Numerical Methods (DS288): Assignment 4

Name: Subhasis Biswas
Serial Number: 23-1-22571

Question 1:

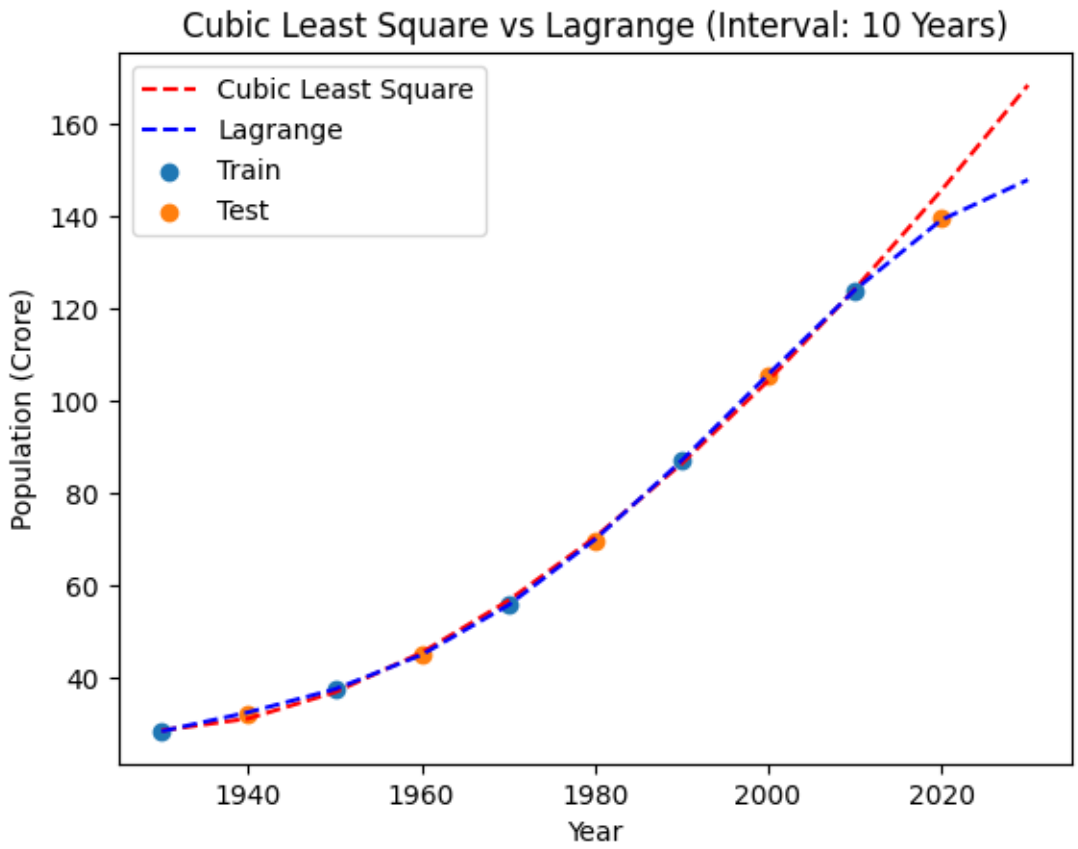
• Table for predictions:

Year	Actual	Cubic	Lagrange
1940	32.400000	31.256881	32.607812
1960	45.100000	45.526503	45.045313
1980	69.700000	70.488502	70.007812
2000	105.700000	104.443833	105.795312
2020	139.600000	145.693447	139.307812

Cubic Least Square RMSE: 2.857
Lagrange RMSE: 0.217

Polynomials found:
Cubic: $-3.5397 \times 10^{-5}x^3 + 0.2215x^2 - 459.3389x + 315961.7998$
Lagrange Interpolation: $-2.9688 \times 10^{-6}x^4 + 0.0234x^3 - 68.9017x^2 + 90308.5142x - 44376964.2047$

•



• The following procedure can be followed to obtain a general least square nth degree polynomial fit to a certain set of given data. We also include a screenshot from the textbook to help us understand the procedure.

