HW8 Problem 2

Austin Marga

In this homework problem, we will explore root-finding algorithms for a few given functions. The root-finding algorithms that I will be using are the Bisection Method and the Secant Method. I am curious to see their differences in accuracy because they are similar in implementation. At least more similar to eachother than either of these to Newton's Method.

The code for these root-finding functions can be found from the following GitHub or website:

- https://github.com/patrickwalls/mathematical-python/
- https://personal.math.ubc.ca/~pwalls/math-python/roots-optimization/newton/ The GitHub repository is provided under a Creative Commons License. All credit goes to Dr. Patrick Walls at UBC. The blog post is extremely well written and is worth checking out.

I have taken the different functions and placed them in a python file `roots.py`.

```
from roots import *
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
f = lambda x: np.tan(x)
a = -math.pi / 2
b = math.pi / 2
N = 2
bisection(f,a,b,N)
```

```
Found exact solution.
```

```
0.0
```

```
secant(f,a,b,N)
```

```
Found exact solution.
```

```
0.0
```

```
f = lambda x: np.tanh(x)
bisection(f,a,b,N)
```

```
Found exact solution.
```

```
0.0
```

```
secant(f,a,b,N)
```

```
Found exact solution.
```

```
0.0
```

Apparently, the functions used are powerful enough to find the root exactly. *Or is it?*

These functions were able to find exact solutions because of the *symmetric boundary conditions* of these odd functions. Let's try to throw a wrench into these functions by using non-symmetric boundary conditions and see how accurate these calculations really are.

We first have to define accuracy. The actual x values of these roots are $x = 0$, so we have to avoid dividing by zero. What I will do for this problem is to define accuracy as follows.

$$\text{acc} = \frac{|1 - |\text{experimental}||}{1} \times 100$$

With this unit 1 offset, we can now have meaningful accuracies.

y(x) = tan(x)

```
f = lambda x: np.tan(x)
a = -1.5
b = 1
N = 10
exp = bisection(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tan(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
For tan(x), the bisection method finds the root to be 0.000244140625.
Giving it an accuracy of 99.9755859375%.
The number of partitions is 10.
```

For only 10 partitions, that is a suprisingly good accuracy. Only .025% off of the actual value.

Let's try the secant method now.

```
f = lambda x: np.tan(x)
a = -1.5
b = 1
N = 10
exp = secant(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tan(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
For tan(x), the bisection method finds the root to be 0.16567442026935986.
Giving it an accuracy of 83.43255797306401%.
The number of partitions is 10.
```

The secant method seems to be significantly worse. How about the efficiency of the two functions? If you try out different values of N, you'll see that the 'accuracy' doesn't linearly increase with an increase in N. To try and see when the accuracy is rather high and trying to avoid tha anomalous successive guess, let's see the number of partition required to get an accuracy of 99.9999%.

```
f = lambda x: np.tan(x)
a = -1.5
b = 1
N = 18
exp = bisection(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tan(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
For tan(x), the bisection method finds the root to be 9.5367431640625e-07.
Giving it an accuracy of 99.99990463256836%.
The number of partitions is 18.
```

```
f = lambda x: np.tan(x)
a = -1.5
b = 1
N = 116
exp = secant(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tan(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
For tan(x), the bisection method finds the root to be 9.866221215570903e-07.
Giving it an accuracy of 99.99990133778785%.
The number of partitions is 116.
```

Summary for tan(x)

- The bisection method is extremely accurate, giving an accuracy of over 99.97% in only 10 partitions.
- The secant method is significantly less accurate, giving an accuracy of about 83.43% in 10 partitions.
- It took the bisection method only 18 partitions to get an accuracy of 99.9999%.
- It took the secant method 116 paritions to get an accuracy of 99.9999%.

y(x) = tanh(x)

Let's try the same thing for the function $tanh(x)$. We will use the same boundary conditions of $a = -1.5$ and $b = 1$ with the initial partition number of 10 and the target accuracy of 99.9999%.

```
f = lambda x: np.tanh(x)
a = -1.5
b = 1
N = 10
exp = bisection(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tanh(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
For tanh(x), the bisection method finds the root to be 0.000244140625.
Giving it an accuracy of 99.9755859375%.
The number of partitions is 10.
```

This is the same accuracy as the function $y(x) = tan(x)$. Let's try to get the accuracy of 99.9999% for the bisection method here.

```
f = lambda x: np.tanh(x)
a = -1.5
b = 1
N = 18
exp = bisection(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tanh(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
For tanh(x), the bisection method finds the root to be 9.5367431640625e-07.
Giving it an accuracy of 99.99990463256836%.
The number of partitions is 18.
```

We can see that the exact same situation happens as for the previous function.

Now, lets try the **secant method**. We will see some interesting results.

```
f = lambda x: np.tanh(x)
a = -1.5
b = 1
N = 10
exp = secant(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tanh(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
Found exact solution.
For tanh(x), the bisection method finds the root to be 0.0.
Giving it an accuracy of 100.0%.
The number of partitions is 10.
```

Due to the nature of the secant method, we can find an exact solution of $y = tanh(x)$. This works for many boundary conditions. The only requirement is that $a < 0$ and $b > 0$. If you try significantly more random boundary conditions, you may not get an exact solution. Let's try some random boundary conditions.

```
f = lambda x: np.tanh(x)
a = -203.5
b = 13.54
N = 10
exp = secant(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tanh(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
Found exact solution.
For tanh(x), the bisection method finds the root to be 0.0.
Giving it an accuracy of 100.0%.
The number of partitions is 10.
```

If we try boundary conditions too far, the function starts to break down.

```
f = lambda x: np.tanh(x)
a = -20252453.5
b = 1323452435.54
N = 10
exp = secant(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
```

```
print("For tanh(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
For tanh(x), the bisection method finds the root to be 86829.48839843692.
Giving it an accuracy of -8682848.839843692%.
The number of partitions is 10.
```

We can alleviate this by adding partitions.

```
f = lambda x: np.tanh(x)
a = -20252453.5
b = 1323452435.54
N = 100
exp = secant(f,a,b,N)
ab = np.absolute(exp)
acc = (1 - ab)*100
perc = str(acc) + "%"
print("For tanh(x), the bisection method finds the root to be {}.".format(exp))
print("Giving it an accuracy of {}%.".format(acc))
print("The number of partitions is {}.".format(N))
```

```
Found exact solution.
For tanh(x), the bisection method finds the root to be 0.0.
Giving it an accuracy of 100.0%.
The number of partitions is 100.
```

Given *enough partitions*, the secant method appears to be able to find the exact solution for tanh(x).

Summary for tanh(x)

- The bisection method is extremely accurate, giving an accuracy of over 99.97% in only 10 partitions.
- The secant method can find the exact solution for this function, the accuracy is 100%!
- It took the bisection method only 18 partitions to get an accuracy of 99.9999%.
- The secant method can find the exact root of this specific function given enough partions. Otherwise, it fails.