1: Questions 1 & 2: Intro to Splines

For Questions 1-7, we deal primarily with the function given by

$$f(x) = \frac{x^{1+J_0(x)}}{\sqrt{(1+100x^2)(1-x)}} = \frac{x^{1+J_0(x)}}{\sqrt{1-x+100x^2-100x^3}} \tag{1}$$

We perform various schemes of spline interpolation (not-a-knot, clamped) considering uniform spacing of the points. We also observe how incorrectly specifying the end derivative value causes the error to blow up, and use this to gain insight into the spline algorithm.

We then come up with a non-uniform point spacing scheme which uses the (almost) minimum number of points to achieve 6-digit accuracy. We then compare the performance spline interpolation and rational interpolation in fitting this particular function.

1.1 Background

Spline interpolation uses the knowledge of the function derivatives to fit a smooth polynomial function between points. We focus in particular on the cubic spline, which uses the 2nd-derivatives to fit a cubic polynomial. This ensures that the 2nd-derivative is continuous throughout the domain.

Say the cubic spline interpolated value at x is given by some y. We find the nearest 2 points, say x_j, and x_{j+1}. Let the function values at these points be y_j and y_{j+1}, and let the 2nd-derivative values be y''_j and y''_{j+1}. The interpolated value y is given by,

$$y = Ay_j + By_{j+1} + Cy''_j + Dy''_{j+1} \tag{2}$$

where A and B are given as

$$A = \frac{x_{j+1}-x}{x_{j+1}-x_j} \qquad B = \frac{x-x_j}{x_{j+1}-x_j} \tag{3}$$

and the 2nd-derivative coefficients C and D are given by

$$C = \frac{1}{6}(A^3-A)(x_{j+1}-x_j)^2 \qquad D = \frac{1}{6}(B^3-B)(x_{j+1}-x_j)^2 \tag{4}$$

Using these, we can obtain expressions for the first and second derivative of y.

$$\frac{dy}{dx} = \frac{y_{j+1}-y_j}{x_{j+1}-x_j} - \frac{3A^2-1}{6}(x_{j+1}-x_j)y''_j + \frac{3B^2-1}{6}(x_{j+1}-x_j)y''_{j+1} \tag{5}$$

$$\frac{d^2y}{dx^2} = Ay''_j + By''_{j+1} \tag{6}$$

which ensures that the interpolation is Lagrangian in both the function values and the 2nd-derivative. This ensures that the values at the data points are exactly matched.

We assume knowledge of the 2nd-derivative while performing cubic-spline interpolation. To obtain these, we further stipulate continuity of the 1st-derivatives across the interval boundaries.

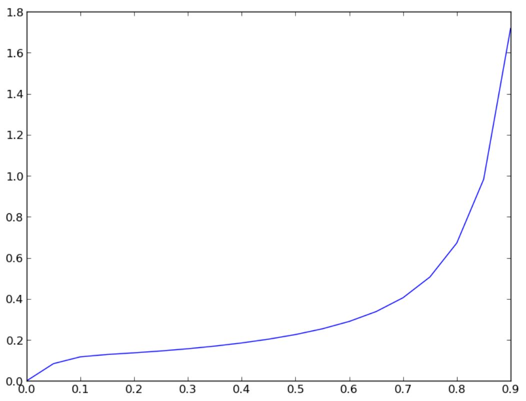


Figure 1: Function plot $f(x)$

We do this by using the neighbouring sets of points (x_{j-1}, x_j) and (x_j, x_{j+1}) in (5). This gives us the condition (for $j = 2, 3, \ldots, N-1$)

$$\frac{x_j - x_{j-1}}{6} y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3} y''_j + \frac{x_{j+1} - x_j}{6} y''_j = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}} \tag{7}$$

This gives us $N-2$ linear equations in the N unknowns $y''_1, y''_2, \ldots, y''_N$. We need 2 more equations to solve for the 2nd-derivatives. We use various boundary conditions as the other 2 equations. Depending on the nature of equations we use, we get different types of cubic splines. These are:

- **Normal Spline** - The second derivatives are zero at the ends.
- **Clamped Spline** - The first derivative of the function is specified at the end points. This translates to setting (5) to a given value.

$$\begin{aligned} \frac{y_2 - y_1}{x_2 - x_1} - \frac{1}{3}(x_2 - x_1) y''_1 - \frac{1}{6}(x_2 - x_1) y''_2 &= y'_1 \\ \frac{y_N - y_{N-1}}{x_N - x_{N-1}} - \frac{1}{6}(x_N - x_{N-1}) y''_{N-1} - \frac{1}{3}(x_N - x_{N-1}) y''_N &= y'_N \end{aligned} \tag{8}$$

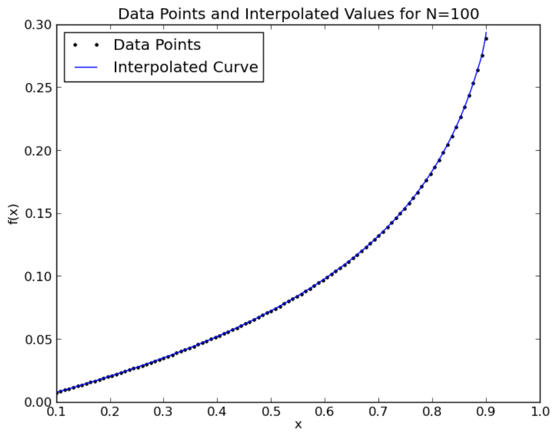
- **Not-a-Knot spline** - Without specifying any extra conditions at the end points, the not-a-knot spline requires that the 3rd-derivative of the spline is continuous at x_2 and x_{N-1}. We can derive these conditions to be

$$\begin{aligned} (x_3 - x_2) y''_1 - (x_3 - x_1) y''_2 + (x_2 - x_1) y''_3 &= 0 \\ (x_N - x_{N-1}) y''_{N-2} - (x_3 - x_{N-2}) y''_{N-1} + (x_{N-1} - x_{N-2}) y''_N &= 0 \end{aligned} \tag{9}$$

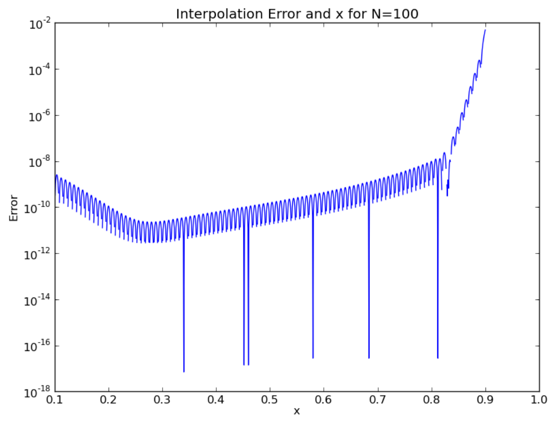
We prefer not-a-knot spline if there is no prior knowledge of the function derivatives, as this ensures maximal smoothness. We use the base SPLINE.C code for the cubic spline interpolation.

1.2 Function Plot

The plot of the function $f(x)$ given by (1) over $[0.1, 0.9]$ is shown in Figure 1. The function takes 0 value at $x = 0$ and goes to ∞ as $x \to 1$. The function is analytic in $[0, 1)$



(a) Theoretical and interpolated values of $f(x)$



(b) Spline Interpolation Error for $N = 100$

Figure 2: Cubic Spline Interpolation using $N = 100$, function plot and error.

