Scientific Computing - Finding roots of function

Mateusz Pełechaty

23 October 2022

1 Exercise

Write a function that finds roots of function by using bisection method

Description

Bisection is an approximation method that tries to find a root of a function in a given interval. It works by repeatedly bisecting the interval in half and then selecting the subinterval in which the root must lie for further processing. This method is guaranteed to find a root if the given function is continuous and ends of given interval are of different signs. In it's principle it relies on **Intermediate value theorem (IVT)**

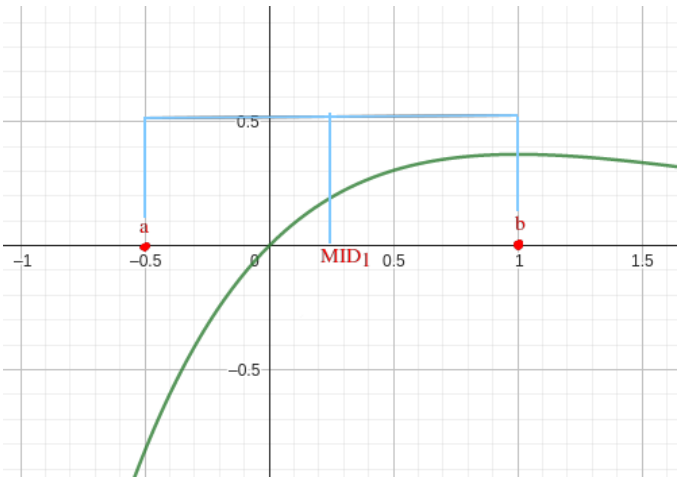

Figure 1: One iteration of bisection method for a = -0.5 and b = 1. The root is in the interval [-0.5, 0.25]

Solution and tests

Function with documentation can be found in *src/roots.jl*
Tests can run with *test/test_ roots.jl* and their definitions are in *test/utils.jl*

2 Exercise

Write a function that finds roots of function by using newton method

Description

Newton's method is a technique for finding roots of a functions by upgrading an initial guess to a better approximation of the root. It is based on the idea that if a function is differentiable, then a tangent line to the function at a given point can be used to find a better approximation of the root. For it to function properly we need to know the derivative of the function and the initial guess which should be close enough to the real root.

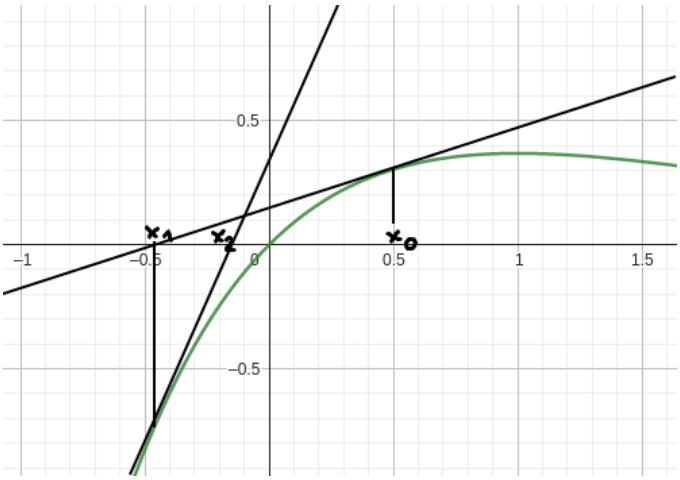

Figure 2: One iteration of newton method for a = -0.5 and b = 1. The root is in the interval [-0.5, 0.25]

Solution and tests

Function with documentation can be found in *src/roots.jl*
Tests can run with *test/test_roots.jl* and their definitions are in *test/utils.jl*

3 Exercise

Write a function that finds roots of function by using secant method

Description

The secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function. It starts by drawing a secant line between two points, and then uses the intersection of the line and the x-axis as the next approximation. Two starting approximations should also be chosen close enough to the real root. If they are too far apart, the method may fail to converge.

