Chapter 5

Roots: Bracketing Methods

Chapter Objectives

- Determine a root graphically.
- Use incremental search method and understand its shortcomings.
- Use bisection method to solve a roots problem.
- Estimate the error in bisection method.

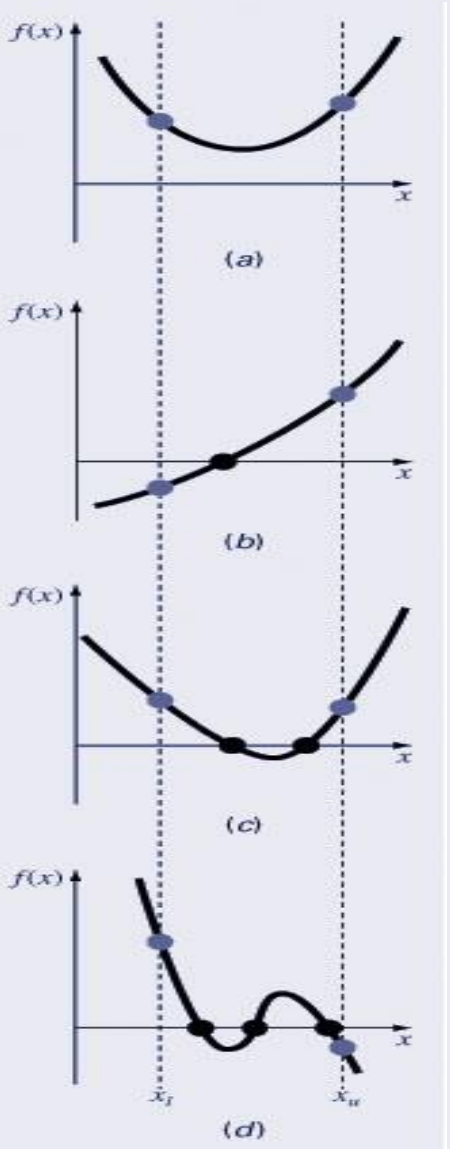
Roots

"Roots" problems occur when some function *f* can be written in terms of one or more independent variables *x*, where the solutions to *f(x)=0* is required

Graphical Methods

- A simple method for obtaining the estimate of the root of the equation *f(x)*=0 is to make a plot of the function and observe where it crosses the *x*-axis.
- Graphing the function can also indicate where roots may be and where some root-finding methods may fail:

a) Same sign, no roots
b) Different sign, one root
c) Same sign, two roots
d) Different sign, three roots