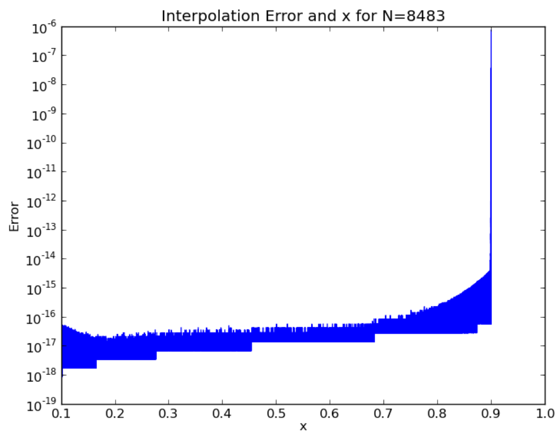
4: Question 4: Clamped Splines

In this question, we evaluate the performance of spline interpolation when the derivative at the end points are specified. We also observe the effect of varying point spacing on interpolation error, find the first 6 digit accurate interpolation scheme, and observe how the error evolves with point spacing. Giving more information can potentially improve the performance of the algorithm. We analytically compute the value of the first derivative at $x = 0.1, 0.9$, and pass this to the SPLINE function. We define FUNC DERIV to compute the derivative value at these points. To do this, we let $N(x) = x^{1+J_0(x)}$ and $D(x) = \sqrt{1 - x + 100x^2 - 100x^3}$. Then $f(x) = \frac{N(x)}{D(x)}$. We can then evaluate $f'(x)$ as,

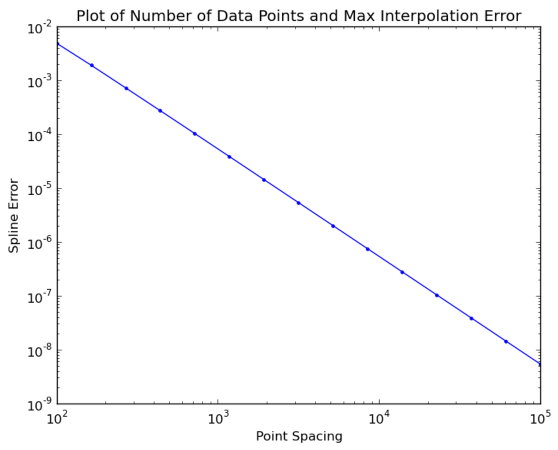
$$\begin{aligned} N'(x) &= x^{1+J_0(x)}\left(-J_1(x)log(x) + \frac{1+J_0(x)}{x}\right) \\ D'(x) &= \frac{-1+200x-300x^2}{2\sqrt{1-x+100x^2-100x^3}} \\ f'(x) &= \frac{D(x)N'(x) - N(x)D'(x)}{(D(x))^2} \end{aligned} \tag{10}$$

Using this, we perform spline interpolation. We use LOGSPACE to systematically vary up with the spacing of sample points. Figures (2a) and (2b) show the interpolation for $N = 100$ points uniformly spaced in $[0.1, 0.9]$. We see that the error blows up as $x \to 0.9$, as we get closer to the singularity.

We see that the first 6-digit accurate interpolation scheme uses $N = 8483$. This means that the point spacing $\Delta x < 10^{-4}$. We see from Figure (3a) that the error is low over most fo the interval, but shoots up near the singularity. The high point spacing just ensures that the shoot up still maintains 6-digit accuracy. Figure (3b), shows how the max interpolation error decreases almost linearly with number of data points ($loglog$ scale). So, $Err \propto \frac{1}{(No\ of\ Points)^\alpha}$



