M 348 HOMEWORK 8

ZUN CAO

1. Chapter 3.3 Ex # 4

2. Chapter 3.4 Ex # 4

3. Additional Problem 1:

4. Additional Problem 2:

Date: 4/4/2024.

M348 HW8

3.3 Ex#4. $n=5+1=6$, $a=0.6$, $b=1.0$

$t_i=\cos\frac{(2i-1)\pi}{2\cdot 6}$, $x_i=\frac{1-0.6}{2}\cos\frac{(2i-1)\pi}{12}+\frac{1-0.6}{2}$ on $[0.6,1.0]$

Because

$$|(x-x_1)(x-x_2)\cdots(x-x_n)|\le\frac{(\frac{b-a}{2})^n}{2^{n-1}}=\frac{(\frac{1-0.6}{2})^n}{2^{n-1}}=\frac{(0.2)^6}{2^5}$$

($\searrow n=6$)

then,

$$|e^x-Q_5(x)|\le\frac{|f^6(c)|}{6!}\cdot\frac{(0.2)^6}{2^5}$$

Since $0.6<c<1$, $|f^6(c)|\le e^1$ on $[0.6,1]$

Therefore.

$$|e^x-Q_5(x)|\le\frac{e^1(0.2)^6}{6!\cdot 2^5}\approx 7.55\times10^{-9}=\underbrace{0.0755\times10^{-7}}_{<0.5}\Rightarrow \text{place}=7.$$

7 decimal places will be corrected.

3.4 Ex.#4 interval $[0,3]\Rightarrow x_1=0$, $x_2=1$, $x_3=2$, $x_4=3$

Since this is a cubic spline, 3 properties must be followed: (Apply P1 & P2)

① $S_i(x_i)=y_i$ and $S_i(x_{i+1})=y_{i+1}$ for $i=1,\cdots,n-1$

$i=1\Rightarrow S_1(1)=y_2=1$

$x=1$: $4+k_1x+2x^2-\frac{1}{6}x^3=1$

$4+k_1+2-\frac{1}{6}=1$

$k_1=-\frac{29}{6}$

$i=2\Rightarrow S_2(2)=y_3=1$

$1-\frac{4}{3}(2-1)+k_2(2-1)^2-\frac{1}{6}(2-1)^3=1$

$1-\frac{4}{3}+k_2-\frac{1}{6}=1$

$k_2=\frac{3}{2}$

② $S'_{i-1}(x_i)=S'_i(x_i)$ for $i=2,3,\cdots,n-1$.

$$\Rightarrow S'(x)=\begin{cases}k_1+4x-\frac{1}{2}x^2 & [0,1]\\ -\frac{4}{3}+2k_2(x-1)-\frac{1}{2}(x-1)^2 & [1,2]\\ k_3+2(x-2)-\frac{1}{2}(x-2)^2 & [2,3]\end{cases}$$

$i=3\Rightarrow S'_2(x_3)=S'_2(2)=S'_3(2)$

$\rightarrow -\frac{4}{3}+2k_2(2-1)-\frac{1}{2}(2-1)^2=k_3+2(2-2)-\frac{1}{2}(2-2)^2$

$-\frac{4}{3}+2\cdot\frac{3}{2}-\frac{1}{2}=k_3+0$

$k_3=\frac{7}{6}$

For natural spline, prop 4a $\rightarrow S_1''(x_1)=0$ and $S''_{n-1}(x_n)=0$.

In this case, $n=4$

$\therefore S_1''(X_1) = S_1''(0) = 0$ and $S_3''(X_4) = S_3''(3) = 0$

$$\Rightarrow S''(x) = \begin{cases} 4-x & [0,1] \\ 2k_2-(x-1) = 3-(x-1) = 4-x & [1,2] \\ 2-(x-2) = 4-x & [2,3] \end{cases}$$

However, $S_1''(0) = 4 \neq 0$ and $S_3''(3) = 1 \neq 0$.

Therefore, it's not a Natural Spline.

For a parabolically terminated cubic spline,

$d_1 = 0 = d_{n-1}$, and we require $C_1 = C_2$ and $C_{n-1} = C_n$.

However, we observe $C_1 = 2 \neq C_2 = \frac{3}{2}$.

Therefore, it's not a parabolically terminated cubic spline.