Example

Plot the function

$f(x) = \cos(x)+x\sin(x)$

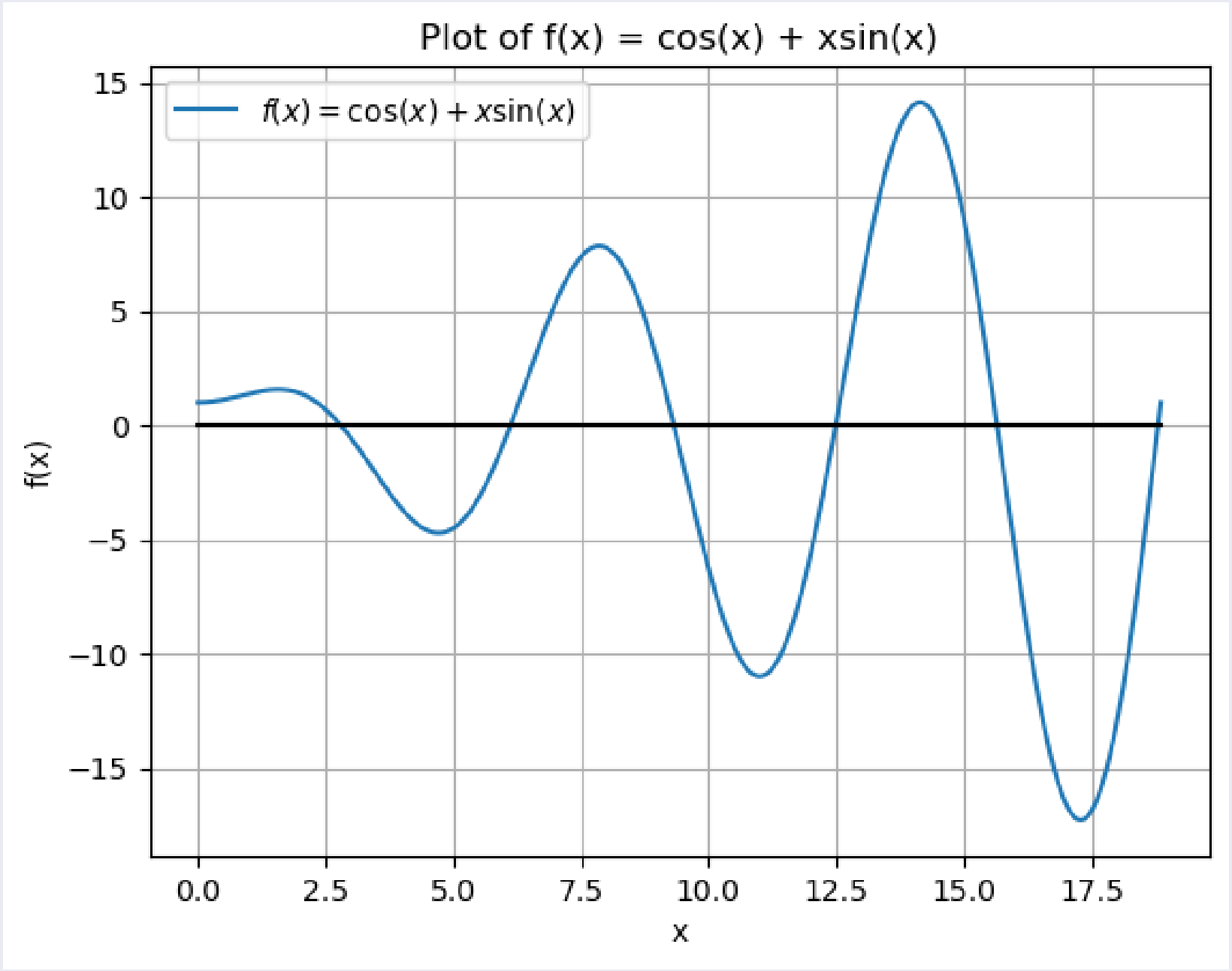
using plot in the interval from 0 to 6π.

- Plot the x-axis using

  xa=[0 6*π]; ya=[0 0];

- Enter grid
- Estimate roughly the zeros of f(x)

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(x):
    return np.cos(x) + x * np.sin(x)

# Define the x values
x = np.linspace(0, 6 * np.pi, 1000)

# Calculate the y values
y = f(x)

# Plot the function
plt.plot(x, y, label=r"$f(x) = \cos(x) + x\sin(x)$")

# Plot the x-axis
plt.plot([0, 6 * np.pi], [0, 0], color='black')

# Add grid
plt.grid(True)

# Set labels and title
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = cos(x) + xsin(x)')

# Show the plot
plt.legend()
plt.show()
```


Plot of f(x) = cos(x) + xsin(x)
f(x) = cos(x) + xsin(x)
15
10
5
0
−5
−10
−15
f(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
x

Bracketing Methods

- *Bracketing methods* are based on making two initial guesses that “bracket” the root - that is, are on either side of the root.

- Brackets are formed by finding two guesses x_l and x_u where the sign of the function changes; that is, where $f(x_l)\, f(x_u) < 0$

- The *incremental search* method tests the value of the function at evenly spaced intervals and finds brackets by identifying function sign changes between neighboring points.

The function incsearch

- Consider the function incsearch provided
- Go through the function to understand how it works
- Apply it to the function f(x)=cos(x)+xsin(x)
- Compare with the graphical estimate

```python
def incsearch(func, xmin, xmax, ns=50):
    """
    Incremental search root locator
    xb = incsearch(func, xmin, xmax, ns)

    Finds brackets of x that contain sign changes of a function in an interval.

