# M 348 HOMEWORK 7

ZUN CAO

## 1. Chapter 3.1 Ex # 1, 2 (b)(c)

## 2. Chapter 3.1 Ex # 4

## 3. Additional Problem 1:

*Date*: 3/27/2024.

M348 HW7

3.1 Ex #1&2 (b),(C).

(b) (-1,0), (2,1), (3,1), (5,2)

① Lagrange interpolation $P_3(x) = 0L_1(x) + 1L_2(x) + 1L_3(x) + 2L_4(x)$ (n=4)

$L_1(x) = \frac{(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)}$ $L_2(x) = \frac{(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)}$

$L_3(x) = \frac{(x-x_1)(x-x_2)(x-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)}$ $L_4(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)}$

Plug in numbers, we get:

$$L_1(x) = \frac{(x-2)(x-3)(x-5)}{(-1-2)(-1-3)(-1-5)} = \frac{(x-2)(x-3)(x-5)}{-72}$$

$$L_2(x) = \frac{(x-(-1))(x-3)(x-5)}{(2-(-1))(2-3)(2-5)} = \frac{(x+1)(x-3)(x-5)}{9}$$

$$L_3(x) = \frac{(x-(-1))(x-2)(x-5)}{(3-(-1))(3-2)(3-5)} = \frac{(x+1)(x-2)(x-5)}{-8}$$

$$L_4(x) = \frac{(x-(-1))(x-2)(x-3)}{(5-(-1))(5-2)(5-3)} = \frac{(x+1)(x-2)(x-3)}{36}$$

$$\therefore P_3(x) = 0 + \frac{(x+1)(x-3)(x-5)}{9} + \frac{(x+1)(x-2)(x-5)}{(-8)} + 2\left(\frac{(x+1)(x-2)(x-3)}{36}\right)$$

$$P_3(x) = \frac{8(x+1)(x-3)(x-5) - 9(x+1)(x-2)(x-5) + 4(x+1)(x-2)(x-3)}{72}$$

$$P_3(x) = \frac{3x^3 - 18x^2 + 33x + 54}{72} = \frac{x^3 - 6x^2 + 11x + 18}{24} = \boxed{\frac{x^3}{24} - \frac{x^2}{4} + \frac{11x}{24} + \frac{3}{4}}$$

② Newton Divided Difference

| | | | | |
|---|---|---|---|---|
| -1 | 0 | | | |
| | | $\frac{1-0}{2-(-1)} = \frac{1}{3}$ | | |
| 2 | 1 | | $\frac{0-\frac{1}{3}}{3-(-1)} = -\frac{1}{12}$ | |
| | | $\frac{1-1}{3-2} = 0$ | | $\frac{\frac{1}{6}-(-\frac{1}{12})}{5-(-1)} = \frac{1}{24}$ |
| 3 | 1 | | $\frac{\frac{1}{2}-0}{5-2} = \frac{1}{6}$ | |
| | | $\frac{2-1}{5-3} = \frac{1}{2}$ | | |
| 5 | 2 | | | |

$$P(x) = 0 + \frac{1}{3}(x-(-1)) - \frac{1}{12}(x-(-1))(x-2) + \frac{1}{24}(x-(-1))(x-2)(x-3)$$

$$P(x) = \frac{1}{3}(x+1) - \frac{1}{12}(x+1)(x-2) + \frac{1}{24}(x+1)(x-2)(x-3)$$

$$P(x) = \boxed{\frac{x^3}{24} - \frac{x^2}{4} + \frac{11x}{24} + \frac{3}{4}}$$ (same as previous)

After checking, we see the two methods generate the same polynomial $P(x) = \frac{x^3}{24} - \frac{x^2}{4} + \frac{11x}{24} + \frac{3}{4}$

Therefore, agreement verfied.

