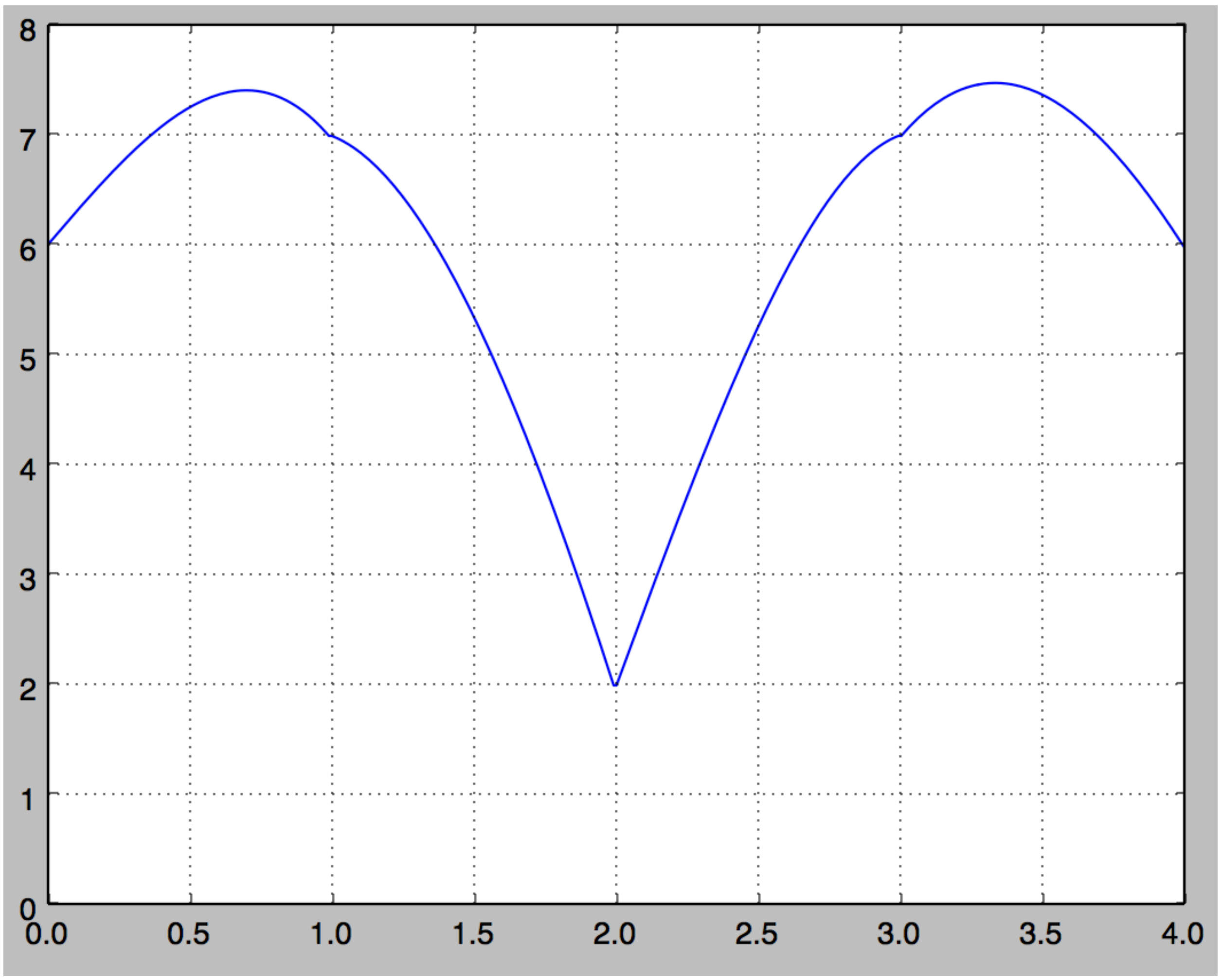
Abu Butt April 5, 2017
CSc-30100

Assignment 3

1)

