ASTR 119: Session 7
Line fitting, root finding

Outline

1) New homework due 10/29 at 8:00am
2) Visualization of the Day
3) Line fitting, continued
4) Root finding: Bisection Search
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.

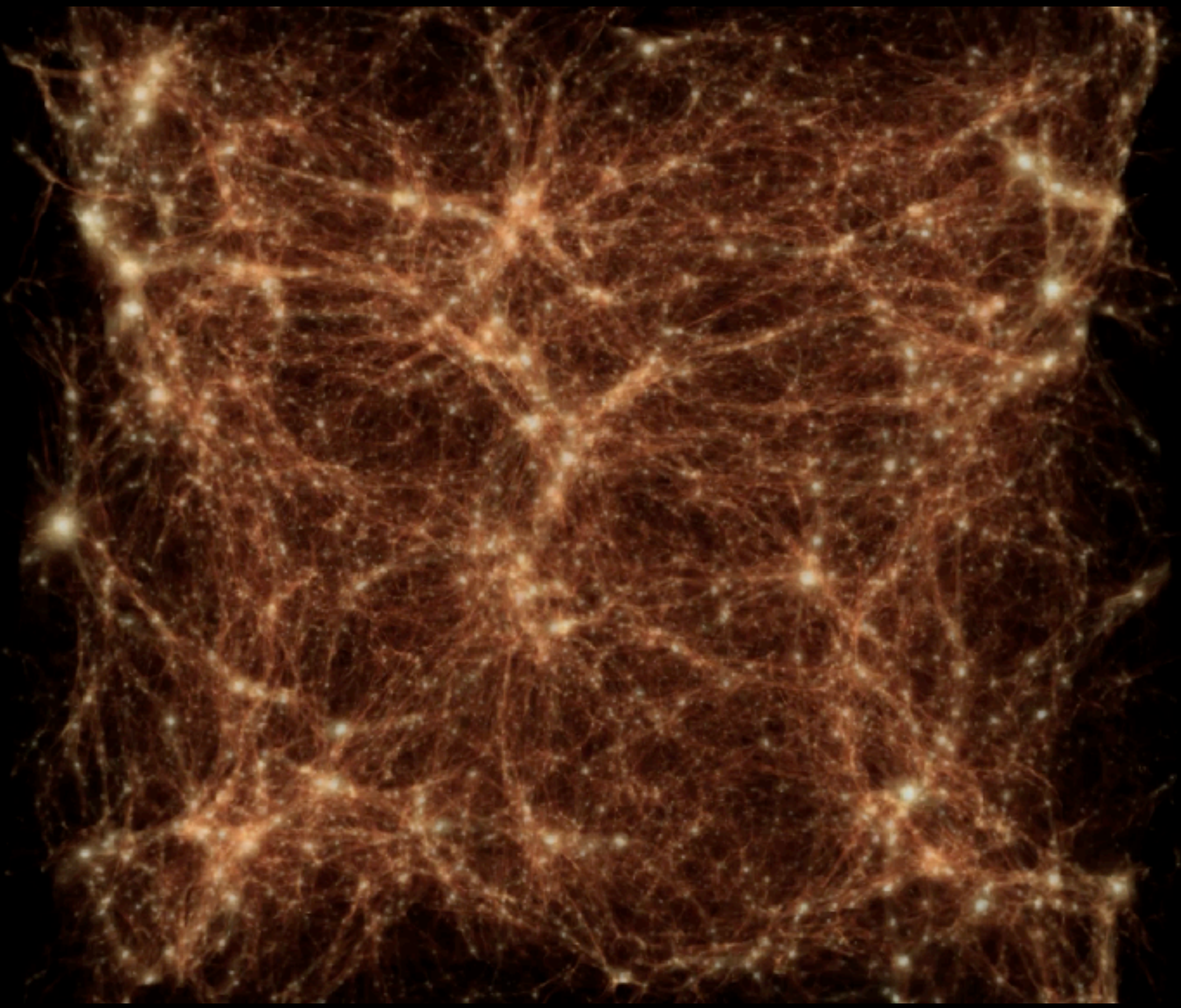
2) Given your starting guesses for the bracketing values around the roots, how many iterations does your method take to converge?

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 z=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”.



Line Fitting

Using python, we can quickly perform least squares line fitting

Example of performing linear least squares fitting

First we import numpy and matplotlib as usual.

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

Now, let's generate some random data about a trend line.

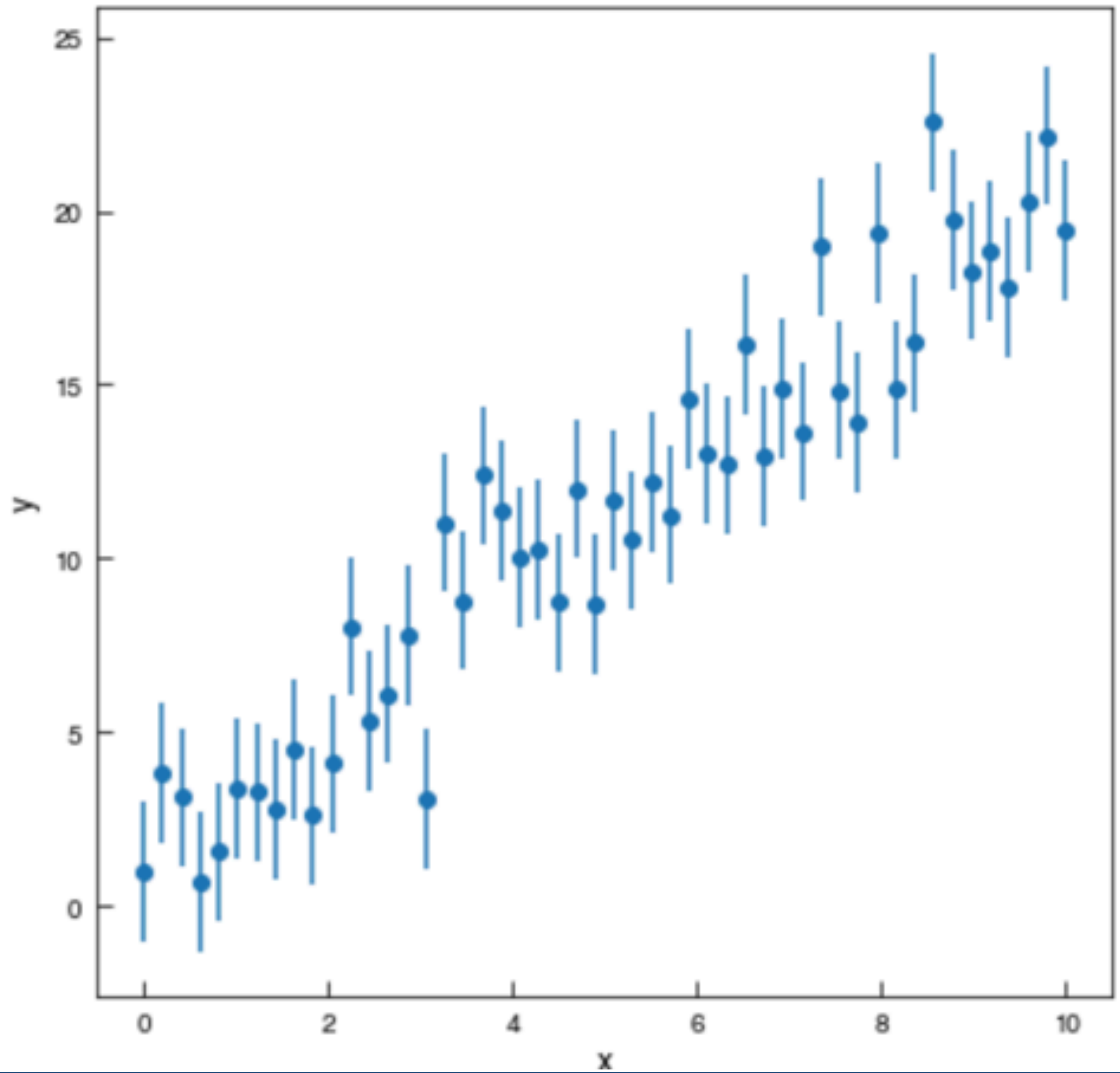
In [20]:
```
#set a random number seed
np.random.seed(119)

#set number of data points
npoints = 50

#set x
x = np.linspace(0,10.,npoints)

#set slope, intercept, and scatter rms
m = 2.0
b = 1.0
sigma = 2.0

#generate y points
y = m*x + b + np.random.normal(scale=sigma,size=npoints)
y_err = np.full(npoints,sigma)
```

