

## Lab 02. Searching with interpolation methods

Mohamed Stifi

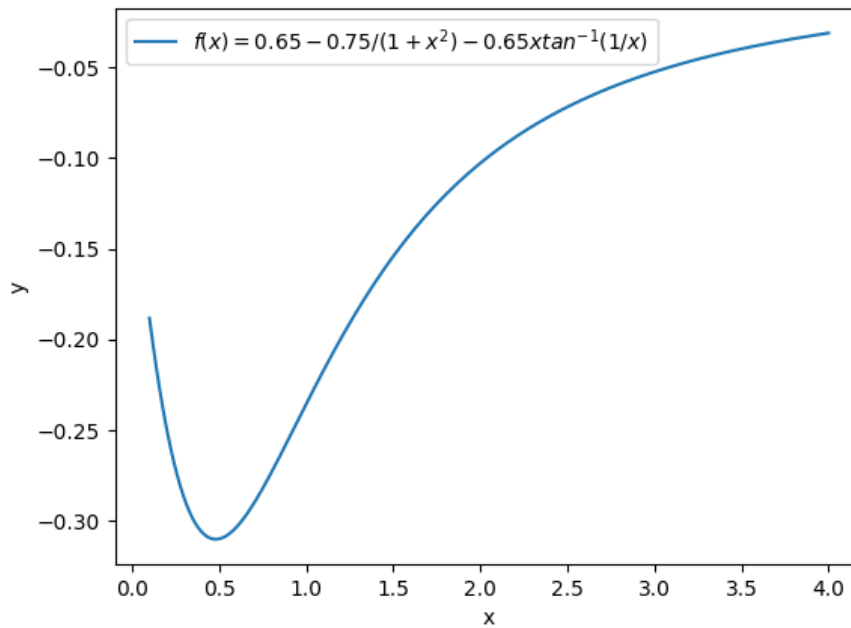
2IA

### 1 Code

#### 1.1 global variable and library

```
1 import numpy as np
2 from numdifftools import Derivative as Df
3 import matplotlib.pyplot as plt
4 from matplotlib.animation import FuncAnimation
5 from time import time
6
7 #-----| global variable |-----
8 f = lambda x : 0.65 - 0.75/(1 + x**2) - 0.65*x*np.arctan(1/x)
9 df = Df(f)
10 d2f = Df(df)
11 x0 = 0.2
12 a = 0.2
13 b = 3
14 delta = 1e-3
15 X = np.linspace(0.1, 4, 200)
16 Y1 = f(X)
17 Y2 = df(X)
18 #-----| plot the function |-----
19 plt.plot(X, Y1, label = "$f(x) = 0.65 - 0.75/(1+x^2) - 0.65x\tan^{-1}(1/x)$")
20 plt.xlabel('x')
21 plt.ylabel('y')
22 plt.legend()
23 plt.show()
```

##### 1.1.1 Screenshots



## 1.2 A python implementation of the optimization methods

```

1
2 #-----| Newton-Rapson method |-----
3 def Newton_Rapson_method( df , d2f, x0,num_iter_max= 1000, eps = 1e-9)
4     ↪ :
5     """
6     params:
7         df : the first derivative of f
8         d2f : the second derivative of f
9         x0 : The initial guess for x optimal
10        num_iter_max : Maximum number of iterations (default set to
11        ↪ 1000)
12        eps : A small value (default set to 1e-9)
13    return :
14        x0 : the final x optimal
15        iter_ : the final iterations
16    """
17    f1 = df(x0)
18    iter_ = 0
19    while np.abs(f1) > eps and num_iter_max > iter_ :
20        f2 = d2f(x0)
21        if f2 != 0 :
22            x0 -= f1/f2
23            f1 = df(x0)
24            iter_ += 1
25        else :
26            x0 -= f1/eps
27            f1 = df(x0)
28            iter_ += 1
29
30    return x0 , iter_
31 #-----|Quasi Newton Method |-----

