Soils and Rocks
An International Journal of Geotechnical and Geoenvironmental Engineering
www.soilsandrocks.com
ISSN 1980-9743
ISSN-e 2675-5475

RSF: a python-based analytical framework for geotechnical engineering solutions

Rouhollah Basirat[1#]

Technical Note

Keywords
Computational geotechnics
Python
Analytical methods
Geotechnical problems

Abstract

This paper presents the design and development of a comprehensive standalone application for geotechnical engineering, built entirely using Python. Unlike conventional commercial platforms or numerical packages, the application focuses on providing transparent, reproducible, and analytically grounded tools for core geotechnical challenges. This article aims to present the topic in a straightforward and easy-to-understand manner. The application includes different independent modules that cover critical areas of soil and rock mechanics, foundation engineering, tunnel lining design, slope stability, and ground improvement. All modules are based on classical solutions, derived from principles such as Rankine's and Coulomb's earth pressure theories, Terzaghi's bearing capacity equations, consolidation and elastic settlement theory, the Hoek–Brown rock failure criterion, etc. The software leverages Python's scientific libraries for data operations, interpolation, plotting, regression, and features a graphical user interface (GUI) built by Tkinter for interactive input and visualization. By adopting a modular architecture, each analytical solution can be accessed, maintained, and extended independently, promoting both scalability and usability. The program has been validated against standard geotechnical design examples and empirical relationships, showing excellent agreement. It serves both educational and professional purposes.

1. Introduction

Geotechnical engineering encompasses a wide range of problems dealing with the mechanical behavior of earth materials and the analysis and design of foundations, slopes, retaining walls, and underground structures. The need for accurate, efficient, and adaptable computational tools in this field is driven by the demands of civil infrastructure projects, mining operations, and natural hazard assessments. Analytical methods—derived from classic theories like Terzaghi's bearing capacity, Rankine's earth pressure theory, settlement estimation from elastic and consolidation frameworks, the Hoek-Brown failure criterion, etc.—remain vital in day-to-day engineering practice and academia. These solutions are transparent, fast, and typically require only basic input parameters.

Despite the rise of finite element modeling and other numerical techniques, analytical methods retain a unique advantage in simplicity, speed, and conceptual clarity. Yet, in practice, applying these formulas accurately and consistently still requires time, discipline, and often repetitive manual calculations in different areas such as tunneling (Basirat, 2025), sheet pile (Hassani et al., 2016), different types of walls (Das, 2016) etc. Commercial geotechnical software attempts to automate this process but tends to obscure computational logic, making them less transparent or modifiable. Moreover, such software can be prohibitively expensive and inaccessible in academic settings.

In addition, Python package can simulate soil with a single python script, saving you effort and money and providing reliable results (Zhu et al., 2018). Krishna et al. (2024) used Python programming serves as a pivotal tool for forecasting soil classification, with a specific focus on sieve analysis, plastic limit, and liquid limit parameters. Kim et al. (2025) explored the application of ChatGPT by applying prompt engineering to enable the model to process a design standard documentation and generate Python code for calculating the vertical bearing capacity of piles. Jürgens & Henke (2021) implemented the statistical model using a highly accurate, reliable, and easy-to-use python script for soil simulation.

#Corresponding author. E-mail address: r.basirat@modares.ac.ir
[1]WBI GmbH, Weinheim, Germany.

Submitted on August 5, 2025; Final Acceptance on December 15, 2025; Discussion open until November 30, 2026.
Editor: Renato P. Cunha
https://doi.org/10.28927/SR.2026.012825

To bridge the gap between analytical methods and numerical modeling as well as programming, RSF (Rock Soil Foundation) is developed based on Python programming and is composed of 20 separate code modules. Each module implements a specific analytical model used in geotechnical engineering. The application is structured to allow engineers, students, and researchers to calculate key parameters—such as earth pressures, settlements, factors of safety for slopes, pile capacity, and tunnel support—without the need for complex software licenses or black-box algorithms.

The objective of this paper is to present not just the software product, but the logic and methodology behind its development in a simple and accessible way. We categorize the modules by engineering domain, describe the libraries and algorithms used, explain how the GUI is constructed, and provide annotated code snippets to clarify implementation choices. The result is a modular, extensible, and robust geotechnical toolkit that can evolve into a hybrid analytical-numerical platform in future iterations.

