Homework 5

2/15/22

Jessica Ho

Sections 2.4, 2.5, 3.1

2.4.3 (a, b)

$\boxed{2.4.3}$ (a)

$$\begin{bmatrix} 3 & 7 \\ 6 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -11 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 7 \\ 6 & 1 \end{bmatrix} \underset{A_1, A_2}{\overset{(1)}{\longleftrightarrow}} \begin{bmatrix} 6 & 1 \\ 3 & 7 \end{bmatrix} \underset{A_1 - \frac{1}{2}A_2}{\overset{(3)}{\rightsquigarrow}} \begin{bmatrix} 6 & 1 \\ \frac{1}{2} & \frac{13}{2} \end{bmatrix} \quad \Rightarrow \quad U = \begin{bmatrix} 6 & 1 \\ 0 & \frac{13}{2} \end{bmatrix}, \quad \mathcal{L} = \begin{bmatrix} 1 & 0 \\ 1/2 & 1 \end{bmatrix}$$

$$P_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

check: $PA = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 3 & 7 \\ 6 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 1 \\ 3 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1/2 & 1 \end{bmatrix}\begin{bmatrix} 6 & 1 \\ 0 & 13/2 \end{bmatrix} = \mathcal{L}U$

$\mathcal{L}\vec{c} = \hat{b} \Rightarrow \begin{bmatrix} 1 & 0 \\ 1/2 & 1 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} -11 \\ 1 \end{bmatrix}$ $\qquad c_1 = -11$, $c_2 = 1 - \frac{1}{2}(-11) = \frac{13}{2}$

$\Rightarrow \vec{c} = \begin{bmatrix} -11 \\ 13/2 \end{bmatrix}$

$U\vec{x} = \vec{c} \Rightarrow \begin{bmatrix} 6 & 1 \\ 0 & 13/2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -11 \\ 13/2 \end{bmatrix}$ $\qquad 6x_1 + x_2 = -11 \Rightarrow 6x_1 = -11 - (1) = -12 \Rightarrow x_1 = -2$

$\frac{13}{2}x_2 = \frac{13}{2} \Rightarrow x_2 = 1$

$\Rightarrow \vec{x} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$

$\boxed{2.4.3}$ (b)

$$\begin{bmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{bmatrix} \underset{A_1, A_2}{\overset{(1)}{\rightsquigarrow}} \begin{bmatrix} 6 & 3 & 4 \\ 3 & 1 & 2 \\ 3 & 1 & 5 \end{bmatrix} \underset{\substack{A_2 - \frac{1}{2}A_1 \\ A_3 - \frac{1}{2}A_1}}{\overset{(3)}{\rightsquigarrow}} \begin{bmatrix} 6 & 3 & 4 \\ 1/2 & -1/2 & 0 \\ 1/2 & -1/2 & 3 \end{bmatrix} \underset{A_3 - A_2}{\overset{(3)}{\rightsquigarrow}} \begin{bmatrix} 6 & 3 & 4 \\ 1/2 & -1/2 & 0 \\ 1/2 & 1 & 3 \end{bmatrix}$$

$$P_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$L\vec{c} = \hat{\vec{b}}$:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/2 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}$$

$c_1 = 1$
$c_2 = -\frac{1}{2}$
$c_3 = 3 - \frac{1}{2}(1) - (-\frac{1}{2}) = 3 - \frac{1}{2} + \frac{1}{2} = 3$

$$\Rightarrow \vec{c} = \begin{bmatrix} 1 \\ -1/2 \\ 3 \end{bmatrix}$$

$U\vec{x} = \vec{c}$:

$$\begin{bmatrix} 6 & 3 & 4 \\ 0 & -1/2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1/2 \\ 3 \end{bmatrix}$$

$6x_1 = 1 - 3(1) - 4(1) = -6 \Rightarrow x_1 = -1$
$x_2 = 1$
$x_3 = 1$

$$\Rightarrow \vec{x} = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}$$

In-Depth Problem: 2.4.6

Suppose we are looking for the 4×4 matrix P such that left multiplying a matrix A on the left by P causes its resulting matrix's second and fourth rows to be exchanged. Then, by matrix multiplication we observe that **P** is

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

However, if we investigate the effect of multiplying on the **right** by **P** we notice that the second and fourth columns of the resulting matrix are exchanged.

