中国科学技术大学

实验报告

计算方法 B

Project 1

学生姓名： 朱云沁

学生学号： PB20061372

完成时间： 二〇二二年三月十六日

目　　录

一、 实验题目

问题 1 下面给出美国1920～1970 年的人口表：

表 1 美国1920～1970 年的人口表

年份	1920	1930	1940	1950	1960	1970
人口（千人）	105711	123203	131669	150697	179323	203212

用表 1 中数据构造一个 5 次 Lagrange 插值多项式，并用此估计 1910、1965 和 2002 年的人口。1910 年的实际人口数约为 91772000，请判断插值计算得到的 1965 年和 2002 年的人口数据准确性是多少？

问题 2 数据同表 1，用 Newton 插值估计：

（1） 1965 年的人口数；

（2） 2012 年的人口数。

二、 实验结果

设 x 为年份，$f(x)$ 为人口（千人），即被插函数。将表 1 中年份从左至右分别记作 $x_0, x_1, x_2, x_3, x_4, x_5$，并令 $x_6 = 1910$。

1、 Lagrange 插值

编写程序，以 $x_0, x_1, x_2, x_3, x_4, x_5$ 为一组插值节点，构造 Lagrange 插值多项式 $L_5(x)$；以 $x_1, x_2, x_3, x_4, x_5, x_6$ 为另一组插值节点，构造 Lagrange 插值多项式 $\tilde{L}_5(x)$。插值计算得 1910、1965、2002 和 2012 年的人口，用事后估计方法计算得误差 $f(x) - L_5(x)$。结果如表 2 所示。

表 2 Lagrange 插值数值结果

x	$L_5(x)$	$\tilde{L}_5(x)$	$f(x) - L_5(x)$
1910	31872.000000	91772.000000	5.990000e+04
1965	193081.511719	193354.493490	-1.228418e+03
2002	26138.748416	-233411.305984	2.128310e+06
2012	-136453.125184	-801550.139584	6.118893e+06