For a Not-a-knot cubic spline,

$S_1'''(X_2) = S_2'''(X_2)$ and $S_{n-2}'''(X_{n-1}) = S_{n-1}'''(X_{n-1})$

$$S''(x) = \begin{cases} -1 & [0,1] \\ -1 & [1,2] \\ -1 & [2,3] \end{cases}$$

$S_1'''(1) = -1 = S_2'''(1)$

$S_2'''(2) = -1 = S_3'''(2)$

Therefore, this is a Not-a-knot cubic spline.

$$\begin{cases} k_1 = -\frac{29}{6} \\ k_2 = \frac{3}{2} \\ k_3 = \frac{7}{6} \end{cases}$$

AD1. $a=0, b=1, f(x)=\sin x$.

$$\text{error} \le \frac{|f^{(n)}(c)|}{n!}|(x-x_1)\cdots(x-x_n)| \le \frac{|f^{(n)}(c)|}{n!}\frac{1}{2^{n-1}}\left(\frac{1-0}{2}\right)^n = \frac{|f^{(n)}(c)|}{n!}\cdot\frac{1}{2^{2n-1}},\ c\in[0,1]$$

$$f^{(n)}(x) = \sin\left(x+\frac{n\pi}{2}\right)$$

$$0<c<1,\ |f^{(n)}(c)| \le \sin\left(1+\frac{n\pi}{2}\right) \text{ on } [0,1].$$

$$\therefore \text{error} \le \frac{\sin\left(1+\frac{n\pi}{2}\right)}{n!\,2^{2n-1}} < \frac{1}{2}\times 10^{-7}$$ (correct to 7 decimal place we want).

Use trail of error. We can find that:

$n=5$ $\quad EB = 0.0879\times10^{-4} \rightarrow$ 4 places

$n=6$ $\quad EB = -0.571\times10^{-6} \rightarrow$ 6 places

$n=7$ $\quad EB = -1.309\times10^{-8} = -0.1309\times10^{-7} \rightarrow$ 7 places

$\Rightarrow$ degree $= d = n-1 = 7-1 = 6$ $\quad\therefore$ the least degree d is 6

Next, we calculate those 7 notes.

Change of interval is needed since its $[0,1]$.

$$x_i = \frac{b-a}{2}\cos\frac{(2i-1)\pi}{2n} + \frac{b+a}{2} = \frac{1}{2}\cos\frac{(2i-1)\pi}{14} + \frac{1}{2}$$

By calculator: $x_1 = \frac{1}{2}\cos\frac{\pi}{14}+\frac{1}{2} \approx 0.987464$, $x_2 \approx 0.890916$, $x_3 \approx 0.716942$

$x_4 = 0.5$, $x_5 \approx 0.283058$, $x_6 \approx 0.109084$, $x_7 \approx 0.012536$

Therefore, the 7 nodes are:

$[0.012536, 0.109084, 0.283058, 0.5, 0.716942, 0.890916, 0.987464]$

AD2. $x_1=1,\ x_2=2,\ x_3=3$

$y_1=0,\ y_2=4,\ y_3=\frac{22}{3}$

$\delta_1=x_2-x_1=2-1=1$

$\delta_2=x_3-x_2=3-2=1$

$\Delta_1=y_2-y_1=4-0=4$

$\Delta_2=y_3-y_2=\frac{22}{3}-4=\frac{10}{3}$

since this is a clamped cubic spline,

$S_1'(x_1)=S_1'(1)=3=v_1$

$S_{3-1}'(x_3)=S_2'(3)=3=v_3$

$\Rightarrow 2\delta_1c_1+\delta_1c_2=3(\frac{\Delta_1}{\delta_1}-v_1)$

$\delta_2c_2+2\delta_2c_3=3(v_3-\frac{\Delta_2}{\delta_2})$

we can build a linear system for c_i:

$$\begin{bmatrix}2\delta_1 & \delta_1 & 0\\ \delta_1 & 2(\delta_1+\delta_2) & \delta_2\\ 0 & \delta_2 & 2\delta_2\end{bmatrix}\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix}=\begin{bmatrix}3(\frac{\Delta_1}{\delta_1}-v_1)\\ 3(\frac{\Delta_2}{\delta_2}-\frac{\Delta_1}{\delta_1})\\ 3(v_3-\frac{\Delta_2}{\delta_2})\end{bmatrix}$$

$$\begin{bmatrix}2 & 1 & 0\\ 1 & 4 & 1\\ 0 & 1 & 2\end{bmatrix}\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix}=\begin{bmatrix}3(\frac{4}{1}-3)\\ 3(\frac{10}{3}-\frac{4}{1})\\ 3(3-\frac{10}{3})\end{bmatrix}=\begin{bmatrix}3\\-2\\-1\end{bmatrix}$$

