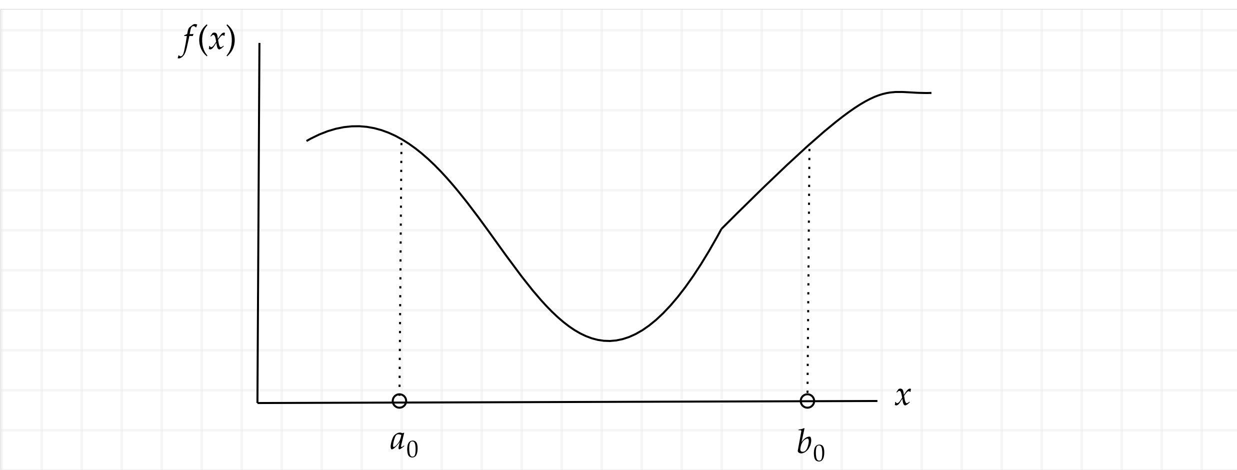
ONE_DIMENSIONAL SEARCH METHODS

❑ Introduction

In this chapter, we are interested in the problem of minimizing an objective function $f : R \to R(i.e.\ One - dimensional\ problem)$. The approach is to use an iterative search algorithm, also called a line-search method. One-dimensional search methods are of interest for the following reasons. First, they are special case of search methods used in multivariate algorithms.

In an iterative algorithm,we start with an initial candidate solution $x^{(0)}$ and generate a sequence of $iterates\ x^{(1)}, x^{(2)}, \ldots$ For each iteration $k = 0, 1, 2, \ldots,$ the next point $x^{(k+1)}\ depends\ on\ x^{(k)}$ and the objective function f. The algorithm may use the value of f at specific points, or perhaps its first derivative f', or even its second derivative f''.
The algorithms studies in this chapter :

1. Golden Section Method (uses only f)
2. Fibonacci Method(uses only f)
3. Bisection Method (uses only f')
4. Secant Method (uses only f')
5. Newton Method (uses $f'$and f'')



Unimodal Function

❑ Golden Section Search

The search methods we discuss in this and the next two section allow us to determine the minimizer of an objective function $f : R \to R$ over a closed interval, say $[a_0, b_0]$. The only property we assume of the objective function f is that it is $Unimodal$, which means f has only one local minimizer.

The methods we discuss are based on evaluating the objective function at different points in the interval $[a_0, b_0]$.

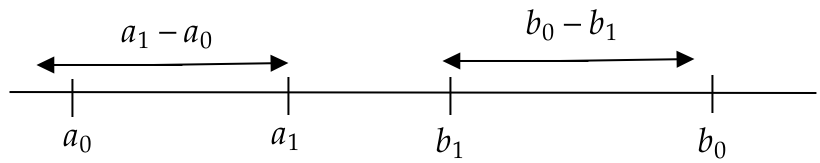
Consider a *unimodal* function f of one variable and the interval $[a_0, b_0]$. If we evaluate f at only one intermediate point of the interval, we cannot narrow the range within which we know the minimizer is located. *We have to evaluate f at two intermediate points. We choose the intermediate points in such a way that the reduction in the range is symmetric, in the sense that :*