注：数据保留原始输出格式，下同

绘制得区间 $[1900, 2010]$ 上 $L_5(x)$、$\tilde{L}_5(x)$ 以及 $f(x) - L_5(x)$ 的图象如图 1 所示。

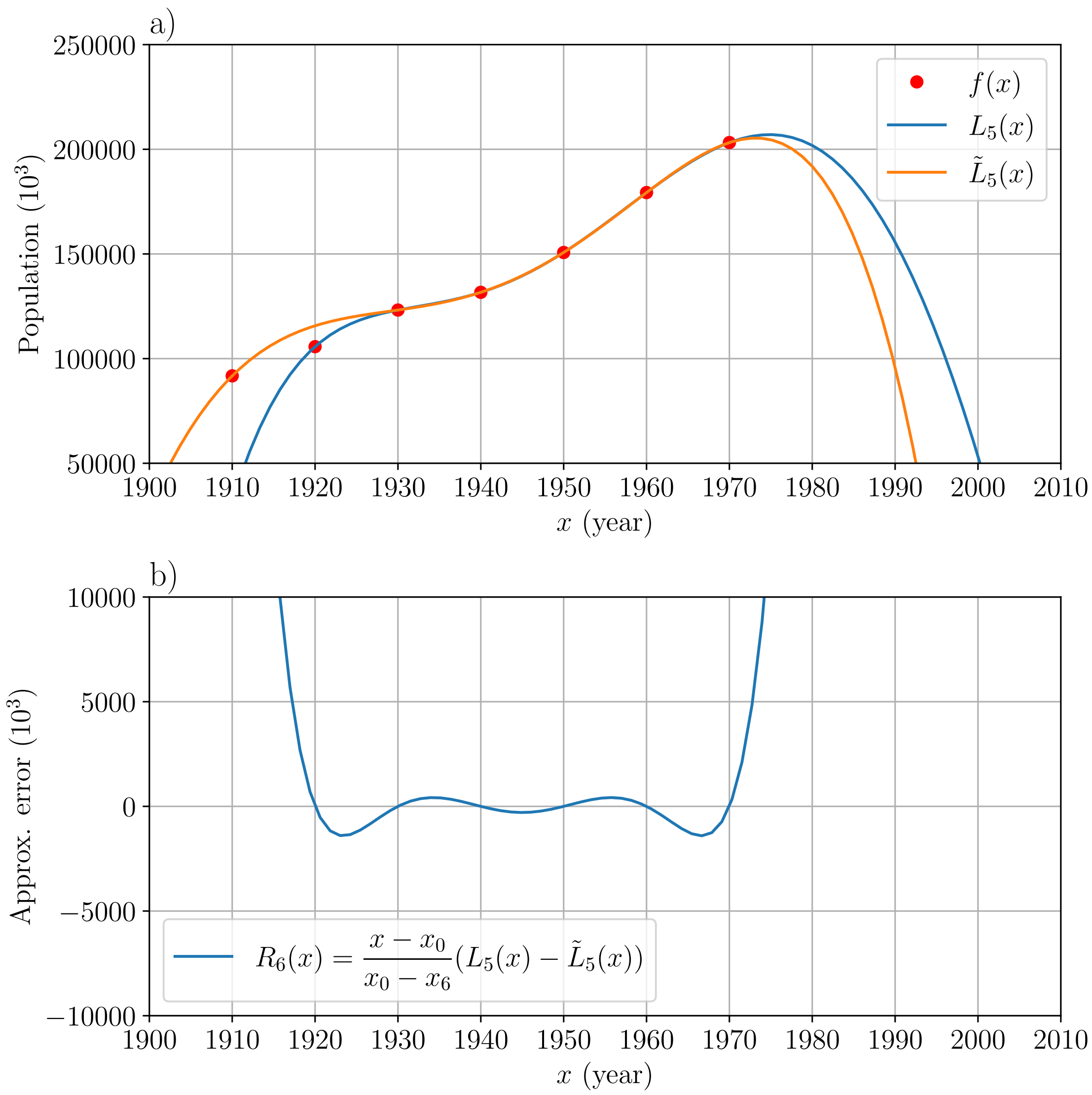

图 1　Lagrange 插值

2、 Newton 插值

编写程序，以 $x_0, x_1, x_2, x_3, x_4, x_5$ 为插值节点，构造牛顿插值多项式 $N_5(x)$，插值计算得 1910、1965、2002 和 2012 年的人口。结果如表 3 所示。

表 3　Newton 插值数值结果

x	$N_5(x)$
1910	31872.000000000015
1965	193081.51171875
2002	26138.748416000046
2012	-136453.1251840004

