Samir Banjara 09/18/23

Question 1: Program Newton's Method and apply it to the function $f(x)=x^3+4x^2-10$. This function has a rooti= in [1,2][1,2][1,2]

In […

```
import numpy as np
import pandas as pd

def fixed_point (p0,e, max_it,g):
  p = np.zeros(max_it)
  p[0] = p0
  i = 1
  while i < max_it:
    try:
      p[i] = g(p[i-1])
    except:
      print('Arithmetic error')
      return(p)
    if abs(p[i] - p[i-1]) <= e:
      return(p)

      i += 1

    print('max number of iteration exceeded')
    return(p)

def newton_bisection(a, b, e, max_it, f):
  FA =  f(a)
  i = 0
  p = np.zeros(max_it)
  while i< max_it:
    p[i] = (a + b)  / 2
    FP = f(p[i])
    if (abs(b - a) <= e/2):
      return p
    if FA * FP < 0:
      b = p[i]
    else:
        a = p[i]
        FA = FP
    i += 1
  return(p)
  print('Warning. Max Iter Reached!')

def f(x):
  y = x**3 + 4 * x**2 - 10
  return y

def g1(x):
  y = x - (x**3 + 4 * x**2 - 10)
  return y
```

```
def g2(x):
  y = np.sqrt(10/x - 4 * x)
  return y

def g3(x):
    y = (1/2) * np.sqrt(10 - x**3)
    return y

def g4(x):
  y = np.sqrt(10/(4+x))
  return y

def g5(x):
  y = x - (x**3 + 4 * x**2 - 10)/(3 * x**2 + 8 * x)
  return y
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/Users/samirbanjara/Downloads/math625_assignment_1_Samir_Banjara.ipynb Cell
3 line 1
----> <a href='vscode-notebook-cell:/Users/samirbanjara/Downloads/math625_as
signment_1_Samir_Banjara.ipynb#W2sZmlsZQ%3D%3D?line=0'>1</a> import numpy as
np
      <a href='vscode-notebook-cell:/Users/samirbanjara/Downloads/math625_as
signment_1_Samir_Banjara.ipynb#W2sZmlsZQ%3D%3D?line=1'>2</a> import pandas a
s pd
      <a href='vscode-notebook-cell:/Users/samirbanjara/Downloads/math625_as
signment_1_Samir_Banjara.ipynb#W2sZmlsZQ%3D%3D?line=3'>4</a> def fixed_point
(p0,e, max_it,g):

ModuleNotFoundError: No module named 'numpy'
```

In […

```
p0 = 1.35
max_it = 30
e = 1e-8
a = 1
b = 2

p1 = fixed_point(p0, e, max_it, g1)
p2 = fixed_point(p0, e, max_it, g2)
p3 = fixed_point(p0, e, max_it, g3)
p4 = fixed_point(p0, e, max_it, g4)
p5 = fixed_point(p0, e, max_it, g5)
pn = newton_bisection(a, b, e, max_it, f)
```

```
max number of iteration exceeded
max number of iteration exceeded
max number of iteration exceeded
max number of iteration exceeded
max number of iteration exceeded
```

In […

```
np.array([p1, p2, p3, p4, p5, pn]).reshape(6,max_it).transpose()
```

```
Out[ …  array([[1.35      , 1.35      , 1.35      , 1.35      , 1.35      ,
         1.5       ],
        [1.599625  , 1.41683006, 1.37291888, 1.36717185, 1.36534501,
         1.25      ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.375     ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.3125    ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.34375   ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.359375  ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.3671875 ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36328125],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36523438],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36425781],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36474609],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36499023],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.3651123 ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36517334],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36520386],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36521912],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36522675],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36523056],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36522865],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36522961],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36523008],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36522985],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36522996],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36523002],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36522999],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36523001],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36523002],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         1.36523001],
```

```
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        1.36523001],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        ]])
```

In […

```
pd.DataFrame(np.array([p1,p2,p3,p4,p5,pn]).reshape(6,max_it).transpose(), co
```

