geomodelr Documentation

Release 0.1.5

Geomodelr, Inc.

Aug 26, 2019

Contents

Geomodelr is a web tool for creating geological models easily. To create a geological model:

- Go to https://www.geomodelr.com.
- Register.
- Create a study and a model.
- Create and download a model version of your model.

You might want to use the model for calculations, geostatistics, simulations, or simply to know what geological unit is present at a given point. With this tool you can do all that.

CHAPTER 1

Contents

1.1 Geomodelr, Introduction.

To use geomodelr query tool you just need to:

```
import geomodelr
# load your model.
model = geomodelr.model_from_file('/path/to/your/model_version.json')
# query your model.
unit, gmlr_distance = model.closest((1000, 1000, 0.0))
# do stuff...
if unit == 'Batholith':
    ...
```

You can also use this tool as a script:

```
$ geomodelr -q /path/to/your/model_version.json
1000 1000 0
Batholith
```

1.1.1 Features

- Query the model in the coordinate system you defined.
- Query the topography heights and query the model with topography in mind.
- Query the intersection of faults and planes.
- Generate grids, use it as a help tool to generate meshes, assign properties for simulations or create block models.
- What do you want to do?

1.1.2 Installation

The requirements of Geomodelr Query Tool are: - Currently, Linux and Mac OS X are supported in Python 2.7 but we plan to support Windows and Python 3.5 in the near future. - C++ Build tools that support C++11. - Boost Libraries. - numpy and (pip will install them).

In general, you can install geomodelr by calling:

```
pip install geomodelr
```

Ubuntu Linux

You can install it from the command line:

```
# This will install boost.
sudo apt-get install libboost-all-dev
# This will install geomodelr globally.
sudo pip install geomodelr
# OR You can also use virtualenv.
virtualenv env && source env/bin/activate
pip install geomodelr
```

Mac OS X

- Install mac ports from https://www.macports.org/
- Now from the command line:

```
sudo ports install boost
sudo ports install pip
INCLUDE_DIRS="/opt/local/include/boost" LIBRARY_DIRS="/opt/local/lib" LIBRARIES=
↪"boost_python-mt" pip install geomodelr --user
```

Mac OS X El Capitan has a binary wheel so you don't need to install boost or anything besides pip and geomodelr.

1.1.3 Support

If you are having problems, write to support@geomodelr.com.

1.1.4 License

This project is licensed under the Affero GPL license https://www.gnu.org/licenses/

1.2 geomodelr package

1.2.1 Submodules

1.2.2 geomodelr.model module

class geomodelr.model.GeologicalModel(geolojson, delete=True, params={'faults': 'basic', 'map': 'disabled'})

Bases: `geomodelr.cpp.Model`

Interface to query a Geological model from Geomodelr.com. The models in Geomodelr.com are saved in Geological JSON. A Geological JSON is a set of GeoJSON FeatureCollections with a transformation. Go to Geomodelr.com, create a new model and use it with this tool.

closest((Model)arg1, (object)point) -> tuple :

Given a point, it finds the geological unit that's defined as the closest to that point.

The basic definition of the algorithm is that, given a match between geological units, the distance from the point to the unit is the sum of the in-section distance to the point averaged by the distance to the cross section.

Args: (tuple) point: The three coordinates of the point in the given coordinate system.

Returns: (tuple): A tuple with the geological unit and the defined distance to that unit.

C++ signature : boost::python::tuple closest(ModelPython {lvalue},boost::python::api::object)

closest_aligned((Model)arg1, (object)point) -> tuple :

Same as closest but in the coordinate system of the parallel cross sections model.

The basic definition of the algorithm is that, given a match between geological units, the distance from the point to the unit is the sum of the in-section distance to the point averaged by the distance to the cross section.

This algorithm returns the lowest value of the defined distance.

Args: (tuple) point: The three coordinates of the point in the parallel sections coordinate system.

Returns: (tuple): A tuple with the geological unit and the defined distance to that unit.

C++ signature : boost::python::tuple closest_aligned(ModelPython {lvalue},boost::python::api::object)

closest_topo((Model)arg1, (object)point) -> tuple :

Same as closest but it returns (AIR, inf) if the point is above the topography.

