Bisection Method

MATLAB Implementation

Tamas Kis | kis@stanford.edu

TAMAS KIS
https://github.com/tamaskis

Copyright © 2021 Tamas Kis

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

LICENSE

Contents

bisection_method

Calculates the fixed-point of a univariate function using fixed-point iteration.

Syntax

```
root = bisection_method(f,a,b)
root = bisection_method(f,a,b,TOL)
root = bisection_method(f,a,b,[],imax)
root = bisection_method(f,a,b,TOL,imax)
root = bisection_method(__,'all')
```

Description

`root = bisection_method(f,a,b)` returns the root of a function $f(x)$ specified by the function handle `f`, where `a` and `b` define the initial guess for the interval $[a,b]$ containing the root. The default tolerance and maximum number of iterations are `TOL = 1e-12` and `imax = 1e6`, respectively.

`root = bisection_method(f,a,b,TOL)` returns the root of a function $f(x)$ specified by the function handle `f`, where `a` and `b` define the initial guess for the interval $[a,b]$ containing the root and `TOL` is the tolerance. The default maximum number of iterations is `imax = 1e6`.

`root = bisection_method(f,a,b,[],imax)` returns the root of a function $f(x)$ specified by the function handle `f`, where `a` and `b` define the initial guess for the interval $[a,b]$ containing the root and `imax` is the maximum number of iterations. The default tolerance is `TOL = 1e-12`.

`root = bisection_method(f,a,b,TOL,imax)` returns the root of a function $f(x)$ specified by the function handle `f`, where `a` and `b` define the initial guess for the interval $[a,b]$ containing the root, `TOL` is the tolerance, and `imax` is the maximum number of iterations.

`root = bisection_method(__,'all')` returns a vector, where the first element of this vector is the initial guess, all intermediate elements are the intermediate estimates of the root, and the last element is the converged root. This identifier `'all'` may be appended to any of the syntaxes used above.

Examples

Example 1

Find the root(s) of $f(x) = x^2 - 1$.

■ *SOLUTION*

Defining $f(x)$,

```
f = @(x) x^2-1;
```

We know $f(x) = x^2 - 1$ has roots at $x = \pm 1$, but let's pretend we don't know this, and solve this problem using a more general approach. Since $f(x)$ is a quadratic function, we know that it will have either 0 roots (in the case where $f(x)$ does not cross the x-axis) or 2 roots. Let's assume the latter case (otherwise it would be pointless to try and find roots of $f(x)$). Therefore, we use the bisection method twice, with two different guesses for the interval containing the root. Let's pick $[-10, 0]$ and $[0, 10]$ as our initial guesses for this interval.

```
% finds first root of f(x)=x^2-1 using the bisection method
root1 = bisection_method(f,-10,0)

% finds second root of f(x)=x^2-1 using the bisection method
root2 = bisection_method(f,0,10)
```

This yields the result

```
root1 =

   -1.0000


root2 =

    1.0000
```

Example 2

Find a root of $g(x) = h(m(x))$, where $h(x) = 5x^2 - 4$ and $m(x) = \cosh\sqrt{x}$. Additionally, plot the intermediate root estimates vs. iteration number.

■ *SOLUTION*

First, let's define $g(x)$. Instead of defining it as an anonymous function, we define it as a regular MATLAB function (note that we must either put this function in a separate `.m` file or place it at the end of the script).

```
function g = gx(x)
    m = cosh(sqrt(x));
    h = 5*m^2-4;
    g = h;
end
```

However, we cannot use `gx` directly with `bisection_method`. Instead, we first have to assign a function handle to `gx` (this allows us to pass the function to another function as an input parameter).

```
g = @(x) gx(x);
```

Due to the complexity of $g(x)$, we have no idea where its root(s) is/are[a]. Let's make the initial guess $[a, b] = [-5, 5]$. Solving for the root with the bisection method,

```
root = bisection_method(g,-5,5)
```

