MATH 5610/6860: Assignment 4: Jordan Saethre

1. **Spline versus Cubic Hermite Interpolation.** Let the function $s(x)$ be defined by

$$s(x) = \begin{cases} (\gamma-1)(x^3-x^2)+x+1 & \text{if } x \in [0,1] \\ \gamma x^3 - 5\gamma x^2 + 8\gamma x - 4\gamma + 2 & \text{if } x \in [1,2] \end{cases}$$

a. Show that s is the piecewise cubic Hermite interpolant to the data:

$$s(0)=1, \quad s(1)=s(2)=2, \quad s'(0)=1, \quad s'(1)=\gamma, \quad s'(2)=0$$

To verify we plug in the values given: $s(0) = (\gamma-1)(0^3-0^2)+0+1 = 1$.

$$s(1) = \begin{cases} (\gamma-1)(1^3-1^2)+1+1 = 2 & \text{if } x \in [0,1] \\ \gamma(1)^3 - 5\gamma(1)^2 + 8\gamma(1) - 4\gamma + 2 = 2 & \text{if } x \in [1,2] \end{cases}$$

$s(2) = \gamma(2^3) - 5\gamma(2^2) + 8\gamma(2) + 2 = 2$

$$c \begin{cases} 3(\gamma-1)x^2 - 2(\gamma-1)x + 1 & \text{if } x \in [0,1] \\ \gamma 3x^2 - 10\gamma x + 8\gamma & \text{if } x \in [1,2] \end{cases}$$

$s'(0) = 3(\gamma-1)(0)^2 - 2(\gamma-1)(0) + 1 = 1$

$$s'(1) = \begin{cases} 3(\gamma-1)(1)^2 - 2(\gamma-1)(1) + 1 = \gamma & \text{if } x \in [0,1] \\ \gamma 3(1)^2 - 10\gamma(1) + 8\gamma = \gamma & \text{if } x \in [1,2] \end{cases}$$

$s'(2) = \gamma 3(2)^2 - 10\gamma(2) + 8\gamma = 0$

Therefore $s(x)$ is the piecewise cubic Hermite interpolant.

b. For what value of γ does s become a cubic spline?

For $s(x)$ to become a cubic spline we need $s(x)$ to be a third degree polynomial on each subinterval and both $s'(x)$ and $s''(x)$ to be continuous on the entire interval $[0,2]$. This means we need to find a γ such that $s'(1)$ is the same value no matter if you evaluate it in the interval $[0,1]$ or $[1,2]$. Similarly $s''(1)$ must be the same value no matter which interval you evaluate it. Since it was shown above that $s'(1) = \gamma$ the only restriction on γ will come from $s''(x)$.

$$s''(x) = \begin{cases} 6(\gamma-1)x - 2(\gamma-1) = 4\gamma - 4 & \text{if } x \in [0,1] \\ 6\gamma x - 10\gamma = -4\gamma & \text{if } x \in [1,2] \end{cases}$$

