廣東工業大學

《Python 数值分析》

课程报告

学　　院　计算机学院

专　　业　计算机科学与技术

班　　级　4 班

学　　号　3120005057

学生姓名　陈嘉浩

授课教师　龚怡

2022 年　05 月

目录

1 绪论

1.1 数值分析的发展综述

早在在三十年前, 计算数学的先驱之一 L. N. Trefethen 就给出了数值分析的定义:

Numerical analysis is the study of algorithms for the problems of continuous problems.

—Lloyd N. Trefethen, Cornell University

翻译过来就是: 数值分析是研究连续问题的算法的科学. 其中, 最主要的概念就是算法和连续问题. 首先, 连续问题是从物理或者其它学科中抽象出来的复杂模型问题, 一般是无穷维问题且几乎无法找到解析解. 这些棘手的连续问题就自然成为数值分析的目标对象. 其次, 求解连续问题的算法的设计和分析是数值分析的核心内容, 它们的目的是将连续的无穷维的问题离散化, 得到一个离散的有限维的可解问题, 进而得到近似解. 如果没有数值分析, 现代科学与工程应用研究将很快陷入停滞.

数值分析领域比现代计算机的发明早了好几个世纪。线性插值在 2000 多年前就已经在使用。过去许多伟大的数学家都专注于数值分析，从牛顿法、拉格朗日插值多项式、高斯消元法或欧拉法等重要算法的名称中可以明显看出这一点。

为方便手工计算，制作了带有插值点和函数系数等公式和数据表的大型书籍。使用这些表格，对于某些函数，通常计算到小数点后 16 位或更多，人们可以查找值以插入给定的公式并获得对某些函数的非常好的数值估计。该领域的权威著作是由 Abramowitz 和 Stegun 编辑的 NIST 出版物，这本 1000 多页的书中包含了大量常用的公式和函数，以及它们在许多点上的值。当有计算机可用时，函数值不再很有用，但大量的公式列表仍然非常方便。

机械计算器也被开发为手动计算的工具。这些计算器在 1940 年代演变成电子计算机，然后发现这些计算机也可用于管理目的。但是计算机的发明也影响了数值分析领域，因为现在可以进行更长更复杂的计算。

当今的主要研究领域有：1. 函数求值；2. 内插法、外推法、曲线拟合及回归；3. 求解方程及方程组；4. 求解特征值或奇异值问题；5. 最优化；6. 积分计算；7. 微分方程

1.2 报告主要内容及结构

本文将根据上课的顺序进行展开。

本报告将主要分为 6 章。

第一章是绪论，简要介绍数值分析的发展综述。

第二章是误差相关实验与分析。

第三章是非线性方程的数值解法实验与分析。

第四章是线性方程组的数值解法实验与分析。

第五章是插值法的实验与分析。

第六章最小二乘拟合的实验与分析

第七章数值积分、微分和常微分方程的数值解法
第八章是总结与展望。

2 误差相关实验及分析

2.1 误差的成因与处理手段探讨

观察误差：数学模型的原始输入数据和参数与实际数据之前存在误差

模型误差：数学模型与实际问题之间存在的误差

截断误差: 计算方法得到的近似解与数学模型的理论准确解之前存在的误差

舍入误差: 计算机的计算结果与理论结果存在的误差

(1) 避免两个相近的数相减

(2) 防止大数吃小数

(3) 避免采用绝对值很小的数作除数

(4) 简化运算步骤，减少运算次数

(5) 控制计算方法的误差传播，保证计算方法的稳定性

2.2 学习数值计算方法的目的

首先，众多生产实践与科学研究问题本身并不具备解析形式，或者运用纯数学方法难以找到问题的解析解。例如，对于超越方程，我们只能采用近似的计算方法来得到问题的答案。

其次，一些问题虽然具有解析形式，但过于复杂，计算机无法在可接受的时间内求解。

一些问题的解析解还可能含有无穷多项，这样就只能使用近似的计算方法去逼近问题的解。

再者，在分析实验、观察数据时，也需要使用插值、拟合等多种数值计算方法把一系列离散的数据关联起来。

2.3 综合实验: 减少运算次数的实验

2.3.1 实验题目

比较不同算法求多项式的运算次数与用时。

设计 2 种不同的算法计算以下函数的值, 分别测试 $x = 0.1, 1, 2$ 。

$$f_n(x) = 1 + 2x + 3x^2 + \mathrm{L} + 100001x^{100000}$$

2.3.2 实验条件

计算机配置:

CPU: Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz

内存大小: 16GB

操作系统: Windows10

2.3.3 算法介绍

算法一:

直接法: 直接计算出每一个项的值并相加。

算法二：

秦九韶算法:

$$f_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

$$\begin{cases} S_n = a_n \\ S_k = xS_{k+1} + a_k, \quad k = n-1, n-2, \cdots, 1, 0 \\ f_n(x) = S_0 \end{cases}$$

2.3.4 实验结果及分析

x	算法	函数结果f	乘法次数	加法次数	用时 (秒)
0.1	算法 1	1.234567901234568	200000	100000	0.031
	算法 2	1.234567901234568	100000	100000	0.031
1	算法 1	5000150001	200000	100000	0.047
	算法 2	5000150001	100000	100000	0.047
2	算法 1	$Len = 30109$	200000	100000	0.56
	算法 2	$Len = 30109$	100000	100000	0.36

结果讨论：

X 越大时，算法的优越性更加明显，用时显著降低。

2.3.5 附录: 源代码

算法一:

```
from time import * # 引入时间库


countMul = 0 # 统计乘法次数
countAdd = 0 # 统计加法次数

startT = time() # 记录起始时间

x = 1
f = 1
```

```
t = x
for i in range(100000):
    f = f + (i + 2) * t
    t *= x
    countMul += 2
    countAdd += 1

print("result = ", f)

endT = time() # 记录结束时间
print("time = %.2g 秒\n" % (endT - startT))

print("乘法次数", countMul)
print("加法次数", countAdd)
```

