Project report (3)

Name (300493343)

Question 1

Introduction

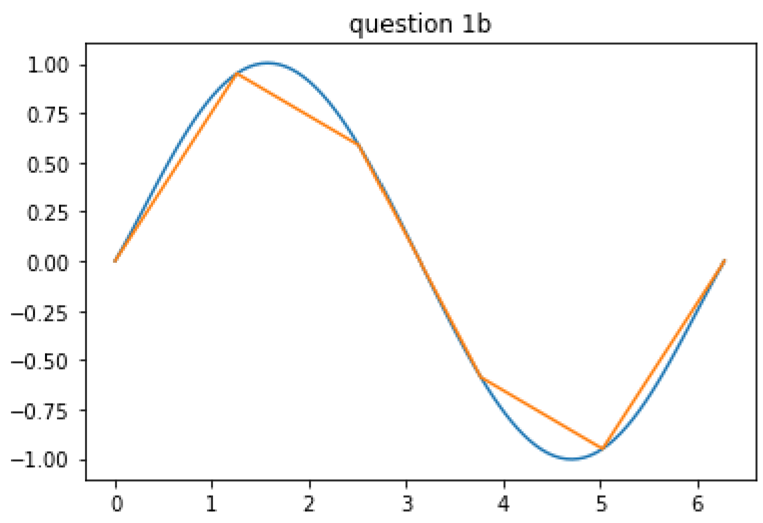
The purpose of this project was to explore different methods of interpolating graphs form a given set of data.

Procedure

In [12]:

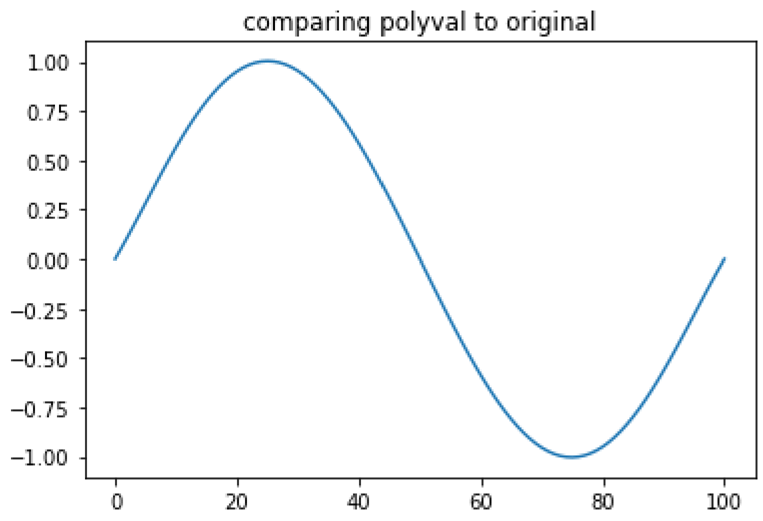
```
import numpy as np
import scipy.interpolate as sc
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

#Question 1a
#evaluate polynomial (horners formula)
def evalp(a,xdata,x):
    n = len(xdata) - 1 #Degree of polynomial
    p = a[n]
    for k in range(1,n+1):
        p = a[n-k] + (x - xdata[n-k])*p
    return p

#coeffiecents of the polynomial (newton's polynomial)
def coef(xdata, ydata):
    m = len(xdata)
    a = ydata.copy()
    for k in range(1,m):
        a[k:m] = (a[k:m] - a[k-1])/(xdata[k:m] - xdata[k-1])
    return a

def newtinterp(x, y, z):

    xdata = x.copy()                       #xdata array
    ydata = y.copy()                       #ydata array
    at_points = z.copy()                   #evaluating at points z array
    #get coefs array
    a_coefficient = coef(xdata, ydata) #array same size as ydata
    points = z.copy()                      #same size as z
    for i in range(len(at_points)):        #storing a list of point

        points[i] = evalp(a_coefficient, xdata, at_points[i]) #at_points[i] same size as x
data and it takes only 1 value rest passes on as array

    return points

#Question 1b
xinput = np.linspace(0, 2*np.pi, 6)
yinput = np.sin(xinput)
zpoints = np.linspace(0, 2*np.pi, 101)
intp = newtinterp(xinput, yinput, zpoints)

fig1 = plt.plot(zpoints, intp) #interpolating polynomial, 101 degree
fig1 = plt.plot(xinput, yinput)#interpolating polynomial, 5th degree
fig1 = plt.title("question 1b")
```

```
[0.         1.25663706 2.51327412 3.76991118 5.02654825 6.28318531]
```



