Bisection Method

MATLAB Implementation

Tamas Kis | `kis@stanford.edu`

Tamas Kis
https://github.com/tamaskis

Contents

1 Download and Installation

$$c = \frac{a+b}{2}$$

1.1 Download from MATLAB Central's File Exchange

The `bisection_method` function is available for download on MATLAB® Central's File Exchange at https://www.mathworks.com/matlabcentral/fileexchange/87042-bisection-method-bisection_method.

1.2 Download from GitHub

The `bisection_method` function is available for download on GitHub® at https://github.com/tamaskis/bisection_method-MATLAB.

1.3 Files Included With Download

There are **five** files included in the downloaded `zip` file:

1. `Bisection Method - MATLAB Implementation.pdf` – *this PDF*
2. **`bisection_method.m`** – ***MATLAB function implementing the bisection method***
3. `EXAMPLES.M` – *examples for using the* `bisection_method` *function*
4. `LICENSE` – *license for the* `bisection_method` *function*
5. `README.md` – *markdown file for GitHub documentation*

1.4 Accessing the `bisection_method` Function in a MATLAB Script

There are **four** options for accessing the `bisection_method` function in a MATLAB script:

1. Copy the `bisection_method` function to the *end* of your MATLAB script.
2. Place the `bisection_method.m` file in the same folder as the MATLAB script.
3. Place the `bisection_method.m` file into whatever folder you want, and then use the `addpath(folderName)` command[1] where the `folderName` parameter is a string that stores the filepath of the folder that `bisection_method.m` is in *relative to* the folder that your script is in.
4. Make a toolbox by first opening `bisection_method.m`, then going to the `HOME` tab in MATLAB, and finally selecting `Package Toolbox` in the drop-down menu under `Add-Ons`. Once you package the `bisection_method` function as a toolbox, you can use it in any script.

[1] https://www.mathworks.com/help/matlab/ref/addpath.html

2 bisection_method

Calculates the root of a univariate function using the bisection method.

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 2.1

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 2.2

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

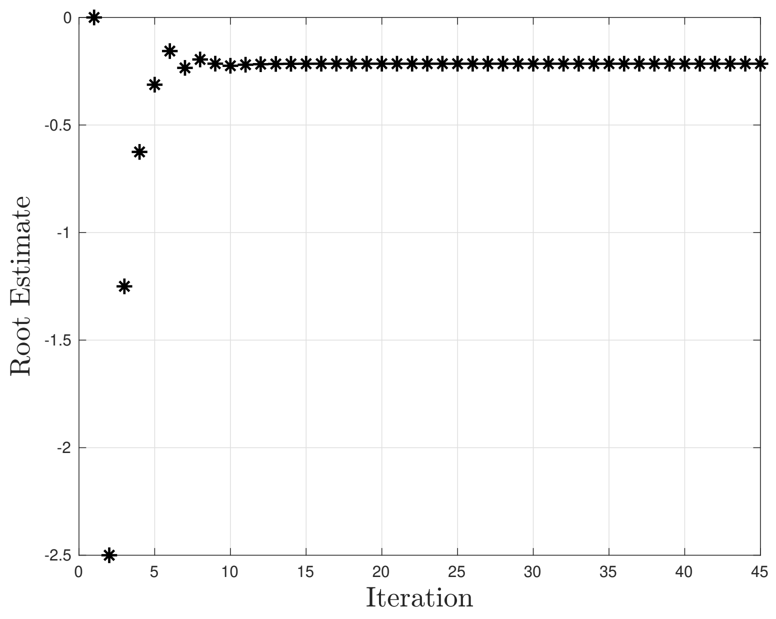
```
root =

   -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.

3 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 $(b - a) < \text{TOL}$.

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.

```
1  **Given:** $f(x)$, $a$, $b$, TOL, $i_{\max}$
   // sets initial guess
2  $c = \frac{a+b}{2}$
   // bisection method
3  $i = 1$
4  **while** $(b - a) > \mathrm{TOL}$ ***and*** $i < i_{\max}$ **do**
       // updates interval
5      **if** $f(c) = 0$ **then**
6          **Stop**
7      **else if** $\mathrm{sgn}[f(c)] = \mathrm{sgn}[f(a)]$ **then**
8          $a = c$
9      **else**
10         $b = c$
11     **end**
       // updates root estimate
12     $c = \frac{a+b}{2}$
       // increments loop index
13     $i = i + 1$
14 **end**
   // returns root
15 $\mathrm{root} = c$
16 **return** $\mathrm{root}$
```

Algorithm 2: Bisection method with intermediate root estimates.

```
1  Given: $f(x)$, $a$, $b$, TOL, $i_{\max}$
2  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
3  $x_1 = \frac{a+b}{2}$
   // bisection method
4  $i = 1$
5  while $\varepsilon > \mathrm{TOL}$ and $i < i_{\max}$ do
       // updates interval
6      if $f(x_i) = 0$ then
7          Stop
8      else if $\mathrm{sgn}[f(x_i)] = \mathrm{sgn}[f(a)]$ then
9          $a = x_i$
10     else
11         $b = x_i$
12     end
       // updates root estimate
13     $x_{i+1} = \frac{a+b}{2}$
       // increments loop index
14     $i = i + 1$
15 end
   // stores intermediate root estimates (where last element of root is the converged root)
16 $\mathrm{root} = (x_1, ..., x_i)^T$
   // returns root
17 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.