$$\left[\begin{array}{ccc|c}2 & 1 & 0 & 3\\ 1 & 4 & 1 & -2\\ 0 & 1 & 2 & -1\end{array}\right]\xrightarrow{R_2-\frac{1}{2}R_1}\left[\begin{array}{ccc|c}2 & 1 & 0 & 3\\ 0 & \frac{7}{2} & 1 & -\frac{7}{2}\\ 0 & 1 & 2 & -1\end{array}\right]\xrightarrow{R_3-\frac{2}{7}R_2}\left[\begin{array}{ccc|c}2 & 1 & 0 & 3\\ 0 & \frac{7}{2} & 1 & -\frac{7}{2}\\ 0 & 0 & \frac{12}{7} & 0\end{array}\right]$$

$\begin{bmatrix}2c_1+c_2=3\\ \frac{7}{2}c_2+c_3=-\frac{7}{2}\\ \frac{12}{7}c_3=0\end{bmatrix}$ $2c_1-1=3,\ c_1=2$ $\frac{7}{2}c_2+0=-\frac{7}{2},\ c_2=-1$ $c_3=0$ $\quad c=\begin{bmatrix}2\\-1\\0\end{bmatrix}$

$d_1=\frac{c_2-c_1}{3\delta_1}=\frac{-1-2}{3}=-1$ $\quad b_1=\frac{\Delta_1}{\delta_1}-\frac{\delta_1}{3}(2c_1+c_2)=\frac{4}{1}-\frac{1}{3}(2\cdot2-1)=3$

$d_2=\frac{c_3-c_2}{3\delta_2}=\frac{0-(-1)}{3}=\frac{1}{3}$ $\quad b_2=\frac{\Delta_2}{\delta_2}-\frac{\delta_2}{3}(2c_2+c_3)=\frac{10}{3}-\frac{1}{3}(2\cdot(-1)+0)=4$

$$S(x)=\begin{cases}S_1(x)=0+3(x-1)+2(x-1)^2-(x-1)^3 & \text{on } [1,2]\\ S_2(x)=4+4(x-2)-(x-2)^2+\frac{1}{3}(x-2)^3 & \text{on } [2,3]\end{cases}$$

5. Coding Exercise Write a code in Matlab, Python, or C++ that approximates f(x)=cos(x) using 2 different methods:

(a) $cos1(x)$, an interpolant at 3 equally spaced points on $[0, \frac{\pi}{2}]$.

(b) $cos2(x)$, a Chebyshev interpolant with 3 Chebyshev nodes on the same domain $[0, \frac{\pi}{2}]$.

```
C:\Users\13464\AppData\Local\Programs\Python\Python310\python.exe C:\Users\13464\Desktop\M348\HW7\newtonDD3.py
        x              cos(x)             cos1(x)               Error
  -1000.0            0.562379           0.575450            0.013071
    -14.0            0.136737           0.153347            0.016610
     -4.0           -0.653644          -0.658838            0.005194
     -3.0           -0.989992          -0.977803            0.012189
     -2.0           -0.416147          -0.437748            0.021602
     -1.0            0.540302           0.555024            0.014722
      1.0            0.540302           0.555024            0.014722
      2.0           -0.416147          -0.437748            0.021602
      3.0           -0.989992          -0.977803            0.012189
      4.0           -0.653644          -0.658838            0.005194
     14.0            0.136737           0.153347            0.016610
   1000.0            0.562379           0.575450            0.013071

        x              cos(x)             cos2(x)               Error
  -1000.0            0.562379           0.572057            0.009678
    -14.0            0.136737           0.140348            0.003611
     -4.0           -0.653644          -0.657558            0.003914
     -3.0           -0.989992          -0.987068            0.002925
     -2.0           -0.416147          -0.431062            0.014915
     -1.0            0.540302           0.551129            0.010827
      1.0            0.540302           0.551129            0.010827
      2.0           -0.416147          -0.431062            0.014915
      3.0           -0.989992          -0.987068            0.002925
      4.0           -0.653644          -0.657558            0.003914
     14.0            0.136737           0.140348            0.003611
   1000.0            0.562379           0.572057            0.009678

Process finished with exit code 0
```

Solution. Bonus:

(a) see picture1

(b) see picture2

(c) see picture3

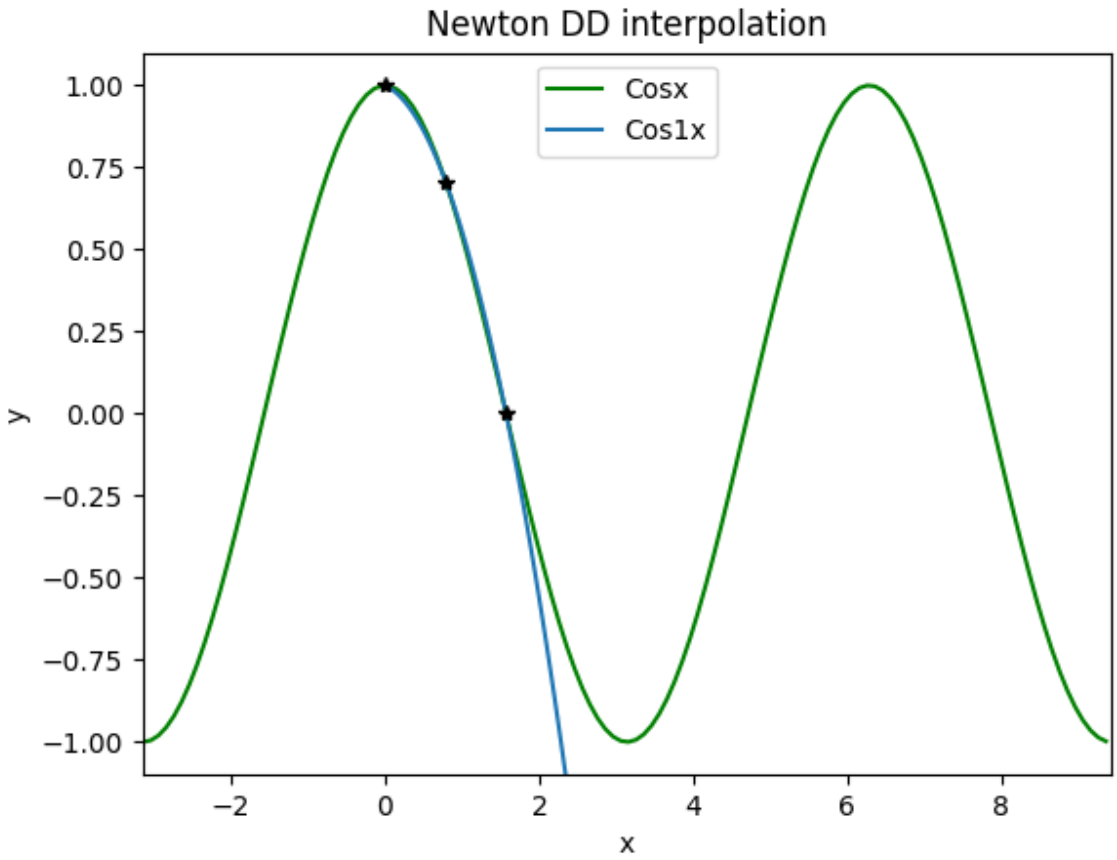

FIGURE 1. Cosx and Cos1x

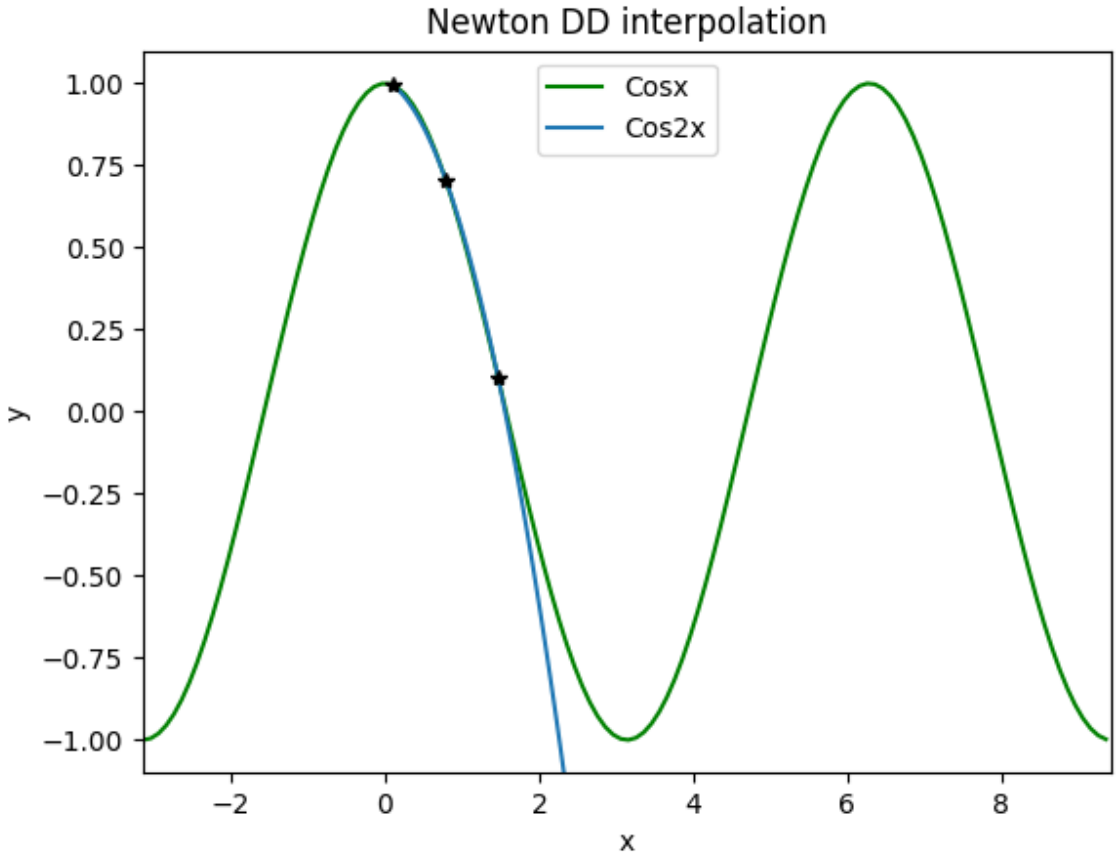

FIGURE 2. Cosx and Cos2x

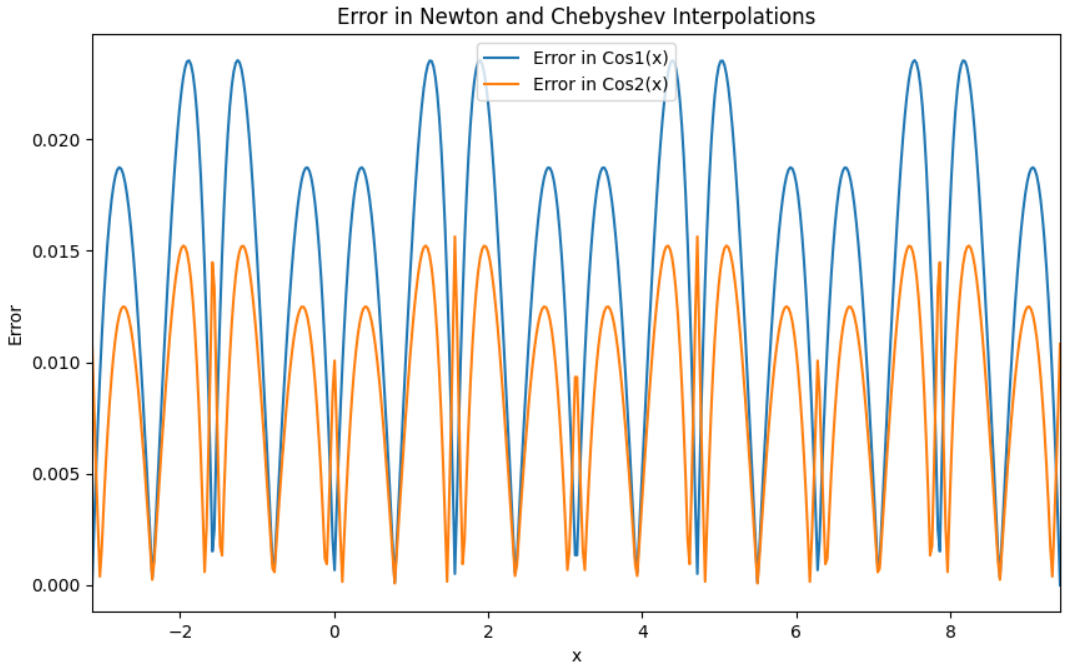

FIGURE 3. Error of Cos1(x) and Cos2(x)

```
# Newton Divided Difference Interpolation

import numpy as np
import matplotlib.pyplot as pyp
import math
import argparse


############################## 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

def chebyshev_nodes(n, a, b):
    return [0.5 * (a + b) + 0.5 * (b - a) * math.cos((2*k - 1) *
                                        math.pi / (2 * n)) for k in
                                         range(1, n + 1)]

def domain_cos(x_value):
    x_revised = []
    for x in x_value:
        s = 1
        x = x % (2*math.pi)
        if x > math.pi:
            x = 2*math.pi - x
        if x > math.pi/2:
            s = -1
            x = math.pi - x
        x_revised.append((x, s))
    return x_revised

# x,y are 1d- arrays
def newtonDD(x,y,name):
    n = len(x)
    if (len(y) != n): exit(1)


    coefs = newtonDDsetup(x,y)


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

    # plot
    x1 = np.arange(-np.pi,3*np.pi, 0.1)
```

```
    y1 = [np.cos(x) for x in x1]
    # Plotting coine Graph
    pyp.plot(x1, y1, color='green')
    pyp.plot(t,val)
    for i in range(n):
        pyp.plot(x[i],y[i],'k*')
    pyp.xlabel("x")
    pyp.ylabel("y")
    pyp.xlim(-np.pi,3*np.pi)