$$a_1 - a_0 = b_0 - b_1 = \rho(b_0 - a_0),$$

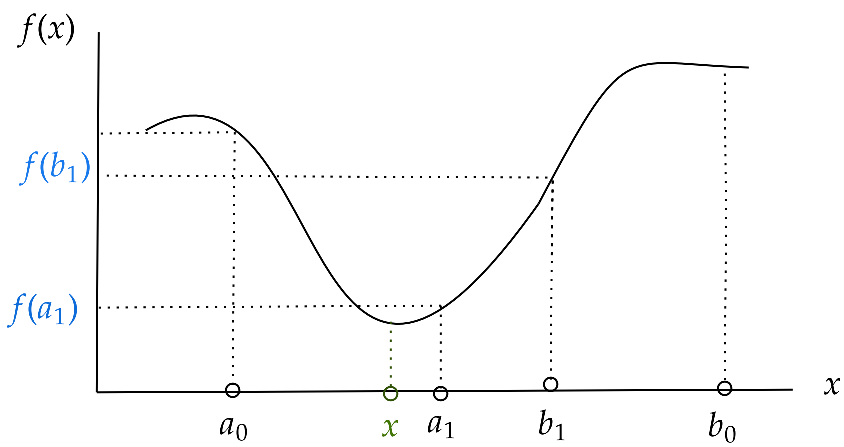
where

$$\rho < \frac{1}{2}$$

We then evaluate f at the intermediate points. If $f(a_1) < f(b_1)$, then the minimizer must lie in the range $[a_0, b_1]$. If, on the other hand, $f(a_1) > f(b_1)$, then the minimizer is located in the range $[a_1, b_0]$.



Evaluating the objective function at two intermediate points



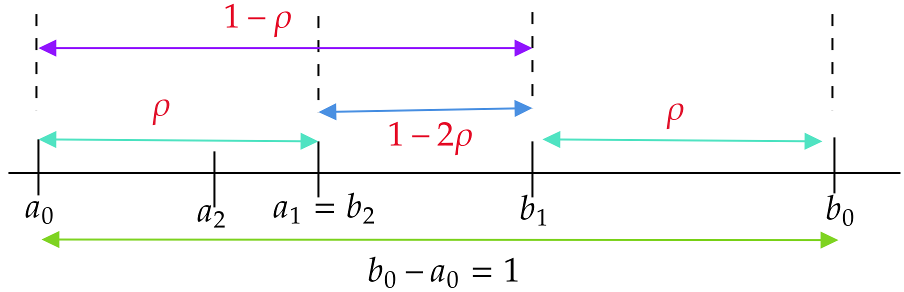
The case where $f(a_1) < f(b_1)$; *the minimizer* $x^* \in [a_0, b_1]$

Starting with the reduced range of uncertainty, we can repeat the process and similarly find two points, say a_2 *and* b_2, using the same value of $\rho < \frac{1}{2}$ as before. However, we would like to minimize the number of objective function evaluations while reducing the width of the uncertainty interval. Suppose, for example , that $f(a_1) < f(b_1)$. Then we know that $x^* \in [a_0, b_1]$. Because $a_1$is already in the uncertainty interval and $f(a_1)$ is already known, we can make a_1 coincide with b_2. Thus, only one new evaluation of f at a_2 would be necessary. To fine the value of ρ that results in only one new evaluation of f. Without loss of generality, imagine that the original range $[a_0, b_0]$ is of unit length. Then, to have only new evaluation of f it is enough to choose ρ so that:

$$\rho(b_1 - a_0) = b_1 - b_2$$

Because $b_1 - a_0 = 1 - \rho$ and $b_1 - b_2 = 1 - 2\rho, we\ have:$

$$\rho(1 - \rho) = 1 - 2\rho$$



Finding value of ρ resulting in only one new evaluation of f.