绘制得区间 $[1900, 2010]$ 上 $N_5(x)$ 的图象如图 2 所示。

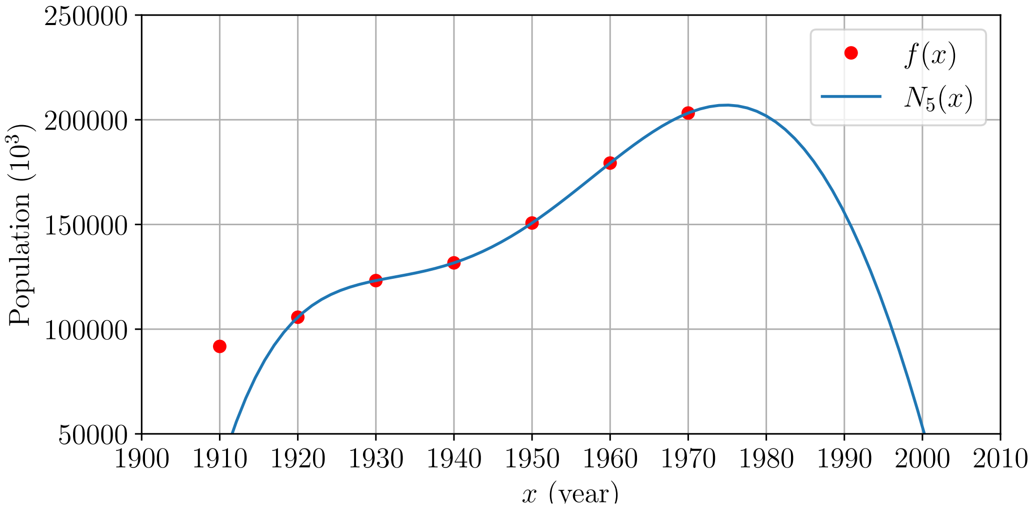

图 2　Newton 插值

3、 Neville 算法

编写程序，以 $x_0, x_1, x_2, x_3, x_4, x_5$ 为插值节点，用 Neville 算法构造插值多项式 $P_{0,1,\cdots,5}(x)$，插值计算得 1910、1965、2002 和 2012 年的人口。结果如表 4 所示。

表 4　Neville 算法插值数值结果

x	$P_{0,1,\cdots,5}(x)$
1910	31872.0
1965	193081.51171875
2002	26138.748415998853
2012	-136453.12518400417

绘制得区间 $[1900, 2010]$ 上 $P_{0,1,\cdots,5}(x)$ 的图象如图 3 所示。

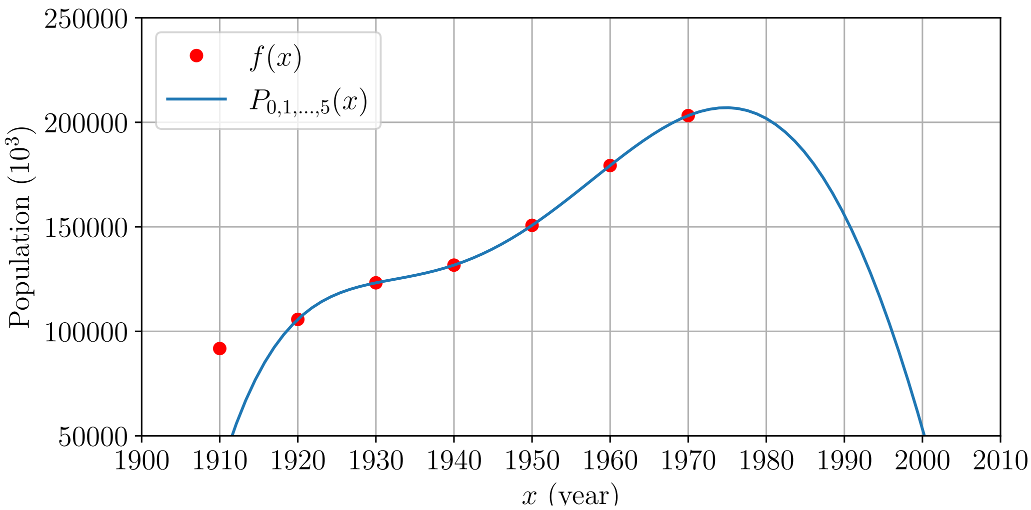

图 3　Neville 算法

三、 算法分析

1、 Lagrange 插值

关于插值节点 $x_0, x_1, \cdots, x_n$ 的 n 次 Lagrange 插值多项式为

$$L_n(x) = \sum_{i=0}^{n} l_i(x) f(x_i) \tag{1}$$

其中，插值基函数 $l_i(x)$ 的表达式为

$$l_i(x) = \prod_{\substack{0 \leqslant j \leqslant n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \tag{2}$$

于是，立即得到计算 Lagrange 多项式的算法：

算法 1　Lagrange 插值

Data: $(x_i, f(x_i)), i = 0, 1, \cdots, n$；待计算的点 x。
Result: $L_n(x)$

$L_n(x) \leftarrow 0$;
for $i \leftarrow 0$ **to** n **do**
　$l_i(x) \leftarrow 1$;
　for $j \leftarrow 0$ **to** n **do**
　　$l_i(x) \leftarrow l_i(x) \cdot (x - x_j)/(x_i - x_j)$;　　// 对于给定 x，计算 $l_i(x)$
　end
　$L_n(x) \leftarrow L_n(x) + l_i(x) \cdot f(x_i)$;　　// 求和，计算 $L_n(x)$
end

