Homework #4

Name - Rakshya Sharma

Subject - ASTR-119

Assignment - Homework 4

Date - October 24, 2021

Purpose - The purpose of this assignment is to write a jupyter notebook to perform Bisection Search root finding.

References - Professor Brant's code from Thursday, October 14 lecture and session 8 slides, [https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hlines.html (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hlines.html)](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hlines.html) , [https://numpy.org/doc/stable/reference/generated/numpy.linspace.html (https://numpy.org/doc/stable/reference/generated/numpy.linspace.html)](https://numpy.org/doc/stable/reference/generated/numpy.linspace.html)

In [155]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Defining a function for which we are finding the roots.

In [156]:
```
def function_for_roots(x):
    a = 1.01
    b = -3.04
    c = 2.07
    return a*x**2 + b*x + c #get the roots of quadratic
```

Validate our initial bracket

In [157]:
```
def check_initial_values(f, x_min, x_max, tol):

    #check our initial guesses
    y_min = f(x_min)
    y_max = f(x_max)

    #check that xmin and xmax bracket a root
    if(y_min*y_max>0.0):
        print("No zero crossing found in the range = ", x_min, x_max)
        s = "f(%f) = %f, f(%f) = %f" % (x_min, y_min, x_max, y_max)
        print(s)
        return 0
    # if x_min is a root, then return flag ==1
    if(np.fabs(y_min)<tol):
        return 1
    # if x_max is a root, then return flag ==2
    if(np.fabs(y_max)<tol):
        return 2

    #if we reach this point, the bracket is valid

    return 3
```

Main work function

this work function does the bisection root finding a loop and then we will have a driving routine that calls this function to do the root planning.

In [158]:
```
def bisection_root_finding(f, x_min_start, x_max_start, tol):
    #this function use bisectio search to find a root of f

    x_min = x_min_start #minimum x in bracket
    x_max = x_max_start  #maximum x in bracket
    x_mid = 0.0 #mid point

    y_min = f(x_min)
    y_max = f(x_max)
    y_mid = 0.0

    imax = 10000 #max number of iterations
    i = 0 #iteration counter

    #check the initial values
    flag = check_initial_values(f, x_min, x_max, tol)

    if(flag==0):
        print("Error in bisection_root_finding")
        #raise a value error exception
        raise ValueError("Initial values invalid ", x_min, x_max)
    elif(flag==1):
        #got lucky
        return x_min
    elif(flag==2):
        #got lucky
        return x_max
    #if we reach here, then we conduct the search

    #set a flag
    flag = 1

    #enter a while loop
    while(flag):

        #set our mid point
        x_mid = 0.5*(x_min+x_max)
        y_mid = f(x_mid) #function at a x_mid

        #check if x_mid is a root
        if(np.fabs(y_mid)<tol):
            flag = 0
        else:
            #x_mid is not a root

            #if the product of the function at the midpoint
            #and at one of the end points is greater then
            #zero replace this end point

            if(f(x_min)*f(x_mid)>0):
                #replace x_min with x_mid
                x_min = x_mid
            else:
                #replace x_max with x_mid
                x_max = x_mid

        #print out the iteration
        print(x_min, f(x_min), x_max, f(x_max))

        #count the iteration
        i += 1

        #if we have exceed the max numbers
        # of iterations, exit

        if(i>= imax):
            print("Exceeded max number of iterations = ", i)
            s = "Min bracket f(%f) = %f" % (x_min, f(x_min))
            print(s)
            s = "Max bracket f(%f) = %f" % (x_max, f(x_max))
            print(s)
            s = "Mid bracket f(%f) = %f" % (x_mid, f(x_mid))
            print(s)
            raise StopIteration('Stopping iterations after ', i)

        #we are done
    return x_mid
```