```

```

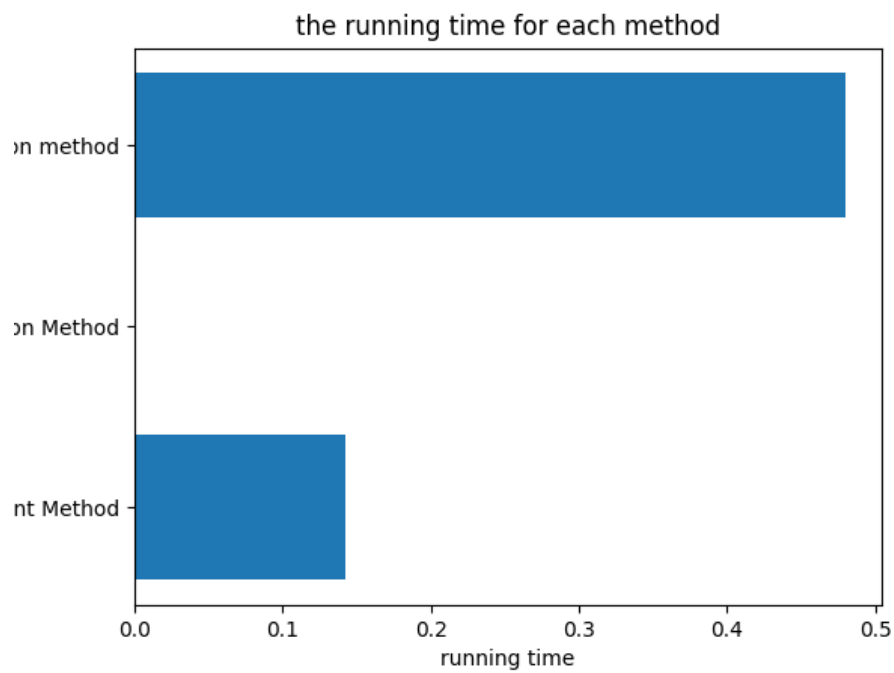
31 def Quasi_Newton_Method(f, x0, delta, num_iter_max = 1000, eps = 1e-8):
32     """
33     params:
34         f : The target function f
35         x0 : The initial guess for x optimal
36         num_iter_max : Maximum number of iterations (default set to
37             ↪ 1000)
38         delta : A small value
39         eps : A small value (default set to 1e-9)
40     return :
41         x0 : the final x optimal
42         iter_ : the final iterations
43     """
44     f1 = f(x0 + delta)
45     f2 = f(x0 - delta)
46     delta_2 = 2*delta
47     df = (f1 - f2)/delta_2
48     iter_ = 0
49     while np.abs(df) > eps and num_iter_max > iter_ :
50         x0 -= (delta*(f1 - f2))/(2*(f1 - 2*f(x0) + f2))
51         f1 = f(x0 + delta)
52         f2 = f(x0 - delta)
53         df = (f1 - f2)/delta_2
54         iter_ += 1
55     return x0 , iter_
56
57 #-----| Secant Method |-----
58 def Secant_Method(df, a, b, num_iter_max = 1000, eps = 1e-8):
59     """
60     params:
61         df
62         a : The initial value of the lower bound
63         b : The initial value of the upper bound
64         num_iter_max : Maximum number of iterations (default set to
65             ↪ 1000)
66         eps : A small value (default set to 1e-9)
67     return :
68         x0 : the final x optimal
69         iter_ : the final iterations
70     """
71     dfx = df(a)
72     x = a
73     iter_ = 0
74     while np.abs(dfx) > eps and num_iter_max > iter_ :
75         df1 = df(a)
76         df2 = df(b)
77         x = a - (df1*(b- a))/(df2 - df1)
78         dfx = df(x)
79         if dfx >= 0 :
80             b = x
81             df2 = dfx
82         else :
83             a = x
84             df1 = dfx
85         iter_ += 1
86     return x , iter_

