CSE3365 - Project 2
Analysis of polynomial interpolation

Eric Smith
Bobby B. Lyle School of Engineering,
Computer Science

March 2017

I. Project Description

This article attempts to demonstrate two different methods of polynomial interpolation: interpolation on uniform and Chebyshev points and cubic spline interpolation. First, given the function:

$$f(x) = \frac{-1}{1 + 25x^2} \tag{1}$$

we will generate a set of points to be interpolated. In *Part I*, we will use both uniform and Chebyshev distributions to interpolate these points. Then in *Part II*, we will use cubic spline interpolation for the same set of points generated by *f(x)*. The results of using these two methods will be discussed in *Numerical Results and Discussion*.

II. Concepts and Theory

Polynomial interpolation involves constructing a polynomial to approximate a function given a discreet set of data points within a finite interval. In Part I, we use *n+1* points to find a unique polynomial

$$P_n(x) \text{ , } with\ degree \leq n$$

using Newton's method . The points are selected by one of two ways: uniform or Chebyshev distributions. Uniform distributions are divided evenly across an interval, meanwhile, Chebyshev points are distributed across an interval using the following set of equations:

$$x_i = -\cos(\theta_i), \quad \theta_i = ih$$
$$h = \frac{\pi}{n}, \quad i = 0 : n \tag{2}$$

The results are distributions illustrated in *Figure 1* and *Figure 2*.

Figure 1: *Uniform distribution n=16*

Figure 2: *Chebyshev distribution n=16*

While both distributions will allow us to accurately interpolate *f(x)*, interpolation along uniform points only gives a good approximation near the center of a given interval. Meanwhile, the Chebyshev distribution, which has points clustered near the endpoints of an interval, gives a good approximation of the entire interval.

For the outcomes of polynomial interpolation of the function *f(x)*, with both uniform and Chebyshev points, see *Numerical Results and Discussion.*

The second form of interpolation used in this paper is *cubic spline interpolation*. Given by a set of points, this process involves finding a cubic polynomial on *n* subintervals created by the set of *n+1* points. This cubic polynomial, *S(x)*, and its first and second derivatives are continuous at the interior points x_1, x_2, ..., x_i.

Unlike normal polynomial interpolation, cubic spline interpolation involves interpolating on small subsections of the interval rather than the whole interval itself. This results in the smoothest of interpolating curves, making it more accurate than most interpolations. (See *Numerical Results and Discussion*)

III. Numerical Implementation

Part I consisted of using both uniform and Chebyshev distributions for polynomial interpolation. Using an interval of [-1, 1], we use subintervals of sizes *n = 4,8,16* with both types of point distributions (uniform and Chebyshev) to calculate a total of six polynomials.

These polynomials were plotted against a very fine mesh alongside the original set of points and the function *f(x)*. (See *Figure 3* and *Figure 4*.)

Part II used a cubic spline, *S(x)*, to interpolate the same set of points generated by *f(x)* using interval sizes of *n = 2,4,8,16*. Interpolation with a cubic spline consists of four main parts. The first of which is to construct a linear

polynomial, $S''(x)$ in the form:

$$S_i''(x) = a_i\left(\frac{x_{i+1}}{h}\right) + a_{i+1}\left(\frac{x - x_i}{h}\right)$$
$$a_i,\ a_{i+1}:\ \textit{to be determined}, \quad i = 1 : n+1 \tag{3}$$

Here, i represents the the interval being evaluated. Meanwhile, the curvatures of each interval are represented by a_i and a_{i+1}. These a_i's can be obtained from solving the linear algebraic system:

$$Ax = b\ ,\ \textit{where}\ A \in R^{(n-1)\times(n-1)}$$
$$a, b \in R^{n-1}\ ,\ \textit{and}\ a_0, a_n = 0 \tag{4}$$

Integrating $S''(x)$ twice gives us the form of the cubic splice:

$$S_i(x) = \frac{a_i(x_{i+1} - x)^3}{6h} + \frac{a_{i+1}(x - x_i)^3}{6h}$$
$$+ b\ \left(\frac{x_{i+1} - x}{h}\right) + c_i\ \left(\frac{x - x_i}{h}\right)$$
$$\textit{where}\ b_i = y_i - \frac{a_i h^2}{6}\ \textit{and}\ c_i = y_{i+1} - \frac{a_{i+1}h^2}{6} \tag{5}$$

The points n = *2,4,8,16* were interpolated using $S(x)$ after calculating the curvatures for each interval. These were then plotted against a fine mesh.