Line Fitting

Let's just plot the data first

In [14]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,sigma,fmt='o')
plt.xlabel('x')
plt.ylabel('y')
```

Out[14]: Text(0,0.5,'y')



UC SANTA CRUZ

Line Fitting

Method #1, polyfit()

In [28]:
```
m_fit, b_fit = np.poly1d(np.polyfit(x, y, 1, w=1./y_err))  #weight with uncertainties
print(m_fit, b_fit)

y_fit = m_fit * x + b_fit
```

```
1.96340434704 1.2830106813
```

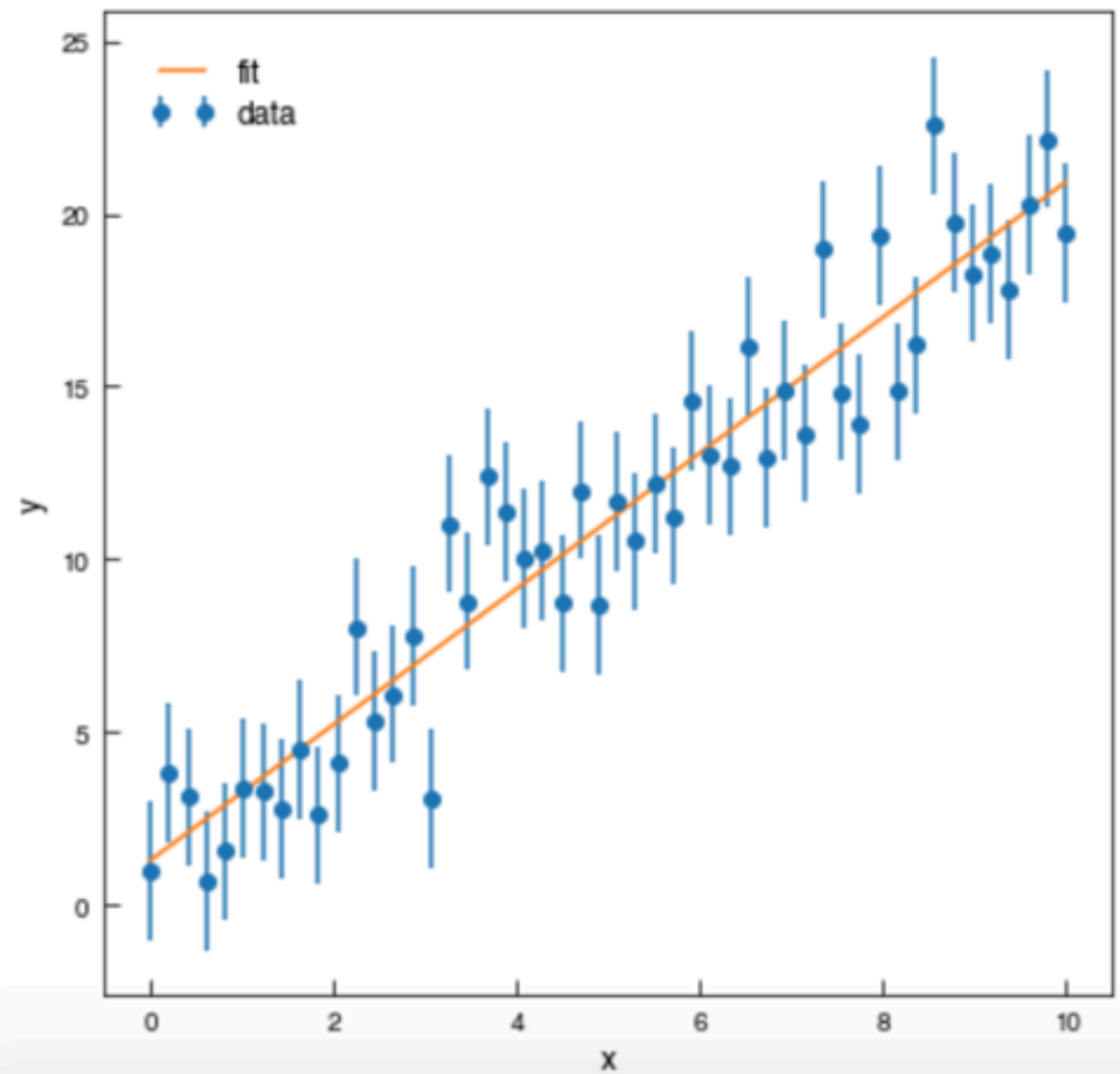
UC SANTA CRUZ

Line Fitting

Plot result

In [29]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o',label='data')
plt.plot(x,y_fit,label='fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=2,frameon=False)
```

Out[29]: <matplotlib.legend.Legend at 0x10fbbcef0>



Line Fitting

Method #2, scipy + optimize

In [31]:

```
#import optimize from scipy
from scipy import optimize

#define the function to fit
def f_line(x, m, b):
    return m*x + b

#perform the fit
params, params_cov = optimize.curve_fit(f_line,x,y,sigma=y_err)

m_fit = params[0]
b_fit = params[1]
print(m_fit,b_fit)
```

