

To workout a third order method for approximating the first derivative of a function  $f$ , based on a non-symmetric 4-point difference formula for points  $x + 2h, x + h, x - 2h$  and  $x$ , I used the 3rd order Taylor polynomial expansion in each of the points around  $x$ , resulting in 4 equations:

$$f(x + 2h) = f(x) + 2hf'(x) + 2h^2f''(x) + \frac{4}{3}h^3f'''(x) + \frac{2}{3}h^4f^{(4)}(c_1), \quad c_1 \in [x, x + 2h] \quad (1)$$

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(c_2), \quad c_2 \in [x, x + h] \quad (2)$$

$$f(x - 2h) = f(x) - 2hf'(x) + 2h^2f''(x) - \frac{4}{3}h^3f'''(x) + \frac{2}{3}h^4f^{(4)}(c_3), \quad c_3 \in [x - 2h, x] \quad (3)$$

$$f(x) = f(x) \quad (4)$$

By looking at these equalities, we want to find a linear combination of them that results in nulifying terms where  $f(x)$ ,  $f''(x)$  and  $f'''(x)$  appear. As such, we can consider the following linear system for which we want to find the null space:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & \frac{1}{2} & 2 & 0 \\ \frac{4}{3} & \frac{1}{6} & -\frac{4}{3} & 0 \end{bmatrix}$$

The resulting null space is given by:

$$\begin{bmatrix} 3z & -16z & z & 12z \end{bmatrix}^T, \quad z \in \mathbb{R}$$

Taking  $z = 1$  and doing a linear combination of the resulting vectors components by (1), (2), (3) and (4) respectively, we end up with

$$f'(x) = \frac{-3f(x + 2h) + 16f(x + h) - 12f(x) - f(x - 2h)}{12h} + \frac{h^3}{6}f^{(4)}(c_4), \quad c_4 \in [x - 2h, x + 2h]$$

Thus, arriving to our third order approximation for the first derivative of  $f$  based on a non-symmetric 4-point difference formula

```
1 import numpy as np
2 """
3 This second of the code fetches the machine precision, and in combination with the
4 method order computes the best theoretical tK such that h=10^(-k) gives the best
5 approximation
6 """
7
8 epsilon = np.finfo(float).eps
9 order = 3
10 tK = -np.log10(epsilon)/(order+1)
11
12 print(f"Machine precision epsilon = {epsilon}")
13 print(f"Method order n = {order}")
14 print(f"Base 10 highest theoretical k = {tK}\n")

```

```
Machine precision epsilon = 2.220446049250313e-16
Method order n = 3
Base 10 highest theoretical k = 3.9133899436317554
```

```
1 """
2 Here we define the functions:
3 - f() -> our original function given x
4 - df() -> the analytic derivative of f() given x
5 - method() -> the function that computes the approximation given x and h,
6   based on function f()
7 - out() -> function that iterates through 1 to 15 steps k, defining
8   h = 10^(-k), and computing the error with the difference between
9   method() and df(), given a point x
10 """
11
12 def f(x: float) -> float:
13     return np.power(x,1/3.0) + x
14
15 def df(x: float) -> float:
16     return (1/3.0)*np.power(x,-2/3) + 1
17
18 def method(x: float, h: float) -> float:
19     return (-3*f(x+2*h)+16*f(x+h)-12*f(x)-f(x-2*h))/(12*h)
20
21
```

```

22
23 def out(x: float):
24     h:float = 1.0
25     for k in range(1,16):
26         h = h*0.1
27         aprox = method(x,h)
28         real = df(x)
29         error = real-aprox
30         print(f"h = 10^{k} => f'(x) ~ {aprox} | error = {error}")

1 """
2 For this part of exercise we'll evaluate the approximation in point x=1
3 and confirm the best value k integer occurs near our theoretical tK.
4 """
5 out(1)

h = 10^(1) => f'(x) ~ 1.3334903108595733 | error = -0.0001569775262399986
h = 10^(2) => f'(x) ~ 1.333333496781802 | error = -1.6344846875959718e-07
h = 10^(3) => f'(x) ~ 1.3333333334975916 | error = -1.642583846717116e-10
h = 10^(4) => f'(x) ~ 1.333333333313357 | error = 1.9975132659055816e-12
h = 10^(5) => f'(x) ~ 1.333333333457684 | error = -1.2435164009616528e-11
h = 10^(6) => f'(x) ~ 1.333333332236441 | error = 1.0968914665454577e-10
h = 10^(7) => f'(x) ~ 1.3333333341118223 | error = -7.784890510009745e-10
h = 10^(8) => f'(x) ~ 1.3333333659382156 | error = -3.260488234957393e-08
h = 10^(9) => f'(x) ~ 1.3333334436538276 | error = -1.1032049429537949e-07
h = 10^(10) => f'(x) ~ 1.3333334436538273 | error = -1.1032049407333488e-07
h = 10^(11) => f'(x) ~ 1.333333443653827 | error = -1.1032049385129028e-07
h = 10^(12) => f'(x) ~ 1.3330447856674246 | error = 0.0002885476659086894
h = 10^(13) => f'(x) ~ 1.332267629550187 | error = 0.001065703783146299
h = 10^(14) => f'(x) ~ 1.3470706032118556 | error = -0.013737269878522307
h = 10^(15) => f'(x) ~ 1.99840144432528 | error = -0.6650681109919467

```