Numerical Python

Let a function f(x) . If for x=ξ, f(ξ)=0, we can say ξ is a root of the equation f(x)=0. If f(x) is quadratic , biquadratic or some other known form , we can solve it analytically. But if f(x)=0 is a polynomial of higher degree or the equation involves some transcendental or trigonometric function like e^x , ln(x), sin(x), tan(x) etc. such that no formula exists, then we can find roots through variuous approximate numerical methods. Within the help of computer programming we can efficiently solve by approximate methods.

1. **Bisection Methods:**

Let f(x) is a continuous function between x=a and x=b and f(a) and f(b) are of opposite signs. We can say that there exists a point ξ between a and b so that f(ξ)=0 ,then the condition for a root exists : **f(a) * f(b) < 0 .**

To find the root we bisect the interval (a,b) by x_m = (a+b)/2 . If f(x_m)=0 , x_m is the root . If f(x_m)≠0, then the root exists either in interval [a,x_m] or [x_m, b].

So, the condition is :

i. f(x_m)=0 , x_m is the root of the equation.

ii. f(a)* f(x_m) < 0 , thr root exists in the interval [a, x_m]

iii. f(x_m)*f(b) < 0 , thr root exists in the interval [x_m, b]

We repeatedly bisect the exact interval unless the exact root is found with desired accuracy.

Algorithm.

1. Define function f(x)
2. Define the interval a,b
3. Define accuracy limit tol.
4. If f(a)*f(b) > 0 , no root exists.
5. Else set x_m=(a+b)/2
6. If f(x_m) = 0 , x_m is the roo f(x_m)*t, exit.
7. If f(a)*f(x_m) < 0 , set b=x_m else a= x_m
8. Until |b-a |> tol, repeat step 2 onwards
9. Finally the root is (a+b)/2

Python Program :

```
def f(x):
    return x**3-2*x-5
a,b,tol=2.0,3.0,0.001
while f(a)*f(b) < 0:
    print('no root exist')
    break
while abs(b-a) >= tol :
    xm=(a+b)/2.0
    if f(xm)==0:
            print('Root=',xm)
            break
    if f(a)*f(xm)<0:
            b=xm
    else :
            a=xm
print('Root=',(a+b)/2.0)
```

2. Newton Raphson Method

This is an improved version of earlier method i.e. Bisection Method.

Let x_0 be an approximate root of f(x) =0.

We will expand the function in Taylor's series around the point x_0.

Let $x_1 = x_0 + h, \quad for\ f(x_1) = 0$

$(f(x_1) = f(x_0 + h) = 0$

$f(x_0 + h) = f(x_0) + h\, f'(x_0) + \frac{h^2}{2!} f''(x_0)$, neglecting 2nd and higher order terms we get,

$$f(x_0) + hf'(x_0) = 0$$

h = - $\frac{f(x_0)}{f'(x_0)}$, $x_{n+1} = x_n - \frac{f(x_0)}{f'(x_0)}$

In general , $x_{n+1} = x_n - \frac{f(x_0)}{f'(x_0)}$

Algorithm :

1. Define f(x)
2. Get x_0 and tol value
3. Get h = - $\frac{f(x_0)}{f'(x_0)}$ and update x until |f(x)| > tol

Python Script :

```
def (x):
    return x**3-2*x-5
def df(x):
    return 3*(x**2)-2
x,tol=3,0.001
while abs(f(x)) >=tol:
    x=x-f(x)/df(x)
    print(x)
print('Root=',x)
```

NB : in case f'(x) can not be obtained analytically , we follow the formula,

$$f'(x) = \frac{f(x+dx)-f(x-dx)}{2\,dx}$$

Python Script :

f(x)=cos(x)-x exp(x)

```
from math import cos, exp
def f(x):
    return cos(x)-x*exp(x)
x.tol,dx=1,0.001,0.01
while abs(f(x))>=tol:
    dfx=(f(x+dx)-f(x-dx))/(2*x)
    x=x-f(x)/dfx
    print(x)↑↑
print('Root=',x)
```

3. Interpolation :

Interpolation is a type of estimation where new data points are created within the given range of data points. We assume a curve which passes through each of data points.

Give data points : $(x_0, y_0)\ (x_1, y_1)\ (x_2, y_2)$, $(x_i = x_0 + ih)$, i=0,1,2,.....n. Our aim is to find f(x) where $x_{0 \le x \le} x_n$.

Lagrange Interpolation :

$$\emptyset(x) = \frac{(x-x_1)(x-x_2).....(x-x_n)}{(x_0-x_1)\,(x_0-x_2).\ .\ .\ .\ .(x_0-x_n)}\ y_0$$

$$+\quad \frac{(x-x_0)(x-x_2).....(x-x_n)}{(x_1-x_0)\,(x_1-x_2).\ .\ .\ .\ .(x_1-x_n)}\ y_1$$

$$+\ .\ .\ .\ .\ .\ .\ .\ .\ .\ .$$

$$+\frac{(x-x_0)(x-x_1).....(x-x_{n-1})}{(x_n-x_0)\,(x_n-x_1).\ .\ .\ .\ .(x_n-x_{n-1})}\ y_n$$

$$=\sum_0^n L_i\,(x)\,y_i$$

Where $L_i\,(x) = \prod_{j=0, j\neq i}^{n} \frac{(x-x_j)}{x_i-x_j}$

Example :

x	5	10	15	20	25	30
y	45	105	174	259	364	496

Python Script :

```
X=[5,10,15,20,25,30]
Y=[45,105,174,259,364,496]
Xt=18
n=len(x)
s=0
for I in range():
    L=0
    for j in range(n):
        if L != j:
            L=L*(xt-x[j])/(x[i]-x[j])
    S+=L*y[i]
Print('interpolated value=',s)
```

Comparison of interpolated data with original data

Let f(x)=x^2e^{-x}

We will generate data points with f(x). Then we will find intermediate data through through interpolation. Then plot both dada sets. (using Lagrange formula).

```
Import numpy as np
X=np.arange(0,10,0.5)
Y=(x**2)*(np.exp(-x))
```

```
Xt=np.arange(0.25,10,0.5)
def Lagrange(x,y,xt):
    N=x.size
    def L(i):
        u=np.prod(xt-x[:i])*np.prod(xt-x[i+1:])
        d= np.prod(x[i]-x[:i])*np.prod(x[i]-x[i+1:])
        return u/d
    return sum([)*y[i] for I in range(n)])
yt=[Lagrange(x,y,i) for I in xt]
import matplotlib.pyplot as plt
plt.plot (x,y,'o')
plt.plot(xt,yt,'+')
plt.show()
```

Interpolation by finite difference method :

Forward Difference ;

If $y_0, y_1, y_2, \ldots\ldots y_n$ are few y-values

$\Delta y_0 = y_1 - y_0$, $\Delta y_1 = y_2 - y_1$, $\ldots\ldots \Delta y_{n-1} = y_n - y_{n-1}$ are called 1st forward differences.

2nd forward differences:

$\Delta^2 y_0 = \Delta y_1 - \Delta y_0 = (y_2 - y_1) - (y_1 - y_0) = = (y_2 - 2\,y_1 + y_0)$

Simillarly $\Delta^2 y_1 = \Delta y_2 - \Delta y_1 = (y_3 - y_2) - (y_2 - y_1) = = (y_3 - 2\,y_2 + y_1)$

3rd forward differences:

$\Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0 = (y_3 - 3y_2 + 3y_1 - y_0)$

$\Delta^4 y_0 = (y_4 - 4y_3 + 6y_2 - 4y_1 + y_0)$

Forward difference Table :

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$
x_0	y_0	Δy_0	$\Delta^2 y_0$	$\Delta^3 y_0$	$\Delta^4 y_0$
x_1	y_1	Δy_1	$\Delta^2 y_1$	$\Delta^3 y_1$	
x_2	y_2	Δy_2	$\Delta^2 y_2$		
x_3	y_3	Δy_3			
x_4	y_4				

Note : for derivation .

Let E -→ a shift operator such that $Ey_k = y_{k+1}$ for any k

$E^2 y_k = E\,(Ey_k) = E\,y_{k+1} = y_{k+2}$ and $E^n y_k = y_{k+n}$

So, $\Delta y_0 = y_1 - y_0 = E\,y_0 - y_0 = (E-1)\,y_0$

$$\Delta = \boldsymbol{E - 1}$$

$\Delta^3 y_0 = (E-1)^3 = (E^3 - 3E^2 + 3E - 1)\,y_0 = y_3 - 3y_2 + 3y_1 - y_0$

Interpolation with forward differences

when the exact form of f(x) is unknown , the relaton is approximated by a simple function $\varphi(x)$, so that f(x) and $\varphi(x)$ agree at all given points (x_i, y_i).

If $\varphi(x)$ is considered a polynomial of some degree, it is called polynomial interpolation. Other kind of interpolation is trigonometric interpolation(int. function is a trigonometric function) etc.

Now let $\varphi(x)$ is a polynomial of degree n , which passes through all given points.

$$\varphi(x) = A_0 + A_1\,(x - x_0) + A_2\,(x - x_0)(x - x_1)$$
$$+A_3\,(x - x_0)(x - x_1)\,(x - x_2) + \ldots\ldots +$$
$$A_n\,(x - x_0)(x - x_1)\,(x - x_2)\ldots(x - x_{n-1}) \quad \ldots\ldots\ldots \quad (1)$$

We have to calculate constant A_i.

i). x=x_0, $(x) = y_0$, $A_0 = y_0$

ii) x=x_1 , $\varphi(x) = y_1$, $y_1 = A_0 + A_1\,(x_1 - x_0) = y_0 + A_1\,h$,

$$A_1 = \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{h}$$

iii) x=x_2 , $\varphi(x) = y_2$, $y_2 = A_0 + A_1\,(x_2 - x_0) = y_0 + A_1\,h + A_2\,(x_2 - x_0)\,(x_2 - x_1)$.

Put value of (A_0, A_1) and get (A_2)

$$(A_2) = \frac{(y_2 - 2y_1 + y_0)}{2h^2} = \frac{\Delta^2 y_0}{2!\,h^2}$$

Similarly $(A_3) = \dfrac{\Delta^3 y_0}{3!\,h^3}$

For any arbitrary x, x=x_0+ t h x=x_0+ t h

So, x - x_0 = t h , x-x_1 = x - x_0 - h

$\varphi(x)$

$$\varphi(x) = A_0 + A_1\,(x - x_0) + A_2\,(x - x_0)(x - x_1) + \ldots\ldots$$
$$+A_n\,(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

$$= y_0 + th\,\frac{\Delta y_0}{} + th(th - h)\frac{\Delta^2 y_0}{2!\,h^2} + \ldots\ldots$$

$$= y_0 + t\,\Delta y_0 + t(t - 1)\frac{\Delta^2 y_0}{2!} + \ldots\ldots$$

Previous example :

x	5	10	15	20	25	30
y	45	105	174	259	364	496

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$	$\Delta^5 y$
5	45	60	9	7	-3	6
10	105	69	16	4	3	
15	174	85	20	7		
20	259	105	27			
25	364	132				
30	496					

Let x=18, t=(18-5)/5 =2.6,

$\varphi(18) = 45 + 2.6$ X 60 + 2.6(2.6-1)/(2!) X 9 + ... = 222.83

Algorithm :

1. Input x,y
2. Calculate t
3. Loop starts
4. Compute differences of y-values
5. Compute sum
6. Update co-eff for next iteration

Python Script :

```
x=[5,10,15,20,25,30]
Y=[45,105,174,259,364,496]
H=5.0
n=len(y)
t=(18-x[0])/h
coeff=t
s=y[0]
k=1
for I in range(n,1,-1):
    y=[y[j+1]-y[j] for j in range(i-1)]
    s+=coeff*(t-k)/(k+1)
print('interpolated value=',s)
```

Newton Interpolation with forward difference (Graphical Demonstration) :

```
Import numpy as np
# input data
X=[0.,0.7,1.4,2.1,2.8,3.5,4.2,4.9,5.6]
Y=[0.,0.64,0.99,0.86,0.33,-0.35,-0.87,-0.98,-0.63
h=0.7
# function defined for interpolation
Def interpole(t,y)
        Coeff,k=t,1
        s=y[0]
        for  I in range(len(y),1,-1):
                y=np.diff(y)
                s+=coeff*y[0]
                coeff*=(t-k)/(k+1)
        return s
x_int=np.arange(0.3,5.7,0.7)
y_int==[interpole((i-x[0])/h,y) for i in x_int]
# plotting
Import matplotlib.pyplot as plt
Plt.subplot(121)
Plt.plot(x,y,'__o',c='k',ms=8)
Plt.plot(x_int,y_int,'_^',c='k',ms=8)
Plt.subplot(122)
Plt.plot(x+x_int,y+y_int,'*',c='k',ms=8)
Plt.show()
```

Error calclulation :

From Newton's Forward difference formula we got function approximate

function $\emptyset(x)$ which passes through all given data point (x_0, y_0), $(x_1, y_1), \ldots\ldots\ldots, (x_n, y_n)$. But they may not pass through other intermediate points. So, to determine the y- value at any intermediate x-values there may be an error. We can define the error at any point z where $[x_0 \leq z \leq x_n]$ is

$$\in (x) = f(x) - \emptyset(x)$$

$$= \frac{(x-x_0)(x-x_1)\ldots\ldots\ldots(x-x_n)}{(n+1)!} f^{(n+1)}(z)$$

$$= \frac{t(t-1)(t-2)\ldots\ldots\ldots(t-n)}{(n+1)!} f^{(n+1)}(z)$$

$f^{(n+1)}(z)$ is (n+1)th derivative of function f(x) at the point 'z' with $x_0 \leq z \leq x_n$.

End