Mappymatch

Release 0.4.1

National Renewable Energy Laboratory

May 08, 2024

DOCUMENTATION INDEX:

Mappymatch is a pure-python package developed and open sourced by the National Renewable Energy Laboratory. It contains a collection of "Matchers" that enable matching a GPS trace (series of GPS coordinates) to a map.

CHAPTER

ONE

THE CURRENT MATCHERS ARE:

- `LCSSMatcher` A matcher that implements the LCSS algorithm described in this paper. Works best with high resolution GPS traces.
- `OsrmMatcher` A light matcher that pings an OSRM server to request map matching results. See the official documentation for more info.
- `ValhallaMatcher` A matcher to ping a Valhalla server for map matching results.

CHAPTER

TWO

CURRENTLY SUPPORTED MAP FORMATS ARE:

- Open Street Maps

2.1 Quickstart

Checkout this notebook for a detailed example of using the LCSSMatcher.

2.2 Install

2.2.1 From Source (recommended)

Clone the repo:

```
git clone https://github.com/NREL/mappymatch.git && cd mappymatch
```

Then, use the environment.yml file (in the repo) to install dependencies:

```
conda env create -f environment.yml
```

To activate the mappymatch environment:

```
conda activate mappymatch
```

2.2.2 From PyPI

```
pip install mappymatch
```

Warning: Some users have reported problems installing GDAL (a dependency of geopandas) on windows. Here are two possible solutions:

1) Install GDAL from source This is the most difficult solution, but is trusted.

Before installing the required dependencies, install GDAL into the system. This process is documented by UCLA.

2) Install GDAL from binary wheel This is the easiest solution, but it is from an untrusted source and is not to be used in sensitive environments.

A precompiled binary wheel is provided by Christoph Gohlke of the Laboratory for Fluorescence Dynamics at the University of California. If you use this approach, both GDAL and Fiona wheels need to be installed.

- GDAL wheels
- Fiona wheels

1. Download the correct GDAL and Fiona wheels for your architecture and Python version
2. Install the GDAL wheel into the virtual environment *pip install <path_to_GDAL_whl>*
3. Install the Fiona wheel into the virtual environment *pip install <path_to_Fiona_whl>*
4. Use pip to install the remaining packages *pip install <path/to/mappymatch>*

2.2.3 From Conda

We're currently working on a Conda package for mappymatch.

2.3 Examples

2.3.1 Example Usage

Checkout this notebook for a detailed example of using the LCSSMatcher.

2.4 API Reference

2.4.1 Constructs

Constructs are the fundamental data structures shared between all matching implementations. For readability, the full path is listed below and not used in other parts of the documentation.

Available Constructs

For readability, the full path is listed below and not used in other parts of the documentation.

- *mappymatch.constructs.coordinate.Coordinate*
- *mappymatch.constructs.geofence.Geofence*
- *mappymatch.constructs.match.Match*
- *mappymatch.constructs.road.Road*
- *mappymatch.constructs.trace.Trace*

Coordinate

class Coordinate(*coordinate_id: Any*, *geom: Point*, *crs: CRS*)

Represents a single coordinate with a CRS and a geometry

coordinate_id

The unique identifier for this coordinate

Type Any

geom

The geometry of this coordinate

Type shapely.geometry.point.Point

crs

The CRS of this coordinate

Type pyproj.crs.crs.CRS

x

The x value of this coordinate

y

The y value of this coordinate

classmethod from_lat_lon(*lat: float*, *lon: float*) → *mappymatch.constructs.coordinate.Coordinate*

Build a coordinate from a latitude/longitude

Parameters

- **lat** – The latitude
- **lon** – The longitude

Returns A new coordinate

to_crs(*new_crs: Any*) → *mappymatch.constructs.coordinate.Coordinate*

Convert this coordinate to a new CRS

Parameters **new_crs** – The new CRS to convert to

Returns A new coordinate with the new CRS

Raises **A ValueError if it fails to convert the coordinate** –

Geofence

class Geofence(*crs: pyproj.crs.crs.CRS*, *geometry: shapely.geometry.polygon.Polygon*)

A geofence is basically a shapely polygon with a CRS

Parameters

- **geom** – The polygon geometry of the geofence
- **crs** – The CRS of the geofence

classmethod from_geojson(*file: Union[pathlib.Path, str]*) → *mappymatch.constructs.geofence.Geofence*

Creates a new geofence from a geojson file.

