Package ‘geoops’

Type Package

Title 'GeoJSON' Topology Calculations and Operations

Description Tools for doing calculations and manipulations on 'GeoJSON', a 'geospatial' data interchange format (<https://tools.ietf.org/html/rfc7946>). 'GeoJSON' is also valid 'JSON'.

Version 0.3.0

License MIT + file LICENSE

URL https://docs.ropensci.org/geoops,
    https://github.com/ropensci/geoops

BugReports https://github.com/ropensci/geoops/issues

Imports Rcpp (>= 0.12.12)

Suggests roxygen2 (>= 7.1.0), testthat, knitr, rmarkdown, jsonlite

LinkingTo Rcpp

SystemRequirements C++11

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.0

X-schema.org-applicationCategory Geospatial

X-schema.org-keywords geojson, geospatial, conversion, data, bbox, coordinates, distance, bearing

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation yes

Author Scott Chamberlain [aut, cre],
Niels Lohmann [cph]

Maintainer Scott Chamberlain <myrmecocystus+r@gmail.com>

Repository CRAN

Date/Publication 2020-05-17 00:40:03 UTC
Tools for doing calculations and manipulations on GeoJSON, a 'geospatial' data interchange format (https://tools.ietf.org/html/rfc7946). GeoJSON is also valid JSON.

Author(s)

Scott Chamberlain <myrmecocystus+r@gmail.com>
Examples

```r
library("geoops")

# Calculate distance between two GeoJSON points
pt1 <- '{
  "type": "Feature",
  "properties": {
    "marker-color": "#f00"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [-75.343, 39.984]
  }
}
pt2 <- '{
  "type": "Feature",
  "properties": {
    "marker-color": "#0f0"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [-75.534, 39.123]
  }
}
geo_distance(pt1, pt2)
```

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature</th>
</tr>
</thead>
</table>

Description

A GeoJSON object with the type "Feature" is a feature object:

- A feature object must have a member with the name "geometry". The value of the geometry member is a geometry object as defined above or a JSON null value.
- A feature object must have a member with the name "properties". The value of the properties member is an object (any JSON object or a JSON null value).
- If a feature has a commonly used identifier, that identifier should be included as a member of the feature object with the name "id".

See Also

Other geo types: `FeatureCollection, GeometryCollection, LineString, MultiLineString, MultiPoint, MultiPolygon, Point, Polygon, geojson-types`
Examples

```
{
    "type": "Feature",
    "properties": {
        "population": 200
    },
    "geometry": {
        "type": "Point",
        "coordinates": [10.724029, 59.926807]
    }
}
```

Description

A GeoJSON object with the type "FeatureCollection" is a feature collection object. An object of type "FeatureCollection" must have a member with the name "features". The value corresponding to "features" is an array. Each element in the array is a feature object as defined above.

See Also

Other geo types: Feature, GeometryCollection, LineString, MultiLineString, MultiPoint, MultiPolygon, Point, Polygon, geojson-types