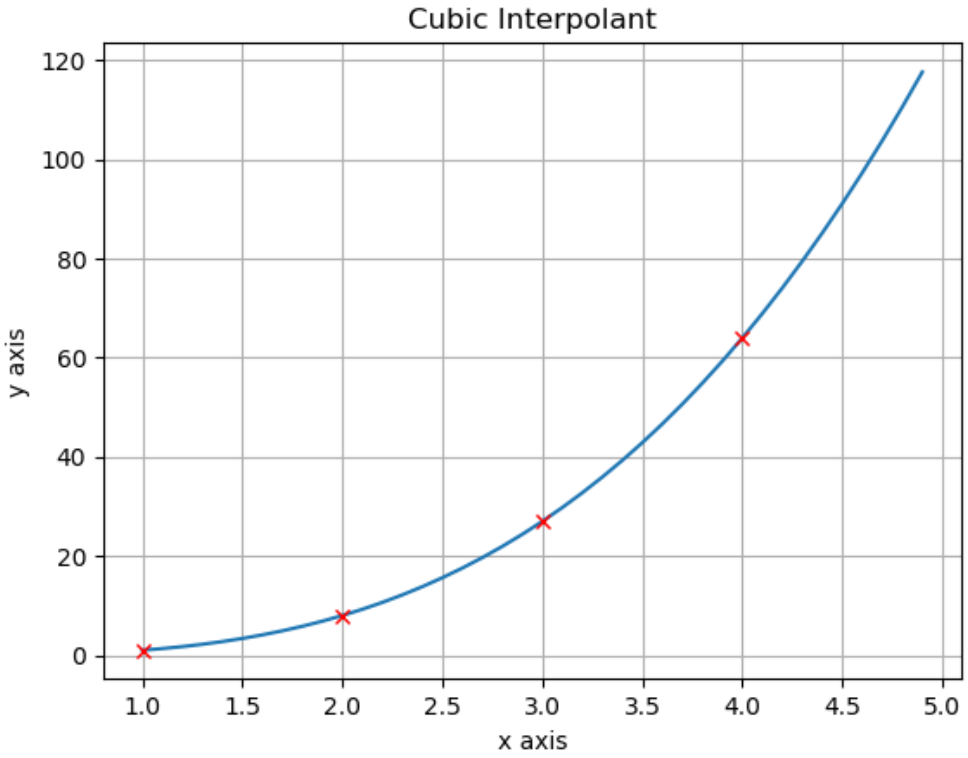
$$4\gamma - 4 = -4\gamma \implies \gamma = \frac{1}{2}$$

2. **More on Cubic Splines** Consider the data

x_i :	1	2	3	4
y_i :	1	8	27	64

a. Find the cubic polynomial p that satisfies $p(x_i) = y_i, \quad i = 1, 2, 3, 4$ and draw its graph.

$$\begin{aligned} p(x) &= (1)\frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)} + (8)\frac{(x-1)(x-3)(x-4)}{(2-1)(2-3)(2-4)} \\ &+(27)\frac{(x-1)(x-2)(x-4)}{(3-1)(3-2)(3-4)} + (64)\frac{(x-1)(x-2)(x-3)}{(4-1)(4-2)(4-3)} \\ &= -\frac{1}{6}(x-2)(x-3)(x-4) + 4(x-1)(x-3)(x-4) \\ &-\frac{27}{2}(x-1)(x-2)(x-4) + \frac{32}{3}(x-1)(x-2)(x-3) \end{aligned}$$



b. Construct the interpolating *natural cubic spline* and draw its graph.

Start by constructing $f''_{i,i+1}(x) = k_i\left(\frac{x-x_{i+1}}{x_i-x_{i+1}}\right) + k_{i+1}\left(\frac{x-x_i}{x_{i+1}-x_i}\right)$. Integrate this function twice to get $f_{i,i+1}(x)$.

$$f_{i,i+1}(x) = \int\int k_i\left(\frac{x-x_{i+1}}{x_i-x_{i+1}}\right) + k_{i+1}\left(\frac{x-x_i}{x_{i+1}-x_i}\right) dx\ dx$$

$$f_{i,i+1}(x) = \int \frac{k_i}{x_i - x_{i+1}} \frac{(x - x_{i+1})^2}{2} + \frac{k_{i+1}}{x_{i+1} - x_i} \frac{(x - x_i)^2}{2} + C \; dx$$

$$f_{i,i+1}(x) = \frac{k_i}{x_i - x_{i+1}} \frac{(x - x_{i+1})^3}{6} + \frac{k_{i+1}}{x_{i+1} - x_i} \frac{(x - x_i)^3}{6} + Cx + D$$

Let $C = A - B$ and $D = Bx_i - Ax_{i+1}$

$$f_{i,i+1}(x) = \frac{k_i}{x_i - x_{i+1}} \frac{(x - x_{i+1})^3}{6} + \frac{k_{i+1}}{x_{i+1} - x_i} \frac{(x - x_i)^3}{6} + A(x - x_{i+1}) - B(x - x_i)$$