2. Program structure and architecture

2.1 Why python for geotechnical engineering

Python is increasingly becoming the language of choice for engineers and scientists due to its simplicity, powerful ecosystem, and open-source nature. In geotechnical engineering, where calculations often involve combinations of empirical models, closed-form analytical expressions, and iterative problem-solving, Python provides a unique blend of flexibility and computational power.

2.2 Libraries

The use of Python in this application was not incidental but deliberate. Many analytical models in geotechnical engineering rely on manipulating equations involving soil parameters (e.g., φ, c, γ), depth, and geometry. With libraries like NumPy and SciPy, complex trigonometric, logarithmic, and statistical calculations can be executed with high precision and minimal code. These are crucial when calculating stress distributions, pressure envelopes, and capacity factors.

Python's Matplotlib and Seaborn libraries allow for clear graphical representation of output such as soil pressure vs. depth profiles, stress-strain curves, and settlement-time graphs. These outputs not only provide insight but also validation, as visual checks help confirm algorithm performance. The PIL (Pillow) can also be used for displaying the images.

The integration of Tkinter adds a user interface layer where engineers can enter data interactively, toggle design conditions (e.g., dry vs. saturated soil), and instantly view results. This makes the application accessible to those with minimal programming background. Furthermore, Python's clean syntax ensures that code remains readable and modifiable for future development.

For extending analytical models using machine learning, Python also supports Scikit-learn, which enables regression, classification, and predictive modeling. Though not central to the current application, such tools have been integrated where linear interpolation or extrapolation of field or empirical data is needed.

2.3 RSF modules

The RSF application is composed of 20 modules (Figure 1), all structured around consistent logic: isolate each engineering problem into an independent script while adhering to a common interface philosophy. If necessary, the modules can also be supplemented. This app is available in https://www.rock-soil-foundation.com. The key structural features are as follows:

- Modular Design: Each engineering solution (e.g., bearing capacity, tunnel lining pressure, pile capacity) is encapsulated in a separate file.
- Analytical Core: Every module includes equations coded directly from theory, without relying on black-box numerical solvers.
- Function-Based Architecture: Inputs are defined as dictionaries or parameters; outputs are returned as dictionaries, tuples, or objects.
- Interpolation/Extrapolation: Many modules use SciPy's grid data or interpolated to perform 2D/3D interpolation of empirical tables (e.g., influence factors, N_γ values).
- Error Handling: Try-except blocks handle non-convergence and user entry issues.
- Units: Consistent use of SI units (kPa, m, degrees) across modules.
- Data Flow: Some modules calculate intermediate steps (e.g., net pressure, influence depth) and feed results into subsequent computations.
- Output Visualization: Functions return plots using Matplotlib, often with pressure distributions, settlement over time, or polar diagrams for tunnels.

2.4 Key functions, code patterns, and technical features

This section outlines how the analytical models are implemented using Python functions. Most modules follow a functional programming paradigm, allowing calculations to be modular, repeatable, and easy to debug. Below is an outline of typical components in the functions across the modules.

2.4.1 Input structure

Each module accepts either a dictionary of input values or discrete arguments. Inputs often include:

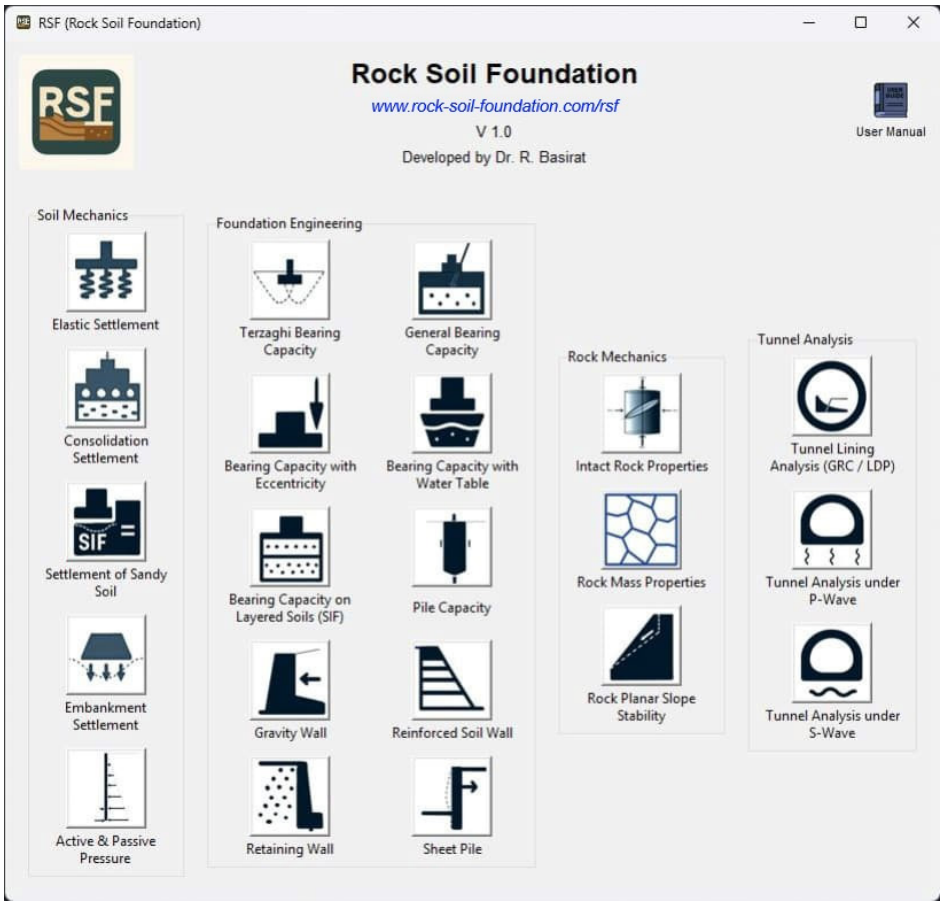

Figure 1. RSF modules.

- Geometric parameters: footing width (B), depth (D_f), height (H)
- Soil properties: friction angle (φ), unit weight (γ), cohesion (c), E_s (modulus), v (Poisson's ratio)
- Load data: surcharge (q), axial or eccentric loads (Q, M_x, M_y)
- Environmental: groundwater depth, layer thickness, seismic coefficients (k_h, k_v)

2.4.2 Code snippet

Every calculation module defines a function (using the def command) that contains the formula and calculation algorithms. For example, the following code is written for calculating earth pressure coefficient by Rankine method:

```
import math
def rankine_earth_pressure(phi):
    phi_rad = math.radians(phi)
    Ka = math.tan(math.radians(45 - phi
/ 2)) ** 2
    Kp = math.tan(math.radians(45 + phi
/ 2)) ** 2
    return Ka, Kp
```

This basic structure is replicated across modules with appropriate extensions, including surcharge handling, wall inclination, or cohesion. This function-based structure ensures that each analytical problem is handled independently while maintaining a unified logic across all modules. Such modularization not only simplifies debugging and further development but also allows users to adapt or extend each function to new boundary conditions or site-specific cases. The simplicity of the function syntax (def) makes it intuitive even for readers with limited programming background.

2.4.3 Interpolation, extrapolation, and tabular logic

Many empirical formulas in geotechnical engineering require lookup tables and interpolated values (e.g., bearing capacity factors based on φ). These are handled using SciPy's interpolate tools. This technique can be used in different geotechnical issues such as elastic settlement shape and depth factors, stress influence charts, rock classification tables, etc. As an illustrative example, the interpolation approach was applied to estimate the variation of the ratio c_a'/c_1' with respect to q_2/q_1 (Figure 2). The python code is as follows:

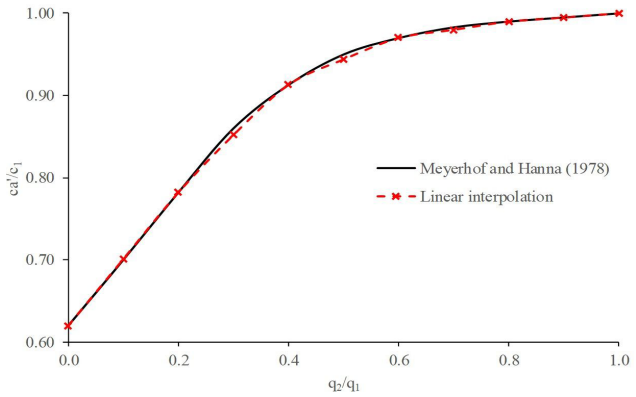

Figure 2. Comparing the variation of the ratio c_a/c_1 with respect to q_2/q_1 with the empirical method of Meyerhof and Hanna (1978).

```
import numpy as np
from scipy.interpolate import interp1d
# Graph c2/c1 versus ca/c1
table1 = np.array([
    [0.0, 0.62],
    [0.2, 0.78],
    [0.4, 0.91],
    [0.6, 0.97],
    [0.8, 0.99],
    [1.0, 1.00]
])
# Extract columns
c2_c1_values = table1[:, 0]
ca_c1_values = table1[:, 1]
# Interpolate ca/c1
ca_c1_interp = interp1d(
    c2_c1_values, ca_c1_values,
   kind='linear', fill_value="extrapolate"
)
def get_values(c2_c1):
     """Interpolates ca/c1 for a given
c2/c1."""
    return ca_c1_interp(c2_c1)
```

The results were computed using an interpolation interval of 0.1 for q_2/q_1 and compared with the reference curve proposed by Meyerhof & Hanna (1978). As shown in Figure 3, the interpolated curve closely follows the original relationship, confirming the reliability of the implemented interpolation method for representing empirical geotechnical correlations.

The interpolation/extrapolation functions are designed to be general, meaning that similar procedures can be easily reused across different geotechnical problems — for example, estimating bearing capacity factors, influence coefficients, or stress distribution parameters. This unified interpolation logic enhances both consistency and computational efficiency throughout the RSF framework.

The curve fitting using Scikit-learn and sometimes NumPy.polyfit is useful, when field data is incomplete, and predictive modeling is needed to extrapolate rock stiffness from known values. For example, this feature is used in the rock properties estimation modules for estimating *UCS* vs. Modulus as well as plotting trend lines across test data.

```
from sklearn.linear_model import
LinearRegression
import numpy as np
def fit_modulus_curve(UCS_data, modulus_
data):
   X = np.array(UCS_data).reshape(-1, 1)
    y = np.array(modulus_data)
    model = LinearRegression().fit(X, y)
    E_pred = model.predict(X)
    plt.scatter(UCS_data, modulus_data,
label='Measured')
  plt.plot(UCS_data, E_pred, label='Fitted
Line', color='red')
     plt.xlabel("Unconfined Compressive
Strength (MPa)")
    plt.ylabel("Modulus (GPa)")
    plt.title("Modulus vs UCS")
    plt.legend()
    plt.grid(True)
    plt.show()
   return model.coef_[0], model.intercept_
```

2.4.4 Plotting, visualization, and geometry representation

Visualization is not merely aesthetic; in geotechnical engineering, plots are critical for:

- Interpreting soil pressure distributions
- Visualizing geometry, such as slope failure planes
- Displaying results, such as stress-strain curves or settlement over time

In most modules, Matplotlib is used to show distributions. These plots help confirm linear vs. nonlinear pressure distributions and help validate input errors (e.g., negative depths). The following example shows the pressure distribution vs depth:

```
import matplotlib.pyplot as plt
def plot_pressure(depths, pressures):
    plt.plot(pressures, depths)
    plt.gca().invert_yaxis()
    plt.xlabel("Pressure (kPa)")
    plt.ylabel("Depth (m)")
      plt.title("Active Earth Pressure
Distribution")
    plt.grid(True)
    plt.show()
```