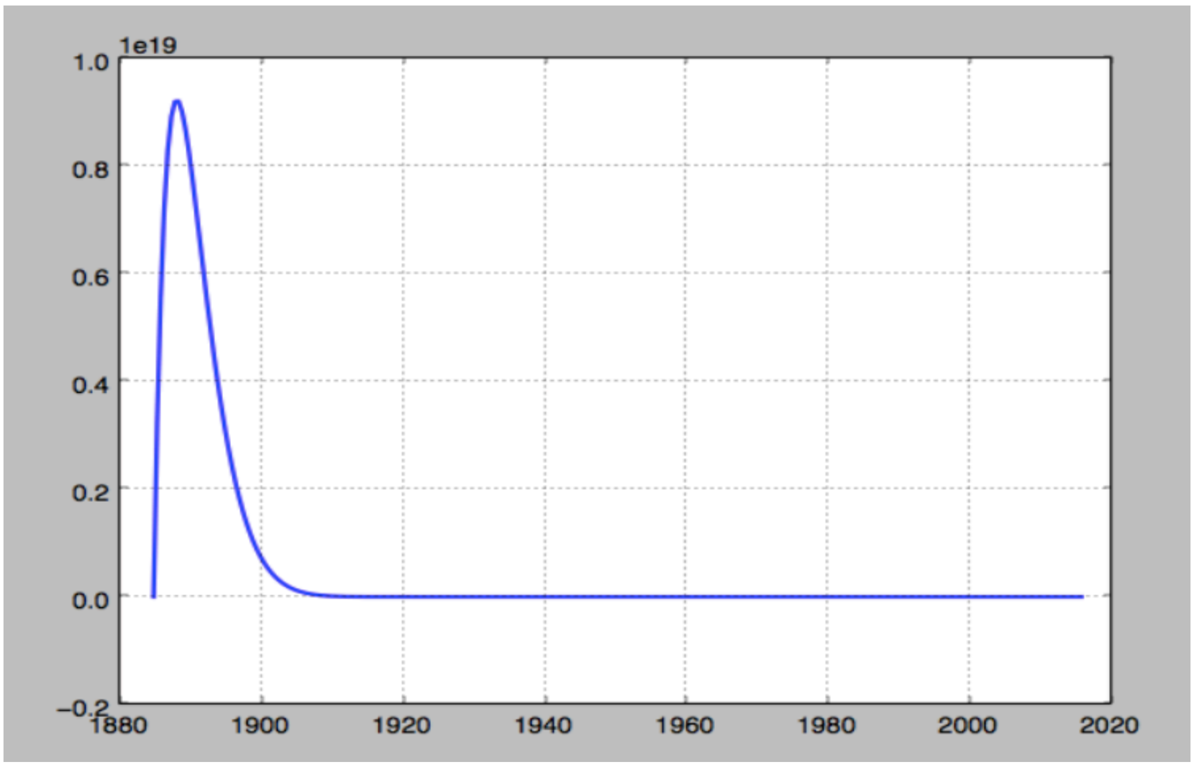
As we know interpolation through spline method can be used to approximate a parametric curve, which has x(t), y(t). Spline interpolation is used on both functions but separately. Then graphing the functions y vs x. For this question, I chose 5 points for this interpolation. The points are (0, 6), (1, 7), (2, 2), (3, 7), (4, 6), which will draw a script letter. I created 2 different arrays with different values for the functions. This python code interpolates through these points, which draws a “V”. The end points are set to zero, which creates a natural cubic spline for this interpolation. I had many functions to draw this interpolation. First you need to map the data points on the graph. Then to compute the spline formula, we need to compute Ai, Bi, Ci, Di, Sx. Each function is implemented using different values and variables.



```python
def Spline(X_values, Y_values, ii):
    def S(x):
        x_delta = []
        y_delta = []
        for i in range(len(X_values) - 1):
            x_delta.append(X_values[i+1] - X_values[i])
            y_delta.append(Y_values[i+1] - Y_values[i])

        Ai = Ai_Calculate(Y_values)
        Ci = Ci_Calculate(x_delta, y_delta)
        Di = Di_Calculate(Ci, x_delta)
        Bi = Bi_Calculate(x_delta, y_delta, Ci)
        s_x = (Ai[ii] + Bi[ii] * (x - X_values[ii]) + (Ci[ii] * (x - X_values[ii]) ** 2) + (Di[ii] * (x - X_values[ii]) ** 3))

        return s_x
    return S

def set_data_points(X_values, Y_values):
    y_list = []
    for i in range(2):
        x = np.linspace(X_values[i], X_values[i+1], 100)
        S = Spline(X_values, Y_values,i)
        y = map(S, x)
        y_list = y_list + y
    return y_list

def Spline_plot(y, starting_point, ending_point):
    x = np.linspace(starting_point, ending_point, 400)
    plt.axis([0, 4, 0, 8])
    plt.plot(x, y, linewidth=1)
    plt.grid(True)
    plt.show()

def Di_Calculate(Ci, x_delta):
    Di_temp=[]
    for i in range(len(Ci)-1):
        Di_temp.append(((Ci[i+1] - Ci[i])/(3*x_delta[i])))
    return Di_temp

def Ci_Calculate(x_delta, y_delta):
    Ci_temp = []
    Ci_temp.append(0)
    i = 0

    while(i<len(x_delta)-1):
        temp = ((3 * ((y_delta[i+1] / x_delta[i+1]) - (y_delta[i] / x_delta[i]))) / (2 * (x_delta[i] + x_delta[i+1])))
        Ci_temp.append(temp)
        i+=1
    Ci_temp.append(0)
    return Ci_temp

def Bi_Calculate(x_delta, y_delta, Ci):
    Bi_temp = []
    for i in range(len(Ci)-1):
        Bi_temp.append( ((y_delta[i] / x_delta[i]) - (x_delta[i] * (2 * Ci[i] + Ci[i+1])) / 3) )
    return Bi_temp

def Ai_Calculate(Y_values):
    Ai_temp = []
    for i in range(len(Y_values)-1):
        Ai_temp.append(Y_values[i])
```