```

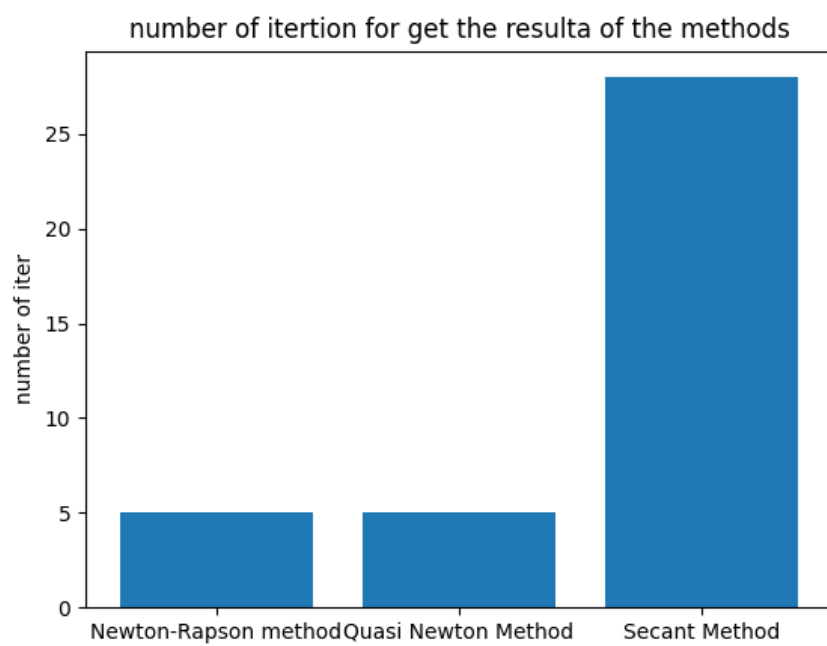
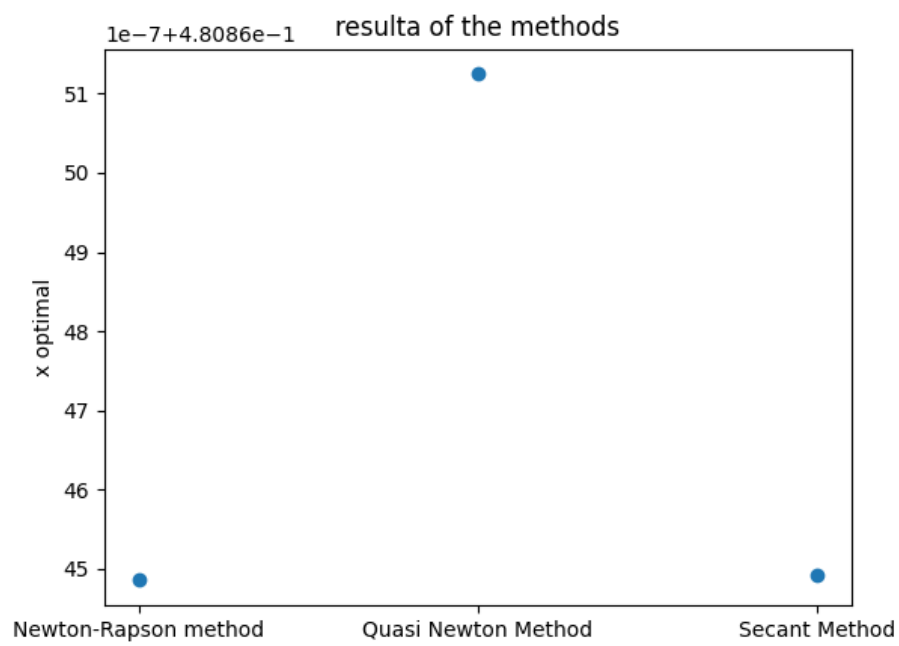
### 1.3 Compute the running time for each method

```
1 #-----| time complicity |-----
2 methods = ['Newton-Rapson method', 'Quasi Newton Method', 'Secant
   ↪ Method']
3 times = []
4 x_optm = []
5 optm_iter = []
6 t1 = time()
7 N_x , N_iter_ = Newton_Rapson_method( df, d2f, x0,num_iter_max= 1000,
   ↪ eps = 1e-9)
8 t2 = time()
9 times.append(t2-t1)
10 x_optm.append(N_x)
11 optm_iter.append(N_iter_)
12 # -----
13 t1 = time()
14 Q_x , Q_iter_ = Quasi_Newton_Method(f, x0, delta,num_iter_max = 1000,
   ↪ eps = 1e-8)
15 t2 = time()
16 times.append(t2-t1)
17 x_optm.append(Q_x)
18 optm_iter.append(Q_iter_)
19 # -----
20 t1 = time()
21 S_x , S_iter_ = Secant_Method(df, a, b, num_iter_max = 1000, eps = 1e
   ↪ -8)
22 t2 = time()
23 times.append(t2-t1)
24 x_optm.append(S_x)
25 optm_iter.append(S_iter_)
26
27 # -----
28 print('times : ', times)
29 fig, ax = plt.subplots()
30 y = np.arange(len(times))
31 ax.barh(y ,times, align='center')
32 ax.set_yticks(y, labels=methods)
33 ax.invert_yaxis() # labels read top-to-bottom
34 ax.set_xlabel('running time')
35 ax.set_title('the running time for each method')
36
37 plt.show()
38
39 #-----
40 print('x optimal : ',x_optm)
41 plt.scatter(methods,x_optm)
42 plt.ylabel('x optimal')
43 plt.title('resulta of the methods')
44 plt.show()
45 ## -----
46 print('number of iterition : ',optm_iter)
47 plt.bar(methods,optm_iter)
48 plt.ylabel('number of iter')
49 plt.title('number of iteration for get the resulta of the methods')
50 plt.show()
```