Perform the search

In [159]:
```
x_min = 0.0
x_max = 1.5
tolerance = 1.0e-6

#print the initial guesses
print(x_min, function_for_roots(x_min))
print(x_max, function_for_roots(x_max))

#call the bisection root finding routine
#the root finding routine returns the root as its answer
x_root = bisection_root_finding(function_for_roots, x_min, x_max, tolerance)
y_root = function_for_roots(x_root)
s = "Root found with y(%f) = %f" %(x_root, y_root)
print(s)
```

```
0.0 2.07
1.5 -0.2175000000000007
0.75 0.3581249999999996 1.5 -0.2175000000000007
0.75 0.3581249999999996 1.125 -0.07171875000000005
0.9375 0.10769531249999975 1.125 -0.07171875000000005
1.03125 0.009111328124999485 1.125 -0.07171875000000005
1.03125 0.009111328124999485 1.078125 -0.033522949218749876
1.03125 0.009111328124999485 1.0546875 -0.012760620117187482
1.03125 0.009111328124999485 1.04296875 -0.0019633483886720704
1.037109375 0.0035393142700193003 1.04296875 -0.0019633483886720704
1.0400390625 0.0007793140411376243 1.04296875 -0.0019633483886720704
1.0400390625 0.0007793140411376243 1.04150390625 -0.0005941843986509987
1.040771484375 9.202301502186927e-05 1.04150390625 -0.0005941843986509987
1.040771484375 9.202301502186927e-05 1.0411376953125 -0.0002512161433698701
1.040771484375 9.202301502186927e-05 1.04095458984375 -7.963042706249368e-05
1.040863037109375 6.1878282573424315e-06 1.04095458984375 -7.963042706249368e-05
1.040863037109375 6.1878282573424315e-06 1.0409088134765625 -3.6723415833161965e-05
1.040863037109375 6.1878282573424315e-06 1.0408859252929688 -1.5268322895334308e-05
1.040863037109375 6.1878282573424315e-06 1.0408744812011719 -4.540379595852073e-06
1.040863037109375 6.1878282573424315e-06 1.0408744812011719 -4.540379595852073e-06
Root found with y(1.040869) = 0.000001
```