算法二:

```
from time import * # 引入时间库


countMul = 0 # 统计乘法次数
countAdd = 0 # 统计加法次数

startT = time() # 记录起始时间

x = 1
f = 100001
for i in range(100000, 0, -1):
    f = f * x + i
    countMul += 1
    countAdd += 1

print("result = ", f)

endT = time() # 记录结束时间
print("time = %.2g 秒\n" % (endT - startT))

print("乘法次数", countMul)
print("加法次数", countAdd)
```

3 非线性方程的数值解法实验与分析

3.1 求解非线性方程的二分法

3.1.1 实验题目

用二分法求方程

$$7x^5 - 13x^4 - 21x^3 - 12x^2 + 58x + 3 = 0, \quad x \in [1, 2]$$

在区间 [1, 2] 上的根，设定停止条件为相邻求值得到的解之前的误差绝对值小于 $\varepsilon = 10^{-5}$。

3.1.2 算法介绍

原理: 若 $f \in C[a, b]$, 且 $f(a) \cdot f(b) < 0$, 则 f 在 (a, b) 上必有一根。
程序中的中点函数值 >0，则区间新上限值置为中点值。不断计算，直至 $|x_n - x_{n-1}| < \varepsilon_1$。

3.1.3 实验结果及分析

迭代次数	x_n	迭代次数	x_n
0	1.50000	9	1.30957
1	1.25000	10	1.30908
2	1.37500	11	1.30884
3	1.31250	12	1.30896
4	1.28125	13	1.30890
5	1.29688	14	1.30893
6	1.30469	15	1.30891
7	1.30859	16	1.30892
8	1.31055		

3.1.4 附录: 源代码

```
import math
from time import *

LIMIT = 1e-5
xlow = 1
xup = 2

startT = time() # 记录起始时间
```

```
9
10
11  def f(x): # 使用秦九韶算法减少运算次数
12      coefficient = [7, -13, -21, -12, 58, 3]
13      f = coefficient[0]
14      for i in range(1, len(coefficient)):
15          f = f * x + + coefficient[i]
16      return f
17
18
19  def sign(x):
20      if x == 0:
21          return 0
22      return int(math.copysign(1, x))
23
24
25
26  iter = 0
27  xn = 2
28  tmp = 1
29  while math.fabs(xn - tmp) >= LIMIT:
30      xmid = xlow + (xup - xlow) / 2
31      xn, tmp = xmid, xn
32      print("{:d} {:.5f}".format(iter, xmid))
33      iter += 1
34      if sign(f(xmid)) * sign(f(xlow)) < 0:
35          xup = xmid
36      else:
37          xlow = xmid
```

3.2 Python 绘图模拟非线性方程求解过程

3.2.1 实验题目

$$x = \varphi_5(x) = \left(\frac{-58x - 3}{7x^3 - 13x^2 - 21x - 12} \right)^{\frac{1}{2}}$$

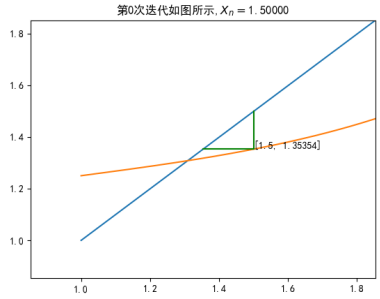
采用不动点迭代法求解方程。

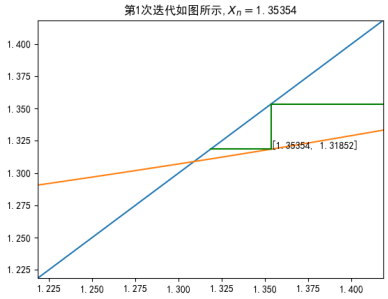
3.2.2 算法介绍

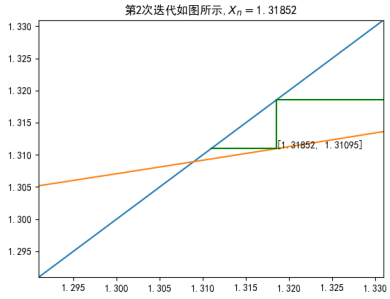
给定初值 $x_0 = 1.5$, 采用 $x_n = \varphi(x_{n-1})$ 的迭代形式求解。
停止条件设为 $|x_n - x_{n-1}| < \varepsilon = 10^{-5}$。

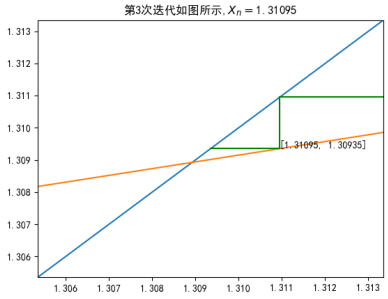
3.2.3 实验结果及分析

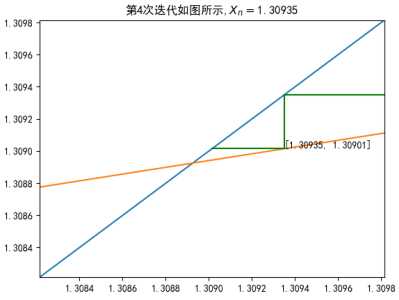
迭代次数	x_n	迭代次数	x_n
0	1.50000	5	1.30901
1	1.35354	6	1.30894
2	1.31852	7	1.30893
3	1.31095	8	1.30893
4	1.30935		

