Numerical Algorithm Assignment 2

Zhiwei Cao, SCSE, G2100614H

Abstract

This paper provides the details for three different techniques that are applied in assignment 2 for curve interpolation(fitting purpose as well as the numerical integration based on composite Simpson's rule. We first present the technique based on polynomial least square and discuss the curve fitting result against the number of sample data points as well as the degree of the polynomial. After that, we discuss the curve fitting result based on the cubic B-spline interpolation that has been discussed intensively in assignment 1 and compare the curve fitting accuracy with different number of sample data points. In what follows, we present the curve fitting approach based on trigonometric interpolation and discuss the impacts of different number of basis functions applied in the interpolation. Finally, we demonstrate the composite Simpson's rule in one-dimensional numerical integration and compare its performance with the composite trapezoid's rule, showing its superiority over the counterpart method.

I. Task I: Curve Visualisation

In this task, we are required to plot the curve defined with the parametric curve as following:

$$\begin{cases} x(u) = 1.5\left[e^{1.5\sin(6.2u-0.027h)}+0.1\right]\cos(12.2u), & u\in[0,1] \\ y(u) = \left[e^{\sin(6.2u-0.027h)}+0.1\right]\sin(12.2u), & u\in[0,1] \end{cases} \tag{1}$$

The plotting result is illustrated in Fig. 1.

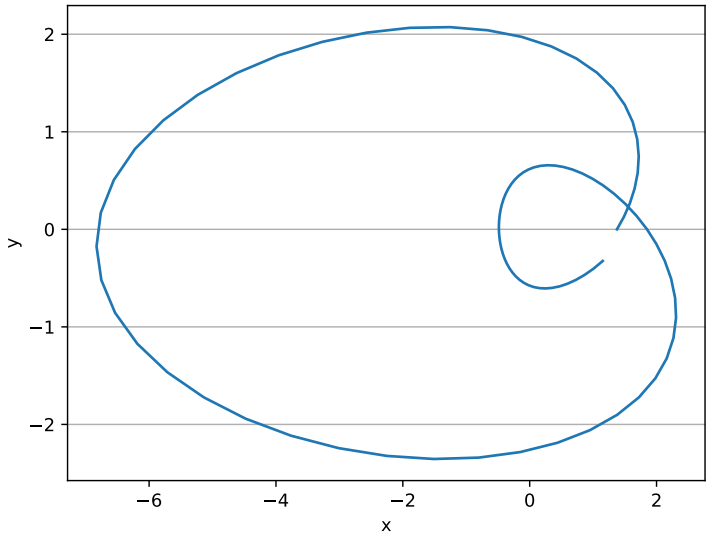

Fig. 1: The curve parametrized by Eq. (1)

.

II. Task II: Least Squares Fitting

In this section, we are required to propose a least-squares method to fit the curve specified in Eq. (1) with a parametric cubic polynomial curve. To achieve that, we propose to fit $x(u)$ and $y(u)$ in Eq. (1) separately. To be specific, we first uniformly sample N different u within $[0,1]$, starting from 0 and ending with 1.0. After the sampling, we obtain sample vector$\mathbf{u} = [u_1, u_2, ..., u_N] \in \mathbb{R}^N$. Let the parametric cubic polynomial function be $f(u) = au^3 + bu^2 + c^u + d$ with four degree of freedom. The goal for the least-square fitting is to find the optimal value for the four parameters a, b, c, d so that the squared error between the function value of the original curve and the cubic polynomial curve can be minimized in the sampled data points. Without loss of generality, we take the curve fitting for $x(u)$ as an example to demonstrate the whole process and the

same procedure is applicable for $y(u)$. With the sampled vector $\mathbf{u}$, one can establish 50 equations in the sampled data points as follows:

$$\begin{aligned} au_1^3 + bu_1^2 + cu_1 + d &= x(u_1) \\ au_2^3 + bu_2^2 + cu_2 + d &= x(u_2) \\ au_3^3 + bu_3^2 + cu_3 + d &= x(u_3) \\ &... \\ au_N^3 + bu_N^2 + cu_N + d &= x(u_N) \end{aligned} \tag{2}$$