Parameters **file** – The path to the geojson file

Returns A new geofence

classmethod from_trace(*trace: mappymatch.constructs.trace.Trace*, *padding: float = 1000.0*, *crs: pyproj.crs.crs.CRS = <Geographic 2D CRS: EPSG:4326> Name: WGS 84 Axis Info [ellipsoidal]: - Lat[north]: Geodetic latitude (degree) - Lon[east]: Geodetic longitude (degree) Area of Use: - name: World. - bounds: (-180.0, -90.0, 180.0, 90.0) Datum: World Geodetic System 1984 ensemble - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *buffer_res: int = 2*) → *mappymatch.constructs.geofence.Geofence*

Create a new geofence from a trace.

This is done by computing a radial buffer around the entire trace (as a line).

Parameters

- **trace** – The trace to compute the bounding polygon for.
- **padding** – The padding (in meters) around the trace line.
- **crs** – The coordinate reference system to use.
- **buffer_res** – The resolution of the surrounding buffer.

Returns The computed bounding polygon.

to_geojson() → str

Converts the geofence to a geojson string.

Match

class Match(*road: Optional[*mappymatch.constructs.road.Road*]*, *coordinate:* mappymatch.constructs.coordinate.Coordinate, *distance: float*)

Represents a match made by a Matcher

road

The road that was matched; None if no road was found;

Type Optional[*mappymatch.constructs.road.Road*]

coordinate

The original coordinate that was matched;

Type *mappymatch.constructs.coordinate.Coordinate*

distance

The distance to the matched road; If no road was found, this is infinite

Type float

set_coordinate(*c:* mappymatch.constructs.coordinate.Coordinate)

Set the coordinate of this match

Parameters **c** – The new coordinate

Returns The match with the new coordinate

to_flat_dict() → dict

Convert this match to a flat dictionary

Returns A flat dictionary with all match information

Road

class Road(*road_id: RoadId*, *geom: LineString*, *metadata: Optional[dict] = None*)

Represents a road that can be matched to;

road_id

The unique identifier for this road

Type *mappymatch.constructs.road.RoadId*

geom

The geometry of this road

Type shapely.geometry.linestring.LineString

origin_junction_id

The unique identifier of the origin junction of this road

destination_junction_id

The unique identifier of the destination junction of this road

metadata

an optional dictionary for storing additional metadata

Type Optional[dict]

to_dict() → Dict[str, Any]

Convert the road to a dictionary

to_flat_dict() → Dict[str, Any]

Convert the road to a flat dictionary

class RoadId(*start*, *end*, *key*)

Trace

class Trace(*frame: geopandas.geodataframe.GeoDataFrame*)

A Trace is a collection of coordinates that represents a trajectory to be matched.

coords

A list of all the coordinates

crs

The CRS of the trace

index

The index of the trace

property coords: List[*mappymatch.constructs.coordinate.Coordinate*]

Get coordinates as Coordinate objects.

property crs: pyproj.crs.crs.CRS

Get Coordinate Reference System(CRS) to underlying GeoDataFrame.

downsample(*npoints: int*) → *mappymatch.constructs.trace.Trace*

Downsample the trace to a given number of points

Parameters **npoints** – the number of points to downsample to

Returns The downsampled trace

drop(*index=typing.List*) → *mappymatch.constructs.trace.Trace*

Remove points from the trace specified by the index parameter

Parameters **index** – the index of the points to drop (0 based index)

Returns The trace with the points removed

classmethod from_csv(*file: Union[str, pathlib.Path]*, *xy: bool = True*, *lat_column: str = 'latitude'*, *lon_column: str = 'longitude'*) → *mappymatch.constructs.trace.Trace*

Builds a trace from a csv file.

Expects the file to have latitude / longitude information in the epsg 4326 format

Parameters

- **file** – the csv file
- **xy** – should the trace be projected to epsg 3857?
- **lat_column** – the name of the latitude column
- **lon_column** – the name of the longitude column

Returns The trace built from the csv file

classmethod from_dataframe(*dataframe: pandas.core.frame.DataFrame*, *xy: bool = True*, *lat_column: str = 'latitude'*, *lon_column: str = 'longitude'*) → *mappymatch.constructs.trace.Trace*

Builds a trace from a pandas dataframe

Expects the dataframe to have latitude / longitude information in the epsg 4326 format

Parameters

- **dataframe** – pandas dataframe with _one_ trace
- **xy** – should the trace be projected to epsg 3857?
- **lat_column** – the name of the latitude column
- **lon_column** – the name of the longitude column

Returns The trace built from the pandas dataframe

classmethod from_geo_dataframe(*frame: geopandas.geodataframe.GeoDataFrame*, *xy: bool = True*) → *mappymatch.constructs.trace.Trace*

Builds a trace from a geopandas dataframe

Expects the dataframe to have geometry column

Parameters

- **frame** – geopandas dataframe with _one_ trace
- **xy** – should the trace be projected to epsg 3857?