(c) (0,−2), (2,1), (4,4)

① Method 1 = Lagrange Interpolation

$$P_2(X) = -2L_1(X) + 1L_2(X) + 4L_3(X)$$

$$L_1(X) = \frac{(X-X_2)(X-X_3)}{(X_1-X_2)(X_1-X_3)} = \frac{(X-2)(X-4)}{(0-2)(0-4)} = \frac{(X-2)(X-4)}{8}$$

$$L_2(X) = \frac{(X-X_1)(X-X_3)}{(X_2-X_1)(X_2-X_3)} = \frac{(X-0)(X-4)}{(2-0)(2-4)} = \frac{X(X-4)}{-4}$$

$$L_3(X) = \frac{(X-X_1)(X-X_2)}{(X_3-X_1)(X_3-X_2)} = \frac{(X-0)(X-2)}{(4-0)(4-2)} = \frac{X(X-2)}{8}$$

$$P_2(X) = -2\left(\frac{(X-2)(X-4)}{8}\right) + \frac{X^2-4X}{(-4)} + 4\left(\frac{X^2-2X}{8}\right)$$

$$P_2(X) = -\frac{X^2-6X+8}{4} - \frac{X^2-4X}{4} + \frac{X^2-2X}{2}$$

$$P_2(X) = \frac{-(X^2-6X+8)-(X^2-4X)+2(X^2-2X)}{4}$$

$$\boxed{P_2(X) = \frac{3}{2}X - 2}$$

② Method 2 = Newton's Divided Difference

| | |
|---|---|
| 0 | −2 |
| 2 | 1 |
| 4 | 4 |

$$\frac{1-(-2)}{2-0} = \frac{3}{2}, \quad \frac{4-1}{4-2} = \frac{3}{2}, \quad \frac{\frac{3}{2}-\frac{3}{2}}{4-0} = 0$$

$$P(X) = -2 + \frac{3}{2}(X-0) + 0(X-0)(X-2)$$

$$\boxed{P(X) = \frac{3}{2}X - 2}$$

After checking, we find the two methods generate the same polynomial $P(X) = \frac{3}{2}X - 2$.
Therefore, agreement verified.

3.1 Ex 4.

(a) We can use Newton's divided difference to find the polynomial.

$$
\begin{array}{c|c|c|c|c}
0 & 0 & & & \\
 & & \frac{1-0}{1-0}=1 & & \\
1 & 1 & & \frac{1-1}{2-0}=0 & \\
 & & \frac{2-1}{2-1}=1 & & \frac{2-0}{3-0}=\frac{2}{3} \\
2 & 2 & & \frac{5-1}{3-1}=2 & \\
 & & \frac{7-2}{3-2}=5 & & \\
3 & 7 & & &
\end{array}
$$

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

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

$\boxed{P_3(x)=\frac{2}{3}x^3-2x^2+\frac{7}{3}x}$

→ b/c we already have 4 points.

(b) Since we want to find polynomial more than degree 3, we can add extra points.

Polynomial 1 — Add (4,3). Therefore ↓

$$
\begin{array}{c|c|c|c|c|c}
0 & 0 & & & & \\
 & & \frac{1-0}{1-0}=1 & & & \\
1 & 1 & & \frac{1-1}{2-0}=0 & & \\
 & & \frac{2-1}{2-1}=1 & & \frac{2-0}{3-0}=\frac{2}{3} & \\
2 & 2 & & \frac{5-1}{3-1}=2 & & \frac{-\frac{13}{6}-\frac{2}{3}}{4-0}=-\frac{17}{24} \\
 & & \frac{7-2}{3-2}=5 & & \frac{-\frac{9}{2}-2}{4-1}=-\frac{13}{6} & \\
3 & 7 & & \frac{-4-5}{4-2}=-\frac{9}{2} & & \\
 & & \frac{3-7}{4-3}=-4 & & & \\
4 & 3 & & & &
\end{array}
$$

$P_4(x)=P_3(x)+(-\frac{17}{24})(x-0)(x-1)(x-2)(x-3)$

$P_4(x)=\frac{2}{3}x^3-2x^2+\frac{7}{3}x-\frac{17}{24}x(x-1)(x-2)(x-3)$

By calculator we get → $\boxed{P_4(x)=-\frac{17}{24}x^4+\frac{59}{12}x^3-\frac{235}{24}x^2+\frac{79}{12}x}$

Polynomial 2 — Add (5,9). Therefore ↓

$$
\begin{array}{c|c|c|c|c|c}
0 & 0 & & & & \\
 & & \frac{1-0}{1-0}=1 & & & \\
1 & 1 & & \frac{1-1}{2-0}=0 & & \\
 & & \frac{2-1}{2-1}=1 & & \frac{2-0}{3-0}=\frac{2}{3} & \\
2 & 2 & & \frac{5-1}{3-1}=2 & & \frac{-\frac{5}{6}-\frac{2}{3}}{5-0}=-\frac{3}{10} \\
 & & \frac{7-2}{3-2}=5 & & \frac{-\frac{4}{3}-2}{5-1}=-\frac{5}{6} & \\
3 & 7 & & \frac{1-5}{5-2}=-\frac{4}{3} & & \\
 & & \frac{9-7}{5-3}=1 & & & \\
5 & 9 & & & &
\end{array}
$$