Examples

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "properties": {
                "population": 200
            },
            "geometry": {
                "type": "Point",
                "coordinates": [-112.0372, 46.608058]
            }
        }
    ]
}
```
Description

Description of GeoJSON data types

GeoJSON object

GeoJSON always consists of a single object. This object (referred to as the GeoJSON object below) represents a geometry, feature, or collection of features.

- The GeoJSON object may have any number of members (name/value pairs).
- The GeoJSON object must have a member with the name "type". This member's value is a string that determines the type of the GeoJSON object.
- The value of the type member must be one of: "Point", "MultiPoint", "LineString", "MultiLineString", "Polygon", "MultiPolygon", "GeometryCollection", "Feature", or "FeatureCollection". The case of the type member values must be as shown here.
- A GeoJSON object may have an optional "crs" member, the value of which must be a coordinate reference system object (see 3. Coordinate Reference System Objects).
- A GeoJSON object may have a "bbox" member, the value of which must be a bounding box array (see 4. Bounding Boxes).

See Also

Other geo types: FeatureCollection, Feature, GeometryCollection, LineString, MultiLineString, MultiPoint, MultiPolygon, Point, Polygon

GeometryCollection

Description

Each element in the geometries array of a GeometryCollection is one of the geometry objects described above.

See Also

Other geo types: FeatureCollection, Feature, LineString, MultiLineString, MultiPoint, MultiPolygon, Point, Polygon, geojson-types
geo_along

Examples

`{
    "type": "GeometryCollection",
    "geometries": [
        {
            "type": "Point",
            "coordinates": [100.0, 0.0]
        },
        {
            "type": "LineString",
            "coordinates": [[101.0, 0.0], [102.0, 1.0]]
        }
    ]
}`

geo_along

Takes a LineString and returns a Point at a specified distance along the line.

Description

Takes a LineString and returns a Point at a specified distance along the line.

Usage

geo_along(line, distance, units)

Arguments

line (character) a Feature<LineString>
distance (numeric) distance along the line
units (character) can be degrees, radians, miles, or kilometers (default)

Value

Feature<Point> distance (at X units) along the line

Examples

line <- `{
    "type": "Feature",
    "properties": {},
    "geometry": {
        "type": "LineString",
        "coordinates": [
            [-77.031669, 38.878605],
            [-77.029609, 38.881946],
            [-77.020339, 38.884084],
            [-77.025661, 38.885821],
            [-77.021884, 38.889563],
            [-77.018957, 38.893376],
            [-77.020339, 38.896194],
            [-77.021884, 38.898031],
            [-77.023428, 38.899869],
            [-77.025061, 38.901706],
            [-77.026694, 38.903543],
            [-77.028328, 38.905380],
            [-77.030061, 38.907217],
            [-77.031669, 38.909054],
            [-77.033272, 38.910891],
            [-77.034905, 38.912728]
        ]
    }
}`


geo_area

Takes one or more features and returns their area in square meters.

Description

Takes one or more features and returns their area in square meters.

Usage

geo_area(x)

Arguments

x (character) a Feature or FeatureCollection

Value

(numeric) area in square meters

Examples

polygons <- '{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Polygon",
        "coordinates": [[
          [-67.031021, 10.458102],
          [-67.031021, 10.53372],
          [-66.929397, 10.53372],
          [-66.929397, 10.458102],
          [-67.031021, 10.458102]
        ]]
      }
    },
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Polygon",
        "coordinates": [[
          [-67.031021, 10.458102],
          [-67.031021, 10.53372],
          [-66.929397, 10.53372],
          [-66.929397, 10.458102],
          [-67.031021, 10.458102]
        ]]
      }
    }
  ]
}'

geo_along(line, 10, 'kilometers')
geo_bearing

Calculate bearing

Description
Calculate bearing

Usage
geo_bearing(point1, point2)

geo_bbox_polygon  BBOX polygon

Description
Takes a bbox and returns an equivalent Feature<Polygon>

Usage
geo_bbox_polygon(bbox)

Arguments
bbox extent in [minX, minY, maxX, maxY] order

Value
Feature<Polygon> a Polygon representation of the bounding box

Examples
geo_bbox_polygon(c(0, 0, 10, 10))
geo_bbox_polygon(c(-90, -30, -70, -10))
geo_bbox_polygon(c(0, 0, 10, 10))
Arguments

point1 start geojson Feature<Point>
point2 end geojson Feature<Point>

Value

(number) bearing in decimal degrees

Examples

point1 <- '{
  "type": "Feature",
  "properties": {
    "marker-color": "#f00"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [-75.343, 39.984]
  }
}'

point2 <- '{
  "type": "Feature",
  "properties": {
    "marker-color": "#0f0"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [-75.534, 39.123]
  }
}'

geo_bearing(point1, point2)

geo_destination

Calculate a destination

Description

Calculate a destination

Usage

geo_destination(from, distance, bearing, units = "kilometers")
**geo_distance**

**Arguments**

- **from** (character) from starting Feature<Point>
- **distance** (numeric) distance from the starting Feature<Point>
- **bearing** (numeric) ranging from -180 to 180
- **units** (character) miles, kilometers, degrees, or radians

**Value**

(character) destination Feature<Point>

**Examples**

