FA3_KHAFAJI

March 10, 2025

[1]:
```
from sympy import *
import numpy as np
import polars as pl
```

0.0.1 1.

[2]:
```
x = symbols('x')
h= 0.1
x_list = [2.9, 3, 3.1, 3.2]
fx_list = [-4.827866, -4.240058, -3.496909, -2.596792]
```

[3]:
```
def three_point(x_vals, fx_vals, h):

    f_prime_vals = []
    for idx, fval in enumerate(fx_vals):
        if idx == 0:
            x_plus_h_idx = x_vals.index(x_vals[idx] + h)
            x_plus_2h_idx = x_vals.index(x_vals[idx] + 2*h)
            fx_plus_h = fx_vals[x_plus_h_idx]
            fx_plus_2h = fx_vals[x_plus_2h_idx]

            f_prime = (1/(2*h))*(-3*fval +4*fx_plus_h-fx_plus_2h) # left approx
            f_prime_vals.append(f_prime)

        elif idx == len(fx_vals)-1:
            x_minus_h_idx = x_vals.index(x_vals[idx] - h)
            x_minus_2h_idx = x_vals.index(x_vals[idx] - 2*h)
            fx_minus_h = fx_vals[x_minus_h_idx]
            fx_minus_2h = fx_vals[x_minus_2h_idx]

            f_prime = (1/(2*h))*(3*fval -4*fx_minus_h +fx_minus_2h) # right␣
↪approx
            f_prime_vals.append(f_prime)

        else:
            x_plus_h_idx = x_vals.index(x_vals[idx] + h)
            x_minus_h_idx = x_vals.index(x_vals[idx] - h)
            fx_plus_h = fx_vals[x_plus_h_idx]
```

```
            fx_minus_h = fx_vals[x_minus_h_idx]

            f_prime = (1/(2*h))*(-fx_minus_h+fx_plus_h) #midpoint
            f_prime_vals.append(f_prime)

    return f_prime_vals
```

[4]:
```
fprime_x_list = three_point(x_list, fx_list,h)
data = {
    "x":x_list,
    "fx":fx_list,
    "fprime_x":fprime_x_list
}
```

[5]:
```
df = pl.DataFrame(data)
df
```

[5]:
```
shape: (4, 3)

 x     fx          fprime_x
 ---   ---         ---
 f64   f64         f64

 2.9   -4.827866   5.101375
 3.0   -4.240058   6.654785
 3.1   -3.496909   8.21633
 3.2   -2.596792   9.78601
```

0.0.2 2.

[6]:
```
x = symbols('x')
fx_2 = tan(x)
h = 0.1
nodes = [ 2+0.1*n for n in range(1,6)]
print(nodes)
```

```
[2.1, 2.2, 2.3, 2.4, 2.5]
```

[7]:
```
fx_2_5prime = simplify(diff(fx_2, x, 4))

max_in_interval = maximum(fx_2_5prime, x, Interval(2.1,2.5))

error = (max_in_interval)*(h**4)/30

print(f"the absolute maximum error is {abs(error)}")
```

```
the absolute maximum error is 0.000114034352711179
```

0.0.3 3.

```
[8]: x_list_3 = [1.28, 1.29, 1.3, 1.31, 1.4]
y_list_3 = [11.59006, 13.78176, 14.04276, 14.30741, 16.86187]
h=0.01

approx_3 = (y_list_3[1]-2*y_list_3[2]+y_list_3[3])/(h**2)

print(f"The approximate derivative of f(1.3) is {round(approx_3,10)}")
```

```
The approximate derivative of f(1.3) is 36.5
```

0.0.4 4.

