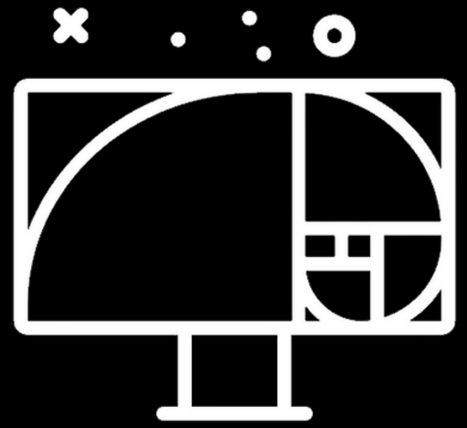
DECEMBER
2023



FIBONACCI SEARCH

Code & Explanation

Prepared by T. Mohamed Nadhim (3122 21 3002 058) as a partial fulfilment for the requirement of the course
UIT2522 - Optimization Techniques for Machine Learning (Honors)

The given equation:

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

Range: [0, 2]

No. of Iterations (N): 4

Programming approach:

In order to implement the Fibonacci Search Algorithm, we need the following data:

- A list containing Fibonacci Series
- A list containing all “ρ” values (Eg: ρ1, ρ2……. upto ρN)

Programming approach:

1. Let us create a list containing ‘N’ Fibonacci Numbers

```
#creating a list with N+1 fibonacci numbers
def fibonacci(n):
    first = 1
    second =1
    fib = [1,1]
    if n<=2:
        return fib

    for i in range(n):
        third = first+second
        fib.append(third)
        first,second = second,third

    return fib
```

Programming approach:

The function “fibonacci(n)” returns a list containing “N” fibonacci numbers

```
>>> fibonacci(10)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
>>>
```

Programming approach:

2. Let us create a list containing all “N” values of “ρ”

The general Formula of “ρ” is given by:

$$\rho_k = 1 - \frac{F_{N-k+1}}{F_{N-k+2}}$$

Eg:

$$\rho_1 = 1 - \frac{F_N}{F_{N+1}}$$

$$\rho_2 = 1 - \frac{F_{N-1}}{F_N}$$

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

Programming approach:

Let us create a function “p_list()” which generates a list containing “ρ” values using the general formula :

```
#Creating a list containing p1,p2,....pN
def p_list(n):

    p=[0]

    for k in range(1,n+1):
        p_temp = 1 - (fib[n-k+1]/fib[n-k+2])
        p.append(p_temp)

    return p
```

Note:
The variable “fib” is the list we created that contains “N” fibonacci numbers

Programming approach:

3. Let us create a function which returns a value after substituting the variable in the given equation

```
#Getting the value of the equation
def eqn(x):
    equation = (x**4) - (14*(x**3)) + (60*(x**2)) - (70*x)
    return equation
```

Programming approach:

After obtaining the values, we have to write a looping statement to calculate the iterations.

The equation of the “start (a1)” and “end (b1)” of the range during the 1st Iteration is given by:

$$a_1 = a_0 + \rho_1(b_0 - a_0)$$

$$b_1 = a_0 + (1 - \rho_1)(b_0 - a_0)$$

Let us implement the generalized formulas inside a loop in a function called “iterations()”

1. Let’s create two temporary variables “start1” and “end1” to store the value of the current “start” and “end” range:

```
#Calculation for N iterations
def iterations(n):
    global start,end
    start1 = start
    end1 = end

    for i in range(1,n+1):
        start1 = start + (p[i]*(end-start))
        end1 = start + ((1-p[i])*(end-start))

        f_start = eqn(start1)
        f_end = eqn(end1)

        if f_start<f_end:
            end = end1
        else:
            start = start1
        print(f"Iteration {i}:",[start,end])
```

2. Now let us calculate the new “start” and “end” values by substituting the corresponding values and substitute in variables “start1” and “end1”.

```
#Calculation for N iterations
def iterations(n):
    global start,end
    start1 = start
    end1 = end

    for i in range(1,n+1):
        start1 = start + (p[i]*(end-start))
        end1 = start + ((1-p[i])*(end-start))

        f_start = eqn(start1)
        f_end = eqn(end1)

        if f_start<f_end:
            end = end1
        else:
            start = start1
        print(f"Iteration {i}:",[start,end])
```

3. Now let us substitute “start1” and “end1” in the function “eqn()” to find the values of “f(a1)” and “f(b1)” and store them in variables “f_start” and “f_end”

```
#Calculation for N iterations
def iterations(n):
    global start,end
    start1 = start
    end1 = end

    for i in range(1,n+1):
        start1 = start + (p[i]*(end-start))
        end1 = start + ((1-p[i])*(end-start))

        f_start = eqn(start1)
        f_end = eqn(end1)

        if f_start<f_end:
            end = end1
        else:
            start = start1
        print(f"Iteration {i}:",[start,end])
```

4. if the value of f(a1)<f(b1), then the upper limit of the range is changed to the new value “b1" while the lower limit remains the same

$$f(a_1) = -24.34$$
$$f(b_1) = -18.65$$
$$f(a_1) < f(b_1)$$

The new range:

$$[a_0, b_1] = [0, \tfrac{5}{4}]$$

```
#Calculation for N iterations
def iterations(n):
    global start,end
    start1 = start
    end1 = end

    for i in range(1,n+1):
        start1 = start + (p[i]*(end-start))
        end1 = start + ((1-p[i])*(end-start))

        f_start = eqn(start1)
        f_end = eqn(end1)

        if f_start<f_end:
            end = end1
        else:
            start = start1
        print(f"Iteration {i}:",[start,end])
```

5. Similarly, if the value of f(a1)<f(b1), then the lower limit of the range is changed to the new value while the upper limit remains the same. Eg: during 2nd Iteration

$$f(a_2) = -21.69$$
$$f(b_2) = f(a_1) = -24.34$$
$$f(a_2) > f(b_2)$$

The new range:

$$[a_2, b_1] = [\frac{1}{2}, \frac{5}{4}]$$

```python
#Calculation for N iterations
def iterations(n):
    global start,end
    start1 = start
    end1 = end

    for i in range(1,n+1):
        start1 = start + (p[i]*(end-start))
        end1 = start + ((1-p[i])*(end-start))

        f_start = eqn(start1)
        f_end = eqn(end1)

        if f_start<f_end:
            end = end1
        else:
            start = start1
        print(f"Iteration {i}:",[start,end])
```

6. The same has been implemented in the snippet of code as well

```
#Calculation for N iterations
def iterations(n):
    global start,end
    start1 = start
    end1 = end

    for i in range(1,n+1):
        start1 = start + (p[i]*(end-start))
        end1 = start + ((1-p[i])*(end-start))

        f_start = eqn(start1)
        f_end = eqn(end1)

        if f_start<f_end:
            end = end1
        else:
            start = start1
        print(f"Iteration {i}:",[start,end])
```

Programming approach:

Now, let us obtain the inputs and call all the functions in chronological order

1. Obtaining Inputs:

```
#The equation in use
print("x^4 - 14x^3 + 60x^2 - 70x")


#No. of Iterations
n = int(input("Enter the no. of iterations: "))

#Defining the range [0,2]
start = float(input("Enter the range start: "))
end = float(input("Enter range end: "))
```

Programming approach:

2. Calling the functions in chronological order:

```
fib = fibonacci(n)
p = p_list(n)

iterations(n)
```

Programming approach:

After writing all the code, save and run your program to implement Fibonacci Search for “N” no. of iterations for the given equation

Click on the following link to access the python file:

fibonacciSearch.py

THANK YOU