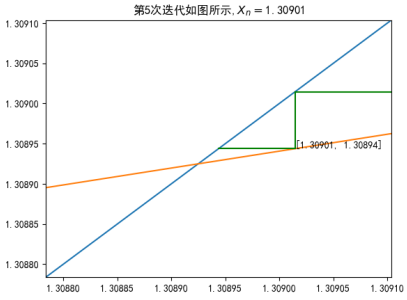


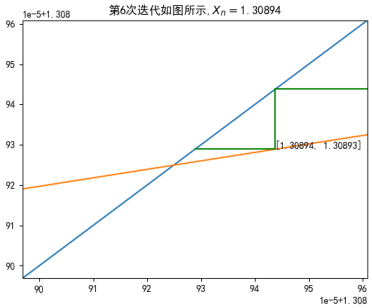


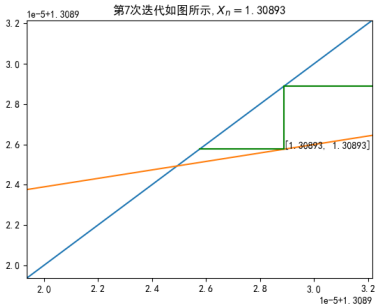












3.2.4 附录: 源代码

```python
import math
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams['font.sans-serif'] = ['SimHei'] # 中文字体设置-黑体
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题


# x = fai5(x)

def f(x):
    return pow((-58 * x - 3) / (7 * pow(x, 3) - 13 * pow(x, 2) - 21 * x - 12), 1 / 2)


LIMIT = 1e-5
xlow = 1
xup = 2
xn = 1.5
tmp = 2
iter = 0
points = []
while math.fabs(xn - tmp) >= LIMIT: # tmp 表示 x_n-1
    print("{:d} {:.5f}".format(iter, xn))
    iter += 1
    xn, tmp = f(xn), xn
    points.append([tmp, xn])
print("{:d} {:.5f}".format(iter, xn))


i = 0
a = 0.5
x0 = np.linspace(1, 2, 100)
y0 = np.linspace(1, 2, 100)
plt.plot(x0, y0)
```

```
x1 = np.linspace(1, 2, 10000)
y1 = np.array([f(t) for t in x1])
plt.plot(x1, y1)
for x, y in points:
    plt.title("第{:d}次迭代如图所示,$X_n=${:.5f}".format(i, round(x, 5)))

    result_x = round(x, 5)
    result_y = round(y, 5)
    plt.plot([x, x], [x, y], 'g')
    plt.text(x, y, [result_x, result_y])
    plt.pause(1)
    plt.xlim(y - a, y + a)
    plt.ylim(y - a, y + a)
    plt.plot([x, y], [y, y], 'g')
    # plt.savefig('./第{}次迭代.png'.format(i))
    plt.pause(1)
    a = a / 5
    i += 1
```

3.3 Aitken 和 Steffensen 方法加速求根

3.3.1 实验题目

编写 Aitken 和 Steffensen 方法加速例 2.2 的 5 种迭代格式的结果，写成分析报告（计算结果输出到文件，精度要求为 1e-5）

3.3.2 算法介绍

Aitken 方法:

对于一个收敛到 p 的数列 x_n ，如果其收敛阶为 1，那么

$$\lim_{n\to\infty}\frac{|x_{n+1}-p|}{|x_n-p|}=C, 0<C<1$$

可以认为当 $n\to\infty$ 时，有

$$\frac{x_n-p}{x_{n-1}-p}\approx\frac{x_{n-1}-p}{x_{n-2}-p}$$

解得

$$p\approx x_n-\frac{(x_n-x_{n-1})^2}{x_n-2x_{n-1}+x_{n-2}}$$

得到迭代数列

$$\hat{x}_n=x_n-\frac{(x_n-x_{n-1})^2}{x_n-2x_{n-1}+x_{n-2}}\overset{\text{或者}}{=}x_{n-2}-\frac{(x_{n-1}-x_{n-2})^2}{x_n-2x_{n-1}+x_{n-2}}, n=2,3,\ \mathrm{L}$$

称 x_n 为 Aitken 数列，其中 $\hat{x}_0 = x_0, \hat{x}_1 = x_1$。

算法思路:
已知初值为 x_0, 第一步，先求出 $x_1 = \phi(x_0)$, 可得 $\hat{x}_1 = x_1$;
第二步，先求出 $x_2\phi(x_1)$, 由迭代数列得

$$\hat{x}_2 = x_0 - \frac{(x_1 - x_0)^2}{x_2 - 2x_1 + x_0}$$

第三步，先求出 $x_3 = \phi(x_2)$, 由迭代数列得

$$\hat{x}_3 = x_1 - \frac{(x_2 - x_1)^2}{x_3 - 2x_2 + x_1}$$

Steffensen 方法：

$$x_n = \psi(x_{n-1}) = x_{n-1} - \frac{[\varphi(x_{n-1}) - x_{n-1}]^2}{\varphi(\varphi(x_{n-1})) - 2\varphi(x_{n-1}) + x_{n-1}}$$

由上式产生的序列称为 Steffensen 迭代序列。

算法思路:
已知初值为 x_0, 第一步，从 x_0 开始迭代，由 Steffensen 迭代序列得

$$\hat{x}_1 = x_0 - \frac{[\varphi(x_0) - x_0]^2}{\varphi(\varphi(x_0)) - 2\varphi(x_0) + x_0}$$

第二步，从 $\hat{x}_1$ 开始迭代, 由 Steffensen 迭代序列得

$$\hat{x}_2 = \hat{x}_1 - \frac{[\varphi(\hat{x}_1) - \hat{x}_1]^2}{\varphi(\varphi(\hat{x}_1)) - 2\varphi(\hat{x}_1) + \hat{x}_1}$$

以此类推，直到迭代停止。

3.3.3 实验结果及分析

表 1: Aitken 方法迭代过程数据

迭代次数 n	(1)	(2)	(3)	(4)	(5)
0	1.50000	1.50000	1.50000	1.50000	1.50000
1	1.50002	1.02102	-2.10438	1.30825	1.30751

2	0	0.307844	2.76904	1.30889	1.30887
3	...	-15.7335	2.76685	1.30892	1.30892
4		4.32142			
5		3.2787			
6		2.99777			
7		2.88398			
8		2.82991			
9		2.80201			
10		2.78689			
11		2.77845			
12		2.77363			
13		2.77085			
14		2.76922			
15		2.76826			
16		2.76769			
17		2.76735			
18		2.76715			
19		2.76703			
20		2.76696			
21		2.76691			
22		2.76689			

表 2: Steffensen 方法迭代过程数据

迭代次数 n	(1)	(2)	(3)	(4)	(5)
0	1.50000	1.50000	1.50000	1.50000	1.50000
1	1.50002	1.02102	-2.10438	1.30825	1.30751
2	1.50004	1.38449+0.543174j	2.65482	1.30892	1.30892
3	1.50007	1.94712-0.0143892j	2.76673	1.30892	1.30892
4	1.50009	3.47192+0.0760038j	2.76685		
5	1.50011	2.81754+0.00840694j			
6	1.50013	2.76729+0.000146022j			
7	1.50015	2.76685+2.38983e-08j			
8	1.50018				
...	...				

