Computional Homework4 Report

Student: 纪浩正, jihz2023@mail.sustech.edu.cn

1 Comparison of Bilinear Interpolation and Bicubic Spline for the 2D Gaussian Function $e^{-9(x^2+y^2)}$ on $[-1,1]\times[-1,1]$

1.1 Bilinear Interpolation

If we want to get the value at (x, y), we should find the indices i and j, such that $x_i < x < x_{i+1}$ and $y_j < y < y_{j+1}$.

For example, we can get the three points along the x axis. Consider the following:

- At (x_i, y):

$$A = (1 - \frac{y - y_j}{y_{j+1} - y_j})\, f(x_i, y_j) + \frac{y - y_j}{y_{j+1} - y_j}\, f(x_i, y_{j+1})$$

- At (x_{i+1}, y):

$$B = (1 - \frac{y - y_j}{y_{j+1} - y_j})\, f(x_{i+1}, y_j) + \frac{y - y_j}{y_{j+1} - y_j}\, f(x_{i+1}, y_{j+1})$$

Now, we use the two boundary points to interpolate the value at the middle point (x, y) along the x direction:

$$f(x, y) = (1 - \frac{x - x_i}{x_{i+1} - x_i})A + \frac{x - x_i}{x_{i+1} - x_i}B$$

Thus, we get the final interpolation formula:

$$\begin{aligned} f(x, y) = (1 - \frac{x - x_i}{x_{i+1} - x_i}) &\left[(1 - \frac{y - y_j}{y_{j+1} - y_j})\, f(x_i, y_j) + \frac{y - y_j}{y_{j+1} - y_j}\, f(x_i, y_{j+1})\right] \\ + \frac{x - x_i}{x_{i+1} - x_i} &\left[(1 - \frac{y - y_j}{y_{j+1} - y_j})\, f(x_{i+1}, y_j) + \frac{y - y_j}{y_{j+1} - y_j}\, f(x_{i+1}, y_{j+1})\right] \end{aligned}$$

This procedure can be described as first linearly interpolating along the y direction for both x_i and x_{i+1}, and then linearly interpolating in the x direction using the resulting values.

A Python implementation for bilinear interpolation is shown below:

```
class Bilinear_Interpolation:
    def __init__(self, xi, yi, zi):
        self.xi = xi
```

```
        self.yi = yi
        self.zi = zi
        self.Nx = len(xi)
        self.Ny = len(yi)

    def bi_inter(self, x, y):
        x_idx = np.clip(np.searchsorted(self.xi, x) - 1, 0, self.Nx - 2)
        y_idx = np.clip(np.searchsorted(self.yi, y) - 1, 0, self.Ny - 2)
        n = (x - self.xi[x_idx]) / (self.xi[x_idx + 1] - self.xi[x_idx])
        u = (y - self.yi[y_idx]) / (self.yi[y_idx + 1] - self.yi[y_idx])
        z_pred = (1 - n) * (
            (1 - u) * self.zi[y_idx, x_idx] + u * self.zi[y_idx + 1, x_idx]
        ) + n * (
            (1 - u) * self.zi[y_idx, x_idx + 1] + u * self.zi[y_idx + 1, x_idx + 1]
        )
        return z_pred
```

1.2 Bicubic spline

Firstly, let us define how to calculate $(x, L(x))$ using cubic spline interpolation in one dimension.

```
class Cubic_Interpolation:
    def __init__(self, xi, yi, zi):
        self.xi = xi
        self.yi = yi
        self.zi = zi
        self.Nx = len(xi)
        self.Ny = len(yi)

    def _make_single_cubic(self, xi, yi, x):
        N = len(yi)
        a = np.zeros(N)
        b = np.zeros(N)
        A = np.zeros(N)
        B = np.zeros(N)
        M = np.zeros(N)
        h = np.zeros(N)
        for ii in range(N - 1):
            h[ii] = xi[ii + 1] - xi[ii]
        for ii in range(1, N - 1):
            a[ii] = h[ii] / (h[ii] + h[ii - 1])
            b[ii] = (6 / (h[ii] + h[ii - 1])) * ((yi[ii + 1] - yi[ii]) / h[ii] - (yi[ii] - yi[ii
                - 1]) / h[ii - 1])
            A[ii] = -a[ii] / (2 + (1 - a[ii]) * A[ii - 1])
            B[ii] = (b[ii] - (1 - a[ii]) * B[ii - 1]) / (2 + (1 - a[ii]) * A[ii - 1])
```

```
        M[N - 2] = (b[N - 2] - (1 - a[N - 2]) * B[N - 3]) / (2 + (1 - a[N - 2]) * A[N - 3])
        for ii in range(N - 3, 0, -1):
            M[ii] = A[ii] * M[ii + 1] + B[ii]
        ii = np.clip(np.searchsorted(xi, x) - 1, 0, len(xi) - 2)
        y_out = (M[ii] * (xi[ii + 1] - x) ** 3 / (6 * h[ii])
                + M[ii + 1] * (x - xi[ii]) ** 3 / (6 * h[ii])
                + (yi[ii] - M[ii] * h[ii] ** 2 / 6) * (xi[ii + 1] - x) / h[ii]
                + (yi[ii + 1] - M[ii + 1] * h[ii] ** 2 / 6) * (x - xi[ii]) / h[ii])
        return y_out
```

