Problem 1

let $f(x) = 3(x+1)(x-\frac{1}{2})(x-1)$, use the Bisection method to find p_3 on $[-2, 1.5]$

[2]

```python
def f(x):
    return 3*(x+1)*(x-0.5)*(x-1)

def get_pn (n, func, interval):
    mid_point = (interval[0] + interval[1]) / 2
    if n == 1:
        return mid_point
    left = func(interval[0])
    right = func(interval[1])
    mid = func(mid_point)
    if mid == 0:
        return mid_point
    if left * mid < 0:
        return get_pn(n-1, func, (interval[0], mid_point))
    if right * mid < 0:
        return get_pn(n-1, func, (mid_point, interval[1]))
    # function may reach non return
```

[3]

```python
interval = (-2, 1.5)
print(f"p_1: {get_pn(1, f, interval)}")
print(f"p_2: {get_pn(2, f, interval)}")
print(f"p_3: {get_pn(3, f, interval)}")
```

```
p_1: -0.25
p_2: -1.125
p_3: -0.6875
```

{}

So the value of p_3 is -0.6875

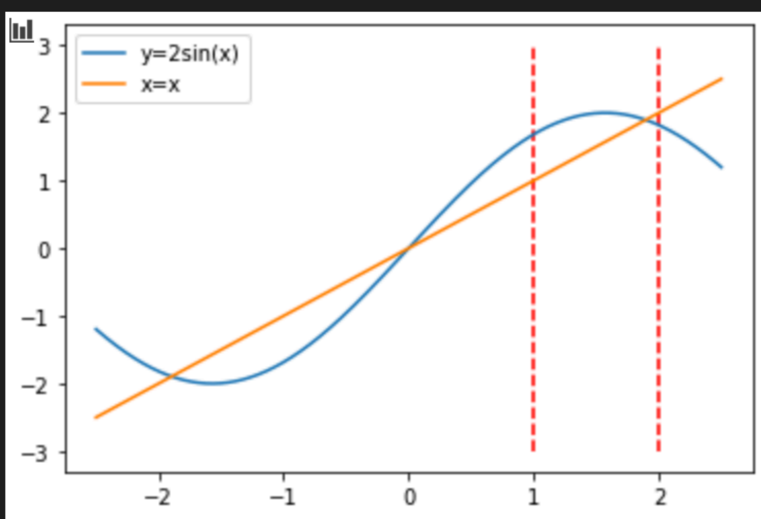
Probelm 2

(a) Sketch the graph of $y = x$ and $y = 2sinx$

[4]

```
x = np.linspace(-2.5, 2.5, 1000)
plt.plot(x, 2*np.sin(x), label = "y=2sin(x)", )
plt.plot(x, x, label = "x=x")
plt.vlines(x=1, ymin=-3, ymax=3, linestyles='dashed', color='r')
plt.vlines(x=2, ymin=-3, ymax=3, linestyles='dashed', color='r')
plt.legend()
```

<matplotlib.legend.Legend at 0x7fee9b938c90>



(b) use the Bisection method fo find an approximation to within 10^{-5} ot the first positive value of x with $x = 2sinx$

Answer:

- The problem is asking to find the root of $2sinx - x = 0$ on $(0, \infty)$
- From the plot, we can locate the root is in $(1, 2)$, so we can play bisection on it
- The final answer found by following code is `1.8954925537109375`

[5]

[5]

```python
# define root function
class func_p2():
    def __call__(self, x):
        return 2*np.sin(x) - x
    def __repr__(self):
        return "2sinx = x"
```

[6]

```python
# This function is adopted from bisect.m on CCLE
def bisection(func=func_p2(), a=1, b=2, rootEps=10**(-5), resitualEps=10**(-5), 
max_iter=1000):
    left = func(1)
    right = func(b)
    if(abs(left) < resitualEps):
        print(f"Approximate root of {func} is {a}")
        return(interval[0])
    if(abs(right) < resitualEps):
        print(f"Approximate root of {func} is {b}")
        return(interval[1])
    for i in range(max_iter):
        if (b-a)/2 < rootEps:
            break
        c = (a + b) / 2
        print(f"iter: {i}, check {round(c, 15)}")
        mid = func(c)
        if abs(mid) < resitualEps:
            break
        if left*mid < 0:
            b = c
            right = mid
        if right*mid < 0:
            a = c
            left = mid
    if i == max_iter:
        print("Oops, didn't converge within {max_iter} iterations")
    print(f"Approximate root of {func} is {c}")
    print(f"Error bound = {round((b-a/2), 15)}")
    print(f"Residual = {abs(mid)}")
    print(f"Iteration: {i}")
    return round(c, 15)
```