$P_4(x)=P_3(x)-\frac{3}{10}(x-0)(x-1)(x-2)(x-3)$

$P_4(x)=\frac{2}{3}x^3-2x^2+\frac{7}{3}x-\frac{3}{10}x(x-1)(x-2)(x-3)$

By calculator we get → $\boxed{P_4(x)=-\frac{3}{10}x^4+\frac{37}{15}x^3-\frac{53}{10}x^2+\frac{62}{15}x}$

(c) No. Not exist. 4 points $\to n=4$ $0 \le 3 \le 4-1$, so $P_3(x)$ is unique.

Since $P_3(x)=\frac{2}{3}x^3-2x^2+\frac{7}{3}x$ is the unique cubic polynomial which pass through the first three points, we can plug in $x=4$ to see whether $P_3(x)=2$ to decide whether there exist a polynomial of degree 3 or less that pass through these 4 points.

$P_3(4)=\frac{2}{3}(4)^3-2(4)^2+\frac{7}{3}\cdot 4=\frac{128}{3}-32+\frac{28}{3}=20 \neq 2$

Since the only possible polynomial fails, there is no such polynomial that pass through these 4 pts. $\square$

Additional Problem1.

(a) $|2^x-P_4(x)| \le \frac{(x-0)(x-0.1)(x-0.2)(x-0.3)(x-0.4)}{5!} f^{(5)}(c)$

$f^{(5)}(c)=2^c \cdot \ln^5(2)$, $0<c<0.4$, $\|f^{(5)}(c)\| \le 2^{0.4}\ln^5(2)$ on $[0,0.4]$

$\therefore |2^x-P_4(x)| \le \frac{(x-0)(x-0.1)(x-0.2)(x-0.3)(x-0.4)}{5!} 2^{0.4}\ln^5(2)$

At $x=0.05$:

According to calculator

$|2^{0.05}-P_4(0.05)| \le \frac{0.05(0.05-0.1)(0.05-0.2)(0.05-0.3)(0.05-0.4)}{5!} 2^{0.4}\ln^5(2) \approx 5.77294\times 10^{-8}$

At $x=0.25$

According to Calculator

$f^{(5)}(0.4)=2^{0.4}\ln^5(2) \approx 0.211125$

$|2^{0.25}-P_4(0.25)| \le \frac{0.25(0.25-0.1)(0.25-0.2)(0.25-0.3)(0.25-0.4)}{5!} 2^{0.4}\ln^5(2) \approx 2.47412\times 10^{-8}$

(b) When $x=0.05$, the upper error bound is $5.77294\times 10^{-8} = 0.57729\times 10^{-7}$ $\to 7-1=6$ places.

$0.57729>0.5$

6 decimal places is guaranteed to be correct.

When $x=0.25$, the upper error bound is $2.47412\times 10^{-8}=0.247412\times 10^{-7}$ $\to 7$ places.