```r
point <- '{
  "type": "Feature",
  "properties": {
    "marker-color": "#0f0"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [-75.343, 39.984]
  }
}'
geo_destination(point, 50, 90, 'miles')
geo_destination(point, 200, 90)
```

---

**geo_distance**

*Calculate distance between two GeoJSON points*

**Description**

Calculate distance between two GeoJSON points

**Usage**

```r
geo_distance(from, to, units = "kilometers")
```

**Arguments**

- **from** Origin Feature<Point>
- **to** Destination Feature<Point>
- **units** (character) Can be degrees, radians, miles, or kilometers (default)

**Value**

the distance, a single numeric value, in units given in units parameter
Examples

```r
point1 <- '{
  "type": "Feature",
  "properties": {
    "marker-color": "#f00"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [-75.343, 39.984]
  }
}'

point2 <- '{
  "type": "Feature",
  "properties": {
    "marker-color": "#0f0"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [-75.534, 39.123]
  }
}'

geo_distance(point1, point2)
geo_distance(point1, point2, units = "miles")
geo_distance(point1, point2, units = "degrees")
geo_distance(point1, point2, units = "radians")

pt1 <- '{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": 
      [0.5, 
        0.5
      ]
  }
}'

pt2 <- '{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [2, 2]
  }
}'

geo_distance(pt1, pt2, units = "miles")
```

geo_get_coords  Get coordinates
**Description**

Get coordinates

**Usage**

`geo_get_coords(x)`

**Arguments**

- `x` geojson string

**Details**

You can use jsonlite to convert the output to R objects

**Value**

a character vector with coordinates as JSON

**Examples**

```r
## Not run:
x <- '{"type": "Feature", "geometry": {"type": "Point","coordinates": [1, 2]},"properties": {}}'
geo_get_coords(x)
x <- '{"type": "Point", "coordinates": [1, 2]}'
geo_get_coords(x)
x <- '[0, 5]'  
geo_get_coords(x)
## End(Not run)
```

---

**geo_line_distance**

Calculate length of GeoJSON LineString or Polygon

**Description**

FIXME: doesn’t support FeatureCollection’s yet - fix c++ code

**Usage**

`geo_line_distance(line, units = "kilometers")`

**Arguments**

- `line` a LineString to measure
- `units` (character) Can be degrees, radians, miles, or kilometers (default)
Value

Single numeric value

Examples

```r
# LineString
line <- ' {
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "LineString",
    "coordinates": [
      [-77.031669, 38.878605],
      [-77.029609, 38.881946],
      [-77.020339, 38.884084],
      [-77.025661, 38.885821],
      [-77.021884, 38.889563],
      [-77.019824, 38.892368]
    ]
  }
}
'
geo_line_distance(line)
geo_line_distance(line, units = "miles")
geo_line_distance(line, units = "degrees")
geo_line_distance(line, units = "radians")

# Polygon
x <- ' {
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Polygon",
    "coordinates": [[-67.031021,10.458102],
                    [-67.031021,10.53372],
                    [-66.929397,10.53372],
                    [-66.929397,10.458102],[[-67.031021,10.458102]]]
  }
}
geo_line_distance(x)