We write the quadratic equation as :

$$\rho^2 - 3\rho + 1 = 0$$

The solutions are :

$$\rho_1 = \frac{3+\sqrt{5}}{2} \ , \ \rho_2 = \frac{3-\sqrt{5}}{2}$$

Because we require that $\rho < \frac{1}{2}$, we take

$$\rho = \frac{3-\sqrt{5}}{2} \approx 0.382$$

Observe that

$$1-\rho = \frac{\sqrt{5}-1}{2}$$

$$\textit{and} \ \frac{\rho}{1-\rho} = \frac{3-\sqrt{5}}{\sqrt{5}-1} = \frac{\sqrt{5}-1}{2} = \frac{1-\rho}{1}$$

that is

$$\frac{\rho}{1-\rho} = \frac{1-\rho}{1}.$$

Thus, dividing a range in the ration of ρ *to* $1-\rho$ has the effect that the ratio of the shorter

segment to the longer equals the ratio of the longer to the sum of the two. This rule was referred to by ancient geometers as the *golden section*.
Using the golden section rule means that at every stage of the uncertainty range reduction (except the first), the objective function f need only to be evaluated at one new point. The uncertainty range is reduced by the ratio $1-\rho \approx 0.61803$ at every stage. Hence, N steps of reduction using the golden section method reduces the range by the factor.

$$(1-\rho)^N \approx (0.61803)^N$$

Example : Suppose we wish to use the golden section search method to find the value of x that minimizes

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

in the interval [0,2]. We wish to locate the value of x within a range 0.3.

Solution :
After N stages the interval is reduced by $(0.61803)^N$. So, we choose N so that

$$\begin{gathered}(0.61803)^N \leq 0.3/2 \\ N\, log(0.61803) \ \leq log(0.3/2) \\ N \leq (-0.8239)/(-0.2089) \\ N \geq 3.94.\end{gathered}$$

Four stages of reduction will do; that is, $N = 4$.

$Iteration: 1$. We evaluate f at two intermediate points a_1 and b_1. We have

$$\begin{aligned} a_1 &= a_0 + \rho(b_0 - a_0) \Longrightarrow 0 + 0.3819(2) = 0.7638 \\ b_1 &= a_0 + (1-\rho)(b_0 - a_0) = 0 + 0.6181*2 = 1.2362 \end{aligned}$$

where $\rho = \left(3-\sqrt{5}\right)/2$. We compute :

$$\begin{gathered} f(a_1) = 0.763^4 - 14*0.763^3 + 60*0.763^2 - 70*0.763 = -24.35 \\ f(b_1) = 1.2362^4 - 14*1.2362^3 + 60*1.2362^2 - 70*1.2362 = -18.96 \end{gathered}$$

Thus, $f(a_1) < f(b_1)$, so the uncertainty interval is reduced to

$$[a_0, b_1] = [0, 1.236]$$

$Iteration: 2$. We choose b_2 to coincide with a_1, so f need only be evaluated at one new point

$$a_2 = a_0 + \rho(b_1 - a_0) = 0 + 0.3819*1.236 = 0.472$$

We have

$$f(a_2) = f(0.472) = 0.472^4 - 14*0.472^3 + 60*0.472^2 - 70*0.472 = -21.09$$

Thus, $f(b_2) < f(a_2)$, so the uncertainty interval is reduced to

$$[a_2, b_1] = [0.472, 1.236]$$

$Iteration: 3$. We choose a_3 to coincide with b_2, so f need only be evaluated at one new point

$$b_3 = a_2 + (1-\rho)(b_1 - a_2) = 0.472 + 0.6181*[1.236 - 0.472] = 0.944$$

We have

$$f(b_3) = f(0.944) = 0.944^4 - 14*0.944^3 + 60*0.944^2 - 70*0.944 = -23.59$$

Thus, $f(a_3) < f(b_3)$. Checking the interval length : $|\ 1.236 - 0.472\ | = 0.764.$ Hence the uncertainty interval is further reduced to :

$$[a_2, b_3] = [0.472, 0.944]$$

$Iteration: 4$. We choose b_4 to coincide with a_3, so f need only be evaluated at one new point

$$a_4 = a_2 + (1-\rho)(b_3 - a_2) = 0.472 + 0.3819*[0.944 - 0.472] = 0.6525$$

We have :

$$f(a_4) = 0.6525^4 - 14*0.6525^3 + 60*0.6525^2 - 70*0.6525 = -23.84$$
$$f(b_4) = f(a_3) = -24.36.$$

Thus, $f(a_4) > f(b_4)$. Thus, the value of x that minimizes f is located in the interval

$$[a_4, b_3] = [0.6525, 0.9443]$$

Note that $b_3 - a_4 = 0.292 < 0.3$.