Polynomial Least Squares

The general problem of approximating a set of data, $\{(x_i, y_i) \mid i = 1, 2, \ldots, m\}$, with an algebraic polynomial

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0,$$

of degree $n < m - 1$, using the least squares procedure is handled similarly. We choose the constants $a_0, a_1, \ldots, a_n$ to minimize the least squares error $E = E_2(a_0, a_1, \ldots, a_n)$, where

$$\begin{aligned} E &= \sum_{i=1}^{m} (y_i - P_n(x_i))^2 \\ &= \sum_{i=1}^{m} y_i^2 - 2\sum_{i=1}^{m} P_n(x_i)y_i + \sum_{i=1}^{m} (P_n(x_i))^2 \end{aligned}$$

;

$$\begin{aligned} &= \sum_{i=1}^{m} y_i^2 - 2\sum_{i=1}^{m} \left(\sum_{j=0}^{n} a_j x_i^j \right) y_i + \sum_{i=1}^{m} \left(\sum_{j=0}^{n} a_j x_i^j \right)^2 \\ &= \sum_{i=1}^{m} y_i^2 - 2\sum_{j=0}^{n} a_j \left(\sum_{i=1}^{m} y_i x_i^j \right) + \sum_{j=0}^{n} \sum_{k=0}^{n} a_j a_k \left(\sum_{i=1}^{m} x_i^{j+k} \right). \end{aligned}$$

As in the linear case, for E to be minimized it is necessary that $\partial E/\partial a_j = 0$, for each $j = 0, 1, \ldots, n$. Thus, for each j, we must have

$$0 = \frac{\partial E}{\partial a_j} = -2\sum_{i=1}^{m} y_i x_i^j + 2\sum_{k=0}^{n} a_k \sum_{i=1}^{m} x_i^{j+k}.$$

This gives $n + 1$ **normal equations** in the $n + 1$ unknowns a_j. These are

$$\sum_{k=0}^{n} a_k \sum_{i=1}^{m} x_i^{j+k} = \sum_{i=1}^{m} y_i x_i^j, \quad \text{for each } j = 0, 1, \ldots, n. \tag{8.3}$$

It is helpful to write the equations as follows:

$$\begin{aligned} a_0 \sum_{i=1}^{m} x_i^0 + a_1 \sum_{i=1}^{m} x_i^1 + a_2 \sum_{i=1}^{m} x_i^2 + \cdots + a_n \sum_{i=1}^{m} x_i^n &= \sum_{i=1}^{m} y_i x_i^0, \\ a_0 \sum_{i=1}^{m} x_i^1 + a_1 \sum_{i=1}^{m} x_i^2 + a_2 \sum_{i=1}^{m} x_i^3 + \cdots + a_n \sum_{i=1}^{m} x_i^{n+1} &= \sum_{i=1}^{m} y_i x_i^1, \\ &\vdots \\ a_0 \sum_{i=1}^{m} x_i^n + a_1 \sum_{i=1}^{m} x_i^{n+1} + a_2 \sum_{i=1}^{m} x_i^{n+2} + \cdots + a_n \sum_{i=1}^{m} x_i^{2n} &= \sum_{i=1}^{m} y_i x_i^n. \end{aligned}$$

These *normal equations* have a unique solution provided that the x_i are distinct (see Exercise 14).