Use initial conditions to find A and B:

$$f_{i,i+1}(x_i) = \frac{k_i}{x_i - x_{i+1}} \frac{(x_i - x_{i+1})^3}{6} + A(x - x_{i+1}) = y_i$$

$$\implies A = \left(y_i - \frac{k_i}{x_i - x_{i+1}} \frac{(x_i - x_{i+1})^3}{6} \right) \left(\frac{1}{x_i - x_{i+1}} \right)$$

$$A = \frac{y_i}{x_i - x_{i+1}} - \frac{k_i}{6}(x_i - x_{i+1})$$

Similarly we can find B by using $f_{i,i+1}(x_{i+1}) = y_{i+1}$

$$B = \frac{y_{i+1}}{x_{i+1} - x_i} - \frac{k_{i+1}}{6}(x_{i+1} - x_i)$$

Finally we can put all of the terms together and explicitly write down $f(x)$:

$$f_{i,i+1}(x) = \frac{k_i}{6} \left(\frac{(x - x_{i+1})^3}{x_i - x_{i+1}} - (x - x_{i+1})(x_i - x_{i+1}) \right)$$

$$- \frac{k_{i+1}}{6} \left(\frac{(x - x_i)^3}{x_i - x_{i+1}} - (x - x_i)(x_i - x_{i+1}) \right)$$

$$+ \frac{y_i(x - x_{i+1}) - y_{i+1}(x - x_i)}{x_i - x_{i+1}}$$

We need $f'_{i-1,i}(x_i) = f'_{i,i+1}(x)$. Skipping the arithmetic this yields:

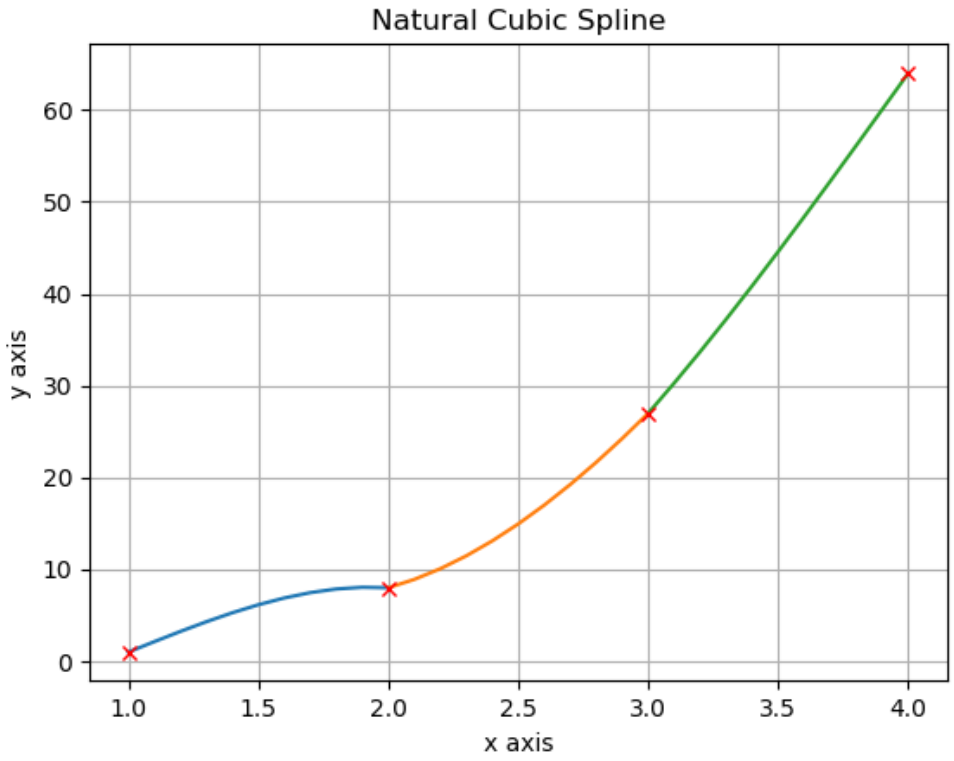
$$k_{i-1}(x_{i-1} - x_i) + 2k_i(x_{i-1} - x_{i+1}) + k_{i+1}(x_i - x_{i+1}) = 6 \left(\frac{y_{i-1} - y_i}{x_{i-1} - x_i} - \frac{y_i - y_{i+1}}{x_i - x_{i+1}} \right)$$