    pyp.ylim(-1.1, 1.1)
    pyp.title("Newton DD interpolation")
    pyp.legend(["Cosx", name],loc="best")
    pyp.show()



def compute_error(x_value, coefs, nodes):
    x_revised = domain_cos(x_value)
    y_actual = [np.cos(val) for val in x_value]
    y_interpolated = [s*newtonEval(x, coefs, nodes) for (x, s) in
                                            x_revised]
    errors = [abs(actual - approx) for actual, approx in zip(
                                            y_actual, y_interpolated)]
    return errors

# Function to plot the errors
def plot_errors(x_vals, error1, error2):
    pyp.figure(figsize=(10, 6))
    pyp.plot(x_vals, error1, label='Error in Cos1(x)')
    pyp.plot(x_vals, error2, label='Error in Cos2(x)')
    pyp.xlabel('x')
    pyp.ylabel('Error')
    pyp.title('Error in Newton and Chebyshev Interpolations')
    pyp.legend()
    pyp.xlim([-np.pi, 3*np.pi])
    pyp.show()
```

```
############################## MAIN ##############################
x_known = [0, np.pi / 4, np.pi / 2]
y_known = [np.cos(xi) for xi in x_known]
x_chebyshev = chebyshev_nodes(3, 0, math.pi/2)
y_chebyshev = [np.cos(xi) for xi in x_chebyshev]
'''y_known = [1, 1/math.sqrt(2),0]'''

coefs = newtonDDsetup(x_known,y_known)
coefs2 = newtonDDsetup(x_chebyshev,y_chebyshev)
x_interpolated = [-1000, -14, -4, -3, -2, -1, 1, 2, 3, 4, 14, 1000]
x_revised = domain_cos(x_interpolated)
y_actual = [np.cos(val) for val in x_interpolated]