[7]

```python
            print(f"Approximate root of {func} is {b}")
            return(interval[1])
        for i in range(max_iter):
            if (b-a)/2 < rootEps:
                break
            c = (a + b) / 2
            print(f"iter: {i}, check {round(c, 15)}")
            mid = func(c)
            if abs(mid) < resitualEps:
                break
            if left*mid < 0:
                b = c
                right = mid
            if right*mid < 0:
                a = c
                left = mid
        if i == max_iter:
            print("Oops, didn't converge within {max_iter} iterations")
        print(f"Approximate root of {func} is {c}")
        print(f"Error bound = {round((b-a/2), 15)}")
        print(f"Residual = {abs(mid)}")
        print(f"Iteration: {i}")
        return round(c, 15)
```

[7]

```python
root = bisection()
```

```
iter: 0, check 1.5
iter: 1, check 1.75
iter: 2, check 1.875
iter: 3, check 1.9375
iter: 4, check 1.90625
iter: 5, check 1.890625
iter: 6, check 1.8984375
iter: 7, check 1.89453125
iter: 8, check 1.896484375
iter: 9, check 1.8955078125
iter: 10, check 1.89501953125
iter: 11, check 1.895263671875
iter: 12, check 1.8953857421875
iter: 13, check 1.89544677734375
iter: 14, check 1.895477294921875
iter: 15, check 1.895492553710938
Approximate root of 2sinx = x is 1.8954925537109375
Error bound = 0.947769165039062
Residual = 2.806497545249087e-06
Iteration: 15
```

Problem 3

Find an approximation to $\sqrt{3}$ correct to within 10^{-4} using the Bisection Algorithm (hint: consider $f(x) = x^2 - 3$)

Answer:

- The problem is asking to find the root of $x^2 - 3 = 0$
- We can achieve this by using `bisection above` with modified paramemters
- We know the root $\sqrt{3} \in [1, 2]$
- The final answer is `1.7320556640625`

[8]

```python
# define root function
class func_p3():
    def __call__(self, x):
        return x**2 - 3
    def __repr__(self):
        return "x^2 - 3"
root = bisection(func=func_p3(), a=1, b=2, rootEps=10**(-4), resitualEps=10**(-4),
max_iter=1000)
```

```
iter: 0, check 1.5
iter: 1, check 1.75
iter: 2, check 1.625
iter: 3, check 1.6875
iter: 4, check 1.71875
iter: 5, check 1.734375
iter: 6, check 1.7265625
iter: 7, check 1.73046875
iter: 8, check 1.732421875
iter: 9, check 1.7314453125
iter: 10, check 1.73193359375
iter: 11, check 1.732177734375
iter: 12, check 1.7320556640625
Approximate root of x^2 - 3 is 1.7320556640625
Error bound = 0.8662109375
Residual = 1.6823410987854004e-05
Iteration: 12
```

Problem 4

Using the theorem in Section 2.1, find a bound for the number of iterations needed to achieve an
approximation with accuracy 10^{-4} to the solution of $x^3 - x - 1 = 0$ lying in the interval $[1, 2]$
.
Find an approximation to the root with this degree of accuracy

```
{}
```

Answer

- We need to solve

$$|P_n - P| \leq \frac{b-a}{2^n} \leq 10^{-4}$$

$$\frac{1}{2^n} \leq 10^{-4}$$

$$log_2 10^4 \leq n$$

$$n = \lceil 13.287712379549449 \rceil = 14$$

- We need at most `14` iterations
- Approximation root is `1.32470703125`

[9]

```
print(f" log2(10^4) = {np.log2(10**4)}")
```

```
log2(10^4) = 13.287712379549449
```

[10]

```
# define root function
class func_p4():
    def __call__(self, x):
        return x**3 - x - 1
    def __repr__(self):
        return "x^3 - x - 1 = 0"
root = bisection(func=func_p4(), a=1, b=2, rootEps=10**(-4), resitualEps=10**(-4),
max_iter=1000)
```

```
iter: 0, check 1.5
iter: 1, check 1.25
iter: 2, check 1.375
```

{}

Answer

- We need to solve

$$|P_n - P| \leq \frac{b-a}{2^n} \leq 10^{-4}$$

$$\frac{1}{2^n} \leq 10^{-4}$$

$$log_2 10^4 \leq n$$

$$n = \lceil 13.287712379549449 \rceil = 14$$

- We need at most 14 iterations
- Approximation root is 1.32470703125

[9]

```python
print(f" log2(10^4) = {np.log2(10**4)}")
```

```
log2(10^4) = 13.287712379549449
```