Since this is a *natural* cubic spline we set $k_1 = k_4 = 0$ and solve for k_2 and k_3 using two equations we will construct with our specific given data using the last equation. Plugging in our values we get a system of equations:

$$\begin{cases} 4k_2 + k_3 = 120 \\ k_2 + 4k_3 = 84 \end{cases} \implies k_2 = \frac{132}{5} \quad \text{and} \quad k_3 = \frac{72}{5}$$

All of this leads to our natural cubic spline:

$$f(x) = \begin{cases} -(\frac{132}{5})(\frac{1}{6})(-(x-1)^3 + (x-1)) - (x-2) + 8(x-1) & x \in [1,2] \\ (\frac{132}{5})(\frac{1}{6})(-(x-3)^3 + (x-3)) + (\frac{72}{5})(\frac{1}{6})(-(x-2)^3 + (x-2)) - 4(x-3) + 27(x-2) & x \in [2,3] \\ -(\frac{72}{5})(\frac{1}{6})(-(x-4)^3 + (x-4)) - 27(x-4) + 64(x-3) & x \in [3,4] \end{cases}$$



c. Comment on your results.

Constructing the cubic interpolating polynomial in part (a) was much faster to come up with than the natural cubic spline. The cubic spline is more accurate though because it interpolates the derivatives at the data sites as well.

Code For Interpolant Plots

```
import numpy as np
from matplotlib import pyplot as plt

# Part (a) Construct the Cubic Interpolant

my_poly = lambda x: (-1/6)*(x-2)*(x-3)*(x-4) + 4*(x-1)*(x-3)*(x-4)
                    - (27/2)*(x-1)*(x-2)*(x-4)
                    + (32/3)*(x-1)*(x-2)*(x-3)

```

```
# Plot the function
t = np.arange(1,5,.1)
f = my_poly(t)
x = [1, 2, 3, 4]
y = [1, 8, 27, 64]
plt.title("Cubic Interpolant")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.plot(t,f)
plt.plot(x, y, 'rx')
plt.grid(True)
plt.show()

# Part (b) Construct the Natural Cubic Spline

my_cubic_spline_1_2 = lambda x: ((132/5)/6)*(-(x-1)**3
                                + (x-1)) - (x-2) + 8*(x-1)
my_cubic_spline_2_3 = lambda x: ((132/5)/6)*(-(x-3)**3 + (x-3))
                                - ((72/5)/6)*(-(x-2)**3 + (x-2))
                                - 8*(x-3) + 27*(x-2)
my_cubic_spline_3_4 = lambda x: ((72/5)/6)*(-(x-4)**3
                                + (x-4)) - (27)*(x-4) + 64*(x-3)

# Plot the function
t12 = np.arange(1,2.1,.1)
t23 = np.arange(2,3.1,.1)
t34 = np.arange(3,4.1,.1)
f12 = my_cubic_spline_1_2(t12)
f23 = my_cubic_spline_2_3(t23)
f34 = my_cubic_spline_3_4(t34)
x = [1, 2, 3, 4]
y = [1, 8, 27, 64]
plt.title("Natural Cubic Spline")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.plot(t12,f12)
plt.plot(t23,f23)
plt.plot(t34,f34)
plt.plot(x, y, 'rx')
plt.grid(True)
plt.show()
```