For example, supposed we have a matrix A,

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 3 & 5 \\ 2 & 1 & 2 & 1 \\ 3 & 2 & 2 & 1 \\ 0 & 1 & 1 & 3 \end{bmatrix}.$$

Then, left multiplication by **P** is

$$\mathbf{PA} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 & 5 \\ 2 & 1 & 2 & 1 \\ 3 & 2 & 2 & 1 \\ 0 & 1 & 1 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 & 5 \\ 0 & 1 & 1 & 3 \\ 3 & 2 & 2 & 1 \\ 2 & 1 & 2 & 1 \end{bmatrix}.$$

In contrast, right multiplication by **P** is

$$\mathbf{AP} = \begin{bmatrix} 1 & 0 & 3 & 5 \\ 2 & 1 & 2 & 1 \\ 3 & 2 & 2 & 1 \\ 0 & 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 3 & 0 \\ 2 & 1 & 2 & 1 \\ 3 & 1 & 2 & 2 \\ 0 & 3 & 1 & 1 \end{bmatrix}.$$

This is just as we expect by rules of matrix multiplication.

2.4.7

Given the following equation,

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 5 & 6 \\ 5 & 6 & 7 & 8 \\ 7 & 8 & 9 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 6 & 7 & 8 \\ 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix}.$$

To make the equation correct, the leftmost matrix must be

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

This is because the 1st and 4th rows need to switched. Followed by the 3rd row and new 4th row (original 1st row) need to be switched.

2.5.1 (a, b, c)

$\boxed{2.5.1}$ (a) $\begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$ $\quad$ $3u - v = 5$, $\; -u + 2v = 4$

Initial Condition: $\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

Jacobi:

$$u_i = \frac{5 + v_{i-1}}{3} \qquad v_i = \frac{4 + u_{i-1}}{2}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5+0}{3} \\ \frac{4+0}{2} \end{bmatrix} = \begin{bmatrix} 5/3 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5+2}{3} \\ \frac{4+5/3}{2} \end{bmatrix} = \begin{bmatrix} 7/3 \\ 17/6 \end{bmatrix}$$

Gauss-Seidel

$$u_i = \frac{5 + v_{i-1}}{3} \qquad v_i = \frac{4 + u_i}{2}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5+0}{3} = 5/3 \\ \frac{4+5/3}{2} \end{bmatrix} = \begin{bmatrix} 5/3 \\ 17/6 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5+17/6}{3} \\ \frac{4+47/18}{2} \end{bmatrix} = \begin{bmatrix} 47/18 \\ 119/36 \end{bmatrix}$$

(b) $\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$

$$2u - v = 0$$
$$-u + 2v - w = 2$$
$$-v + 2w = 0$$

Initial Condition:

$$\begin{bmatrix} u_0 \\ v_0 \\ w_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Jacobi:

$$u_i = \frac{v_{i-1}}{2}$$
$$v_i = \frac{2 + u_{i-1} + w_{i-1}}{2}$$
$$w_i = \frac{v_{i-1}}{2}$$

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 + \frac{0}{2} \\ 2 + \frac{0+0}{2} \\ 0/2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ \frac{2+0+0}{2} \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1 \\ 1/2 \end{bmatrix}$$

Gauss-Seidel:

$$u_i = \frac{v_{i-1}}{2}$$
$$v_i = \frac{2 + u_i + w_{i-1}}{2}$$
$$w_i = \frac{v_i}{2}$$

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{2+0+0}{2} \\ 1/2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1/2 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ \frac{2 + 1/2 + 1/2}{2} = 6/4 \\ 6/4/2 = 6/8 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 3/2 \\ 3/4 \end{bmatrix}$$