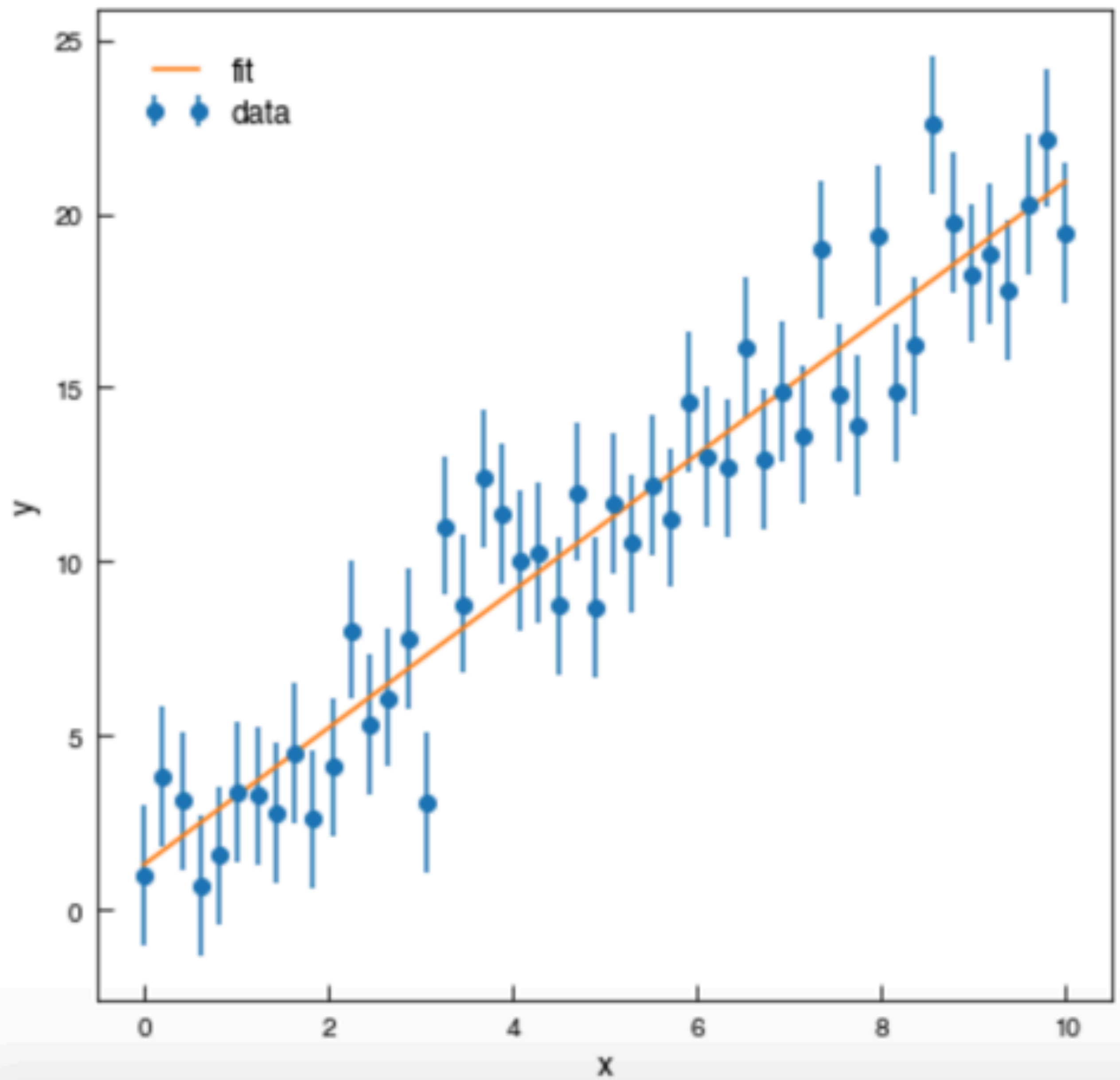
```
1.96340434575 1.28301068905
```

Line Fitting

Plot the result

In [33]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o',label='data')
plt.plot(x,y_fit,label='fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=2,frameon=False)
```

Out[33]: <matplotlib.legend.Legend at 0x112928fd0>



Line Fitting

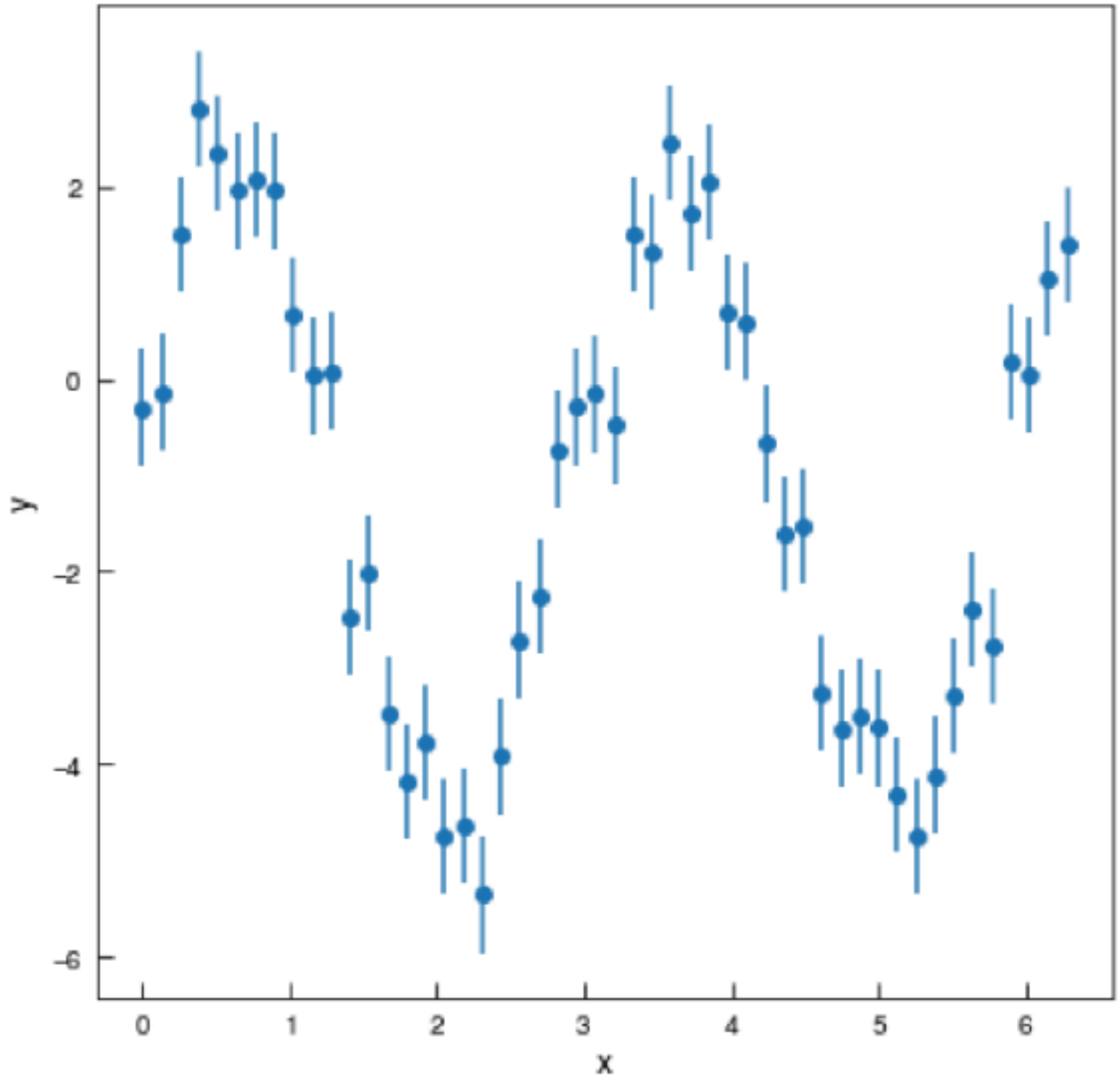
We can perform much more complicated fits....

In [38]:

```
#redefine x and y
npoints = 50
x = np.linspace(0.,2*np.pi,npoints)

#make y a complicated function
a = 3.4
b = 2.1
c = 0.27
d = -1.3
sig = 0.6

y = a * np.sin( b*x + c) + d + np.random.normal(scale=sig,size=npoints)
y_err = np.full(npoints,sig)

f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o')
plt.xlabel('x')
plt.ylabel('y')
```

Line Fitting

```
Out[38]: Text(0,0.5,'y')
```



Line Fitting

Perform a fit using scipy.optimize.curve_fit()

In [45]:
```
#import optimize from scipy
from scipy import optimize

#define the function to fit
def f_line(x, a, b, c, d):
    return a * np.sin( b*x + c) + d

#perform the fit
params, params_cov = optimize.curve_fit(f_line,x,y,sigma=y_err,p0=[1,2.,0.1,-0.1])

a_fit = params[0]
b_fit = params[1]
c_fit = params[2]
d_fit = params[3]

print(a_fit,b_fit,c_fit,d_fit)

y_fit = a_fit * np.sin(b_fit * x + c_fit) + d_fit
```

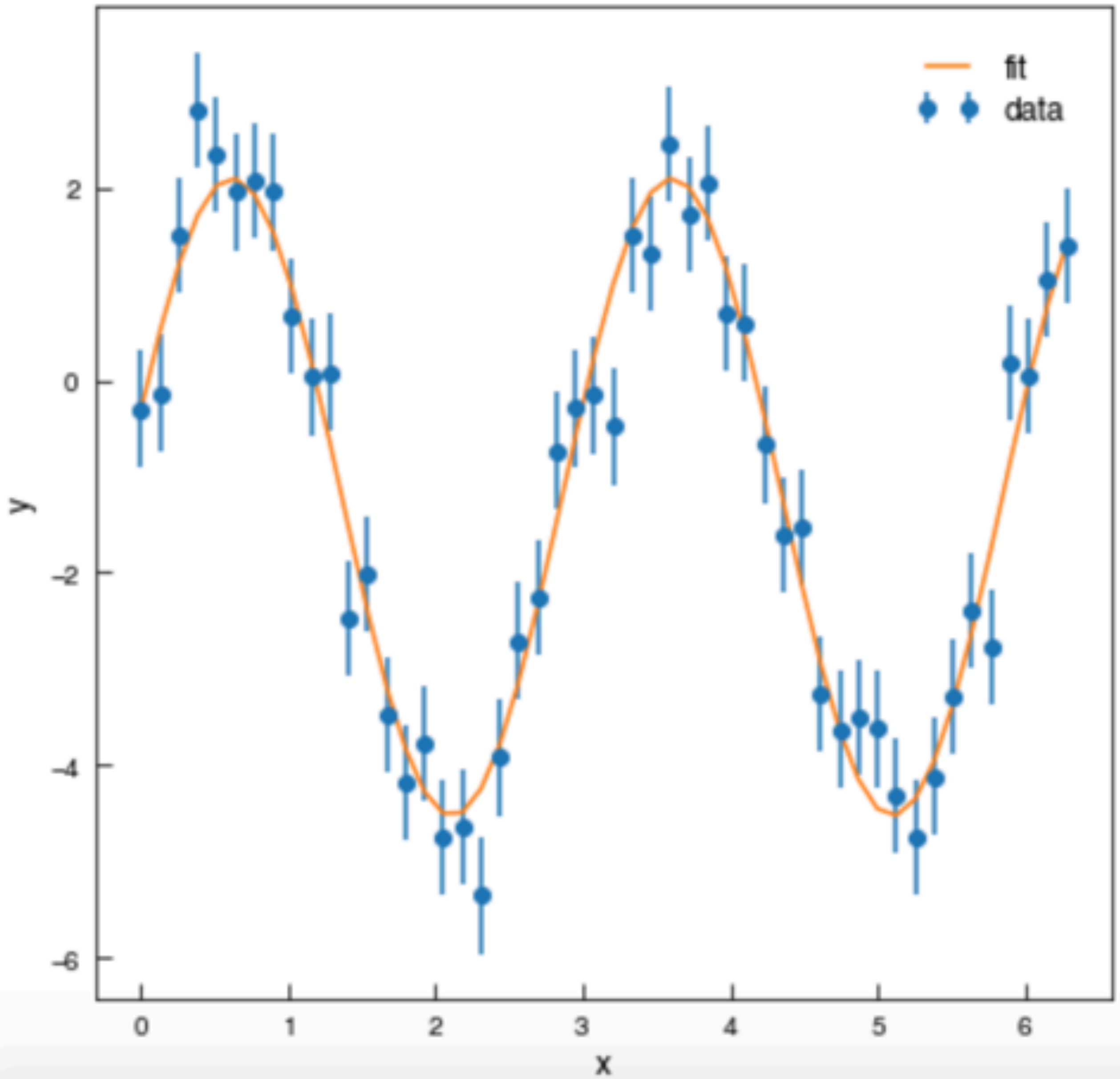
```
3.31470667373 2.10036419339 0.278528774808 -1.21522166095
```

Line Fitting

Plot the fit

In [48]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o',label='data')
plt.plot(x,y_fit,label='fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=0,frameon=False)
```