It first looks if the point is above the topography and returns (AIR, inf) in that case. Otherwise it returns the same as closest.

Args: (tuple) point: The three coordinates (easting, northing, altitude a.s.l) of the point in the given coordinate system.

Returns: (tuple): A tuple with the geological unit and the defined distance to that unit or AIR if it's above the topography.

C++ signature : boost::python::tuple closest_topo(ModelPython {lvalue},boost::python::api::object)

closest_topo_aligned((Model)arg1, (object)point) -> tuple :

Same as closest_topo, but in the coordinate system of the cross sections.

Args: (tuple) point: The three coordinates (easting, northing, altitude a.s.l) of the point in the given coordinate system.

Returns: (tuple): A tuple with the geological unit and the defined distance to that unit or AIR if it's above the topography.

C++ signature : boost::python::tuple closest_topo_aligned(ModelPython {lvalue},boost::python::api::object)

geomodelr_distance((Model)arg1, (unicode)unit, (list)point) -> float :

C++ signature : double geomodelr_distance(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::list)

height((Model)arg1, (object)point) -> float :

Returns: the height at the given point at the topography.

It returns the height at the point stored in the topography. In case the point it's outside the bounds of the model, it returns the height of the closest point inside.

Args: (tuple)point: The two coordinates (easting, northing) of the point in the given coordinate system.

Returns: (real) The height as stored in the topography.

C++ signature : double height(ModelPython {lvalue},boost::python::api::object)

info((Model)arg1) -> dict :

C++ signature : boost::python::dict info(ModelPython {lvalue})

intersect_plane((Model)arg1, (list)arg2) -> dict :

Intersects a plane with the faults of the Geological Model.

Takes a plane represented with its four corners and returns the set of lines that intersect that plane with the faults.

Args: (list) plane: list with the four corners of the plane that we want to intersect the fault with.

Returns: (dict): a dictionary with fault names as keys, and lines, (list of points) as values. The coordinates go from the lower left corner, (0.0, 0.0).

C++ signature : boost::python::dict intersect_plane(ModelPython {lvalue},boost::python::list)

intersect_planes((Model)arg1, (list)arg2) -> dict :

Intersects a set of planes with the faults of the Geological Model. Takes a set of plane represented with its four corners and returns the set of lines that intersect that plane with the faults. The coordinates start from the first plane lower corner, and increase by dist(plane[i][0], plane[i][1]) for the next plane.

Args: (list) plane: List with planes. Each plane has a list with four corners that we want to intersect the fault with.

Returns: (dict): a dictionary with fault names as keys, and lines, (list of points) as values.

C++ signature : boost::python::dict intersect_planes(ModelPython {lvalue},boost::python::list)

intersect_topography((Model)arg1, (dict)arg2) -> dict :

C++ signature : boost::python::dict intersect_topography(ModelPython {lvalue},boost::python::dict)

inverse_point((Model)arg1, (object)internal_point) -> tuple :

From internal coordinates, it returns the point in the given coordinate system.

It returns easting, northing and altitude from in-section x coordinate, in-section y coordinate, cut coordinate

Args: (tuple) point: The three coordinates of the internal point.

Returns: (tuple) The point in the given coordinate system

C++ signature : boost::python::tuple inverse_point(ModelPython {lvalue},boost::python::api::object)

make_matches()

Prepares the model to query by matching polygons and lines. It finds which polygons, when projected to the next cross section, intersect. After that, it tries to match faults with the same name by triangulating them and trying to find a continuous set of triangles between the two lines that go from the ends to the other side.

model_point((Model)arg1, (object)point) -> tuple :

Translates the point to internal coordinates

It returns in-section x coordinate, in-section y coordinate, cut coordinate

Args: (tuple) point: The three coordinates (esting, norting, altitute a.s.l) of the point in the given coordinate system.

Returns: (tuple) The point in the internal coordinate system.

C++ signature : boost::python::tuple model_point(ModelPython {lvalue},boost::python::api::object)

print_information(verbose=False)

Prints the information of the geological model just loaded.

Prints the version, coordinate system and valid coordinates that the geological model takes.

