ASTR 119: Session 8
Root finding
Bisection & Newton-Raphson

Outline

1) Homework due 10/29 at 8:00am
2) Visualization of the Day
3) Root finding: Bisection Search
4) Root finding: Newton-Raphson
5) Save your work to GitHub

Homework, due Oct 29, 8:00am

1) Write a jupyter notebook to perform Bisection Search root finding. Numerically find the two roots of the function:

$$f(x) = 1.01x^2 - 3.04x + 2.07$$

Use a tolerance of 1.0e-6 for the allowed deviation of f(x) from 0.

2) Given your starting guesses for the bracketing values around the roots, how many iterations does your method take to converge?

3) Have your notebook make a plot of f(x) vs. x as a line, and indicated with differently colored points your initial bracketing values and the roots. In the plot, use limits of x=[0,3] and y=[-0.5, 2.1]. Add a horizontal line at y=0. Plot f(x) at a 1000 evenly spaced values of x=[0,3].

4) Create an issue for your repository and tag your TA. CLEAR ALL THE CELLS BEFORE YOU COMMIT THE NOTEBOOK.

5) Your TA will clone your code and email you commented version of the code and a grade. To get the full grade possible, all the notebooks will need to run to completion without errors and produce the requested plots.

6) Call the repository “astr-119-hw-4” and the notebook “hw-4.ipynb”.

Algorithm for Bisection method

1. Declare variables.

2. Set maximum number of iterations to perform.

3. Set tolerance to a small value (eg. 1.0e-6). 4. Set the two initial bracket values.
   (a) Check that the values bracket a root or singularity.
   (b) Determine value of function fnct at the two bracket values.
   (c) Make sure produce of functional values is less than 0.0. If not, then report this and stop.
   (d) If the absolute value of one of the functional values is less than tolerance, then a root is found and write value to terminal and stop.

5. Set the counter of the number of iterations to zero. 6. Begin Bisection loop
   (a) Find value midway between bracket values.
   (b) Determine functional value at this midpoint.
   (c) If the absolute value of function value at midpoint is less than tolerance, then exit Bisection loop.
   (d) If produce of functional values at midpoint and at one of the endpoints is greater than zero, then replace this endpoint and its functional value with midpoint and its functional value.
   (e) Otherwise, replace the other endpoint and its functional value with midpoint and its functional value.
   (f) Increment the count of the number of iterations.
   (g) If we have exceeded the maximum number of iterations, then exit Bisection loop.

7. End Bisection Loop

8. If root was not found in maximum number of iterations, write a warning message to the terminal.

9. Write to screen the value of root

Function funct: Given 1 argument (type float):

1. Declare any additional variables.
2. Calculate value of function at the given point.
3. Return value as a float.

Bisection Search

jupyter bisection_search_demo Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Help

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Define a function for which we'd like to find the roots

In [2]:
```
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 ax^2 + bx + c
```

Bisection Search

We need a function to check whether our initial values are valid

```
In [9]: 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 x_min and x_max contain a zero crossing
            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
            #and we will return 3
            return 3
```

Bisection Search

Now we will define the main work function that actually performs the iterative search

```
def bisection_root_finding(f, x_min_start, x_max_start, tol):

    # this function uses bisection search to find a root

    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)  #function value at x_min
    y_max = f(x_max)  #function value at x_max
    y_mid = 0.0       #function value at mid point

    imax = 10000      #set a maximum 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 ValueError('Initial values invalid',x_min,x_max)
    elif(flag==1):
        # lucky guess
        return x_min
    elif(flag==2):
        # another lucky guess
        return x_max

    #if we reach here, then we need to conduct the search
```

Bisection Search

```
#if we reach here, then we need to conduct the search

#set a flag
flag = 1

#enter a while loop
while(flag):
    x_mid = 0.5*(x_min+x_max) #mid point
    y_mid = f(x_mid)         #function value at 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 than
        #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))
```

UC SANTA CRUZ

Bisection Search

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

        #count the iteration
        i += 1

        #if we have exceeded the max number
        #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
```

Bisection Search

Perform the search

In [57]:

```python
x_min =  0.0
x_max =  1.5
tolerance = 1.0e-6

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

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 -0.001963348388672070
1.037109375 0.0035393142700193003 1.04296875 -0.001963348388672070
1.0400390625 0.0007793140411376243 1.04296875 -0.001963348388672070
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
1.04086303710937 6.187828257342431e-06 1.04095458984375 -7.963042706249368e-05
1.04086303710937 6.187828257342431e-06 1.0409088134765625 -3.672341583161965e-05
1.04086303710937 6.187828257342431e-06 1.0408859252929688 -1.526832289533430e-05
1.04086303710937 6.187828257342431e-06 1.0408744812011719 -4.540379595852073e-06
1.04086303710937 6.187828257342431e-06 1.0408744812011719 -4.540379595852073e-06
Root found with y(1.040869) = 0.000001
```

Newton-Raphson Root Finding

ROOT FINDING: NEWTON-RAPHSON

Consider the Taylor series approximation of f(x) about position x_i:

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2!}f''(x_i) + \ldots$$

ROOT FINDING: NEWTON-RAPHSON

Consider the Taylor series approximation of f(x) about position x_i:

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2!}f''(x_i) + \ldots$$

We want to find where f(x)=0. We can make a guess at the value of the root x by using the Taylor expansion to linear order to find an approximation to the root, x_{i+1}:

$$\tilde{f}(x) = f(x_i) + (x - x_i)f'(x_i)$$

ROOT FINDING: NEWTON-RAPHSON