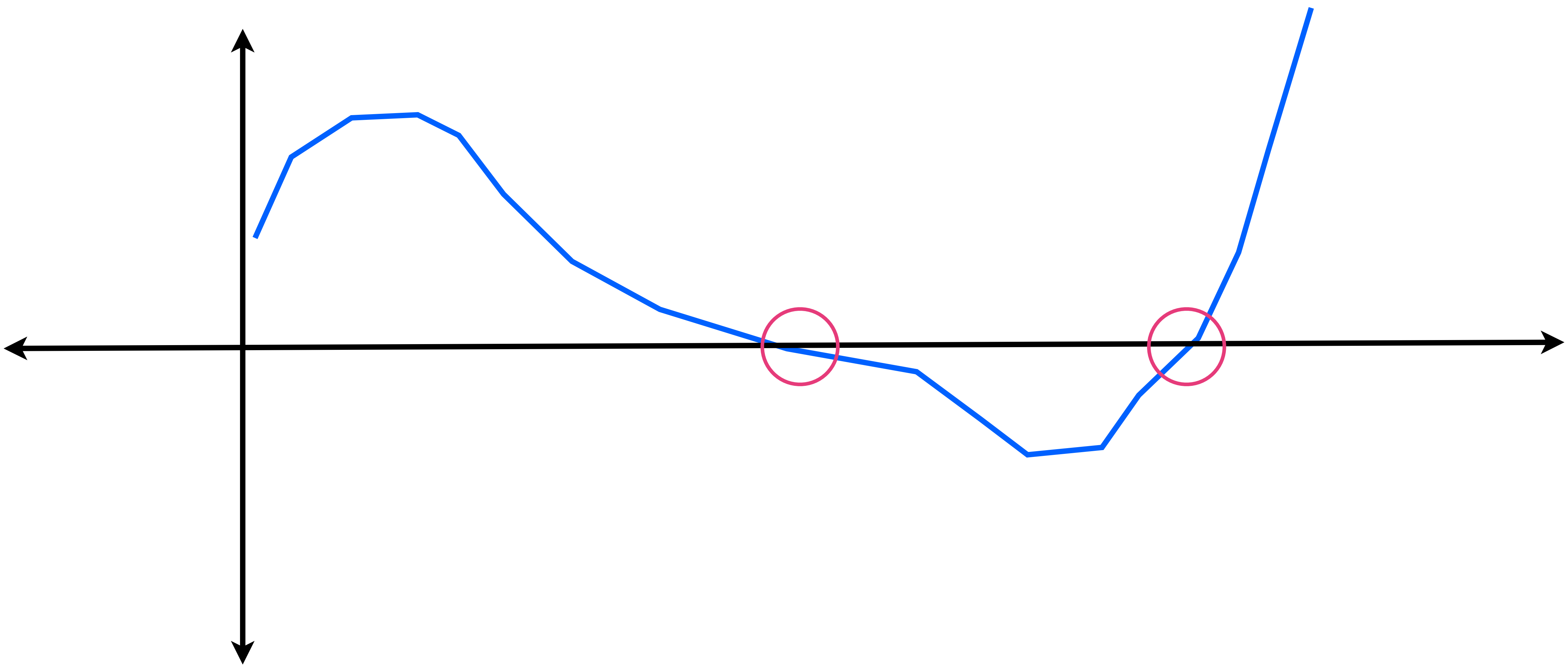
Out[48]: <matplotlib.legend.Legend at 0x11346b198>



Root Finding!

What is root finding?

Root finding is the process of finding the zero crossing of a mathematical function.



What is root finding?

For simple polynomial functions, root finding can be done analytically:

$$f(x) = x^2 - 3x + 2$$

What are the roots of this function?

What is root finding?

For simple polynomial functions, root finding can be done analytically:

$$f(x) = x^2 - 3x + 2$$

What are the roots of this function?

$$x = 1, x = 2$$

What is root finding?

However, some similar functions are difficult to analyze:

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

What are the roots of this function?

What is root finding?

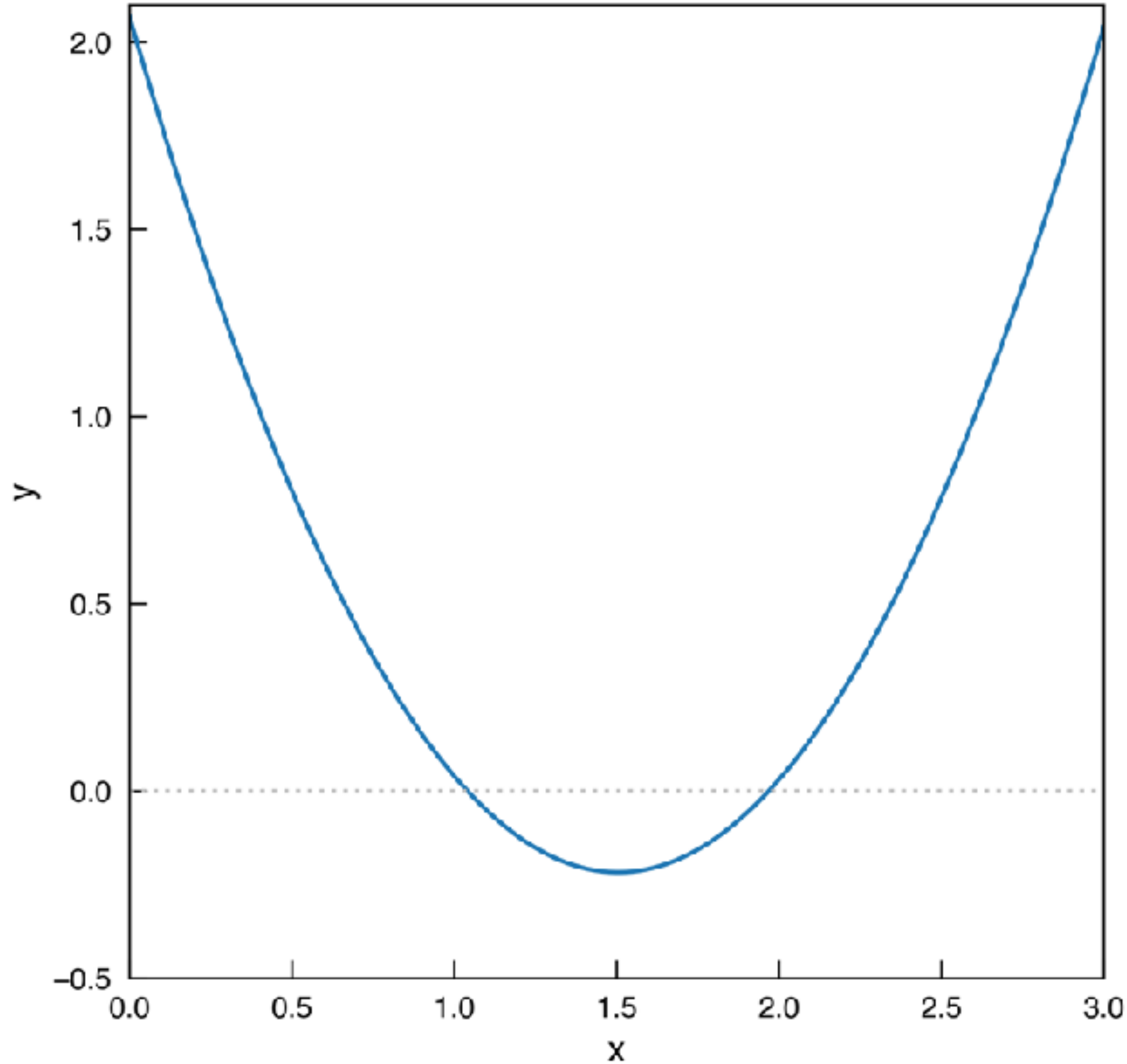
However, some similar functions are difficult to analyze:

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

What are the roots of this function?