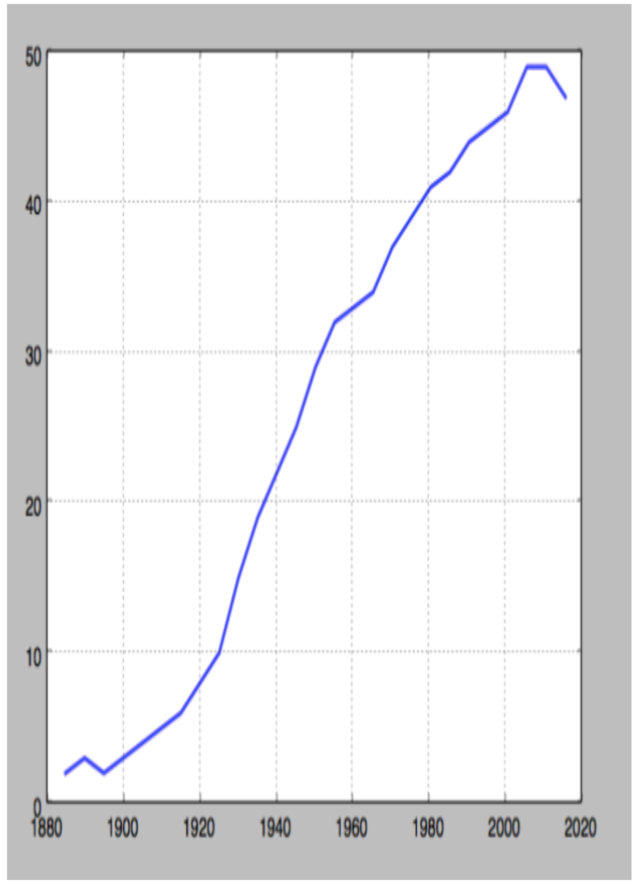
2) For this question, we are asked to find the postage stamp data from the website and find the Newton interpolation polynomial for these data. We will also determine the not-a-knot cubic spline from the given data. By using both method, we will estimate the year when the postage cost will be 50 cents.