Note that the system is still linear w.r.t the four parameters a, b, c, d even the parametric curve is nonlinear. Let the vector $\mathbf{p} = [a, b, c, d]^T \in \mathbb{R}^4$ and the matrix $A = [\mathbf{u}_3, \mathbf{u}_2, \mathbf{u}_1, \mathbf{u}_0] \in \mathbb{R}^{N\times 4}$ where $\mathbf{u}_i$ denotes the vector $[u_1^i, ..., u_N^i]^T \in \mathbb{R}^N$. Denote the RHS of the linear system (2) as $\mathbf{x} = [x(u_1), x(u_2), ..., x(u_N)] \in \mathbb{R}^N$. With the notations, the linear system (2) can be written as:

$$\mathbf{Ap} = \mathbf{x}. \tag{3}$$

Noted that the number of equations N is larger than the number of free parameters, the linear system (3) is over-determined, meaning that there probably does not exist an exact solution that can satisfy all equations simultaneously. Hence, we resort to the optimal solution in the sense of least squares. Formally, we solve the following minimization problem to obtain the optimal parameter vector $\mathbf{p}^*$:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \quad ||\mathbf{Ap} - \mathbf{x}| |_2^2. \tag{4}$$

It is easy to verify that the optimization problem (4) is a unconstrained convex optimization which can be solved by enfocing the gradient of the objective function w.r.t. $\mathbf{p}$ to be zero:

$$\nabla_{\mathbf{p}} ||\mathbf{Ap} - \mathbf{x}| |_2^2 = \mathbf{A}^T(\mathbf{Ap} - \mathbf{x}) = \mathbf{0} \tag{5}$$

By rearranging Eq. (5), the optimal parameter vector $\mathbf{p}^*$ can obtained by solving the following linear system:

$$\mathbf{A}^T\mathbf{Ap} = \mathbf{A}^T\mathbf{x}. \tag{6}$$

Theoretically, it is possible to obtain the closed-form solution for $\mathbf{p}^*$ as $\mathbf{p}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{Ax}$. However, in reality, the matrix $\mathbf{A}^T\mathbf{A}$ might have large condition number, or in other word, be ill-posed, making the matrix inverse numerical unstable. Therefore, in practical, we adopt Gaussian Elimination to solve the linear system (6) and obtain the optimal parameter vector $\mathbf{p}^*$. We apply the same process to both $x(u)$ and $y(u)$ and finally obtain two least-squares fitted cubic polynomial curve $\hat{x}(u)$ and $\hat{y}(u)$. In the curve plotting phase, we evaluate $\hat{x}(u)$ and $\hat{y}(u)$ at different input u to obtain the approximate curve.

In the following, we demonstrate the curve fitting results with the cubic polynomial least square. In the experiment, we set the number of sample points $N = 50$ and the fitted curve is shown in Fig. (2). It can be seen from Fig. (2) that the fitted

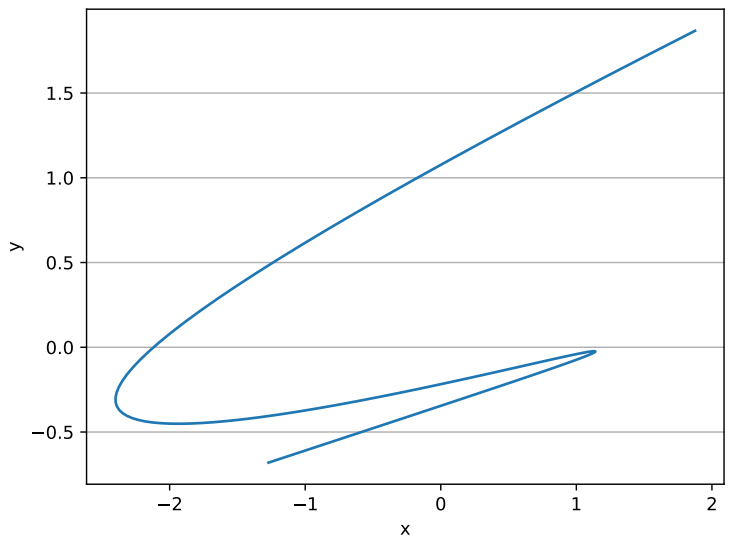

Fig. 2: The least-sauares fitted cubic polynomial curve with 50 sample points.

.

curve does not respect the general shape of the original curve shown in Fig. (1). To further analysis the reason for it, we

conjecture that such a phenomenon is because the cubic polynomial cannot approximate the original function $x(u)$ and $y(u)$ very well. To validate it, we conduct some experiments with $N = 100$ while varying the degree of polynomials. For higher degree polynomials, the procedure described previously can be extended with minor modification, and we do not present the detailed process again here. Fig (3) shows the evaluation results for various degree polynomials. From Fig. 3, it can be clearly

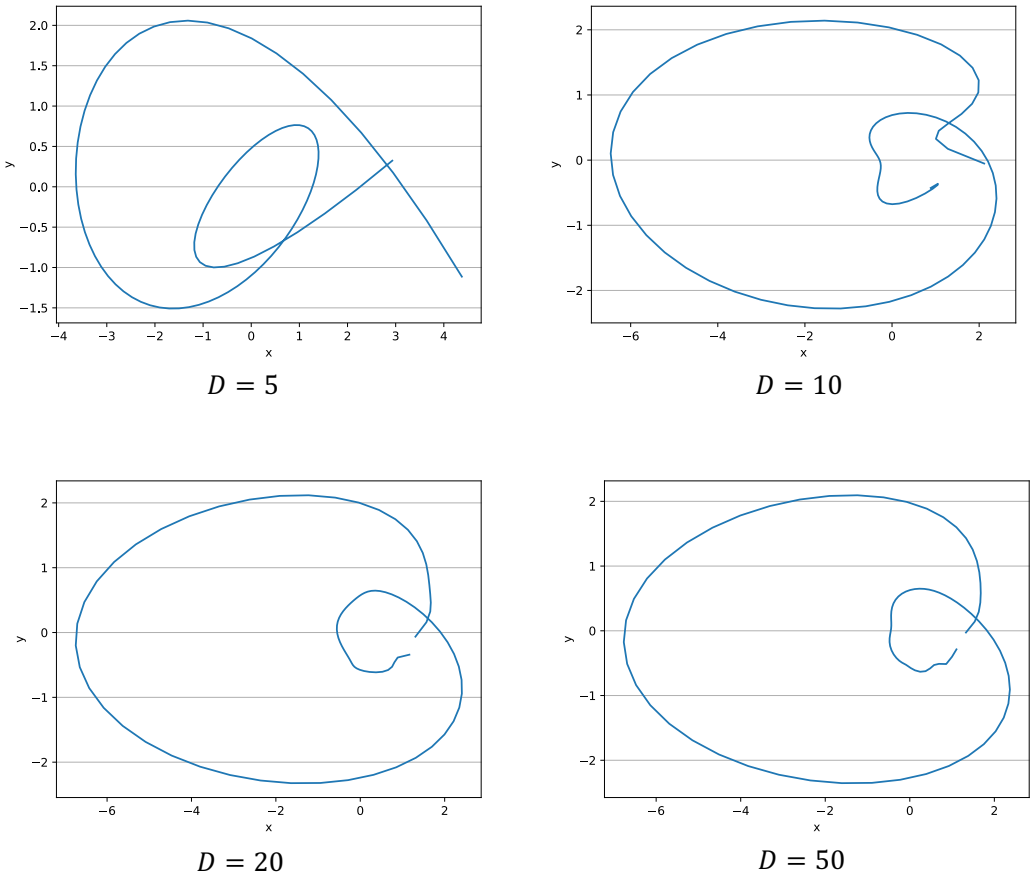

Fig. 3: The least-squares fitted cubic polynomial curve with different degree polynomial functions $(D = 5, 10, 20, 50)$ and fixed $N = 100$.

seen that with the increasing degree of polynomial functions, the curve fitting quality gets improved consistently. Hence, we conclude that the cubic pronominal function is not capable of approximating the complicated curve as shown in Fig. (1) and higher degree of polynomial function should be considered when approximating the original curve.

III. Task III: Cubic B-spline Interpolation Based Curve Fitting

In this section, we are required to use the cubic B-spline interpolation technique developed in Assignment 1 to find a cubic B-spline curve that approximates the original curve. To achieve that, similar to what we've done in Section II. we first uniformly sampled data points in the original curve shown in Fig. (1). Following that, we treat them as the data points that the final cubic B-spline curve will interpolate and leverage the program which we have developed for Assignment 1 to find the deBoor points and the knot vector. With these at hand, we evaluate cubic B-spline curve and visualize it as shown in Fig. (**??**). To gain insights on the impacts of different number of data points on the final fitted curve quality, we vary the number of sampled data points and the obtained cubic B-spline curves are illustrated in Fig. (4). It can be seen in Fig. 4 that with the increasing number of data points, the curve fitting quality improves monotonically and 50 uniformly sample data points are capable of obtaining a cubic B-spline curve with reasonable quality. Compared Fig. (4) and Fig. (2), it is obvious that the cubic B-spline curve outperform the cubic polynomial function significantly despite the fact that both of them are based on cubic polynomial functions. The reasons for this strike fitting quality difference are two fold:

- Cubic B-spline interpolation uses piecewise cubic function to approximate the original curve while the cubic polynomial least squares with the cubic function to approximate the curve globally. Since the parametric functions defined in Eq. (1) are very complicated, it is unlikely to use a cubic function to approximate them with satisfactory quality. However, the cubic B-spline interpolation bypasses it by first fitting a set of cubic polynomial functions locally and finally assembling

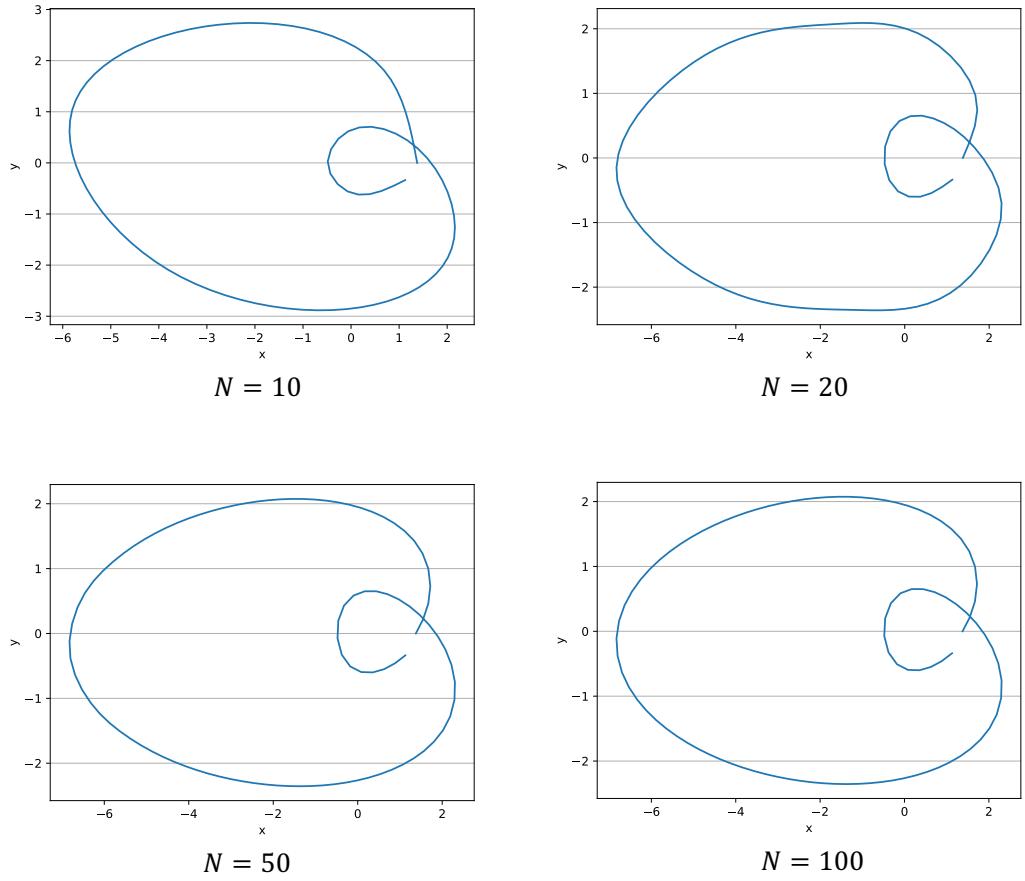

Fig. 4: The fitted cubic B-spline curve with different number of data points ($N = 10, 20, 50, 100$).

them into a complete curve through the interpolation equations. Even though the perfect global approximation with cubic polynomial is not feasible, it is also possible to use cubic polynomial for the local regions of the original function, and thus, the cubic B-spline interpolation has much better quality compared with cubic polynomial least-squares.

- Cubic B-spline uses the Bernstein polynomials as basis functions, which has some nice properties, while the cubic polynomial least squares uses common power basis functions.

In conclusion, we observe that the cubic B-spline interpolation provide much better curve fitting quality compared with the cubic polynomial least squares. We also find that one can achieve similar approximation performance with cubic B-spline interpolation in comparison to the high-degree polynomial least square (see Fig. (4) and Fig. (3)), showing the superiority of ensemble of local approximation over global approximation.

IV. Trigonometric Interpolation with Discrete Fourier Transform

In this section, we are required to use the trigonometric function to interpolate a given set of data points $x_0, x_1, ..., x_{N-1}$. The basic idea is to find proper weights for a set of trigonometric functions which are pair-wise orthogonal, so that the linear combination of these trigonometric functions will interpolate the given data points. To achieve that, one can follow the step approach: 1) computing the Discrete Fourier Transform (DFT) of the given data points, yielding N Fourier coefficients $y_0, y_1, ..., y_{N-1}$; 2) using the Fourier coefficients as the weight for the trigonometric functions to obtain the interpolation function. In the rest of this section, I will present how to compute the DFT of a given data sequence, and how to use the DFT result to form the interpolation function.

A. Discrete Fourier Transform (DFT)

Let $\mathbf{x} = [x_0, x_1, ..., x_{N-1}]$ be the data sequence where we want to perform DFT, and $\omega = e^{j\frac{2\pi}{N}}$, the DFT of $\mathbf{x}$ can formulated as:

$$y_j = \frac{1}{\sqrt{N}} \sum_{k=1}^{N-1} x_k \omega^{-jk}, \quad j = 0, 1, ..., N-1. \tag{7}$$

where y_j is the j-th element of the DFT result. It is important to note that y_j will be a *complex number* and using the fact that $e^{j\theta} = \cos(\theta) + j\sin(\theta)$, one can immediately derive that any y_j will be the linear combination of a set of trigonometric function weighted by $\mathbf{x}$.

Similarly, using the orthogonality of trigonometric functions, we can define the Inverse Discrete Fourier Transform (IDFT) as:

$$x_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} y_j \omega^{jk}. \tag{8}$$

where y_j is the j-th element of the DFT result of the data sequence $\mathbf{x}$.

B. Trigonometric Interpolation

Given N data points $\mathbf{x} = [x_0, ..., x_{N-1}]$, let $\Delta\theta = \frac{2\pi}{N}$ and $\theta_j = j\Delta\theta$ for $j = 0, 1, ..., N-1$, we want to find a function $P_n(\theta)$ which is the linear combination of N trigonometric basis functions so that:

$$P_n(\theta_j) = x_j, \quad j = 0, 1, ..., N-1 \tag{9}$$

Let the DFT of $\mathbf{x}$ be $\mathbf{y} = \mathcal{F}(\mathbf{x})$, we construct the following complex function:

$$\begin{aligned} Q(\theta) &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} y_k e^{jk\theta} \\ &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} (a_k + jb_k) \left[\cos(jk\theta) + j\sin(jk\theta)\right] \\ &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} [a_k \cos(k\theta) - b_k \sin(k\theta)] + j \cdot [b_k \cos(k\theta) + a_k \sin(k\theta)]. \end{aligned} \tag{10}$$

Using the definition of IDFT, we know the fact that:

$$Q(\theta_j) = Q(j \cdot \Delta\theta) = x_j, \quad j = 0, 1, ..., N-1 \tag{11}$$

Therefore, $Q(\theta)$ can serve as our interpolation function which is a linear combination of a set of trigonometric basis functions. From Eq. (11), one can observe that the DFT results of $\mathbf{x}$ serve as the coefficients to weight different trigonometric functions.

In the following, we use some nice properties of DFT to simplify the trigonometric interpolation function (11). If $\mathbf{x} \in \mathbb{R}^N$ (most common case in practice), denote the real part of $Q(\theta)$ as $P(\theta) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} [a_k \cos(k\theta) - b_k \sin(k\theta)]$, we can easily verify that $P(\theta_j) = x_j$ holds. Hence, we simplify the original interpolation function $Q(\theta)$ by only using its real part $P(\theta)$ as our interpolation function. We now proceed our simplification by discussing two cases for N.

- If N is even, note that for $j = 0, 1, ..., N/2 - 1$, the following equations hold:

$$\begin{aligned} \cos(k\theta_j) &= \cos[(N-k)\theta_j] \\ \sin(k\theta_j) &= -\sin[(N-k)\theta_j] \end{aligned} \tag{12}$$

  Therefore, we can further simplify $P(\theta)$ as:

$$P_n(\theta) = \frac{a_0}{\sqrt{N}} + \frac{1}{\sqrt{N}} \sum_{k=1}^{N/2-1} [(a_k + a_{N-k}) \cos(k\theta) + (b_k - b_{N-k}) \sin(k\theta)] + \frac{a_{N/2}}{\sqrt{N}} \cos(\frac{N}{2}\theta). \tag{13}$$

- If N is odd, similarly we can derive the simplified version of $P(\theta)$ as:

$$P_n(\theta) = \frac{a_0}{\sqrt{N}} + \frac{1}{\sqrt{N}} \sum_{k=1}^{(N-1)/2} [(a_k + a_{N-k}) \cos(k\theta) + (b_k - b_{N-k}) \sin(k\theta)] \tag{14}$$

Furthermore, since $\mathbf{x}$ is a real-numbered vector, according to the symmetric property of DFT, for $j = 0, 1, ..., N-1$, the following equities hold:

$$\begin{aligned} a_k &= a_{N-k} \\ b_k &= -b_{N-k} \end{aligned} \tag{15}$$

Therefore, if N is oven, we can further simplify $P_n(\theta)$ as:

$$P_n(\theta) = \frac{a_0}{\sqrt{N}} + \frac{2}{\sqrt{N}} \sum_{k=1}^{N/2-1} [a_k \cos(k\theta) + b_k \sin(k\theta)] + \frac{a_{N/2}}{\sqrt{N}} \cos(\frac{N}{2}\theta). \tag{16}$$