Consider the Taylor series approximation of f(x) about position x_i:

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2!}f''(x_i) + \ldots$$

We want to find where f(x)=0. We can make a guess at the value of the root x by using the Taylor expansion to linear order to find an approximation to the root, x_{i+1}:

$$\tilde{f}(x) = f(x_i) + (x - x_i)f'(x_i)$$

IF $\tilde{f}(x_{i+1}) = 0$ **THEN** $x_{i+1} = x_i - f(x_i)/f'(x_i)$

ROOT FINDING: NEWTON-RAPHSON

Consider the Taylor series approximation of f(x) about position x_i:

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2!}f''(x_i) + \ldots$$

We want to find where f(x)=0. We can make a guess at the value of the root x by using the Taylor expansion to linear order to find an approximation to the root, x_{i+1}:

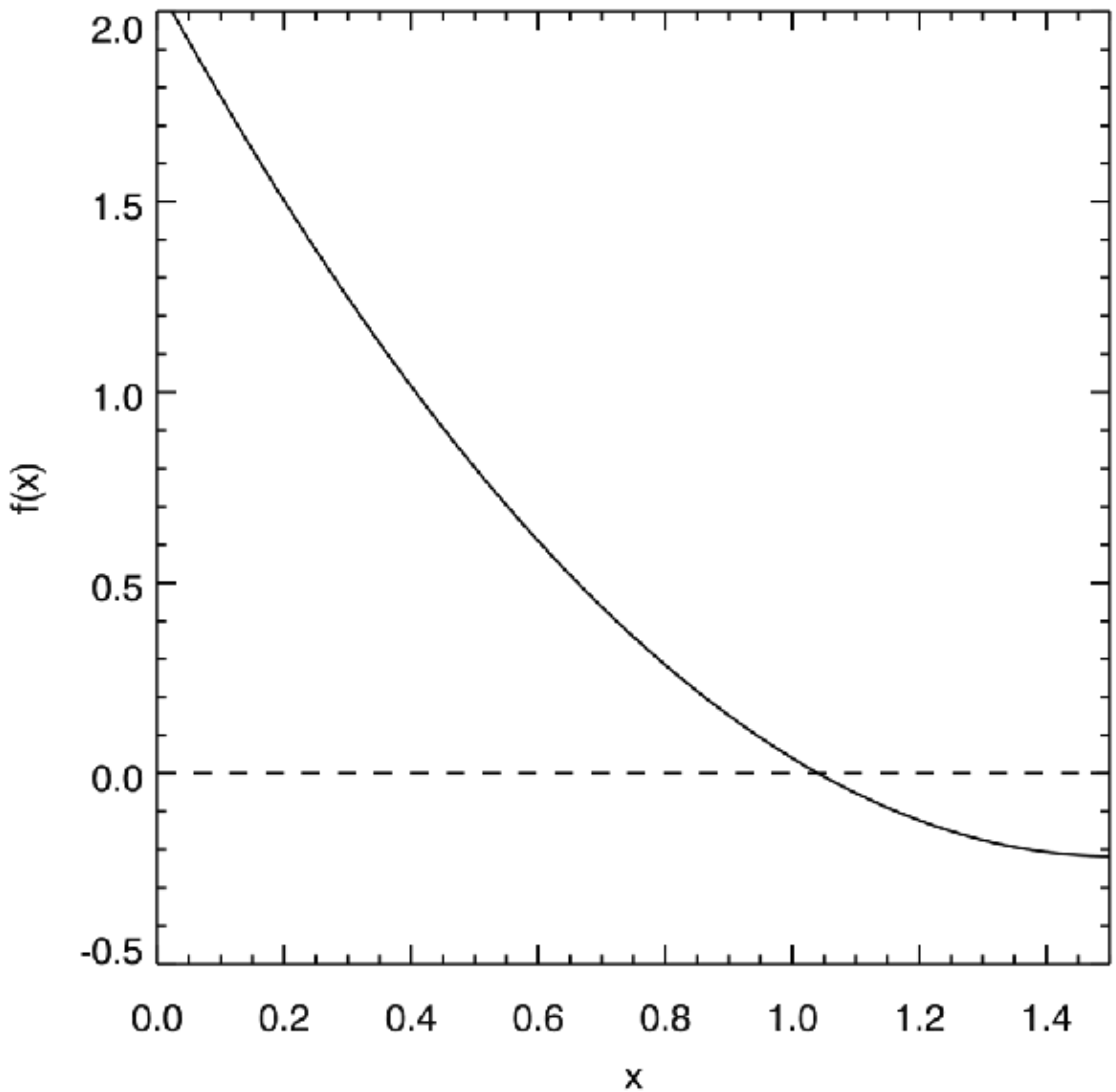
$$\tilde{f}(x) = f(x_i) + (x - x_i)f'(x_i)$$

IF $\tilde{f}(x_{i+1}) = 0$ **THEN** $x_{i+1} = x_i - f(x_i)/f'(x_i)$

We can then replace x_i with x_{i+1}, and try again until f(x_i) = 0.

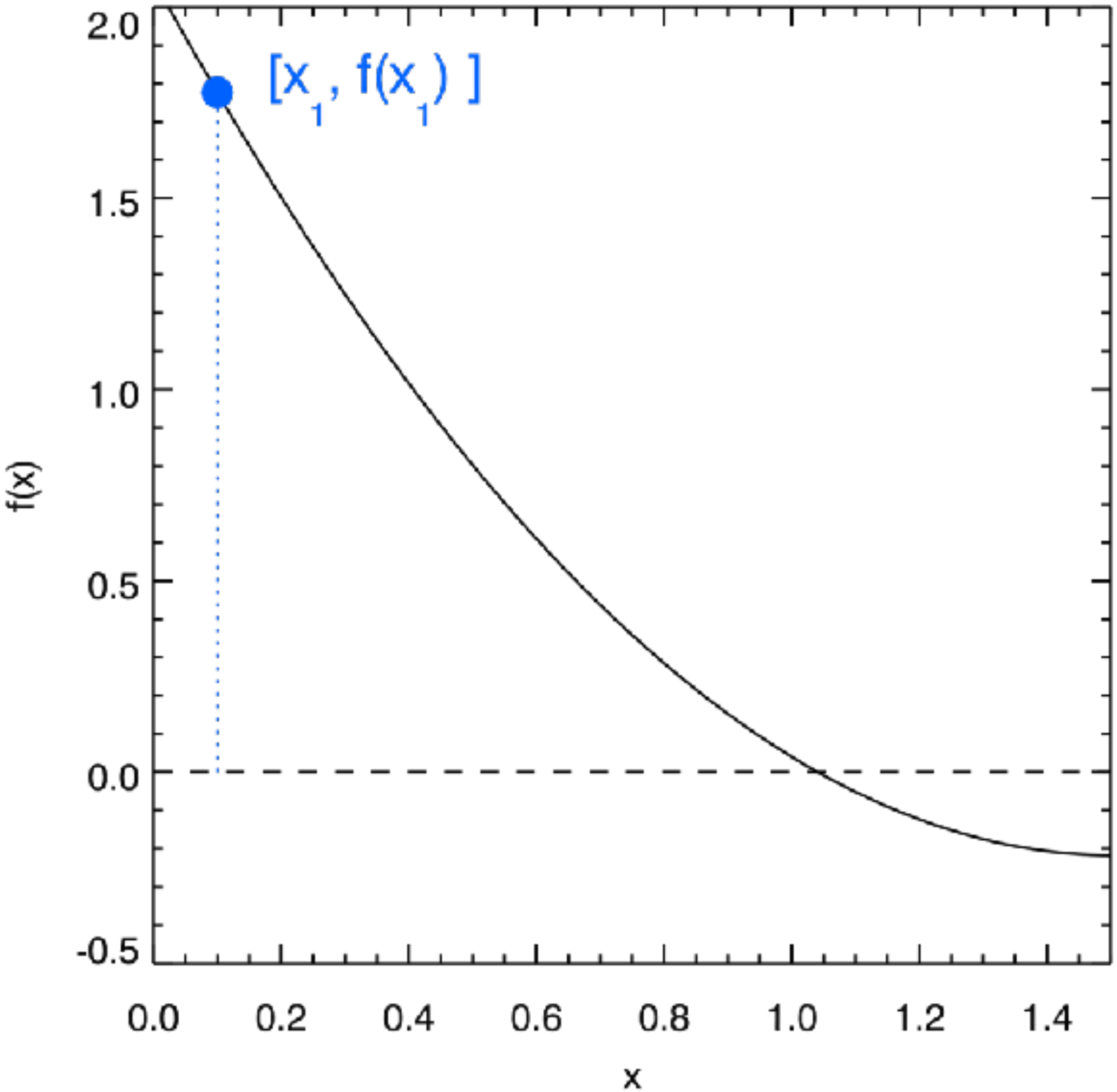
ROOT FINDING:

Graphically, the Newton-Raphson method looks like:



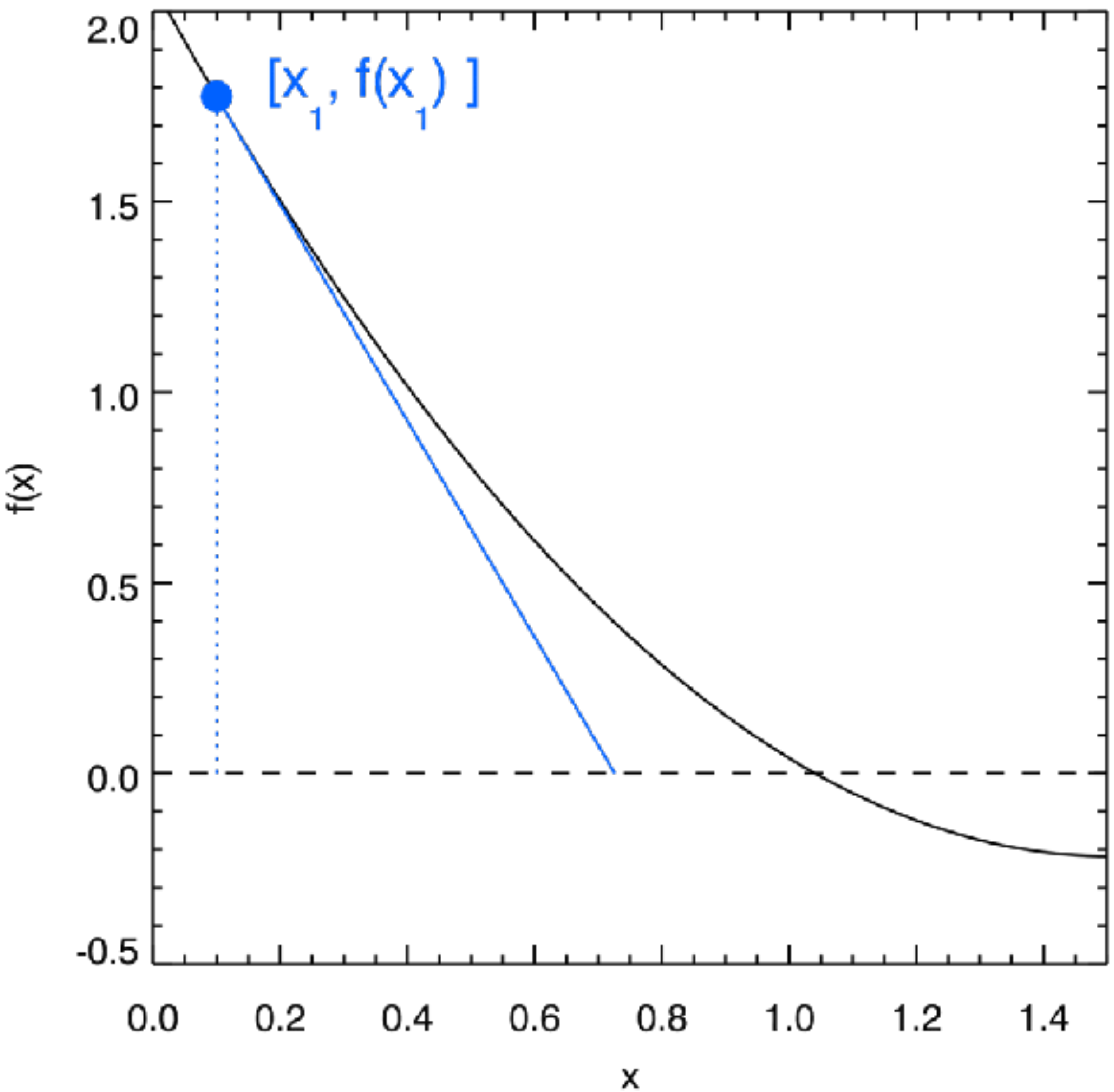
ROOT FINDING:

Graphically, the Newton-Raphson method looks like:



ROOT FINDING:

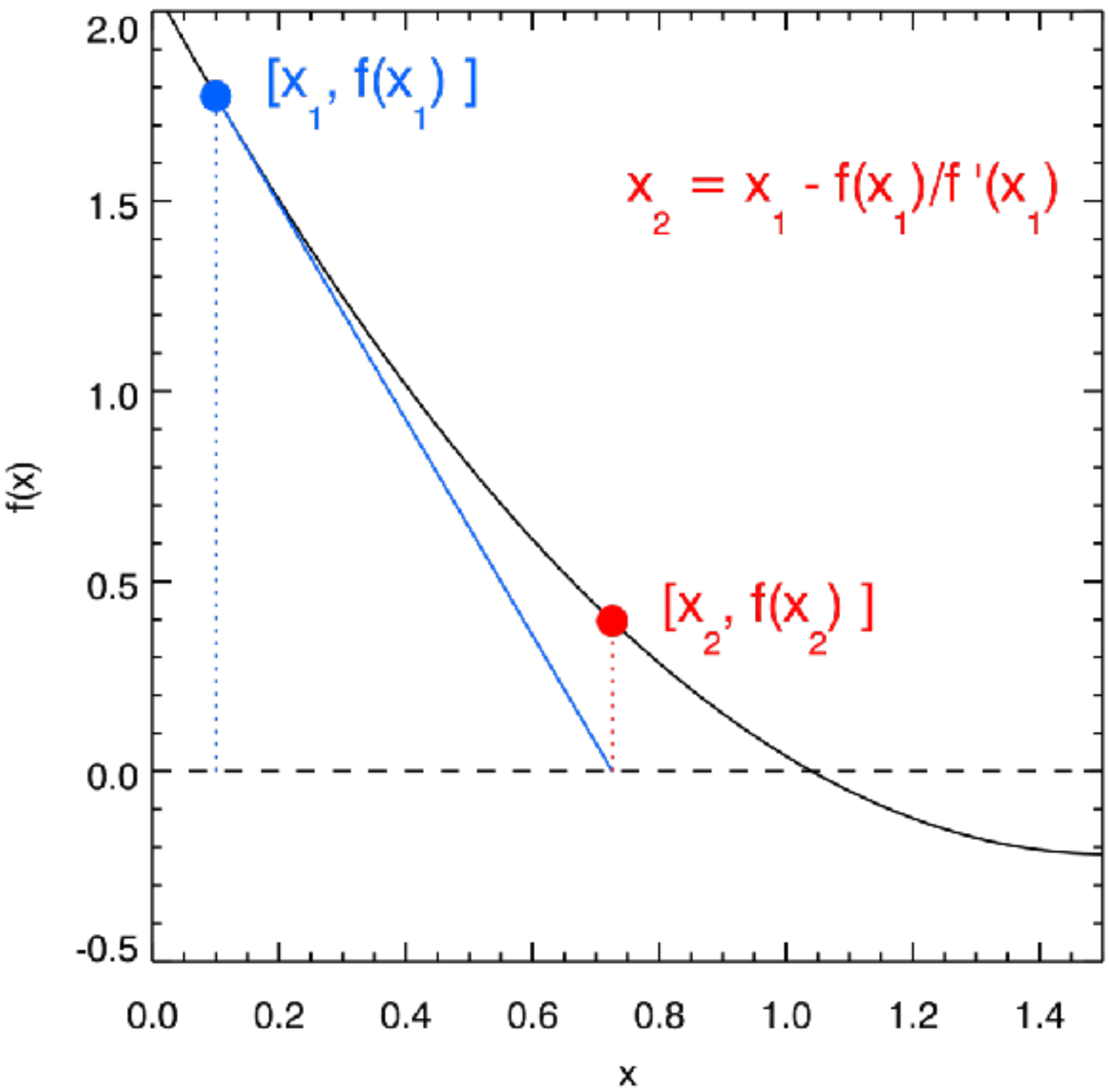
Graphically, the Newton-Raphson method looks like:



UC SANTA CRUZ

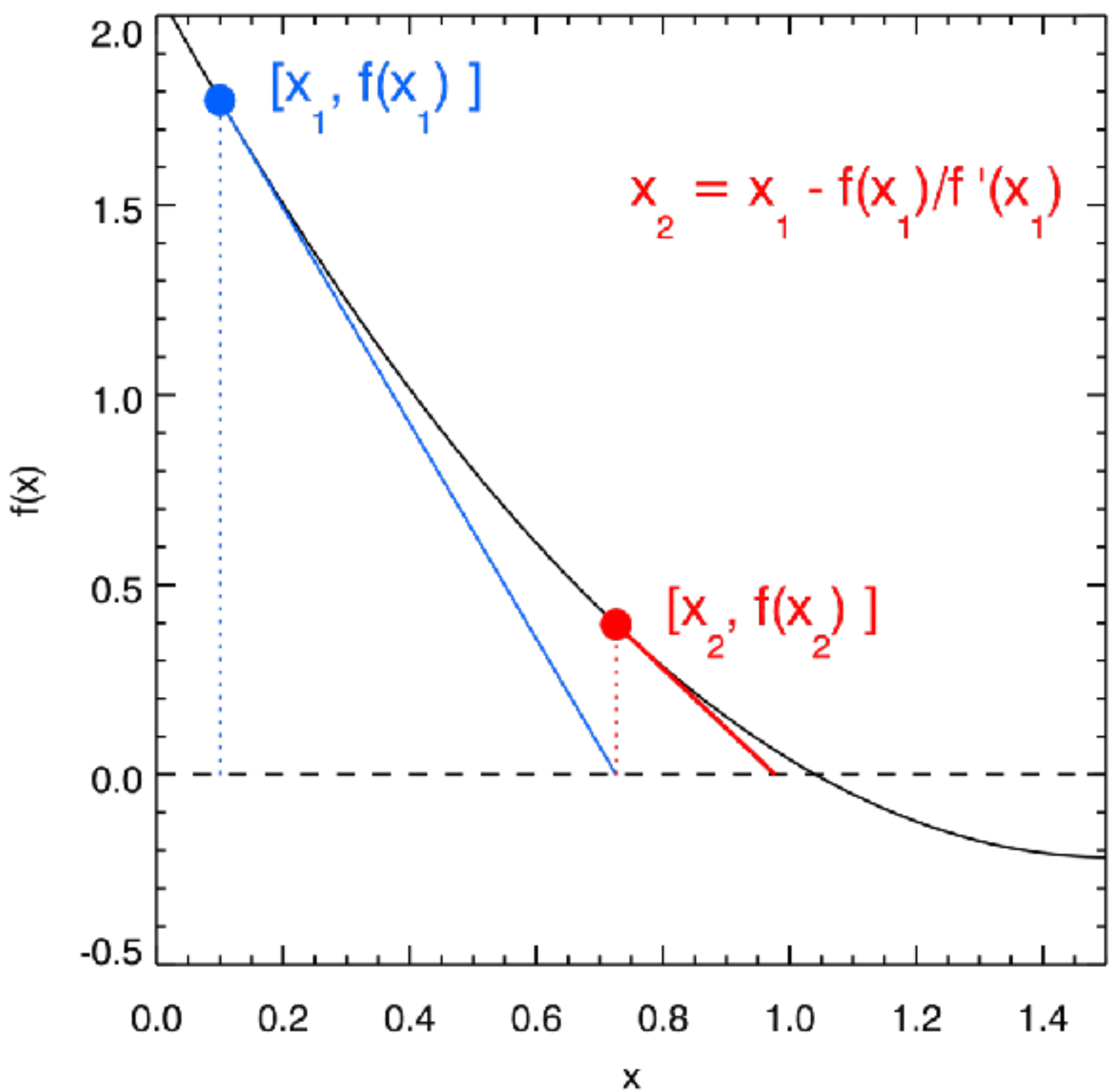
ROOT FINDING:

Graphically, the Newton-Raphson method looks like:



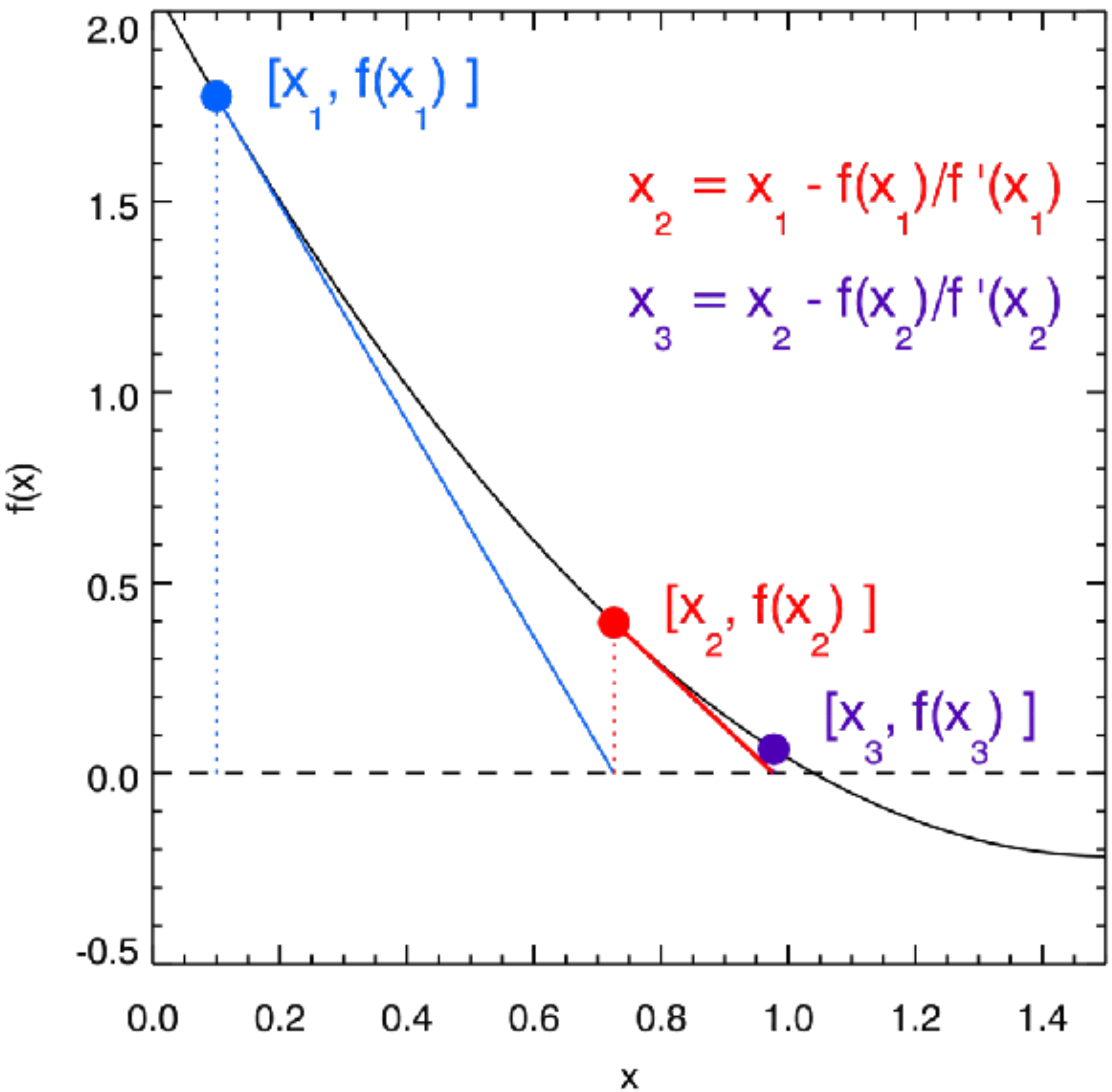
ROOT FINDING:

Graphically, the Newton-Raphson method looks like:



ROOT FINDING:

Graphically, the Newton-Raphson method looks like:



ALGORITHM FOR NEWTON-RAPHSON

Function fnct: Given 1 argument (type float):
1. Declare any additional variables.
2. Calculate value of function at the given point.
3. Return value as a float.

Function fnct_prime: Given 1 argument (type float):
1. Declare any additional variables.
2. Calculate the value of derivative of the function at the given point.
3. Return value as a float.

ALGORITHM FOR NEWTON-RAPHSON

1. Declare variables.

2. Set maximum number of iterations to perform.

3. Set tolerance to a small value (eg. 1.0e-6).

4. Set the initial guess. Set an iteration counter to zero.

5. Begin Newton-Raphson loop
   (a) Find next guess via new_root = root - fnct(root)/fnct_prime(root)
   (b) If the absolute value of fnct(new_root) is less than tolerance, then a root is found and write value to terminal and stop.
   (c) Increment the count of the number of iterations.
   (d) If we have exceeded the maximum number of iterations, then stop Newton-Raphson loop.

6. If root was not found in maximum number of iterations, write a warning message to the terminal.

7. Write to terminal the value of root and number of iterations performed.

Newton-Raphson

A Newton-Raphson Root Finding Implementation

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Define the function for root finding