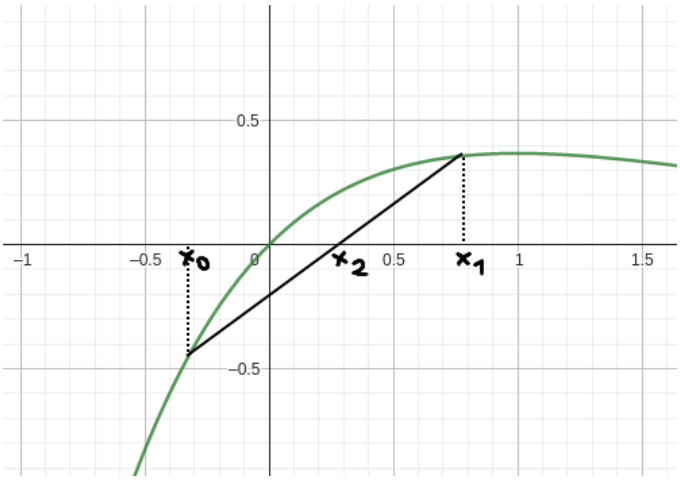

Figure 3: One iteration of secant method for a = -0.5 and b = 1. The root is in the interval [-0.5, 0.25]

Solution and tests

Function with documentation can be found in *src/roots.jl*
Tests can run with *test/test_roots.jl* and their definitions are in *test/utils.jl*

4 Exercise

Use previously calculated functions to find root of

$$f(x) = sinx - (\frac{1}{2}x)^2$$

Functions should be called with following arguments:

- Bisection method with start range of $[1.5, 2]$ and $\delta = \frac{1}{2}10^{-5}$ and $\epsilon = \frac{1}{2}10^{-5}$
- Newton's method with starting approximation of $x_0 = 1.5$ and $\delta = \frac{1}{2}10^{-5}$ and $\epsilon = \frac{1}{2}10^{-5}$
- Secant method with starting approximations of $x_0 = 1$, $x_1 = 2$ and $\delta = \frac{1}{2}10^{-5}$ and $\epsilon = \frac{1}{2}10^{-5}$

4.1 Solution and results

Solution can be found in *src/exercise4.jl*
Results can be found in **Table 1.** and **Figure 1.**

Method	**root**	$f(root)$	**iterations**	**error**
bisection	1.9337539672851562	-2.7027680138402843e-7	16	0
newton	1.933753779789742	-2.2423316314856834e-8	4	0
secant	1.933753644474301	1.564525129449379e-7	4	0

Table 1: Results of finding roots of $f(x) = sin(x) + (\frac{1}{2} \cdot x)^2$

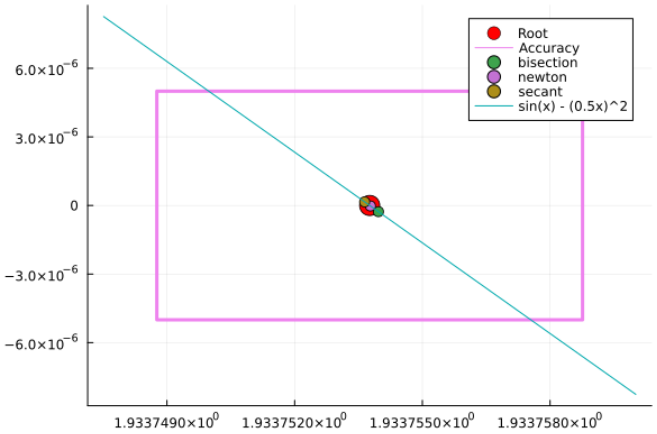

Figure 4: Results of finding roots of $f(x) = sin(x) + (\frac{1}{2} \cdot x)^2$. Real root was found on WolframAlpha

4.2 Conclusions

We can see that root finding methods work properly for $f(x) = sin(x) + (\frac{1}{2} \cdot x)^2$. For test data given as a example, **newton** method worked best, because it has found asked accuracy with least amount of iterations.