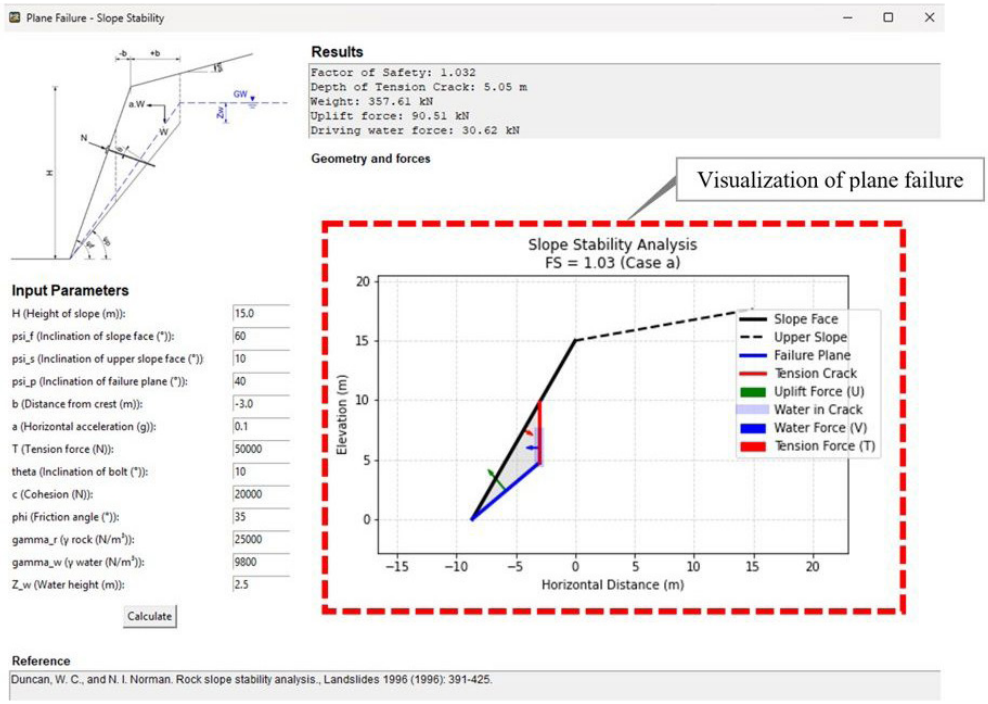

Figure 3. The output of rock plane failure modules.

To better understand the geometry of geotechnical issues, it should be visualized. This can be easily visualized by Python with defining points and line. As an example, the script for visualizing the plane failure (sliding plane), as well as ground surface in the slope stability calculations modules is shown as follows and the output of this code is provided in Figure 3:

```
def plot_slope_geometry(H, slope_angle, failure_plane_angle):
    import numpy as np
    import matplotlib.pyplot as plt
   x_ground = [0, H/np.tan(np.radians(slope_angle))]
    y_ground = [0, H]
   x_plane = [0, H/np.tan(np.radians(failure_plane_angle))]
    y_plane = [0, H]
      plt.plot(x_ground, y_ground, label="Ground Surface")
   plt.plot(x_plane, y_plane, label="Failure Plane", linestyle='--')
    plt.fill_between(x_plane, 0, y_plane, alpha=0.3, color='orange')
    plt.gca().invert_yaxis()
    plt.legend()
    plt.title("Slope Failure Geometry")
    plt.xlabel("Distance (m)")
    plt.ylabel("Depth (m)")
    plt.grid(True)
    plt.show()
```

Each visualization function in the RSF toolkit follows the same modular principle — data generation and plotting are separated, allowing the same computational output to be represented graphically or exported numerically. This approach ensures transparency of the analytical process and facilitates result verification by direct visual inspection. This feature helps students and engineers visualize failure mechanisms and link math with reality.

2.5 GUI implementation

The graphical user interface (GUI) is created using the built-in tkinter library, offering simplicity, cross-platform support, and good performance for desktop tools. The window is divided into different sections such as input fields, schematic image for description of geotechnical issues and also the meaning of input parameters, output result tabs, graphing panel, and reference list box. Figure 4 shows the GUI for reinforced soil wall modules as an example. In this figure all parts of a GUI have been shown. This is vital for engineers who do not code, as it allows them to use reliable analytical models without learning programming. Moreover, it makes the tool accessible for educational demonstrations and workshops, and it makes the software educationally friendly. An example for adding input widgets. is shown as:

```
for key, val in self.inputs.items():
   ttk.Label(frame, text=val['label']).grid(row=..., column=0)
```

```
entry = ttk.Entry(frame, width=12)
entry.insert(0, str(val['default']))
```

This dynamic loop ensures flexibility and compact code for managing 20+ inputs.