Link to the Colab Notebook : [Golden Section Method Algorithm]

```
# Function definition
def fun(x):
```

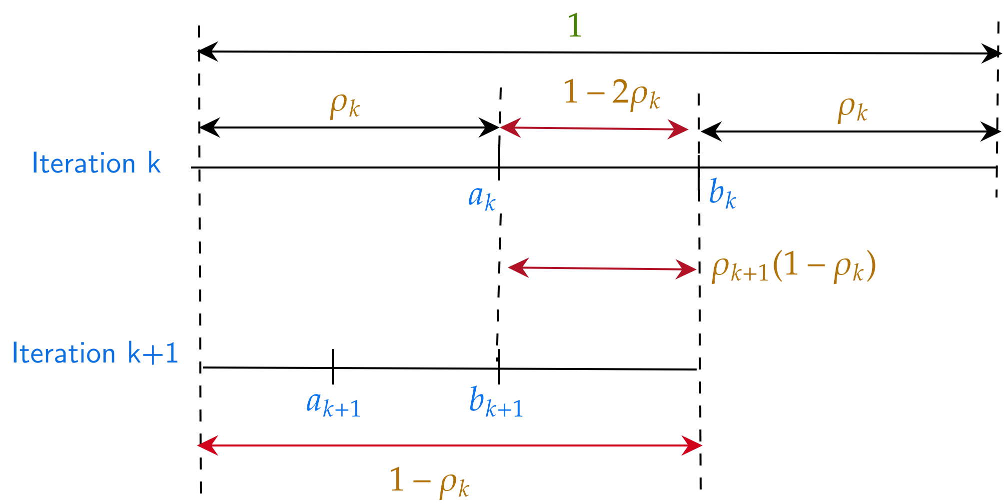
```
  val = x**4-14*x**3+60*x**2-70*x
  return val
# Importting required python libraries
import math
import numpy as np
from termcolor import colored

# Golden section search function
def golden_section_search(a,b,ro):

    ''' The number of iterations needed to reach the given range 
between two values,i.e|a-b|< epsilon
         (1-ro)^N <= 0.3(final range)/2(given initial range)'''

    N=math.ceil(np.log10(0.3/2)/np.log10(0.6181))

    ''' We will use iteration to arrive at the required range.
      Since, for the first time we need to calculate 2 evaluation 
points, we will keep the first iteration
      out of the iteration look'''
    #Iteration : 1
    a1 = a + ro*(b-a)
    b1 = a + (1-ro)*(b-a)
    f1= fun(a1)
    print("Value of function evaluation on the left of the 
minimum:",f1,"\t")
    f2 = fun(b1)
    print("Value of function evaluation on the right of the 
minimum:",f1,"\t")

    for i in range(1,N+2):
      f1= fun(a1)
      f2= fun(b1)
      if abs(a-b) <= 0.3:
        print("The final range between a & b : ",abs(b-a))
        return
      if f1 <= f2:
        print("Current values of a & b","a:",a,"b:",b,"\t")
```

```
        print("Iteration :",i,"value of function evaluation","f1:
",f1," f2: ",f2,"\t")
        b=b1
        b1=a1
        a1 = a + ro*(b-a)
      else:
        print("Current values of a & b","a:",a,"b:",b,"\t")
        print("Iteration :",i,"value of function evaluation","f1:
",f1," f2: ",f2,"\t")
        a = a1
        a1 = b1
        b1 =  a + (1-ro)*(b-a)

#Execution of the function
a=0
b=2
ro = 0.3819
golden_section_search(a,b,ro)

#Output
Value of function evaluation on the left of the minimum:
-24.360539847524606
Value of function evaluation on the right of the minimum:
-24.360539847524606
Current values of a & b a: 0 b: 2
Iteration : 1 value of function evaluation f1:
-24.360539847524606  f2:  -18.955293886084604
Current values of a & b a: 0 b: 1.2362
Iteration : 2 value of function evaluation f1:  -21.09781971314749
  f2:  -24.360539847524606
Current values of a & b a: 0.47210478 b: 1.2362
Iteration : 3 value of function evaluation f1:
-24.360539847524606  f2:  -23.591352580102786
Current values of a & b a: 0.47210478 b: 0.9443920354819999
Iteration : 4 value of function evaluation f1:  -23.83739668077605
  f2:  -24.360539847524606
The final range between a & b :  0.2919207526134241
```