getting $N_3(h)$, an approximation to $f'(x_0)$ for $f(x) = x + e^x$, with h=0.4

```
[9]: x = symbols('x')
x_list_4 = [-0.4, 0, 0.4]
f_3 = x + E**(x)
h= 0.4

n1_h = (1/(2*h))*(-f_3.subs(x,-0.4)+f_3.subs(x,0.4)) #midpoint

n1_hdiv2 = (1/(2*(h/2)))*(-f_3.subs(x,-0.2)+f_3.subs(x,0.2))

n1_hdiv4 = (1/(2*(h/4)))*(-f_3.subs(x,-0.1)+f_3.subs(x,0.1))

n2_h = (1/3)*(4*n1_hdiv2 - n1_h)

n2_hdiv2 = (1/3)*(4*n1_hdiv4 - n1_hdiv2)

n3_h = (1/15)*(16*n2_hdiv2-n2_h)

print(f"The approximation to the derivative of the given function at x_0 = 0 is␣
↪{n3_h}")
```

```
The approximation to the derivative of the given function at x_0 = 0 is
2.00000001273551
```

0.0.5 5.

approximate

$$\int_0^{\frac{\pi}{4}} xsin(x)\, dx$$

using the Trapezoidal rule

```
[10]: x = symbols('x')
f_5 = x*sin(x)
```

```
x_max_5 = (pi/4)
x_min_5 = 0
h_5 = x_max_5 - x_min_5
approx_5 = ( h_5/2) *(f_5.subs(x,x_max_5)+f_5.subs(x,x_min_5))

approx_5 = simplify(approx_5)

print(f"The approximation of the integral using the trapezoidal rule is:␣
 ↪{approx_5}, which is approximately 0.2180895062")
approx_5
```

```
The approximation of the integral using the trapezoidal rule is:
sqrt(2)*pi**2/64, which is approximately 0.2180895062
```

[10]:

$$\frac{\sqrt{2}\pi^2}{64}$$

0.0.6 6.

approximate

$$\int_0^{\frac{\pi}{4}} xsin(x)\,dx$$

using the Simpson's rule

[11]:

```
a = 0
b = pi/4
h_6 = (b-a)/2
x_mid_6 = a+h_6

x = symbols('x')
f_6 = x * sin(x)

simpsons_approx_6 = (h/3) * (
        f_6.subs(x,a) + 4*f_6.subs(x, x_mid_6) + f_6.subs(x, b)
)

simpsons_approx_6 = simplify(simpsons_approx_6)

print(f"The Simpson's Approximate of the Polynomial is: {simpsons_approx_6},␣
 ↪which is approximately 0.1513826289")

simpsons_approx_6
```

```
The Simpson's Approximate of the Polynomial is: pi*(0.0166666666666667*sqrt(2) +
0.0333333333333333*sqrt(2 - sqrt(2))), which is approximately 0.1513826289
```

[11]:

$$\pi\left(0.0166666666666667\sqrt{2} + 0.0333333333333333\sqrt{2-\sqrt{2}}\right)$$

0.0.7 7.

Give the errors for 5 and 6

for 5. the error is given by $\frac{h^3}{12}f''(\xi)$

for 6. the error is given by $\frac{h^5}{90}f^{(4)}(\xi)$

[12]:
```
# solving for error for number 5

f_2ndprime_7 = diff(f_5, x, 2)

print(f"The 2nd derivative of the given function is: {f_2ndprime_7}")

print("Since x is small in the domain, and the range of both sinx and cosx in␣
 ↪the domain are from 0 to 0.785, the function will yield the highest value in␣
 ↪the interval at f''(0)")

error_trap_7 =  ((h_5**3)/12) * f_2ndprime_7.subs(x, 0)

error_trap_7 = simplify(error_trap_7)

print(f"The absolute maximum error for number 5 is {abs(error_trap_7)}")
abs(error_trap_7)
```

```
The 2nd derivative of the given function is: -x*sin(x) + 2*cos(x)
Since x is small in the domain, and the range of both sinx and cosx in the
domain are from 0 to 0.785, the function will yield the highest value in the
interval at f''(0)
The absolute maximum error for number 5 is pi**3/384
```

[12]: $\frac{\pi^3}{384}$

[13]:
```
# solving for error in number 6

f_4thprime_7 = diff(f_6, x, 4)

print(f"The 4th derivative of the given function is: {f_2ndprime_7}")

print("Since x is small in the domain, and the range of both sinx and cosx in␣
 ↪the domain are from 0 to 0.785, the function will yield the highest value in␣
 ↪the interval at f''(0)")

error_simp_7 =  ((h_6**5)/90) * f_4thprime_7.subs(x, 0)

error_simp_7 = simplify(error_simp_7)

print(f"The absolute maximum error for number 5 is {abs(error_simp_7)}")
abs(error_simp_7)
```

```
The 4th derivative of the given function is: -x*sin(x) + 2*cos(x)
Since x is small in the domain, and the range of both sinx and cosx in the
domain are from 0 to 0.785, the function will yield the highest value in the
interval at f''(0)
The absolute maximum error for number 5 is pi**5/737280
```