IV. Numerical Results and Discussion

i. Part A

As n increased, the uniform distribution gave a reasonably accurate interpolation of $f(x)$ towards the center of the function (*See Figure 3*). However as theorized, this distribution was not as accurate towards the endpoints of $f(x)$.

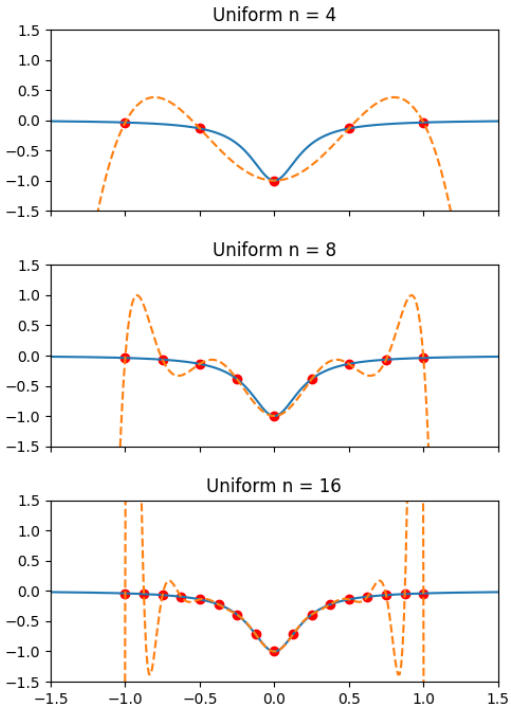


Figure 3: *Interpolation on Uniform Distribution*

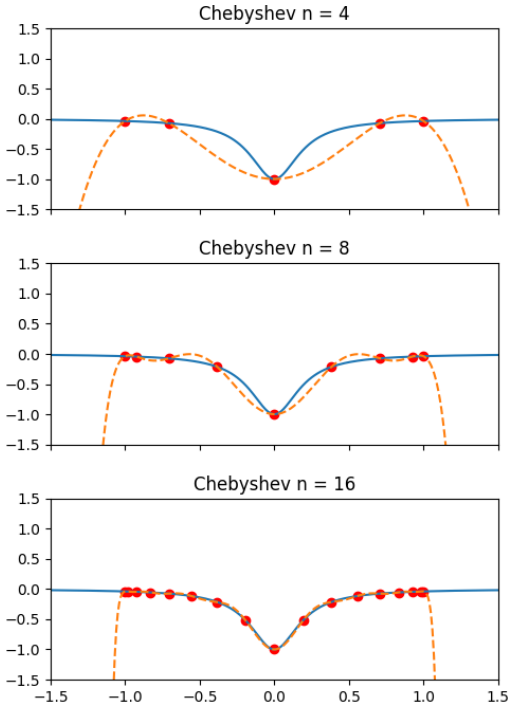


Figure 4: *Interpolation on Chebyshev Distribution*

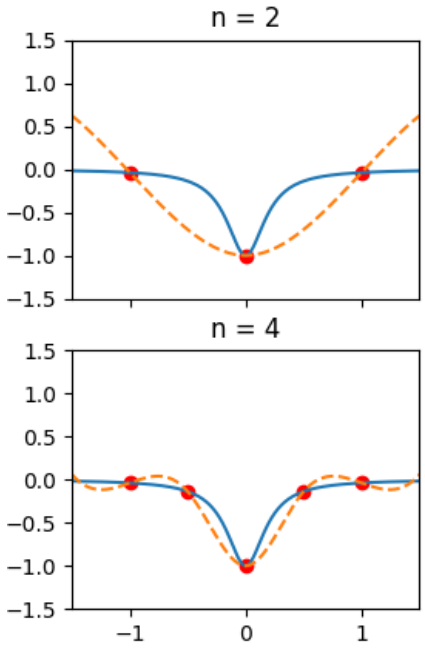


Figure 5: *Cubic Spline n = 2,4*

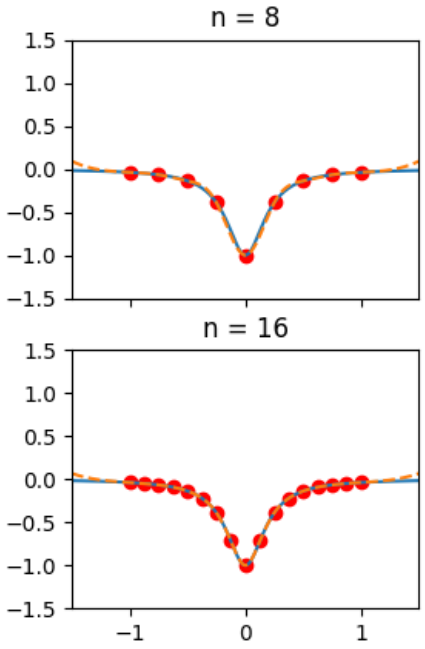


Figure 6: *Cubic Spline n = 8,16*

