NM Homework 10

Adam Kit - 3707437

18 June 2020

1 First Order derivatives with three point formula

The three-point method for approximating $f'(x_j)$ is given by Eq. 1. The points are represented in terms of step size (grid spacing), i.e., $x_0 = x_0$, $x_1 = x_0 + h$, $x_2 = x_0 + 2h$.

$$f'(x_j) = f(x_0)\left[\frac{2x_j - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}\right] + f(x_1)\left[\frac{2x_j - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}\right] + f(x_2)\left[\frac{2x_j - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}\right] + E_j \tag{1}$$

where the error term is represented by

$$E_j = \frac{f^{(2)}(\varepsilon(x_j))}{6} \prod_{k=0;k\neq j}^{2} (x_j - x_k)$$

When we look at the point x_0, we find:

$$\begin{aligned}
f'(x_0) = f(x_0)&\left[\frac{2x_0 - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}\right] + f(x_1)\left[\frac{2x_0 - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}\right] + f(x_2)\left[\frac{2x_0 - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}\right] + E_0 \\
&= f(x_0)\left[\frac{-3h}{(-h)(-2h)}\right] + f(x_1)\left[\frac{-2h}{(h)(-h)}\right] + f(x_2)\left[\frac{-h}{(2h)(h)}\right] + E_0 \\
&= f(x_0)\left[\frac{-3}{2h}\right] + f(x_1)\left[\frac{2}{h}\right] + f(x_2)\left[\frac{-1}{2h}\right] + E_0 \\
&= \frac{1}{2h}\left[-3f(x_0) + 4f(x_1) - f(x_2)\right] + E_0
\end{aligned}$$

The same type of derivation goes for point x_1:

$$\begin{aligned}
f'(x_1) = f(x_0)&\left[\frac{2x_1 - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}\right] + f(x_1)\left[\frac{2x_1 - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}\right] + f(x_2)\left[\frac{2x_1 - x_0 - x_1}{(x_1 - x_0)(x_1 - x_2)}\right] + E_1 \\
&= f(x_0)\left[\frac{-h}{2h^2}\right] + f(x_1)\left[\frac{0}{-h^2}\right] + f(x_2)\left[\frac{h}{2h^2}\right] + E_1 \\
&= -f(x_0)\left[\frac{1}{2h}\right] + f(x_2)\left[\frac{1}{2h}\right] + E_1 = \frac{1}{2h}\left[-f(x_0) + f(x_2)\right] + E_1
\end{aligned}$$

And for x_2:

$$\begin{aligned}
f'(x_2) &= f(x_0)\left[\frac{2x_2 - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}\right] + f(x_1)\left[\frac{2x_2 - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}\right] + f(x_2)\left[\frac{2x_2 - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}\right] + E_2 \\
&= f(x_0)\left[\frac{h}{2h^2}\right] + f(x_1)\left[\frac{2h}{-h^2}\right] + f(x_2)\left[\frac{3h}{2h^2}\right] + E_2 = f(x_0)\left[\frac{1}{2h}\right] - f(x_1)\left[\frac{2}{h}\right] + f(x_2)\left[\frac{3}{2h}\right] + E_2 \\
&= \frac{1}{2h}\left[f(x_0) - 4f(x_1) + 3f(x_2)\right]
\end{aligned}$$

1.1 Error

2 Second order derivatives and the three point formula

The second order derivative for the three point formula given in the lecture notes is written below:

$$\frac{f(x+h) - 2f(x) + f(x-h)}{h^2} = f''(x) + \frac{h^2}{12}f^{(4)}(\varepsilon) \tag{2}$$

The analytical derivative of $f(x)$ is:

$$f'(x) = -\frac{1}{x^2} \Rightarrow f'(1) = -1$$

and the second derivative:

$$f''(x) = 2x^{-3} \Rightarrow f''(1) = 2$$

The approximate absolute error is found as:

$$\epsilon = |y - y_{approx}|$$

and the relative error:

$$\nu = \frac{\epsilon}{|y|}$$

2.1 $h = 0.1$

With $h = 0.1$ and $x = 1$, equation 2 becomes:

$$\frac{f(1+0.1) - 2f(1) + f(1-0.1)}{0.1^2} = \frac{\frac{1}{1.1} - 2 + \frac{1}{0.9}}{0.01} = 2.020202$$