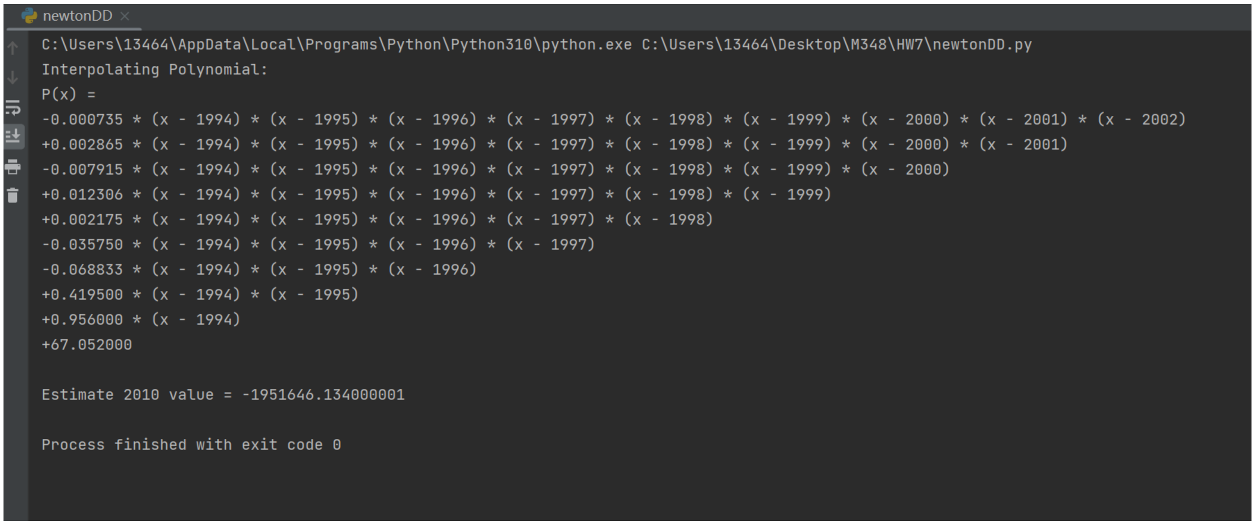
$0.24741<0.5$

7 decimal places is guaranteed to be correct.

**4. Chapter 3.2 CP # 3** The total world oil production in millions of barrels per day is shown in the table that follows. Determine and plot the degree 9 polynomial through the data. Use it to estimate 2010 oil production. Does the Runge phenomenon occur in this example? In your opinion, is the interpolating polynomial a good model of the data? Explain.

To plot the interpolating polynomial, you may use newtonDD. Submit:

- the interpolating polynomial;
- a plot of the data points and the polynomial;
- the estimated value of oil production;
- answers and explanations about the Runge phenomenon (per the textbook's description).

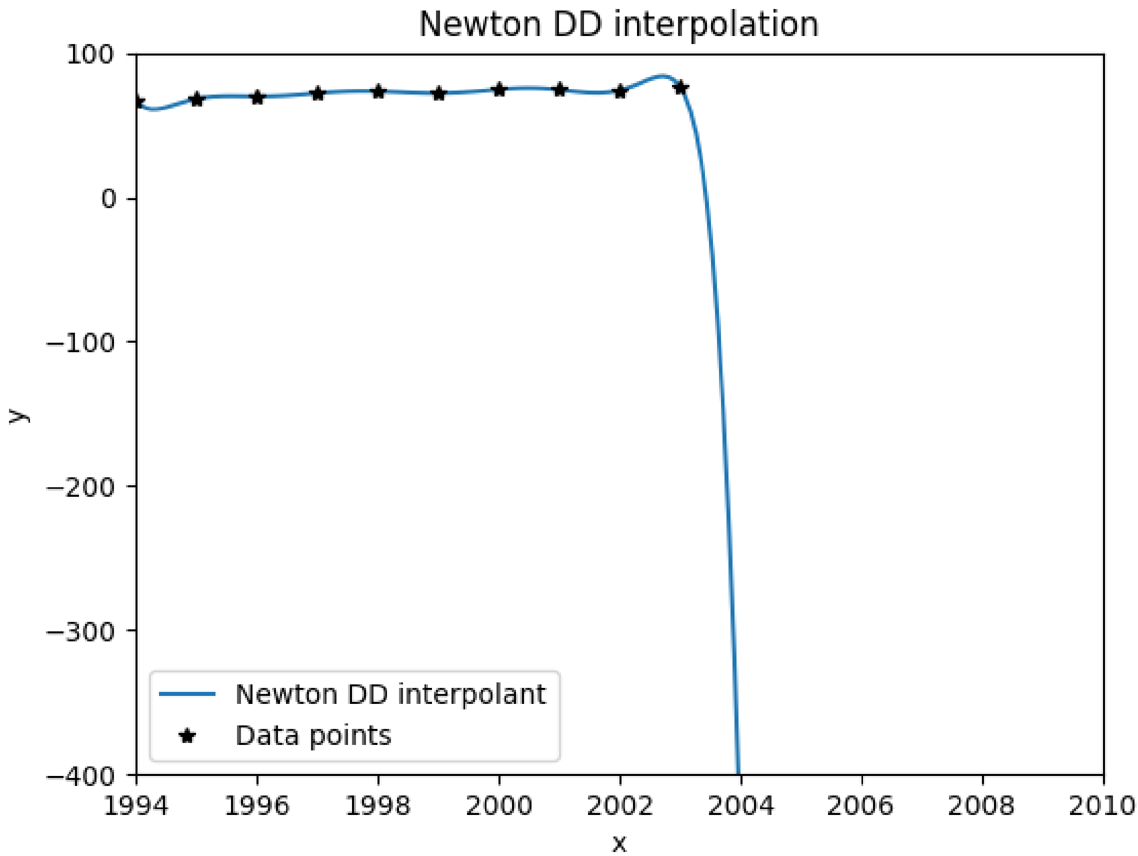


```
newtonDD
C:\Users\13464\AppData\Local\Programs\Python\Python310\python.exe C:\Users\13464\Desktop\M348\HW7\newtonDD.py
Interpolating Polynomial:
P(x) =
-0.000735 * (x - 1994) * (x - 1995) * (x - 1996) * (x - 1997) * (x - 1998) * (x - 1999) * (x - 2000) * (x - 2001) * (x - 2002)
+0.002865 * (x - 1994) * (x - 1995) * (x - 1996) * (x - 1997) * (x - 1998) * (x - 1999) * (x - 2000) * (x - 2001)
-0.007915 * (x - 1994) * (x - 1995) * (x - 1996) * (x - 1997) * (x - 1998) * (x - 1999) * (x - 2000)
+0.012306 * (x - 1994) * (x - 1995) * (x - 1996) * (x - 1997) * (x - 1998) * (x - 1999)
+0.002175 * (x - 1994) * (x - 1995) * (x - 1996) * (x - 1997) * (x - 1998)
-0.035750 * (x - 1994) * (x - 1995) * (x - 1996) * (x - 1997)
-0.068833 * (x - 1994) * (x - 1995) * (x - 1996)
+0.419500 * (x - 1994) * (x - 1995)
+0.956000 * (x - 1994)
+67.052000

Estimate 2010 value = -1951646.134000001

Process finished with exit code 0
```