2.6 Program validation

After programming, it should be validated against standard geotechnical design examples and serves both educational and professional purposes. To achieve this goal, every module has been checked using different references such as Das (2016), Bowles (1997), EN1998 (CEN, 1998), Norrish & Wyllie (1996), Hoek (2007), FHWA (2009), Hassani & Basirat (2016), Hassani et al. (2016), Basirat et al. (2019), Basirat (2025) etc. Each script was rigorously tested, bugs were identified, and subsequently fixed. Figure 5 shows the validation results for the intact rock properties module, using the example provided in Hoek (2007, Chapter "Rock Mass Properties"). The results are consistent with those reported in the reference.

3. Comparison with manual spreadsheets

Using spreadsheets (e.g., Excel) has traditionally been the default for many engineers. The comparison of spreadsheets vs. Python shows in Table 1. For example, computing earth pressure for various φ and depths in Excel would require dozens of cells with repeated logic, manual chart updates, and complex conditional formatting. But in Python can be used: a single function does all the math; a loop runs every scenario and a Matplotlib plot dynamically updates. This is why Python is a future-proof solution for geotechnical analysis.

This comparison highlights that Python not only replicates spreadsheet calculations but also extends them through automation, visualization, and reproducibility, making it a more robust framework for analytical geotechnical analysis.

Table 1. Comparison of spreadsheets vs. Python.

spreadsheets	Python
Lack of errors handling	Transparent function logic
Hidden logic buried in cells	Reusable scripts and functions
Limited plotting for multiple dimensions	Scalable modules that grow with project scope
No reusability or version control	Advanced plotting, machine learning, and GUI capabilities
Difficult to automate or scale	

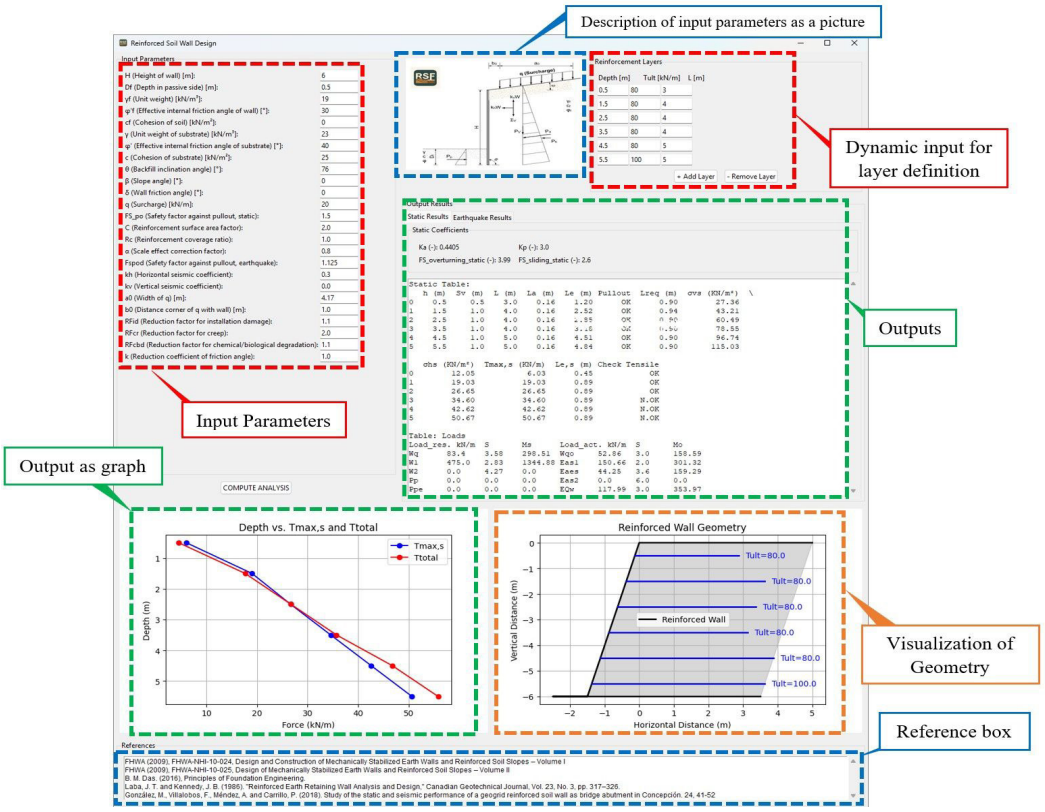

Figure 4. GUI for reinforced soil wall modules.