### 1.3.1 Screenshots



```
times : [0.48031139373779297, 0.0, 0.14236807823181152]
x optimal : [0.4808644852929133, 0.48086512483253496, 0.48086449227165107]
number of iteration : [5, 5, 28]
```



## 1.4 Newton-Rapson method animation

```
1 # Create the figure and subplots
2 fig, axs = plt.subplots(1, 2)
3 # Set the x-axis and y-axis labels for each subplot.
4 axs[0].set_title("f")
5 axs[1].set_title("f'")
6 axs[0].set_xlabel('X')
7 axs[0].set_ylabel('f')
8 axs[1].set_xlabel('X')
9 axs[1].set_ylabel("f'")
10 # Initialize empty scatter plots for animation
11 axs[0].plot(X,Y1, c='b')
12 axs[1].plot(X,Y2, c='b')
13 sc1 = axs[0].scatter([], [], c='y')
14 sc2 = axs[1].scatter([], [], c='y')
15
16 # Function
17 def Newton_Rapson_method(f, df, d2f, x0, num_iter_max=1000, eps=1e-9):
18     xv, y1, y2 = [x0], [f(x0)], [df(x0)]
19     iter_ = 0
20     while abs(y2[-1]) > eps and num_iter_max > iter_:
21         f2 = d2f(xv[-1])
22         if f2 != 0:
23             xv.append(xv[-1] - y2[-1] / f2)
24             y1.append(f(xv[-1]))
25             y2.append(df(xv[-1]))
26             iter_ += 1
27         else:
28             xv.append(xv[-1] - y2[-1] / eps)
29             y1.append(f(xv[-1]))
30             y2.append(df(xv[-1]))
31             iter_ += 1
32     return xv, y1, y2
33
34 # get the values
35 xv, y1, y2 = Newton_Rapson_method(f, df, d2f, x0, num_iter_max=1000,
36     ↪ eps=1e-9)
37
38 # Animation update function
39 def update(frame):
40     x1, y1_, x2, y2_ = xv[:frame], y1[:frame], xv[:frame], y2[:frame]
41     sc1.set_offsets(np.c_[x1, y1_])
42     sc2.set_offsets(np.c_[x2, y2_])
43     return sc1, sc2
44
45 # Create the animation
46 ani = FuncAnimation(fig, update, frames=len(xv), blit=True, interval
47     ↪ =300)
48
49 # Display the plot
50 plt.show()
```