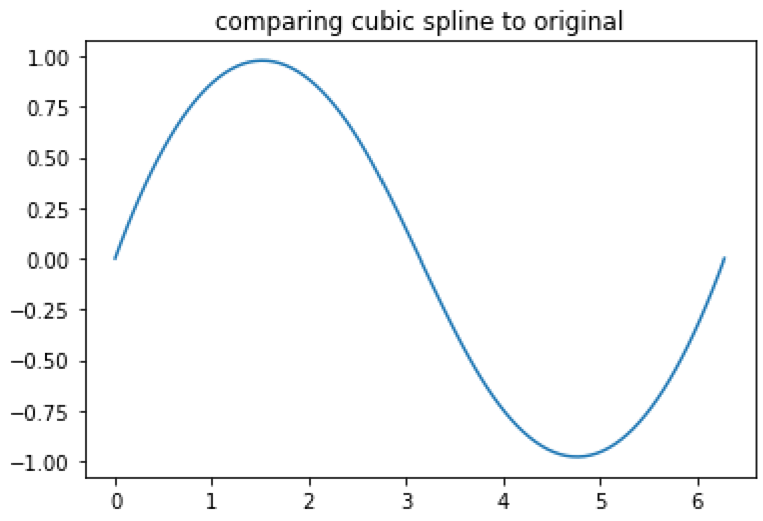
In [28]:

```
#Question 1c

polyfit_z = np.polyfit(xinput, yinput, 5)
polyval = np.polyval(polyfit_z, zpoints)
fig2 = plt.title("comparing polyval to original")
fig2 = plt.plot(polyval)
```



In [82]:

```
cs = CubicSpline(xinput, yinput)
z1 = cs(zpoints)
fig3 = plt.plot(zpoints, z1)
fig3 = plt.title("comparing cubic spline to original")
```



From my obesrvation, my rendition of lagrange interpolation and scipy's cubicspline numpy's polyfit/polyval looks identical in terms of shape

Observation

i have noticed that largrange interpolation is very effective as it uses a set of data x's and y's to interpolate a function. Comparing my results with python's polyfit, polyval and cubic spline it looks identical in terms of shape; prducing a sine wave. However, it is worth mentioning that polyval increments by sample size. Polyfit uses the least squares method.

Discussion

I believe when newton polynomial function (coef) returns a list of coefficients it's able to feed into the evalp function which is able to help refine the polynomial depending on how many degrees it wants to be refined to; largrange interpolation. This can be extremely important if you're given a set of discrete data and you'd like to refine it into a more visually coherent graph (continous time). This can be proven by how we refined 6 points from a sin function into an identical looking sin function. polyfit's least sqaare method allows the computer to estimate a line of best fit hence being able to find the line of best fit through those 5 discrete points creating a sine wave.

