An Optimization Engine

Documentation

Ankit Mehra
IIT-Mandi

Libraries used in the project

1. **Math** :- To perform simple mathematical operations. (import math)
2. **Matplotlib** :- To plot graphs of various equations. (import matplotlib.pyplot)
3. **Numpy** :- To perform relatively complex mathematical problems. (import numpy)
4. **Sympy** :- To represent equations in the format as we write in the normal text. (import sympy)
5. **Time** :- To give time delay between graphs of multiple iterations. (import time)
6. **IPython** :- To represent multiple graph of multiple iterations in an animation on the same graph.(from IPython.display import clear_output)

Project Architecture

Step 1:- Creating a Parsing method that takes equation in string format and converts it into numpy readable format.
Step 2:- Creating Functions for various Numerical methods that we will use for our operations.

- Bisection Method
- Newton Raphson method
- Derivative function (will be used in Newton Raphson)
- Golden section method.
- Plotting method will be used for plotting various graphs.
- Functions to assist the main functions (maxima , minima , position check)

Step 3:- Creating an interface in which a user can give various inputs to perform operation of his/her choice.

Parsing of an equation

The input that we take from the user is not in string format as can not be read by the computer as it is , so we need to change that expression into numpy readable expressions.

I have taken ***Polynomial , Sinusoidal, Exponential and Logarithmic univariate functions*** only.

```
Format of Equation
 For Polynomial - a*x^n+b*x^(n-1)+c
 For trignometric - sin(a*x^2)+cos(b*x^4)
 For logarithmic - ln(ax)-ln(b*x^2)
 For Exponential - e^(a*x^2)-e^(b*x)
 where 'a','b','c' are constants and n, n-1 are powers
 NOTE-Use proper brackets for the priority
```

1.Input taken from user in this format .

```
equation = input("Enter the Equation for evaluation:")
xlimit1 = float(input()) #Minimum value of x
xlimit2 = float(input()) #Maximum value of x
x= smp.symbols('x ', real=True)
y=smp.sympify(equation)
```

2.User will provide the input as a string.

```
e = 2.71828
equation = equation.replace('^' , '**')
equation = equation.replace('sin(' , 'np.sin(')
equation = equation.replace('cos(' , 'np.cos(')
equation = equation.replace('ln(' , 'np.log(')
equation = equation.replace('exp(' , 'np.exp(')
equation = equation.replace('e^(' , 'np.exp(')
```

3.Equation will be changed to understandable format

```
def func(x):
    x = x
    return eval(equation)
```

4.Finally equation will be solved using **EVAL** function

Numerical Methods to find Roots

1:-Bisection Method

The Bisection method is one of the simplest and most reliable of iterative methods for the solutions of nonlinear equations. This method, also known as binary chopping or half interval method, relies on the fact that if f(x) is real and continuous in the interval $a<x<b$, and f(a) and f(b) are of opposite signs, that is,

$$\mathbf{f(a)\, f(b)<0}$$

Then there is at least one real root between a and b. There may be more than one root in the interval. Let, $x1=$ a and $x2=$ b. Let us also define another point $x0$ to be the middle point between a and b, that is,

$$\boldsymbol{x0= (x1+x2)/2}$$

Now, there exists the following three conditions:

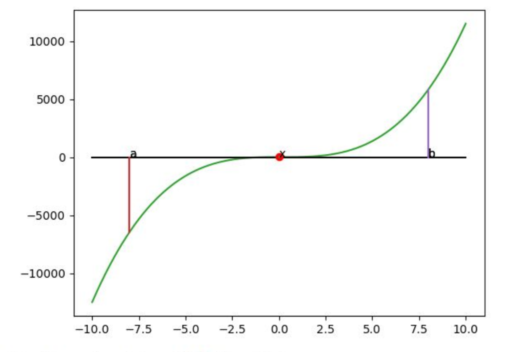
1. If $f(x0)=0$, we have a root at $x0$.
2. If $f(x0)\, f(x1)<0$, then there is a root between $x0$ and $x1$.
3. If $f(x0)\, f(x2)<0$, then there is a root between $x0$ and $x2$.

Reference-https://www.cbpbu.ac.in/userfiles/file/2020/STUDY_MAT/PHYSICS/NP%202.pdf

Bisection Method

```python
def bisect(a, b):     #Bisection method algorithm
    if (func(a) * func(b) >= 0):
        print("You have not assumed right a and b\n")
        return

    c = a
    n=0
    while ((b - a) >= 0.01):
        n+=1
        # Find middle point
        c = (a + b) / 2

        # Check if middle point is root
        if (func(c) == 0.0):
            break

        # Decide the side to repeat the steps
        if (func(c) * func(a) < 0):
            b = c
        else:
            a = c

    print("The value of root is : ", "%.4f" % c)
```

1.Code for Bisection Method



2.Graph plotted for 2nd iteration in matplotlib for "$12x3—5x2+6$"

```python
def bisection():
    for i in range(1000):
        a = float(input("First approximation root:"))
        b = float(input("Second approximation root:"))
        n=int(input("Number of iteration"))
        if func(a)*func(b) < 0:
            bisect(a , b )
            break
        else:
            print("Given approximate value do not bracket the root")
            print("Try again with different values")
    plt.plot([a,a],[0,func(a)])
    plt.annotate('a' , xy=(a-0.01,-0.2))

    plt.plot([b,b],[0,func(b)])
    plt.annotate('b' , xy=(b-0.01,-0.2))

    x_itterated = []
    for i in range(1 , n+1):
        plt.plot(x_axis , y_axis, [xlimit1 , xlimit2] , [0 , 0] , 'k')
        plt.plot([a,a],[0,func(a)])
        plt.annotate('a' , xy=(a-0.01,-0.2))

        plt.plot([b,b],[0,func(b)])
        plt.annotate('b' , xy=(b-0.01,-0.2))
        x = (a + b)/2
        plt.plot(x,func(x),'ro',label='x')
        plt.plot([x,x],[0,func(x)],'k')
        print("Iteration number:" , i , "|" "x=" , x , "|" ,  "f(x)=" , func(x))
        time.sleep(1)
        x_itterated.append(x)
        clear_output(wait=True)
        plt.plot([x,x] , [0,func(x)])
        plt.annotate('x' , xy = (x-0.01,-0.2))
        plt.show()

        if func(x) == 0:
            print("The approximate root=" , x)
            break
        elif func(a)*func(x) > 0:
            a = x
        else:
            b = x
        time.sleep(1)
```

Taking user input for 1st and 2nd approximations, asking for no. of iteration.

Plotting the graph and using Ipython.display to show multiple cases as the graph converges.

2.Newton Raphson Method

The Newton-Raphson method, or Newton Method, is a powerful technique for solving equations numerically. Like so much of the differential calculus, it is based on the simple idea of linear approximation. The Newton Method, properly used, usually homes in on a root with devastating efficiency.

2.1- The Newton Raphson Iteration

Let x0 be a good estimate of r and let $r = x0 + h$. Since the true root is r, and $h = r - x0$, the number h measures how far the estimate x0 is from the truth.

Since h is ‘small,’ we can use the linear (tangent line) approximation to conclude that

$$\mathbf{0 = f(r) = f(x0 + h) \approx f(x0) + hf0\ (x0),}$$

and therefore, unless f0 (x0) is close to 0,

$$\mathbf{h \approx -\ f(x0)\ /f'(x0)}$$

It follows that,

$$\mathbf{r = x0 + h \approx x0 - \{f(x0)/f'(x0)\}}$$

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$$

Our new improved estimate x1 of r is therefore given by,

$\mathbf{x1 = x0 - \{f(x0)/f'(x0)\}}$ and the next iteration is $\mathbf{x2 = x1 - \{f(x1)/f'(x1)\}}$.

Newton Raphson iterative method.

Centred difference is used to calculate differentiation as it has min error(2nd order).

f'(x) ≈ {f(x + h) — f(x — h)}/2h

```
def derivFunc(x):
    d=0.0001
    num=func(x+d)-func(x-d)
    den=2*d
    der=num/den
    return der
```

The function below is used in the code till the threshold error value (here 0.0001) is reached.

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$$

```
def newtonRaphson( x ):
    h = func(x) / derivFunc(x)
    while abs(h) >= 0.0001:
        h = func(x)/derivFunc(x)
        
        # x(i+1) = x(i) - f(x) / f'(x)
        x = x - h
    
    print("The value of the root is : ",
                            "%.4f"% x)

# Driver program to test above
x0 = int(input("Enter Initial value")) # Initial guess asked from user
newtonRaphson(x0)
```

Numerical method to find maximum value

Golden Section Method

Golden Section method is a Optimization technique which is used for finding the extremum(Maximum or Minimum)of a given function.

It tries to find the extremum by the narrowing the searching interval. In each iteration the interval is updated to a new interval by the given process:

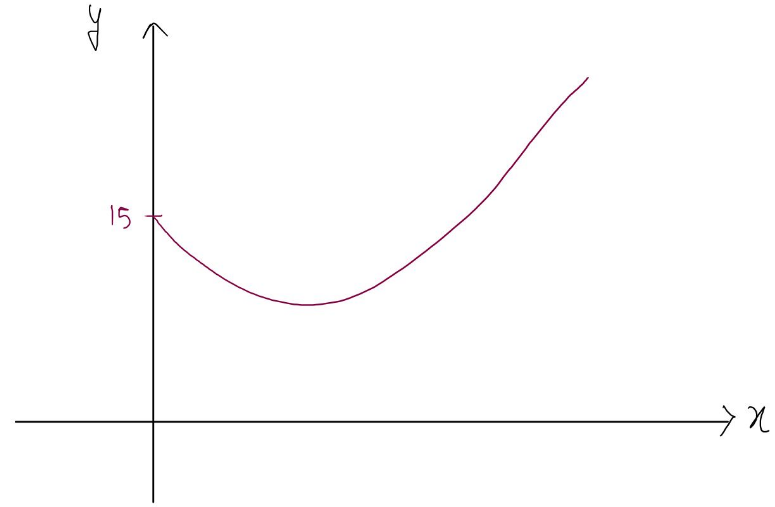
1. First we find the distance ‘d’ by the formula:

$$d=(xu-xl)/GR$$

2. Where xl and xu are the Initial upper and lower limits respectively and GR is Golden Ratio which is equal to 1.618 approx.
3. Now x1 and x2 are found from points xl and xu respectively where x1 = xl+d and x2 = xu-d
4. Those four variables(xl,xu,x1,x2) will be updated in each iteration and we again find the new x1 and x2 with respect to new xl and xu.

For example: let us take the function $f(x) = x^2 - 6x + 15$

Let us take xl as 0 and xu as 10



Now d = 6.18 , x1 = 6.18 and x2 = 3.82.

f(x1) = 16.1124 and f(x2) = 6.6724

Here x1 > x2 and f(x1) > f(x2) so we can discard [x1 , xu]

Finally new interval is [xl , x1] means [0 , 6.18]

Our new xl is 0 and new xu is x1

Again following the same process d = 0.618(6.18) = 3.81924 , x1 = 3.81924 and x2 = 2.36076

f(x1) = 6.67115 and f(x2) = 6.40863

Here x1 > x2 and f(x1) > f(x2) so we can discard [x1 , xu] means [3.81924 , 6.18]

Finally new interval is [xl , x1] means [0 , 3.81924]

```python
def update_interior(xl,xu):   #upd
    d=((np.sqrt(5)-1)/2)*(xu-xl)
    x1=xl+d
    x2=xu-d
    return x1,x2
```

Step 1-Update interior
Updates the value of x1 and x2
After each iteration

```python
def check_pos(x1,x2): #checking the position
    if x2<x1:
        label='right'
    else:
        label=''
    return label
```

Step 2- Check Position,checks in which direction will the x1 and x2 move.

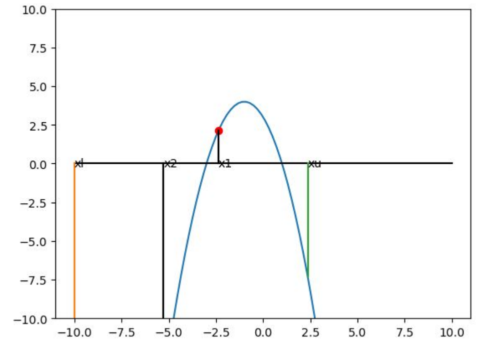
```python
def find_max(xl,xu,x1,x2,label):
    fx1=func(x1)
    fx2=func(x2)
    if fx2>fx1 and label=='right':
        xl=xl
        xu=x1
        new_x=update_interior(xl,xu)
        x1=new_x[0]
        x2=new_x[1]
        xopt=x2
    else:
        xl=x2
        xu=xu
        new_x=update_interior(xl,xu)
        x1=new_x[0]
        x2=new_x[1]
        xopt=x1
    return xl,xu,xopt
```

Step 3- Finding Max value

```python
def golden_search(xl,xu,et):
    it=0
    e=1
    while e>=et:
        new_x=update_interior(xl,xu)
        x1=new_x[0]
        x2=new_x[1]
        fx1=func(x1)
        fx2=func(x2)
        label=check_pos(x1,x2)
        clear_output(wait=True)
        plot_graph(xl,xu,x1,x2) #PLOTTING
        plt.show()
        new_boundary=find_max(xl,xu,x1,x2,label)
        xl=new_boundary[0]
        xu=new_boundary[1]
        xopt=new_boundary[2]

        it+=1
        print ('Iteration: ',it)
        r=(np.sqrt(5)-1)/2 #GOLDEN RATIO
        e=((1-r)*(abs((xu-xl)/xopt)))*100 #Error
        print('Error:',e)
        time.sleep(1)
```

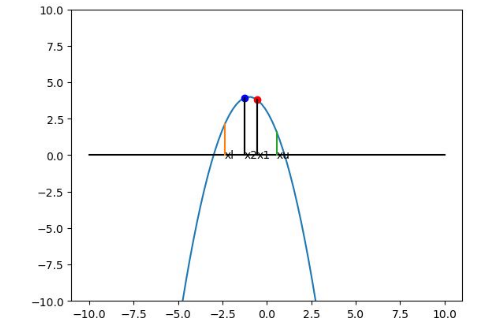
Step 4-Golden Section and plotting

<u>Iteration 2</u>



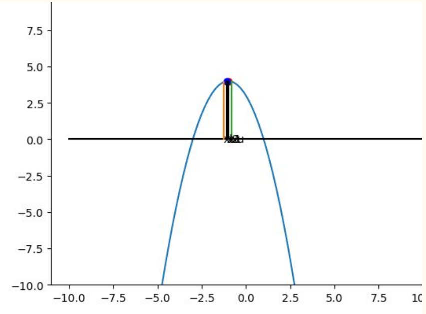
Iteration: 2
Error: 523.6067977499797

<u>Iteration 5</u>



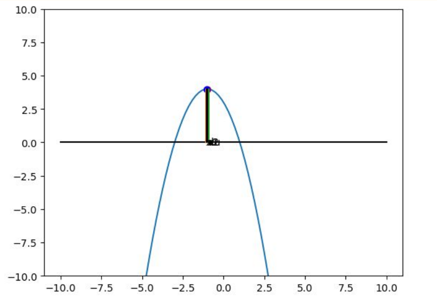
Iteration: 5
Error: 41.20226591665968

<u>Iteration 9</u>



Iteration: 9
Error: 10.913314607145871

<u>Iteration 11</u>



Iteration: 11
Error: 4.001703624175112