5 Exercise

With usage of bisection method, find cross point of functions $f(x) = 3x$, $g(x) = e^x$. Required accuracies: $\delta = 10^{-4}$, $\epsilon = 10^{-4}$

5.1 Solution and results

To compute cross points of f and g we are going to find roots of $f - g$. As there are 2 cross points, we need to find two roots.
Solution can be found in *src/exercise5.jl*
Results can be found in **Table 2.** and **Figure 2.** and **Figure 3.**

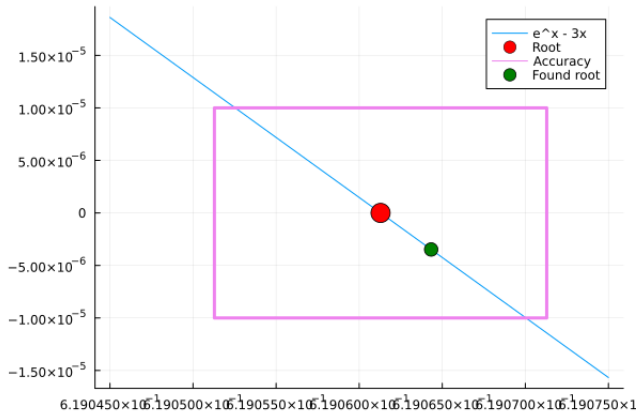

Figure 5: Found smaller root

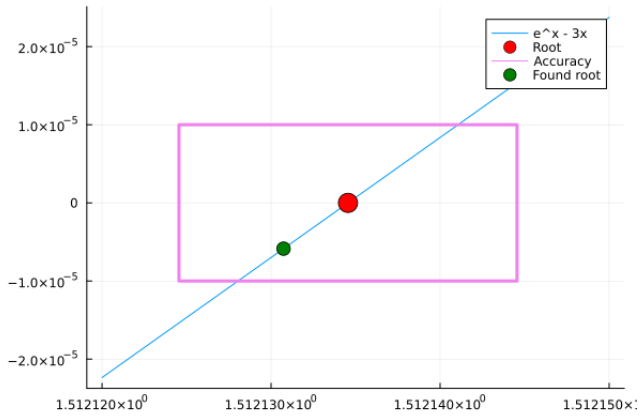

Figure 6: Found bigger root

start_range	end_range	found root	f(root)	iterations	error
0.0	1.0	0.6190643310546875	-3.4790879874790903e-6	16	0
1.0	2.0	1.5121307373046875	-5.86035312810651e-6	16	0

Table 2: Roots of $f(x) = e^x - 3x$ calculated by bisection method

6 Exercise

6.1 Task

Find root of $f_1(x) = e^{1-x} - 1$ and $f_2(x) = xe^{-x}$ by using bisection, Newton and secant methods. Required accuracies: $\delta = 10^{-5}$, $\epsilon = 10^{-5}$. Choose accordingly starting approximations

6.1.1 Analysis of functions

$f_1(x) = e^{1-x} - 1$ has got only one root: $x = 1$. For $x > 1$ we have $f_1(x) = e^{1-x} - 1 < e^0 - 1 = 0$. For $x < 1$ we have $f_1(x) = e^{1-x} - 1 > e^0 - 1 = 0$. Because of it we are going to use starting approximations that are close to 1.
$f_2(x) = xe^{-x}$ has got only one root: $x = 0$. For $x > 0$, both x and e^{-x} are positive, so it's product is also positive. For $x < 0$, $f_2(x) < 0$, because product of positive and negative number is negative. Because of it we are going to use starting approximation that are close to 0.

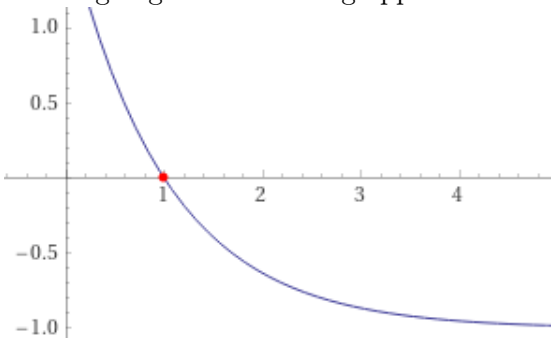

Figure 7: $f_1(x)$ drawn on WolframAlpha

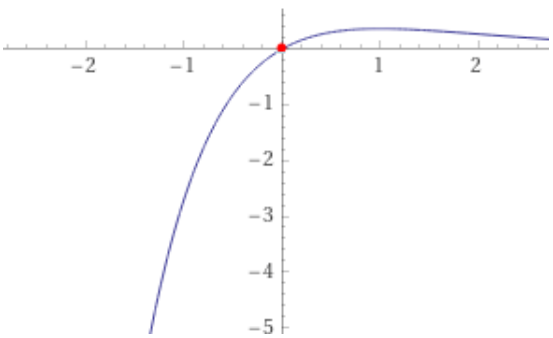

Figure 8: $f_2(x)$ drawn on WolframAlpha

6.1.2 Results of tests

function	start_range	end_range	delta/epsilon	found root	f(root)	iterations	error
f	0.0	2.0	1.0e-5	1.0	0.0	1	0
g	-1.0	1.0	1.0e-5	0.0	0.0	1	0