Returns The trace built from the geopandas dataframe

classmethod from_geojson(*file: Union[str, pathlib.Path]*, *index_property: Optional[str] = None*, *xy: bool = True*)

Reads a trace from a geojson file; If index_property is not specified, this will set any property columns as the index.

Parameters

- **file** – the geojson file
- **index_property** – the name of the property to use as the index
- **xy** – should the trace be projected to epsg 3857?

Returns The trace built from the geojson file

classmethod from_gpx(*file: Union[str, pathlib.Path]*, *xy: bool = True*) → *mappymatch.constructs.trace.Trace*

Builds a trace from a gpx file.

Expects the file to have simple gpx structure: a sequence of lat, lon pairs

Parameters

- **file** – the gpx file
- **xy** – should the trace be projected to epsg 3857?

Returns The trace built from the gpx file

classmethod from_parquet(*file: Union[str, pathlib.Path]*, *xy: bool = True*)

Read a trace from a parquet file

Parameters

- **file** – the parquet file
- **xy** – should the trace be projected to epsg 3857?

Returns The trace built from the parquet file

property index: pandas.core.indexes.base.Index

Get index to underlying GeoDataFrame.

to_crs(*new_crs: pyproj.crs.crs.CRS*) → *mappymatch.constructs.trace.Trace*

Converts the crs of a trace to a new crs

Parameters **new_crs** – the new crs to convert to

Returns A new trace with the new crs

to_geojson(*file: Union[str, pathlib.Path]*)

Write the trace to a geojson file

Parameters **file** – the file to write to

2.4.2 Maps

The type of map used to display data can be changed based on what works best for you application. Each map class must conform to the map interface defined in *mappymatch.maps.map_interface.MapInterface*.

Available Maps

For readability, the full path is listed below and not used in other parts of the documentation.

- *mappymatch.maps.nx.nx_map.NxMap*

NxMap

class NxMap(*graph: networkx.classes.multidigraph.MultiDiGraph*)

A road map that uses a networkx graph to represent its roads.

g

The networkx graph that represents the road map

crs

The coordinate reference system of the map

property distance_weight: str

Get the distance weight

Returns The distance weight

classmethod from_dict(*d: Dict[str, Any]*) → *mappymatch.maps.nx.nx_map.NxMap*

Build a NxMap instance from a dictionary

classmethod from_file(*file: Union[str, pathlib.Path]*) → *mappymatch.maps.nx.nx_map.NxMap*

Build a NxMap instance from a file

Parameters **file** – The graph pickle file to load the graph from

Returns A NxMap instance

classmethod from_geofence(*geofence:* mappymatch.constructs.geofence.Geofence, *xy: bool = True*, *network_type:* mappymatch.maps.nx.readers.osm_readers.NetworkType = *NetworkType.DRIVE*) → *mappymatch.maps.nx.nx_map.NxMap*

Read an OSM network graph into a NxMap

Parameters

- **geofence** – the geofence to clip the graph to
- **xy** – whether to use xy coordinates or lat/lon
- **network_type** – the network type to use for the graph

Returns a NxMap

nearest_road(*coord:* mappymatch.constructs.coordinate.Coordinate, *buffer: float = 10.0*) → *mappymatch.constructs.road.Road*

A helper function to get the nearest road.

Parameters

- **coord** – The coordinate to find the nearest road to
- **buffer** – The buffer to search around the coordinate

Returns The nearest road to the coordinate

road_by_id(*road_id:* mappymatch.constructs.road.RoadId) → Optional[*mappymatch.constructs.road.Road*]

Get a road by its id

Parameters **road_id** – The id of the road to get

Returns The road with the given id, or None if it does not exist

property roads: List[*mappymatch.constructs.road.Road*]

Get a list of all the roads in the map

Returns A list of all the roads in the map

set_road_attributes(*attributes: Dict[*mappymatch.constructs.road.RoadId*, Dict[str, Any]]*)

Set the attributes of the roads in the map

Parameters **attributes** – A dictionary mapping road ids to dictionaries of attributes

Returns None

shortest_path(*origin:* mappymatch.constructs.coordinate.Coordinate, *destination:* mappymatch.constructs.coordinate.Coordinate, *weight: Optional[Union[str, Callable]] = None*) → List[*mappymatch.constructs.road.Road*]