2.5.1

(c)

$$\begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 5 \end{bmatrix}$$

$$3u + v + w = 6$$
$$u + 3v + w = 3$$
$$u + v + 3w = 5$$

Initial Condition:

$$\begin{bmatrix} u_0 \\ v_0 \\ w_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Jacobi:

$$u_i = \frac{6 - v_{i-1} - w_{i-1}}{3}$$

$$v_i = \frac{3 - u_{i-1} - w_{i-1}}{3}$$

$$w_i = \frac{5 - u_{i-1} - v_{i-1}}{3}$$

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} \frac{6-0-0}{3} \\ \frac{3-0-0}{3} \\ \frac{5-0-0}{3} \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 5/3 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{6-1-5/3}{3} \\ \frac{3-2-5/3}{3} \\ \frac{5-2-1}{3} \end{bmatrix} = \begin{bmatrix} \frac{18-3-5}{9} \\ \frac{9-6-5}{9} \\ 2/3 \end{bmatrix} = \begin{bmatrix} 10/9 \\ -2/3 \\ 2/3 \end{bmatrix}$$

Gauss-Seidel:

$$u_i = \frac{v_{i-1}}{2}$$

$$v_i = \frac{2 + u_i + w_{i-1}}{2}$$

$$w_i = \frac{v_i}{2}$$

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 6/3 = 2 \\ \frac{3-2-0}{3} = 1/3 \\ \frac{5-2-1/3}{3} \end{bmatrix} = \begin{bmatrix} 2 \\ 1/3 \\ \frac{15-6-1}{9} \end{bmatrix} = \begin{bmatrix} 2 \\ 1/3 \\ 8/9 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{6-1/3-8/9}{3} = \frac{54-3-8}{27} = \frac{43}{27} \\ \frac{3-43/27-8/9}{3} = \frac{81-43-24}{81} = \frac{14}{81} \\ \frac{5-43/27-14/81}{3} = \frac{405-129-14}{243} \end{bmatrix} = \begin{bmatrix} 43/27 \\ 14/81 \\ \frac{352}{243} \end{bmatrix}$$

3.1.7

Find P(0), where P(x) is degree 10 polynomial that's zero at x=1,...., 10 and satisfies P(12) = 44. By definition, P(x) pass through (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), (7,0), (8,0), (9,0), (10,0), and (12, 44) because we are given y at each x. Therefore,

$$\begin{aligned}P(x) &= y_1 L_{1,1}(x) + \cdots + y_{10} L_{10,1}(x) + y_{12} L_{12,1}(x)\\ &= 0 + \cdots + 44\left(\frac{(x-1)(x-2)(x-3)\ldots(x-10)}{(12-1)(12-2)(12-3)\ldots(12-10)}\right)\\ &= 44\left(\frac{(x-1)(x-2)(x-3)\ldots(x-10)}{(12-1)(12-2)(12-3)\ldots(12-10)}\right)\end{aligned}$$

For $x = 0:$

$$\begin{aligned}P(0) &= 44\left(\frac{(0-1)(0-2)(0-3)\ldots(0-10)}{(11)(10)(9)\ldots(2)}\right)\\ &= \frac{44}{11}\\ &= 4\end{aligned}$$

3.1.12

Can a degree 3 polynomial intersect a degree 4 polynomial in exactly 5 points? Explain.
No, this is not possible. Using the rule, $0 \leq d \leq n-1$, we know that a degree 4 polynomial can pass through ≤ 4 points, and a degree 3 polynomial can pass through ≤ 4 points. Each one individually could potentially pass through 5 points. However, because the interpolation polynomial is unique, there is only one polynomial of degree ≤ 4 can pass through 5 different points. So it is impossible for both a cubic (degree 3) and quartic (degree 4) polynomial to intersect 5 points.