Conclusion

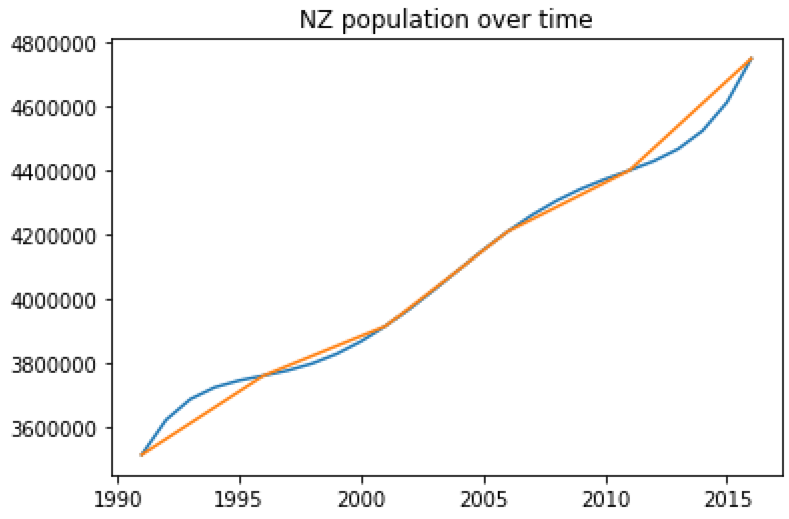
In conclusion, I believe largrange interpolation is very effective at processing sets of discrete data and refine it to a more coherent graph.

Question 2

Procedure

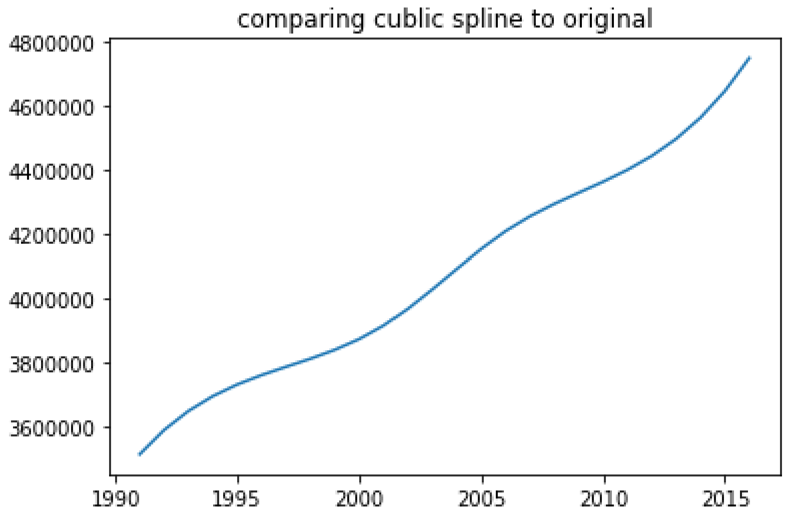
In [92]:

```
#Question 2a
t = np.linspace(1991,2016,6)
y = np.array([3516000.0, 3762300.0, 3916200.0, 4209100.0, 4399400.0, 4747200.0])
points = np.linspace(1991,2016,26)
population_intp = newtinterp(t, y, points)

fig1 = plt.plot(points, population_intp)
fig1 = plt.plot(t, y)
fig1 = plt.title("NZ population over time")
```



In [93]:

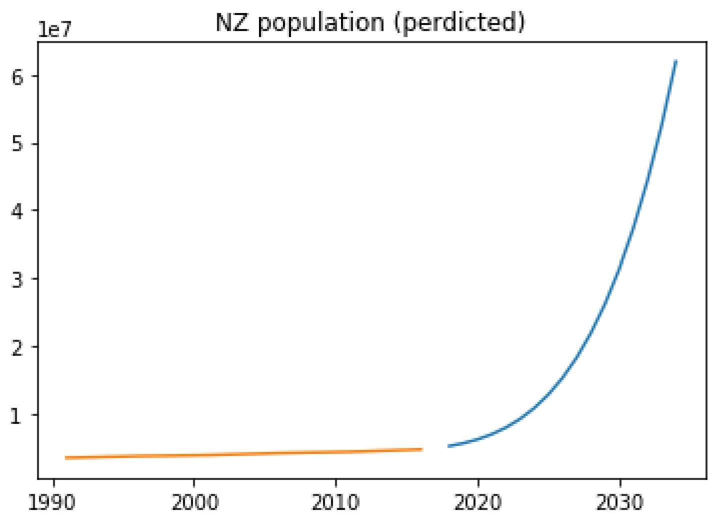
```
#Question 2b
cs = CubicSpline(t, y)
z2 = cs(points)
fig3 = plt.plot(points, z2)
fig3 = plt.title("comparing cublic spline to original")
```



from this observation both the interplation and cublic spline looks identical. However cublic spline looks smoother