(a) Spline Interpolation Error for $N = 8483$

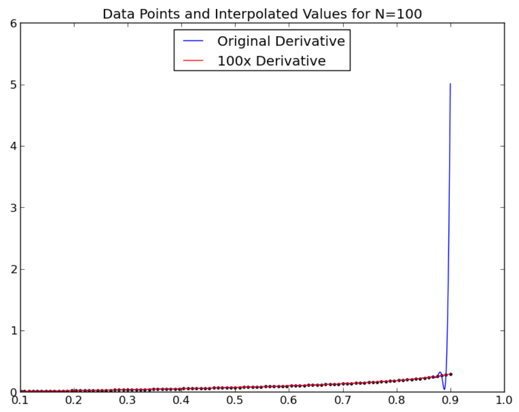


(b) Variation of Interpolation Error with Spacing

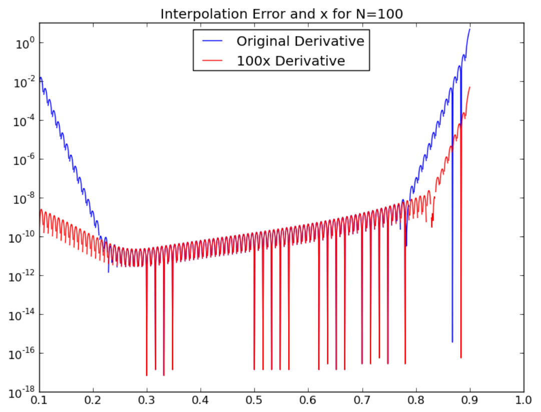
Figure 3: Cubic Spline Interpolation using $N = 100$, function plot and error.

5: Questions 5: Giving a different derivative

In this question, we evaluate the performance of the clamped spline algorithm when the end point derivatives are incorrectly specified. This is of tremendous importance, since we might not actually know the derivatives, or might make approximations. Ideally, we want our algorithm to do reasonably well under derivative change. It is evident that the interpolation will take a hit near the end points, but we would ideally like the intermediate interpolation error to be more or less the same. This goes back to the stability arguments of the Spline algorithm.

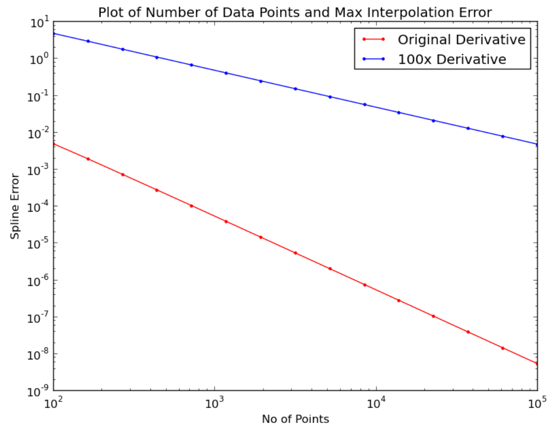


(a) Interpolation for $N = 100$

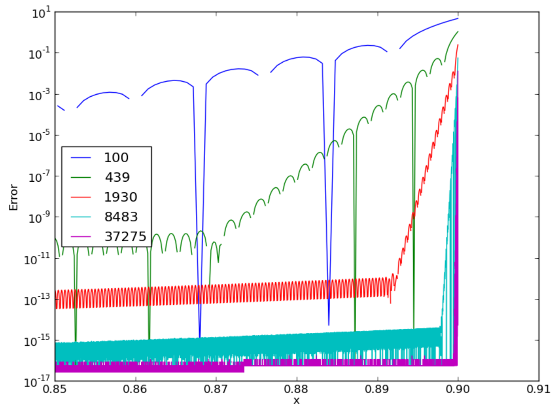


(b) Spline Interpolation Error for $N = 100$

Figure 4: Cubic Spline Interpolation using $N = 100$, using 100x thee actual derivative (compared with the original derivative)



(a) Variation of Interpolation Error with Spacing



(b) Focused variation with Spacing

Figure 5: Variation of error with point spacing using 100x the actual derivative, and the zoomed in error for $[0.85, 0.9]$

Figure (4a) compares for interpolated function for using the original derivative values at $x = 0.1, 0.9$, and 100 times the actual value (using $N = 100$ points). Figure (4b) shows the error variation for the same. As we scale up the end point derivative, the error near the end points shoots up. But interestingly enough, the error in the middle regions (sufficiently far away from the edges) is **exactly** the same as using the theoretical derivative values.

This means that somehow, the cubic spline tridiagonal matrix solver gives exactly the same 2nd-derivative values at the intermediate points, and it doesn't vary very much with the clamped derivative value.

Figure (5a) shows the comparison of how the *maximum*-error decays with the point spacing change. We see that using 100x the derivative gives a strictly higher error decay. We see that using this, achieving 6-digit accuracy is not possible for similar order of points. There is an exponential scaling in the decay, but the LOGLOG linear behaviour is maintained.