    Inputs:
    - func: the function to evaluate
    - xmin, xmax: the endpoints of the interval
    - ns: the number of subintervals (default ns=50)

    Outputs:
    - xb: a list of tuples, where xb[k][0] is the lower bound of the k-th sign change,
          and xb[k][1] is the upper bound of the k-th sign change.
    - If no brackets are found, xb is an empty list.
    """

    # Create equally spaced points between xmin and xmax
    x = np.linspace(xmin, xmax, ns)
    f = []
    for k in range(ns-1):
        f.append(func(x[k]))

    nb = 0
    xb = []  # Empty list if no sign change detected

    # Loop through points to find sign changes

    for k in range(ns- 1):
        if func(x[k])*func(x[k+1])<0:  # Check for sign change
            nb = nb + 1 # increment the bracket counter
            xb.append((x[k],x[k+1])) # save the bracketing pair

    # Output the result
    if nb == 0:
        print('No brackets found')
    else:
        return nb, xb
```

```
incsearch(lambda x: np.cos(x) + x*np.sin(x),0, 6 * np.pi)
```

```
(6,
 [(2.6927937030769655, 3.077478517802246),
  (5.770272220879211, 6.154957035604492),
  (9.232435553406738, 9.617120368132019),
  (12.309914071208985, 12.694598885934266),
  (15.387392589011231, 15.772077403736512),
  (18.464871106813476, 18.84955592153876)])
```

Example

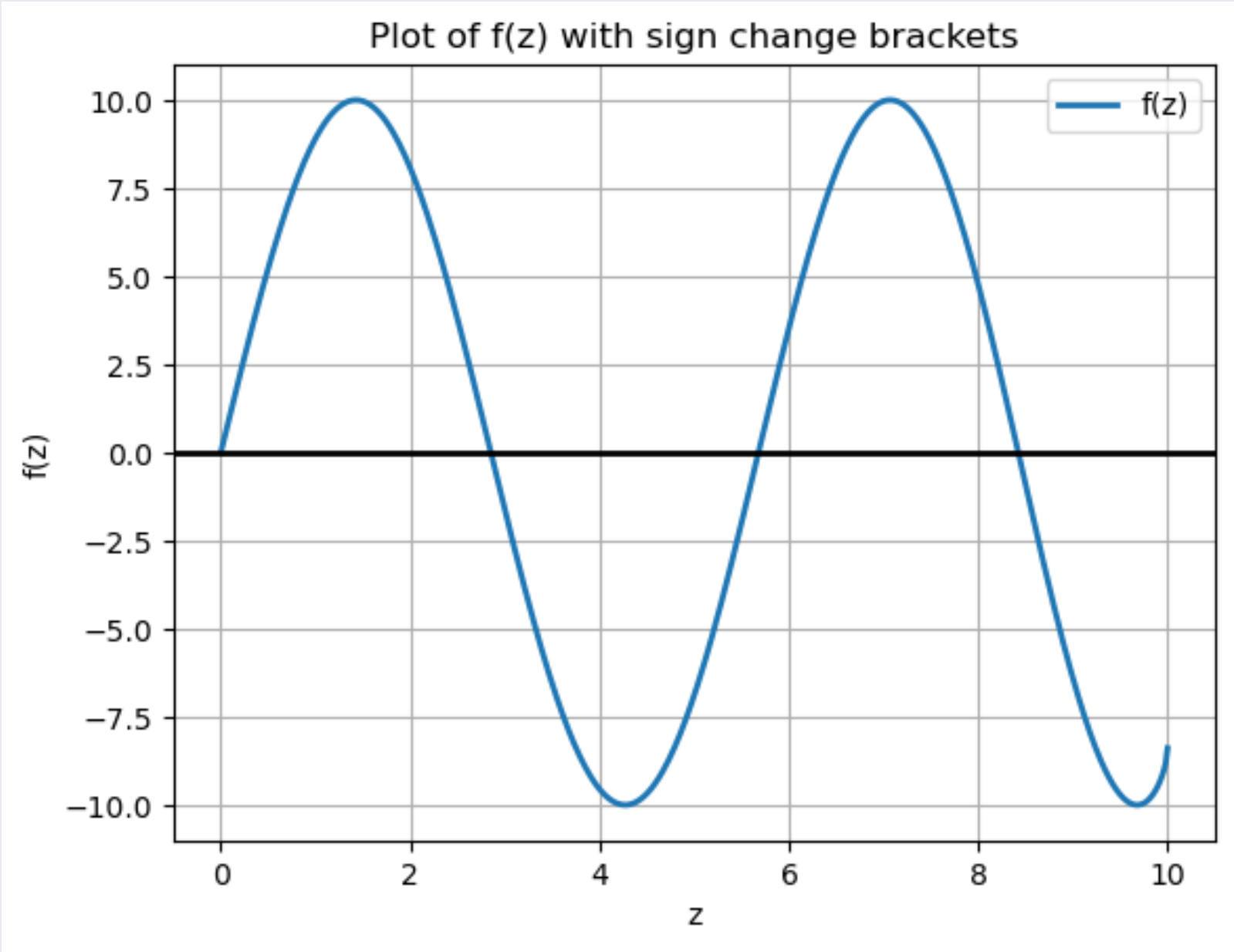
- Download incsearch in your working directory
- Open a file and name it **example_incsearch**
- Using the **handle function** define the function

$$f(z) = \sqrt{z_0^2 - z^2}\,\sin(z) + z\cos(z)$$

Set $z_0 = 10$ before you define the function.

- Use incsearch to bracket the roots.
- Plot the function

