pySlope: A 2D Slope Stability Software using Bishop's Method

Jesse Bonanno

07 September 2022

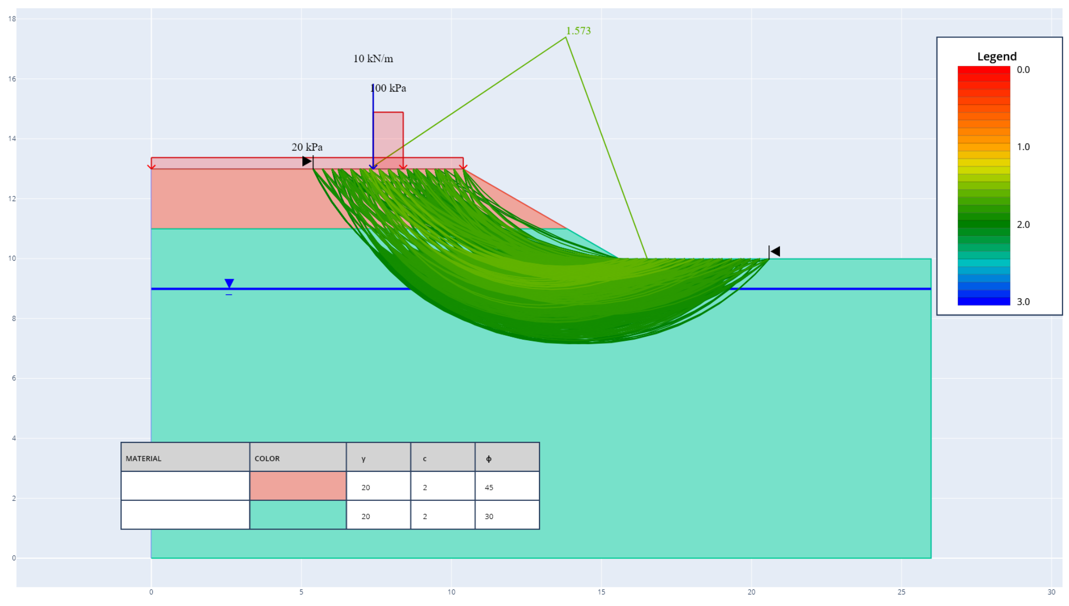
pypi v1.1.9 | license MIT | codefactor A | codecov 87% | docs passing | CI passing

Summary

`pySlope` is a 2D slope stability module based on bishops method of slices (Bishop, 1955). This module allows for the following parameters:

- unlimited horizontal geological units
- water table
- uniform & line loads
- slope search limits

The program searches for the critical failure slope using an 'entry and exit' search method (GEO-SLOPE, 2022) or can calculate the factor of safety for predefined failure planes. The user can return plots for the critical failure slope or plots for all failure slopes below a certain factor of safety. An example plot is shown below.



MATERIAL	COLOR	γ	c	ϕ
	🟥	20	2	45
	🟩	20	2	30

The `pySlope` [package repository](#) can be found on Github and is ready for installation using `pip`. A text-based example of the package can be found on this [Jupyter Notebook](#) and a web-based graphical user interface (GUI) is available at https://pyslope.herokuapp.com/.

Project Purpose

The purpose of this project is to:

1. Create a free online slope stability software website
2. Provide a pythonic solution to implementing Bishop's method based on object oriented coding principles
3. Create a package that makes sensitivity analysis fast and easy

Performing a slope stability calculation by hand is extremely uneconimical and time consuming. The problem involves a lot of geometrical mathematics which can make the calculation hard to achieve with only excel. Python packages exist for geometrical mathematics which makes Python well suited for implementing a slope stability analysis package. There is however, no well-documented open-source slope stability software that can currently be found online. This package aims to fill that gap.

Although there are several easy to use slope stability tools (slide2, 2012) (hyrcan, 2021) iterating through different models can be laborious. There are no projects which have an Application Program Interface (API) that can be used to perform a quick and easy sensitivity analysis, except for this project.

Key Considerations

Implementing a working package is rather straightforward however to be practical the following key constraints were considered in determining the usability of the program;

1. Speed
2. Accuracy
3. Code readability
4. Modelling flexibility

The first major design decision required to address the key constraints was to use a 1D based input method to get a 2D model. This involved limiting the geometry to only have one slope, and to only have horizontal material boundaries. This is useful for simple problems, especially those which are a part of Construction Phase Service (CPS) engineering works, but may not be suitable for complex slopes. By simplifying inputs to 1D the code base was simplified, modelling became easier and faster for the user and the analysis speed was enhanced.

To further speed up the program multiprocessing is used for large iterations. Cython was considered as an option to increase speed but was not pursued due to its effect on code readability and a less streamlined development process.

Accuracy of the software has been assessed by comparing to the results of two (2) other programs (slide2, 2012) (Hyrcan, 2021) as detailed in the documentation.

To ensure accuracy and readability are maintained Github actions are used as a part of CI/CD practices to check code formatting and that the code passes all unit tests.

Shortfalls

The solution only uses Bishop's method of analysis and doesnt consider other methods such as Spencer's or Janbu's Method. The solution doesn't allow for tension cracks, horizontal loads or anchors in its current version.

The website is hosted on Heroku free of cost. When Heroku transitions to a paid only model, a new host will need to be found.

Functionality and usage

A typical use case of the `pySlope` package involves the following steps:

1. Create a `Slope` object
2. Create `Material` objects and assign to `Slope`
3. Create `Udl` or `LineLoad` objects and assign to `Slope`
4. Set water table
5. Set analysis limits
6. Analyse slope for critical factor of safety
7. Create plots

You can follow along with this example below in this web based Jupyter notebook

Creating a Slope

The creation of a `Slope` instance involves the input of the:

- slope height (m) and
- angle (deg) or
- length (m)

Only one of the values is used out of the length and angle, the other value should be set to None.

```
s = Slope(height=3, angle=30, length=None)
```

Defining Materials

The creation of a `Material` object involves the input of:

- unit weight (kN/m3)
- friction angle
- cohesion (kPa)
- depth from top of slope to bottom of material layer (m)

Once a material is defined it can then be assigned to the `Slope` instance.

```
m1 = Material(
    unit_weight=20,
    friction_angle=45,
    cohesion=2,
    depth_to_bottom=2
)
m2 = Material(20, 30, 2, 5)     # Material defined with positional arguments
s.set_materials(m1, m2)         # An unlimited number of materials can be assigned at
one time
```

The slope will know to order the materials based on the depth to the bottom of the strata so the order that the materials are provided isn't important. It is important that the same depth isnt provided twice for two different materials, and attempting this will raise an error.

Defining Uniform Loads

The creation of a `Udl` (uniform distributed load) object involves the input of:

- magnitude of load (kPa)
- offset of load from crest of slope (m) (default 0 m)
- length of load (m) (default infinite)

```
u1 = Udl(magnitude = 100, offset = 2, length = 1)

# by default offset = 0 (m) and length = None.
u2 = Udl(magnitude = 20)

# assign uniform loads to model
s.set_udls(u1, u2)
```

Defining Line Loads

The creation of a `LineLoad` object involves the input of:

- magnitude of load (kN / m)
- offset of load from crest of slope (m) (default 0 m)

```

# define line load, similiar to Udl except there is no length parameter and magnitude
is in units (kN/m)
p1 = LineLoad(magnitude = 10, offset = 3)

# assign line loads to slope
s.set_lls(p1)
```

Defining Water Table

By default there is no water table. The water table is defined by its depth from the top of the slope (m).

```
s.set_water_table(4)
```

Defining Analysis Limits

Analysis limits can be specified as a general left and right limit, OR as a set of limits which control the range from which the top of failures can occur and the bottom of failures can occur.

Currently the model coordinates are dynamic in that the overall model dimensions are based on the size of the slope.

The `get_top_coordinates` and `get_bottom_coordinates` methods can be useful to help define limits in reference to the top and bottom of the slope.

```
s.set_analysis_limits(s.get_top_coordinates()[0] - 5, s.get_bottom_coordinates()[0] +
5)
```

.

Analysing Slope

To analyse the `Slope` the analyse_slope() method is called. By default 2000 iterations are run with 50 slices per failure plane.

```
# The user can change the number of slices and iterations with the method below.
# The line below is implicitly called and only required by the user if they want to
change iterations
s.update_analysis_options(slices=50, iterations=2500)

# run analysis
s.analyse_slope()
```

Interpretting results

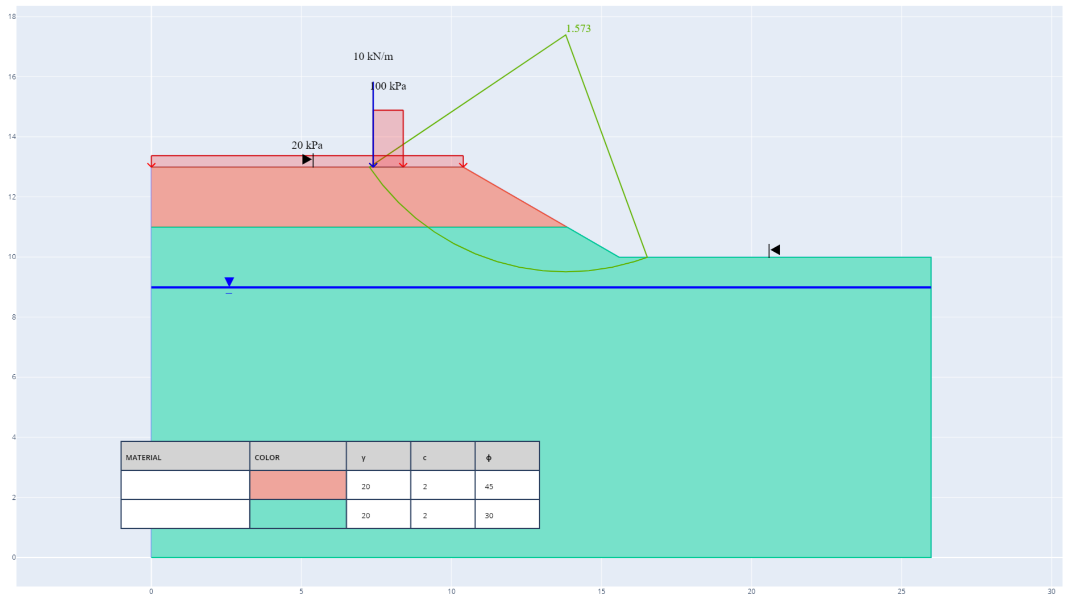
After analysing the slope the critical factor of safety can be taken as below.

```
print(s.get_min_FOS())
```

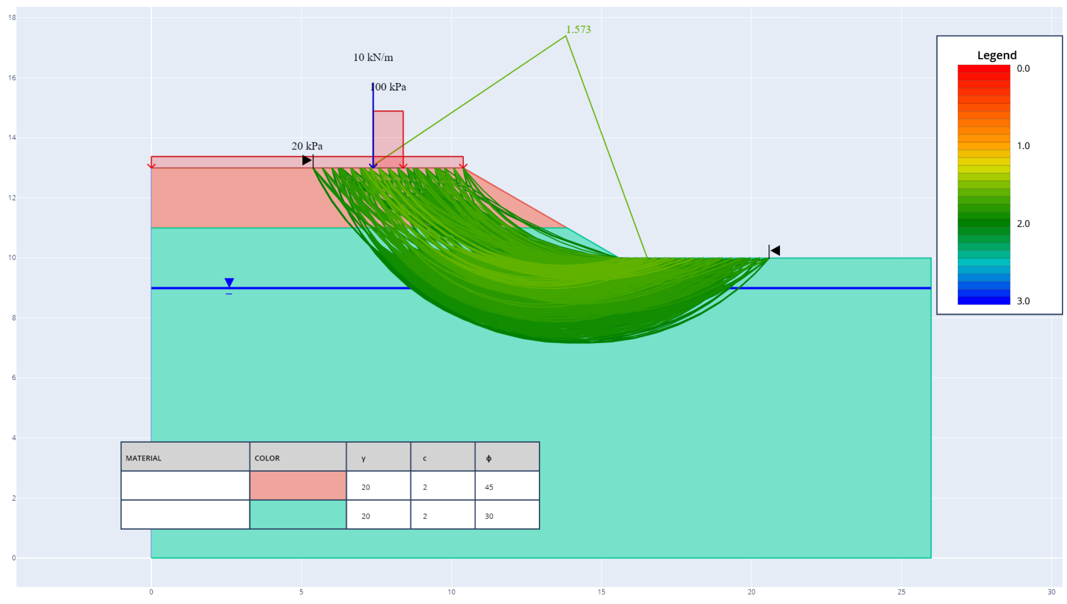
A more useful output might be a plot. Currently there are 3 main plots that can be called.

```
s.plot_boundary()  # plots only the boundary
s.plot_critical()  # plots the boundary with the critical failure of the slope
s.plot_all_planes(max_fos=i) # plots boundary with all slope failures below fos i
(where i is number)
```

Examples of the plots are shown below.