Figure (5b) shows how the error using 100x derivative varies in $[0.85, 0.9]$, that is near the singularity. As expected, the error is very high for low values of point spacing, and the error gets lower for lower point spacing. The spike in the error near 0.9 is higher for a larger number of points, but the actual error is lower.

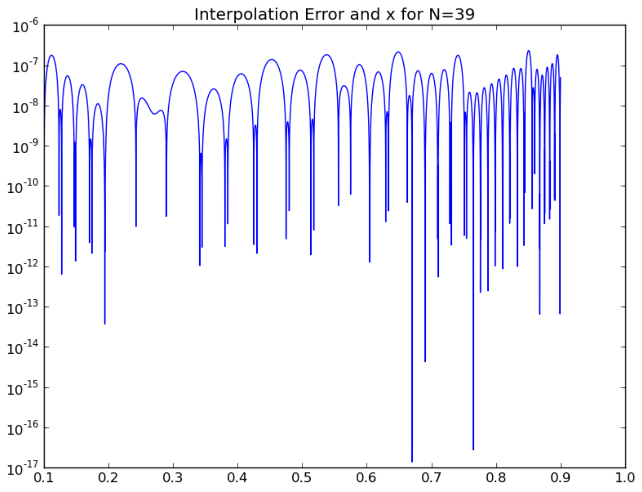


Figure 6: Error variation using non-uniform spacing

6: Question 6: Non-Uniform Spacing and Interpolation

In this section, we try to use non-uniform spacing to achieve 6-digit accuracy with the lowest number of data points. This means that we require a choice of points / spacing such that the error everywhere is strictly $< 10^{-6}$.

In a very rough sense, the spline interpolation error grows as $\mathbb{O}(\frac{f^{(4)}(\zeta)}{4!}\Delta x^4)$, where ζ is chosen to maximize the 4th-derivative. This means that the interpolation error is upper bounded by the maximum value the 4th-derivative can take, and the the point-spacing raised to the 4th-power. We can use this to help achieve 6-digit accuracy.

Divide the range $[0.1, 0.9]$ into windows of size say w ($w = 0.095$ was used). Now within each window, find the maximum value of the 4th-derivative (say at point ζ). For some constant c ($c = 1$ was used), find Δx such that

$$\frac{f^{(4)}(\zeta)}{4!}\Delta x^4 c < 10^{-6} \tag{11}$$

Now sample the function within the window at this value of spacing Δx. Repeat this over all windows, and we have the sampled function.

An important design consideration here is the choice of the window size. If the window size is too small, we end up oversampling (as atleast one point is taken from each sample window). On the flip side, if the size of the sample window is too large and the derivative value is high in a window, the entire window is sampled at a very fine spacing (while it may not have been required for a smaller window size). We need to strike a fine balance between these 2. We found that with $c = 1$ and

Table 1: Window and Number of sample points

Window interval	No of Sample points
0.100000 - 0.195000	5
0.195000 - 0.290000	2
0.290000 - 0.385000	2
0.385000 - 0.480000	3
0.480000 - 0.575000	3
0.575000 - 0.670000	4
0.670000 - 0.765000	5
0.765000 - 0.860000	9
0.860000 - 0.900000	6

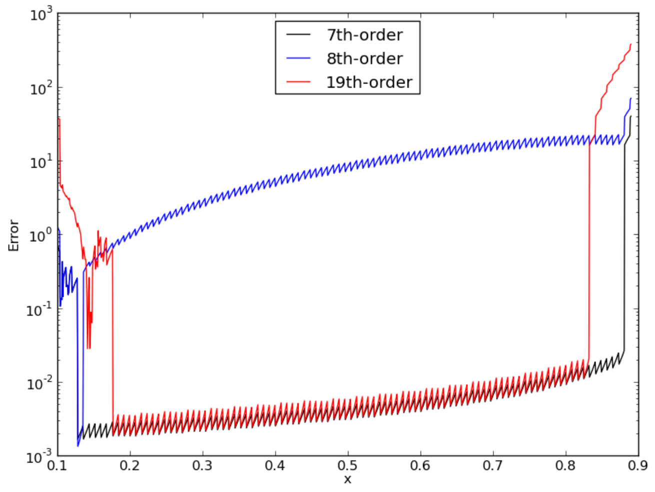


Figure 7: Error using Rational Interpolation, and variation with order

a window spacing of 0.095, we were able to achieve the required accuracy using just **39 sample points**. The number of points required in each interval are shown in Table (1)