3. **Polynomial Interpolation** Suppose you want to interpolate to the data (x_i, y_i), $i = 0, \ldots, n$ by a polynomial of degree n. Recall that the interpolating polynomial p can be written in its Lagrange form as

$$p(x) = \sum_{i=0}^{n} y_i L_i(x) \quad \text{where} \quad L_i(x) = \frac{\Pi_{i \neq k}(x - x_k)}{\Pi_{i \neq k}(x_i - x_k)}.$$

Show that

$$\sum_{i=0}^{n} x_i^j L_i(x) = x^j \quad \text{for} \quad j = 0, \ldots, n.$$

Proof. Suppose $j = 0$ and $n = 1$. Our Lagrange Polynomial will look like the following:

$$\sum_{i=0}^{n} x_i^j L_i(x) = \frac{x - x_1}{x_0 - x_1} x_0^0 + \frac{x - x_0}{x_1 - x_0} x_1^0 = \frac{x - x_1}{x_0 - x_1} - \frac{x - x_0}{x_0 - x_1} = \frac{x - x_0 - x + x_0}{x_0 - x_1} = 1$$

Now suppose $j = 1$ and $n = 1$:

$$\sum_{i=0}^{n} x_i^j L_i(x) = \frac{x - x_1}{x_0 - x_1} x_0^1 + \frac{x - x_0}{x_1 - x_0} x_1^1 = \frac{x - x_1}{x_0 - x_1} x_0 - \frac{x - x_0}{x_0 - x_1} x_1$$

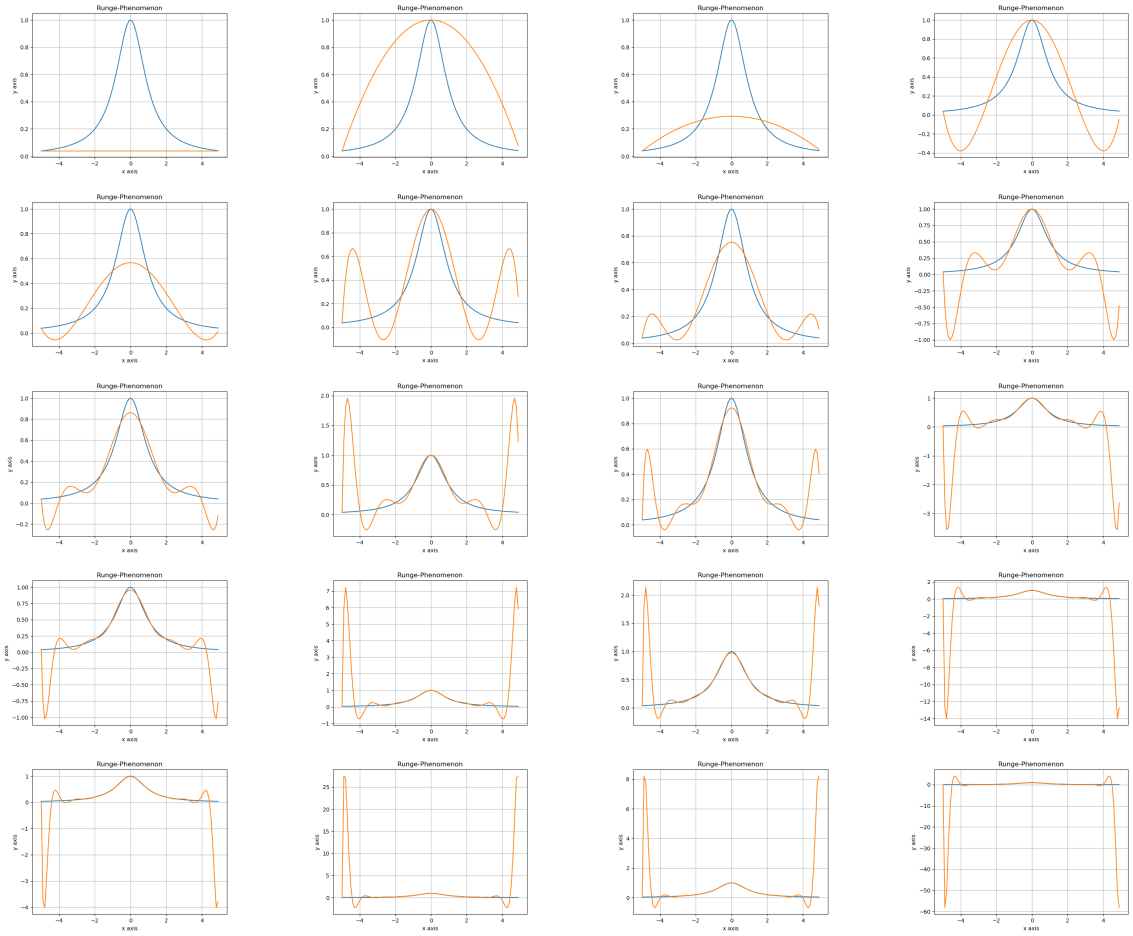
$$= \frac{x x_0 - x_0 x_1 - x x_1 + x_0 x_1}{x_0 - x_1} = \frac{x(x_0 - x_1)}{x_0 - x_1} = x$$

