Lab02

January 31, 2018

1 Lab 02

1.1 L1 [60] Bisection & Newton Method

```
In [107]: import numpy as np
          import matplotlib as plt
```

```
In [1]: def function_f(x):
            return x**3 - 4*x + 2
```

positive value region : [0, 1] , [1, 2]

1.1.1 a) Find the two positive roots

bisection method

```
In [73]: def fun_root(f, region):
             x_ini = region[0]
             x_end = region[1]
             i = 1
             while 1:
                 i = i + 1
                 print(i,"th step")
                 if f(x_ini) > 0.0:
                     x_temp = (x_ini + x_end)/2
                     if f(x_temp) > 0.0:
                         x_ini = x_temp
                     else:
                         x_end = x_temp

                     if x_end - x_ini < 0.0001:
                         #print(x_ini)
                         break
                 elif f(x_ini) < 0.0:
                     x_temp = (x_ini + x_end)/2
                     if f(x_temp) < 0.0:
                         x_ini = x_temp
                     else:
```

```
                x_end = x_temp

            if x_end - x_ini < 10**(-14):
                #print(x_ini)
                break
    return x_ini
```

In [74]:
```
#%%timeit
region_1 = [0.0, 1.0]
region_2 = [1.0, 2.0]

print("the first positive root is:", fun_root(function_f, region_1))
print("the second positive root is:", fun_root(function_f, region_2))
```

```
2 th step
3 th step
4 th step
5 th step
6 th step
7 th step
8 th step
9 th step
10 th step
11 th step
12 th step
13 th step
14 th step
15 th step
the first positive root is: 0.5391845703125
2 th step
3 th step
4 th step
5 th step
6 th step
7 th step
8 th step
9 th step
10 th step
11 th step
12 th step
13 th step
14 th step
15 th step
16 th step
17 th step
```

```
18 th step
19 th step
20 th step
21 th step
22 th step
23 th step
24 th step
25 th step
26 th step
27 th step
28 th step
29 th step
30 th step
31 th step
32 th step
33 th step
34 th step
35 th step
36 th step
37 th step
38 th step
39 th step
40 th step
41 th step
42 th step
43 th step
44 th step
45 th step
46 th step
47 th step
48 th step
the second positive root is: 1.6751308705666403
```

Newton method

```
In [2]: def fun_dev(x):
            return 3*x**2 - 4
```

```
In [69]: def Newton_root(f, f_dev, point):
             x_ini = point
             x_next = 0.0
             i = 1

             while 1:
                 i = i+1
                 print(i,"th step")
```

```
            x_next = x_ini - f(x_ini)/f_dev(x_ini)
            #print("x_next is:", x_next)
            #print("x_ini is:", x_ini)

            if abs(x_next - x_ini) < 10**(-14):
                break
            else:
                x_ini = x_next

            #print(x_ini)

        return x_next
```

```
In [70]: #%%timeit
         print("the first positive root is:",Newton_root(function_f, fun_dev, 0.0))

         print("the second positive root is:",Newton_root(function_f, fun_dev, 2.0))
```

```
2 th step
3 th step
4 th step
5 th step
6 th step
7 th step
the first positive root is: 0.5391888728108891
2 th step
3 th step
4 th step
5 th step
6 th step
7 th step
the second positive root is: 1.6751308705666461
```

```
In [66]: #%%timeit
         Newton_root(function_f, fun_dev, 0.0)

         Newton_root(function_f, fun_dev, 2.0)
```

```
7.42 ţs ś 236 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)
```

1.1.2 Newton method : 7 steps

1.1.3 bisection method: 15 to 50 steps

1.2 L2 [40] Ballistic Trajectory

```
In [114]: g = 1.0
          f = np.array([0, -g])
          alpha = 0
          v0 = 1.0
          vel_ini = np.array([v0*np.cos(alpha), v0*np.sin(alpha)])
          vel_next = np.array([0.0, 0.0])

          r_ini = np.array([0.0, 1.0])

          dt = 0.1

          x_position = []
          y_position = []

          while r_ini[1] > 0.0:
              #print(r_next[1])

              r_next = r_ini + vel_ini*dt
              vel_next = vel_ini + f*dt

              r_ini = r_next
              vel_ini = vel_next

              x_position.append(r_next[0])
              y_position.append(r_next[1])
```

```
In [118]: y_position
```

```
Out[118]: [1.0,
           0.9899999999999999,
           0.9699999999999997,
           0.9399999999999995,
           0.8999999999999991,
           0.8499999999999987,
           0.7899999999999981,
           0.7199999999999986,
           0.6399999999999999,
           0.5499999999999993,
           0.4499999999999996,
           0.3399999999999997,
           0.2199999999999997,
```

```
          0.089999999999999969,
          -0.050000000000000044]
```

In [119]:
```
x_position
```

Out[119]:
```
[0.10000000000000001,
 0.20000000000000001,
 0.30000000000000004,
 0.40000000000000002,
 0.5,
 0.59999999999999998,
 0.69999999999999996,
 0.79999999999999993,
 0.89999999999999991,
 0.99999999999999989,
 1.0999999999999999,
 1.2,
 1.3,
 1.4000000000000001,
 1.5000000000000002]
```