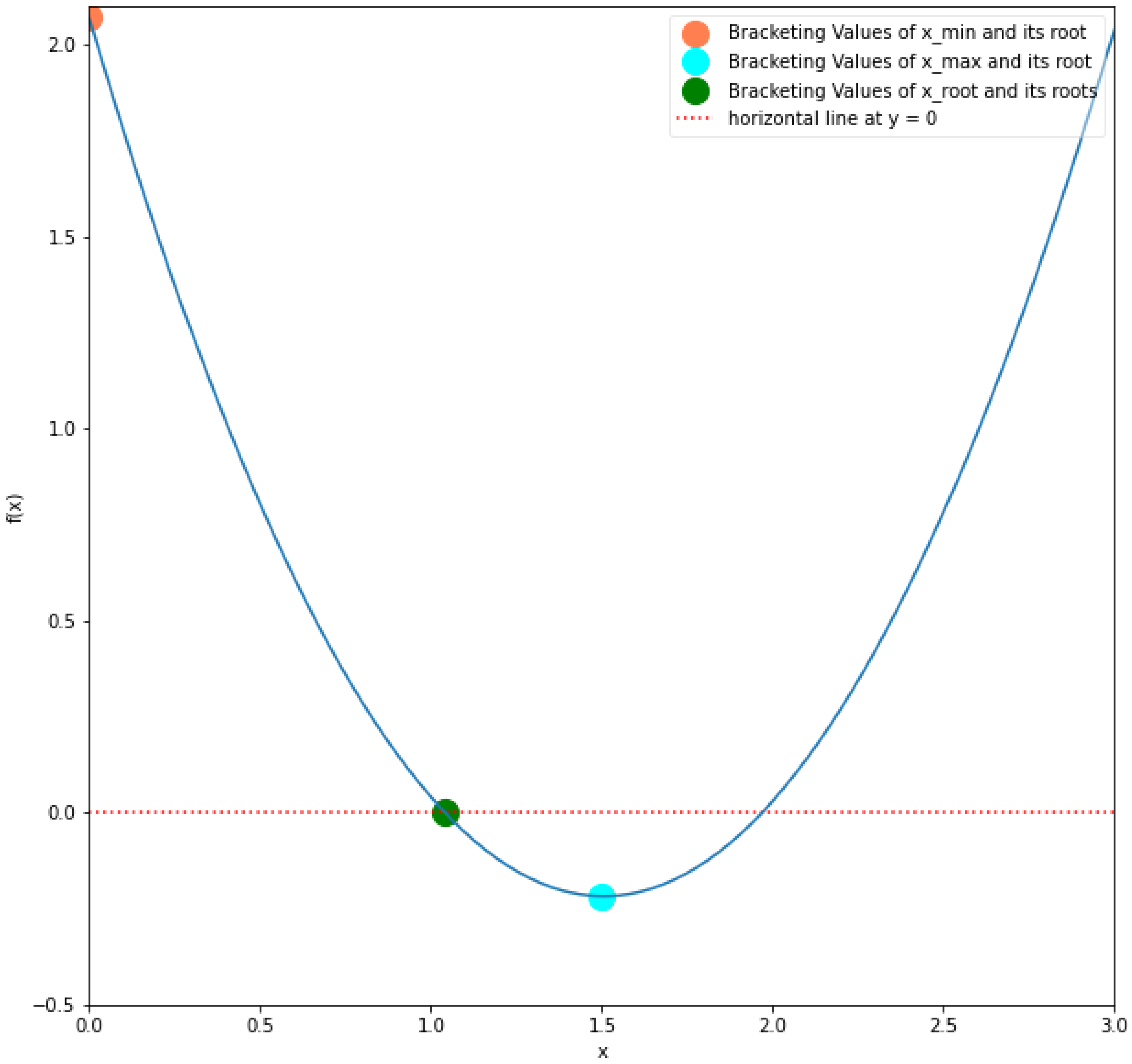
We can see our first bracket we have two values, x_min as 0 and parabola is 2.07. In the next bracket we have x_min = 1.5 and parabola equals -0.2175000000000007. These are our initial guesses.

Then in the third line we can see the iteration, we are halfway between 0.0 and 1.5. Our new bracket is 0.75 and value of parabola is 0.3581249999999996 whereas the value of parabola at 1,5 is still -0.2175000000000007.

Then again in next iteration we have pulled in the maximum value from 1.5 to 1.25 where the value of parabola -0.07171875000000005 and the miminum value as 0.9375 with the value of parabola as 0.10769531249999975.

Here we continue to alternate to shrink the bisection search bracket to the point where the midpoint is within the tolerance in the end to the point where we want. The method took sixteen iterations to converge.

In [160]:
```
x = np.linspace(0, 3, 1000)# returns a row vector of 1000 evenly spaced points between x and f(x).

fig = plt.figure(figsize=[10,10]) # sets the figure size to 10 x 10

plt.xlim([0,3]) # sets the x range to [0,3]
plt.ylim([-0.5,2.1])  # sets the y range to [-0.5,2.1]
plt.plot(x,function_for_roots(x)) # f(x)= 1.01*x**2 -3.04*x +2.07

#Using the plt.scatter to plot the points
plt.scatter(x_min,function_for_roots(x_min),s = 200, c ='coral', label = 'Bracketing Values of x_min and its root'
)

plt.scatter(x_max,function_for_roots(x_max),s = 200, c ='aqua',label = 'Bracketing Values of x_max and its root')

plt.scatter(x_root,y_root,s = 200, c ='green',label = 'Bracketing Values of x_root and its roots')

#plotting the horizontal line at y= 0
plt.hlines(y = 0, xmin = 0, xmax = 3, colors='red', linestyles='dotted', label = 'horizontal line at y = 0')


plt.xlabel('x') # labels the x axis
plt.ylabel('f(x)') # labels the y axis
plt.legend(loc = 1, framealpha = 0.4) #semi transparent

plt.show()
```



In [ ]: