***I. J. Mathematical Sciences and Computing*, 2023, 4, 20-28**
Published Online on December 8, 2023 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijmsc.2023.04.03

Modelling Taylor's Table Method for Numerical Differentiation in Python

Pankaj Dumka*
Department of Mechanical Engineering, Jaypee University of Engineering and Technology, Guna-473226, Madhya Pradesh, India
E-mail: p.dumka.ipec@gmail.com
ORCID ID: https://orcid.org/0000-0001-5799-6468
*Corresponding Author

Rishika Chauhan
Department of Electronics and Communication Engineering, Jaypee University of Engineering and Technology, Guna-473226, Madhya Pradesh, India
ORCID ID: https://orcid.org/0000-0001-8483-865X

Dhananjay R. Mishra
Department of Mechanical Engineering, Jaypee University of Engineering and Technology, Guna-473226, Madhya Pradesh, India
ORCID ID: https://orcid.org/0000-0002-5107-0012

Received: 02 July, 2023; Revised: 01 August, 2023; Accepted: 23 September, 2023; Published: 08 December, 2023

Abstract: In this article, an attempt has been made to explain and model the Taylor table method in Python. A step-by-step algorithm has been developed, and the methodology has been presented for programming. The developed TT_method () function has been tested with the help of four problems, and accurate results have been obtained. The developed function can handle any number of stencils and is capable of producing the results instantaneously. This will eliminate the task of hand calculations and the use can directly focus on the problem solving rather than working hours to descretize the problem.

Index Terms: Taylor's Table method; Applications in numerical differentiation; Simulations in Python programming

1. Introduction

Differentiation is a very important tool in science and engineering as in any scientific study the understanding of how the physical variables are changing with respect to space or time is very much required. In majority of the cases the formulas derived from the physics of the problem are easily differentiated analytically. Still many data sets and curves obtained from the experiments (or via observations of the natural world) and other empirical methods requires numerical differentiation techniques to be analyzed properly [1,2]. This is the reason why numerical differentiation becomes so important from the standpoint of ordinary differential equations (ODE's) and partial differential equations (PDE's) [3,4]. By saying numerical differentiation the concept of finite difference comes into picture [5–7].

In finite difference the derivative of a formula or a data set is obtained by comparing the value of output data (or $y(x)$) with those of inputs (x) at the point where the derivative is required and its adjacent points [3]. To understand it consider the Fig. 1. Now the slope of line joining points P and Q will approximately be equal to the tangent to the cure at the midpoint of x_i and x_{i+1}, this is nothing but the simplest form of the differentiation. Now to find the slope of the function at point P the spacing (h) between x_i and x_{i+1} has to reduce i.e. bring x_{i+1} close to x_i.

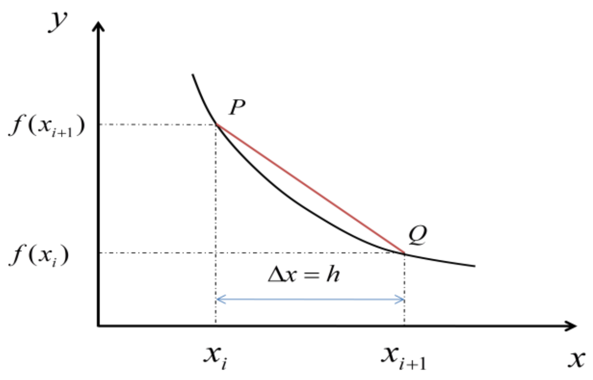


Fig. 1. Functions value at two adjacent locations

This is called as the geometrical approach to find the slope at a point. Analytically Taylor series expansion of $f(x)$ results in the finite difference approximation. The expansion of function at $x_i + h$ can be written as:

$$f(x_i + h) = f(x_i) + f'(x_i)h + f''(x_i)\frac{h^2}{2!} + f'''(x_i)\frac{h^3}{3!} + f''''(x_i)\frac{h^4}{4!} + \dots \tag{1}$$

If the first derivative $(f'(x_i))$ term is kept on one side and the remaining are kept on the other then the value of first derivative can be written as:

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h} - \frac{1}{h}\left[f''(x_i)\frac{h^2}{2!} + f'''(x_i)\frac{h^3}{3!} + f''''(x_i)\frac{h^4}{4!} + \dots\right] \tag{2}$$

Now, one can observe that the terms in the square bracket will be having highest term of order h (as $h < 1$). Therefore the approximate (numerical) value of the first order derivative of function at x_i is:

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h} \tag{3}$$

Now as all the remaining terms which are of $O(h)$ are truncated so this scheme is first order accurate. Moreover, as the value of $f'(x_i)$ is obtained by moving in the increasing direction of x (forward direction) this is called as forward difference [7]. Similarly if the Taylor series expansion $f(x_i - h)$ is done then one can get the $f'(x_i)$ by moving in the backward direction (Backward difference) as follows:

$$f(x_i - h) = f(x_i) - f'(x_i)h + f''(x_i)\frac{h^2}{2!} - f'''(x_i)\frac{h^3}{3!} + f''''(x_i)\frac{h^4}{4!} - \dots \tag{4}$$

$$f'(x_i) = \frac{f(x_i) - f(x_i - h)}{h} \tag{5}$$

This is also first order accurate i.e. have order of accuracy of $O(h)$. To obtain the expression for the derivative with higher order accuracy, say $O(h^2)$, one can add eq. 1 and 4 which will result in:

$$f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h} \tag{6}$$

To simplify the representation of above formulas the grid shown in Fig. 2 is very useful and this is what can be seen in many standard texts [6]. Here, $x_{i+1} = x_i + h$, $x_{i-1} = x_i - h$ and so on. Therefore, the f's can be written as $f(x_{i+1}) = f_{i+1}$, $f(x_{i-1}) = f_{i-1}$ and likewise others.

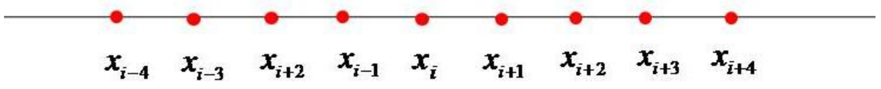


Fig. 2. Numerical Grid

As in the forward difference case for first order derivative, x_{i+1} and x_i are used, which are jointly called as stencil. Therefore the points which are taken to evaluate a derivative numerically is called as stencil. The choice of stencil will tell about the type of finite difference scheme used and its accuracy.

Now a pertinent question comes that how the finite difference approximation of higher order terms with different stencils can be obtained? The answer will again comes from Taylor series but, in tabulated form, which is also called as Taylor's Table method of obtaining finite difference approximations of derivatives [7]. But as the stencil size and the order of derivative increases the chances of calculation mistakes also increases. Also, sometimes it becomes very difficult to solve the problem in pen and paper.

Here's when it becomes crucial to use Python programming to automate the process of applying the Taylor table notion [8–10]. When it comes to using a programming language for scientific calculation, Python is highly beautiful and simple [11]. The fundamental strengths of Python are its vast community and extensive library. The NumPy [12], SciPy [13], SymPy [14,15], and Matplotlib are some of the modules which are competent of performing scientific, numeric, symbolic computations as well as data visualization and interpretation. With a very small number of lines of code, the modules are extremely capable of carrying out any mathematical computation, including those using algebra, trigonometry, calculus, set theory, etc. Despite being a computer tool, researchers are now automating research methods across all branches of science using Python.

In main objective of this article is to write a Python function to model the Taylor's Table. The procedure mentioned above is programmed using NumPy and the validity of the developed function is tested against few numerical problems. The developed function can handle any number of stencils and is capable of producing the results instantaneously. To best of our knowledge no such function exists therefore, the use of this function can reduce a substantive amount of time of the researchers.

2. Taylor's Table Method

In this case first we have to select the stencil i.e. the x locations which are to be incorporated in the difference formula. Then Taylor series expansion about each point is done. The terms in the Taylor series will be dictated by the order of derivative which has to be approximated by finite difference. If the order of the derivative is 2 (i.e. $\frac{d^2 f}{dx^2}$) then the series will be written till the second derivative (i.e. third term). The rows of the Taylors table will be the coefficients of the Taylor series for the respective x_i.

Let say $\frac{d^2 f}{dx^2}$ has to be approximated by considering x_{i-1}, x_i, and x_{i+1}. Then it can be written as:

$$\frac{d^2 f}{dx^2} = af_{i-1} + bf_i + cf_{i+1} \tag{7}$$

The purpose of Taylor table is to evaluate a, b, and c in the above equation. To make the task easy to understand the Taylor series has to be written in the general form as follows:

$$f(x_i + \alpha h) = f(x_i) + \alpha f'(x_i)h + \frac{\alpha^2}{2!} f''(x_i)h^2 + \frac{\alpha^3}{3!} f'''(x_i)h^3 + \frac{\alpha^4}{4!} f''''(x_i)h^4 + \dots \tag{8}$$

Now in the form shown in Eq. 7 the $f(x)$'s can be written as $f_{i+\alpha}$ (for more elaboration refer Table 1).

Table 1. f and α relation

α	$f_{i+\alpha}$
0	$f_{i+0} = f_i$
1	f_{i+1}
2	f_{i+2}
-1	f_{i-1}
-2	f_{i-2}

For the problem in hand the Taylor table is shown in Table 2.

Table 2. Taylor table

	$f'(x_i)$	$f'(x_i)h$	$f''(x_i)h^2$
f_{i-1}	1	−1	1/2
f_i	1	0	0
f_{i+1}	1	1	1/2

If the rows of the table are multiplied by a, b, and c and then added then Eq. 7 will be reproduced as:

$$0f(x)+0f'(x_i)+f''(x_i)=(a+b+c)f(x_i)+(-a+c)f'(x_i)h+\left(\frac{a}{2}+\frac{c}{2}\right)f''(x_i)h^2 \tag{9}$$

On comparing the terms of $f(x)$ and its derivatives on both the sides we will get:

$$a+b+c=0$$
$$-a+c=0$$
$$\left(\frac{a}{2}+\frac{c}{2}\right)=\frac{1}{h^2}$$

By applying Gaussian Elimination the values of a, b, and c will become:

$$a=\frac{1}{h^2},\ b=-\frac{2}{h^2},\ c=\frac{1}{h^2}$$

Therefore the finite difference approximation of $\frac{d^2f}{dx^2}$ considering f_{i-1}, f_i, and f_{i+1} as the stencil will become:

$$\frac{d^2f}{dx^2}=\frac{1}{h^2}f_{i-1}-\frac{2}{h^2}f_i+\frac{1}{h^2}f_{i+1}=\frac{1}{h^2}\left(f_{i-1}-2f_i+f_{i+1}\right) \tag{10}$$

Points to remember:

- The order of the derivative to be approximated should always be less than the number of terms in the stencil.
- The number of terms to be taken in the Taylor series should be equal to the number of terms in the stencil.

To model Taylor table method in Python one extra step is to be done, basically it is nothing but the comparison part which has been done in Eq. 9 which has been embedded in the table as extra row. The respective coefficients of $f(x_i)$ and its derivatives are written corresponding to the top header. If this is done then the table becomes what has been shown in Table 3.

Table 3. Final Taylor table for Python programming

	$f(x_i)$	$f'(x_i)h$	$f''(x_i)h^2$
f_{i-1}	1	−1	1/2
f_i	1	0	0
f_{i+1}	1	1	1/2
	0	0	1

In the program, the main table is called as TT and the extra appended row as RHS. Then the transpose of TT and RHS are done and any of the linear algebra modes can be used to solve matrix $\mathrm{TT^T}$ and vector $\mathrm{RHS^T}$.

3. Programming Taylor' Table in Python

First function will be written to return array of Taylor series coefficients based on the number of terms (n_t) required and the value of α. This can be done easily once the coefficients in the Eq. 8 can be written in index form as shown in Eq. 11.

$$coeff_i = \sum_{i=0}^{n_t} \left(\frac{\alpha^i}{i!} \right) \tag{11}$$

The python function to do this task is as follows:

```
#   TAYLOR SERIES COEFFICIENTS
def taylor_coeff(n_t,a):
    """
    n_t: Number of terms in the series
    a: Represents α
    """
    coeff=empty(n_t)
    for i in range(n_t):
        coeff[i]=a**i/factorial(i)

    return coeff
```

The function to perform Taylor Table method will require two arguments (inputs); one will be the stencil (st) and second the order (O) of derivative whose finite difference approximation is required. The array for stencil (st) will contain the value of α in the order in which you want to write the approximation. For example if the equation is Eq. 7 then the stencil array will be :

```
st=array([-1,0,1])
```

Once array is supplied, the length of array is evaluated and passed to the variable 'nt' (**`nt=len(st)`**). Then an empty array is created which will later on contain the elements of Taylor table (**`TT=empty((nt,nt))`**). Now **`taylor_coeff()`** will be called in a loop to generate the table as follows:

```
for i in range(nt):
    TT[i,:]=taylor_coeff(nt,st[i])
```

Here one can observe that the index is running row wise and the columns are filled by the Taylor series coefficients with the help of function **taylor_coeff()** for every row i.e. for every term in the stencil.

Now the array for RHS vector is created with 'nt' number of elements in it. Then based on the order of derivative the Oth element of RHS is replace by '1' as follows:

```
RHS=zeros(nt)
RHS[O]=1
```

Then the transpose of matrix TT and RHS are solved with the help of **`linalg.solve()`** function of NumPy and the result is stored in the array 'a'. The elements of array 'a' represent the a, b, and c of Eq. 7 with missing h terms. Finally the result is declared by dividing the elements of array 'a' by the h^O as this is the only power of h which is left corresponding to non-zero element of RHS vector. The procedure is shown in the following code snippet:

```
a=linalg.solve(transpose(TT),transpose(RHS))

for i in range(nt):
    print(f"a[{i}] = {a[i]}/h^{O}")
```

One important thing to note that the provision for O to be less than the number of terms in the stencils should be done using **`if - else`** block, otherwise the error will popup. The function for TT method is shown below:

```
# TAYLOR'S TABLE METHOD
def TT_method(st,O):

    nt=len(st) # number of terms in the Taylor series

    if O<nt:

        TT=empty((nt,nt))
        for i in range(nt):
            TT[i,:]=taylor_coeff(nt,st[i])
        # TT

        # RHS vector:
        RHS=zeros(nt)


        RHS[O]=1

        a=linalg.solve(transpose(TT),transpose(RHS))

        for i in range(nt):
            print(f"a[{i}] = {a[i]}/h^{O}")
    else:
        print("The order of DE is more than the number of terms in
stencil")
```

4. Implementation of TT_method() function to the problems

Problem 1. Evaluate the forward difference formula for the first order derivative for the following stencil:

$$\frac{df}{dx} = a_0 f_i + a_1 f_{i+1} + a_2 f_{i+2}$$

Solution: Now the time has come to call the function. The only precaution to be taken is while supplying the data to the stencil.

Code	Output

`# Following points should be`
`considered before supplying`
`data to the stencil array`
`# 1. f_i    --> 0`
`# 2. f_(i-1)-->-1`
`# 3. f_(i+1)--> 1`
`# 4. f_(i-2)-->-2`
`# 5. f_(i+2)--> 2`
 	
`#Stencil`
`order=1`
`st=array([0,1,2])`
`TT_method(st,order)`
	`a[0] = -1.5/h^1`
	`a[1] = 2.0/h^1`
	`a[2] = -0.5/h^1`

Problem 2: Evaluate the backward difference formula for the first order derivative for the following stencil:

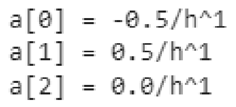
$$\frac{df}{dx} = a_0 f_{i-1} + a_1 f_{i+1} + a_2 f_{i+2}$$

Solution:

Code

```
# Note:
# 1. f_i    --> 0
# 2. f_(i-1)-->-1
# 3. f_(i+1)--> 1
# 4. f_(i-2)-->-2
# 5. f_(i+2)--> 2
st=array([-1,1,2]) #Stencil
order=1
TT_method(st,order)
```

Output



```
a[0] = -0.5/h^1
a[1] = 0.5/h^1
a[2] = 0.0/h^1
```

Problem 3: Evaluate the finite difference formula for the second order derivative for the following stencil:

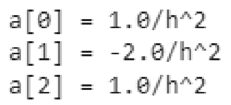
$$\frac{d^2 f}{dx^2} = a_0 f_{i-1} + a_1 f_i + a_2 f_{i+1}$$

Solution:

Code

```
# Note:
# 1. f_i    --> 0
# 2. f_(i-1)-->-1
# 3. f_(i+1)--> 1
# 4. f_(i-2)-->-2
# 5. f_(i+2)--> 2
st=array([-1,0,1]) #Stencil
order=2
TT_method(st,order)
```

Output



```
a[0] = 1.0/h^2
a[1] = -2.0/h^2
a[2] = 1.0/h^2
```

Problem 4: Evaluate the finite difference formula for the second order derivative for the following stencil:

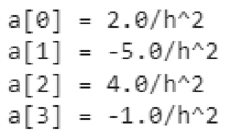
$$\frac{d^2 f}{dx^2} = a_0 f_i + a_1 f_{i+1} + a_2 f_{i+2} + a_3 f_{i+3}$$

Solution:

Code

```
# Note:
# 1. f_i    --> 0
# 2. f_(i-1)-->-1
# 3. f_(i+1)--> 1
# 4. f_(i-2)-->-2
# 5. f_(i+2)--> 2
st=array([0,1,2,3]) #Stencil
order=2
TT_method(st,order)
```

Output



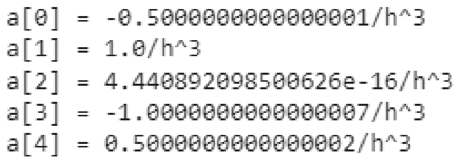
```
a[0] = 2.0/h^2
a[1] = -5.0/h^2
a[2] = 4.0/h^2
a[3] = -1.0/h^2
```

Problem 5: Evaluate the finite difference formula for the third order derivative for the following stencil:

$$\frac{d^2 f}{dx^2} = a_0 f_{i-2} + a_1 f_{i-1} + a_2 f_i + a_3 f_{i+1} + a_4 f_{i+2}$$

Solution:

Code	Output

`# Note:`
`# 1. f_i    --> 0`
`# 2. f_(i-1)-->-1`
`# 3. f_(i+1)--> 1`
`# 4. f_(i-2)-->-2`
`# 5. f_(i+2)--> 2`
`st=array([-2,-1,0,1,2])`
`#Stencil`
`order=3`
`TT_method(st,order)`
	
	`a[0] = -0.5000000000000001/h^3`
	`a[1] = 1.0/h^3`
	`a[2] = 4.440892098500626e-16/h^3`
	`a[3] = -1.0000000000000007/h^3`
	`a[4] = 0.5000000000000002/h^3`

5. Conclusion

In this short article, Taylor table method for the evaluation of finite difference formulation of derivatives has been explained. Python function for Taylor table method has been developed with the help of a detailed algorithm elucidated in the 2nd section. The developed function (TT_method( )) is tested onto different derivatives and stencils. The outcome of the results is very encouraging as the function is capable to handle any stencil with any order of derivative. The developed function and problem-solution strategy will help beginners of numerical computations to get a fair degree of understanding of the method and will help them in learning finite difference.

References

[1] R.A. Usmani, P.J. Taylor, Finite Difference Methods for Solving, Int. J. Comput. Math. 14 (1983) 277–293. https://doi.org/10.1080/00207168308803391.

[2] J.F. Epperson, An introduction to numerical methods and analysis, John Wiley \& Sons, 2021. https://doi.org/10.1002/9781119604754.

[3] P. Dumka, R. Dumka, D.R. Mishra, Numerical Methods Using Python, BlueRose, 2022.

[4] B. Hashemi, Y. Nakatsukasa, Least-squares spectral methods for ODE eigenvalue problems, (2021). http://arxiv.org/abs/2109.05384.

[5] P.S. Pawar, D.R. Mishra, P. Dumka, Solving First Order Ordinary Differential Equations using Least Square Method : A comparative study, Int. J. Innov. Sci. Res. Technol. 7 (2022) 857–864.

[6] R. Anderssen, M. Hegland, For numerical differentiation, dimensionality can be a blessing!, Math. Comput. 68 (1999) 1121–1141. https://doi.org/10.1090/s0025-5718-99-01033-9.

[7] V. T., J.C. Strikwerda, Finite Difference Schemes and Partial Differential Equations., SIAM, 1990. https://doi.org/10.2307/2008454.

[8] A. Marowka, On parallel software engineering education using python, Educ. Inf. Technol. 23 (2018) 357–372. https://doi.org/10.1007/s10639-017-9607-0.

[9] P. Dumka, R. Chauhan, A. Singh, G. Singh, D. Mishra, Implementation of Buckingham ' s Pi theorem using Python, Adv. Eng. Softw. 173 (2022) 103232. https://doi.org/10.1016/j.advengsoft.2022.103232.

[10] P. Dumka, K. Rana, S. Pratap, S. Tomar, P.S. Pawar, D.R. Mishra, Modelling air standard thermodynamic cycles using python, Adv. Eng. Softw. 172 (2022) 103186. https://doi.org/10.1016/j.advengsoft.2022.103186.

[11] J.W.B. Lin, Why python is the next wave in earth sciences computing, Bull. Am. Meteorol. Soc. 93 (2012) 1823–1824. https://doi.org/10.1175/BAMS-D-12-00148.1.

[12] R. Johansson, Numerical python: Scientific computing and data science applications with numpy, SciPy and matplotlib, Second edition, Apress, Berkeley, CA, 2018. https://doi.org/10.1007/978-1-4842-4246-9.

[13] C. Fuhrer, O. Verdier, J.E. Solem, C. Führer, O. Verdier, J.E. Solem, Scientific Computing with Python. High-performance scientific computing with NumPy, SciPy, and pandas, Packt Publishing Ltd, 2021.

[14] A. Meurer, C.P. Smith, M. Paprocki, O. Čertík, S.B. Kirpichev, M. Rocklin, Am.T. Kumar, S. Ivanov, J.K. Moore, S. Singh, T. Rathnayake, S. Vig, B.E. Granger, R.P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M.J. Curry, A.R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, A. Scopatz, SymPy: Symbolic computing in python, PeerJ Comput. Sci. 2017 (2017) 1–27. https://doi.org/10.7717/peerj-cs.103.

[15] P. Dumka, P.S. Pawar, A. Sauda, G. Shukla, D.R. Mishra, Application of He's homotopy and perturbation method to solve heat transfer equations: A python approach, Adv. Eng. Softw. 170 (2022) 103160. https://doi.org/10.1016/j.advengsoft.2022.103160.

Authors' Profiles

Pankaj Dumka has completed his B. Tech. (Mechanical Engineering) from Inderprastha Engineering College, Ghaziabad in 2007, M. Tech. from Indian Institute of Technology, Kanpur in "Fluids and Thermal Sciences" in 2010, and PhD from Jaypee University of Engineering and Technology, Guna in 2021. He has been working with the Jaypee University of Engineering and Technology as an assistant professor in the Department of Mechanical Engineering since 2011. He has published more than 40 research articles in reputed SCI and SCOPUS indexed Journals. His area of interest includes Thermodynamics, Fluid Dynamics, Computational Fluid Dynamics, Numerical Computations, Python programming, and Solar water desalination.

Rishika Chauhan has been working with the Jaypee University of Engineering and Technology as an assistant professor in the Department of Electronics and Communication Engineering since 2011. She teaches both undergraduate and postgraduate students of engineering at the University. Her area of interest includes Wireless Communication, Numerical Computations, Artificial Neural Networks, and Discrete Mathematics. Mrs. Chauhan has published number of research publications in numerous journals of national and international repute.

Dhananjay R. Mishra has obtained his bachelor's degree (Mechanical Engineering) from Amravati University (M.S.), Master's degree (Production Engineering) in 2007 from BIT, Durg of Pt. Ravi Shankar Shukla University, Raipur (C.G.), and Doctor of Philosophy in Mechanical Engineering from National Institute of Technology Raipur, Raipur, on August 2016. He is working as an Assistant Professor (SG) in the Mechanical Engineering Department of Jaypee University of Engineering and Technology, Guna (M.P.). He has published more than 125 research articles in peer-reviewed international, national journals, and conferences. His area of interest includes Solar thermal Applications, Renewable Energy, Python Programming, Numerical Computations, Internal combustion engines, and Solar Water Desalination.

How to cite this paper: Pankaj Dumka, Rishika Chauhan, Dhananjay R. Mishra, "Modelling Taylor's Table Method for Numerical Differentiation in Python", International Journal of Mathematical Sciences and Computing(IJMSC), Vol.9, No.4, pp. 20-28, 2023. DOI: 10.5815/ijmsc.2023.04.03