Out[…

	g1	g2	g3	
0	1.350000000000000	1.350000000000000	1.350000000000000	1.350
1	1.599624999999998	1.416830055937340	1.372918879613796	1.367
2	0.000000000000000	0.000000000000000	0.000000000000000	0.000
3	0.000000000000000	0.000000000000000	0.000000000000000	0.000
4	0.000000000000000	0.000000000000000	0.000000000000000	0.000
5	0.000000000000000	0.000000000000000	0.000000000000000	0.000
6	0.000000000000000	0.000000000000000	0.000000000000000	0.000
7	0.000000000000000	0.000000000000000	0.000000000000000	0.000
8	0.000000000000000	0.000000000000000	0.000000000000000	0.000
9	0.000000000000000	0.000000000000000	0.000000000000000	0.000
10	0.000000000000000	0.000000000000000	0.000000000000000	0.000
11	0.000000000000000	0.000000000000000	0.000000000000000	0.000
12	0.000000000000000	0.000000000000000	0.000000000000000	0.000
13	0.000000000000000	0.000000000000000	0.000000000000000	0.000
14	0.000000000000000	0.000000000000000	0.000000000000000	0.000
15	0.000000000000000	0.000000000000000	0.000000000000000	0.000
16	0.000000000000000	0.000000000000000	0.000000000000000	0.000
17	0.000000000000000	0.000000000000000	0.000000000000000	0.000
18	0.000000000000000	0.000000000000000	0.000000000000000	0.000
19	0.000000000000000	0.000000000000000	0.000000000000000	0.000
20	0.000000000000000	0.000000000000000	0.000000000000000	0.000
21	0.000000000000000	0.000000000000000	0.000000000000000	0.000
22	0.000000000000000	0.000000000000000	0.000000000000000	0.000
23	0.000000000000000	0.000000000000000	0.000000000000000	0.000
24	0.000000000000000	0.000000000000000	0.000000000000000	0.000
25	0.000000000000000	0.000000000000000	0.000000000000000	0.000

	g1	g2	g3	
26	0.000000000000000	0.000000000000000	0.000000000000000	0.000
27	0.000000000000000	0.000000000000000	0.000000000000000	0.000
28	0.000000000000000	0.000000000000000	0.000000000000000	0.000

In […

```
## Using a Pandas data frame, we can look at the convergence.
all_ps = {'p1' :p1, 'p2':p2, 'p3':p3, 'p4':p4, 'p5':p5, 'pn':pn}
#print(all_ps)

pd.DataFrame (dict([(k, pd.Series(v)) for k, v in all_ps.items()]))
```

Out[…

	p1	p2	p3	
0	1.350000000000000	1.350000000000000	1.350000000000000	1.350
1	1.599624999999998	1.416830055937340	1.372918879613796	1.367
2	0.000000000000000	0.000000000000000	0.000000000000000	0.000
3	0.000000000000000	0.000000000000000	0.000000000000000	0.000
4	0.000000000000000	0.000000000000000	0.000000000000000	0.000
5	0.000000000000000	0.000000000000000	0.000000000000000	0.000
6	0.000000000000000	0.000000000000000	0.000000000000000	0.000
7	0.000000000000000	0.000000000000000	0.000000000000000	0.000
8	0.000000000000000	0.000000000000000	0.000000000000000	0.000
9	0.000000000000000	0.000000000000000	0.000000000000000	0.000
10	0.000000000000000	0.000000000000000	0.000000000000000	0.000
11	0.000000000000000	0.000000000000000	0.000000000000000	0.000
12	0.000000000000000	0.000000000000000	0.000000000000000	0.000
13	0.000000000000000	0.000000000000000	0.000000000000000	0.000
14	0.000000000000000	0.000000000000000	0.000000000000000	0.000
15	0.000000000000000	0.000000000000000	0.000000000000000	0.000
16	0.000000000000000	0.000000000000000	0.000000000000000	0.000
17	0.000000000000000	0.000000000000000	0.000000000000000	0.000
18	0.000000000000000	0.000000000000000	0.000000000000000	0.000
19	0.000000000000000	0.000000000000000	0.000000000000000	0.000
20	0.000000000000000	0.000000000000000	0.000000000000000	0.000
21	0.000000000000000	0.000000000000000	0.000000000000000	0.000
22	0.000000000000000	0.000000000000000	0.000000000000000	0.000
23	0.000000000000000	0.000000000000000	0.000000000000000	0.000
24	0.000000000000000	0.000000000000000	0.000000000000000	0.000
25	0.000000000000000	0.000000000000000	0.000000000000000	0.000

	p1	p2	p3	
26	0.000000000000000	0.000000000000000	0.000000000000000	0.000
27	0.000000000000000	0.000000000000000	0.000000000000000	0.000
28	0.000000000000000	0.000000000000000	0.000000000000000	0.000

