Secant Method

MATLAB Implementation

Tamas Kis | `kis@stanford.edu`

TAMAS KIS
https://github.com/tamaskis

Contents

1 Download and Installation

1.1 Download from MATLAB Central's File Exchange

The `secant_method` function is available for download on MATLAB® Central's File Exchange at https://www.mathworks.com/matlabcentral/fileexchange/85745-secant-method-secant_method.

1.2 Download from GitHub

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

1.3 Files Included With Download

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

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

1.4 Accessing the `secant_method` Function in a MATLAB Script

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

1. Copy the `secant_method` function to the *end* of your MATLAB script.
2. Place the `secant_method.m` file in the same folder as the MATLAB script.
3. Place the `secant_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 `secant_method.m` is in *relative to* the folder that your script is in.
4. Make a toolbox by first opening `secant_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 `secant_method` function as a toolbox, you can use it in any script.

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

2 secant_method

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

Syntax

```
root = secant_method(f,x0)
root = secant_method(f,x0,TOL)
root = secant_method(f,x0,[],imax)
root = secant_method(f,x0,TOL,imax)
```

Description

`root = secant_method(f,x0)` returns the root of a function $f(x)$ specified by the function handle `f`, where `x0` is an initial guess of the root. The default tolerance and maximum number of iterations are `TOL = 1e-12` and `imax = 1e9`, respectively.

`root = secant_method(f,x0)` returns the root of a function $f(x)$ specified by the function handle `f`, where `x0` is an initial guess of the root and `TOL` is the tolerance. The default maximum number of iterations is `imax = 1e9`.

`root = secant_method(f,x0,[],imax)` returns the root of a function $f(x)$ specified by the function handle `f`, where `x0` is an initial guess of the root and `imax` is the maximum number of iterations. The default tolerance is `TOL = 1e-12`.

`root = secant_method(f,x0,TOL,imax)` returns the root of a function $f(x)$ specified by the function handle `f`, where `x0` is an initial guess of the root, `TOL` is the tolerance, and `imax` is the maximum number of iterations.

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 secant method twice, with two different guesses. Let's pick -10 and 10 as our initial guesses.

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

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

This yields the result

```
root1 =

    -1


root2 =

     1
```

Example 2.2

Find a root of $g(x) = h(m(x))$, where $h(x) = 5x^2 - 4$ and $m(x) = \cosh\sqrt{x}$.

■ *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 `secant_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. Let's make the initial guess $x_0 = 5$. Solving for the root with the secant method,

```
root = secant_method(g,5)
```

This yields the result

```
root =

   -0.2150
```

From this example, we can see that the secant method is much more versatile than Newton's method. The secant method also allows us to find roots of functions that aren't strictly mathematical; that is, we could define a computational function that encapsulates an extremely complicated physical process, and as long as that process has a single input and a single output, we could find its root.

3 Secant Method

Newton's method is a root-finding technique that uses the derivative of a function to find its root[2]. Newton's method is defined iteratively as [2, Eq. (1)]

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{1}$$

But what if we don't know $f'(x)$? Then we need to approximate it using some numerical method. Specifically, for the secant method, we use the backward approximation of a derivative, given by Eq. (2) below.

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \tag{2}$$

This approximation can be visualized using the finite difference stencil shown in Fig. 1 below.

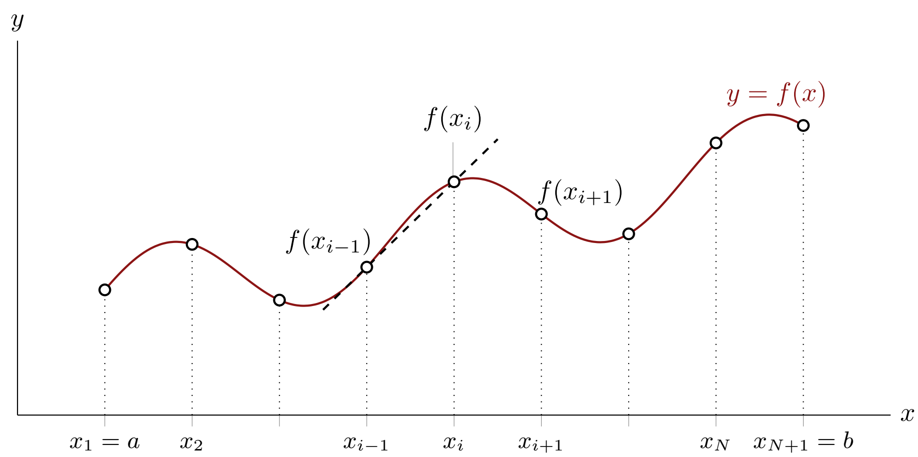

Figure 1: Backward approximation.