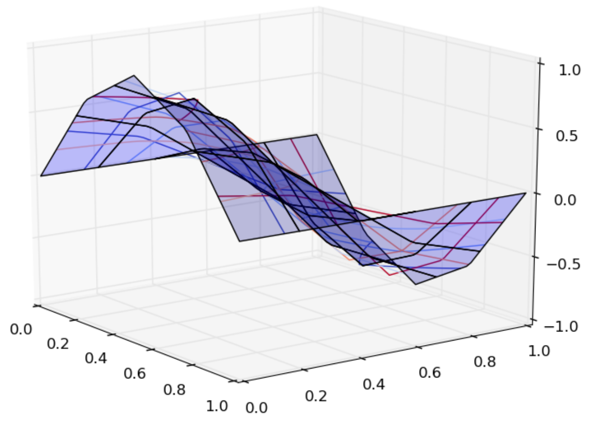
Figure (6) shows how the error varies with x. We see an almost smooth peak variation, and the maximum interpolation error is about $2.4x10^{-7}$ using the 39 point non-uniform sampling scheme.

Note that it is possible to use an adaptive algorithm to come up with an even better point choice. It might be possible to combine points in the intervals further away from the singularity and maintain the accuracy. Our current point choice is fairly good since each interval has strictly more than 1 point. We chose $w = 0.095$ so as to enable the last interval to be smaller, and thus the effects of high derivative is felt in the smaller interval.

7: Question 7: Rational Interpolation

In this section, we use rational interpolation on the function given above, and compare its performance to that of spline interpolation. Rational interpolation fits a rational polynomial to the function, of a specified maximum order. It is exceedingly useful in fitting functions with poles / which have rational semblance. Our given function is not exactly rational, but the interpolation should capture the pole behaviour quite well. We use ideas developed in [1] to write a Python script for rational interpolation, and thus fit it to the function.

Figure (7) shows the error variation for rational interpolation, and the order variation. The error for odd-order functions seems to be invariant of the order, indicating that it is well approximated by even a low order function. However, the interpolation value seems to blow up near the end points. Increasing the order doesn't seem to improve the best case performance. And the even-order rational interpolation seems to perform quite badly. As a whole, spline interpolation seems to do a lot better, and the error is more controllable.


(a) Bilinear Interpolation function plot

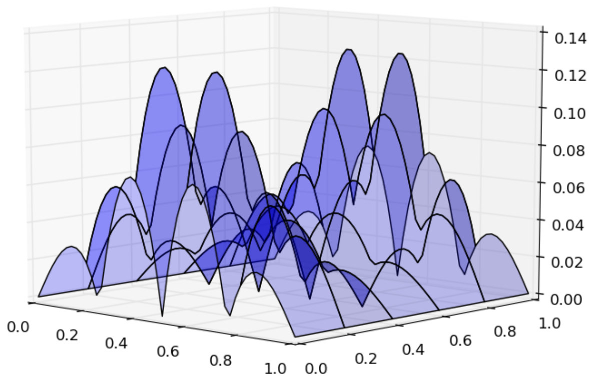

(b) Error in Bilinear interpolation

Figure 8: Bilinear Interpolation function plot and error over the 50x50 grid.

8: Question 8: Two-Dimensional Interpolation

In this question, we analyse and compare various schemes of 2-dimensional interpolation. To do this, we create a 5x5 grid of sample points for the smooth function

$$f(x,y) = sin(\pi x)cos(\pi y) \tag{12}$$

in the range $[0,1]$ on both axes. This is a very well-behaved function, with known derivative values, so we use this as the testing ground for our algorithms. We want to interpolate to a 50x50 mesh of points in the same interval. The various subsections to follow cover methods of interpolation, and how they compare.

