number of cores to use for running the function. If unspecified, the function will run in a single process

Yields *GeomObject* – *Non-matching input* geometrical features.

Notes

If the *spatial reference system* of the *input* geometrical features is different from that of the database's features, the *input features*' coordinates are reprojected on-the-fly, before being compared to features stored in the database. If an *input feature* is **NOT** considered to be a *match*, it is yielded back unchanged (its coordinates in the original *spatial reference system*).

missing_geometries(*geoms_iter*, *geom_type=None*, *aoi_geom=None*, *geoms_epsg=None*, *geoms_tab_name=None*, *geoms_match=None*, *get_search_frame=None*, *ncores=None*)

Identify (missing) *non-matching* **reference** geometries.

The function takes as input geometrical features, and searches for *reference features* in one table of the internal database which geometries are **NOT** considered to *match* the geometry of any feature from the input set. All *reference features* that **DO NOT** have a geometry that *matches* the geometry of any *input features* will be yielded by the function.

Parameters

- **geoms_iter** (iterable of *GeomObject*) – Iterable of input geometrical features to compare to the features of the internal SQLite database.
- **geom_type** (*SpatialiteGeomType*, optional) – Geometry type of the input geometrical features. If the `geom_type` is not specified by the user, the function will assume that the *input features* have the same `geom_type` as the *reference features*.
- **aoi_geom** (*GeomObject*, optional) – *Area of interest*, within which the database's features must lie.
- **geoms_epsg** (`int`, optional) – EPSG code of the input geometrical features (including `aoi_geom` if specified). If the `geoms_epsg` is not specified by the user, the function will assume that the *input features* are in the same spatial reference system as the *reference features*.
- **geoms_tab_name** (`str`, optional) – Name of the table where database's features that will be used as reference are stored. If no argument is passed to the `geoms_tab_name` parameter, the function will search for *reference features* in a table named *default_table*.
- **geoms_match** (`callable`, optional) – Comparison function that takes two positional arguments:
  - `gtest`: *input* geometry (*GeomObject*)
  - `gref`: *reference* geometry (*GeomObject*)

  The function returns `True` if it finds that both geometries *match*, else returns `False`. If this parameter is omitted, the *input* geometrical feature will always be considered as a *match* in the case where its *search frame* (see `get_search_frame` parameter) interesects with one of the features from the database's table.
- **get_search_frame** (`callable`, optional) – Function that takes as single argument an *input* geometry (*GeomObject*) and returns its *search frame* (*GeomObject*). If this parameter is omitted, the *search frame* will be the same as the *input* geometry.
- **ncores** (`int`, optional) – Number of cores to use for running the function. If unspecified, the function will run in a single process

Yields *GeomObject* – *Non-matching reference* geometrical features.

Notes

If the *spatial reference system* of the *input* geometrical features is different from that of the database's features, the *input features*' coordinates are reprojected on-the-fly, before being compared to features stored in the database.

geomcompare.geomutils module

geomcompare.geomutils.**GeomObject = typing.Union[shapely.geometry.polygon.LinearRing, shapely.geometry.linestring.LineString, shapely.geometry.multilinestring.MultiLineString, shapely.geometry.multipoint.MultiPoint, shapely.geometry.multipolygon.MultiPolygon, shapely.geometry.point.Point, shapely.geometry.polygon.Polygon]**

Type: _UnionGenericAlias

Type for shapely geometrical objects.

geomcompare.geomutils.**to_2D**(*geom*)

Remove the third dimension of a geometrical object's coordinates.

Parameters **geom** (*GeomObject*) – Shapely geometrical object with XYZ-coordinates.

Returns Geometrical object with its Z-coordinates removed.

Return type *GeomObject*

geomcompare.geomutils.**get_transform_func**(*epsg_in*, *epsg_out*)

Get function to transform a geometrical object to another SRS.

Create and return a function that transforms the XY-coordinates of *GeomObject* instances from one spatial reference system to another. The function identifies input and output spatial reference systems by the EPSG code.

Parameters

- **epsg_in** (int) – EPSG code of the input spatial reference system.
- **epsg_out** (int) – EPSG code of the output spatial reference system.

Returns Function that takes one *GeomObject* as positional argument and returns the *GeomObject* with its XY-coordinates transformed to the output spatial reference system.

Return type callable

geomcompare.io module

namedtuple geomcompare.io.**ConnectionParameters**(*host*, *dbname*, *user*, *password*, *port=5432*)

Bases: NamedTuple

Parameters to open a connection to a PostGIS database.

Instances of this class are intended to be used as parameter for the *fetch_geoms_from_pg* function.

Fields

0) **host** (str) – Database host address.
1) **dbname** (str) – Database name.
2) **user** (str) – User name used to authenticate.
3) **password** (str) – Password used to authenticate.