[13]: $\frac{\pi^5}{737280}$

0.0.8 8.

approximate $\int_1^1 0 \frac{1}{x}\, dx$ using Closed and Open Newton-Cotes formula for n=3, Are the accuracies consistent with the error formulas?

[14]:
```python
# closed newton cotes
x = symbols('x')

f_8 = 1/x
n=3
a_8 = 1
b_8 = 10
h_8 = (10-1)/n

x1_8_closed = a_8 + h_8
x2_8_closed = a_8 +2*h_8

approx_closed_8 = ((3*h_8)/8)*(
    f_8.subs(x, a_8)
    + 3*f_8.subs(x, x1_8_closed)
    + 3 *f_8.subs(x, x2_8_closed)
    + f_8.subs(x, b_8)
)

approx_closed_8 = simplify(approx_closed_8)

print(f"The approximated integral using the Closed Newton-Cotes formula is␣
↪{approx_closed_8}")
```

```
The approximated integral using the Closed Newton-Cotes formula is
2.56339285714286
```

[15]:
```python
h_8_open = (b_8 - a_8)/(n+2)

x_vals_8 = [ 1 + h_8_open*p for p in range(1,5)]


approx_open_8 = ((5*h_8_open)/24)*(
    11* f_8.subs(x, x_vals_8[0])
    + f_8.subs(x, x_vals_8[1])
```

```
    + f_8.subs(x, x_vals_8[2])
    + 11*f_8.subs(x, x_vals_8[3])
)

approx_open_8 = simplify(approx_open_8)

print(f"The approximated integral using the Open Newton-Cotes formula is␣
 ↪{approx_open_8}")
```

```
The approximated integral using the Open Newton-Cotes formula is
2.11637855533253
```

[16]:
```
# getting the actual integral

# getting the indefinite integral:
integral_8 = integrate(f_8, x)

print(f"The indefinite integral is {integral_8}, which is the natural␣
 ↪logarithm")



# evaluating the integral:
integral_evaluated_8 = integral_8.subs(x, b_8) - integral_8.subs(x, a_8)
integral_evaluated_8 = simplify(integral_evaluated_8)
print(f"The evaluated integral results to: {integral_evaluated_8}, which is␣
 ↪approximately equal to 2.302585093")
```

```
The indefinite integral is log(x), which is the natural logarithm
The evaluated integral results to: log(10), which is approximately equal to
2.302585093
```

[19]:
```
f_4thdiff_8 = diff(f_8, x, 4)

print(f"the 4th prime of our function is {f_4thdiff_8}. From the interval of␣
 ↪[1,10], x=1 provides the maximum value, giving us f(x) = 24")

error_closed = ((3*(h_8**5))/80)*24
error_open = (95/144)*(h_8_open**5)*24

print(f"\nThe absolute maximum error for the Closed Newton-Cotes formula is␣
 ↪{abs(round(error_closed,3))}, while it is {abs(error_open)} for the Open␣
 ↪Newton-Cotes formula")

print("\nThe error values we obtained are very big, which can be blamed on our␣
 ↪huge intervals (3 for closed, 1.8 for open). Although our approximations are␣
 ↪well within the error range, it is still worrying whether or not our␣
 ↪approximations are approximations or wild guesses.")
```

```
the 4th prime of our function is 24/x**5. From the interval of [1,10], x=1
```

provides the maximum value, giving us f(x) = 24

The absolute maximum error for the Closed Newton-Cotes formula is 218.7, while it is 299.1816 for the Open Newton-Cotes formula

The error values we obtained are very big, which can be blamed on our huge intervals (3 for closed, 1.8 for open). Although our approximations are well within the error range, it is still worrying whether or not our approximations are approximations or wild guesses.

[17]:

```

```