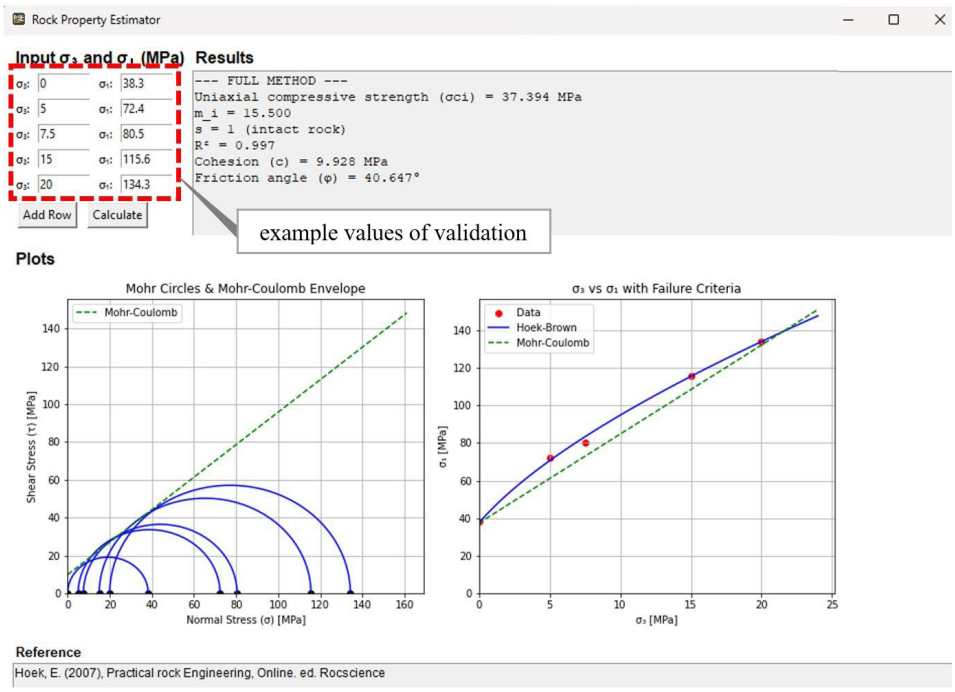

Figure 5. The intact rock properties module with validation example results.

4. Conclusion

This paper discussed the structure, design philosophy, and implementation of a Python application developed specifically to consolidate and automate these classical methods into a modular suite of tools in a simple language. Unlike commercial geotechnical software which often obfuscates computation through graphical layers, the RSF application exposes and preserves the analytical procedures within user-friendly Python scripts.

What distinguishes this application from others is its educational openness, reliance on classical methods, and use of modern programming principles. Python, being free and widely adopted in scientific computing, was the ideal choice for such an application. With packages like NumPy and SciPy, the software can perform numerical operations, interpolation, and symbolic computation with efficiency. Visualization tools such as Matplotlib provide instant graphical feedback, while Scikit-learn offers capabilities for predictive modeling (e.g., regression for parameter estimation). Finally, Tkinter makes it possible to deploy GUIs that allow users to interact with the calculations without writing any code.

Declaration of interest

The author has no conflicts of interest to declare. The author has observed and affirmed the contents of the paper and there is no financial interest to report.

Data availability

All data produced or examined in the course of the current study are included in this article.

Declaration of use of generative artificial intelligence

This work was prepared with the assistance of generative artificial intelligence (ChatGPT) for the purpose of language editing and grammar checking only. The entire process of using this tool was supervised, reviewed and when necessary edited by the authors. The authors assume full responsibility for the content of the publication that involved the aid of GenAI.