### 1.4.1 Screenshots

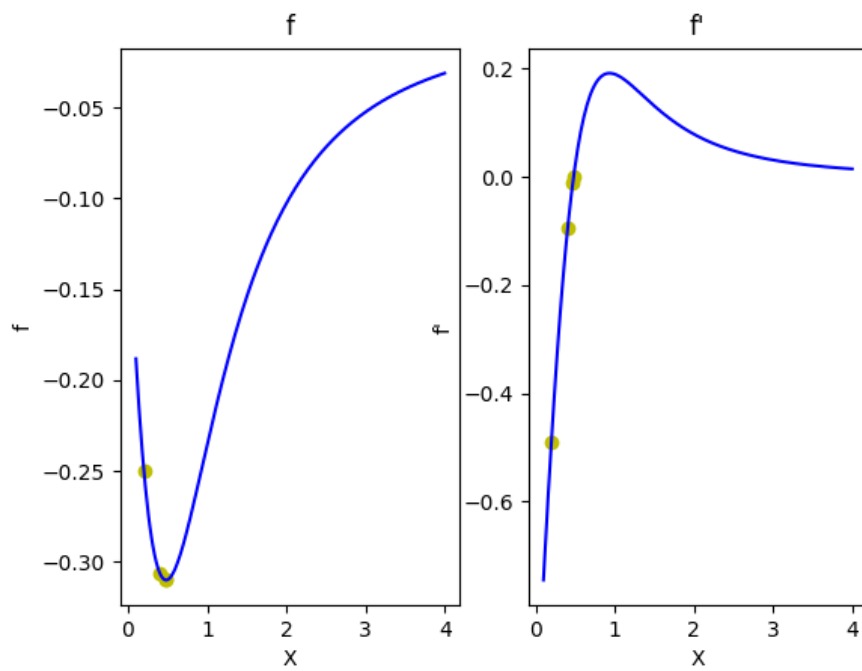


Figure 1: les valeurs de  $f$  et  $f'$  à chaque étape de méthode Newton-Rapson



## 1.5 Quasi Newton Method animation

```
1
2 # Create the figure and subplots
3 fig, axs = plt.subplots(1, 2)
4 # Set the x-axis and y-axis labels for each subplot.
5 axs[0].set_title("f")
6 axs[1].set_title("f'")
7 axs[0].set_xlabel('X')
8 axs[0].set_ylabel('f')
9 axs[1].set_xlabel('X')
10 axs[1].set_ylabel("f'")
11 # Initialize empty scatter plots for animation
12 axs[0].plot(X,Y1, c='b')
13 axs[1].plot(X,Y2, c='b')
14 sc1 = axs[0].scatter([], [], c='y')
15 sc2 = axs[1].scatter([], [], c='y')
16
17 # Function
18
19 def Quasi_Newton_Method(f, x0, delta, num_iter_max = 1000, eps = 1e-8):
20     f1 = f(x0 + delta)
21     f2 = f(x0 - delta)
22     delta_2 = 2*delta
23     df = (f1 - f2)/delta_2
24     xv, y, y1 = [x0],[f(x0)],[df]
25     iter_ = 0
26     while np.abs(df) > eps and num_iter_max > iter_ :
27         x0 -= (delta*(f1 - f2))/(2*(f1 - 2*f(x0) + f2))
28         f1 = f(x0 + delta)
29         f2 = f(x0 - delta)
30         df = (f1 - f2)/delta_2
31         xv.append(x0)
32         y.append(f(x0))
33         y1.append(f1)
34         iter_ += 1
35     return xv, y, y1
36
37 # get the values
38 xv, y1, y2 = Quasi_Newton_Method(f, x0, delta, num_iter_max = 1000, eps
    ↪ = 1e-8)
39
40 # Animation update function
41 def update(frame):
42     x1, y1_, x2, y2_ = xv[:frame], y1[:frame], xv[:frame], y2[:frame]
43     sc1.set_offsets(np.c_[x1, y1_])
44     sc2.set_offsets(np.c_[x2, y2_])
45     return sc1, sc2
46
47 # Create the animation
48 ani = FuncAnimation(fig, update, frames=len(xv), blit=True, interval
    ↪ =300)
49
50 # Display the plot
51 plt.show()
```