y_interpolated = [s*newtonEval(x, coefs, x_known) for (x, s) in
                                        x_revised]
errors = [abs(actual - approx) for actual, approx in zip(y_actual,
                                        y_interpolated)]

y_interpolated2 = [s*newtonEval(x, coefs2, x_chebyshev) for (x, s)
                                        in x_revised]
errors2 = [abs(actual - approx) for actual, approx in zip(y_actual,
                                        y_interpolated2)]
print("{:>10} {:>20} {:>20} {:>20}".format("x", "cos(x)", "cos1(x)",
                                         "Error"))
for i, val in enumerate(x_interpolated):
    print("{:10.1f} {:20.6f} {:20.6f} {:20.6f}".format(val, y_actual
                                            [i], y_interpolated[i],
                                            errors[i]))

print()
print("{:>10} {:>20} {:>20} {:>20}".format("x", "cos(x)", "cos2(x)",
                                         "Error"))
for i, val in enumerate(x_interpolated):
    print("{:10.1f} {:20.6f} {:20.6f} {:20.6f}".format(val, y_actual
                                            [i], y_interpolated2[i],
                                            errors2[i]))

newtonDD(x_known, y_known, "Cos1x")
```

```
newtonDD(x_chebyshev, y_chebyshev, "Cos2x")
x_value = np.linspace(-np.pi, 3*np.pi, num=500)

error1 = compute_error(x_value, coefs, x_known)
error2 = compute_error(x_value, coefs2, x_chebyshev)
plot_errors(x_value, error1, error2)
```