Similar to bilinear interpolation, we can calculate the interpolation along the y-axis for many points with different x but the same y value (i.e., along the x-axis at a fixed y). We use the cubic interpolation to get this y-line.

The code below means we first calculate the y-line $(x_i, y, L(x_i, y))$ for every x_i using cubic spline along the y-direction. Then, we use the obtained line to perform a cubic spline interpolation along the x-direction to calculate the value at (x, y).

```
def cubic_inter(self, x, y):
    y_line = np.zeros(self.Nx)
    for ii in range(self.Nx):
        y_line[ii] = self._make_single_cubic(self.yi, self.zi[:, ii], y)
    pred = self._make_single_cubic(self.xi, y_line, x)
    return pred
```

1.3 Interpolation Results Visualization

In this section, I present visualizations to compare the performance of cubic and bilinear interpolation methods on our test function.

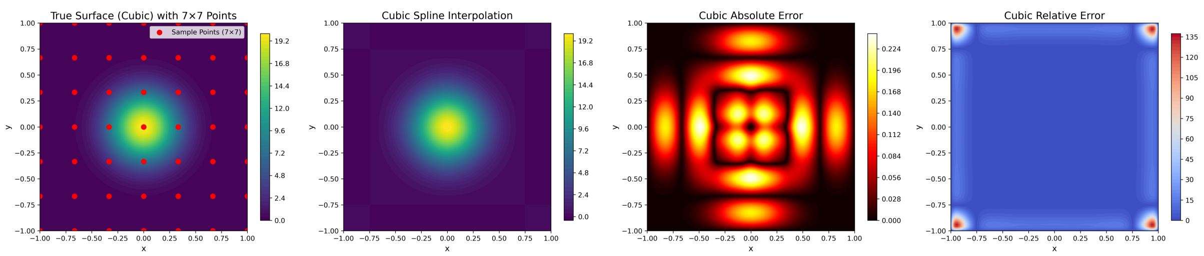

Figure 1 Visualization of cubic interpolation: true surface, interpolated surface, absolute error, and relative error using 7×7 sample points.

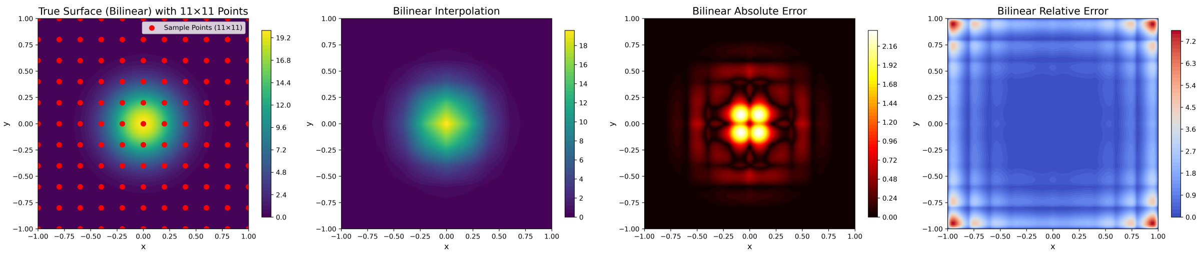

Figure 2 Visualization of bilinear interpolation: true surface, interpolated surface, absolute error, and relative error using 11×11 sample points.

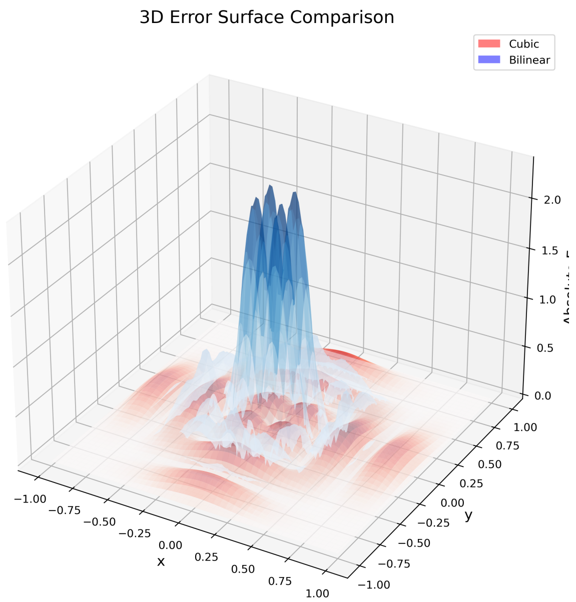

Figure 3 3D comparison of the absolute interpolation errors between cubic and bilinear methods.