4) **port** (int) – Connection port number.

namedtuple geomcompare.io.**SchemaTableColumn**(*schema*, *table*, *column*)

Bases: NamedTuple

Location of a geometry column in a PostGIS database.

Instances of this class are intended to be used as parameter for the *fetch_geoms_from_pg* function.

Fields

0) **schema** (str) – Schema name of the PostGIS database, where the table containing the geometrical features is located.

1) **table** (str) – Table name, where the geometrical features can be found.

2) **column** (str) – Column name, where the geometrical features can be found.

geomcompare.io.**fetch_geoms_from_pg**(*conn=None*, *conn_params=None*, *sql_query=None*, *geoms_col_loc=None*, *aoi=None*, *aoi_epsg=None*, *output_epsg=None*)

Fetch geometrical features from a PostGIS database.

Generator function which connects or uses an existing connection to a PostGIS database, and yields geometrical features from specified geometry column (within a given area or not), or based on a user-defined SQL query. If the connection to the database is opened by the function, it will be closed automatically after the last geometrical feature is yielded.

Parameters

- **conn** (psycopg2.extensions.connection, optional) – Pre-opened connection to the PostGIS database.
- **conn_params** (*ConnectionParameters*, optional) – Parameters to open a connection to the PostGIS database.
- **sql_query** (str, optional) – SQL query to use to extract geometrical features from the PostGIS database.
- **geoms_col_loc** (*SchemaTableColumn*, optional) – Geometry column location within the PostGIS database.
- **aoi** (*GeomObject*, optional) – *Area of interest*, where the geometrical features lies.
- **aoi_epsg** (int, optional) – EPSG code of the *area of interest* geometry/ies.
- **output_epsg** (int, optional) – EPSG code of the yielded geometrical features. This parameter can be used to transform the yielded geometries to a different Spatial Reference System from the one used in the PostGIS database.

Yields *GeomObject* – Geometrical features from the PostGIS database.

Raises

- **ValueError** – If both conn and conn_params parameters are not passed an argument different from None.
- **ValueError** – If both sql_query and geoms_col_loc parameters are not passed an argument different from None.

Notes

In the case where the `sql_query` parameter is given, the parameters `geoms_col_loc`, `aoi`, `aoi_epsg` and `output_epsg` will be ignored, as SQL queries can include filtering and reprojection.

geomcompare.io.**LayerID = typing.Union[str, int]**
Type: _UnionGenericAlias

Type for identifying layers.

namedtuple geomcompare.io.**LayerFilter**(*layer_id=None*, *aoi=None*, *aoi_epsg=None*, *attr_filter=None*, *fids=None*)

Bases: NamedTuple

Filter for extraction of geometrical features from file.

Instances of this class are intended to be used as parameter for the *extract_geoms_from_file* function, for filtering and choosing the geometrical features to extract.

Fields

0) **layer_id** (Union[str, int, None]) – Name or index of the layer the filter will be applied to. If set to None, the filter will be applied on all layers.
1) **aoi** (Union[LinearRing, LineString, MultiLineString, MultiPoint, MultiPolygon, Point, Polygon, None]) – *GeomObject*, optional: *Area of interest*, where the geometrical features lies. All features lying outside the *area of interest* will be filtered out (not extracted).
2) **aoi_epsg** (Optional[int]) – EPSG code of the *area of interest* geometry/ies. If set to None, the same Spatial Reference System as the layer will be used.
3) **attr_filter** (Optional[str]) – Valid string representation of an attribute filter (e.g. "attr_name = 'value'").
4) **fids** (Optional[Sequence[int]]) – IDs of the features to extract from the layer. This parameter will be ignored if either the `aoi` or the `attr_filter` parameters are specified by the user.

geomcompare.io.**extract_geoms_from_file**(*filename*, *driver_name*, *layers=None*, *layer_filters=None*)

Extract geometrical features from a GDAL/OGR-readable file.

Generator function which opens a file located on disk, with one of the existing GDAL/OGR drivers, and yields geometrical features, from one or several layers. The function also permits the use of filters to allow for fine-grained extraction of the geometrical features.

Parameters

- **filename** (str) – Path to the file to extract the geometrical features from.
- **driver_name** (*str*) – Name of the GDAL/OGR driver to use for opening the file.
- **layers** (sequence of *LayerID*, optional) – Layers from which the geometrical features will be extracted. If set to None (default), geometrical features will be extracted from all layers.
- **layer_filters** (sequence of *LayerFilter*, optional) – Filters to apply to the layer(s) when extracting the geometrical features.

Yields *GeomObject* – Geometrical features from the file.