Extra Problem

Extra Problem

Points ordered: (2,2),(0,1),(3,4),(1,0)

$$
\begin{array}{c|ccccccc}
2 & 2 & & & & & & \\
 & & \frac{1}{2} & & & & & \\
0 & 1 & & \frac{1}{2} & & & & \\
 & & 1 & & -\frac{1}{2} & & & \\
3 & 4 & & 1 & & & & \\
 & & 2 & & & & & \\
1 & 0 & & & & & &
\end{array}
$$

Calculations:

Layer 1: $\frac{1-2}{0-2} = \frac{-1}{-2} = \frac{1}{2}$

$\frac{4-1}{3-0} = \frac{3}{3} = 1$

$\frac{0-4}{1-3} = \frac{-4}{-2} = 2$

Layer 2: $\frac{1-\frac{1}{2}}{3-2} = \frac{\frac{1}{2}}{1} = \frac{1}{2}$

$\frac{2-1}{1-0} = \frac{1}{1} = 1$

Layer 3: $\frac{1-\frac{1}{2}}{1-2} = \frac{\frac{1}{2}}{-1} = \frac{-1}{2}$

$$
\begin{aligned}
P_3(x) &= 2 + \tfrac{1}{2}(x-2) + \tfrac{1}{2}(x-2)(x-0) - \tfrac{1}{2}(x-2)(x-0)(x-3) \\
&= 2 + \tfrac{1}{2}x - 1 + \tfrac{1}{2}(x-2)(x-0) - \tfrac{1}{2}(x-2)(x-0)(x-3) \\
&= 1 + \tfrac{1}{2}x + \tfrac{1}{2}(x-2)(x-0) - \tfrac{1}{2}(x-2)(x-0)(x-3)
\end{aligned}
$$

Observe this is the same result as the points ordered as (0,1),(2,2),(3,4),(1,0). The divided differences triangle here is

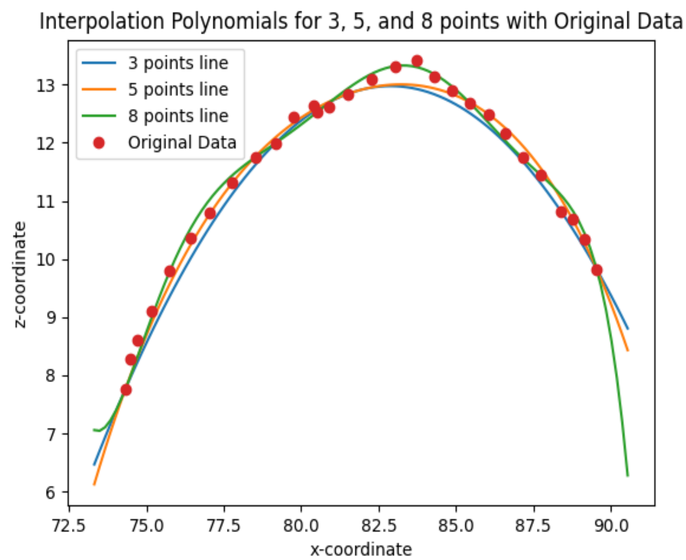
$$
\begin{array}{c|ccccccc}
0 & 1 & & & & & & \\
 & & \frac{1}{2} & & & & & \\
2 & 2 & & \frac{1}{2} & & & & \\
 & & 2 & & -\frac{1}{2} & & & \\
3 & 4 & & 0 & & & & \\
 & & 2 & & & & & \\
1 & 0 & & & & & &
\end{array}
$$

$$P_3(x) = 1 + \tfrac{1}{2}(x-0) + \tfrac{1}{2}(x-0)(x-2) - \tfrac{1}{2}(x-0)(x-2)(x-3)$$

Thus, the newly order and old order achieve the same result.