This has a absolute error of .02 and a relative error of 0.01 or 1%

2.2 $h = 0.01$

With $h = 0.01$ and $x = 1$, equation 2 becomes:

$$\frac{f(1+0.01) - 2f(1) + f(1-0.01)}{0.01^2} = \frac{\frac{1}{1.01} - 2 + \frac{1}{0.99}}{0.0001} = 2.000200020002$$

Which will have an absolute error of 0.0002 and a relative error of 0.0001 or 0.01%

2.3 $h = 0.001$

With $h = 0.001$ and $x = 1$, equation 2 becomes:

$$\frac{f(1+0.001) - 2f(1) + f(1-0.001)}{0.001^2} = \frac{\frac{1}{1.001} - 2 + \frac{1}{0.999}}{0.000001} = 2.000002000002000002$$

Which will have an absolute error of 0.000002 and a relative error of 0.000001 or 0.0001% The errors for each h are all on the order of h^2

3 Practicality

For the first derivative, we can use the modified 5-point endpoint formula (see eq 3), which in this case places x_0 at 1.5 and $h = 0.5$.

$$f'(x_0) = \frac{1}{12h}\left[-25f(x_0) - 48f(x_0+h) - 36f(x_0+2h) + 16f(x_0+3h) - 3f(x_0+4h)\right] + \frac{h^4}{5}f^{(5)}(\varepsilon) \quad (3)$$

So plugging in for $x_j = x_0 = 1.5$, we see:

$$f'(x_0) = \frac{1}{12h}\left[-25(3.375) - 48(7) - 36(13.625) + 16(38.875) - 3(59)\right] + \frac{h^4}{5}f^{(5)}(\varepsilon)$$

4 Integration

Okay a more fun approach to this is Monte Carlo integration, however it is sometimes not as effecient. I have shown this in my python tutorials found in my github `fusionby2030.github.io`. The actual value of our integral from which we approximate the error is 2.

4.1 Trapezoidal Rule

The trapezoidal rule is defined below, where we consider a partition of the domain we wish to integrate over $(a = 0, b = \pi)$ as $\{x_k\}$:

$$\int_b^a f(x)dx \approx \sum_{k=1}^{N} \frac{f(x_{k-1}) + f(x_k)}{2}\Delta x_k \quad (4)$$

The code for this problem can be found in the file: and is clipped in figure 1 where we find that $\int_0^\pi \sin(x)dx \approx 1.9663$, which has an absolute error of 0.03368 and relative error of 0.01684 or $\approx 1.6\%$

4.2 Simpsons Rule

The formula for Simson's rule is below where N is the number of partitions of $[a, b]$ and $\Delta x = (b-a)/N$ and $x_i = a + i\Delta x$

$$S_N(f) = \frac{\Delta x}{3}\sum_{i=1}^{N/2}(f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})) \quad (5)$$

the code for this problem can be foudn in the figure 2 where we find $\int_0^\pi \sin(x)dx \approx 2.000269$, which then has an absolute error of: 0.000269 and a relative error of 0.000135 or 0.0135%

```python
"""
Trapezoidal Rule
partition domain [a,b] into N equally spaced (h) points
sum from 1st point to N then multiply by h/2
"""
func = lambda x: np.sin(x)

def traprule(f, a, b, n):
    partition, h = np.linspace(a, b, num=n, retstep=True)
    sum = 0
    for i in range(1, len(partition)):
        sum += h*(func(partition[i-1]) + func(partition[i-1]))/2.0
    return sum

traprule(func, 0, np.pi, 8) 1.9663166787658921
```

Figure 1: Trapezoidal Rule in Python

```python
def simpsonrule(f, a, b, n):
    partition= np.linspace(a, b, n+1)
    h = (b-a)/n
    y = f(partition)
    S = h/3.0*np.sum(y[0:-1:2] + 4*y[1::2] + y[2::2])
    return S
simpsonrule(func, 0, np.pi, 8) 2.0002691699483877
```

Figure 2: Simpsons Rule in Python