3213320	2.76685

3.3.4 附录: 源代码

```
def f(x):
  return 7 * x ** 5 - 13 * x ** 4 - 21 * x ** 3 - 12 * x ** 2 + 58 * x + 3


def phi1(x):
  return 7 * x ** 5 - 13 * x ** 4 - 21 * x ** 3 - 12 * x ** 2 + 59 * x + 3


def phi2(x):
  return ((13 * x ** 4 + 21 * x ** 3 + 12 * x ** 2 - 58 * x - 3) / 7) ** (1 / 5)


def phi3(x):
  return (13 + (21 / x) + (12 / (x ** 2)) - (58 / (x ** 3)) - (3 / (x ** 4))) / 7


def phi4(x):
  return ((12 * x ** 2 - 58 * x - 3) / (7 * x ** 2 - 13 * x - 21)) ** (1 / 3)


def phi5(x):
  return ((-58 * x - 3) / (7 * x ** 3 - 13 * x ** 2 - 21 * x - 12)) ** (1 / 2)


def Steffensen(x0, iternum, g, epsilon=1e-5):
  result = []
  with open("Steffensen.txt", "a+") as file:
      for i in range(iternum):
          if (g(g(x0)) - 2 * g(x0) + x0) != 0:
              x1_hat = x0 - (g(x0) - x0) ** 2 / (g(g(x0)) - 2 * g(x0) + x0)

              result.append(x1_hat)
              print("{:d}\t{:.6g}".format(i + 1, x1_hat)) # 第一列：迭代次数 第二列：迭代过程数据
              file.writelines("{:d}\t{:.6g}\n".format(i + 1, x1_hat))
              if abs(x1_hat - g(x1_hat)) < epsilon:
                  return result, x1_hat
              else:
                  x0 = x1_hat
          else:
              print("Steffensen方法失败")
              return None

  print("Steffensen方法失败")
  return None
```

```
45
46
47  def Aitken(x0, iternum, g, epsilon=1e-5):
48      try:
49          with open("Aitken.txt", "a+") as file:
50              for i in range(iternum):
51                  x1 = g(x0)
52                  x2 = g(x1)
53                  # print(x1,x2)
54                  if (x2 - 2 * x1 + x0) != 0:
55                      x2_hat = x2 - (x2 - x1) ** 2 / (x2 - 2 * x1 + x0)
56                      print("{:d}\t{:.6g}".format(i + 1, x2_hat)) # 第一列：迭代次数 第二列：迭代过程数据
57                      file.writelines("{:d}\t{:.6g}\n".format(i + 1, x2_hat))
58                      if abs(x2_hat - g(x2_hat)) < epsilon:
59                          return x2_hat
60                      else:
61                          x0 = x1
62                  else:
63                      print("Aitken方法失败")
64                      return None
65              print("Aitken方法失败")
66
67          return None
68      except OverflowError:
69          print("溢出")
70      except ZeroDivisionError:
71          print("分母为零")
72
73
74  if __name__ == '__main__':
75      x0 = 1.5 # 迭代初值
76      LIMIT = 1e-5 # 精度
77      iternum = 10000 # 最大迭代次数
78
79      # Aitken
80      print("Atiken1:")
81      root1 = Aitken(x0, iternum, phi1)
82      print("Atiken2:")
83      root2 = Aitken(x0, iternum, phi2)
84      print("Atiken3:")
85      root3 = Aitken(x0, iternum, phi3)
86      print("Atiken4:")
87      root4 = Aitken(x0, iternum, phi4)
88      print("Atiken5:")
89      root5 = Aitken(x0, iternum, phi5)
90      # Steffensen
91      print("Steffensen1:")
92      roots1 = Steffensen(x0, iternum, phi1)
93      print("Steffensen2:")
94      roots2 = Steffensen(x0, iternum, phi2)
```

```
print("Steffensen3:")
roots3 = Steffensen(x0, iternum, phi3)
print("Steffensen4:")
roots4 = Steffensen(x0, iternum, phi4)
print("Steffensen5:")
roots5 = Steffensen(x0, iternum, phi5)
```

3.4 综合实验：多种方法对比

3.4.1 实验题目

采用二分法、不动点迭代法、加速收敛方法、牛顿法、弦截法对例 2.1 方程进行求解。列出求解数据表格，并分析。（注意初始值的设置尽量相同）

3.4.2 算法介绍

牛顿法:

解非线性方程 $f(x)=0$ 的牛顿法是一种将非线性函数线性化的方法。取初值 $x_0 \approx p$, 将 $f(x)$ 做一阶泰勒展开：

$$f(x)=f\left(x_n\right)+f'\left(x_n\right)\left(x-x_n\right)+\frac{f''(\xi)}{2!}\left(x-x_n\right)^2, \xi \in [x_n, x]$$

去掉 2 阶及 2 阶以上项（即线性化），则有

$$f(x) \approx f\left(x_n\right)+f'\left(x_n\right)\left(x-x_n\right)$$

设$f'\left(x_n\right) \neq 0$, 用$x_{n+1}$ 代替x, 由$f(x)=0$ 得迭代公式

$$x_{n+1}=x_n-\frac{f\left(x_n\right)}{f'\left(x_n\right)}$$

使用此迭代公式的求根算法称为牛顿法。

定端点弦截法:

给定 x_0 和 x_1 的值，对于 $n=1,2,K$ 有

$$x_{n+1}=x_n-\frac{x_n-x_0}{f\left(x_n\right)-f\left(x_0\right)} f\left(x_n\right)$$

使用此迭代公式的求根算法称为定端点弦截法。

动端点弦截法:

给定 x_0 和 x_1 的值，对于 $n = 1, 2, K$ 有

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_0)} f(x_n)$$

使用此迭代公式的求根算法称为动端点弦截法。

3.4.3 实验结果及分析

迭代	二分法	不动点 φ_4	不动点 φ_5	Aitken 加速		Steffensen 加速		牛顿法	定截弦法	动截弦法
				φ_4	φ_5	φ_4	φ_5			
0	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	2	2
1	1.25	1.36538	1.35354	1.30825	1.30751	1.30825	1.30751	1.3261	1	1
2	1.375	1.32528	1.31852	1.30889	1.30887	1.30892	1.30892	1.30914	1.21359	1.21359
3	1.3125	1.31364	1.31095	1.30892	1.30892	1.30892	1.30892	1.30892	1.28755	1.34572
4	1.28125	1.31028	1.30935					1.30892	1.30474	1.30612
5	1.29688	1.30932	1.30901						1.30813	1.30885
6	1.30469	1.30904	1.30894						1.30878	1.30893
7	1.30859	1.30896	1.30893						1.3089	1.30892
8	1.31055	1.30893							1.30892	
9	1.30957								1.30892	
10	1.30908									
11	1.30884									
12	1.30896									
13	1.3089									
14	1.30893									
15	1.30891									
16	1.30892									
17	1.30893									

3.4.4 附录: 源代码

牛顿法:

```
import math
from time import *


# x = fai5(x)

def f(x): # 使用秦九韶算法减少运算次数
    coefficient = [7, -13, -21, -12, 58, 3]
    f = coefficient[0]
    for i in range(1, len(coefficient)):
```

```
        f = f * x + + coefficient[i]
    return f


def diff_f(x):
    coefficient = [35, -52, -63, -24, 58]
    f = coefficient[0]
    for i in range(1, len(coefficient)):
        f = f * x + + coefficient[i]
    return f


LIMIT = 1e-5
xlow = 1
xup = 2
xn = 1.5
tmp = 2
iter = 0

while math.fabs(xn - tmp) >= LIMIT: # tmp 表示 x_n-1
    print("{:d} {:.5f}".format(iter, xn))
    iter += 1
    xn, tmp = xn - f(xn) / diff_f(xn), xn
print("{:d} {:.5f}".format(iter, xn))
```

定截弦法：

```
import math
from time import *


# x = fai5(x)

def f(x): # 使用秦九韶算法减少运算次数
    coefficient = [7, -13, -21, -12, 58, 3]
    f = coefficient[0]
    for i in range(1, len(coefficient)):
        f = f * x + + coefficient[i]
    return f


LIMIT = 1e-5
x0 = 1
xn = 2
tmp = 1.5
iter = 1
print("{:d} {:.5f}".format(0, 1))
while math.fabs(xn - tmp) >= LIMIT: # tmp 表示 x_n-1
    print("{:d} {:.5f}".format(iter, xn))
```

```
    iter += 1
    xn, tmp = xn - ((xn - x0) / (f(xn) - f(x0))) * f(xn), xn
print("{:d} {:.5f}".format(iter, xn))
```

动截弦法：

```
import math
from time import *


# x = fai5(x)

def f(x): # 使用秦九韶算法减少运算次数
    coefficient = [7, -13, -21, -12, 58, 3]
    f = coefficient[0]
    for i in range(1, len(coefficient)):
        f = f * x + + coefficient[i]
    return f


LIMIT = 1e-5
xn = 2
tmp = 1
iter = 1
print("{:d} {:.5f}".format(0, 1))
while math.fabs(xn - tmp) >= LIMIT: # tmp 表示 x_n-1
    print("{:d} {:.5f}".format(iter, xn))
    iter += 1
    xn, tmp = xn - ((xn - tmp) / (f(xn) - f(tmp))) * f(xn), xn
print("{:d} {:.5f}".format(iter, xn))
```

4 线性方程组的数值解法实验与分析

4.1 *Gauss* 消去法与列主元 *Gauss* 消去法的比较

4.1.1 实验题目

$$\begin{cases} 2.51x_1 + 1.48x_2 + 4.53x_3 = 0.05 \\ 1.48x_1 + 0.93x_2 - 1.3x_3 = 1.03 \\ 2.68x_1 + 3.04x_2 - 1.48x_3 = -0.53 \end{cases}$$

(1) 使用 *Gauss* 消去法解以上方程组；

(2) 使用列主元 *Gauss* 消去法解以上方程组；

(3) 检验 (1) 和 (2) 中得到的两个解中，哪一个更接近准确解。

(计算过程中保留三位小数，方程的准确解为 $x_1 = 1.4530,\ x_2 = -1.589195,\ x_3 = -0.2748947$)

4.1.2 算法介绍

Gauss 消元法：

消元公式: $(k = 1, 2, \cdots, n-1)$

$$l_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)} \quad (i = k+1, \ldots, n)$$

$$\begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)} \\ b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)} \end{cases} \quad (i, j = k+1, \ldots, n)$$

回代过程:

$$\begin{cases} x_n = b_n^{(n)} / a_{nn}^{(n)} \\ x_i = \frac{b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)} x_j}{a_{ii}^{(i)}} \end{cases} \quad (i = n-1, \ldots, 1)$$

列主元 *Gauss* 消去法:

每次消元前选出待消元（按列）的未知数对应的系数中绝对值最大的元素为主元素，然后交换两行后再消元。

Step k :