Computer Problems

Bullet 1: Interpolation Polynomials

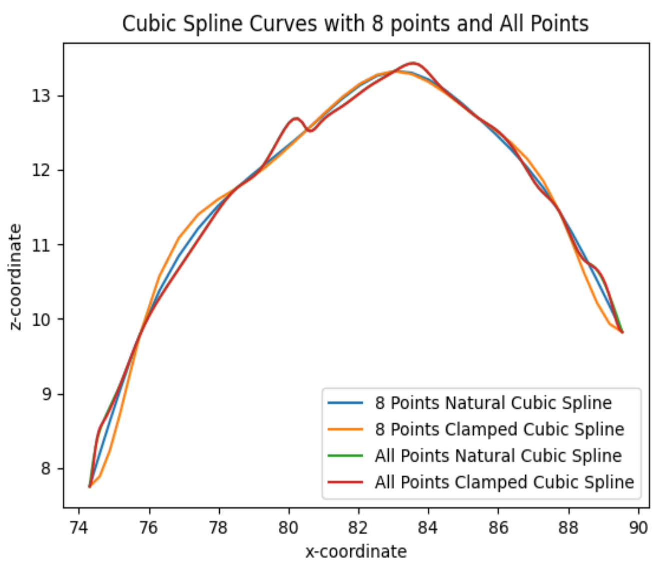



```
import numpy as np
import matplotlib.pyplot as plt
from HW4_ProgrammingProblems import gaussianElimination

# Part 1
# Program 3.1 Newton Divided Difference Interpolation Method
↪  #Computes coefficients of interpolating polynomial
# Input: x and z are vectors containing the x and z coordinates #
↪  of the n data points
# Output: coefficients c of interpolating polynomial in nested
↪  form #Use with nest.m to evaluate interpolating polznomial
def newtdd(a,b,n):      # a = x, b = z
    v = np.zeros((n,n))        # initialize matrix
    c = np.zeros(n)     # passing tuple
    for j in range(n):
        v[j][0]=b[j]            # Fill in z column of Newton
        ↪  triangle
    for i in range(1,n):      # For column i,
        for j in range(n-i):    # fill in column from top to
        ↪  bottom
            v[j][i]=(v[j+1][i-1]-v[j][i-1])/(a[j+i]-a[j])
```

```
    for i in range(n):
        c[i]=v[0][i]        # Read along top of triangle
    return c


#Program 0.1 Nested multiplication
#Evaluates polynomial from nested form using Horner's Method
↪  #Input: degree d of polynomial,
#       array of d+1 coefficients c (constant term first),
# x-coordinate x at which to evaluate, and
# array of d base points b, if needed #Output: value z of
↪  polynomial at x
### compute the coeff, compute polynomial,
def nest(c,a,x):            # c = coeff, a = output base
↪  coordinates, x= unknown input
    d = len(c)-1
    z = c[d]
    for i in range(d-1,-1,-1):
        z = z * (x-a[i])+c[i]
    return z


if __name__ == "__main__":

    # Inputs
    des_cols = (0, 2)
    total_rows = 29
    file_name = "shots.txt"
    num_pts_list = [3,5,8]
    num_x_plot = 100

    # Original Data Points
    load_data = np.loadtxt(file_name, usecols=(0, 2))
    desired_lines = load_data[0:29]           # All 29 points
    xOD = desired_lines[:, 0].tolist()
    zOD = desired_lines[:, 1].tolist()

    # Final Version
    for num_pts in num_pts_list:
        deln = int((total_rows - 1) / (num_pts - 1))
        load_data_3 = np.loadtxt(file_name, usecols=(0, 2))
        desired_lines = load_data_3[0:total_rows:deln]
        x = desired_lines[:, 0].tolist()
        z = desired_lines[:, 1].tolist()
        c = newtdd(x, z, num_pts)
        inputs = np.linspace(min(x) - 1, max(x) + 1,
```

```
                             num_x_plot)  # array of numbers from
                             ↪  0 to 100 that are evenly spaced
        outputs = np.zeros(100)
        for ind in range(len(inputs)):
            outputs[ind] = nest(c, x, inputs[ind])
        plt.figure(1)
        plt.xlabel('x-coordinate')
        plt.zlabel('z-coordinate')
        plt.plot(inputs, outputs, label = f'{num_pts} points
        ↪  line')
    plt.plot(xOD, zOD, 'o', label = 'Original Data')
    plt.legend(loc = 'best')
    plt.title('Interpolation Polynomials for 3, 5, and 8 points
    ↪  with Original Data')
```