```python
import numpy as np
import matplotlib.pyplot as plt

def example_incsearch():
    # Clear variables, console, and close all plots (similar to Octave's clear, clc, clf)
    plt.clf()  # Clears the current figure
    z0 = 10

    # Define the function f as a lambda expression
    f = lambda z: np.sqrt(z0**2 - z**2) * np.sin(z) + z * np.cos(z)

    # Call the incsearch function to find the brackets
    zb = incsearch(f, 0, z0)

    # Plot the function using matplotlib's plot (fplot equivalent)
    z_vals = np.linspace(0, z0, 500)
    f_vals = f(z_vals)

    plt.plot(z_vals, f_vals, label='f(z)', linewidth=2)

    # Plot x-axis line (equivalent to xa=[0,z0], ya=[0,0] in Octave)
    plt.axhline(0, color='black', linewidth=2)

    # Add grid
    plt.grid(True)

    # Add brackets text at a specific location
    #plt.text(3.3, 4.5, str(zb), fontsize=14)

    # Display the plot
    plt.xlabel('z')
    plt.ylabel('f(z)')
    plt.title('Plot of f(z) with sign change brackets')
    plt.legend()
    plt.show()

    print(zb)
# Define the incsearch function (you already have this)
def incsearch(func, xmin, xmax, ns=50):
    # Create equally spaced points between xmin and xmax
    x = np.linspace(xmin, xmax, ns)
    f = []
    for k in range(ns-1):
        f.append(func(x[k]))