Meanwhile as n increased, the interpolation on the Chebyshev distribution became more accurate on the whole interval (*See Figure 4*). These results agree with the theory described in *Concepts and Theory*.

ii. Part B

For the cubic spline, as the value of n increased, the interpolation on the interval also became more accurate. Even at n=4, the cubic spline interpolation was more accurate than the interpolation of Chebyshev at n=4. This made the cubic spline interpolation the most accurate on all intervals of the three types of interpolations used. (See *Figure 5* and *Figure 6*.)

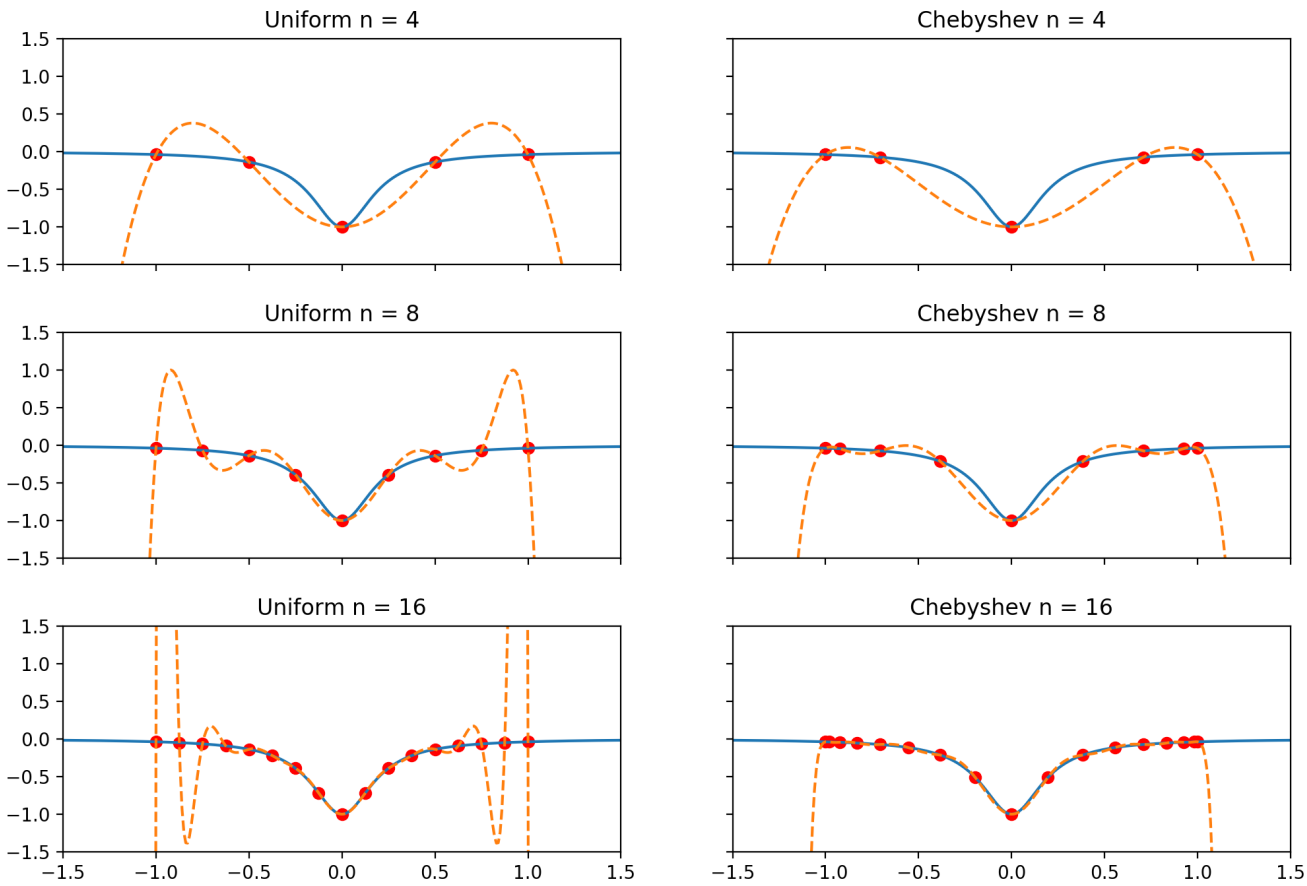
These results agree with the theory discussed in *Concepts and Theory*.