In […

```
## Testing Newton's Method
def g1(x):
  y = np.cos(x)
  return y

def g2(x):
  y = x + (np.cos(x) - x)/(np.sin(x) + 1)
  return y

p0 = np.pi / 4

max_it = 10
e = 1e-8


p1 = fixed_point(p0, e, max_it, g1)
p2 = fixed_point(p0, e, max_it, g1)

all_ps = {'p1':p1, 'p2':p2}
D = pd.DataFrame(dict([(k,pd.Series(v)) for k, v in all_ps.items()]))
pd.options.display.float_format = '{:,.15f}'.format
print(D)
```

```
max number of iteration exceeded
max number of iteration exceeded
                  p1                p2
0 0.785398163397448 0.785398163397448
1 0.707106781186548 0.707106781186548
2 0.000000000000000 0.000000000000000
3 0.000000000000000 0.000000000000000
4 0.000000000000000 0.000000000000000
5 0.000000000000000 0.000000000000000
6 0.000000000000000 0.000000000000000
7 0.000000000000000 0.000000000000000
8 0.000000000000000 0.000000000000000
9 0.000000000000000 0.000000000000000
```

In […

```
## Multiple Roots
def fixed_point(p0, e, max_it, g):
  p = []
  p.append(p0)
  i = 1
  while i <= max_it:
    try:
      p.append(g(p0))
    except:
      print('Arithmetic error')
      return(p)
```

```
    if abs(p[i] - p0) <= e:
      return(p)
    p0 = p[i]
    i += 1

  print('max number of iteration exceeded')
  return(p)
```

Question 2: Use the `matplotlib library to plot the function. Place a red dot on the figure where the estimated root is. Make sure that the xxx and yyy axes are visible.

In […

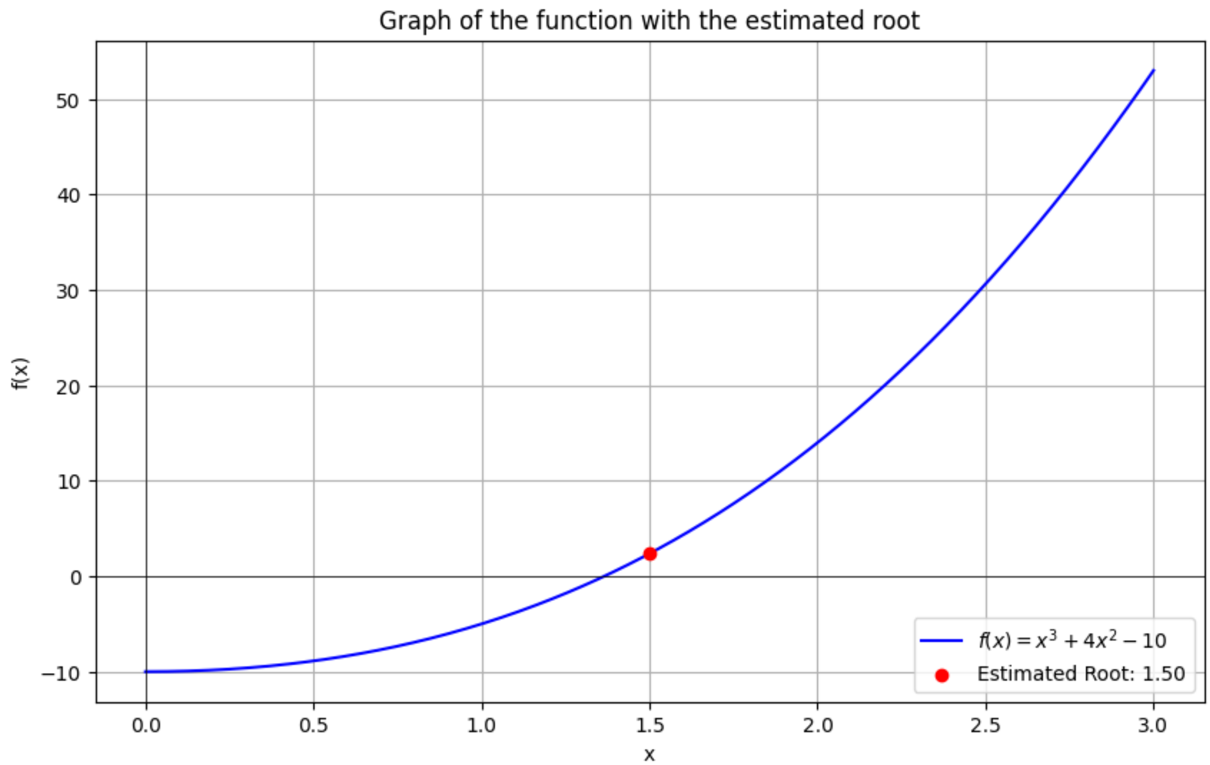
```
import matplotlib.pyplot as plt

# Get the estimated root using the newton_bisection method
pn = newton_bisection(a, b, e, max_it, f)
estimated_root = pn[0]
```