Args: (boolean) verbose: You can print more information with verbose=True.

signed_distance((Model)arg1, (unicode)unit, (object)point) -> float :

Given unit U and a point P, it finds the geomodelr distance to U minus the geomodelr distance to the closest unit different to U

It returns a signed distance that's zero at the boundary of the unit, negative inside the unit and possitive outside the unit

Args: (string) unit: The unit to measure the signed distance to

(tuple) point: The three coordinates (esting, norting, altitute a.s.l) of the point in the given coordinate system.

Returns: (double) The signed distance from the unit to the point.

C++ signature : double signed_distance(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::api::object)

signed_distance_aligned((Model)arg1, (unicode)unit, (object)point) -> float :

Same as signed_distance but in the coordinate system of the cross sections.

Args: (string) unit: The unit to measure the signed distance to

(tuple) point: The three coordinates (esting, norting, altitute a.s.l) of the point in the given coordinate system.

Returns: (double) The signed distance from the unit to the point.

C++ signature : double signed_distance_aligned(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::api::object)

signed_distance_bounded((Model)arg1, (unicode)unit, (object)point) -> float :

Given unit U and a point P, it finds the geomodelr distance to U minus the geomodelr distance to the closest unit different to U

It returns a signed distance that's zero at the boundary of the unit, negative inside the unit and possitive outside the unit

unlike signed_distance, when the point is outside the bounds of the model, or above the topography, it returns a positive number (outside)

Args: (string) unit: The unit to measure the signed distance to

(tuple) point: The three coordinates (esting, norting, altitute a.s.l) of the point in the given coordinate system.

Returns: (double) The signed distance from the unit to the point.

C++ signature : double signed_distance_bounded(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::api::object)

signed_distance_bounded_aligned((Model)arg1, (unicode)unit, (object)point) -> float :

Same as signed_distance_bounded but in the coordinate system of the cross sections.

Args: (string) unit: The unit to measure the signed distance to

(tuple) point: The three coordinates (esting, norting, altitute a.s.l) of the point in the given coordinate system.

Returns: (double) The signed distance from the unit to the point.

C++ signature : double signed_distance_bounded_aligned(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::api::object)

signed_distance_unbounded((Model)arg1, (unicode)unit, (object)point) -> float :

Given unit U and a point P, it finds the geomodelr distance to U minus the geomodelr distance to the closest unit different to U

It returns a signed distance that's zero at the boundary of the unit, negative inside the unit and possitive outside the unit

unlike signed_distance unbounded, it just returns a positive number when the point is above the topography. It does not always produce solids

Args: (string) unit: The unit to measure the signed distance to

(tuple) point: The three coordinates (esting, norting, altitute a.s.l) of the point in the given coordinate system.

Returns: (double) The signed distance from the unit to the point.

C++ signature : double signed_distance_unbounded(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::api::object)

signed_distance_unbounded_aligned((Model)arg1, (unicode)unit, (object)point) -> float :

Same as signed_distance_unbounded but in the coordinate system aligned with the cross sections.

Args: (string) unit: The unit to measure the signed distance to

(tuple) point: The three coordinates (esting, norting, altitute a.s.l) of the point in the given coordinate system.

Returns: (double) The signed distance from the unit to the point.

C++ signature : double signed_distance_unbounded_aligned(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::api::object)

signed_distance_unbounded_aligned_restricted((Model)arg1, (unicode)arg2, (object)arg3, (object)arg4) -> float :

C++ signature : double signed_distance_unbounded_aligned_restricted(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::api::object,boost::python::api::object)

signed_distance_unbounded_restricted((Model)arg1, (unicode)arg2, (object)arg3, (object)arg4) -> float :

C++ signature : double signed_distance_unbounded_restricted(ModelPython {lvalue},std::__cxx11::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >,boost::python::api::object,boost::python::api::object)

validate()

Validates that the Geological JSON has correct information.

class geomodelr.model.GeologicalSection(geolojson, delete=True, params={'faults': 'basic'})

Bases: `geomodelr.cpp.Section`

Interface to query a single Geological Cross Section or Map.

closest((Section)arg1, (object)arg2) -> tuple :

C++ signature : boost::python::tuple closest(SectionPython {lvalue},boost::python::api::object)

distance((Section)arg1, (list)arg2, (int)arg3) -> float :

C++ signature : double distance(SectionPython {lvalue},boost::python::list,int)

info((Section)arg1) -> dict :

C++ signature : boost::python::dict info(SectionPython {lvalue})

geomodelr.model.model_from_file(filename)

Entry point for the API. It creates the geological model from the file path. The geological model is a model of geomodelr.com, downloaded as a version.