Computes the shortest path between an origin and a destination

Parameters

- **origin** – The origin coordinate
- **destination** – The destination coordinate
- **weight** – The weight to use for the path, either a string or a function

Returns A list of roads that form the shortest path

property time_weight: str

Get the time weight

Returns The time weight

to_dict() → Dict[str, Any]

Convert the map to a dictionary

to_file(*outfile: Union[str, pathlib.Path]*)

Save the graph to a pickle file

Parameters **outfile** – The file to save the graph to

Utility Functions

Map interface

Each Matcher class must conform to the matcher interface defined below.

class MapInterface

Abstract base class for a Matcher

abstract property distance_weight: str

Get the distance weight

Returns The distance weight

abstract nearest_road(*coord:* mappymatch.constructs.coordinate.Coordinate) → *mappymatch.constructs.road.Road*

Return the nearest road to a coordinate

Parameters **coord** – The coordinate to find the nearest road to

Returns The nearest road to the coordinate

abstract road_by_id(*road_id:* mappymatch.constructs.road.RoadId) → Optional[*mappymatch.constructs.road.Road*]

Get a road by its id

Parameters **road_id** – The id of the road to get

Returns The road with the given id or None if it does not exist

abstract property roads: List[*mappymatch.constructs.road.Road*]

Get a list of all the roads in the map

Returns A list of all the roads in the map

abstract shortest_path(*origin:* mappymatch.constructs.coordinate.Coordinate, *destination:* mappymatch.constructs.coordinate.Coordinate, *weight: Optional[Union[str, Callable]] = None*) → List[*mappymatch.constructs.road.Road*]

Computes the shortest path on the road network

Parameters

- **origin** – The origin coordinate
- **destination** – The destination coordinate
- **weight** – The weight to use for the path

Returns A list of roads that form the shortest path

abstract property time_weight: str

Get the time weight

Returns The time weight

2.4.3 Matchers

There are several matchers that can be used to determine the road that a coordinate is *matched* to. Matchers may have tradeoffs between speed and accuracy.

Available Matchers

For readability, the full path is listed below and not used in other parts of the documentation.

- *mappymatch.matchers.lcss.lcss.LCSSMatcher*
- *mappymatch.matchers.line_snap.LineSnapMatcher*
- *mappymatch.matchers.valhalla.ValhallaMatcher*
- *mappymatch.matchers.osrm.OsrmMatcher*

LCSS

class LCSSMatcher(*road_map:* mappymatch.maps.map_interface.MapInterface, *distance_epsilon: float = 50.0*, *similarity_cutoff: float = 0.9*, *cutting_threshold: float = 10.0*, *random_cuts: int = 0*, *distance_threshold: float = 10000*)

A map matcher based on the paper:

Zhu, Lei, Jacob R. Holden, and Jeffrey D. Gonder. "Trajectory Segmentation Map-Matching Approach for Large-Scale, High-Resolution GPS Data." Transportation Research Record: Journal of the Transportation Research Board 2645 (2017): 67-75.

Parameters

- **road_map** – The road map to use for matching
- **distance_epsilon** – The distance epsilon to use for matching (default: 50 meters)
- **similarity_cutoff** – The similarity cutoff to use for stopping the algorithm (default: 0.9)
- **cutting_threshold** – The distance threshold to use for computing cutting points (default: 10 meters)
- **random_cuts** – The number of random cuts to add at each iteration (default: 0)
- **distance_threshold** – The distance threshold above which no match is made (default: 10000 meters)

match_trace(*trace:* mappymatch.constructs.trace.Trace) → mappymatch.matchers.match_result.MatchResult

Take in a trace of gps points and return a list of matching link ids

Parameters **trace** – The trace to match

Returns A list of Match objects

match_trace_batch(*trace_batch: List[*mappymatch.constructs.trace.Trace*]*, *processes: int = 1*) → List[mappymatch.matchers.match_result.MatchResult]

Take in a batch of traces and return a batch of matching link ids

Parameters **trace_batch** – The batch of traces to match

Returns A batch of Match objects

Line Snap

class LineSnapMatcher(*road_map:* mappymatch.maps.map_interface.MapInterface)

A crude (but fast) map matcher that just snaps points to the nearest road network link.

map

The map to match against

match_trace(*trace:* mappymatch.constructs.trace.Trace) → mappymatch.matchers.match_result.MatchResult