In [94]:

```
#Question 2b
t1 = np.linspace(1991,2016,6)
y1 = np.array([3516000.0, 3762300.0, 3916200.0, 4209100.0, 4399400.0, 4747200.0])
points1 = np.linspace(2018,2034,17)
population_intp1 = newtinterp(t1, y1, points1)

fig1 = plt.plot(points1, population_intp1)
fig1 = plt.plot(t1, y1)
fig1 = plt.title("NZ population (perdicted)")
```



this have proven to be a relable source of prediction; orange is the original data set where as blue is the precedicted data set.

Observation

I have noticed that having points outside the intial polynomial can produce an estimate of the future values.

Discussion

I believe once the coefficents of the poynomial has already been determined it can be treated as a continous time graph thus being able to estimate future values. For exmaple a country's population is suppose to increase exponentially as more people can create more people. thus it is safe to assume that future values will also increase exponentially. largrange interpolation may be proven to be very useful for a company to determine sales.

Conclusion

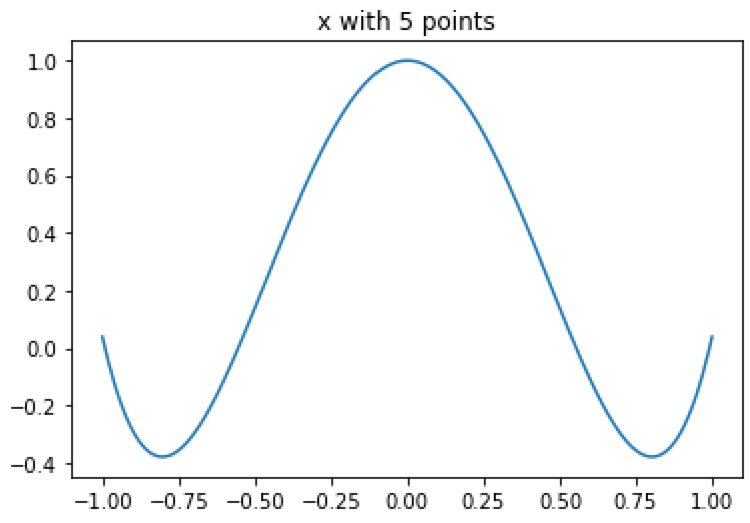
In conclusion, lagrange's interpolation can also produce estimate for future values just along with being able to turn discrete data into more time continous data.

Question 3

Procedure

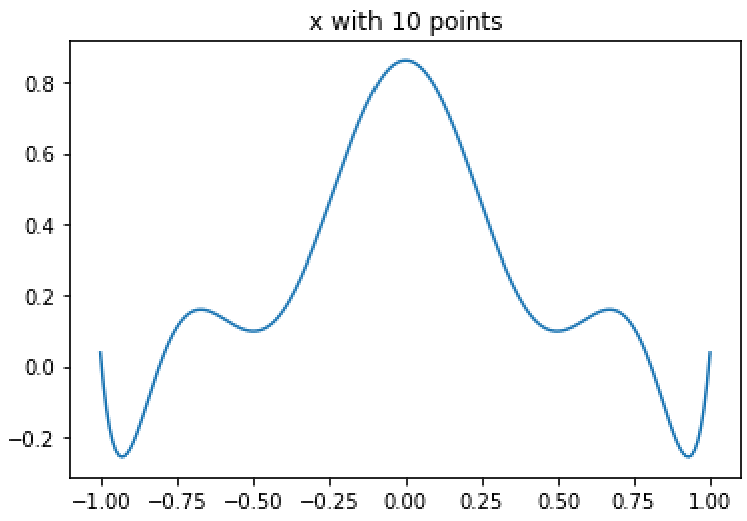
In [95]:

```
#Question 3a
def fx(x):
    y = 1.0/(1.0 + 25.0*(x**2.0))
    return y


def produce_fx(x,fx):
    output = x.copy()
    for i in range(len(x)):
        output[i] = fx(x[i])
    return output

x = np.linspace(-1,1,5)
x1 = np.linspace(-1,1,10)
x2 = np.linspace(-1,1,20)
x3 = np.linspace(-1,1,100)

y = produce_fx(x,fx)
y1 = produce_fx(x1,fx)
y2 = produce_fx(x2,fx)
y3 = produce_fx(x3,fx)



z = np.linspace(-1,1,201)

interpolate = newtinterp(x, y, z)
interpolate1 = newtinterp(x1, y1, z)
interpolate2 = newtinterp(x2, y2, z)
interpolate3 = newtinterp(x3, y3, z)

fig1 = plt.plot(z, interpolate)
fig1 = plt.title("x with 5 points")
```