(1) 选取 $|\boldsymbol{a}_{i_k,k}| = \max_{k \le i \le n} |\boldsymbol{a}_{ik}| \neq 0$

(2) If $i_k \neq k$ then 交换第 k 行与第 i_k 行;

(3) 消元 (消元过程即 *Gauss* 消去法)

消元完回代即可求解出 x。

4.1.3 问题求解与分析

(1):

$$[A|b] = \begin{bmatrix} 2.51 & 1.48 & 4.53 & 0.05 \\ 0 & 0.057 & -3.973 & 1.0 \\ 0 & 0 & 95.375 & -26.179 \end{bmatrix}$$

回代得: $x_1 = 1.431$, $x_2 = -1.554$, $x_3 = -0.274$。

(2):

$$[A|b] = \begin{bmatrix} 2.68 & 3.04 & -1.48 & -0.53 \\ 0 & -1.368 & 5.917 & 0.547 \\ 0 & 0 & -3.72 & 1.024 \end{bmatrix}$$

回代得: $x_1 = 1.453$, $x_2 = -1.589$, $x_3 = -0.275$。

4.1.4 算法源代码

Gauss 消去法:

```
# Gauss消去法解方程组
for k in range(0, n - 1):
    for i in range(k + 1, n):
        l = round(aug_matrix[i][k] / aug_matrix[k][k], 3)
        for j in range(k + 1, n + 1):
            aug_matrix[i][j] = round(aug_matrix[i][j] - l * aug_matrix[k][j], 3)
        aug_matrix[i][k] = 0

for a in aug_matrix:
    print(a)
print()
x = [0.0] * n

x[n - 1] = aug_matrix[n - 1][n] / aug_matrix[n - 1][n - 1]

for k in range(n - 1, -1, -1):
    sum = 0
    for j in range(k + 1, n):
        sum += aug_matrix[k][j] * x[j]
    x[k] = round((aug_matrix[k][n] - sum) / aug_matrix[k][k], 3)
print(x)
print()
```

列主元 *Gauss* 消去法:

```
# 列主元Gauss消去法解方程组
aug_matrix = copy.deepcopy(tmp)
```

```
3
4
5   def swap(k, n):
6       global aug_matrix
7       ans = -1
8       maxn = -1
9       for i in range(k, n):
10          if ans < math.fabs(aug_matrix[i][k]):
11              ans = math.fabs(aug_matrix[i][k])
12              maxn = i
13      if maxn == k:
14          return
15      aug_matrix[k][k::], aug_matrix[maxn][k::] = aug_matrix[maxn][k::], aug_matrix[k][k::]
16
17
18  for k in range(0, n - 1):
19      swap(k, n)
20      for i in range(k + 1, n):
21          l = round(aug_matrix[i][k] / aug_matrix[k][k], 3)
22          for j in range(k + 1, n + 1):
23              aug_matrix[i][j] = round(aug_matrix[i][j] - l * aug_matrix[k][j], 3)
24          aug_matrix[i][k] = 0
25
26  for a in aug_matrix:
27      print(a)
28  print()
29  x = [0.0] * n
30
31  x[n - 1] = aug_matrix[n - 1][n] / aug_matrix[n - 1][n - 1]
32
33  for k in range(n - 1, -1, -1):
34      sum = 0
35      for j in range(k + 1, n):
36          sum += aug_matrix[k][j] * x[j]
37      x[k] = round((aug_matrix[k][n] - sum) / aug_matrix[k][k], 3)
38  print(x)
```

4.2 列主元 $Gauss-Jordan$ 消去法、LU 分解法的对比

4.2.1 实验题目

用列主元 $Gauss-Jordan$ 消去法、LU 分解法解方程组 $Ax=b$

$$A=\begin{bmatrix} 1 & 2 & 1 & -2 \\ 2 & 5 & 3 & -2 \\ -2 & -2 & 3 & 5 \\ 1 & 3 & 2 & 3 \end{bmatrix}, \quad b=(4,7,-1,0)^{\mathrm{T}}$$

4.2.2 算法介绍

列主元 $Gauss-Jordan$ 消去法:
与 $Gauss$ 消去法的主要区别：
每步不计算 l_{ik}, 而是先将当前主元 ${a_{kk}}^{(k)}$ 变为 1
把 ${a_{kk}}^{(k)}$ 所在列的上、下元素全消为 0
$A\vec{x}=\vec{b}\Rightarrow I\vec{x}=A^{-1}\vec{b}$

LU 分解法:
$\boldsymbol{A}\overrightarrow{\boldsymbol{x}}=\boldsymbol{LU}\overrightarrow{\boldsymbol{x}}=\overrightarrow{\boldsymbol{b}}_i \quad (i=1,2,\cdots,k)$
令 $U\vec{x}=\vec{y}$ ，则 $L\vec{y}=\vec{b_i}$
解出 $\vec{y}$ 代入上式即得 $\overrightarrow{\boldsymbol{x}}$ ，那么解方程组变成相继解两个三角方程组。
对于 $\boldsymbol{L}\overrightarrow{\boldsymbol{y}}=\overrightarrow{\boldsymbol{b}}_i$
有 $y_i=b_i-\sum_{k=1}^{i-1}l_{ik}y_k \quad i=1,2,\cdots,n$
对于 $\boldsymbol{U}\overrightarrow{\boldsymbol{x}}=\overrightarrow{\boldsymbol{y}}$
有 $x_i=\left(y_i-\sum_{k=i+1}^{n}u_{ik}x_k\right)/u_{ii} \quad i=n,n-1,\cdots,1$

4.2.3 问题求解与分析

列主元 $Gauss-Jordan$ 消去法:

$$[A\mid b]=\begin{bmatrix}1&0&0&0&2\\0&1&0&0&-1\\0&0&1&0&2\\0&0&0&1&-1\end{bmatrix}$$

得: $x=(2,-1,2,-1)^{\mathrm{T}}$ 。

LU 分解法:

$$[U\mid y]=\begin{bmatrix}1&2&1&-1&4\\0&1&1&2&-1\\0&0&3&-3&9\\0&0&0&3&-3\end{bmatrix}$$