Bullet 2: Cubic Splines



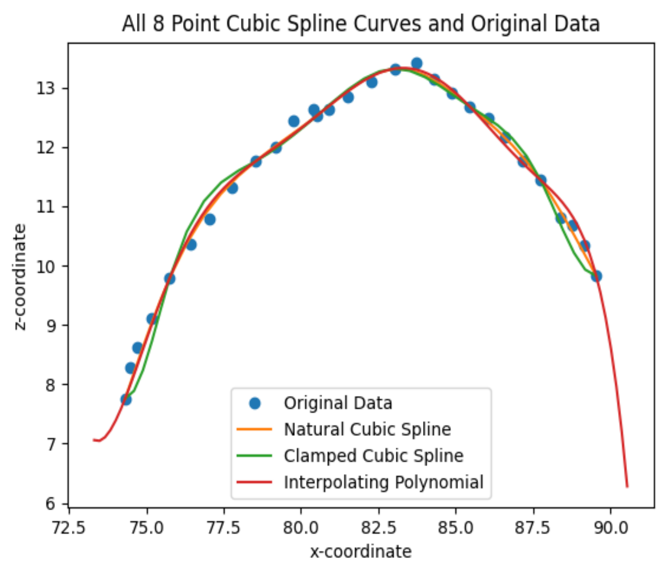

```
# Part 2: Natural Cubic Spline
# Program 3.5 Calculation of spline coefficients
# Calculates coefficients of cubic spline
# Input: x,y vectors of data points
# plus two optional extra data v1, vn
#Output: matrix of coefficients b1,c1,d1 b2,c2,d2 ...
def splinecoeff(x,z,splinetype):
    n=len(x)
```

```
    v1=0
    vn=0
    dx = np.zeros((n-1,1))
    dz = np.zeros((n-1,1))
    A=np.zeros((n,n))        # matrix A is nxn
    r=np.zeros((n,1))        # nx1
    for i in range(0,n-1):      # define the deltas
        dx[i]= x[i+1]-x[i]
        dz[i]=z[i+1]-z[i]
    for i in range(1, n-1):   # load the A matrix
        A[i,i-1:i+2]=[dx[i-1], 2*(dx[i-1]+dx[i]), dx[i]]
        r[i]=3*(dz[i]/dx[i]-dz[i-1]/dx[i-1])

# Set endpoint conditions
# Use only one of following 5 pairs:
    if splinetype == 'natural':
        A[0][0] = 1 # natural spline conditions
        A[n-1][n-1] = 1
    #A(1,1)=2
    # r(1)=v1       # curvature-adj conditions
    #A(n,n)=2 r(n)=vn
    if splinetype == 'clamped':
        A[0][0:2]=[2*dx[0], dx[0]]
        r[0]=3*(dz[0]/dx[0]-v1)     #clamped
        A[n-1][n-2:n]=[dx[n-2], 2*dx[n-2]]
        r[n-1]=3*(vn-dz[n-2]/dx[n-2])
    #A(1,1:2)=[1 -1]        # parabol-term conditions, for n>=3
    #A(n,n-1:n)=[1 -1]
    #A(1,1:3)=[dx(2) -(dx(1)+dx(2)) dx(1)] # not-a-knot, for n>=4
    #A(n,n-2:n)=[dx(n-1) -(dx(n-2)+dx(n-1)) dx(n-2)]
    coeff=np.zeros((n,3))
    #print('This is A, r', A, r)
    coeff[:,1]= gaussianElimination(A, r)       # solve for c
    ↪  coefficients
    for i in range(n-1):          # solve for b and d
        coeff[i][2]=(coeff[i+1][1]-coeff[i][1])/(3*dx[i])

        ↪  coeff[i][0]=dz[i]/dx[i]-dx[i]*(2*coeff[i][1]+coeff[i+1][1])/3
    coeff=coeff[0:n-1,:]
    return coeff



# Program 3.6 Cubic spline plot
# Computes and plots spline from data points
```

```
# Input: x,y vectors of data points, number k of plotted points
↪  per segment
# Output: x1, y1 spline values at plotted points
def splineplot(x,z,k, splinetype):
    n=len(x)
    coeff=splinecoeff(x,z, splinetype)
    x1=np.array([], dtype = np.float64)
    z1=np.array([], dtype = np.float64)
    for i in range(n-1):
        xs=np.linspace(x[i],x[i+1],k+1)
        dx=xs-x[i]
        zs=coeff[i][2]*dx        #  evaluate using nested
         ↪  multiplication
        zs=(zs+coeff[i][1]) *dx
        zs=(zs+coeff[i][0]) *dx+y[i]
        x1 = np.append(x1, np.transpose(xs[0:k]))
        y1 = np.append(y1, np.transpose(zs[0:k]))
        print('x1', x1)
        print()
        print(z1)
    x1=np.append(x1,  x[n-1])
    z1=np.append(z1, z[n-1])
    print('x1,y1', x1, z1)
    return x1, z1

if __name__ == "__main__":
    # Cubic Spline Calls
    k = 5
    x1n, z1n = splineplot(xOD8, zOD8, k, 'natural')
     ↪  #natural 8 points
    x1c, z1c = splineplot(xOD8, zOD8, k, 'clamped')       #
     ↪  clamped 8 points
    x1nOD, z1nOD = splineplot(xOD, zOD, k, 'natural')  # natural
     ↪  All data
    x1cOD, z1cOD = splineplot(xOD, zOD, k, 'clamped')  # clamped
     ↪  all data

    plt.figure(2)
    plt.xlabel('x-coordinate')
    plt.zlabel('z-coordinate')
    plt.plot(x1n,z1n, label='8 Points Natural Cubic Spline')
     ↪  # 8 natural spline
    plt.plot(x1c,z1c, label='8 Points Clamped Cubic Spline')
     ↪  # 8 clamped
    plt.plot(x1nOD,z1nOD, label='All Points Natural Cubic
     ↪  Spline')            # All points natural spline
```

```
plt.plot(x1cOD,z1cOD, label='All Points Clamped Cubic
↪  Spline')            # All Points clamped
plt.legend(loc='best')
plt.title('Cubic Spline Curves with 8 points and All Points')
```