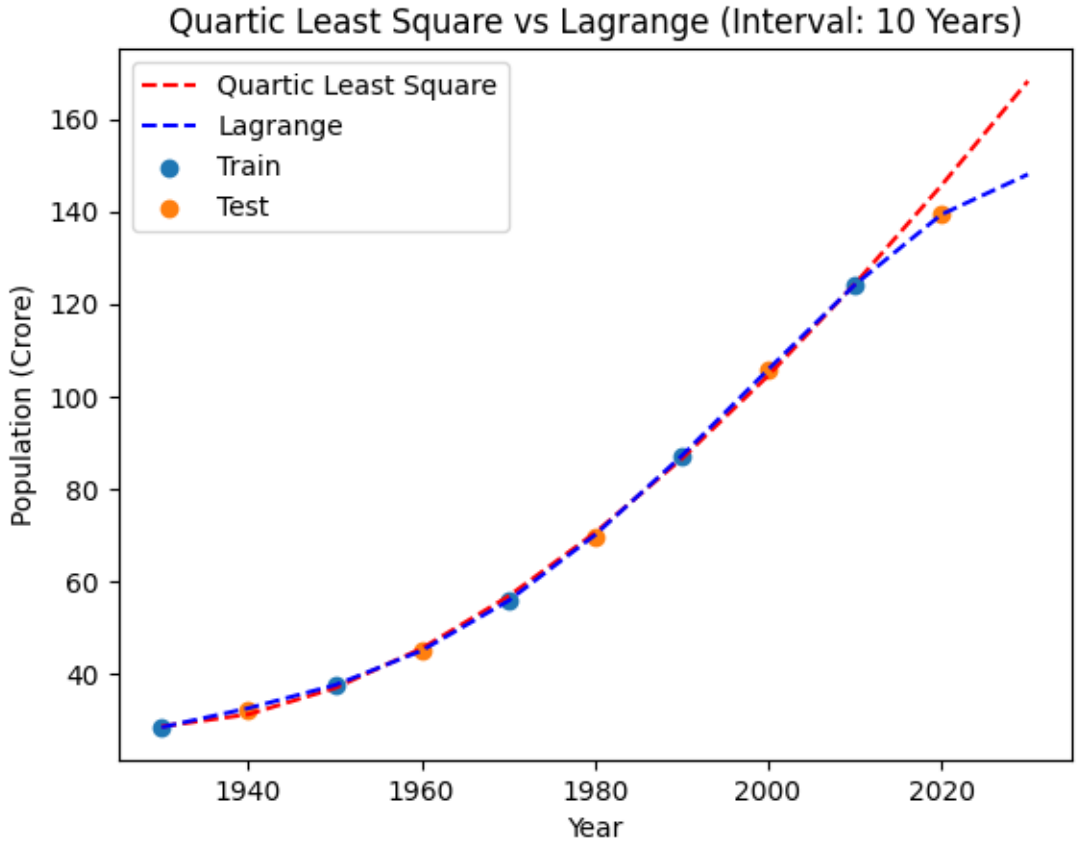
We now create a matrix $\mathbf{A}$ following the footsteps, whose entries are $a_{ij} = \sum_{k=1}^{m} x_k^{i+j}$, where $0 \leq i, j \leq n$ (indexing starts from $(0, 0)$.).

The column matrix $\mathbf{b}$ can be created with entries $b_i = \sum_{k=1}^{m} y_k x_k^i$, with $0 \leq i \leq n$. So, we get $\mathbf{A}$ with dimension $(n + 1) \times (n + 1)$ and $\mathbf{b}$ with dimension $(n + 1) \times 1$. With the entries x_i being unique, we can get a solution to the linear system of equations, which is then solved by the inbuilt numpy function in python. The solution to this system

of equations give us the necessary coefficients of the required least square fit of an nth degree polynomial.

Observation: Directly we can observe that the Lagrange Interpolation gives a better prediction for the population growth, as it can be seen from the plot and the RMSE measure.

However, there are only a few data points to infer from and moreover, alternate years are being taken. So, there might be a case of "overfitting" while using the Lagrange Interpolation. Another aspect that separates these two methods is that: Interpolating functions must pass through the data points, while the least square fit do not necessarily need to do so. For a large set of data, interpolation would not generally indicate the dominating trend of the information, whereas a good least square model does that in general (for these kinds of problems). Since we're working with a small amount of information and in case of the prediction at point *Year*= 2020, there was no significant indication about that "dip" in the increment of the population prior to it (upto 2010), so we can make an educated "guess" that degree 4 least square fit will not be able to capture it as good as the Lagrange Interpolation, but it should capture more or less the same amount as the cubic (there's an issue with the RMSE outcome mentioned later. The least square method takes into consideration the prior trend, and according to it, and from pure observation, the increment in the population should have been on the curve of the least square fit). The following plot and RMSE confirm our suspicion, that the RMSE of both least square fits should be close enough. And Lagrange Interpolation predicting correctly at 2020 without any "prior" hint as mentioned earlier, suggests that indeed it might just be by luck or due to overfitting.