显然，该算法的时间复杂度为 $\mathbb{O}(n^2)$，空间复杂度为 $\mathbb{O}(1)$。

以 $(x_i, f(x_i)), i = 1, 2, \cdots, n+1$ 为插值点，重复上述算法，得 $\tilde{L}_n(x)$，进而用事后估计方法估算得误差为

$$f(x) - L_n(x) \approx \frac{x - x_0}{x_0 - x_{n+1}} (L_n(x) - \tilde{L}_n(x)) \tag{3}$$

2、 Newton 插值

关于插值节点 $x_0, x_1, \cdots, x_n$ 的 n 次 Newton 插值多项式为

$$N(x) = f(x_0) + \sum_{k=1}^{n} f[x_0, x_1, \cdots, x_k] \prod_{i=0}^{k-1} (x - x_i) \tag{4}$$

其中，$f[x_0, x_1, \cdots, x_k]$ 为函数 $f(x)$ 关于 $x_0, x_1, \cdots, x_k$ 的 k 阶差商，满足如下递归关系

$$f[x_0, x_1, \cdots, x_k] = \frac{f[x_1, \cdots, x_k] - f[x_0, x_1, \cdots, x_{k-1}]}{x_k - x_0} \tag{5}$$

构造如下差商表:

表 5　差商表

$f(x_i)$	$f[x_{i-1}, x_i]$	$f[x_{i-2}, x_{i-1}, x_i]$	$\cdots$	$f[x_0, x_1, \cdots, x_i]$
$f(x_0)$				
$f(x_1)$	$f[x_0, x_1]$			
$f(x_2)$	$f[x_1, x_2]$	$f[x_0, x_1, x_2]$		
$\vdots$	$\vdots$	$\vdots$	$\ddots$	
$f(x_n)$	$f[x_{n-1}, x_n]$	$f[x_{n-2}, x_{n-1}, x_n]$	$\cdots$	$f[x_0, x_1, \cdots, x_n]$

观察发现，按从左到右、从下到上的顺序计算上述差商表，只需维护大小为 $n+1$ 的数组，记作 $\mathrm{g}[i], i = 0, 1, \cdots, n$。递推结束后，应有 $\mathrm{g}[k] = f[x_0, x_1, \cdots, x_k]$，进而计算 $N_n(x)$。有如下算法：

算法 2　Newton 插值

Data: $(x_i, f(x_i)), i = 0, 1, \cdots, n$；待计算的点 x。
Result: $N_n(x)$
// 递推计算 $\mathrm{g}[k]$，即 $f[x_0, x_1, \cdots, x_k]$
$\mathrm{g}[i] \leftarrow f(x_i), i = 0, 1, \cdots, n$;
for $k \leftarrow 1$ **to** n **do**
　　for $j \leftarrow n$ **to** k **do**
　　　　$\mathrm{g}[j] = (\mathrm{g}[j] - \mathrm{g}[j-1])/(x_j - x_{j-k})$;
　　end
end
// 以嵌套乘法形式计算 $N_n(x)$
$N_n(x) \leftarrow \mathrm{g}[n]$;
for $i \leftarrow n-1$ **to** 0 **do**
　　$N_n(x) \leftarrow (x - x_i) N_n(x) + \mathrm{g}[i]$
end

该算法预处理（计算差商）的时间复杂度为 $\mathbb{O}(n^2)$，计算某个 x 处的插值函数的时间复杂度为 $\mathbb{O}(n)$；空间复杂度为 $\mathbb{O}(n)$。

3、 Neville 算法

记以 $x_0, x_1, \cdots, x_n$ 为插值节点构造的 n 次插值多项式为 $P_{0,1,\cdots,n}(x)$。存在递归关系

$$P_{0,1,\cdots,n}(x) = \frac{(x-x_0)P_{1,\cdots,n}(x) - (x-x_n)P_{0,\cdots,n-1}}{x_n - x_0} \tag{6}$$