### 1.5.1 Screenshots

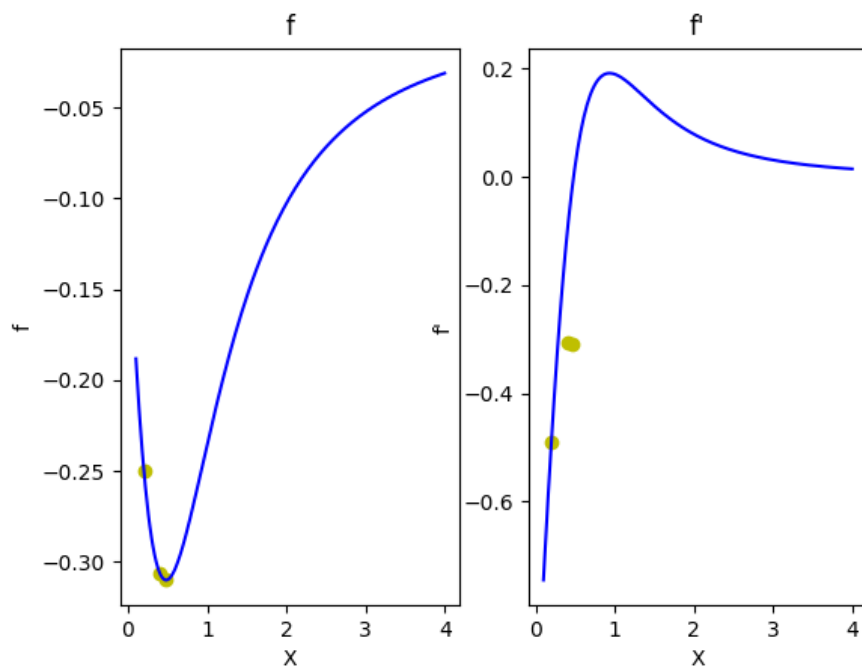


Figure 2: les valeurs de  $f$  et  $f'$  à chaque étape de Quasi méthode Newto

## 1.6 Secant Method Animation

```
1 # Create the figure and subplots
2 fig, axs = plt.subplots(1, 2)
3 # Set the x-axis and y-axis labels for each subplot.
4 axs[0].set_title("f")
5 axs[1].set_title("f'")
6 axs[0].set_xlabel('X')
7 axs[0].set_ylabel('f')
8 axs[1].set_xlabel('X')
9 axs[1].set_ylabel("f'")
10 # Initialize empty scatter plots for animation
11 axs[0].plot(X,Y1, c='b')
12 axs[1].plot(X,Y2, c='b')
13 sc1 = axs[0].scatter([], [], c='y')
14 sc2 = axs[1].scatter([], [], c='y')
15
16 # Function
17
18 def Secant_Method(f, df, a, b, num_iter_max = 1000, eps = 1e-8):
19     dfx = df(a)
20     x = a
21     xv, y, y1 = [x],[f(x)],[dfx]
22     iter_ = 0
23     while np.abs(dfx) > eps and num_iter_max > iter_ :
24         df1 = df(a)
25         df2 = df(b)
26         x = a - (df1*(b- a))/(df2 - df1)
27         dfx = df(x)
28         xv.append(x)
29         y.append(f(x))
30         y1.append(dfx)
31         if dfx >= 0 :
32             b = x
33             df2 = dfx
34         else :
35             a = x
36             df1 = dfx
37         iter_ += 1
38     return xv, y, y1
39 # get the values
40 xv, y1, y2 = Secant_Method(f, df, a, b, num_iter_max = 1000, eps = 1e
    ↪ -8)
41
42 # Animation update function
43 def update(frame):
44     x1, y1_, x2, y2_ = xv[:frame], y1[:frame], xv[:frame], y2[:frame]
45     sc1.set_offsets(np.c_[x1, y1_])
46     sc2.set_offsets(np.c_[x2, y2_])
47     return sc1, sc2
48
49 # Create the animation
50 ani = FuncAnimation(fig, update, frames=len(xv), blit=True, interval
    ↪ =300)
51
52 # Display the plot
53 plt.show()
```