To find the newton interpolation for this problem, we could implement the code from scratch and reuse the code from 2nd homework, which will be much easier. We see that there are 2 different values for year 1981, so to resolve this issue we will take the average of both values, which is 19. The modified code with different data points (years, price) will produce the following graph between the years (1885-2016).



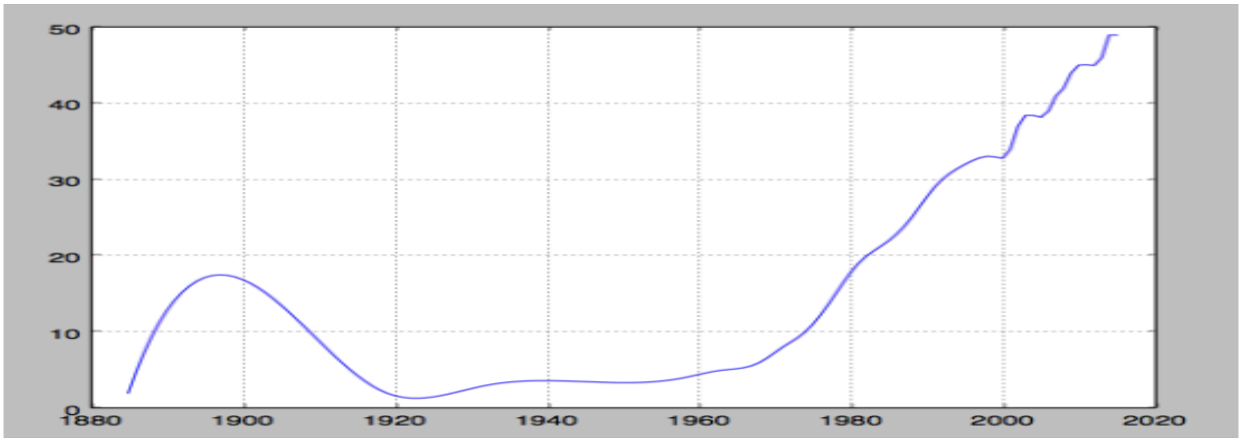
Newton interpolation is not efficient to find the future cost of stamps after the year of 2016.

```
Year: 1989   cost: 49.7101612035
Year: 1990   cost: 48.9023050963
Year: 1991   cost: 29.0
Year: 1992   cost: 8.69171069441
Year: 1993   cost: 3.08364270296
Year: 1994   cost: 14.0994503909
Year: 1995   cost: 32.0
Year: 1996   cost: 44.5657096402
Year: 1997   cost: 46.0491219885
Year: 1998   cost: 39.6536968664
Year: 1999   cost: 33.0
Year: 2000   cost: 31.3008607989
Year: 2001   cost: 34.0
Year: 2002   cost: 37.0
Year: 2003   cost: 37.5290214386
Year: 2004   cost: 36.6026552333
Year: 2005   cost: 36.895507723
Year: 2006   cost: 39.0000000001
Year: 2007   cost: 41.0000000001
Year: 2008   cost: 42.0
Year: 2009   cost: 44.0000000001
Year: 2010   cost: 47.6595668669
Year: 2011   cost: 48.5382976228
Year: 2012   cost: 45.0000000003
Year: 2013   cost: 46.0000000005
Year: 2014   cost: 49.0000000006
Year: 2015   cost: 49.0000000008
Year: 2016   cost: 47.0000000003
Year: 2017   cost: -3248.6371165
Year: 2018   cost: -46023.6478509
Year: 2019   cost: -355961.529686
```



For cubic spline, we would use built in library, which is from spicy import interpolation, which is more precise and output the better result. It will also have a graph, which looks much nicer and smoother then newton's approach.