Likewise, if N is odd, the simplified interpolation function is:

$$P_n(\theta) = \frac{a_0}{\sqrt{N}} + \frac{2}{\sqrt{N}} \sum_{k=1}^{(N-1)/2} [a_k \cos(k\theta) + b_k \sin(k\theta)]. \tag{17}$$

Noted that in Eq. (16) or (17), the range for θ is $[0, 2\pi]$, making it not difficult to be applied in practice. This issue can be addressed by variable replacement. Given an interval $[c, d]$, and let $t_j = c + j\frac{(d-c)}{N}$, the interpolation function can be obtained for even number N:

$$P_n(t) = \frac{a_0}{\sqrt{N}} + \frac{2}{\sqrt{N}} \sum_{k=1}^{N/2-1} a_k \cos\left[\frac{2\pi(t-c)}{d-c}\right] + b_k \sin\left[\frac{2\pi(t-c)}{d-c}\right] + \frac{a_{N/2}}{\sqrt{N}} \cos\left[\frac{\pi(t-c)}{d-c}\right] \tag{18}$$

Similarly, for odd number N, the interpolation function can be derived as:

$$P_n(t) = \frac{a_0}{\sqrt{N}} + \frac{2}{\sqrt{N}} \sum_{k=1}^{(N-1)/2} a_k \cos\left[\frac{2\pi(t-c)}{d-c}\right] + b_k \sin\left[\frac{2\pi(t-c)}{d-c}\right] \tag{19}$$

The interpolation results with the first 8, 16, 32 basis functions are presented in Fig. 5. We can clearly observe that with 16

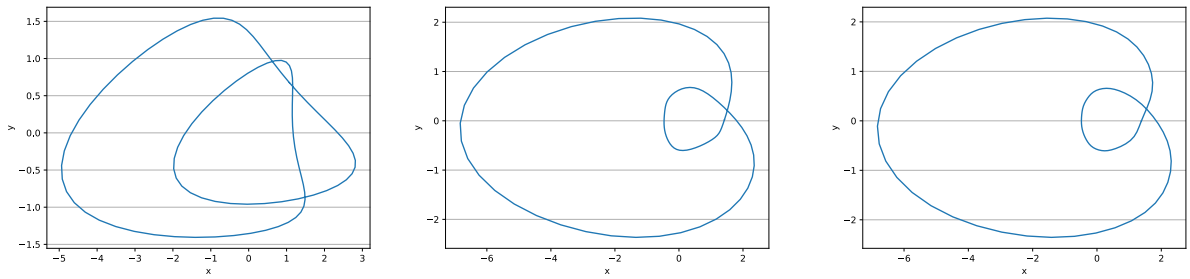

Fig. 5: Trigonometric interpolation result with 8, 16, 32 basis functions. By using more basis functions, we can achieve better curve approximation.

basis function, one can approximate the original curve very well. Compared with cubic B-spline interpolation or polynomial least square approach, the trigonometric interpolation yields best performance with least sample points. The reason is that the parametric function exhibits periodic property, which is quit suitable for using trigonometric functions as basis functions to perform interpolation.

V. Task V: Composite Simpson's rule for Integration

In this section, we are required to compute the following integration:

$$\int_0^1 1.5 \left[e^{1.5\sin(6.2u - 0.027h)} + 0.1\right] \cos(12.2u) \tag{20}$$

using composite Simpson's rule. We first place evenly spaced grid $0 = u_0 < u_1 < u_2 < ... < u_{2m} = 1$ with $u_i = ih, i = 0, 1, ..., 2m$ where $h = \frac{1}{2m}$. According to the Simpson's rule, the integration (20) can be approximated via:

$$\int_0^1 x(u)du \approx \frac{h}{3}\left[x(0) + x(1) + 4\sum_{i=1}^{m} x(u_{2i-1}) + 2\sum_{i=1}^{m} x(u_{2i})\right] \tag{21}$$

Given grid interval h, we can approximate the integration (20) with Eq. (6). According to 6, the approximation quality is highly related to the grid number, and larger grid number will give us more accurate approximation. Therefore, we conduct several experiments on different grid number to investigate the relationship between the approximation error and the number of grids, and the result is illustrated in Fig. (6). As we can see from Fig. (6), as the grid number increases, the integration computed via composite Simpson's rule becomes more accurate and the result converges if the grid number is larger than 5000.