# MultiPolygon
x <- ' {
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "MultiPolygon",
    "coordinates": [
      [
        [-122.62527465820311,
        37.89327929625019],
      ],
      [-122.60467529296875,
        37.9024908518640995],
    ]
  }
}
geo_line_distance(x)
```
geo_line_distance

[[-122.58682250976562, 37.895988598965644],
 [-122.62527465820311, 37.89327929625019]]
geo_midpoint

37.726194088705576
]
]
]
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
 geo_line_distance(x)

geo_midpoint       Midpoint

Description
Takes two Point’s and returns a point midway between them. The midpoint is calculated geodesically, meaning the curvature of the earth is taken into account.

Usage
geo_midpoint(from, to)

Arguments
from       Feature<Point> first point
to         Feature<Point> second point

Value
Feature<Point> a point midway between from and to

Examples
pt1 <- '{
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [144.834823, -37.771257]
  }
}
pt2 <- '{
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [145.14244, -37.830937]
  }
}
geo_midpoint(pt1, pt2)
**geo_nearest**

*Calculate nearest point to a reference point*

**Description**

Takes a reference **Point** and a FeatureCollection of Features with Point geometries and returns the point from the FeatureCollection closest to the reference. This calculation is geodesic.

**Usage**

`geo_nearest(target_point, points)`

**Arguments**

- **target_point** the reference point **Feature<Point>**
- **points** against input point set **FeatureCollection<Point>**

**Value**

A **Feature<Point>** the closest point in the set to the reference point

**Examples**

```json
point1 <- '{
  "type": "Feature",
  "properties": {
    "marker-color": "#0f0"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [28.965797, 41.010086]
  }
}'

points <- '{
  "type": "FeatureCollection",
  "features": [
    { "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Point",
        "coordinates": [28.973865, 41.011122]
      }
    }, {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Point",
        "coordinates": [28.965797, 41.010086]
      }
    }
  ]
}'
```
Description

Takes a triangular plane as a Polygon and a Point within that triangle and returns the z-value at that point. The Polygon needs to have properties a, b, and c that define the values at its three corners.

Usage

geo_planepoint(point, triangle)

Arguments

point Feature<Point> the Point for which a z-value will be calculated
triangle Feature<Polygon> a Polygon feature with three vertices

Value

(numeric) the z-value for interpolatedPoint

Examples

point <- '{
  "type": "Feature",
  "properties": {}
  "geometry": {
    "type": "Point",
    "coordinates": [-75.3221, 39.529]
  }
}'

triangle <- '{
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [[-75.3221, 39.529], [75.3221, 39.529], [75.3221, 40.529], [-75.3221, 40.529], [-75.3221, 39.529]]
  }
}'

geo_nearest(point1, points)
geo_pointgrid

Description
Takes a bounding box and a cell depth and returns a set of Point's in a grid.

Usage
geo_pointgrid(bbox, cell_size, units = "kilometers")

Arguments
bbox extent in [minX, minY, maxX, maxY] order
cell_size (numeric) the distance across each cell
units (character) used in calculating cellSize, can be degrees, radians, miles, or kilometers (default)

Value
FeatureCollection<Point> grid of points

Examples
extent <- c(-70.823364, -33.553984, -70.473175, -33.302986)
cellSize <- 1
units <- 'miles'
x <- geo_pointgrid(extent, cellSize, units)
x
**geo_trianglegrid**

**Description**

Takes a bounding box and a cell depth and returns a set of `Polygon`'s in a grid.

**Usage**

```r
geo_trianglegrid(bbox, cell_size, units = "kilometers")
```

**Arguments**

- `bbox`: extent in `[minX, minY, maxX, maxY]` order
- `cell_size`: (numeric) the distance across each cell
- `units`: (character) used in calculating `cellSize`, can be degrees, radians, miles, or kilometers (default)

**Value**

- `FeatureCollection<Polygon>` grid of polygons

**Examples**

```r
geo_trianglegrid(c(-77.3876, 38.7198, -76.9482, 39.0277), 3, "miles")
geo_trianglegrid(c(-77.3876, 38.7198, -76.9482, 39.0277), 10, "miles")
geo_trianglegrid(c(-77.3876, 38.7198, -76.9482, 39.0277), 30, "miles")
```

---

**LineString**

**Description**

For type "LineString", the "coordinates" member must be an array of two or more positions. A `LinearRing` is closed `LineString` with 4 or more positions. The first and last positions are equivalent (they represent equivalent points). Though a `LinearRing` is not explicitly represented as a GeoJSON geometry type, it is referred to in the `Polygon` geometry type definition.

**See Also**

Other geo types: `FeatureCollection`, `Feature`, `GeometryCollection`, `MultiLineString`, `MultiPoint`, `MultiPolygon`, `Point`, `Polygon`, `geojson-types`
Examples