    nb = 0
    xb = []  # Empty list if no sign change detected

    # Loop through points to find sign changes

    for k in range(ns- 1):
        if func(x[k])*func(x[k+1])<0:  # Check for sign change
            nb = nb + 1 # increment the bracket counter
            xb.append((x[k],x[k+1])) # save the bracketing pair
```



[(2.6530612244897958, 2.857142857142857), (5.510204081632653, 5.714285714285714), (8.36734693877551, 8.571428571428571

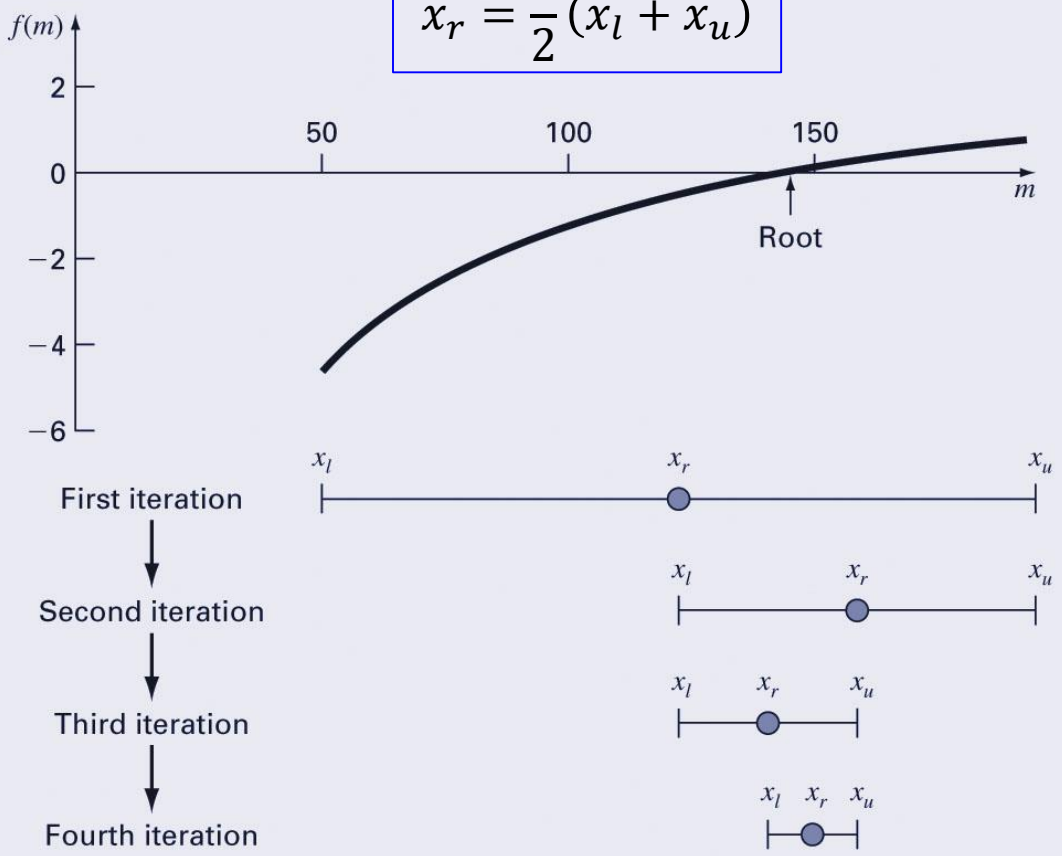
Incremental Search Hazards

- If the spacing between the points of an incremental search are too far apart, brackets may be missed due to capturing an even number of roots within two points.
- Incremental searches cannot find brackets containing even-multiplicity roots regardless of spacing.

Bisection

- The *bisection method* is a variation of the incremental search method in which the interval is always divided in half.
- If a function changes sign over an interval, the function value at the midpoint is evaluated.
- The location of the root is then determined as lying within the subinterval where the sign change occurs.
- The absolute error is reduced by a factor of 2 for each iteration.

$$x_r = \frac{1}{2}(x_l + x_u)$$



Bisection Error

- The absolute error of the bisection method is solely dependent on the absolute error at the start of the process (the space between the two guesses) and the number of iterations:

$$E_a^n = \frac{\Delta x^0}{2^n}$$

- The required number of iterations to obtain a particular absolute error can be calculated based on the initial guesses:

$$n = \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right)$$

Example

Let f(x) = cos(x). Search for the zero between 0 and 4.

Hence, initially xl = 0, xu = 4.

xl	xu	xr	f(xl)	f(xu)	f(xr)	Er
0.000	4.000	2.000	1.000	-0.654	-0.416	2.000
0.000	2.000	1.000	1.000	-0.416	0.540	1.000
1.000	2.000	1.500	0.540	-0.416	0.071	0.500
1.500	2.000	1.750	0.071	-0.416	-0.178	0.250
1.500	1.750	1.625	0.071	-0.178	-0.054	0.125
1.500	1.625	**1.563**	0.071	-0.056	0.008	0.063

Exact solution 1.571

Error $=\Delta x_0/2^n$