类似于 Newton 插值中差商表的构造方式，维护数组 $p[i], i = 0, 1, \cdots, n$，使得递推结束后，$p[n] = P_{0,\cdots,n}(x)$。完整算法如下：

算法 3　Neville 算法

Data: $(x_i, f(x_i)), i = 0, 1, \cdots, n$；待计算的点 x。
Result: $P_{0,\cdots,n}(x)$

$p[i] \leftarrow f(x_i), i = 0, 1, \cdots, n$;
for $i \leftarrow 1$ **to** n **do**
　　for $j \leftarrow 1$ **to** $n - i$ **do**
　　　　$p[j] \leftarrow ((x - x_j)p[j+1] - (x - x_{j+i})p[j])/(x_{j+i} - x_j)$;
　　end
　　// $p[n]$ 即为输出结果 $P_{0,\cdots,n}(x)$
end

显然，该算法的时间复杂度为 $\mathbb{O}(n^2)$，空间复杂度为 $\mathbb{O}(n)$。相较于 Lagrange 插值，Neville 算法的运算次数更少，是以空间效率换取时间效率的做法。

考虑到插值多项式的唯一性，不再重复估算误差。

四、 结果分析

由三种多项式插值算法的数值与图示结果可以看出：

- 用 Lagrange 插值估计得 1910 年的人口数约为 31872 千人，1965 年的人口数约为 193082.5 千人，2002 年的人口数约为 26138.75 千人。
- 利用事后估计方法，计算出 1965 年的人口数绝对误差约为-1228.4 千人，用近似值代替精确值，计算得相对误差约为 -0.64%，准确性较好；2002 年的人口数绝对误差约为 2128310 千人，远远超出误差允许范围，准确性极差。
- 用 Newton 插值估计得 1965 年的人口数约为 193082.5 千人，2012 年的人口数约为-136453 千人。
- 选取相同的插值点，无论 Lagrange 插值多项式、Newton 插值多项式或用 Neville 算法得到的插值多项式，均得到相同的数值和图示结果，映证了插值多项式的唯一性。
- 在插值区间内，例如 1965 年，多项式插值的准确性较好；在插值区间外，例如 2002、2012 年，多项式插值估算得到的值不具有参考价值。

附录 A　Python 程序代码

1、 Lagrange 插值

```python
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('font', family='serif', size=15)
mpl.rc('mathtext', fontset='stix')
mpl.rc('text', usetex=True)
df = pd.read_csv('data_points.csv')
X, Y = df['x'].to_numpy(dtype=int),
    df['f(x)'].to_numpy(dtype=float)
def lagrange_interp(X, Y, x):
    n = len(X)
    y = 0
    for i in range(n):
        l = 1
        for j in range(i):
            l *= (x - X[j]) / (X[i] - X[j])
        for j in range(i+1, n):
            l *= (x - X[j]) / (X[i] - X[j])
        y += l * Y[i]
    return y
def L1(x): return lagrange_interp(X[:-1], Y[:-1], x)
def L2(x): return lagrange_interp(X[1:], Y[1:], x)
def R(x): return (x - X[0]) / (X[0] - X[-1]) * (L1(x) - L2(x))
out_df = pd.DataFrame([[x, L1(x), L2(x), R(x)] for x in [
                      1910, 1965, 2002, 2012]], columns=['x', 'L1',
                          'L2', 'R'])
x_range = np.linspace(1900, 2020, 100)
fig, (ax1, ax2) = plt.subplots(2, figsize=(8, 8),
    constrained_layout=True)
ax1.plot(X, Y, 'ro', label=r'$f(x)$')
ax1.plot(x_range, [L1(x) for x in x_range], label=r'$L_5(x)$')
ax1.plot(x_range, [L2(x) for x in x_range],
    label=r'$\tilde{L}_5(x)$')
ax1.set_title('a)', loc='left')
ax1.set_xlim(1900, 2010)
ax1.set_xticks(np.arange(1900, 2020, 10))
ax1.set_ylim(0.5e5, 2.5e5)
ax1.set_xlabel(r'$x$ (year)')
ax1.set_ylabel(r'Population ($10^3$)')
ax1.grid()
ax1.legend()
ax2.plot(x_range, [R(x) for x in x_range],
         label=r'$$f(x)-L_5(x)\approx
           \frac{x-x_0}{x_0-x_{6}}(L_5(x)-\tilde{L}_5(x))$$')
ax2.set_title('b)', loc='left')
ax2.set_xlim(1900, 2010)
ax2.set_xticks(np.arange(1900, 2020, 10))
ax2.set_ylim(-1e4, 1e4)
ax2.set_xlabel(r'$x$ (year)')
ax2.set_ylabel(r'Approx. error ($10^3$)')
ax2.grid()
ax2.legend()
fig.savefig('lagrange-interpolation.pdf')
print(out_df)
```