❑ Fibonacci Method

Recall that the golden section method uses the same value of ρ *throughout*. Suppose now that we are allowed to vary the value ρ from state to state, so that at the kth stage in the reduction process we use a value ρ_k, at the next state we use $\rho_{k+1,}$ and so on.

As in the golden section search, our goal is to select successive values of $\rho_k, 0 \leq \rho_k \leq 1/2$, such that only one new function evaluation is required at each state. To derive the strategy for selecting evaluation points, consider fig.



Selecting Evaluation Points

From this figure we see that it is sufficient to choose the ρ_k such that :

$$\rho_{k+1}(1-\rho_k) = 1-2\rho_k$$

After some manipulations,we obtain:

$$\rho_{k+1} = 1 - \frac{\rho_k}{1-\rho_k}$$

There are many sequences $\rho_1, \rho_2, \ldots$ that satisfy the law of formation above and the condition that $0 \leq \rho_k \leq 1/2$. For example, the sequence $\rho_1 = \rho_2 = \rho_3 = \ldots = (3-\sqrt{5})/2$ satisfies the conditions above and gives rise to the golden section method.

Suppose that we are given a sequence $\rho_1, \rho_2, \ldots$ satisfying the conditions above and we use this sequence in our search algorithm. Then, after N iterations of the algorithm, the uncertainty range is reduced by a factor of

$$(1-\rho_1)(1-\rho_2)\ldots(1-\rho_N).$$

Depending on the sequence $\rho_1, \rho_2, \ldots,$ we get a different reduction factor. The natural question is as follows: What sequence $\rho_1, \rho_2, \ldots$ minimizes the reduction factor above?This problem is a constrained optimization problem that can be stated formally as

$$\begin{gathered} \textit{minimize } (1-\rho_1)(1-\rho_2)\ldots(1-\rho_N) \\ \textit{subject to } \ \rho_{k+1} = 1 - \frac{\rho_k}{1-\rho_k}, k = 1, \ldots, N-1 \\ 0 \le \rho_k \le \frac{1}{2}, k = 1, \ldots, N \end{gathered}$$

Before, we give the solution to the optimization problem above, we need to introduce the *Fibonacci sequence* $F_1, F_2, F_3, \ldots$. This sequence is defined as follows. First, let F_{-1}=0 and $F_0 = 1$ by convention. Then, for $k \ge 0$,

$$F_{k+1} = F_k + F_{k-1}$$

Some values of elements in the Fibonacci sequence are :

F_1	F_2	F_3	F_4	F_5	F_6
1	2	3	5	8	13

It turns out that the solution to the optimization problem above is :

$$\begin{aligned} \rho_1 &= 1 - \frac{F_N}{F_{N+1}}, \\ \rho_2 &= 1 - \frac{F_{N-1}}{F_N}, \\ &\vdots \\ \rho_k &= 1 - \frac{F_{N-k+1}}{F_{N-k+2}}, \\ &\vdots \\ \rho_N &= 1 - \frac{F_1}{F_2}, \end{aligned}$$

where the F_k are the elements of the *Fibonacci* sequence. The resulting algorithm is called the *Fibonacci search method*.