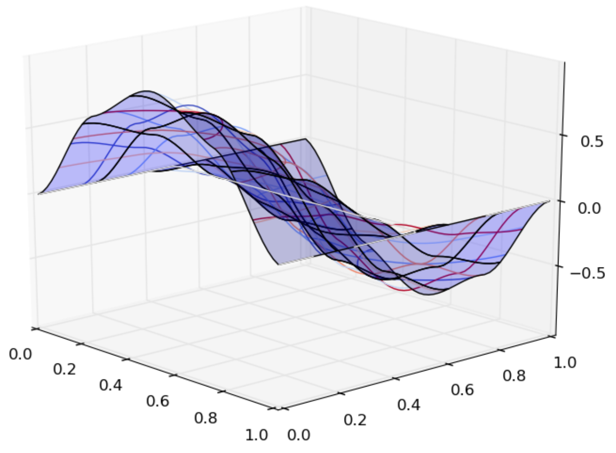
8.1 Bilinear interpolation

Bilinear Interpolation is the most rudimentary interpolation scheme we consider here. In this scheme, for a given (x,y) to be interpolated at, we consider the 4-nearest sample points $(x1,y1),(x1,y2),(x2,y2),(x2,y1)$ taken anti-clockwise from the bottom left $(x1 < x2, y1 < y2)$. Let the function values at these points be $z11, z12, z22, z21$. Then the interpolated value is given by

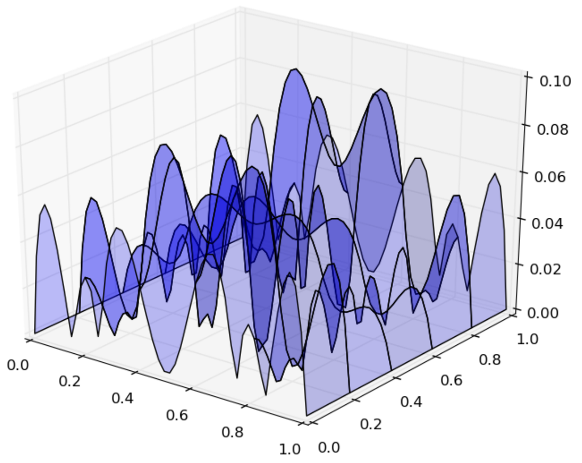
$$z = \frac{z11(x2-x)(y2-y) + z21(x-x1)(y2-y)}{(x2-x1)(y2-y1)} + \frac{z12(x2-x)(y-y1) + z22(x-x1)(y-y1)}{(x2-x1)(y2-y1)} \tag{13}$$

This considers only the function value of the nearest 4 points and uses this to interpolate. We can easily see that this is only 2nd-order accurate in both the dimensions.

Figure (8a) shows a plot of the interpolated function (it resembles cos and sin projected on the axes). Figure (8b) shows the plot of bilinear interpolation error. We see that the peak error is around 0.12.



(a) Bicubic Interpolation function plot



(b) Error in Bicubic interpolation

Figure 9: Bicubic Interpolation function plot and error over the 50x50 grid.

8.2 Bicubic Interpolation

Bicubic Interpolation aims at giving a piecewise smooth interpolation scheme. It requires the user to specify the derivatives at the 4 gridpoints (mentioned above), both the directional derivatives $\partial z/\partial x, \partial z/\partial y$ and the cross derivative $\partial^2 z/\partial x \partial y$. The exact interpolation scheme and the code from [1] was used for this question.

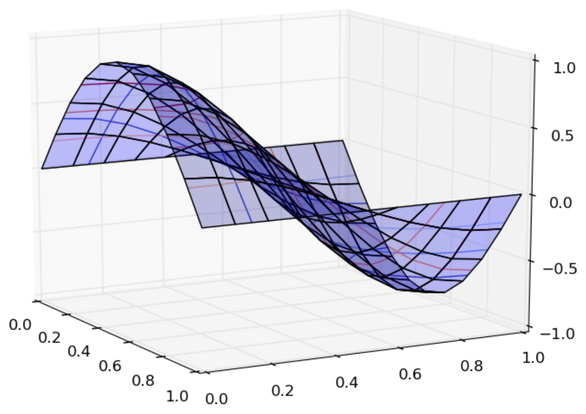
Figure (9a) shows a plot of the interpolated function (it resembles cos and sin projected on the axes). Figure (9b) shows the plot of bicubic interpolation error. We see that the peak error is around 0.08, which is slightly than the linear interpolation error. The curve also a little smoother.