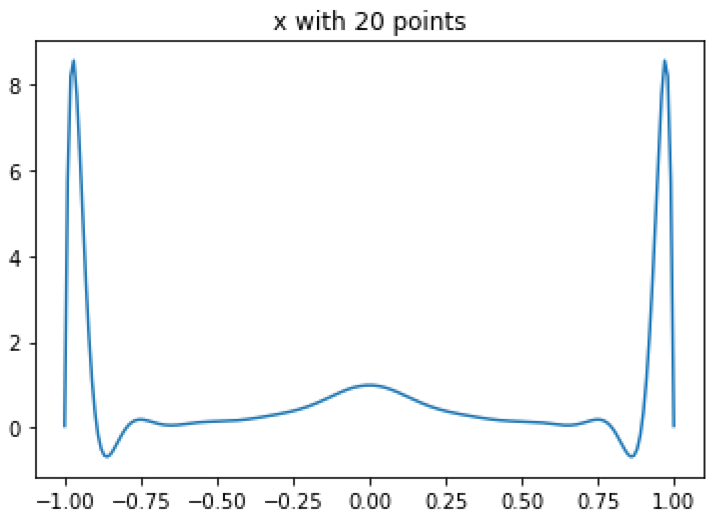
In [96]:

```
fig2 = plt.plot(z, interpolate1)
fig2 = plt.title("x with 10 points")
```



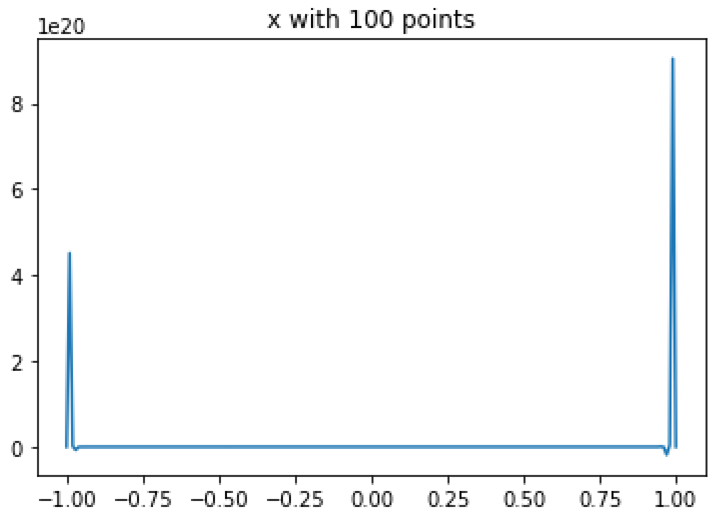
In [97]:

```
fig3 = plt.plot(z, interpolate2)
fig3 = plt.title("x with 20 points")
```