Take in a trace of gps points and return a list of matching link ids

Parameters **trace** – The trace to match

Returns A list of Match objects

match_trace_batch(*trace_batch: List[*mappymatch.constructs.trace.Trace*]*) → List[mappymatch.matchers.match_result.MatchResult]

Take in a batch of traces and return a batch of matching link ids

Parameters **trace_batch** – The batch of traces to match

Returns A batch of Match objects

Valhalla

class ValhallaMatcher(*valhalla_url='https://valhalla1.openstreetmap.de/trace_attributes'*, *cost_model='auto'*, *shape_match='map_snap'*, *attributes={'edge.length', 'edge.speed'}*)

pings a Valhalla server for map matching

match_trace(*trace:* mappymatch.constructs.trace.Trace) → mappymatch.matchers.match_result.MatchResult

Take in a trace of gps points and return a list of matching link ids

Parameters **trace** – The trace to match

Returns A list of Match objects

match_trace_batch(*trace_batch: list[*Trace*]*) → list[MatchResult]

Take in a batch of traces and return a batch of matching link ids

Parameters **trace_batch** – The batch of traces to match

Returns A batch of Match objects

Osrm

class OsrmMatcher(*osrm_address='http://router.project-osrm.org'*, *osrm_profile='driving'*, *osrm_version='v1'*)

pings an OSRM server for map matching

match_trace(*trace:* mappymatch.constructs.trace.Trace) → mappymatch.matchers.match_result.MatchResult

Take in a trace of gps points and return a list of matching link ids

Parameters **trace** – The trace to match

Returns A list of Match objects

match_trace_batch(*trace_batch: list[*Trace*]*) → list[MatchResult]

Take in a batch of traces and return a batch of matching link ids

Parameters **trace_batch** – The batch of traces to match

Returns A batch of Match objects

Matcher Inferface

Each Matcher class must conform to the matcher interface defined below.

class MatcherInterface

Abstract base class for a Matcher

abstract match_trace(*trace:* mappymatch.constructs.trace.Trace) → mappymatch.matchers.match_result.MatchResult

Take in a trace of gps points and return a list of matching link ids

Parameters **trace** – The trace to match

Returns A list of Match objects

abstract match_trace_batch(*trace_batch: List[*mappymatch.constructs.trace.Trace*]*) → List[mappymatch.matchers.match_result.MatchResult]

Take in a batch of traces and return a batch of matching link ids

Parameters **trace_batch** – The batch of traces to match

Returns A batch of Match objects

2.4.4 Readers

(Reserved for future use)

You can use these to generate a map for various sources.

Readers Module

class NetworkType(*value*)

Enumerator for Network Types supported by osmnx.

compress(*g*) → networkx.classes.multidigraph.MultiDiGraph

a hacky way to delete unnecessary data on the networkx graph

Parameters **g** – the networkx graph to compress

Returns the compressed networkx graph

nx_graph_from_osmnx(*geofence:* mappymatch.constructs.geofence.Geofence, *network_type:* mappymatch.maps.nx.readers.osm_readers.NetworkType, *xy: bool = True*) → networkx.classes.multidigraph.MultiDiGraph

Build a networkx graph from OSM data

Parameters

- **geofence** – the geofence to clip the graph to
- **network_type** – the network type to use for the graph
- **xy** – whether to use xy coordinates or lat/lon

Returns a networkx graph of the OSM network

parse_osmnx_graph(*graph: networkx.classes.multidigraph.MultiDiGraph*, *network_type:* mappymatch.maps.nx.readers.osm_readers.NetworkType, *xy: bool = True*) → networkx.classes.multidigraph.MultiDiGraph

Parse the raw osmnx graph into a graph that we can use with our NxMap

Parameters

- **geofence** – the geofence to clip the graph to
- **xy** – whether to use xy coordinates or lat/lon
- **network_type** – the network type to use for the graph

Returns a cleaned networkx graph of the OSM network

2.4.5 Utils

Helper functions and classes.

Modules

For readability, the full path is listed below and not used in other parts of the documentation.

- mappymatch.utils.exceptions
- mappymatch.utils.geo
- mappymatch.utils.plot
- mappymatch.utils.trace
- mappymatch.utils.url

exceptions module

exception MapException

An exception for any errors that occur with the MapInterface

geo module

coord_to_coord_dist(*a:* mappymatch.constructs.coordinate.Coordinate, *b:* mappymatch.constructs.coordinate.Coordinate) → float

Compute the distance between two coordinates.