**Solution.**

- As shown in the picture, we get $P(x) = -0.000735*(x-1994)*(x-1995)*(x-1996)*(x-1997)*(x-1998)*(x-1999)*(x-2000)*(x-2001)*(x-2002)+0.002865*(x-1994)*(x-1995)*(x-1996)*(x-1997)*(x-1998)*(x-1999)*(x-2000)*(x-2001)-0.007915*(x-1994)*(x-1995)*(x-1996)*(x-1997)*(x-1998)*(x-1999)*(x-2000)+0.012306*(x-1994)*(x-1995)*(x-1996)*(x-1997)*(x-1998)*(x-1999)+0.002175*(x-1994)*(x-1995)*(x-1996)*(x-1997)*(x-1998)-0.035750*(x-1994)*(x-1995)*(x-1996)*(x-1997)-0.068833*(x-1994)*(x-1995)*(x-1996)+0.419500*(x-1994)*(x-1995)+0.956000*(x-1994)+67.052000$

- Plot of the data points and the polynomial:




- The estimated value of oil production is around -1951646, (-1951646.134000001), as shown in the bottom of the first picture.
- Runge phenomenon does occur in this example. In this case, the interpolating polynomial is a degree 9 polynomial, which is quite high. This phenomenon is clearly observed in the plot, where the interpolating polynomial deviates dramatically from the trend of the data as we move outside the range of the given data points, specifically after the year 2003, leading to a very large negative prediction for the year 2010.