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

x = [1885, 1917, 1919, 1932, 1958, 1963, 1968, 1971, 1974, 1978, 1981, 1985, 1988, 1991, 1995, 1999,
    2001, 2002, 2006, 2007, 2008, 2009, 2012, 2013, 2014, 2015, 2016]
y = [2, 3, 2, 3, 4, 5, 6, 8, 10, 15, 19, 22, 25, 29, 32, 33, 34, 37, 39, 41, 42, 44, 45, 46, 49, 49, 47]


xvals = np.arange(x[0], x[-1], 1)
func = interpolate.splrep(x, y, s=0)
yvals = interpolate.splev(xvals, func, der=0)

plt.plot(xvals, yvals)
pp = scipy.interpolate.spltopp(func[0][1:-1],func[1],func[2])
print(pp.coeffs)

plt.grid(True)
plt.show()
```



After making few more modifications in code, we could obtain the year, where the stamp will cost 50 cents, and increase the year range to 2020.

knotandnewtoninterp.py | interpolation.py | Splines.py

```
from scipy import interpolate
import matplotlib.pyplot as plt
import scipy.interpolate

x = [1885, 1917, 1919, 1932, 1958, 1963, 1968, 1971, 1974, 1978, 1981, 1985, 1988, 1991, 1995, 1999,
     2001, 2002, 2006, 2007, 2008, 2009, 2012, 2013, 2014, 2015, 2016]
y = [2, 3, 2, 3, 4, 5, 6, 8, 10, 15, 19, 22, 25, 29, 32, 33, 34, 37, 39, 41, 42, 44, 45, 46, 49, 49, 47]

xvals = np.arange(1884, 2020, 1)
func = interpolate.splrep(x, y, s=0)
yvals = interpolate.splev(xvals, func, der=0)

for i in range(len(xvals)):
    print "Year: {}".format(xvals[i]) + "\t" "cost: {}".format(yvals[i])