```
{
  "type": "LineString",
  "coordinates": [
  [100.0, 0.0],
  [101.0, 1.0]
  ]
}
```

---

**MultiLineString**

**MultiLineString**

Description

For type "MultiLineString", the "coordinates" member must be an array of LineString coordinate arrays.

See Also

Other geo types: FeatureCollection, Feature, GeometryCollection, LineString, MultiPoint, MultiPolygon, Point, Polygon, geojson-types

Examples

```
{
  "type": "MultiLineString",
  "coordinates": [
  [[ -105, 39 ], [ -105, 39 ]],
  [ -105, 39 ], [ -105, 39 ]
  ]
}
```

---

**MultiPoint**

**GeoJSON MultiPoint**

Description

For type "MultiPoint", the "coordinates" member must be an array of positions.

See Also

Other geo types: FeatureCollection, Feature, GeometryCollection, LineString, MultiLineString, MultiPolygon, Point, Polygon, geojson-types
MultiPolygon

Examples

```
'{
    "type": "MultiPoint",
    "coordinates": [
        [100.0, 0.0],
        [101.0, 1.0]
    ]
}
```

Description

For type "MultiPolygon", the "coordinates" member must be an array of Polygon coordinate arrays.

See Also

Other geo types: FeatureCollection, Feature, GeometryCollection, LineString, MultiLineString, MultiPoint, Point, Polygon, geojson-types

Point

GeoJSON Point

Description

For type "Point", the "coordinates" member must be a single position.

See Also

Other geo types: FeatureCollection, Feature, GeometryCollection, LineString, MultiLineString, MultiPoint, MultiPolygon, Polygon, geojson-types
Examples

```
{
  "type": "Point",
  "coordinates": [100.0, 0.0]
}
```

Polygon

Description

For type "Polygon", the "coordinates" member must be an array of LinearRing coordinate arrays. For Polygons with multiple rings, the first must be the exterior ring and any others must be interior rings or holes.

See Also

Other geo types: FeatureCollection, Feature, GeometryCollection, LineString, MultiLineString, MultiPoint, MultiPolygon, Point, geojson-types

Examples

```
{
  "type": "Polygon",
  "coordinates": [
    [
      [100.0, 0.0],
      [101.0, 0.0],
      [101.0, 1.0],
      [100.0, 1.0],
      [100.0, 0.0]
    ]
  ]
}
```

version

Description

Get json library version information

Usage

version()
version

**Value**

JSON as a string for now, will make a list soon

**Examples**

`version()`
Index

*Topic package
   geoops-package, 2

Feature, 3, 4–10, 15–17, 19–22
FeatureCollection, 3, 4, 5, 7, 16, 18–22

geo_along, 6
geo_area, 7
geo_bbox_polygon, 8
geo_bearing, 8
geo_destination, 9
geo_distance, 10
geo_get_coords, 11
geo_line_distance, 12
geo_midpoint, 15
geo_nearest, 16
geo_planepoint, 17
geo_pointgrid, 18
geo_trianglegrid, 19
geojson-types, 5
GeometryCollection, 3–5, 5, 19–22
geoops (geoops-package), 2
geoops-package, 2

LineString, 3–6, 12, 19, 20–22

MultiLineString, 3–5, 19, 20, 21, 22
MultiPoint, 3–5, 19, 20, 21, 22
MultiPolygon, 3–5, 19–21, 21, 22

Point, 3–6, 9, 10, 15–21, 21, 22
Polygon, 3–5, 8, 17, 19–21, 22

version, 22