Parameters

- **a** – The first coordinate
- **b** – The second coordinate

Returns The distance in meters

latlon_to_xy(*lat: float*, *lon: float*) → Tuple[float, float]

Tramsform lat,lon coordinates to x and y.

Parameters

- **lat** – The latitude.

- **lon** – The longitude.

Returns Transformed x and y as x, y.

xy_to_latlon(*x: float*, *y: float*) → Tuple[float, float]

Tramsform x,y coordinates to lat and lon

Parameters

- **x** –

X.

- **y** –

Y.

Returns Transformed lat and lon as lat, lon.

plot module

plot_geofence(*geofence:* mappymatch.constructs.geofence.Geofence, *m: Optional[folium.folium.Map] = None*)

Plot geofence.

Parameters

- **geofence** – The geofence to plot
- **m** – the folium map to plot on

Returns The updated folium map with the geofence.

plot_map(*tmap:* mappymatch.maps.nx.nx_map.NxMap, *m: Optional[folium.folium.Map] = None*)

Plot the roads on an NxMap.

Parameters

- **tmap** – The Nxmap to plot.
- **m** – the folium map to add to

Returns The folium map with the roads plotted.

plot_match_distances(*matches: mappymatch.matchers.match_result.MatchResult*)

Plot the points deviance from known roads with matplotlib.

Parameters **matches** (*MatchResult*) – The coordinates of guessed points in the area in the form of a MatchResult object.

plot_matches(*matches: typing.Union[mappymatch.matchers.match_result.MatchResult, typing.List[mappymatch.constructs.match.Match]], crs=<Projected CRS: EPSG:3857> Name: WGS 84 / Pseudo-Mercator Axis Info [cartesian]: - X[east]: Easting (metre) - Y[north]: Northing (metre) Area of Use: - name: World between 85.06°S and 85.06°N. - bounds: (-180.0, -85.06, 180.0, 85.06) Coordinate Operation: - name: Popular Visualisation Pseudo-Mercator - method: Popular Visualisation Pseudo Mercator Datum: World Geodetic System 1984 ensemble - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*)

Plots a trace and the relevant matches on a folium map.

Args: matches: A list of matches or a MatchResult. crs: what crs to plot in. Defaults to XY_CRS.

Returns A folium map with trace and matches plotted.

plot_path(*path: List[mappymatch.constructs.road.Road]*, *crs: pyproj.crs.crs.CRS*, *m: Optional[folium.folium.Map] = None*, *line_color='red'*, *line_weight=10*, *line_opacity=0.8*)

Plot a list of roads.

Parameters

- **path** – The path to plot.
- **crs** – The crs of the path.
- **m** – The folium map to add to.
- **line_color** – The color of the line.
- **line_weight** – The weight of the line.
- **line_opacity** – The opacity of the line.

plot_trace(*trace: mappymatch.constructs.trace.Trace*, *m: Optional[folium.folium.Map] = None*, *point_color: str = 'black'*, *line_color: Optional[str] = 'green'*)

Plot a trace.

Parameters

- **trace** – The trace.
- **m** – the folium map to plot on
- **point_color** – The color the points will be plotted in.
- **line_color** – The color for lines. If None, no lines will be plotted.

Returns An updated folium map with a plot of trace.

trace module

remove_bad_start_from_trace(*trace: mappymatch.constructs.trace.Trace*, *distance_threshold: float*) → *mappymatch.constructs.trace.Trace*

Remove points at the beginning of a trace if it is a gap is too big.

Too big is defined by distance threshold.

Parameters

- **trace** – The trace.
- **distance_threshold** – The distance threshold.

Returns The new trace.

split_large_trace(*trace: Trace*, *ideal_size: int*) → list[*Trace*]

Split up a trace into a list of smaller traces.

Parameters

- **trace** – the trace to split.
- **ideal_size** – the target number of coordinates for each new trace.

Returns A list of split traces.

url module

multiurljoin(*urls: List[str]*) → str

Make a url from uri's.

Parameters **urls** – list of uri

Returns Url as uri/uri/…

2.5 Contributing

Overview

- *Pre-Commit*

2.5.1 Getting Started

We are happy your are even considering helping. We hope this guide makes the process seamless. We really appreciate your help!

If you are new to open source contribution, please checkout *Best practices* and *Tools in our toolbelt*.

Note: All command line code is run from the project root except where noted.

Tip: In `git clone <forked_repo_https_link.git>` , text inside the < >, and the < > get replaced with the appropriate specific text. The < > visually separate the syntax which remains unchanged from the text that needs to be changed. It is a common development pattern that is confusing if you have not seen it before. To see another example of it, type `git --help` on your command line.