plt.plot(xvals, yvals)
plt.grid(True)
plt.show()
```

```
Year: 1991  cost: 29.0
Year: 1992  cost: 30.0141077064
Year: 1993  cost: 30.8076453878
Year: 1994  cost: 31.4473603753
Year: 1995  cost: 32.0
Year: 1996  cost: 32.5061436008
Year: 1997  cost: 32.9016985485
Year: 1998  cost: 33.096404222
Year: 1999  cost: 33.0
Year: 2000  cost: 32.8256228382
Year: 2001  cost: 34.0
Year: 2002  cost: 37.0
Year: 2003  cost: 38.4613327794
Year: 2004  cost: 38.4539188674
Year: 2005  cost: 38.2195455218
Year: 2006  cost: 39.0
Year: 2007  cost: 41.0
Year: 2008  cost: 42.0
Year: 2009  cost: 44.0
Year: 2010  cost: 45.0512509456
Year: 2011  cost: 45.10562244
Year: 2012  cost: 45.0
Year: 2013  cost: 46.0
Year: 2014  cost: 49.0
Year: 2015  cost: 49.0
Year: 2016  cost: 47.0
Year: 2017  cost: 46.0215405612
Year: 2018  cost: 49.0861622447
Year: 2019  cost: 59.2154056117
```

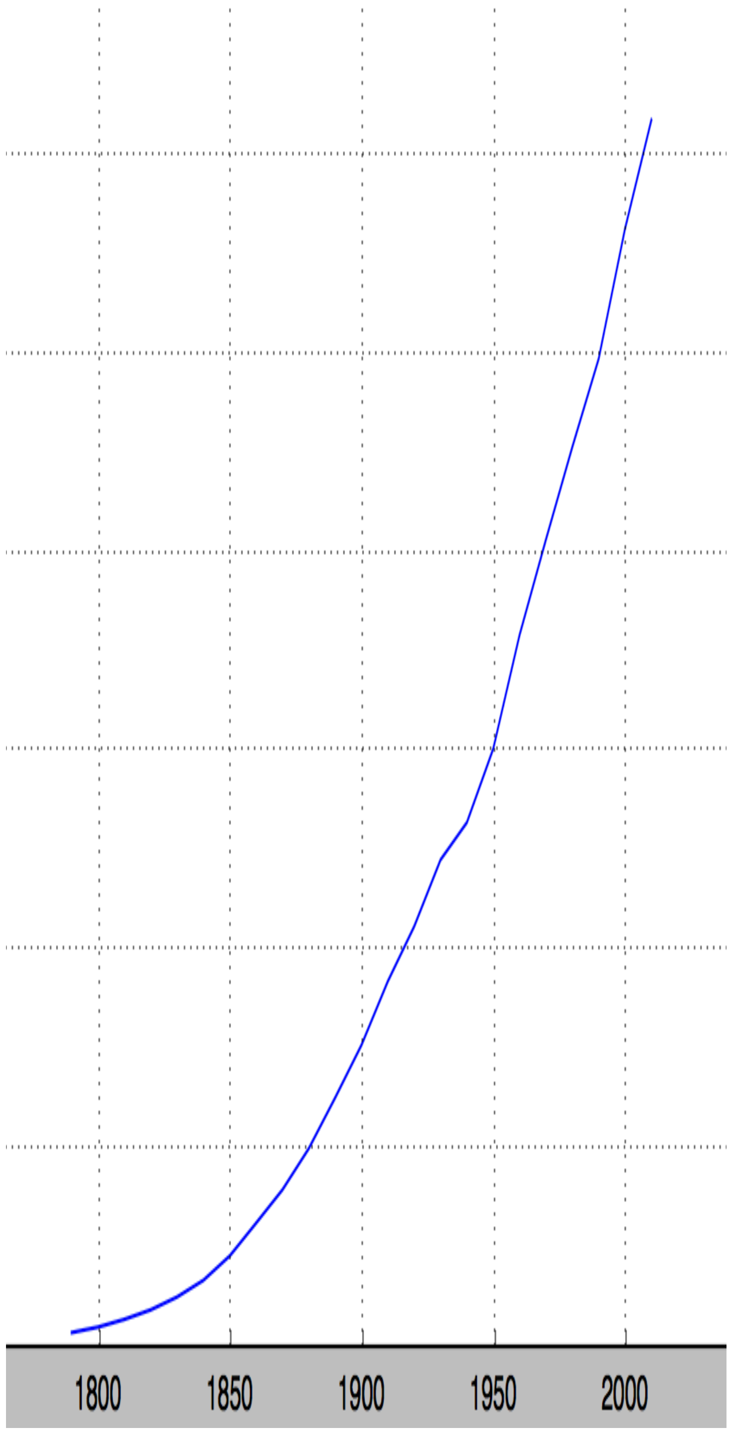
: Run   6: TODO   Python Console   Terminal

The years start from 1884 and continues until 2020. The cost of a stamp will be 50 cents anytime between the year of 2018 and 2019.

3)

For this question, we were asked to find the census data from the website and find the natural cubic spline function to interpolate the data, and lastly plot the data. Then we were to remove one of the data entry from the data, and find the error terms of each year.

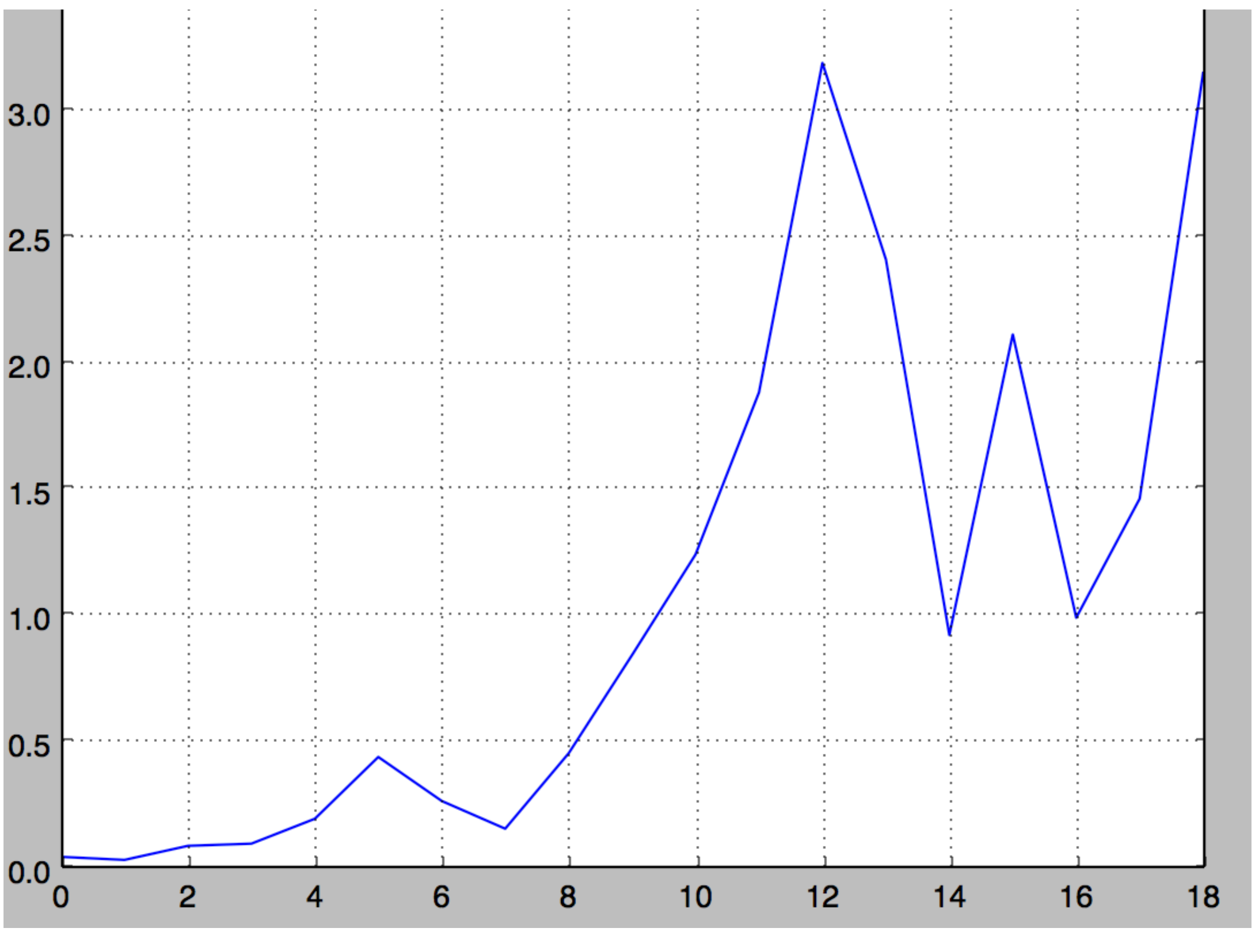
The data from Wikipedia about US census gives the following a natural cubic spline.




