

---

# **rootfinding**

***Release 1.0.3***

**Gabriel S. Gerlero**

**Sep 14, 2020**



# CONTENTS

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	API reference . . . . .	3
1.2	Changelog . . . . .	5
1.3	License . . . . .	6
<b>2</b>	<b>Indices and tables</b>	<b>7</b>
	<b>Python Module Index</b>	<b>9</b>
	<b>Index</b>	<b>11</b>



`rootfinding` finds roots by bisecting a bracketing interval until the value of the function can be considered sufficiently close to zero. The scheme differs slightly from the implementation of bisection in [SciPy](#): it is better suited for cases where a maximum acceptable residual is a more useful termination criterion than a tolerance for the argument variable, and can lead to fewer function evaluations overall.

The small library also contains functionality to automatically search for a bracket of a root when one is not known. While the library will always start by checking the inputs, it is possible to avoid redundant function calls by providing known values of the function—similarly, objects returned (and exceptions raised) by the library helpfully include function values that may be of interest. `rootfinding` was first developed as an internal module of our other project [Fronts](#).

To use this library, install `rootfinding` from [PyPI](#) as `$ pip install rootfinding`. Alternatively, you may copy the `rootfinding.py` file from the source code directly into your project (you must retain the copyright notice and license).



## CONTENTS

## 1.1 API reference

### 1.1.1 `bisect()`

`rootfinding.bisect(f, bracket, ftol=1e-12, maxiter=100, f_bracket=None, None)`

Find root of a function within a bracket using the bisection method.

The function must have opposite signs at the endpoints of the bracket.

Compared to SciPy's `scipy.optimize.bisect()` and `scipy.optimize.root_scalar()` functions, this function tests for a root by looking only at the residual (i.e., the value of  $f$ ).

#### Parameters

- **`f`** (*callable*) – Continuous scalar function.
- **`bracket`** (*sequence of two floats*) – An interval bracketing a root.  $f$  must have different signs at the two endpoints, or a `NotABracketError` will be raised. The endpoints do not need to be listed in order.
- **`ftol`** (*float, optional*) – Absolute tolerance for the value of  $f$  at the root. Must be nonnegative.
- **`maxiter`** (*int or None, optional*) – Maximum number of iterations. Must be nonnegative. An `IterationLimitReached` exception will be raised if the specified tolerance is not achieved within this number of iterations. If `None`, there is no maximum number of iterations.
- **`f_bracket`** (*sequence of two of {None, float}, optional*) – Values of  $f$  at the endpoints of `bracket`, if known (use `None` if a value is not known). For every known value, one fewer call to  $f$  will be required.

**Returns** **`result`** – Contains the root and the final bracket.

**Return type** `Result`

See also:

`bracket_root()` Search for a bracket.

## Notes

The function starts by testing the endpoints of the bracket. If a root is found at one of the endpoints of a valid bracket, no bisection iterations are performed and the root is immediately returned. If both endpoints qualify as roots, the one where the absolute value of  $f$  is lower is returned.

### 1.1.2 `bracket_root()`

`rootfinding.bracket_root` ( $f$ ,  $interval$ ,  $growth\_factor=2$ ,  $maxiter=100$ ,  $f\_interval=None$ ,  $None$ ,  $ftol=None$ )

Find an interval that brackets a root of a function by searching in one direction.

Starting from an interval, it moves and expands the interval in the direction of the second endpoint until the interval brackets a root of the given function.

#### Parameters

- **$f$**  (*callable*) – Continuous scalar function.
- **$interval$**  (*sequence of two floats*) – Starting interval. Must have non-equal endpoints, but they do not need to be listed in order. During the search, the interval will be shifted and expanded in the direction of `interval[1]`.
- **$growth\_factor$**  (*float, optional*) – Factor by which to grow the width of the working interval between iterations. Must be  $\geq 1$ .
- **$maxiter$**  (*int or None, optional*) – Maximum number of iterations. Must be nonnegative. An *IterationLimitReached* exception will be raised if the bracket is not found within the specified number of iterations. If *None*, there is no maximum number of iterations.
- **$f\_interval$**  (*sequence of two of {None, float}, optional*) – Values of  $f$  at the endpoints of the interval, if known (use *None* if a value is not known). For every known value, one fewer call to  $f$  will be required.
- **$ftol$**  (*None or float*) – An optional absolute tolerance for the value of  $f$  at a root. If given, the algorithm will immediately return any root it happens to discover in its execution.

**Returns** **result** – Normally contains a bracket and no root. However, if  $ftol$  is not *None* and a root is found, it will contain that root; in this case, the result will also include a bracket only if one was found at the same time as the root.

**Return type** Result

**See also:**

*bisect()*

## Notes

If  $ftol$  is not *None* and both endpoints of the starting interval qualify as roots, the one where the absolute value of  $f$  is lower is chosen as the root.



### 1.1.3 Exceptions

**exception** `rootfinding.NotABracketError`

Bases: `ValueError`

Exception raised by `bisect()` when the interval passed as *bracket* does not actually contain a root.

**f\_interval**

Values of the *f* at the endpoints of the interval that is not a bracket.

**Type** sequence of two floats

**function\_calls**

Number of calls to *f*.

**Type** int

**exception** `rootfinding.IterationLimitReached`

Bases: `RuntimeError`

Exception raised when a function in this module does not finish within the specified maximum number of iterations.

**interval**

Working interval at the time the iteration limit was reached.

**Type** sequence of two floats

**f\_interval**

Values of the *f* at the endpoints of *interval*.

**Type** sequence of two floats

**function\_calls**

Number of calls to *f*.

**Type** int

## 1.2 Changelog

This project adheres to [Semantic Versioning](#).

### 1.2.1 1.0.3 (2020-09-14)

- Fix use of wrong types in initialization of module exceptions.

### 1.2.2 1.0.2 (2020-06-16)

- Fix *growth\_factor* bug in *bracket\_root()*.
- Improve documentation of *bracket\_root()*.
- *bracket\_root()* now raises *ValueError* on an invalid *growth\_factor*.

### 1.2.3 1.0.1 (2020-06-15)

- `bracket_root()` now raises *ValueError* if passed a negative value as *ftol*.
- Improve descriptions of interval inputs in the API reference.

### 1.2.4 1.0.0 (2020-06-14)

First standalone version.

## 1.3 License

BSD 3-Clause License

Copyright (c) 2019-2020, Gabriel S. Gerlero All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### r

rootfinding, [1](#)



## INDEX

### B

`bisect()` (*in module rootfinding*), 3

`bracket_root()` (*in module rootfinding*), 4

### F

`f_interval` (*rootfinding.IterationLimitReached attribute*), 5

`f_interval` (*rootfinding.NotABracketError attribute*), 5

`function_calls` (*rootfinding.IterationLimitReached attribute*), 5

`function_calls` (*rootfinding.NotABracketError attribute*), 5

### I

`interval` (*rootfinding.IterationLimitReached attribute*), 5

`IterationLimitReached`, 5

### M

`module`  
    *rootfinding*, 1

### N

`NotABracketError`, 5

### R

`rootfinding`  
    *module*, 1