得: $x=(2,-1,2,-1)^{\mathrm{T}}$ 。
列主元 $Gauss-Jordan$ 消去法比 $Gauss$ 消去法多出 $O(\frac{n^2}{3})$ 的比较步骤
LU 分解法总计算量为

$$\frac{n^3+3n^2-n}{3}$$

与 *Gauss* 消去法相同

4.2.4 算法源代码

列主元 $Gauss-Jordan$ 消去法:

```python
# 列主元Gauss-Jordan消去法
def swap(k, n):
    global aug_matrix
    ans = -1
    maxn = -1
    for i in range(k, n):
        if ans < math.fabs(aug_matrix[i][k]):
            ans = math.fabs(aug_matrix[i][k])
            maxn = i
    if maxn != k:
        aug_matrix[k][k::], aug_matrix[maxn][k::] = aug_matrix[maxn][k::], aug_matrix[k][k::]
    l = aug_matrix[k][k]
    for j in range(k, n + 1):
        aug_matrix[k][j] /= l


for k in range(0, n):
    swap(k, n)
    for i in range(0, n):
        if i == k:
            continue
        l = round(aug_matrix[i][k] / aug_matrix[k][k], 3)
        for j in range(k + 1, n + 1):
            aug_matrix[i][j] = round(aug_matrix[i][j] - l * aug_matrix[k][j], 3)
        aug_matrix[i][k] = 0

for a in aug_matrix:
    print(a)
print()

x = [0.0] * n

for i in range(n):
    x[i] = aug_matrix[i][n]
print('列主元Gauss-Jordan消去法:')
print(x)
```

LU 分解法:

```python
# LU分解法解方程组
aug_matrix = copy.deepcopy(tmp)
L = [[0.0] * n for _ in range(n)]
```

```
for k in range(n-1):
    for i in range(k+1, n):
        L[i][k] = aug_matrix[i][k] / aug_matrix[k][k]
        for j in range(n):
            aug_matrix[i][j] = aug_matrix[i][j] - L[i][k] * aug_matrix[k][j]
for i in range(n):
    L[i][i] = 1.0

# 求 y
y = [0.0] * n
y[0] = aug_matrix[0][n]
for i in range(1, n):
    y[i] = aug_matrix[i][n]
    for j in range(i):
        y[i] = y[i] - L[i][j] * y[j]
# print(y)

# aug_matrix 为ux=y 的增广矩阵
for i in range(n):
    aug_matrix[i][n] = y[i]

for a in aug_matrix:
    print(a)
print()
# 求x
x = [0.0] * n
x[n - 1] = aug_matrix[n - 1][n] / aug_matrix[n - 1][n - 1]

for k in range(n - 1, -1, -1):
    sum = 0
    for j in range(k + 1, n):
        sum += aug_matrix[k][j] * x[j]
    x[k] = round((aug_matrix[k][n] - sum) / aug_matrix[k][k], 3)
print('LU分解法解方程组:')
print(x)
print()
```

4.3 范数和条件数的求解

4.3.1 实验题目

求出

$$\boldsymbol{A}=\begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix}$$

的 $\|A\|_\infty, \|A\|_1, \|A\|_2$ 和 $\mathrm{Cond}(A)_2$

4.3.2 预备知识

$\|A\|_1 = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$ （行和范数）
$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|$ （列和范数）
$\|\boldsymbol{A}\|_2 = \sqrt{\rho\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)}$ （谱范数 /* spectral norm */ ）
$\mathrm{cond}(A)_2 = \sqrt{\lambda_{\max}\left(A^T A\right) / \lambda_{\min}\left(A^T A\right)}$

4.3.3 问题求解与分析

$\|A\|_\infty = 4$
$\|A\|_1 = 4$
$\|A\|_2 = 3.6180339887498953$
$\mathrm{Cond}(A)_2 = 9.472135954999562$

4.3.4 算法源代码

```python
import math
from math import inf
import numpy as np

A = np.array([[-2, 1, 0, 0], [1, -2, 1, 0], [0, 1, -2, 1], [0, 0, 1, -2]])

# inf 范数
norm_inf = 0
for i in range(A.shape[0]):
    tmp = 0
    for j in range(A.shape[1]):
        tmp += abs(A[i][j])
    if tmp > norm_inf:
        norm_inf = tmp
print(norm_inf)
# print(np.linalg.norm(A, ord=inf))

# 1 范数
norm_1 = 0
for j in range(A.shape[1]):
    tmp = 0
    for i in range(A.shape[0]):
        tmp += abs(A[i][j])
    if tmp > norm_1:
        norm_1 = tmp
print(norm_1)
# print(np.linalg.norm(A, ord=1))
```

```
28
29  # 2 范数
30  eig, eig_vector = np.linalg.eig(np.dot(A.T, A)) # 特征值,  特征向量
31  norm_2 = math.sqrt(max(eig))
32  print(norm_2)
33  # print(np.linalg.norm(A, ord=2))
34
35  # Cond(A)_2
36  CondA = math.sqrt(max(eig)) / math.sqrt(min(eig))
37  print(CondA)
```

4.4 微小扰动对方程组求解的稳定性分析

4.4.1 实验题目

解如下方程组

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} x = b$$

其中 $\boldsymbol{b} = (32, 23, 33, 31)^{\mathrm{T}}$, 但是由于某些原因使得方程的右端被修改为 $b^* = (32.1, 22.9, 33.1, 30.9)^{\mathrm{T}}$, 求出方程的解, 并算出在 ∞-范数和 1-范数下求出的解与准确解之间、扰动方程的右端项和原右端项的相对误差, 说明原因。

4.4.2 预备知识

解与准确解的相对误差:

$$\delta\bar{x} = \vec{x}^* - \vec{x} \Rightarrow \frac{\|\delta\vec{x}\|_\infty}{\|\vec{x}\|_\infty}$$
$$\delta\bar{x} = \vec{x}^* - \vec{x} \Rightarrow \frac{\|\delta\vec{x}\|_1}{\|\vec{x}\|_1}$$