2.5.2 Select an issue

Look through the open issues and for an issue labelled `good first issue` or `documentation` that is unassigned.

Once you have selected an issue, then assign it to yourself by placing the word `take` as a comment. This will indicate to other contributors that you're working on the issue. Your goal is to finish the process from self assigning the issue to submitting the initial Pull Request (PR) in less than 2 weeks. This keeps your work from diverging too much from the main branch.

That being said, life happens, we appreciate you however you decide to help. If something comes up, and you need to unassign an issue (place the word `remove` as a comment) or post that you are still working on it, then that is okay.

2.5.3 Build from the source (First time only)

1. Go to the repo and click on *Fork* in the upper right.
2. Navigate to your fork which will be <you_user_name>/mappymatch . Click on the green *Code* drop down and copy the https link.
3. From the command line:

Listing 1: Clone the forked repo.

```
git clone <forked_repo_https_link.git>
```

Listing 2: Create the Conda environment, and activate it. You will need to run each command separately.

```
cd mappymatch
conda env create -f environment_dev.yml
conda activate mappymatch_dev
```

Listing 3: Verify installation by running tests.

```
python -m unittest discover
```

Listing 4: Return should look like this, but the number of tests will vary.

```
.............................................
----------------------------------------------------------------
Ran 51 tests in 14.621s

OK
```

2.5.4 Install Pre-Commit hooks (First time only)

```
pre-commit install
```

2.5.5 Setup Git workflow (First time only)

Listing 5: Setup upstream remote.

```
git remote add upstream https://github.com/NREL/mappymatch.git
```

2.5.6 Execute Changes and Git workflow

Listing 6: Checkout a branch from your forked repository

1. `git checkout -b <descriptive_branch_name>`
2. Make your changes and add commits
3. Pull in changes from upstream. This is best done periodically, if you have the branch checked out for a long time.

Listing 7: Switch to main branch, pull changes from upstream, resolve conflicts that arise.

```
git checkout main
git pull upstream main
```

Listing 8: Switch to your branch, pull the changes from your main repository, and resolve conflicts that arise.

```
git checkout <descriptive_branch_name>
git pull origin main
```

4. Push changes to get ready for PR.

Listing 9: Push your changes to remote for your forked repository.

```
git push origin <descriptive_branch_name>
```

2.5.7 Open a PR

1. Go to the repo/PR and click on *New pull request* in the upper right.
2. Click on *Compare across forks* in the top middle.
3. Leave the `base repository` section alone. For the `head repository` section select your fork and your branch.
4. **Review the code diffs** and then click *Create pull request*.
5. Check back after a fewer minutes to make sure the CI steps pass. If they fail, then make the fixes and push your branch to your forked repo again. The PR will update and rerun the CI.

2.5.8 Finish the PR

1. Check back in a few days for maintainer requests for changes. Don't be surprised or offended by the changes. Most PRs require some changes.
2. Make the changes and push your branch to your forked repo again.
3. The maintainer will merge your branch.
4. Delete you branch
5. Pull the changes into your forked repo.

```
git checkout main
git pull upstream main
```

2.5.9 Best practices

Asking questions

Have a question? Rather than opening an issue, please ask questions or post comments in Q&A Discussions . Members of the community are happy to assist.

Providing feedback

Your comments and feedback are very welcome. Please post to General Discussions with lots of detail.

Reporting issues

We are happy to fix bugs. Please report buys using the issues template.

General issue reporting guidelines

- One issue per problem.
- Check through the closed issues before submitting a new one.

Requesting features

If you are interested in coding or requesting a new feature, let us know in Ideas Discussions Please wait for confirmation from a core maintainer before proceeding.

2.5.10 Previewing documentation locally

To preview the documentation locally:

1. From the command line, use Sphinx to rebuild the docs.

```
sphinx-autobuild -b html ./docs/source ./docs/_build
```

2. Open `http://127.0.0.1:8000` with your browser.

2.5.11 Maintainer Information

Adding a new package dependency

To add a new package dependency, add it to the `dependency` list in the `/pyproject.toml` file.

