Experiment Report

Computer-Aided Geometric Design

15 Liu Xing
PB22000150

September 23, 2025

Contents

Chapter 1

Introduction

1.1 Background

In Computer-Aided Geometric Design (CAGD), it is often necessary to smoothly connect a set of points. The method that passes through all the given points is called interpolation. In the two-dimensional case, common interpolation approaches include polynomial interpolation and radial basis function (RBF) interpolation, each with its own characteristics and applicability.

1.2 Objectives

In this experiment, two algorithms will be implemented. The input point set is determined by mouse clicks, and then interpolation curves are obtained and plotted using the two methods. Since there are many quantitative measures to evaluate the quality of interpolation functions, this experiment relies on visual inspection for comparison. In addition, numerical stability will be compared using extreme point data.

Chapter 2

Algorithm Description

2.1 Polynomial Interpolation

Polynomial interpolation involves finding a polynomial of degree $n-1$ that passes through n given points. The Lagrange interpolation formula is commonly used for this purpose.

Newton interpolation is a method for constructing polynomial interpolants. The key idea is to use **divided differences** to build the interpolation polynomial step by step.

Given nodes $\{(x_i, y_i)\}_{i=0}^{n-1}$, the Newton interpolation polynomial can be written as

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

where $f[x_i, \ldots, x_j]$ denotes the corresponding divided difference, which can be computed recursively as follows:

1. Zero-order divided differences (function values):

$$f[x_i] = y_i = f(x_i), \quad i = 0, 1, \ldots, n-1$$

2. First-order divided differences:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

3. Higher-order divided differences for $k \geq 2$:

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$$

The advantages of Newton interpolation are:

- The interpolation polynomial can be constructed recursively, and when a new node is added, only the new divided difference needs to be computed;
- Compared with directly solving the Vandermonde system, it provides better numerical stability.

2.2 Radial Basis Function (RBF) Interpolation

Radial Basis Function (RBF) interpolation is a method that uses radially symmetric functions to approximate data.

Given nodes $\{(x_i, y_i)\}_{i=1}^n$, the general form of the RBF interpolant is

$$s(x) = \sum_{i=0}^{n-1} b_i \, \phi(\|x - x_i\|) + p(x)$$

where:

- $\phi(r)$ is a radial basis function that depends only on the distance $r = \|x - x_i\|$, such as $\phi(r) = \frac{1}{r^2+d}$ or $\phi(r) = e^{-(\varepsilon r)^2}$;
- The coefficients b_i are determined by solving a linear system so that the interpolation conditions $s(x_i) = y_i$ are satisfied;
- $p(x)$ is a low-degree polynomial (or a constant term) used as a tail to ensure solvability and improve numerical stability.

RBF interpolation is advantageous in that it does not rely on mesh structures in higher dimensions and can handle scattered data effectively. However, its numerical stability is sensitive to the choice of kernel parameters (e.g., the shape parameter d or ε).

In this experiment, we chose $\phi(r) = \frac{1}{r^2+d}$ as the basis function, with d set as a fixed hyperparameter adjustable via a slider. We then set the tail as a constant. Since there are $n+1$ unknowns but only n equations, an additional constraint is required. Here, we impose $\sum_{i=0}^{n-1} b_i = 0$ as the constraint, which transforms the system into

$$\begin{bmatrix} A & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ c \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix}$$

with $A = (\phi(\|x_i - x_j\|))_{n \times n}$. For numerical stability, a term λI is added to A during the actual solution process. This form is convenient for solving because the augmented matrix on the left is symmetric, allowing the use of efficient algorithms for symmetric linear systems and ensuring numerical stability.

Chapter 3

Program Explanation

3.1 Programming Platform

This experimental code was written on MATLAB R2024b under the Windows 11 platform, but it is compatible with all MATLAB versions later than R2018b.

3.2 Program Structure

The program consists of a single file, which is divided into three sections using the "%%" delimiter.

The first section handles the UI plotting logic. Most of it is based on the provided code template, but a logarithmic-scale `slider` has been added, and the polygon drawing and event-listening logic has been updated to use `drawpolyline` and `addlistener` to leverage the newer MATLAB graphics engine and ensure longer-term compatibility.

The second section implements the polynomial interpolation function using the Newton interpolation method to improve numerical stability and computational efficiency.

The third section implements the RBF interpolation function, providing interfaces for the shape parameter d and the regularization parameter λ.

3.3 Program Usage

- Run the program in MATLAB. A window will pop up with a blank canvas.
- You can add points by simply clicking on the canvas. Double-click while adding the last point can complete the polygon and display the interpolation functions fitted by the two methods.
- Afterwards, you can drag any of the existing points at any time to change their positions.
- A slider for controlling the shape parameter d also appears at the bottom of the screen; dragging it allows you to adjust d in real time within the range $1 \times 10^{-3} \sim 1 \times 10^{1}$, and the interpolation plot updates immediately.

Chapter 4

Results

4.1 General Results

To compare the applicability of the two interpolation methods, I analyzed three scenarios: interpolation of nearly linear point sets, fitting of convex point sets, and fitting of oscillatory point sets.

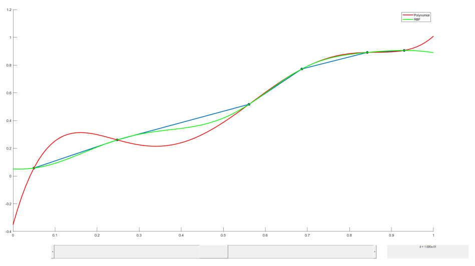
First, we compare the case of points close to a linear function. Both methods produce satisfactory results; however, polynomial interpolation may exhibit oscillations at the boundaries depending on the parity of the number of points, as shown in Figure 4.1a. Additionally, if several points are very close to each other—resulting in larger differences in pairwise slopes—both methods may exhibit oscillations, as illustrated in Figure 4.1b.

Next, we consider convex point sets. Here, ”convex” means that each subsequent point lies above the line connecting the previous two points, rather than the point set itself forming a convex hull. In this scenario, the theoretically expected interpolation should approximate a convex function. Tests show that polynomial interpolation performs better than RBF in this case, as seen in Figure 4.1c. However, increasing the RBF shape parameter d can also produce convex RBF interpolation results, as shown in Figure 4.1d.

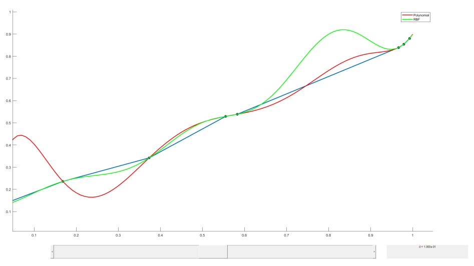
Finally, we examine oscillatory point sets. Polynomial interpolation exhibits a more pronounced Runge phenomenon, whereas RBF interpolation produces a smoother and more natural curve, as shown in Figure 4.1e. However, in this scenario, increasing the RBF shape parameter d can reduce numerical stability, preventing the interpolated function from passing through the given points, as illustrated in Figure 4.1f.

4.2 Extreme Data Cases

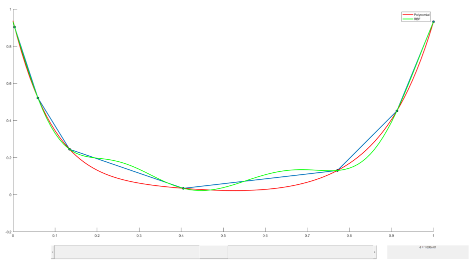
Under certain extreme data conditions, such as points with identical x-coordinates, polynomial interpolation may fail to produce results (due to division by zero), and RBF interpolation may not pass through all the given points, as shown in Figure 4.2. However, the program does not produce errors or crash.