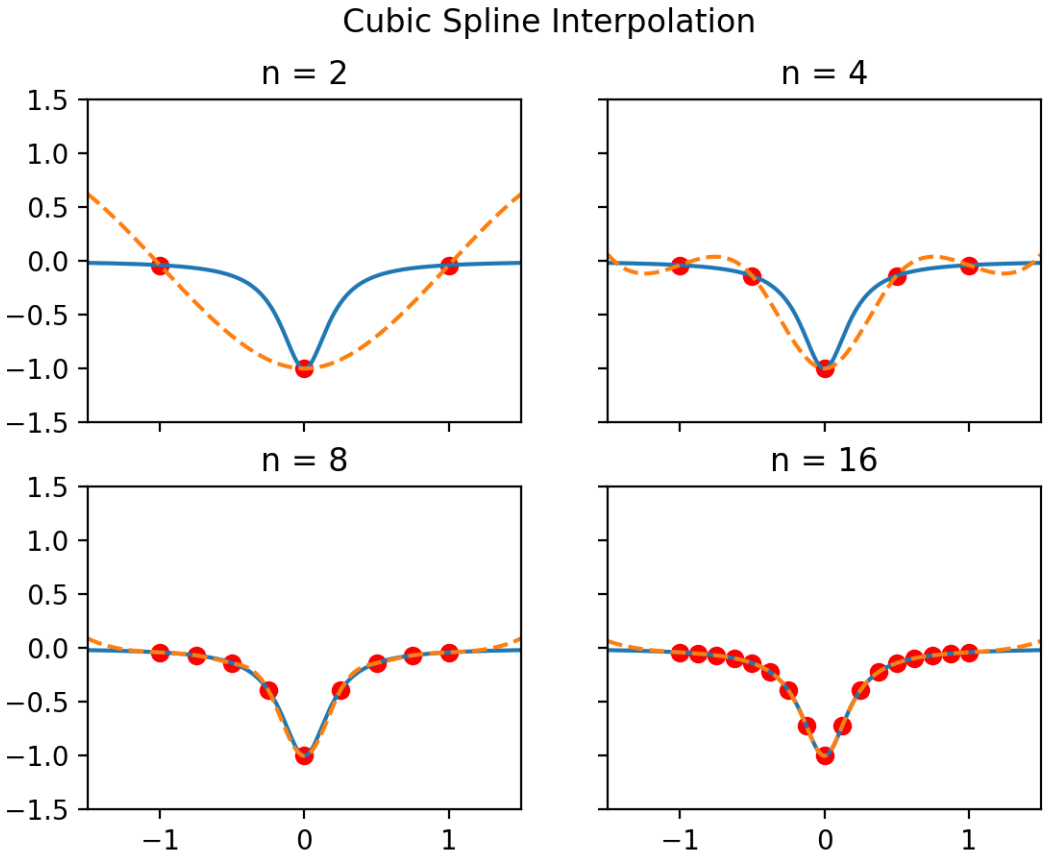
V. Conclusion

In this article, we looked at two different methods of polynomial interpolation: interpolation on uniform and Chebyshev points and cubic spline interpolation. In *Concepts and Theory*, we saw how to generate uniform and Chebyshev points as well as the purpose of cubic splines. Then, in *Numerical Implementation*, we saw how the two types of distributions were used to interpolate $f(x)$. We also saw how to generate a cubic spline for each set of subintervals.