Continuing in this fashion terms always cancel and reduce the polynomial to x^j. For $n > 1$ consider what we did above to simplify the polynomial and generalize the process. We noticed that each term in the polynomial could be combined into one fraction by pulling out a negative sign from the denominator which is what allowed us to cancel terms in the numerator. If $n > 1$ we need to also multiply each term by a special form of 1 in order to get a common denominator. The form of 1 that you will use depends on the missing factor that every other term has which is exactly what was needed in the numerator in order for all terms to cancel except x^j.

□

4. **The infamous Runge-Phenomenon** It is not generally true that higher degree interpolation polynomials yield more accurate approximations. This is illustrated in this problem. Let $f(x) = \frac{1}{1+x^2}$ and $x_j = -5 + jh, \quad j = 0, 1, \cdots, n, \quad h = \frac{10}{n}$. For $n = 1, 2, 3, \cdots, 20$ plot the graph (in the interval [-5,5]) of the interpolant $p(x) = \sum_{i=0}^{n} \alpha_i x^i$ defined by $p(x_i) = f(x_i), \quad i = 0, 1, \cdots, n$.

Plots of Runge-Phenomenon for $n = 1, 2, \cdots 20$



Code For Runge-Phenomenon

```
# MATH 5610 HW 4
# Question 4: The infamous Runge-Phenomenon

import numpy as np
from matplotlib import pyplot as plt
import scipy
from scipy import interpolate

# Define the function that we will interpolate:
my_function = lambda x: 1/(1 + x**2)

```

```
# Get the x_j's
def find_xjs(n):
    xj = np.array([])
    for j in range(0, n + 1):
        xj = np.append(xj, -5 + (10/n)*j)
    return xj

# Get the f_xj's
def find_f_xjs(n):
    f_xj = np.array([])
    for j in find_xjs(n):
        f_xj = np.append(f_xj, my_function(j))
    return f_xj

def find_true_f_xjs():
    f_xj = np.array([])
    for j in np.arange(-5,5,0.1):
        f_xj = np.append(f_xj, my_function(j))
    return f_xj


def plot_poly(n):
    poly_x = np.arange(-5,5,0.1)
    xj = find_xjs(n)
    f_xj = find_f_xjs(n)
    poly = scipy.interpolate.lagrange(xj,f_xj)
    poly_points = np.array([])
    for x in poly_x:
        poly_points = np.append(poly_points, poly(x))
    plt.title("Runge-Phenomenon")
    plt.xlabel("x axis")
    plt.ylabel("y axis")
    f = find_true_f_xjs()
    plt.plot(poly_x, f)
    plt.plot(poly_x,poly_points)
    plt.grid(True)
    plt.show()
```