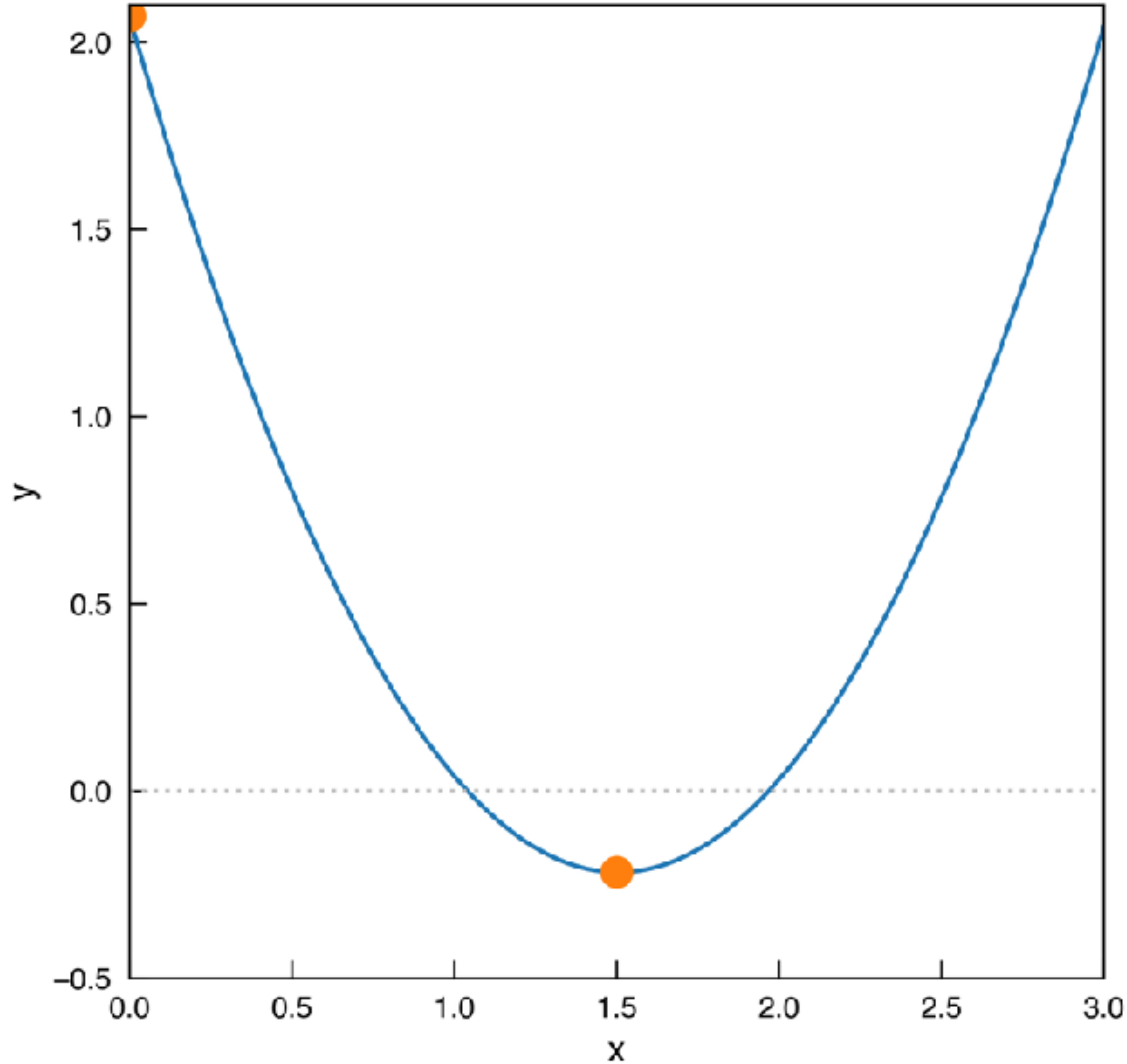
$$x \approx 1.040869, x \approx 1.969032$$

How do we find these roots?



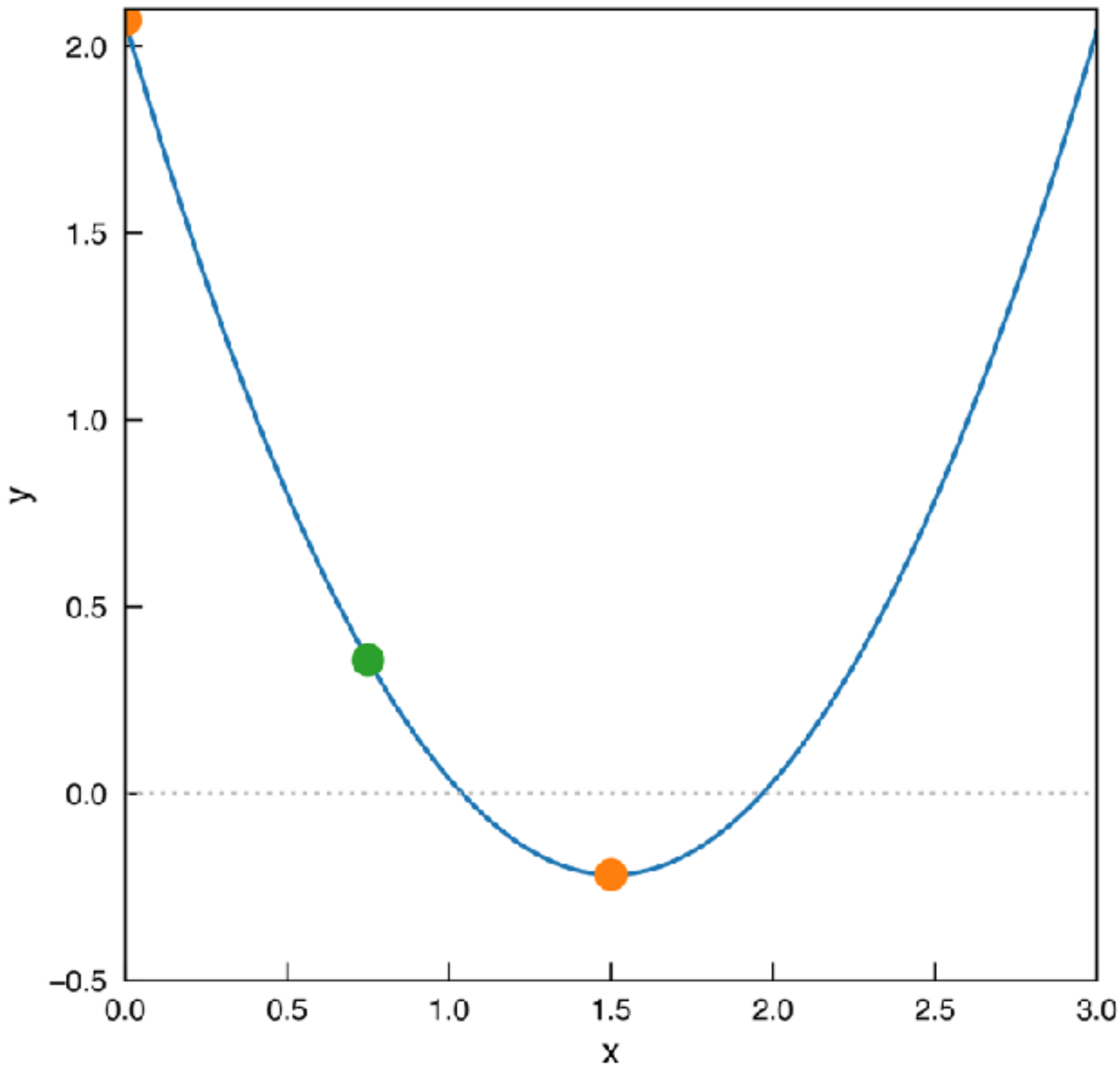
How do we find these roots?