Finally, in *Numerical Results and Discussion*, we saw the results of our experiment: Interpolating with the Chebyshev distribution was more accurate than the uniform distribution but not as accurate as the cubic spline interpolation.

Attached you will find the Python code used to calculate the data sets in this article.


Interpolation on Uniform and Chebyshev Distributions
Uniform n = 4
Chebyshev n = 4
Uniform n = 8
Chebyshev n = 8
Uniform n = 16
Chebyshev n = 16
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Cubic Spline Interpolation
n = 2
n = 4
n = 8
n = 16
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1
0
1

```
"""
Eric Smith
Project 2 - Part 1

April 4th, 2017

"""
import numpy
from math import pi, cos
import matplotlib.pyplot as plt
from newtonPoly import *

def f(x):
    return -1/(1+(25*(x**2)))

# uniform intervals for n = 4,8,16
xData4 = numpy.arange(-1,1.5,0.5) # n = 4
xData8 = numpy.arange(-1,1.25,0.25) # n = 8
xData16 = numpy.arange(-1,1.125,0.125) # n = 16

# Chebyshev intervals for n = 4,8,16
xDataCheby4 = [numpy.float64(-1*cos(i*(pi/4))) for i in range(0,5)]
xDataCheby8 = [numpy.float64(-1*cos(i*(pi/8))) for i in range(0,9)]
xDataCheby16 = [numpy.float64(-1*cos(i*(pi/16))) for i in range(0,17)]

# coefficients for uniform points
coeff4 = coeffts(xData4, [f(x) for x in xData4])
coeff8 = coeffts(xData8, [f(x) for x in xData8])
coeff16 = coeffts(xData16, [f(x) for x in xData16])

# coefficients for Chebyshev points
coeffCheby4 = coeffts(xDataCheby4, [f(x) for x in xDataCheby4])
coeffCheby8 = coeffts(xDataCheby8, [f(x) for x in xDataCheby8])
coeffCheby16 = coeffts(xDataCheby16, [f(x) for x in xDataCheby16])

# fine mesh
mesh = numpy.arange(-1.5,1.51,0.01)

plt.close('all')
fig, axarr = plt.subplots(3, 2,figsize=(12,8), sharex=True, sharey=True)
plt.axis([ -1.5, 1.5, -1.5, 1.5 ])
plt.subplots_adjust(hspace = 0.3)