Since the composite Simpson's rule is a special case of the closed Newton Cotes' rule with degree equals to 2, we further investigate the convergence speed of the composite trapezoid's rule, which is the degree 1 special case of the closed

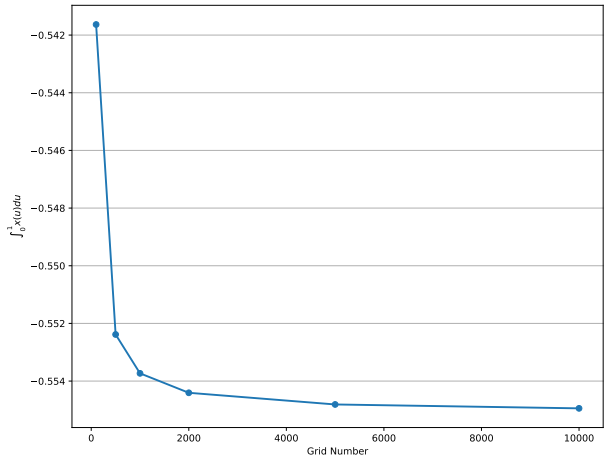

Fig. 6: Integration result for different grid number using composite Simpson's rule. More grids yield more accurate integration result.

Newton Cotes' rule, to demonstrate the superiority of the composite Simpson's rule. In particular, the integration (20) can be approximated via the composite trapezoid's rule as:

$$\int_0^1 x(u)du \approx \frac{h}{2}\left[x(0) + x(1) + 2\sum_{i=1}^{m-1} x(u_i)\right] \tag{22}$$

The comparison result for the composite trapezoid's rule and the composite Simpson's rule is illustrated in Fig. (7). It is

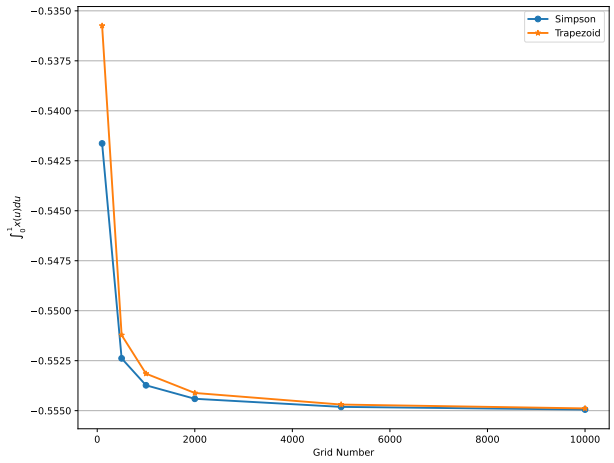

Fig. 7: Comparison between the the composite trapezoid's rule and the composite Simpson's rule. As a local degree 2 Lagrange polynomial approximation to the original function $x(u)$, the composite Simpson's rule has faster convergence speed compared to the composite trapezoid's rule, which is a local degree 1 Lagrange polynomial approximation to $x(u)$.

clear that the composite Simpson's rule has faster convergence speed compared to the composite trapezoid's rule. It is because the composite Simpson's rule utilizes the degree 2 Lagrange polynomial to approximate $x(u)$ locally while the composite trapezoid's rule uses linear function for local approximation of $x(u)$, leading to less accurate approximation since $x(u)$ is very complicated. However, for the same m, the composite Simpson's rule has to evaluate $x(u)$ for $2m+1$ times, while the composite trapezoid' rule needs only $m+1$ function evaluation. Therefore, the composite Simpson's rule trades the computational complexity with the accuracy.