1) bracket the root



How do we find these roots?

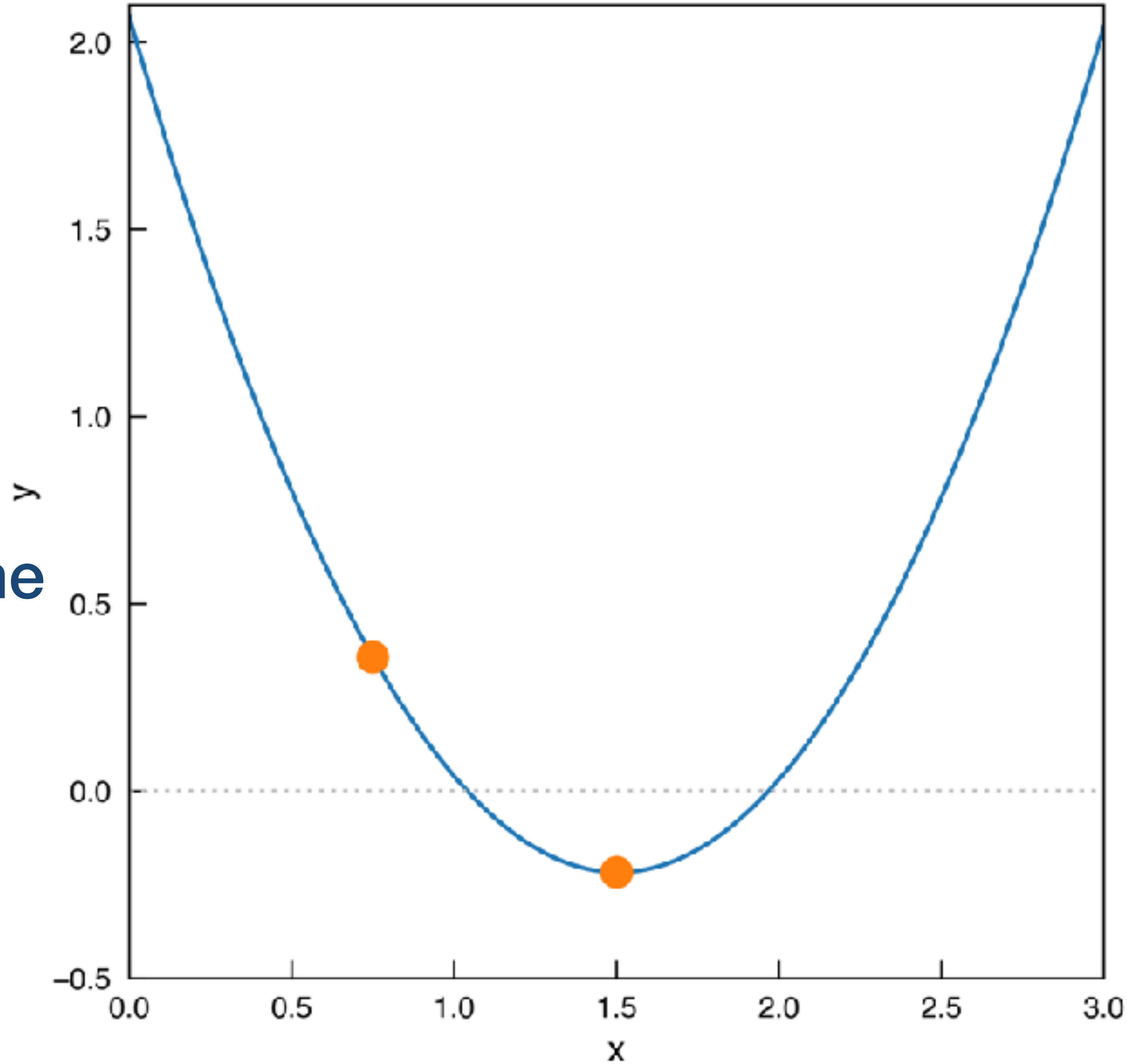
1) bracket the root
2) pick an intermediate value



How do we find these roots?

1) bracket the root
2) pick an intermediate value

3) then we shrink the bracket

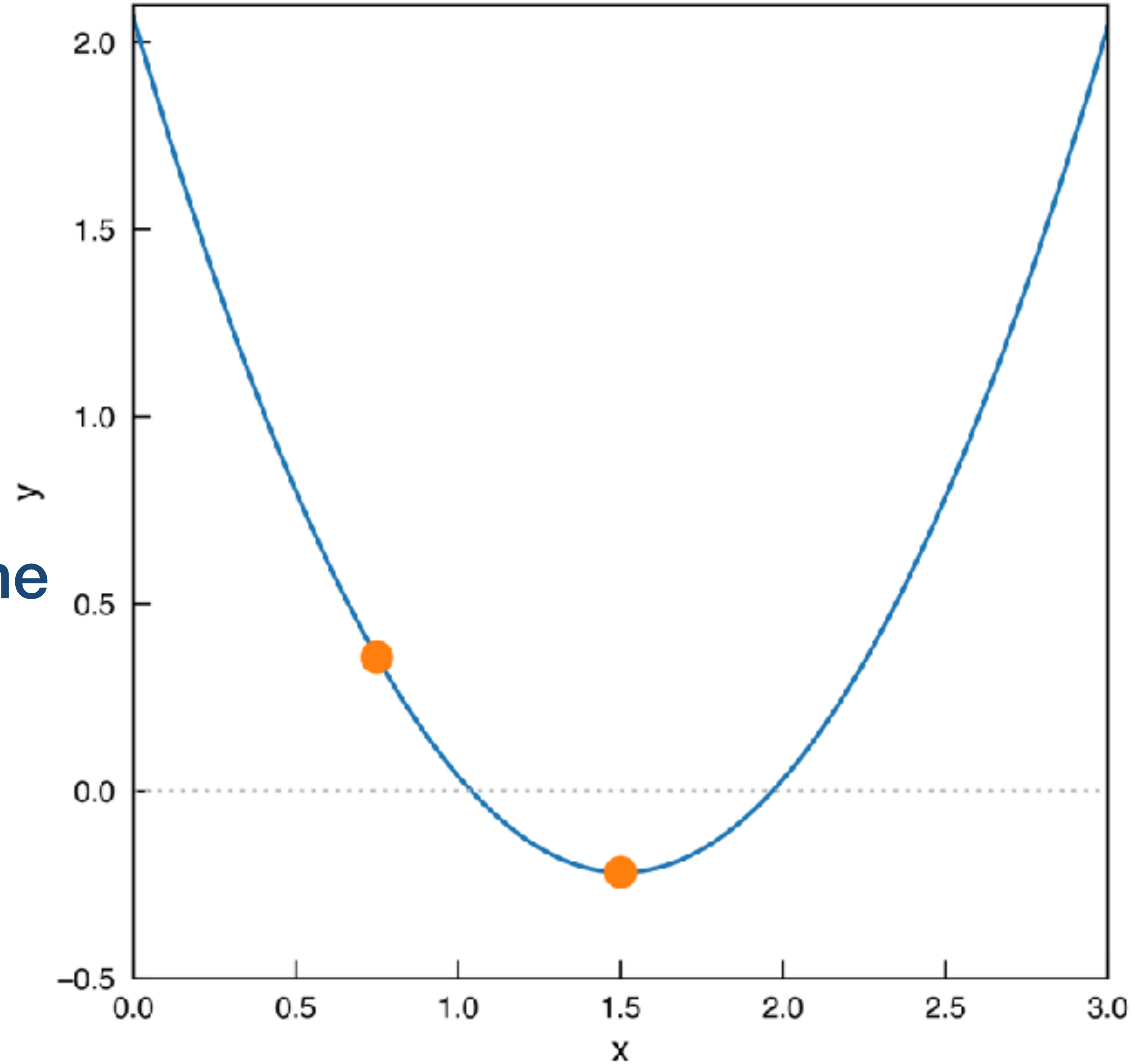


How do we find these roots?