Updating Version Locations

To update the version automatically using tbump:

```
tbump <version_major.version_minor.version_patch> --only-patch
```

To update the version manually update it in the following locations:

1. In the docs `/docs/source/conf.py`
2. In the setup.py `/pyproject.toml` (2 places)

Releasing to PyPi manually

1. Build the wheel

```
python -m build
```

2. Upload to Test PyPi.

```
twine upload -r testpypi dist/* --verbose
```

3. Verify for typos and that the wheel installs. If you spot a mistake, correct it, commit the correction and change the version in the [project] table in `/pyproject.toml` to <major>.<minor>.<patch>.post<#>
4. Delete the old wheel, rebuild the wheel and reupload to Test PyPi.
5. Remove post<#> from the version.
6. If there are no mistakes, add add tag v<major>.<minor>.<patch> to master branch on github.

Listing 10: Tags have to pushed like a separate branch.

```
git tag -a v<major>.<minor>.<patch> -m "version <major>.<minor>.<patch>"
git push origin <tagname>
```

7. Upload to PyPi.

```
twine upload dist/*
```

2.5.12 Tools in our toolbelt

Note: All command line examples use settings configured for the repo. Coverage and Isort automatically find their configuration files.

Black

Implemented as a Pre-Commit hook.

Black is an opinionated code formatter so you don't have to be.

Command line use:

```
black --config pyproject.toml
```

Coverage

Not Implemented as CI

Coverage is a tool used to monitor test coverage. It does so by executing the tests and monitoring which lines are run.

Command line use:

Listing 11: Run the tests with coverage monitoring.

```
coverage -m unittest discover
```

Listing 12: View the coverage report.

```
coverage report -m
```

Flake8

Implemented as CI and Pre-Commit hook.

Flake8 is a linting tool that wraps PyFlakes, pycodestyle, and Ned Batchelder's McCabe script into a single package.

Command Line use:

```
flake8 --config tox.ini
```

Interrogate

Implemented as Pre-Commit hook.

Interrogate reports on the level of and enforces docstring coverage for the code base.

Command line use

```
interrogate -c pyproject.toml
```

Isort

Implemented as Pre-Commit hook.

Isort automatically groups and sorts your import statements so you don't have to.

Command line use:

```
isort
```

Mypy

Implemented in CI.

Mypy is an optional static type checker for Python that aims to combine the benefits of dynamic (or "duck") typing and static typing.

Command Line use:

Listing 13: Run normally.

```
mypy --config-file mypy.ini
```

Listing 14: Run in verbose mode. Used to see location of specific failures.

```
mypy --config-file mypy.ini -v
```

Pre-Commit

Implements all the precommit hooks.

Pre-Commit is a framework for managing and maintaining multilanguage pre-commit hooks. Before the commit executes, pre-commit hooks are run to do useful things like code formatting. This means the unformatted code never enters your code base.

Command line use:

Listing 15: Run once to install hooks configured by .pre-commit-config.yaml

```
pre-commit install
```

Listing 16: Make changes to the code base, add files to the staging area, and commit changes as you normally would.

```
git commit -m "Updated tools in toolchain docs section."
```

You will get a success or failure.

Listing 17: Example output for success. No other steps are needed.

```
black....................................................(no files to check)Skipped
isort (python)...........................................(no files to check)Skipped
interrogate..............................................(no files to check)Skipped
[create_contributing_docs 30c2bf3] Updated tools in toolchain docs section.
1 file changed, 80 insertions(+), 4 deletions(-)
```

Listing 18: Example output for failure. See next code block for follow on steps.

```
black....................................................................Failed
- hook id: black
- files were modified by this hook

reformatted mappymatch\utils\url.py
```

(continues on next page)

(continued from previous page)

```
All done! \u2728 \U0001f370 \u2728
1 file reformatted.

isort (python)..........................................................Passed
interrogate.............................................................Passed
```

Listing 19: Re-add the files to the staging area. Commit again. You should get a success.

```
git add --all
git commit -m "Update contributing docs for precommit-failure."
```

2.6 Changelog

2.7 Contributors

As with any open source project, many people came together to create this library.

2.7.1 Owners

- nreinicke
- jhoshiko

2.7.2 Core maintainers

- machallboyd

2.7.3 Previous maintainers or Owners

- Rowlando13

2.7.4 Special thanks

- The many contributors from Pycon US 2022! You really got the project off the ground.
- The National Renewable Energy Laboratory for open sourcing the initial groundwork.
- Open source maintainers even if you only contributed one commit.

Previous work:

- Lei Zhu, Jacob R. Holden, and Jeffrey D. Gonder for their work on the algorithm that underlies the `LCSSMatcher` detailed in Trajectory Segmentation Map-Matching Approach for Large-Scale, High-Resolution GPS Data

PYTHON MODULE INDEX

m

INDEX

N

O

P

R

S

T

V

X

Y