In the Fibonacci search method, the uncertainty range is reduced by the factor

$$(1-\rho_1)(1-\rho_2)\dots(1-\rho_N) = \frac{F_N}{F_{N+1}}\frac{F_{N-1}}{F_N}\dots\frac{F_1}{F_2} = \frac{F_1}{F_{N+1}} = \frac{1}{F_{N+1}}$$

Because the Fibonacci method uses the optimal values of $\rho_1, \rho_2, \dots \rho_N,$ the reduction factor above is less than of the Golden section method. In other words, the Fibonacci is better than the golden search method in that it gives a smaller final uncertainty range.

We point out that there is an anomaly in the final iteration of the Fibonacci search method, because,

$$\rho_N = 1 - \frac{F_1}{F_2} = \frac{1}{2}$$

Recall that we need two intermediate points at each stage, one that comes from a previous iteration and another that is a new evaluation point .However, with $\rho_N = \frac{1}{2},$ the two intermediate points coincide in the middle of the uncertainty interval, and therefore we cannot further reduce the uncertainty range. To get around this problem, we perform the new evaluation for the last iteration using $\rho_N = 1/2 - \epsilon,$ where ϵ is a small number. In other words the new evaluation point is just left or right of the midpoint of the uncertainty interval.

As a result of the modification above, the reduction in the uncertainty range at the last iteration may be either

$$1 - \rho_N = \frac{1}{2}$$
$$or\ ,\ 1 - (\rho_N - \epsilon) = \frac{1}{2} + \epsilon = \frac{1+2\epsilon}{2},$$

depending on which of the two points has the smaller objective function value. Therefore, in the worst case, the reduction factor in the uncertainty range for the Fibonacci method is:

$$\frac{1+2\epsilon}{F_{N+1}}$$

Example : Consider the function

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

Suppose that we wish to use Fibonacci search method to find the value of x that minimizes f over the range $[0,2],$ and locate this value of x to within the range 0.3

After N steps the range is reduced by $(1+2\epsilon)/F_{N+1}$ is the worst case. We need to choose N such that

$$\frac{1+2\epsilon}{F_{N+1}} \leq \frac{final\ range}{initial\ range} = \frac{0.3}{2} = 0.15$$

Thus, we need

$$F_{N+1} \geq \frac{1+2\epsilon}{0.15}$$

$$If\ we\ choose\ \epsilon \leq 0.1,\ then\ N = 4\ will\ do.$$

$Iteration: 1$ We start with

$$1-\rho_1 = \frac{F_4}{F_5} = \frac{5}{8}$$

We then compute

$$a_1 = a_0 + \rho_1(b_0 - a_0) = \frac{3}{4} = 0.75$$

$$b_1 = a_0 + (1-\rho_1)(b_0 - a_0) = \frac{5}{4} = 1.25$$

$$f(a_1) = 0.75^4 - 14*0.75^3 + 60*0.75^2 - 70*0.75 = -24.33$$

$$f(b_1) = 1.25^4 - 14*1.25^3 + 60*1.25^2 - 70*1.25 = -18.65$$

$f(a_1) < f(b_1)$.

The range is reduced to

$$[a_0, b_1] = [0, 1.25]$$

$Iteration: 2$ We have

$$1-\rho_2 = \frac{F_3}{F_4} = \frac{3}{5}$$

We then compute

$$a_2 = a_0 + \rho_2(b_1 - a_0) = \frac{2}{5}*\frac{5}{4} = \frac{1}{2} = 0.5, \quad b_2 = a_1 = 0.75$$

$$f(a_2) = 0.5^4 - 14*0.5^3 + 60*0.5^2 - 70*0.5 = -21.68$$

$$f(b_2) = -24.33$$

$f(b_2) < f(a_2)$.