1) bracket the root
2) pick an intermediate value

3) then we shrink the bracket

4) iterate until we reach some tolerance

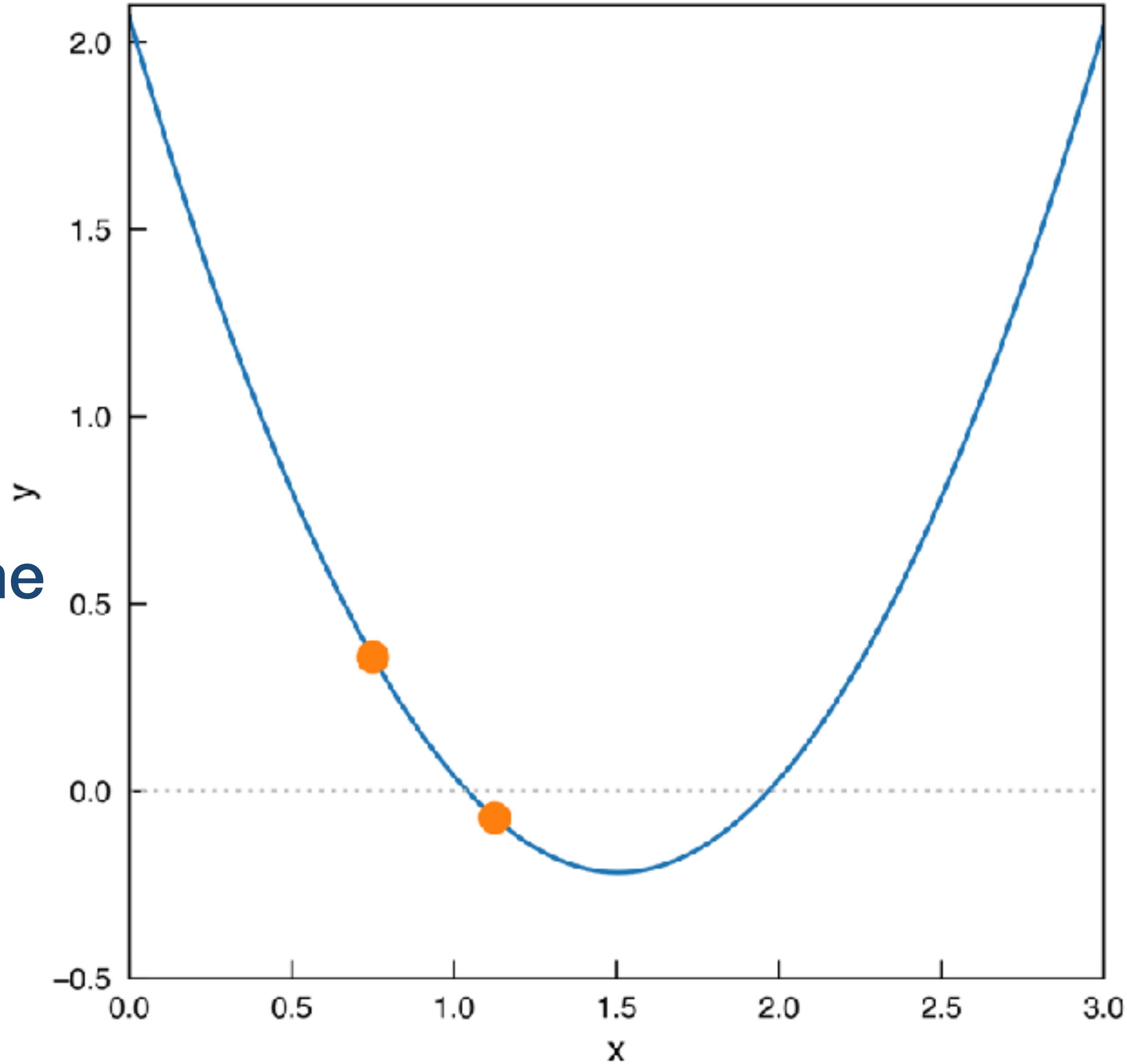


How do we find these roots?

1) bracket the root
2) pick an intermediate value

3) then we shrink the bracket

4) iterate until we reach some tolerance

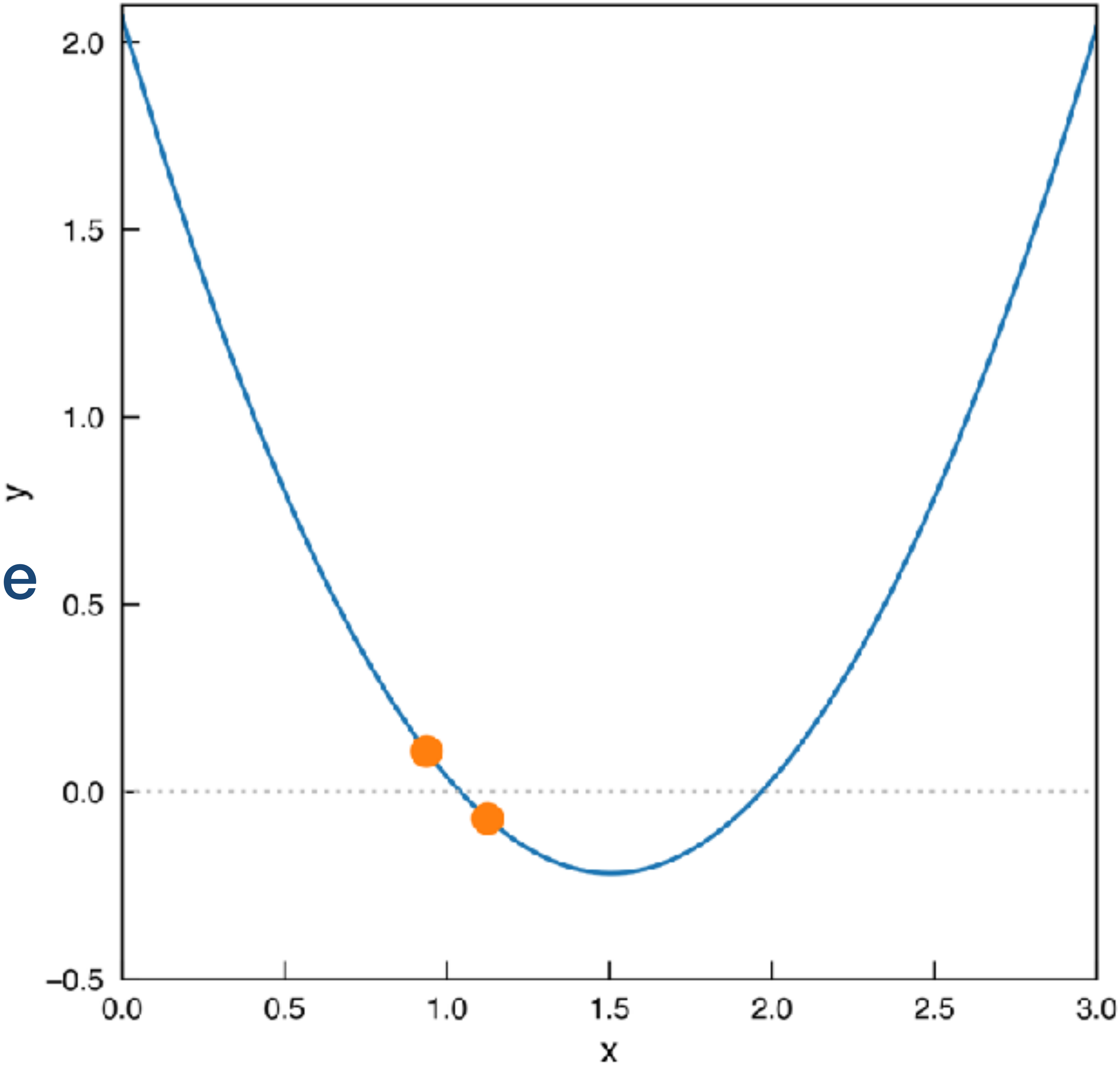


How do we find these roots?