Raises NotImplementedError – If GDAL/OGR is not installed or not importable.

geomcompare.io.**write_geoms_to_file**(*filename*, *driver_name*, *geoms_iter*, *geoms_epsg=None*, *layer=None*, *mode='update'*)

Write multiple geometrical features to disk.

The function takes as input an iterable of geometrical features and writes them to disk using one of the existing GDAL/OGR drivers.

Parameters

- **filename** (str) – Path to the output file where the geometrical features will be written to.
- **driver_name** (str) – Name of the GDAL/OGR driver to use for writing the file.
- **geoms_iter** (iterable of *GeomObject*) – Iterable of the geometrical features to write.
- **geoms_epsg** (int, optional) – EPSG code of the input geometrical features. If the Spatial Reference System of the input geometrical features is specified and differs from that of the layer they will written to (in case of an update, see``mode`` parameter), the coordinates of the geometries will be reprojected to the layer's Spatial Reference System. It is set to None as default (no Spatial Reference System).
- **layer** (*LayerID*, optional) – Layer name/index on which to write the input geometries. In case of a file update (see mode parameter), the index of an existing layer can be passed as argument. If layer is set to None (default), the geometrical features will be written, in update mode, on the first layer available (at index 0), if any. If no layer is available, as well as in overwrite mode, the layer parameter set to None will result in the function writing the input geometries to a layer named default (if the driver supports named layers).
- **mode** ({"update", "overwrite"}) – If set to "update", the function will update an existing file, or will create it if it does not exist. If set to "overwrite", the function will delete any file at the path set to the filename parameter, and will create a new file at this same location.

Return type None

geomcompare.stats module

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

geomcompare, 17
geomcompare.comparefunc, 17
geomcompare.geomrefdb, 17
geomcompare.geomutils, 24
geomcompare.io, 24
geomcompare.stats, 27

INDEX

A

add_geometries() (*geomcompare.geomrefdb.SQLiteGeomRefDB method*), 19

D

db_geom_info() (*geomcompare.geomrefdb.SQLiteGeomRefDB method*), 20
default_epsg (*geomcompare.geomrefdb.SQLiteGeomRefDB property*), 19

E

extract_geoms_from_file() (*in module geomcompare.io*), 26

F

false_positives() (*geomcompare.geomrefdb.PostGISGeomRefDB method*), 17
false_positives() (*geomcompare.geomrefdb.RtreeGeomRefDB method*), 17
false_positives() (*geomcompare.geomrefdb.SQLiteGeomRefDB method*), 22
fetch_geoms_from_pg() (*in module geomcompare.io*), 25
filename (*geomcompare.geomrefdb.SQLiteGeomRefDB property*), 19

G

geomcompare
    module, 17
geomcompare.comparefunc
    module, 17
geomcompare.geomrefdb
    module, 17
geomcompare.geomutils
    module, 24
geomcompare.io
    module, 24
geomcompare.stats
    module, 27
GeomObject (*in module geomcompare.geomutils*), 24
get_geometries() (*geomcompare.geomrefdb.SQLiteGeomRefDB method*), 20
get_transform_func() (*in module geomcompare.geomutils*), 24

I

in_ram (*geomcompare.geomrefdb.SQLiteGeomRefDB property*), 19

L

LayerID (*in module geomcompare.io*), 26
logger (*geomcompare.geomrefdb.SQLiteGeomRefDB property*), 19

M

missing_geometries() (*geomcompare.geomrefdb.PostGISGeomRefDB method*), 17
missing_geometries() (*geomcompare.geomrefdb.RtreeGeomRefDB method*), 17
missing_geometries() (*geomcompare.geomrefdb.SQLiteGeomRefDB method*), 23
module
    geomcompare, 17
    geomcompare.comparefunc, 17
    geomcompare.geomrefdb, 17
    geomcompare.geomutils, 24
    geomcompare.io, 24
    geomcompare.stats, 27

P

PostGISGeomRefDB (*class in geomcompare.geomrefdb*), 17

R

RtreeGeomRefDB (*class in geomcompare.geomrefdb*), 17

S

save_db() (*geomcompare.geomrefdb.SQLiteGeomRefDB method*), 19
SpatialiteGeomType (*in module geomcompare.geomrefdb*), 17
SQLiteGeomRefDB (*class in geomcompare.geomrefdb*), 18
supported_geom_types (*geomcompare.geomrefdb.SQLiteGeomRefDB property*), 19

T

to_2D() (*in module geomcompare.geomutils*), 24
true_positives() (*geomcompare.geomrefdb.PostGISGeomRefDB method*), 17
true_positives() (*geomcompare.geomrefdb.RtreeGeomRefDB method*), 17
true_positives() (*geomcompare.geomrefdb.SQLiteGeomRefDB method*), 21

W

write_geoms_to_file() (*in module geomcompare.io*), 26