The range is reduced to

$$[a_2, b_1] = [0.5, 1.25]$$

Iteration : 3 We have

$$1-\rho_3 = \frac{F_2}{F_3} = \frac{2}{3}$$

We then compute

$$b_3 = a_2 + (1-\rho_3)(b_1 - a_2) = \frac{1}{2} + \frac{2}{3} * \frac{3}{4} = \frac{1}{2} + \frac{1}{2} = 1, \quad a_3 = b_2 = a_1 = 0.75$$

$$f(b_3) = 1^4 - 14*1^3 + 60*1^2 - 70*1 = -23$$
$$f(a_3) = f(b_2) = -24.33$$

$f(a_3) < f(b_3)$.
The range is reduced to

$$[a_2, b_3] = [0.5, 1]$$

Iteration : 4 We choose $\epsilon = 0.05$. We have

$$1-\rho_4 = \frac{F_1}{F_2} = \frac{1}{2}$$

We then compute

$$a_4 = a_2 + (\rho_4 - \epsilon)(b_3 - a_2) = \frac{1}{2} + 0.45 * \frac{1}{2} = 0.725,$$
$$b_4 = a_3 = b_2 = a_1 = 0.75$$

$$f(a_4) = 0.725^4 - 14*0.725^3 + 60*0.725^2 - 70*0.725 = -24.27$$
$$f(b_4) = f(a_3) = f(b_2) = -24.33$$

$f(a_4) > f(b_4)$.
The range is reduced to

$$[a_4, b_3] = [0.725, 1]$$

Note that $b_3 - a_4 < 0.3$.

Link to the Colab Notebook : [Fibonacci search method]

```
# Importing required python libraries
import math
import numpy as np

#function definition
```

```
def fun(x):
  val = x**4-14*x**3+60*x**2-70*x
  return val

#Fibonacci Function
def fibonacci(n):
  f0=0
  f1=1
  #Initialize an empty array
  arr = [0]*n
  arr[0]= 0
  arr[1]= 1
  for i in range(2,n):
    arr[i]=arr[i-1]+arr[i-2]
  return arr[1:]

# Fibonacci search Method
def fibonacci_search(a,b,eps):

    ''' The number of iterations needed to reach the given range
between two values,i.e|a-b|< epsilon
         (1+2*epsilon)^/F_(n+1) <= 0.3(final range)/2(given
initial range)'''

    N = math.ceil((1+2*eps)/(0.3/(b-a)))

    arr = fibonacci(N+1)
    #Comparing the value of N, in our fibonacci sequence

    for i in range(0,len(arr)):
      if arr[i] >= N:
        N=i-1  #because F_(N+1)>= N, so value of iteration N,
will be less than N+1.
        break

    print("We need :",N,"iteration to reach within the given
range")
```

```
    ''' We will use iteration to arrive at the required range.
      Since, for the first time we need to calculate 2 evaluation
points, we will keep the first iteration
      out of the iteration look'''
    #Iteration : 1
    ro = 1 - (arr[N]/arr[N+1])
    a1 = a + ro*(b-a)
    b1 = a + (1-ro)*(b-a)
    f1= fun(a1)
    f2 = fun(b1)

    for i in range(1,N+2):
      if abs(a-b) <= 0.3:
        print("The final range between a & b : ",abs(b-a),"\t")
        break
      f1= fun(a1)
      f2= fun(b1)
      ro = 1 - (arr[N-i]/arr[N+1-i]) # recalculating the value of
rho, for every interval
      if ro == 0.5: #special case for the last iteration being
handled in this if clause.
        if f1 <= f2:
          b = b1
          b1 = a1
          print("a:",a,"b:",b,"\t")
          print("Iteration :",i,"f1: ",f1," f2: ",f2,"\t")
          a1 = a + (ro-eps)*(b-a)
        else:
          a = a1
          a1 = b1
          print("a:",a,"b:",b,"\t")
          print("Iteration :",i,"f1: ",f1," f2: ",f2,"\t")
          b1 =  a + (ro+eps)*(b-a)
      else:  # for rest of the iterations before the last
iteration, the following steps are executed.
        if f1 <= f2:
          print("a:",a,"b:",b,"\t")
          print("Iteration :",i,"f1: ",f1," f2: ",f2,"\t")
```

```
          b=b1
          b1=a1
          a1 = a + ro*(b-a)
        else:
          print("a:",a,"b:",b,"\t")
          print("Iteration :",i,"f1: ",f1," f2: ",f2,"\t")
          a = a1
          a1 = b1
          b1 =  a + (1-ro)*(b-a)

#Function call
a=0
b=2
fibonacci_search(a,b,eps=0.05)

#Output
We need : 4 iteration to reach within the given range
a: 0 b: 2
Iteration : 1 f1:  -24.33984375  f2:  -18.65234375
a: 0 b: 1.25
Iteration : 2 f1:  -21.6875  f2:  -24.33984375
a: 0.5 b: 1.0
Iteration : 3 f1:  -24.33984375  f2:  -23.0
a: 0.5 b: 1.0
Iteration : 4 f1:  -24.271312109374996  f2:  -24.33984375
The final range between a & b :  0.275
```