[10]

```python
# define root function
class func_p4():
    def __call__(self, x):
        return x**3 - x - 1
    def __repr__(self):
        return "x^3 - x - 1 = 0"
root = bisection(func=func_p4(), a=1, b=2, rootEps=10**(-4), resitualEps=10**(-4),
max_iter=1000)
```

```
iter: 0, check 1.5
iter: 1, check 1.25
iter: 2, check 1.375
iter: 3, check 1.3125
iter: 4, check 1.34375
iter: 5, check 1.328125
iter: 6, check 1.3203125
iter: 7, check 1.32421875
iter: 8, check 1.326171875
iter: 9, check 1.3251953125
iter: 10, check 1.32470703125
Approximate root of x^3 - x - 1 = 0 is 1.32470703125
Error bound = 0.6630859375
Residual = 4.659488331526518e-05
Iteration: 10
```

Problem 5

(1) Use algebraic manipulations to show that each of the following functions has a fixed point at p precisely when $f(p) = 0$, where

$$f(x) = x^4 + 2x^2 - x - 3$$

(a) $g_1(x) = (3 + x - 2x^2)^{\frac{1}{4}}$

- Fix point of g_1(x) satisfies

$$x = (3 + x - 2x^2)^{\frac{1}{4}}$$

$$x^4 = (3 + x - 2x^2)$$

$$x^4 + 2x^2 - x - 3 = 0$$

- Which implies x is a root of $f(x)$

(b) $g_2(x) = \left(\frac{x+3-x^4}{2}\right)^{\frac{1}{2}}$

- Fix point of g_2(x) satisfies

$$x = \left(\frac{x+3-x^4}{2}\right)^{\frac{1}{2}}$$

$$x^2 = \left(\frac{x+3-x^4}{2}\right)$$

$$2x = x + 3 - x^4$$

$$x^4 + 2x^2 - x - 3 = 0$$

- Which implies x is a root of $f(x)$

(2) Perform four iterations, if possible, on each of the functions g defined in Exercise 5.1. Let $p_0 = 1$ and $p_{n+1} = g(p_n)$ for $n = 0, 1, 2, 3$.

[11] M↓

```
# define root function
class g1():
```

(2) Perform four iterations, if possible, on each of the functions g defined in Exercise 5.1. Let $p_0 = 1$ and $p_{n+1} = g(p_n)$ for $n = 0, 1, 2, 3$.

[11]

```python
# define root function
class g1():
    def __call__(self, x):
        return (3+x-2*x**2)**(1/4)
    def __repr__(self):
        return "g1"
class g2():
    def __call__(self, x):
        return np.sqrt( (x+3-x**4)/2 )
    def __repr__(self):
        return "g2"
def fix_point(func, start=1, max_iter=4):
    print("p0   = {:>15.15f}      ".format(start) + str(func))
    sequence = [start] #forget about efficiency for now
    for n in range(max_iter):
        pn = func(start)
        start = pn
        print("p{:<2d} = {:>15.15f}      ".format(n+1, start) + str(func))
        sequence.append(start)
    return sequence
```

[12]

```python
seq_g1 = fix_point(g1(), 1, 4)
print()
seq_g2 = fix_point(g2(), 1, 4)
```

```
p0   = 1.000000000000000      g1
p1   = 1.189207115002721      g1
p2   = 1.080057752667562      g1
p3   = 1.149671430589383      g1
p4   = 1.107820529510260      g1

p0   = 1.000000000000000      g2
p1   = 1.224744871391589      g2
p2   = 0.993666159077482      g2
p3   = 1.228568645274987      g2
p4   = 0.987506429150887      g2
```

(3) Which function do you think gives the best approximation to the solution **Answer:** g1 gives better approximation

- g1 converges to 1.12 in 7 iterations
- g2 is not converging within first 20 iterations

[13]

```python
seq_g1 = fix_point(g1(), 1, 20)
print()
seq_g2 = fix_point(g2(), 1, 20)
```

```
p0  = 1.000000000000000      g1
p1  = 1.189207115002721      g1
p2  = 1.080057752667562      g1
p3  = 1.149671430589383      g1
p4  = 1.107820529510260      g1
p5  = 1.133932284504731      g1
p6  = 1.118003117715782      g1
p7  = 1.127857163488397      g1
p8  = 1.121813166001129      g1
p9  = 1.125539874244780      g1
p10 = 1.123249432277919      g1
p11 = 1.124659955364435      g1
p12 = 1.123792378454735      g1
p13 = 1.124326406401652      g1
p14 = 1.123997843842416      g1
p15 = 1.124200050956795      g1
p16 = 1.124075628627704      g1
p17 = 1.124152196623251      g1
p18 = 1.124105080746848      g1
p19 = 1.124134074543470      g1
p20 = 1.124116233019905      g1

p0  = 1.000000000000000      g2
p1  = 1.224744871391589      g2
p2  = 0.993666159077482      g2
p3  = 1.228568645274987      g2
p4  = 0.987506429150887      g2
p5  = 1.232183418318806      g2
p6  = 0.981585828612739      g2
```