CS3320 Module 6
Programming Assignment – Roots Hybrid

Description

Write a function, `zero(a, b, f)`, that takes an interval `[a,b]` containing a sign change in a function `f()`. This function will use a hybrid method based on False Position with the addition of Secant and Bisection methods to speed up the convergence rate while maintain the property of bracketing a root.

Procedure/Algorithm

For each iteration, this hybrid method will find the root as described below.

1. First find the point `c`, using the method of the False position.
2. Perform a secant search using the following rules:
    a. If `sign(f(a)) == sign(f(c))`, then find the point `d` where the secondary secant between `(a, f(a))` and `(c, f(c))` crosses the x-axis. `c` may fall outside the interval `[a, b]` when the function is flat near the root or when `a` and b are very close in distance. In other words, check whether `c<=a or c>=b`. In this case, perform a bisection using `[a, b]`.
    b. If, however, `sign(f(a)) ≠ sign(f(c))`, then use the line between `(b, f(b))` and `(c, f(c))` to find `d`.
    c. It is possible that `d` can fall outside of the open interval `(a,b)`; if so, resort to bisection. Otherwise, the new candidate interval is either `[c, d]` or `[d, c]` (depending on where they fall).
    d. If the length of `[c,d]` (or `[d,c]`) is not less than half of `[a,b]`, then do a bisection step before continuing your false position loop.

Exit Criteria

The algorithm will stop when one of the following criteria is met:

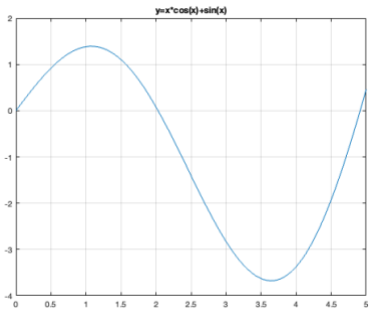
1. The algorithm finds a zero within 1 ulp from 0, i.e., `abs(f(c))` or `abs(f(d))` is within 1 ulp from 0. You should check every function evaluation to see if it lands within 1 ulp of zero.
2. The bisection step refines the interval to 1 ulp at the root. (`c*machine_epsilon`).

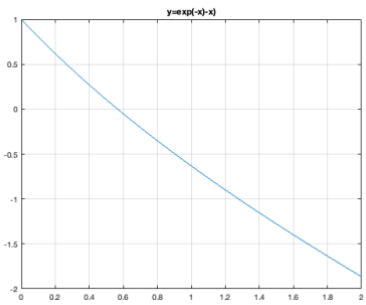
Whenever you exit, you can just return `c`. As you exit, print the number of function evaluations that were used in finding the root as well as `f(c)` (to see how close to zero it is). Design your algorithm so as to minimize the number of function evaluations performed.

Note: whenever you resort to bisection, always use the smallest current interval, according to the work you've already done (e.g., use `c` or `d` in place of `a` or `b`, as appropriate). Also, never evaluate the given function more than once for the same value of `x`.

Test/Validation

Test your function on $f(x) = x\cos x + \sin x$ (find the two smallest positive roots; note: 0 is *not* positive), and on $f(x) = e^{-x} - x$ (there is only one root). For output, print each root along with the number of function evaluations used, and the value of the function at the computed root (i.e., $f(r)$). The graphs of these two functions are shown below. You can use these graphs to choose your interval intervals. Use closest integers as the end points of your intervals.





If you do things correctly, you should use no more than 16 function evaluations for the first root and 9 for the last two roots.

Optimization

When the secant line is quite flat, c can end up outside the interval [a, b], bullet 2 in the "Procedure" section. If this happens, move c closer to the interval's endpoints by replacing your `c <= a or c >= b` check with the following code:

```
if (c <= a)
   c = a + eps*abs(a);  // Move 1-2 ulps past a
else if (c >= b)
   c = b - eps*abs(b);  // Move 1-2 ulps before b
if (c == a || c == b)
   break;                        // Bisect
```

This will speed things up, using only 11 iterations for the first root!