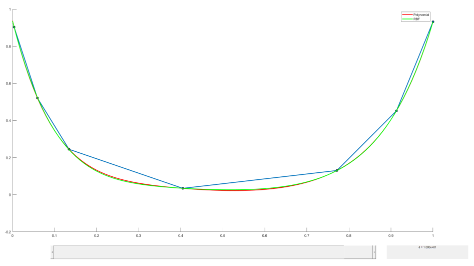
(a) Nearly linear points



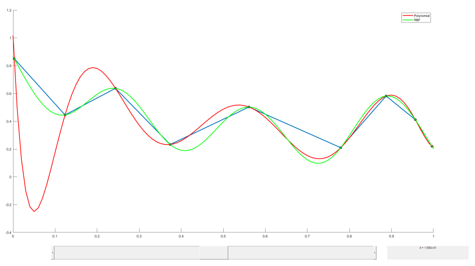
(b) Close points causing oscillation



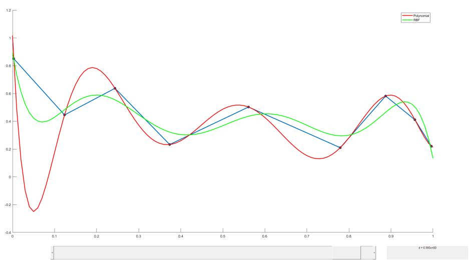
(c) Convex points



(d) RBF with large d, convex



(e) Oscillatory points



(f) RBF with large d, unstable

Figure 4.1: Comparison of interpolation methods under different point sets.

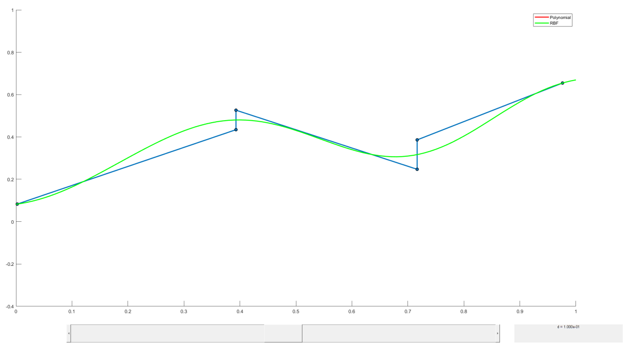


Figure 4.2: Extreme data cases where polynomial interpolation fails and RBF does not pass through all points.

Chapter 5

Discussion

In this experiment, we compared the applicability and performance of polynomial interpolation and radial basis function (RBF) interpolation by analyzing different types of point sets. The results indicate:

1. **Nearly linear points**: Both methods can fit well, but polynomial interpolation may exhibit slight oscillations at the boundaries, especially when the number of points is odd or the spacing between points is uneven. RBF interpolation produces smoother results overall.

2. **Convex points**: Polynomial interpolation naturally preserves convexity, whereas RBF interpolation may slightly violate convexity when the shape parameter is small. Increasing the parameter d can improve convexity.

3. **Oscillatory points**: Polynomial interpolation is prone to the Runge phenomenon, particularly with a larger number of unevenly spaced points. RBF interpolation is smoother, but overly large shape parameters may compromise numerical stability and prevent the interpolated function from passing through all points.

4. **Extreme data conditions**: For points with identical x-coordinates, polynomial interpolation fails due to division by zero, and RBF interpolation cannot pass through all points. However, the program remains stable and does not produce errors or crash.

Overall, polynomial interpolation is suitable for a small number of evenly distributed points, while RBF interpolation is better suited for non-uniform or complex point distributions, with adjustable parameters to achieve smooth and stable results.

Chapter 6

Questions and Analysis

As described in Section 2.2, for a constant tail term, only one additional unknown is introduced, so we can add the constraint $\sum_{i=0}^{n-1} b_i = 0$ to make the system a symmetric linear system. For higher-order tail terms, similar constraints can be added:

$$\sum_{j=0}^{n-1} b_j \, p_k(x_j) = 0, \quad k = 0, \ldots, m$$

This leads to a linear system with a block matrix form:

$$\begin{bmatrix} A & P \\ P^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}$$

which can then be solved accordingly.

Chapter 7

Conclusion

This experiment implemented an interactive visualization tool for both polynomial and RBF interpolation, and the comparison of different point sets leads to the following conclusions:

1. Polynomial interpolation performs well for nearly linear or convex point sets but may exhibit oscillations or numerical issues with oscillatory or extreme data.

2. RBF interpolation is generally smooth and can handle complex or non-uniform point sets, but proper selection of the shape parameter is required to ensure numerical stability.

3. The interactive tool allows users to add and move points in real time and adjust RBF parameters, facilitating observation of the strengths, weaknesses, and parameter sensitivity of both methods.

The experimental results provide an intuitive reference for understanding the applicability of different interpolation methods and offer a basis for visualizing and analyzing more complex data fitting in future studies.