2、 Newton 插值

```python
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('font', family='serif', size=15)
mpl.rc('mathtext', fontset='stix')
mpl.rc('text', usetex=True)

df = pd.read_csv('data_points.csv')
X, Y = df['x'].to_numpy(dtype=int),
↪  df['f(x)'].to_numpy(dtype=float)


def divided_diff(X, Y):
    n = len(X)
    g = Y.copy()
    for i in range(1, n):
        for j in range(n-1, i-1, -1):
            g[j] = (g[j] - g[j-1]) / (X[j] - X[j-i])
    return g


def newton_interp(X, G, x):
    n = len(X)
    y = G[n-1]
    for i in range(n-2, -1, -1):
        y = G[i] + (x - X[i]) * y
    return y


G = divided_diff(X[:-1], Y[:-1])
def N(x): return newton_interp(X[:-1], G, x)


x_range = np.linspace(1900, 2020, 100)
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(X, Y, 'ro', label=r'$f(x)$')
ax.plot(x_range, [N(x) for x in x_range], label=r'$N_{5}(x)$')
ax.set_xlim(1900, 2010)
ax.set_xticks(np.arange(1900, 2020, 10))
ax.set_ylim(0.5e5, 2.5e5)
ax.set_xlabel(r'$x$ (year)')
ax.set_ylabel(r'Population ($10^3$)')
ax.grid()
ax.legend()

fig.savefig('newton-interpolation.pdf')
print(N(1910), N(1965), N(2002), N(2012))
```

3、 Neville 算法

```python
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('font', family='serif', size=15)
mpl.rc('mathtext', fontset='stix')
mpl.rc('text', usetex=True)

df = pd.read_csv('data_points.csv')
X, Y = df['x'].to_numpy(dtype=int), df['f(x)'].to_numpy(dtype=float)


def neville_interp(X, Y, x):
    n = len(X)
    p = np.array(Y)
    for i in range(1, n):
        for j in range(n-i):
            p[j] = ((x - X[j]) * p[j+1] - (x - X[j+i])
                    * p[j]) / (X[j+i] - X[j])
    return p[0]


def P(x): return neville_interp(X[:-1], Y[:-1], x)


x_range = np.linspace(1900, 2020, 100)
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(X, Y, 'ro', label=r'$f(x)$')
ax.plot(x_range, [P(x) for x in x_range], label=r'$P_{0,1,\dots,5}(x)$')
ax.set_xlim(1900, 2010)
ax.set_xticks(np.arange(1900, 2020, 10))
ax.set_ylim(0.5e5, 2.5e5)
ax.set_xlabel(r'$x$ (year)')
ax.set_ylabel(r'Population ($10^3$)')
ax.grid()
ax.legend()

fig.savefig('neville-interpolation.pdf')
print(P(1910), P(1965), P(2002), P(2012))
```

附录 B　CSV 格式数据

```
x,f(x)
1920,105711
1930,123203
1940,131669
1950,150697
1960,179323
1970,203212
1910,91772
```