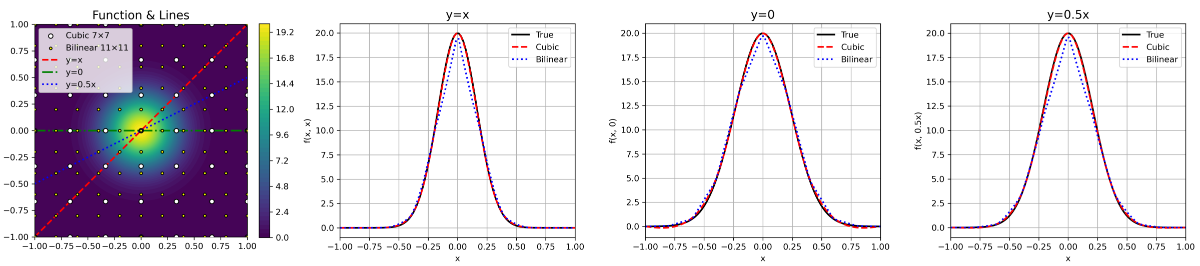

Figure 4 Comparison of true values and interpolation results of cubic and bilinear methods along typical lines (y = x, y = 0, y = 0.5x). The leftmost subplot is a contour map highlighting test lines and sample points.

2 Bilinear Interpolation Example Problem

Problem: Obtain the value $f(0.1, 0.5)$ using Bilinear interpolation with 4 interpolation points $f(1, 1) = 3$, $f(1, -1) = 2$, $f(-1, -1) = 1$, and $f(-1, 1) = 2.2$ (by hand).

Solution: We will use the bilinear interpolation formula derived earlier. Our grid points are:

$$\begin{aligned}
(x_i, y_j) &= (-1, -1) \quad \text{with value} \quad f(-1, -1) = 1 \\
(x_{i+1}, y_j) &= (1, -1) \quad \text{with value} \quad f(1, -1) = 2 \\
(x_i, y_{j+1}) &= (-1, 1) \quad \text{with value} \quad f(-1, 1) = 2.2 \\
(x_{i+1}, y_{j+1}) &= (1, 1) \quad \text{with value} \quad f(1, 1) = 3
\end{aligned}$$

For the point $(0.1, 0.5)$, we have $x_i = -1$, $x_{i+1} = 1$, $y_j = -1$, and $y_{j+1} = 1$.

First, we compute the interpolation along the y-direction:

- At $(x_i, y) = (-1, 0.5)$:

$$\begin{aligned}
A &= \left(1 - \frac{0.5 - (-1)}{1 - (-1)}\right) f(-1, -1) + \frac{0.5 - (-1)}{1 - (-1)} f(-1, 1) \\
&= (1 - 0.75) \cdot 1 + 0.75 \cdot 2.2 = 0.25 \cdot 1 + 0.75 \cdot 2.2 = 0.25 + 1.65 = 1.9
\end{aligned}$$

- At $(x_{i+1}, y) = (1, 0.5)$:

$$\begin{aligned}
B &= \left(1 - \frac{0.5 - (-1)}{1 - (-1)}\right) f(1, -1) + \frac{0.5 - (-1)}{1 - (-1)} f(1, 1) \\
&= \left(1 - \frac{1.5}{2}\right) \cdot 2 + \frac{1.5}{2} \cdot 3 = 0.25 \cdot 2 + 0.75 \cdot 3 = 0.5 + 2.25 = 2.75
\end{aligned}$$

Now, we interpolate along the x-direction:

$$\begin{aligned}
f(0.1, 0.5) &= \left(1 - \frac{0.1 - (-1)}{1 - (-1)}\right) A + \frac{0.1 - (-1)}{1 - (-1)} B \\
&= \left(1 - \frac{1.1}{2}\right) \cdot 1.9 + \frac{1.1}{2} \cdot 2.75 \\
&= (1 - 0.55) \cdot 1.9 + 0.55 \cdot 2.75 \\
&= 0.45 \cdot 1.9 + 0.55 \cdot 2.75 \\
&= 0.855 + 1.5125 = 2.3675
\end{aligned}$$

Therefore, using bilinear interpolation, we estimate $f(0.1, 0.5) \approx 2.37$.