In […

```
x_values = np.linspace(0, 3, 400)
y_values = f(x_values)

# Plotting the function and the estimated root on the same graph
plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, label=r'$f(x) = x^3 + 4x^2 - 10$', color='blue'
plt.scatter(estimated_root, f(estimated_root), color='red', zorder=5, label=
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.title("Graph of the function with the estimated root")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid(True)
plt.legend()
plt.show()
```



Question 3: Show that for a fixed point iteration procedure pn+1=g(pn)pn+1=g(pn)p_{n+1}=g(p_n) with |g'(x)|≤K<1,|pn-p|≤Kn1-K|p1-p0|g '(x)|≤K<1,|pn-p|≤Kn1-K|p1-p0|g'(x)|\leq K < 1, |p_n-p| \leq \cfrac{K^n}{1-K} |p_1-p_0

Proof:

Given a fixed point iteration procedure defined by pn+1=g(pn)pn+1=g(pn) p_{n+1} = g(p_n) we aim to demonstrate that if |g'(x)|≤K<1|g'(x)|≤K<1 |g'(x)| \leq K < 1 for all (x), then |pn-p|≤Kn1-K|p1-p0||pn-p|≤Kn1-K|p1-p0| |p_n-p| \leq \frac{K^n}{1-K} |p_1-p_0| for all (n).

We will employ Mathematical Induction.

Consider the error iteration formula for fixed point iteration: en+1=|pn+1-p|en+1=|pn+1-p| e_{n+1} = |p_{n+1}-p| en=|pn-p|en=|pn-p| e_n = |p_n - p|

By invoking the Mean Value Theorem, which states:

If a function (f) is continuous on the closed interval ([a, b]) and differentiable on the open interval ((a, b)), then there exists at least one number (c) in the open interval ((a, b)) such that: f'(c)=f(b)-f(a)b-af '(c)=f(b)-f(a)b-a f'(c) = \frac{f(b)-f(a)}{b-a}

there exists a number (c) between (p) and (p_n) such that: g'(c)=g(pn)-g(p)pn-pg'(c)=g(pn)-g(p)pn-p g'(c) = \frac{g(p_n)-g(p)}{p_n-p}

Rearranging, we obtain: g(pn)-g(p)=g '(c)(pn-p)g(pn)-g(p)=g'(c)(pn-p) g(p_n)-g(p) = g'(c)(p_n-p)

Given (p_{n+1} = g(p_n)) and (p = g(p)) (since (p) is a fixed point), substituting into the above equation yields: pn+1-p=g'(c)(pn-p)pn+1-p=g'(c)(pn-p) p_{n+1}-p = g'(c)(p_n-p)

Taking the absolute value, we get: |pn+1-p|=|g '(c)|×|pn-p||pn+1-p|=|g'(c)|×|pn-p| |p_{n+1}-p| = |g'(c)| \times |p_n-p| or equivalently, en+1=|g '(c)|×enen+1=|g'(c)|×en e_{n+1} = |g'(c)| \times e_n

Given the condition (|g'(x)| \leq K), it follows that: en+1≤K×enen+1≤K×en e_{n+1} \leq K \times e_n

Base Case: For (n = 1): e2≤K×e1e2≤K×e1 e_2 \leq K \times e_1

This is validated by the previous equation.

Inductive Step:

Assuming the inequality holds for (n = k): ek+1≤Kk1-K×e1ek+1≤Kk1-K×e1 e_{k+1} \leq \frac{K^k}{1-K} \times e_1

From our derived inequality, we infer: ek+2≤K×ek+1ek+2≤K×ek+1 e_{k+2} \leq K \times e_{k+1}

Substituting our inductive assumption into this gives:

$$e_{k+2} \leq K \times \frac{K^k}{1-K} \times e_1$$

or

$$e_{k+2} \leq \frac{K^{k+1}}{1-K} \times e_1$$

By induction, this inequality stands for all n.

Lastly, given $e_1 = |p_1 - p_0|$, we deduce:

$$e_{n+1} \leq \frac{K^n}{1-K} \times |p_1-p_0|$$

This completes the proof of the given inequality.