```

```
44 # four uniform
45 axarr[0, 0].plot(xData4, [f(x) for x in xData4], 'ro')
46 axarr[0, 0].plot(mesh, [f(x) for x in mesh])
47 axarr[0, 0].plot(mesh, evalPoly(coeff4, xData4, mesh), '--'
   )
48 axarr[0, 0].set_title('Uniform n = 4')
49
50 # eight uniform
51 axarr[1, 0].plot(xData8, [f(x) for x in xData8], 'ro')
52 axarr[1, 0].plot(mesh, [f(x) for x in mesh])
53 axarr[1, 0].plot(mesh, evalPoly(coeff8, xData8, mesh), '--'
   )
54 axarr[1, 0].set_title('Uniform n = 8')
55
56 # sixteen uniform
57 axarr[2, 0].plot(xData16, [f(x) for x in xData16], 'ro')
58 axarr[2, 0].plot(mesh, [f(x) for x in mesh])
59 axarr[2, 0].plot(mesh, evalPoly(coeff16, xData16, mesh),
   '--')
60 axarr[2, 0].set_title('Uniform n = 16')
61
62 # four Chebyshev
63 axarr[0, 1].plot(xDataCheby4, [f(x) for x in xDataCheby4],
   'ro')
64 axarr[0, 1].plot(mesh, [f(x) for x in mesh])
65 axarr[0, 1].plot(mesh, evalPoly(coeffCheby4, xDataCheby4,
   mesh), '--')
66 axarr[0, 1].set_title('Chebyshev n = 4')
67
68 # eight Chebyshev
69 axarr[1, 1].plot(xDataCheby8, [f(x) for x in xDataCheby8],
   'ro')
70 axarr[1, 1].plot(mesh, [f(x) for x in mesh])
71 axarr[1, 1].plot(mesh, evalPoly(coeffCheby8, xDataCheby8,
   mesh), '--')
72 axarr[1, 1].set_title('Chebyshev n = 8')
73
74 # sixteen Chebyshev
75 axarr[2, 1].plot(xDataCheby16, [f(x) for x in xDataCheby16]
   , 'ro')
76 axarr[2, 1].plot(mesh, [f(x) for x in mesh])
77 axarr[2, 1].plot(mesh, evalPoly(coeffCheby16, xDataCheby16,
    mesh), '--')
78 axarr[2, 1].set_title('Chebyshev n = 16')
79
80 # Fine-tune figure; hide x ticks for top plots and y ticks
   for right plots
81 plt.setp([a.get_xticklabels() for a in axarr[0, :]],
   visible=False)
82 plt.setp([a.get_yticklabels() for a in axarr[:, 1]],
```

```
82 visible=False)
83
84 plt.show()
```

```
"""
Eric Smith
Project 2 - Part 2

April 4th, 2017

"""
import numpy as numpy
from cubicSpline import *
from math import pi, cos
import matplotlib.pyplot as plt

def f(x):
    return -1/(1+(25*(x**2)))

# uniform intervals for n = 2,4,8,16
xData2 = numpy.asarray(numpy.arange(-1.0,2.0)) # n = 2
xData4 = numpy.asarray(numpy.arange(-1,1.5,0.5)) # n = 4
xData8 = numpy.asarray(numpy.arange(-1,1.25,0.25)) # n = 8
xData16 = numpy.asarray(numpy.arange(-1,1.125,0.125)) # n
 = 16

yData2 = numpy.asarray([f(x) for x in xData2])
yData4 = numpy.asarray([f(x) for x in xData4])
yData8 = numpy.asarray([f(x) for x in xData8])
yData16 = numpy.asarray([f(x) for x in xData16])

k2 = curvatures(xData2, yData2)
k4 = curvatures(xData4, yData4)
k8 = curvatures(xData8, yData8)
k16 = curvatures(xData16, yData16)

mesh = numpy.arange(-1.5, 1.51, 0.01)

plt.close('all')
fig, axarr = plt.subplots(2, 2, sharex=True, sharey=True)
plt.xlim(-1.5,1.5)
plt.ylim(-1.5,1.5)

# n = 2
axarr[0, 0].plot(xData2, yData2, 'ro')
axarr[0, 0].plot(mesh, [f(x) for x in mesh])
axarr[0, 0].plot(mesh, [evalSpline(xData2, yData2, k2, mesh
[i]) for i in range(len(mesh))], '--')
axarr[0, 0].set_title('n = 2')

# n = 4
axarr[0, 1].plot(xData4, yData4, 'ro')
axarr[0, 1].plot(mesh, [f(x) for x in mesh])
axarr[0, 1].plot(mesh, [evalSpline(xData4, yData4, k4, mesh
```

```
48 [i]) for i in range(len(mesh))], '--')
49 axarr[0, 1].set_title('n = 4')
50
51 # n = 8
52 axarr[1, 0].plot(xData8, yData8, 'ro')
53 axarr[1, 0].plot(mesh, [f(x) for x in mesh])
54 axarr[1, 0].plot(mesh, [evalSpline(xData8, yData8, k8, mesh
   [i]) for i in range(len(mesh))], '--')
55 axarr[1, 0].set_title('n = 8')
56
57 # n = 16
58 axarr[1, 1].plot(xData16, yData16, 'ro')
59 axarr[1, 1].plot(mesh, [f(x) for x in mesh])
60 axarr[1, 1].plot(mesh, [evalSpline(xData16, yData16, k16,
   mesh[i]) for i in range(len(mesh))], '--')
61 axarr[1, 1].set_title('n = 16')
62
63 # Fine-tune figure; hide x ticks for top plots and y ticks
   for right plots
64 plt.setp([a.get_xticklabels() for a in axarr[0, :]],
   visible=False)
65 plt.setp([a.get_yticklabels() for a in axarr[:, 1]],
   visible=False)
66
67 plt.show()
```