Degree 4 RMSE: 2.801914334271028

$-5.01780 \times 10^{-8}x^4 + 0.00036x^3 - 0.94566x^2 + 1072.52885x - 437933.23090$
Note: In the code provided in Teams ("A4.ipynb"), the code is as follows:

*A[i] = [sum([x**(2*(m-1)-i-j) for x in xfit]) for j in range(m)]*
*b[i] = sum(yfit*np.array(xfit)**(m-1-i))*

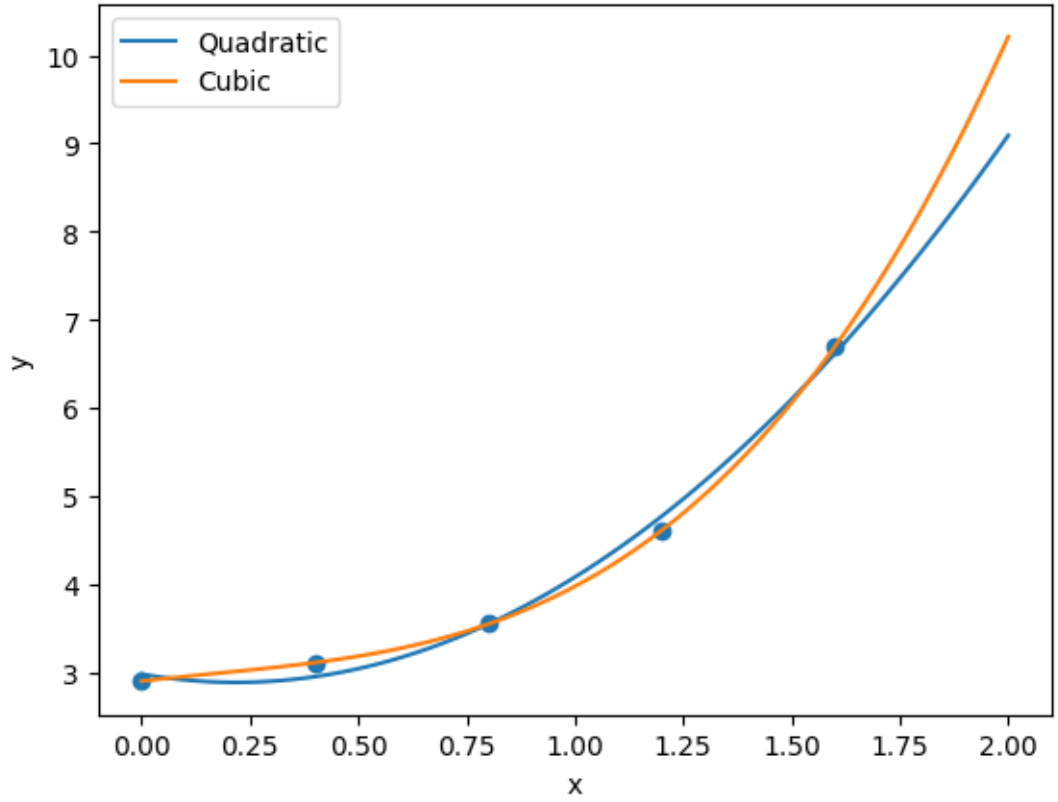
Error for deg 4: 3.07333 as per "A4.ipynb" by changing $m = 4$ to $m = 5$ in code cell In[4].
The index runs backwards as opposed to the code written I wrote as a solution to this problem. Hence, some differences indeed arise in the coefficients of the least square fit of degree 4, although in case of cubic fit, the coefficients remain the same. This might be occuring due to round off errors/overflow etc. When I modified my code to have the index run backwards, same set of coefficients were produced as the snippet provided here.

Question 2:

a)

From the previously written algorithm of obtaining the solution to the set of normal equations, we can figure out what will the coefficients for the quadratic and cubic least square solutions to the provided data.

The Least Square Fits:
Quadratic: $1.9554x^2 - 0.8536x + 2.9777$
Cubic: $1.0417x^3 - 0.5446x^2 + 0.5798x + 2.8977$
RMSE for quadratic: 0.11346
RMSE for cubic: 0.0085523



Visually we can also observe that the cubic fits the data better.

b)

To find the least square solution to the given equation, we take the error function as:

$$E(a,b) = \sum_{i=1}^{m} (ae^{-3x_i} + be^{-2x_i} - y_i)^2$$

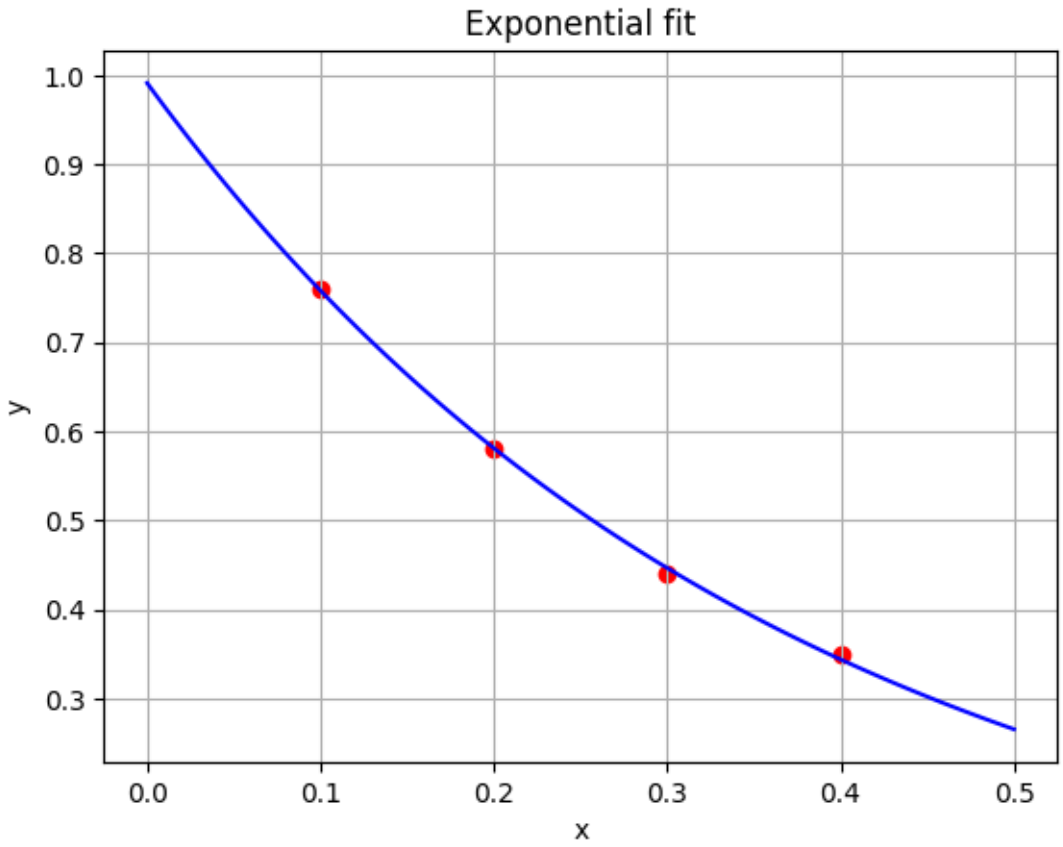
$$\implies \frac{\partial E}{\partial a} = 2\sum_{i=1}^{m} (ae^{-3x_i} + be^{-2x_i} - y_i)e^{-3x_i} = 0 \text{ and}$$

$\frac{\partial E}{\partial b} = 2\sum_{i=1}^{m} (ae^{-3x_i} + be^{-2x_i} - y_i)e^{-2x_i} = 0$ for the solution to be least square. So, we get the normal equations as:

$$a\sum e^{-6x_i} + b\sum e^{-5x_i} = \sum y_i e^{-3x_i}$$

$$a\sum e^{-5x_i} + b\sum e^{-4x_i} = \sum y_i e^{-2x_i}$$

Solving them, we obtain: **a=0.68532816, b=0.30584456**.



c)

Formulae used:
1. First Derivatives:

Three point endpoint: $\dfrac{-3f(x)+4f(x+h)-f(x+2h)}{2h}$

Three point midpoint: $\dfrac{f(x+h)-f(x-h)}{2h}$

Five point midpoint: $\dfrac{-f(x+2h)+8f(x+h)-8f(x-h)+f(x-2h)}{12h}$

2. Second Derivatives:

Three point midpoint: $\dfrac{f(x+h)-2f(x)+f(x-h)}{h^2}$

Five point midpoint: $\dfrac{-f(x+2h)+16f(x+h)-30f(x)+16f(x-h)-f(x-2h)}{12h^2}$

3. Third Derivatives:

Five point midpoint: $\dfrac{f(x+2h)-2f(x+h)+2f(x-h)-f(x-2h)}{2h^3}$

Index	Order	Method	h	Value	Absolute error	3 or 5 point
0	First derivative	Endpoint	0.1	22.03231	0.134	3 points
1	First derivative	Endpoint	-0.1	22.05453	0.113	3 points
2	First derivative	Midpoint	0.1	22.22879	0.062	3 points
3	First derivative	Midpoint	0.2	22.41416	0.247	3 points
4	First derivative	Midpoint	-0.1	22.22879	0.062	3 points
5	First derivative	Midpoint	0.1	22.167	0.00017	5 points
6	First derivative	Midpoint	-0.1	22.167	0.00017	5 points
7	Second derivative	Midpoint	0.1	29.5932	0.0370	3 points
8	Second derivative	Midpoint	0.2	29.70427	0.148	3 points
9	Second derivative	Midpoint	-0.1	29.5932	0.0370	3 points
10	Second derivative	Midpoint	0.1	29.55617	0.00005	5 points
11	Second derivative	Midpoint	-0.1	29.55617	0.00005	5 points
12	Third derivative	Midpoint	0.1	37.0745	0.129	5 points
13	Third derivative	Midpoint	-0.1	37.0745	0.129	5 points

As it can be seen, smaller values of h provide a better approximation. For midpoint formulae, $h = \pm 0.1$ does not alter the value, due to the symmetry of the numerator and the denominator (about the point x, at which the derivative is to be computed).
Five point endpoints formulae are not applicable (even if they exist) since it'll take values from out of the range of the provided dataset.

Question 3:

a)

n= 2
Composite Trapezoidal Rule: 1.5707963267948966
Composite Simpson's Rule: 2.0943951023931953

n= 4
Composite Trapezoidal Rule: 1.5707963267948966
Composite Simpson's Rule: 1.5707963267948966

n= 8
Composite Trapezoidal Rule: 1.5707963267948966
Composite Simpson's Rule: 1.5707963267948966

n= 16
Composite Trapezoidal Rule: 1.5707963267948966
Composite Simpson's Rule: 1.5707963267948968

Romberg Results (Full matrices are given in the code file):
n= 2 : 2.0943951024
n= 4 : 1.5713503996
n= 8 : 1.5707963268
n= 16 : 1.5707963268

b)

In the program, a list containing the first 8 Legendre polynomials was taken. The interval $[-1, 1]$ was broken into 100 equal subintervals and the endpoints was scanned for sign change. The subintervals containing the roots of a polynomial (out of those 8) was taken as initial subintervals and bisection method was applied on those with criteria $|P(x)| \leq 10^{-10}$. Thus, we can find out the quadrature points (which are the roots of the nth degree Legendre polynomial). For figuring out the corresponding weights, composite simpson rule was applied on $\prod_{j=1, j\neq i}^{n} \frac{(x - x_i)}{(x_i - x_j)}$ in the interval $[-1, 1]$ with step size $h = 2/10000$.

Gaussian Quadrature with n=2 on different intervals.
$[0, 0.5]$: 0.060770207464005736
$[-0.5, 0]$: 0.028718288078954376
$[0.5, 1]$: 0.6570944696079725
$[-1, -0.5]$: 0.13180483099411827

Summing up, $\int_{-1}^{1} x^2 e^x dx$= 0.8783877961450508

Gaussian Quadrature with n=4 on different intervals.
$[0, 1]$: 0.7182817683085485
$[-1, 0]$: 0.16060277751465887

Summing up the result on the two intervals, we get the result: $\int_{-1}^{1} x^2 e^x dx = 0.8788845458232074$

Gaussian Quadrature with n=8 on $[-1, 1]$:
$\int_{-1}^{1} x^2 e^x dx$ =0.8788846226018331

True Value: $e - \frac{5}{e}$ **= 0.8788846226018334**
Error with n=2: 0.0004968264567826175
Error with n=4: 7.677862601251917e-08
Error with n=8: 3.3306690738754696e-16

Reasoning: With $n = 8$, the Gaussian quadrature being applied on to the entire interval $[-1, 1]$ produces a much better result than the other two. Because in the case of $n = 4$, we were essentially turning the intervals $[0, 1]$ and $[-1, 0]$ into $[-1, 1]$ via scaling and shifting the endpoints by change in variables and applying the quadrature with 4 points to the function to evaluate its integral on those respective intervals (while applying the formula in $[-1, 1]$) and summing them up. The summation does not eliminate the error arising from using a low number of quadrature points, rather the error gets increased. So, naturally the accuracy is significantly lower than $n = 8$. Same resoning being applied on $n = 2$, we can see why the outcome is so much worse even in comparison with $n = 4$. We can see that the errors are in respective orders: $10^{-4}, 10^{-8}, 10^{-16}$.