```
def bisect(func,xl,xu,es=1.e-7,maxit=30):
    """
    Uses the bisection method to estimate a root of func(x).
    The method is iterated until the relative error from
    one iteration to the next falls below the specified
    value or until the maximum number of iterations is
    reached first.
    Input:
    func = name of the function
    xl = lower guess
    xu = upper guess
    es = relative error specification (default 1.e-7)
    maxit = maximum number of iterations allowed (default 30)
    Output:
    xm = root estimate
    fm = function value at the root estimate
    ea = actual relative error achieved
    i+1 = number of iterations required
    or
    error message if initial guesses do not bracket solution
    """
    if func(xl)*func(xu)>0:
        return 'initial estimates do not bracket solution'
    xmold = xl
    for i in range(maxit):
        xm = (xl+xu)/2
        ea = abs((xm-xmold)/xm)
        if ea < es: break
        if func(xm)*func(xl)>0:
            xl = xm
        else:
            xu = xm
        xmold = xm
    return xm,func(xm),ea,i+1
```

Exercise

Apply ‘by hand’ the bisection method as used in the bisection code to find the root of

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

between ½ and 2.

Calculate the error as Ea=abs(xr-xrold) at each step.

xl	xu	xr	f(xl)	f(xu)	f(xr)	Er
0.500	2.00	1.250	-0.250	2.000	0.313	0.750
0.500	1.250	0.875	-0.250	0.313	-0.109	0.375
0.875	1.250	1.063	-0.109	0.313	0.067	0.188
0.875	1.063	0.969	-0.109	0.067	-0.030	0.094
0.969	1.063	1.016	-0.030	0.067	0.016	0.047
0.969	1.016	**0.993**	-0.030	0.016	0.007	0.023

Exact solution 1.00

Error =|xr-xrold|

Example

- Open function file **example_bisection**

- Define the function

$$f(z) = \sqrt{z_0^2 - z^2}\sin(z) + z\cos(z)$$

Set $z_0 = 10$ before you define the function.

- Use bisect find the root between 5.5 and 5.7

```
def bisect(func,xl,xu,es=1.e-7,maxit=30):

    if func(xl)*func(xu)>0:
        return 'initial estimates do not bracket solution'
    xmold = xl
    for i in range(maxit):
        xm = (xl+xu)/2
        ea = abs((xm-xmold)/xm)
        if ea < es: break
        if func(xm)*func(xl)>0:
            xl = xm
        else:
            xu = xm
        xmold = xm
    return xm,func(xm),ea,i+1

# Example function
def example_bisect():
    z0 = 10
    f = lambda z: np.sqrt(z0**2 - z**2) * np.sin(z) + z * np.cos(z)

    zl = 5.5
    zu = 5.7

    zr, fr, er, n = bisect(f, zl, zu)

    # Print results
    print(f'zr = {zr:.4f},  fr = {fr:e},   er = {er:e},  n = {n}')

    # Run the example function

example_bisect()
```

zr = 5.6792, fr = 2.200813e-06,   er = 6.716953e-08, n = 19

Homework

- Problem 5.22

Extend the problem to different radii.

For $r = 1{:}5$, use a for loop for r and calculate the function $f(h)$, the root of which gives the required height.

Plot $f(h)\ \ vs\ \ h$ for $0 \le h \le 2r$ for every r in the same figure.

Print the values of h for every r in a tabular form with heading $\quad$ "$r \qquad h$"