This yields the result

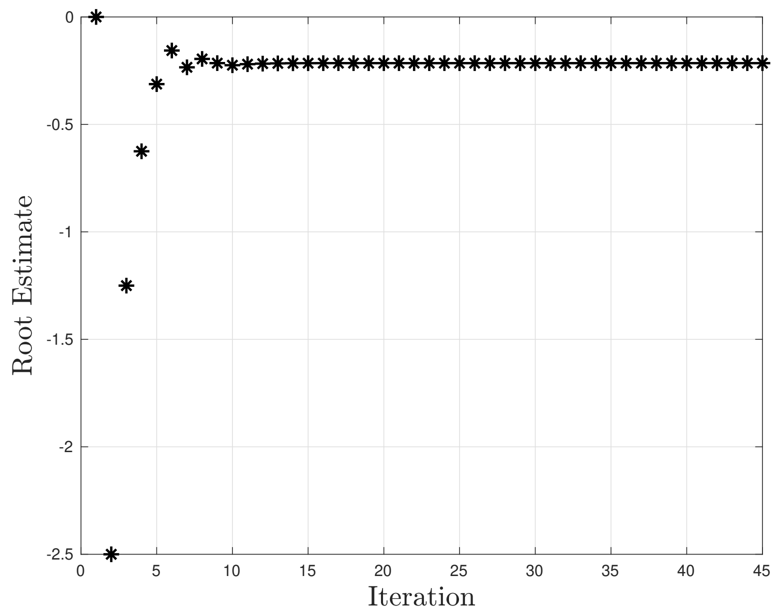
```
root =

   -0.2150
```

To plot the root estimates vs. iteration number,

```
% plots the intermediate root estimates
plot(bisection_method(g,-5,5,[],[],'all'),'k*','markersize',9,'linewidth'
    ,1.5);
grid on;
xlabel('Iteration','interpreter','latex','fontsize',18);
ylabel('Root Estimate','interpreter','latex','fontsize',18);
```

This yields the following plot:



[a] In fact, it has multiple roots. Here, we only solve for one.

Links

MATLAB® Central's File Exchange:
https://www.mathworks.com/matlabcentral/fileexchange/87042-bisection-method-bisection_method

GitHub®:
https://github.com/tamaskis/bisection_method-MATLAB

Bisection Method

The **bisection method** can be used to find the root of a univariate function $f(x)$, with no restrictions on the differentiability of f. The basic idea behind the bisection is starting off with some interval $[a, b]$ containing a root, iteratively "shrinking" this interval until it is below some tolerance threshold, and then taking the root to be the midpoint of this interval. The general procedure is as follows:

1. Make an initial guess for the interval $[a, b]$ containing the root.
2. Assume the root, c, is the midpoint of this interval: $c = (a + b)/2$.
3. Evaluate $f(a)$ and $f(c)$.
   (a) If $f(a) < 0$ and $f(c) > 0$ (i.e. they have different sign), we know the true root is contained in the interval $[a, c]$. Therefore, we update our interval so that a remains the same, but b is updated to be c.
   (b) If $f(a)$ and $f(c)$ have the same sign (either both negative or both positive), we know the true root must be contained in the interval $[c, b]$. Therefore, we update our interval so that b remains the same, but a is updated to be c.
4. Repeating steps 2 and 3, the interval $[a, b]$ will keep shrinking. Once the difference $(b - a)$ is small enough, we say that the estimate of the root has **converged** to the true root, within some **tolerance** (which we denote as TOL). Therefore, if we predetermine that, at most, the root must be restricted to an interval of length TOL, we will keep repeating steps 2 and 3 until $(b - a) < \mathrm{TOL}$.

In some cases, the difference $(b - a)$ may never decrease below TOL, or take too long to decrease to below TOL. Therefore, we also define the **maximum number of iterations** (i_{max}) so that the algorithm does not keep iterating forever, or for too long of a time [1, 2].

There are two basic algorithms for implementing the bisection method. The first implementation, shown in Algorithm 1 below, does *not* store the result of each iteration. On the other hand, the second implementation, shown in Algorithm 2, *does* store the result of each iteration. `bisection_method` implements both of these algorithms.

Since Algorithm 2 first needs to preallocate a potentially huge array to store all of the intermediate solutions, Algorithm 1 is significantly faster. Even if i_{max} (determines size of the preallocated array) is set to be a small number (for example, 10), Algorithm 1 is still faster. The reason we still consider and implement Algorithm 2 is so that convergence studies may be performed.

Algorithm 1: Bisection method.

```
Given: f(x), a, b, TOL, i_max
// sets initial guess
c = (a + b) / 2
// bisection method
i = 1
while (b − a) > TOL and i < i_max do
    // updates interval
    if f(c) = 0 then
        Stop
    else if sgn[f(c)] = sgn[f(a)] then
        a = c
    else
        b = c
    end
    // updates root estimate
    c = (a + b) / 2
    // increments loop index
    i = i + 1
end
// returns root
root = c
return root
```

Algorithm 2: Bisection method with intermediate root estimates.

Given: $f(x)$, a, b, TOL, $i_{\max}$

Preallocate an $i_{\max} \times 1$ vector $\mathbf{x}$, where $\mathbf{x}$ is the vector storing the estimates of the root at each iteration.

// inputs initial guess for root into $\mathbf{x}$ *vector*

$x_1 = \dfrac{a+b}{2}$

// bisection method

$i = 1$

while $\varepsilon >$ TOL ***and*** $i < i_{\max}$ **do**

    // updates interval

    if $f(x_i) = 0$ **then**

        Stop

    else if $\operatorname{sgn}[f(x_i)] = \operatorname{sgn}[f(a)]$ **then**

        $a = x_i$

    else

        $b = x_i$

    end

    // updates root estimate

    $x_{i+1} = \dfrac{a+b}{2}$

    // increments loop index

    $i = i + 1$

end

// stores intermediate root estimates (where last element of root *is the converged root)*

$\text{root} = (x_1, ..., x_i)^T$

// returns root

return root

References

[1] *Bisection method*. https://en.wikipedia.org/wiki/Bisection_method. (accessed: February 7, 2020).

[2] James Hateley. *Nonlinear Equations*. MATH 3620 Course Reader (Vanderbilt University). 2019.