```
def function_for_root(x):
    a =  1.01
    b = -3.04
    c =  2.07
    return a*x**2 + b*x + c
```

Define the function's derivative

```
def derivative_for_root(x):
    a =  1.01
    b = -3.04
    return 2*a*x + b
```

Newton-Raphson

Define the primary work function

```python
def newton_raphson_root_finding(f, dfdx, x_start, tol):

    # this function uses newton-raphson search to find a root

    #set a flag
    flag = 1

    #set a maximum number of iterations
    imax = 10000

    #start a counter
    i = 0

    #define the new and old guesses
    x_old = x_start
    x_new = 0.0
    y_new = 0.0
```

Newton-Raphson

```
    #start the loop
    while(flag):

        #make a new guess
        x_new = x_old - f(x_old)/dfdx(x_old)

        #print out the iteration
        print(x_new,x_old,f(x_old),dfdx(x_old))

        #if the abs value of the new function value
        #is < tol, then stop
        y_new = f(x_new)
        if(np.fabs(y_new)<tol):
            flag = 0 #stop the iteration
        else:
            #save the result
            x_old = x_new
            #increment the iteration
            i += 1

        if(i>=imax):
            printf("Max iterations reached.")
            raise StopIteration('Stopping iterations after ',i)


    #we are done!
    return x_new
```

Newton-Raphson

Perform the search

```
x_start = 0.5
tolerance = 1.0e-6

#print the initial guess
print(x_start,function_for_root(x_start))

x_root = newton_raphson_root_finding(function_for_root,derivative_for_root,x_start,tolerance)
y_root = function_for_root(x_root)

s = "Root found with y(%f) = %f" % (x_root,y_root)
print(s)
```

Newton-Raphson

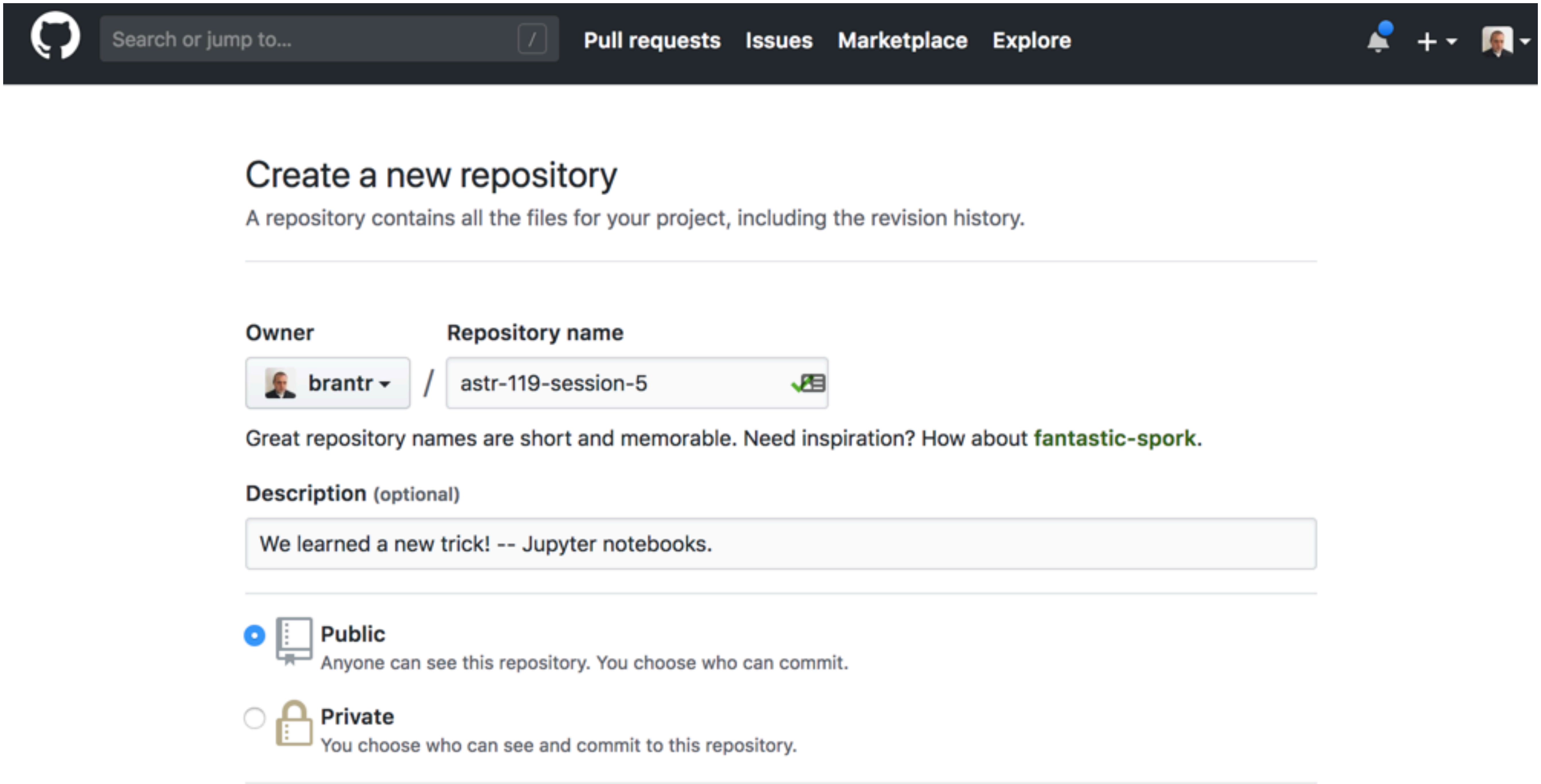
Perform the search

```
x_start = 0.5
tolerance = 1.0e-6

#print the initial guess
print(x_start,function_for_root(x_start))

x_root = newton_raphson_root_finding(function_for_root,derivative_for_root,x_start,tolerance)
y_root = function_for_root(x_root)

s = "Root found with y(%f) = %f" % (x_root,y_root)
print(s)
```

```
0.5 0.8024999999999998
0.8953201970443347 0.5 0.8024999999999998 -2.0300000000000002
1.023494648595172 0.8953201970443347 0.15784083877308386 -1.231453201970443 8
1.040556119705499 1.023494648595172 0.016592976930660974 -0.9725408098377528
1.040869531981685 1.040556119705499 0.00029400473441354436 -0.9380766381948917
Root found with y(1.040870) = 0.000000
```

Save Your Work

Make a GitHub project “astr-119-session-8”, and commit the programs my_first_jupyter_notebook.ipynb and test_matplotlib.ipynb you made today.



UC SANTA CRUZ