List of symbols and abbreviations

c	Cohesion of the soil
c_a/c_1,	Cohesion ratio in the theory of Meyerhof & Hanna (1978)
k_h	Horizontal seismic coefficients
k_v	Vertical seismic coefficients
q	Surcharge or applied uniform pressure
q_2/q_1	Ultimate bearing capacities ratio in the theory of Meyerhof & Hanna (1978)
B	Footing / foundation width
D_f	Foundation depth
E_s	Young's modulus (soil or material modulus)
EN	Eurocode 1998

FHWA	Federal Highway Administration of the United States
GUI	Graphical User Interface
H	Height (wall, layer, or geometry)
K_a	Active earth pressure coefficient
K_p	Passive earth pressure coefficient
M_x	Bending moments about x axes
M_y	Bending moments about y axes
N_γ	Bearing capacity factors
Q	Axial load (e.g., on pile or column)
RSF	Rock–Soil–Foundation application
UCS	Uniaxial Compressive Strength (rock/soil sample)
γ	Unit weight (density) of soil
ν	Poisson's ratio
φ	Angle of internal friction of soil

References

Basirat, R. (2025). Analysis of tunnel lining internal forces under the influence of S and P-waves: an analytical solution and quasi-static numerical method. *Rock Mechanics Bulletin*, 4(1), 100168. https://doi.org/10.1016/j.rockmb.2024.100168.

Basirat, R., Salari-rad, H., & Molladavoodi, H. (2019). Analysis of the monolithic and segmental tunnel lining under earthquake loading. *Ingeniería Sísmica*, 36(4), 140-155.

Bowles, J.E. (1997). *Foundation analysis and design* (5th ed.). McGraw-Hill.

Das, B.M. (2016). *Principles of geotechnical engineering* (8th ed.). Cengage Learning.

European Committee for Standardization – CEN. (1998). *EN1998-5: Eurocode 8: Design of structures for earthquake resistance – Part 5: Foundations, retaining structures and geotechnical aspects*. CEN.

Federal Highway Administration – FHWA. (2009). *Design and construction of mechanically stabilized earth walls, FHWA-NHI-10-024 & 025*. U.S. Department of Transportation.

Hassani, R., & Basirat, R. (2016). Application of hyperstatic reaction method for designing of tunnel permanent lining, Part I: 2D numerical modelling. *Civil Engineering Journal*, 2(6), 244-253. https://doi.org/10.28991/cej-2016-00000030.

Hassani, R., Basirat, R., & Mahmoodian, N. (2016). Classical method and numerical modeling for designing of sheet pile wall (case study: tuti-bahri bridge, sudan). In: *3rd International Conference on Modern Research in Civil Engineering, Architectural and Urban Development*, Berlin.

Hoek, E. (2007). *Practical rock engineering.* Rocscience.

Jürgens, H., & Henke, S. (2021). The design of geotechnical structures using numerical methods. *IOP Conference Series. Earth and Environmental Science*, 727(1), 012021. https://doi.org/10.1088/1755-1315/727/1/012021.

Kim, S., Kim, D., & Youn, H. (2025). Automating vertical bearing capacity calculations using python: Prompt engineering of ChatGPT on API RP 2A. *Developments in the Built Environment*, 21, 100628. https://doi.org/10.1016/j.dibe.2025.100628.

Krishna, T., Chittaranjan, M., Shaik, R., Priya, M., Babu, G., Bhogesh, B., & Prasanth, G. (2024). Forecasting the classification of soil systems by using Python programming. *Research Square*; Advance online publication https://doi.org/10.21203/rs.3.rs-4148930/v1.

Meyerhof, G.G., & Hanna, A.M. (1978). Ultimate bearing capacity of foundations on layered soil under inclined load. *Canadian Geotechnical Journal*, 15(4), 565-572. https://doi.org/10.1139/t78-060.

Norrish, N.I., & Wyllie, D.C. (1996). Rock slope stability analysis. In: A.K. Turner & R.L. Schuster (Eds.), *Landslides: investigation and mitigation* (pp. 391-425). Transportation Research Board; National Research Council.

Zhu, M., McKenna, F., & Scott, M.H. (2018). OpenSeesPy: python library for the OpenSees finite element framework. *SoftwareX*, 7, 6-11. https://doi.org/10.1016/j.softx.2017.10.009.