❏ Bisection Method

Again we consider finding the minimizer of an objective function $f : R \to R$ over an interval $[a_0, b_0]$. As before, we assume that the objective function f is unimodal. Further, suppose that f is continuously differentiable and that we can use values of the derivative f' as a basis for reducing the uncertainty interval.

The *bisection method* is a simple algorithm for successively reducing the uncertainty interval based on evaluations of the derivative. To begin, let $x^{(0)} = (a_0 + b_0)/2$ be the midpoint of the initial uncertainty interval. Next, evaluate $f'\left(x^{(0)}\right)$. If $f'\left(x^{(0)}\right) > 0$, then we deduce that the minimizer lies to the left of $x^{(0)}$. In other words, we reduce the uncertainty interval to

$\left[a_0, x^{(0)}\right]$. On the other hand, if $f'\left(x^{(0)}\right) < 0$, then we deduce that the minimizer lies to the right of $x^{(0)}$. In this case, we reduce the uncertainty interval to $\left[x^{(0)}, b_0\right]$. Finally, if $f'\left(x^{(0)}\right) = 0$, then we conclude $x^{(0)}$ to be the minimizer and terminate our search.

With the new uncertainty interval computed, we repeat the process iteratively. At each iteration k, we compute the midpoint the midpoint of the uncertainty interval. Call this point $x^{(k)}$. Depending on the sign of $f'\left(x^{(k)}\right)$ $(assuming\ that\ it\ is\ nonzero)$, we reduce the uncertainty interval to the left or right of $x^{(k)}$. If at any iteration k we find that $f'\left(x^{(k)}\right) = 0$, then we declare $x^{(k)}$ to be the minimizer and terminate our search.

Two salient features distinguish the bisection method from the golden section and Fibonacci methods. First, instead of using value of f, the bisection methods uses values of f'. Second, at each iteration, the length of the uncertainty interval is reduced by a factor of 1/2. Hence, after N steps, the range is reduced by a factor of $(1/2)^N$. This factor is smaller than in the golden section and Fibonacci Method.

```
# Importing required python libraries
import math
import numpy as np

#defining the derivative of the given function

def derivtive_fun(x):
  val = 4*x**3-42*x**2+120*x-70
  return val

# Bisection Method

def bisection_method(a,b):

    ''' The number of iterations needed can be calculated by
solving (0.5)^N <= 0.3/2'''

    N = math.ceil(np.log10(0.3/(b-a))/np.log10(0.5))

    print("We need :",N,"iteration to reach within the given
range")

    x = (a+b)/2
```

```
    f = derivtive_fun(x)

    i=0

    while f!=0:

      if abs(a-b)<=0.3:

        print("The range between a & b : ",abs(b-a),"\t")

        break

      i=i+1

      x = (a+b)/2

      f = derivtive_fun(x)

      print("Iteration:",i,"a:",a,"b:",b,"x:",x,"f:",f,"\t")

      if f > 0:

        b = x

      elif f < 0:

        a = x

#Bisection Method Function call
a=0
b=2
bisection_method(a,b)

#Output

We need : 3 iteration to reach within the given range
Iteration: 1 a: 0 b: 2 x: 1.0 f: 12.0
```

```
Iteration: 2 a: 0 b: 1.0 x: 0.5 f: -20.0
Iteration: 3 a: 0.5 b: 1.0 x: 0.75 f: -1.9375
The range between a & b :  0.25
```