Homework 1

Arnab Bachhar

September 16, 2023

1. The following five data points are given: f (0.2) = 0.0015681, f (0.3) = 0.0077382, f (0.4) = 0.023579, f (0.5) = 0.054849, f (0.6) = 0.10696. This problem is about writing a code to evaluate f (0.16) and f (0.46) by using a fourth-order polynomial interpolation function.

→ The nth order polynomial formula is:

$$f(x) = \sum_{k=0}^{n+1} f_k P_k^n(x) \tag{1}$$

$$P_i(x) = \Pi_{j\neq i}^{n+1} \frac{(x - x_j)}{(x_i - x_j)} \tag{2}$$

for n+1 data points. The numerical error is expressed as:

$$\delta f(x) = \Pi_{i=0}^{n+1} \frac{f^{n+1}(x)}{(n+1)!}(x - x_i) \tag{3}$$

The result I got from the 4th order polynomial is:

polynomial value = 0.040135374720 error : 0.000000004675

polynomial value = 0.000451940320 error : -0.000000000757

• Write a program for the Lagrange interpolation by using the Neville's method. Compute f (0.16) and f (0.46) and estimate their numerical uncertainties.

$$P(x) = \frac{(x - x_j)P_{0,1,\cdots,j-1,j+1,\cdots,k}(x) - (x - x_i)P_{0,1,\cdots,i-1,i+1,\cdots,k}(x)}{(x_i - x_j)} \tag{4}$$

The result I got from Neville's Lagrange Interpolation is:

polynomial value =0.000451940320 in neville's method at x=0.16 ; numerical uncertainty : 0.000396070080

polynomial value =0.040135374720 in neville's method at x=0.46 ;numerical uncertainty : 0.000011094400

• Compare the above results to those using the linear interpolation.

$$f(x) = f_i + \frac{(f_{i+1} - f_i)}{(x_{i+1} - x_i)}(x - x_i) \tag{5}$$

Results from the linear interpolation method:

Intrapolated value = undetermined as x is outside the data range at x = 0.16

Intrapolated value = 0.042341000000 in linear interpolation method at x = 0.46

If we compare all the three methods, we can see both Neville's Lagrange interpolation and 4th order polynomial method are able to get the interpolated values at both x=0.16 and 0.46; but the linear interpolation method can't get that much accuracy and also can't predict the interpolated value at x=0.16.

2. Write a program to calculate the first-order and second-order derivatives of $f(x) = x^3 cos(x)$ at $x \in [\mathbf{\Pi/2}, \mathbf{\Pi}]$ and estimate the numerical accuracy. You may use the three-point formulas for the calculations of the first-order and second-order derivatives. Use 100 uniform intervals in the range. For the boundary points, you may use the formulas discussed in the class or extrapolated values by using the linear interpolation or the Lagrange interpolation. The numerical accuracy at each x value in the range can be obtained by comparing to the analytical result. The analytical function and it's derivatives are expressed below:

Analytical function:

$$f(x) = x^3 cos(x) \tag{6}$$

Analytic first derivative:

$$f'(x) = 3x^2 cos(x) - x^3 sin(x) \tag{7}$$

Analytic second derivative:

$$f''(x) = 6xcos(x) - 6x^2sin(x) - x^3cos(x) \tag{8}$$

Numerical first derivative (three-point formula):

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{9}$$

Numerical second derivative (three-point formula):

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{10}$$

If the program is run , a table will be displayed as result that contains the analytical results and numerical results and errors between them . I didn't show the table here. If you run the code ,it will display the table.

Maximum error: f'(x) = 0.00340606

Maximum error: f"(x) = 0.00201152

These error are on the boundary points while the error is less than that in intermediate points.

3. Write a program to calculate the integral

$$\int_0^1 e^{-x}dx \tag{11}$$

and estimate its numerical accuracy by using the Simpson rule. The numerical accuracy can be obtained by comparing with the analytical result. The simpson's rule can be expreesed as:

$$\textbf{Integral} = \sum_0^{\frac{n}{2}-1} (h/3) * (f_{2i}) + 4f_{2i+1}) + f_{2i+2})\text{for n=even;} \tag{12}$$

$$\textbf{Integral}+ = \frac{h}{12}(-f_{n-2} + 8f_{n-1} + 5f_n))\text{for n=odd} \tag{13}$$

where h=(b-a)/n; b and a are integral limits.

The extra equation 13 is needed for n=odd number and this is constructed via lagrange interpolation.

Results got from the code mentioned below:

Analytical Integral = 0.632120558829

n = 10; Numerical integral = 0.632120909589; Numerical uncertainty=0.000000350760458

n = 100; Numerical integral = 0.632120558864; Numerical uncertainty=0.000000000035117

n = 1000; Numerical integral = 0.632120558829; Numerical uncertainty=0.000000000000003

n = 10000; Numerical integral = 0.632120558829; Numerical uncertainty=0.000000000000001

We can see from the results that as the n increases the numerical uncertainty decreases i.e accuracy increases. To see the code is working both for even and odd numbers , I am randomly generating the n in the code also.