Substituting Eq. (2) into Eq. (1),

$$\begin{aligned} x_{i+1} = x_i - \left[\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}\right] f(x_i) &= \frac{[f(x_i) - f(x_{i-1})]\, x_i}{f(x_i) - f(x_{i-1})} - \frac{(x_i - x_{i-1})f(x_i)}{f(x_i) - f(x_{i-1})} \\ = \frac{x_i f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})} + \frac{x_{i-1} f(x_i) - x_i f(x_i)}{f(x_i) - f(x_{i-1})} &= \frac{x_i f(x_i) - x_i f(x_i) + x_{i-1} f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})} \end{aligned}$$

$$\boxed{x_{i+1} = \frac{x_{i-1} f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})}} \tag{3}$$

Equation (3) iteratively defines the **secant method**, which can be essentially thought of as a finite difference approximation of Newton's method for finding the root of a univariate function (based on an initial guess[3]). But how do we actually use Eq. (3)? Given an initial guess x_0, we can keep coming up with new estimates of the root. But how

[2] For a discussion/MATLAB implementation of Newton's method, see [2].

[3] Often, a function $f(x)$ will have multiple roots. Therefore, the secant method typically finds the root closest to the initial guess x_0. *However*, this is not always the case; the algorithm depends heavily on the derivative of $f(x)$, which, depending on its form, may cause it to converge on a root further from x_0.

do we know when to stop? To resolve this issue, we define the **error**[4] as

$$\boxed{\varepsilon = |x_{i+1} - x_i|} \tag{4}$$

Once ε 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, we can *tolerate* an error of TOL, then we will keep iterating Eq. (3) until $\varepsilon < \mathrm{TOL}$. In some cases, the error 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.

The algorithm itself is shown below in Algorithm 1 1. First, we have to make an initial guess x_0 for the root. Additionally, we need to set the root estimate at the second iteration (i.e. x_2) to a value slightly different than x_0 – otherwise, we will just have $x_{i+1} = x_i$ for all i and the algorithm will never "get started" (we can think of this has "kick-starting" the algorithm)[5] [1, 3, 4].

Algorithm 1: Secant method.

```
Given: f(x), x_0, TOL, i_max

// initializes the error so the loop will be entered
err = (2)(TOL)

Preallocate an i_max × 1 vector x, where x is the vector storing the estimates of the
  root at each iteration.

// inputs first and second guesses for root into x vector
x_1 = x_0
x_2 = 1.01 x_0

// secant method
i = 2
while ε > TOL and i < i_max do
    // updates estimate of root
    x_{i+1} = (x_{i-1} f(x_i) - x_i f(x_{i-1})) / (f(x_i) - f(x_{i-1}))

    // calculates error
    ε = |x_{i+1} - x_i|

    // increments loop index
    i = i + 1
end

// returns root
root = x_i
return root
```

[4] Note that ε is an *approximate* error. The motivation behind using this definition of ε is that as i gets large (i.e. $i \to \infty$), $x_{i+1} - x_i$ approaches $x_{i+1} - x^*$ (*assuming* this sequence is convergent), where x^* is the true root (and therefore $x_{i+1} - x^*$ represents the *exact* error).

[5] The alternative way to view this is by recalling that the derivative approximation is given by

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

If $x_i = x_{i-1}$, then the approximation to $f'(x_i)$ will become undefined, resulting in an error.

References

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

[2] Tamas Kis. *Newton's Method – MATLAB Implementation*. https://github.com/tamaskis/newtons_method-MATLAB. 2021.

[3] *Newton's method*. https://en.wikipedia.org/wiki/Newton%27s_method. (accessed: June 10, 2020).

[4] *Secant method*. https://en.wikipedia.org/wiki/Secant_method. (accessed: January 15, 2020).