Table 3: Result of running bisection method on f and g

function	x_0	delta/epsilon	found root	f(root)	iterations	error
f	0.9	1.0e-5	0.999999999931772	6.822808984452422e-11	3	0
f	1.1	1.0e-5	0.99999999991094	8.906009263398573e-11	3	0
f	100.0	1.0e-5	100.0	-1.0	1	2
g	0.5	1.0e-5	-3.0642493416461764e-7	-3.0642502806087233e-7	5	0
g	1.1	1.0e-5	14.272123938290509	9.040322779745447e-6	3	0
g	1.0	1.0e-5	1.0	0.36787944117144233	1	2

Table 4: Result of running Newton method on f and g

function	x_0	x_1	delta/epsilon	found root	f(root)	iterations	error
f	0.9	1.1	1.0e-5	1.0000006354049569	-6.354047550338748e-7	3	0
g	0.1	0.2	1.0e-5	-1.387555849546545e-7	-1.3875560420776818e-7	4	0

Table 5: Result of running Secant method on f and g

6.1.3 Conclusions

All methods calculated roots properly if they were set on right starting approximations. Newton method for $x \geq 1$ has not given expected mathematical answer, but it worked as expected.

6.2 Question

What would happen if in Newton method, for f_1 we will choose $x_0 \in (1, \infty)$?

Answer

As we can see on **Table 4.**, if we choose $x_0 = 1.1$ for f_1, then we get good answer, but on the graph we can see that as $x \to \infty$ then $f'(x) \to 0$. Because of it, there exists such x, that $f'(x) < 1.0e - 18$, which is too small for Float64 precision, so $fl(f'(x)) = 0$ and we are going to get an error= 2 for newton method

6.3 Question

What would happen if in Newton method, for f_2 we will choose $x_0 > 1$

Answer

If we choose $x_0 > 1$ then Newton's method will try to find root by increasing x. It will find one, even though the only root is $x_0 = 1$, because $g(x)$ gets very close to 0. On **Table 4.** we can see example when $x_0 = 1.1$. Answer was $root \approx 14.2$ which is nowhere near expected mathematical answer, but it has given the right answer, because $f_2(14.2) \approx 0$ for the given delta. Look on the **Figure 6.** for visualisation for $x_0 = 1.1$ and different accuracies. Note that for accuracy big enough, $f_2'(x) = 0$ in Float64, so Newton method will give error= 2 as an output

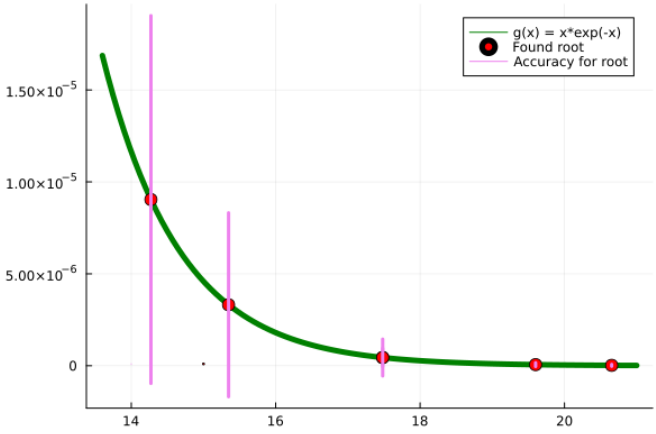

Figure 9: Roots of $f_2(x)$ found by Newton method. $x_0 = 1.1$, accuracies $= \{10^{-5}, \frac{1}{2}10^{-5}, 10^{-6}, \frac{1}{2}10^{-6}, 10^{-7}, \frac{1}{2}10^{-7}\}$

6.4 Question

Can i choose in Newton method, $x_0 = 1$ for f_2 ?

Answer

If we choose $x_0 = 1$ for f_2 then we are going to get error, because $f_2'(x_0) = 0$. We can also see this on **Table 4.**