1) bracket the root
2) pick an intermediate value

3) then we shrink the bracket

4) iterate until we reach some tolerance

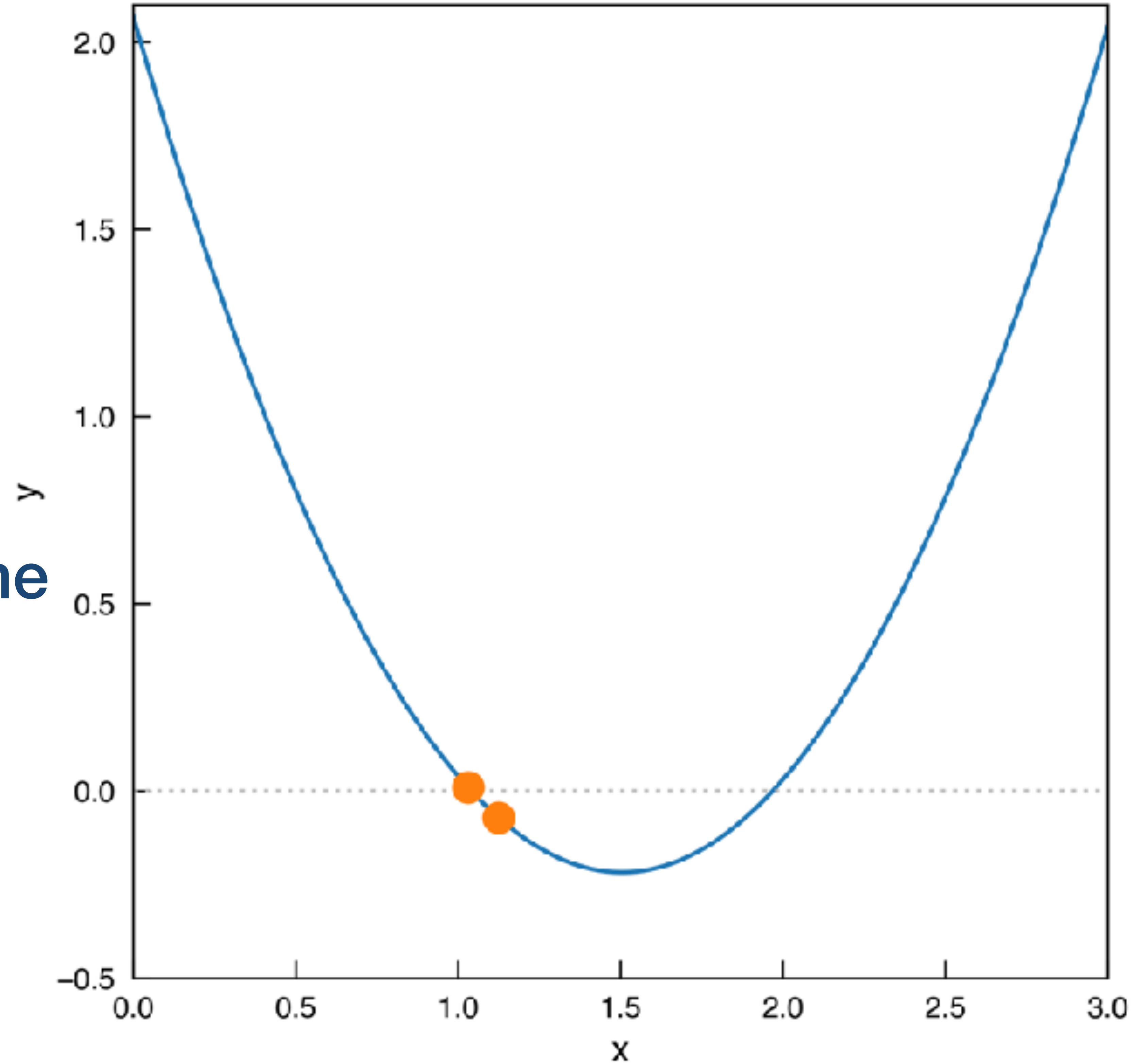


How do we find these roots?

1) bracket the root
2) pick an intermediate value

3) then we shrink the bracket

4) iterate until we reach some tolerance

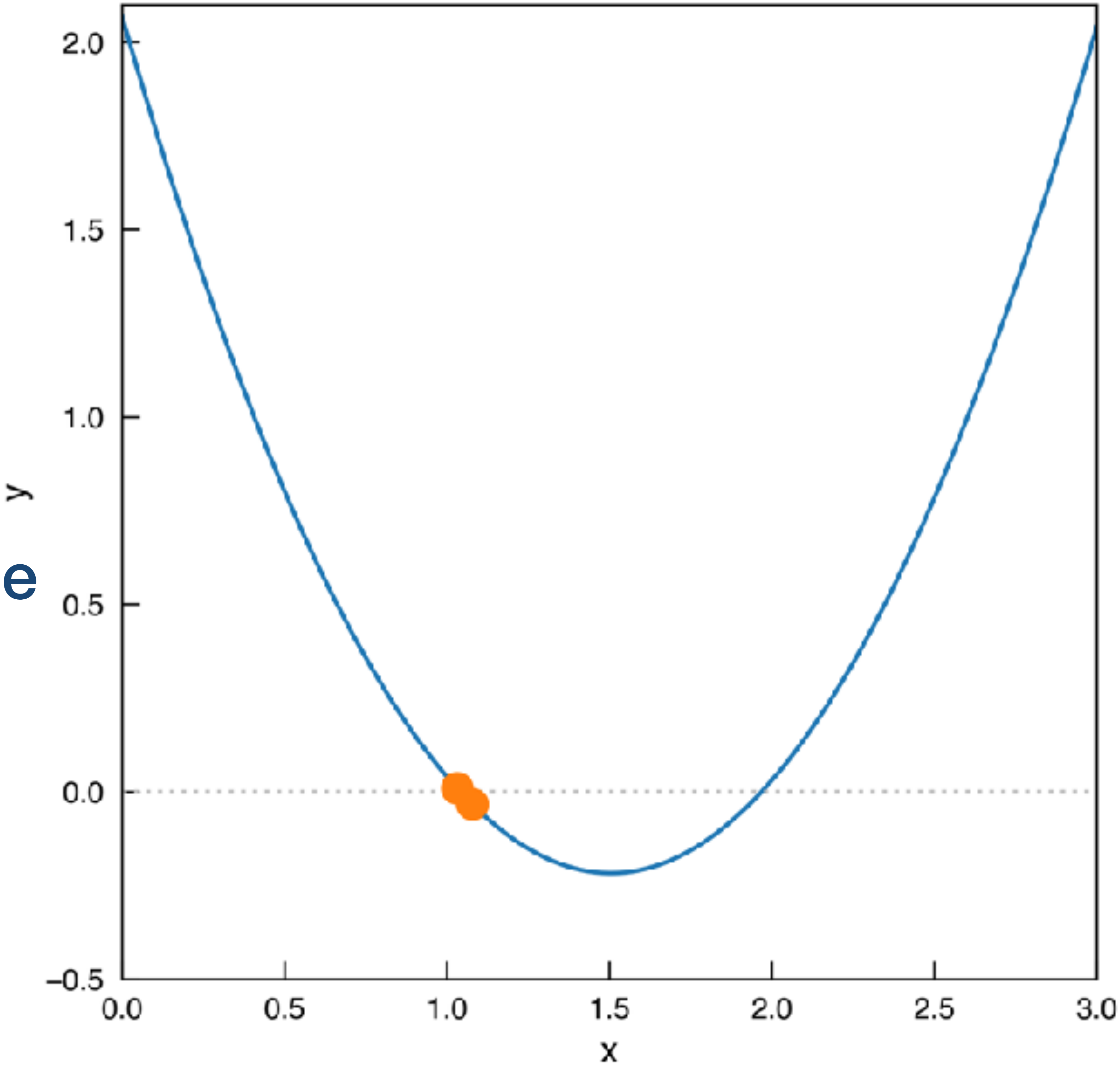


How do we find these roots?

1) bracket the root
2) pick an intermediate value

3) then we shrink the bracket

4) iterate until we reach some tolerance



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

Run Code

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

```python
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))
```

Bisection Search

```python
        #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]:
```
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 -0.0019633483886720704
1.037109375 0.003539314270019003 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 6.187822573424315e-06 1.04095458984375 -7.963042706249368e-05
1.040863037109375 6.187822573424315e-06 1.0409088134765625 -3.6723415833161965e-05
1.040863037109375 6.187822573424315e-06 1.0408859252929688 -1.5268322895334308e-05
1.040863037109375 6.187822573424315e-06 1.0408744812011719 -4.540379595852073e-06
1.040863037109375 6.187822573424315e-06 1.0408744812011719 -4.540379595852073e-06
Root found with y(1.040869) = 0.000001
```

Save Your Work

Make a GitHub project “astr-119-session-7”, and commit the programs my_first_jupyter_notebook.ipynb and test_matplotlib.ipynb you made today.