Args: (str) filename: The path to the Geological JSON file downloaded from Geomodelr.com.

Returns: (GeologicalModel): The output Geological model to query the geological units freely.

1.2.3 geomodelr.cpp module

geomodelr.cpp.calculate_section_bbox((object)arg1, (object)arg2, (object)arg3, (float)arg4) -> tuple :

C++ signature : boost::python::tuple calculate_section_bbox(boost::python::api::object,boost::python::api::object,boost::python

geomodelr.cpp.extend_line((bool)arg1, (object)arg2, (list)arg3) -> list :

C++ signature : boost::python::list extend_line(bool,boost::python::api::object,boost::python::list)

geomodelr.cpp.faultplane_for_lines((list)arg1, (list)arg2) -> list :

C++ signature : boost::python::list faultplane_for_lines(boost::python::list,boost::python::list)

geomodelr.cpp.find_faults_intersection((dict)arg1, (list)arg2) -> dict :

C++ signature : boost::python::dict find_faults_intersection(boost::python::dict,boost::python::list)

geomodelr.cpp.find_mesh_plane_intersection((list)arg1, (list)arg2) -> list :

C++ signature : boost::python::list find_mesh_plane_intersection(boost::python::list,boost::python::list)

geomodelr.cpp.join_lines_tree_test((list)arg1) -> list :

C++ signature : boost::python::list join_lines_tree_test(boost::python::list)

geomodelr.cpp.set_verbose((bool)verbose) -> None :

Sets the operations as verbose.

When creating the model, it will advice the user of problems with geometries or matchings.

Args: (boolean) verbose: if geomodelr should be verbose when creating the model.

C++ signature : void set_verbose(bool)

geomodelr.cpp.topography_intersection((dict)arg1, (dict)arg2) -> dict :

C++ signature : boost::python::dict topography_intersection(boost::python::dict,boost::python::dict)

1.2.4 geomodelr.utils module

geomodelr.utils.generate_fdm_grid(query_func, bbox, grid_divisions, max_refinements)

Generates a grid of points with a FDM like refinment method. It first generates a simple grid. then it checks if a cell needs refinement. If it does, it marks it as a cell to refine. Then it goes through every axis, creating planes where the cell needs refinements, plus marking the cells as not needing refinement.

Args: (function) query_func: a function of the geological model that returns a unit.

(list) bbox: the bounding box to search in.

(int) grid_divisions: the number of points for the grid.

(int) max_refinements: the number of refinements for this FDM scheme.

geomodelr.utils.generate_octtree_grid(query_func, bbox, grid_divisions, fdm_refine, oct_refine)

Generates an octree grid, starting with an FDM refined grid. The octtree grid divides each cell in 8 looking at the differences of material until reaching the number of refinements.

Args: (function) query_func: a function of the geological model that returns a unit.

(list) bbox: the bounding box to search in.

(int) grid_divisions: the number of points for the grid.

(int) fdm_refine: the number of refinements for the fdm scheme.

(int) oct_refine: the number of refinements for the octree scheme

geomodelr.utils.generate_simple_grid(query_func, bbox, grid_divisions)

Returns a uniform grid of sizes grid_divisions x grid_divisions x grid_divisions that covers the given bbox evaluated with the query function.

Args: (function) query_func: a function of the geological model that returns a unit.

(list) bbox: the bounding box to search in.

(int) grid_divisions: the number of points for the grid.

geomodelr.utils.octtree_volume_calculation(query_func, bbox, grid_divisions, oct_refine)

An example of how to get the volumes of all units.

Args: (function) query_func: a function of the geological model that returns a unit.

(list) bbox: the bounding box to search in.

(int) grid_divisions: the number of points for the grid.

(int) oct_refine: the number of refinements for the octree scheme

1.2.5 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search