### 1.6.1 Screenshots

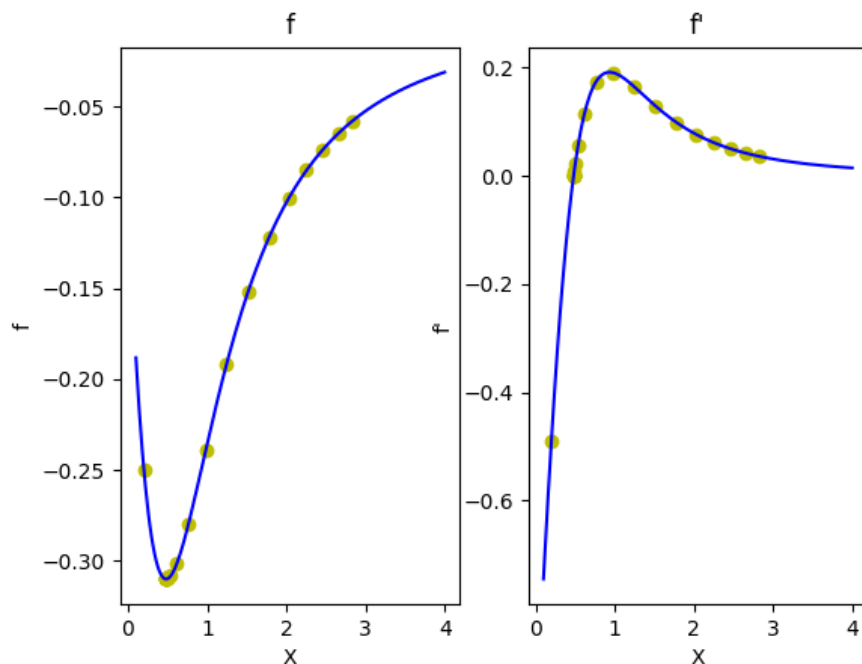


Figure 3: les valeurs de  $f$  et  $f'$  à chaque étape de Secant méthode Newto

### 1.7 Compare the running time of these algorithms and those from lab 1

```

1 from LAB1 import search_with_fixe_step, search_with_accelerat_step, \
2     exhaustive_Search, dichotomous_Search, \
3     interval_Halving, Fibonacci, \
4     golden_Section
5 from main import Newton_Rapson_method, Quasi_Newton_Method, \
6     Secant_Method
7 import time
8 import numpy as np
9 from numdifftools import Derivative as Df
10 import matplotlib.pyplot as plt
11 # -----
12 f = lambda x : 0.65 - 0.75/(1 + x**2) - 0.65*x*np.arctan(1/x)
13 df = Df(f)
14 d2f = Df(df)
15 x0 = 0.2
16 a = 0.2
17 b = 3
18 delta = 1e-3
19 X = np.linspace(0.1, 4, 200)
20 Y = f(X)
21 # -----
22 methods = ['Search method with fixed step size',
23            'Search method with accelerated step size', 'Exhaustive search
24            ↪ method',
25            'Dichotomous search method', 'Interval halving method',
26            'Fibonacci method', 'Golden section method',
27            'Newton-Rapson method', 'Quasi Newton Method', 'Secant Method']

```

```

27 time_of_methods = []
28 # -----
29 t1 = time.time()
30 search_with_fixe_step(f, x_init = 0.01 ,step = 0.00005)
31 t2 = time.time()
32 time_of_methods.append(t2-t1)
33 # -----
34 t1 = time.time()
35 search_with_accelerat_step(f, x_init = 0.01, step = 0.00005)
36 t2 = time.time()
37 time_of_methods.append(t2-t1)
38 # -----
39 t1 = time.time()
40 exhaustive_Search(f, 0.01, 2.0, 10000)
41 t2 = time.time()
42 time_of_methods.append(t2-t1)
43 # -----
44 t1 = time.time()
45 dichotomous_Search(f,0.01, 2.0 , 0.001, eps = 1e-9, max_iter = 10000)
46 t2 = time.time()
47 time_of_methods.append(t2-t1)
48 # -----
49 t1 = time.time()
50 interval_Halving(f,0.01, 2.0 , eps = 1e-9, max_iter = 10000)
51 t2 = time.time()
52 time_of_methods.append(t2-t1)
53 # -----
54 t1 = time.time()
55 Fibonacci(f, 0.01, 2.0, 50)
56 t2 = time.time()
57 time_of_methods.append(t2-t1)
58 # -----
59 t1 = time.time()
60 golden_Section(f, x_l= 0.01, x_u = 2.0, eps = 1e-5)
61 t2 = time.time()
62 time_of_methods.append(t2-t1)
63 # -----
64 t1 = time.time()
65 Newton_Rapson_method( df, d2f, x0,num_iter_max= 1000, eps = 1e-9)
66 t2 = time.time()
67 time_of_methods.append(t2-t1)
68 # -----
69 t1 = time.time()
70 Quasi_Newton_Method(f, x0, delta,num_iter_max = 1000, eps = 1e-8)
71 t2 = time.time()
72 time_of_methods.append(t2-t1)
73 # -----
74 t1 = time.time()
75 Secant_Method(df, a, b, num_iter_max = 1000, eps = 1e-8)
76 t2 = time.time()
77 time_of_methods.append(t2-t1)
78 # -----
79 fig, ax = plt.subplots()
80 y = np.arange(len(time_of_methods))
81 ax.barh(y ,time_of_methods, align='center')
82 ax.set_yticks(y, labels=methods)
83 ax.invert_yaxis() # labels read top-to-bottom
84 ax.set_xlabel('running time')

```

```
85 ax.set_title('the running time for each method')
86 plt.show()
```

### 1.7.1 Screenshots