We also plotted the behaviour of these 2 interpolation schemes using a 10x10 grid of sample points, and a 100x100 grid to be interpolated over, seen in Figures (10) and (11). The results were quite different from the 5x5 case. Bilinear actually seemed to perform better in this case (peak error 0.015), and the error variation was a lot smoother than the bicubic interpolation (peak error 0.035).

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, ”Numerical Recipes in C”

[2] J. Stoer, R. Bulirsch, ”Introduction to Numerical Analysis”


(a) Bilinear Interpolation function plot

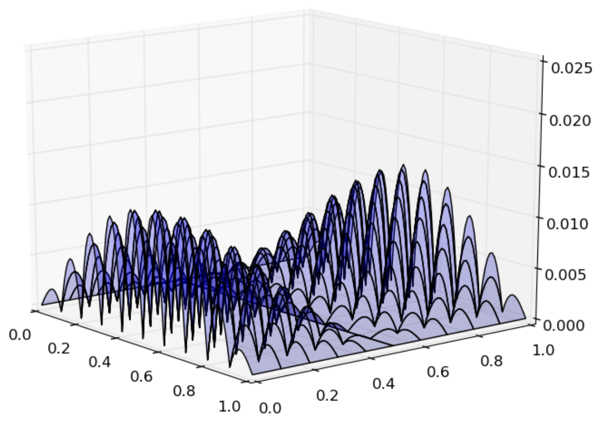
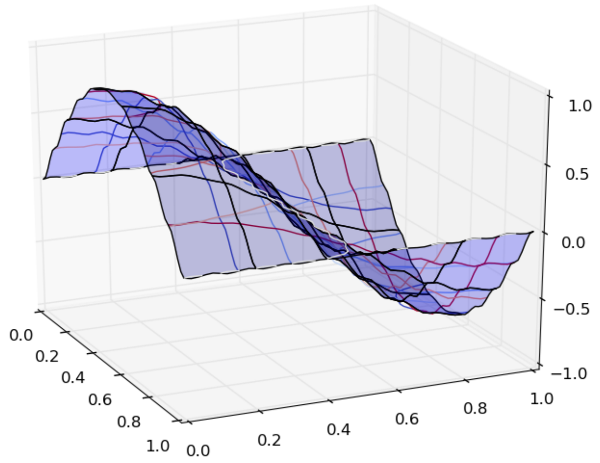

(b) Error in Bilinear interpolation

Figure 10: Bilinear Interpolation function plot and error over the 100x100 grid.


(a) Bicubic Interpolation function plot

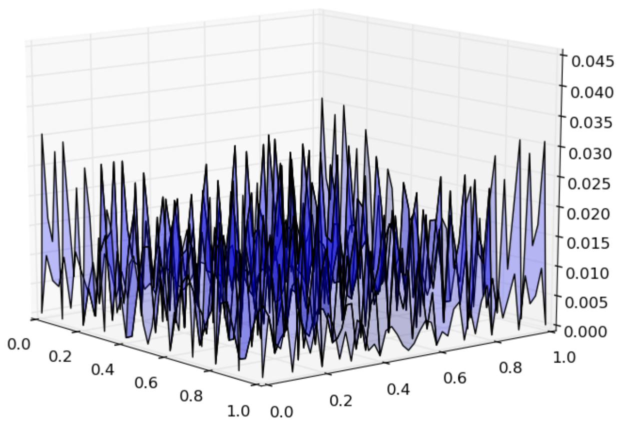

(b) Error in Bicubic interpolation

Figure 11: Bicubic Interpolation function plot and error over the 100x100 grid.