Package 'geoops'

June 18, 2018

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 0.2.0

License MIT + file LICENSE

URL https://github.com/ropensci/geoops

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

Imports Rcpp (>= 0.12.12)

Suggests roxygen2 (>= 6.0.1), testthat, knitr, rmarkdown, jsonlite

LinkingTo Rcpp

SystemRequirements C++11

VignetteBuilder knitr

RoxygenNote 6.0.1

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 2018-06-18 20:06:31 UTC

R topics documented:

`geoops-package` *geoops*

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.

Author(s)

Scott Chamberlain <myrmecocystus+r@gmail.com>

Examples

```
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)
```

`Feature` *Feature*

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]
  }
}'
```

FeatureCollection *FeatureCollection*

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]
      }
    }
  ]
}'
```

geojson-types *Description of GeoJSON data types*

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 *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`

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],
```

```
        [-77.019824, 38.892368]
      ]
    }
}'

geo_along(line, 10, 'kilometers')
```

---

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": [[
```

```
            [-66.919784, 10.397325],
            [-66.919784, 10.513467],
            [-66.805114, 10.513467],
            [-66.805114, 10.397325],
            [-66.919784, 10.397325]
          ]]
        }
      }
    ]
}'
geo_area(polygons)
```

---

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))
```

---

geo_bearing *Calculate bearing*

Description

Calculate bearing

Usage

```
geo_bearing(point1, point2)
```

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")
```

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

```
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

```
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

```
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

```
## 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

```
# 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,
            37.902490518640995
          ],
          [
```

```
        -122.58682250976562,
        37.895988598965644
      ],
      [
        -122.62527465820311,
        37.89327929625019
      ]
    ]
  ],
  [
    [
      [
        -122.52639770507812,
        37.83473402375478
      ],
      [
        -122.53395080566405,
        37.83690319650768
      ],
      [
        -122.51541137695311,
        37.83473402375478
      ],
      [
        -122.52639770507812,
        37.83473402375478
      ]
    ]
  ],
  [
    [
      [
        -122.44331359863283,
        37.726194088705576
      ],
      [
        -122.47833251953125,
        37.73651223296987
      ],
      [
        -122.43095397949219,
        37.74411415606583
      ],
      [
        -122.40898132324217,
        37.77505678240509
      ],
      [
        -122.4103546142578,
        37.72184917678752
      ],
      [
        -122.44331359863283,
```

```
            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

```
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": [28.948459, 41.024204]
      }
    }, {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Point",
        "coordinates": [28.938674, 41.013324]
      }
    }
  ]
}'

geo_nearest(point1, points)
```

`geo_planepoint` *Planepoint*

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",
```

```
  "properties": {
    "a": 11,
    "b": 122,
    "c": 44
  },
  "geometry": {
    "type": "Polygon",
    "coordinates": [[
      [-75.1221, 39.57],
      [-75.58, 39.18],
      [-75.97, 39.86],
      [-75.1221, 39.57]
    ]]
  }
}'

geo_planepoint(point, triangle)
```

`geo_pointgrid` *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 *Trianglegrid*

Description

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

Usage

```
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

```
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 *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`

Examples

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

`MultiPolygon` *MultiPolygon*

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`

Examples

```
'{
  "type": "MultiPolygon",
  "coordinates": [
    [[[102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0], [102.0, 2.0]]],
    [[[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]],
    [[100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8], [100.2, 0.2]]]
  ]
}'
```

`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` *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` *Get json library version information*

Description

Get json library version information

Usage

```
version()
```

Value

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

Examples

```
version()
```

Index