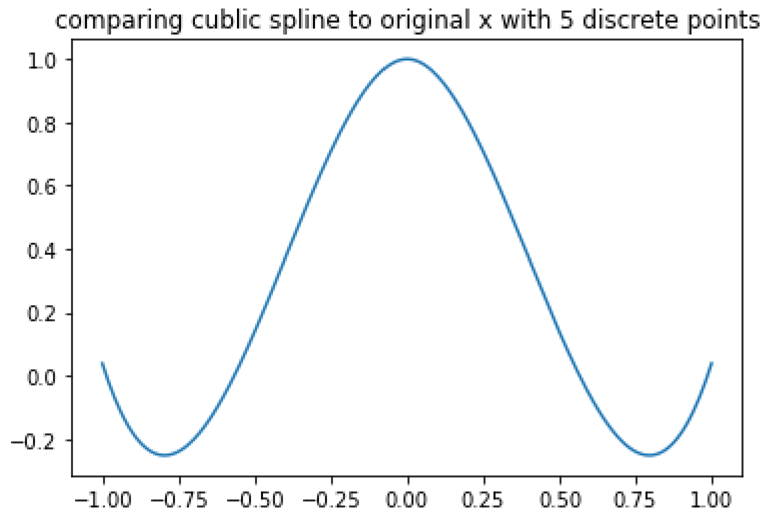
In [98]:

```
fig4 = plt.plot(z, interpolate3)
fig4 = plt.title("x with 100 points")
```



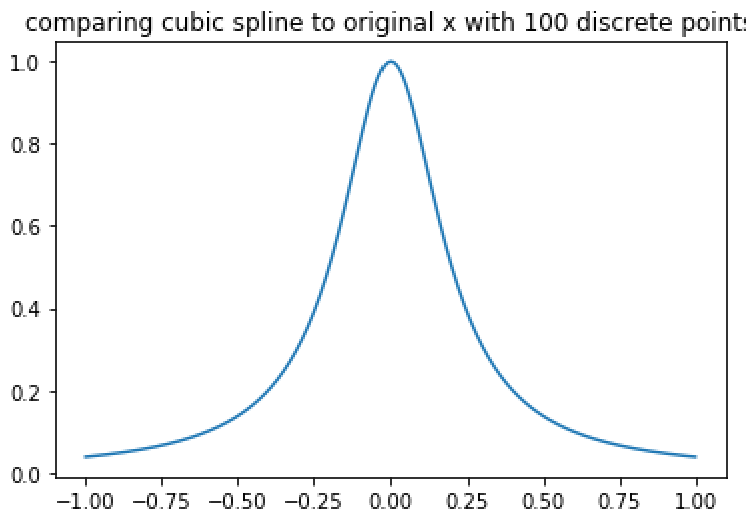
In [99]:

```
#Question 3b
cs1 = CubicSpline(x, y)
z2 = cs1(z) #same z as before in 3a
fig3 = plt.plot(z, z2)

fig3 = plt.title("comparing cublic spline to original x with 5 discrete points")
```



In [100]:

```
cs1 = CubicSpline(x3, y3)
z2 = cs1(z) #same z as before in 3a
fig3 = plt.plot(z, z2)

fig3 = plt.title("comparing cubic spline to original x with 100 discrete points")
```



There is a drastic difference when x has 100 points in cubic Spline and in lagrange interpolation. lagrange interpolation looks like delta function where as cubic spline looks like how the function should be modelled out to be.

Observation

I have noticed that sometimes there will not be any difference between cubic spline and lagrange interpolation. However, in this case there is a major difference, lagrange interpolation did not work for when there are 100 data set between x's limits -1<=x<=1. On another note, when x only has 5 points between -1 and 1, lagrange interpolation looks identical to that of cubic spline. It is often ideal to avoid high order interpolation when possible.

Discussion

I believe something that may have cause this to drastic difference is that question 3 already has a modelled function where as the previous 2 question already have small set of discrete points. The reason to avoid high order interpolation is because the lagrange interpolation is the high-order derivatives in the error Rn(x) since there's are really big near the end. furthermore, this expalins the delta function (massive spike in the order of 1x10^20) when we are given 100 points for the lagrange interpolation.

Conclusion

In conclusion, lagrange interpolation is extremely useful in most cases. However, in higher order interpolation when there are more data sets it can be proven to not be very useful compared to cubic spline.