MATERIAL	COLOR	γ	c	ϕ
	🟥	20	2	45
	🟩	20	2	30



MATERIAL	COLOR	γ	c	ϕ
	🟥	20	2	45
	🟩	20	2	30

Dynamic Analysis

Instead of standard static analysis the user also has the option to make load objects *dynamic*. The user can then perform a dynamic analysis rather than static, which moves the load in order to determine the required offset for a minimum factor of safety.

Considering the example above, we can continue and make u1 dynamic.

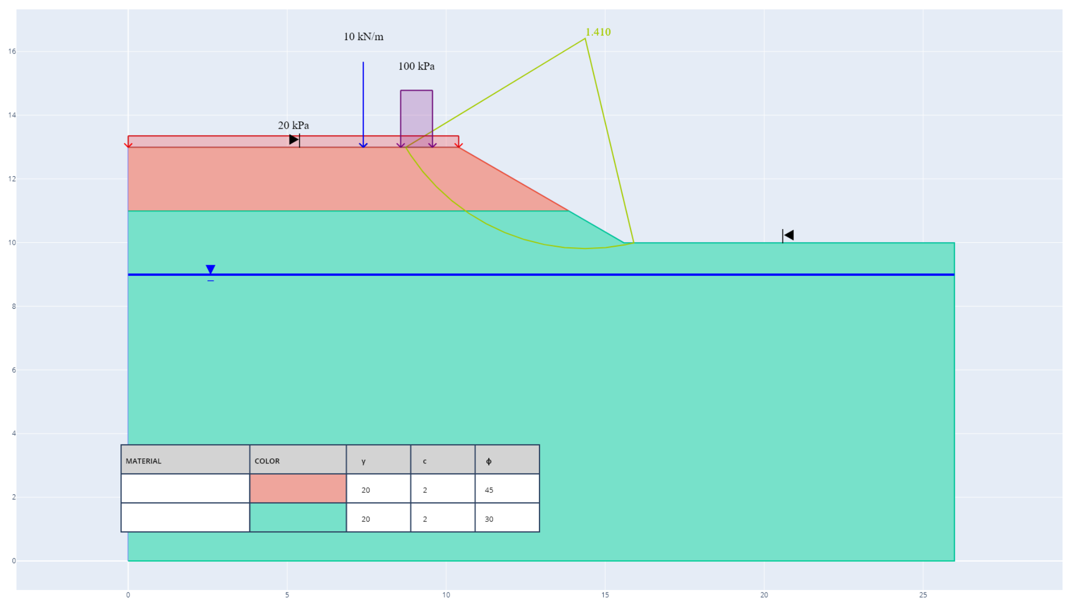
```
# remove udl object load from slope
s.remove_udls(u1)

# now lets add the udl again but this time set the load as 'dynamic'
# for all loads and materials we also have the option to set the color ourselves
# lets try set the color as 'purple'
s.set_udls(
    Udl(magnitude=100, length=1, offset=2, dynamic_offset=True, color='purple')
)

# run dynamic analysis aiming for a FOS of 1.4
s.analyse_dynamic(critical_fos=1.4)

# get dictionary of all determined minimum FOS with key value pairing of offset :
value
s.get_dynamic_results()

# or can print the values out
s.print_dynamic_results()
```

From this we get the following output results:

- Offset: 0.000 m, FOS: 1.288

- Offset: 0.735 m, FOS: 1.402
- Offset: 1.463 m, FOS: 1.510
- Offset: 5.186 m, FOS: 1.684

We can also get a plot as after running dynamic analysis all plots are based on the final iteration of the dynamic analysis.



MATERIAL	COLOR	γ	c	ϕ
	🟥	20	2	45
	🟩	20	2	30

References

Bishop, A., 1955. The use of the Slip Circle in the Stability Analysis of Slopes. Géotechnique, 5(1), pp.7-17.

GEO-SLOPE, 2022. Stability Modelling with SLOPE/w. 8th ed. [ebook] GEO-SLOPE International. Available at: https://geoslope.com [Accessed 5 September 2022].

Slide2, 2012. Torronto: RocScience.

Mikola, R., 2021. Hyrcan. San Fransisco: Geowizard.

Appendix

Licensing and Contributing

The entire code is publicly available for free and the project is licensed under the MIT software license. Anyone is free to download the code and use it in any way they like. If you are interested in contributing to the project see the contribution page on the package directory or get in contact.

Acknowledgments

Thanks to Stevan Lukic, who helped with the website front end and is implementing the package on the civils.ai project.