Bullet 3: All 8-Point Curves




```
# Original Data Points
load_data = np.loadtxt(file_name, usecols=(0, 2))
desired_lines = load_data[0:29]           # All 29 points
xOD = desired_lines[:, 0].tolist()
zOD = desired_lines[:, 1].tolist()

# Interpolating Polynomial for 8 points
del8 = int(28/7)                         # specifically for 8
↪  points
load_data_8 = np.loadtxt(file_name, usecols=(0, 2))
desired_lines8 = load_data_8[0:29:del8]           #
↪  specifically for 8 points
xOD8 = desired_lines8[:, 0].tolist()
zOD8 = desired_lines8[:, 1].tolist()
c = newtdd(xOD8, zOD8, 8)
inputs = np.linspace(min(xOD8) - 1, max(xOD8) + 1,
```

```
                     num_x_plot)  # array of numbers from 0
                     ↪  to 100 that are evenly spaced
outputs = np.zeros(100)
for ind in range(len(inputs)):
    outputs[ind] = nest(c, xOD8, inputs[ind])

# Cubic Spline Calls
k = 5
x1n, z1n = splineplot(xOD8, zOD8, k, 'natural')
↪  #natural 8 points
x1c, z1c = splineplot(xOD8, zOD8, k, 'clamped')       #
↪  clamped 8 points
x1nOD, z1nOD = splineplot(xOD, zOD, k, 'natural')  # natural
↪  All data
x1cOD, z1cOD = splineplot(xOD, zOD, k, 'clamped')  # clamped
↪  all data

# Plotting
plt.figure(3)
plt.xlabel('x-coordinate')
plt.zlabel('z-coordinate')
plt.plot(xOD,zOD,'o', label = 'Original Data')
↪  # real data
plt.plot(x1n,z1n, label='Natural Cubic Spline')             #
↪  8 natural spline
plt.plot(x1c,z1c, label='Clamped Cubic Spline')             #
↪  8 clamped
plt.plot(inputs, outputs, label='Interpolating Polynomial')
plt.legend(loc='best')
plt.title('All 8 Point Cubic Spline Curves and Original
↪  Data')
plt.show()
```