  From my perspective, the interpolating polynomial is not a good model of the data for predictions. The degree of the polynomial is too high for the number of data points, and while it may model the within-sample data accurately, it is unsuitable for making predictions outside that range.

```
    # Newton Divided Difference Interpolation

import numpy as np
import matplotlib.pyplot as pyp
import argparse
# this just makes it so that if you do "python3 newtonDD.py --test", it will run test en
# where numbers are input through the command line. You can modify how this is carried o
# changing the testNewton() function
'''parser = argparse.ArgumentParser(description="Provide Newton's Divided Difference Int
parser.add_argument("-t","--test",action="store_true")
TEST = parser.parse_args().test'''
############################## FUNCTIONS #############################

# Evaluate divided difference interpolant
def newtonEval(t,coefs,x):
    n = len(coefs)
    value = coefs[n-1]
    for i in range(n-2,-1,-1): # same as n-2, n-3, n-4, ..., 0
        value = value*(t-x[i]) + coefs[i]
    return value

# Set up divided difference coefficients
def newtonDDsetup(x,y):
    n = len(x)
    if (len(y) != n):
        print("ERROR CODE 1: x and y are different sizes")
        exit(1)

    # DD level 0
    # coefs[i] = y[i] for i=0,1,2,...,n-1
    coefs = [y[i] for i in range(n)]

    # DD higher levels (bottom to top, overwrite lower entries as they are finished)
```

```
    for level in range(1,n): # 1,2,3,4, ... n-1
        for i in range(n-1,level-1,-1): #n-1, n-2, ..., level
            dx = x[i] - x[i-level]
            if (dx==0): exit(2)
            coefs[i] = (coefs[i]-coefs[i-1])/dx
    return coefs

# x,y are 1d- arrays
def newtonDD(x,y):
    n = len(x)
    if (len(y) != n): exit(1)
    coefs = newtonDDsetup(x, y)
    printPolynomial(coefs, x)
    x_values = np.linspace(np.min(x), np.max(x), 500)
    y_values = [newtonEval(x, y, coefs) for x in x_values]

    estimate_2010 = newtonEval(2010, coefs, x)
    print()
    print("Estimate 2010 value = " + str(estimate_2010))
    '''if TEST: print("x =",x)
    if TEST: print("y =",y)

    if TEST:
        print("The coefs are: ", end=" ")
        for i in range(n):
            print(" %g"%(coefs[i]),end=" ")
        print()'''

    m = 10*n
    minx = min(x)
    maxx = max(x)
    t = np.arange(minx,maxx+1,(maxx-minx)/m)
    val = newtonEval(t,coefs,x)
```

```
    # plot
    pyp.plot(t,val)
    for i in range(n):
        pyp.plot(x[i],y[i],'k*')
    pyp.xlabel("x")
    pyp.ylabel("y")
    pyp.xlim(1994,2010)
    pyp.ylim(-400, 100)
    pyp.title("Newton DD interpolation")
    pyp.legend(["Newton DD interpolant","Data points"],loc="best")
    pyp.show()



# Estimate for 2010 using the interpolating polynomial



def printPolynomial(coefs, x):
    n = len(coefs)

    print("Interpolating Polynomial:")
    print("P(x) =")

    for i in range(n - 1, -1, -1):  # Start from the last coefficient
        term = f"{coefs[i]:+.6f}"  # Include sign in format
        for j in range(i):
            term += f" * (x - {x[j]:.0f})"
        if i > 0:
            print(f"{term}")
        else:
            print(term)
```

```
'''def printPolynomial(coefs, x):
    n = len(coefs)
    terms = []

    # Construct the polynomial as a string
    for i in range(n):
        term = f"{coefs[i]:.6f}"
        for j in range(i):
            term += f"*(x - {x[j]:.3f})"
        terms.append(term)

    # Combine terms into a polynomial string
    polynomial = " + ".join(terms)

    print("Interpolating Polynomial:")
    print(f"P(x) = {polynomial}")'''


'''def testNewton():
    n = int(input("Enter n: "))

    # Get data from command line
    ans = input("Enter data by points? [y/n] ")
    if (ans[0] == 'y' or ans[0] == 'Y'):
        data = input(f"Enter {n} data points (x1 y1 x2 y2 ... xn yn): ").split(" ")
        # other ways to do this; just for testing without entering a file.
        # pull all the x-values and convert to floats: even indices 0, 2, 4, ...
        x = list(map(float, np.array(data)[np.arange(0,len(data),2)]))
        # pull all the y-values and convert to floats: odd indices 1, 3, 5, ...
        y = list(map(float, np.array(data)[np.arange(1,len(data),2)]))
    else:
        x_data = input(f"Enter {n} distinct x values (x1 x2 ... xn): ").split(" ")
        x = list(map(float,x_data))
```

```
        y_data = input(f"Enter {n} distinct y values (y1 y2 ... yn): ").split(" ")
        y = list(map(float,y_data))

    newtonDD(x,y)'''




################################ MAIN ##############################
years = np.array([1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003])
production = np.array([67.052, 68.008, 69.803, 72.024, 73.400, 72.063, 74.669, 74.487, 7
newtonDD(years, production)
```