```
Year: 1790          Population: 3929326.0
Year: 1791          Population: 4035698.7558
Year: 1792          Population: 4149707.21608
Year: 1793          Population: 4271116.57034
Year: 1794          Population: 4399692.0081
Year: 1795          Population: 4535198.71885
Year: 1796          Population: 4677401.89209
Year: 1797          Population: 4826066.71732
Year: 1798          Population: 4980958.38405
Year: 1799          Population: 5141842.08178
Year: 1800          Population: 5308483.0
Year: 1801          Population: 5480646.32822
Year: 1802          Population: 5658097.25595
Year: 1803          Population: 5840600.97268
Year: 1804          Population: 6027922.66791
Year: 1805          Population: 6219827.53115
Year: 1806          Population: 6416080.7519
Year: 1807          Population: 6616447.51966
Year: 1808          Population: 6820693.02392
Year: 1809          Population: 7028582.4542
Year: 1810          Population: 7239881.0
Year: 1811          Population: 7454503.59431
Year: 1812          Population: 7672964.14413
Year: 1813          Population: 7895926.29994
Year: 1814          Population: 8124053.71225
Year: 1815          Population: 8358010.03155
Year: 1816          Population: 8598458.90832
Year: 1817          Population: 8846063.99306
Year: 1818          Population: 9101488.93625
Year: 1819          Population: 9365397.3884
Year: 1820          Population: 9638453.0
Year: 1821          Population: 9921167.33554
Year: 1822          Population: 10213443.6155
Year: 1823          Population: 10515032.9745
Year: 1824          Population: 10825686.5471
Year: 1825          Population: 11145155.4677
Year: 1826          Population: 11473190.8708
Year: 1827          Population: 11809543.8911
Year: 1828          Population: 12153965.6631
Year: 1829          Population: 12506207.3212
Year: 1830          Population: 12866020.0
Year: 1831          Population: 13233398.4845
Year: 1832          Population: 13609312.1617
Year: 1833          Population: 13994974.0689
Year: 1834          Population: 14391597.2435
Year: 1835          Population: 14800394.7228
Year: 1836          Population: 15222579.5444
Year: 1837          Population: 15659364.7455
Year: 1838          Population: 16111963.3635
Year: 1839          Population: 16581588.4359
Year: 1840          Population: 17069453.0
Year: 1841          Population: 17576743.8303
Year: 1842          Population: 18104542.6496
Year: 1843          Population: 18653904.9179
Year: 1844          Population: 19225886.0951
Year: 1845          Population: 19821541.6411
Year: 1846          Population: 20441927.0157
Year: 1847          Population: 21088097.6791
Year: 1848          Population: 21761109.0909
Year: 1849          Population: 22462016.7113
Year: 1850          Population: 23191876.0
Year: 1851          Population: 23950870.7262
Year: 1852          Population: 24735697.8957
Year: 1853          Population: 25542182.8234
Year: 1854          Population: 26366150.8241
Year: 1855          Population: 27203427.2129
Year: 1856          Population: 28049837.3047
Year: 1857          Population: 28901206.4143
Year: 1858          Population: 29753359.8568
Year: 1859          Population: 30602122.9471
Year: 1860          Population: 31443321.0
Year: 1861          Population: 32274076.9858
Year: 1862          Population: 33096704.4956
Year: 1863          Population: 33914814.7757
Year: 1864          Population: 34732019.0724
Year: 1865          Population: 35551928.6323
Year: 1866          Population: 36378154.7016
Year: 1867          Population: 37214308.5266
Year: 1868          Population: 38064001.3538
```

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

x_values = [1790, 1800, 1810, 1820, 1830, 1840, 1850, 1860, 1870, 1880, 1890, 1900, 1910, 1920, 1930, 1940, 1950, 1960,
    1970, 1980, 1990, 2000, 2010]
y_values = [3929326, 5308483, 7239881, 9638453, 12866020, 17069453, 23191876, 31443321, 39818449, 50189209, 62947714,
    76212168, 92228496, 106021537, 122775046, 132164569, 150697361, 179323175, 203302031, 226545805, 248709873,
    281421906, 308745538]

x_vals = np.arange(1790, 2011, 10)
func = interpolate.splrep(x_values, y_values, s=0)
y_vals = interpolate.splev(x_vals, func, der=0)

for i in range(len(x_vals)):
    print "Year: {}".format(x_vals[i]) + "\t" "Population: {}".format(y_vals[i])

plt.plot(x_vals, y_vals)
plt.grid(True)
plt.show()
```

Error’s for Each Year is displayed Below:
By modifying the above script, it will remove each entry at a time from the data given and calculate the cubic interpolation with the remaining points.



```
Year: 1790      Error: 446888.0
Year: 1800      Error: 326672.0
Year: 1810      Error: 883365.533333
Year: 1820      Error: 969707.767857
Year: 1830      Error: 1955544.99522
Year: 1840      Error: 4401273.77436
Year: 1850      Error: 2659153.60357
Year: 1860      Error: 1561390.94588
Year: 1870      Error: 4546094.44274
Year: 1880      Error: 8439208.33459
Year: 1890      Error: 12420195.058
Year: 1900      Error: 18836190.2356
Year: 1910      Error: 31878850.9621
Year: 1920      Error: 24099414.3709
Year: 1930      Error: 9232314.38125
Year: 1940      Error: 21125647.1827
Year: 1950      Error: 9917100.1022
Year: 1960      Error: 14629573.9636
Year: 1970      Error: 31484019.5292
Abus-MBP:Cubic_Interpolation ABUBUTT1$
```

By looking at the error estimate, we can conclude that as the points decrease the errors also decrease. The table also shows the year and the error.