扰动方程的右端项和原右端项的相对误差:

$$\delta\bar{b} = \vec{b}^* - \vec{b} \Rightarrow \frac{\|\delta\vec{b}\|_\infty}{\|\vec{b}\|_\infty}$$
$$\delta\bar{b} = \vec{b}^* - \vec{b} \Rightarrow \frac{\|\delta\vec{b}\|_1}{\|\vec{b}\|_1}$$

4.4.3 问题求解与分析

求得方程的准确解为: $x = (1, 1, 1, 1)^{\mathrm{T}}$
解为: $x^* = (9.2, -12.6, 4.5, -1.1)^{\mathrm{T}}$

$$\frac{\|\delta\vec{x}\|_\infty}{\|\vec{x}\|_\infty} = 13.5999$$
$$\frac{\|\delta\vec{x}\|_1}{\|\vec{x}\|_1} = 6.8500$$
$$\frac{\|\delta\vec{b}\|_\infty}{\|\vec{b}\|_\infty} = 0.003030$$
$$\frac{\|\delta\vec{b}\|_1}{\|\vec{b}\|_1} = 0.0033613$$

虽然 $\frac{\|\delta\vec{b}\|}{\|\vec{b}\|} < 0.01$
但是对于求得的解与准确解之间相对误差极大。

4.4.4 算法源代码

```python
import numpy as np

A = np.array([[10, 7, 8, 7], [7, 5, 6, 5], [8, 6, 10, 9], [7, 5, 9, 10]], dtype=float)

b1 = np.array([32, 23, 33, 31])
b2 = np.array([32.1, 22.9, 33.1, 30.9])

result1 = np.linalg.solve(A, b1)
print(result1)
result2 = np.linalg.solve(A, b2)
print(result2)

# 解与准确解的相对误差
mis_result = result2 - result1
print(np.linalg.norm(mis_result, ord=np.inf) / np.linalg.norm(result1, ord=np.inf))
print(np.linalg.norm(mis_result, ord=1) / np.linalg.norm(result1, ord=1))


# 扰动方程的右端项和原右端项的相对误差
mis_b = b2 - b1
print(np.linalg.norm(mis_b, ord=np.inf) / np.linalg.norm(b1, ord=np.inf))
print(np.linalg.norm(mis_b, ord=1) / np.linalg.norm(b1, ord=1))
```

4.5 $Jacobi$ 和 $Gauss-Seidel$ 迭代法收敛性

4.5.1 实验题目

设有系数矩阵

$$(1)\begin{bmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix} \quad (2)\begin{bmatrix} 2 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & -2 \end{bmatrix}$$

分别检验以上系数矩阵用 $Jacobi$ 迭代法和 $Gauss-Seidel$ 迭代法是否收敛

4.5.2 预备知识

设有方程组 $\vec{x}=B\vec{x}+\vec{g}$，对任意初始向量 $\vec{x}^{(0)}$ 及任意常数向量 $\vec{g}$，解此方程组的迭代法 (即 $\vec{x}^{(k+1)}=B\vec{x}^{(k)}+\vec{g}, k\to\infty$) 收敛的充要条件是

$$\rho(B)<1$$

其中 $\rho(B)=\max_{1\le i\le n}|\lambda_i|$, λ_i 为 B 的特征值。

4.5.3 问题求解与分析

$\rho(B_j)=1.0809e-05$, $\rho(B_s)=2.0$

系数矩阵 1:$Jacobi$ 迭代法收敛，$Gauss-Seidel$ 迭代法不收敛。

$\rho(B_j)=1.118$, $\rho(B_s)=0.5$

系数矩阵 2:$Jacobi$ 迭代法不收敛，$Gauss-Seidel$ 迭代法收敛。

4.5.4 算法源代码

```
import numpy as np

A1 = np.mat([[1, 2, -2], [1, 1, 1], [2, 2, 1]])
A2 = np.mat([[2, -1, 1], [1, 1, 1], [1, 1, -2]])


def judge_convergence(B):
    eig, _ = np.linalg.eig(B)
    print(np.max(abs(eig)))
    if np.max(abs(eig)) < 1:
        return True
    else:
        return False


```

```
16  def solve(A):
17      D = np.mat(np.diag(np.diag(A)))
18      L = np.tril(A, -1)
19      U = np.triu(A, 1)
20      B_j = np.dot(-D.I, L + U)
21      B_s = np.dot(-(D + L).I, U)
22      print('Jacobi迭代法:', judge_convergence(B_j))
23      print('Gauss_Seidel迭代法:', judge_convergence(B_s))
24
25
26  solve(A1)
27  print()
28  solve(A2)
```

4.6 *Jacobi* 和 *Gauss − Seidel* 迭代法解方程组

4.6.1 实验题目

$$\begin{cases} 11x_1 - 3x_2 - 2x_3 = 3 \\ -x_1 + 5x_2 - 3x_3 = 6 \\ -2x_1 - 12x_2 + 19x_3 = -7 \end{cases}$$

取初值为$(0,0,0)^{\mathrm{T}}$, 写出前3 次迭代的结果。

4.6.2 预备知识

Jacobi 迭代法:

$$\begin{aligned} A\vec{x} = \vec{b} &\Leftrightarrow (D+L+U)\vec{x} = \vec{b} \\ &\Leftrightarrow D\vec{x} = -(L+U)\vec{x} + \vec{b} \quad (\text{两边左乘} D^{-1}) \\ &\Leftrightarrow \vec{x}^{(k+1)} = -D^{-1}(L+U)\vec{x}^{(k)} + D^{-1}\vec{b} \\ &\quad x^{(k+1)} = B_{\mathrm{J}}x^{(k)} + \vec{g}_{\mathrm{J}} \end{aligned}$$

Gauss − Seidel 迭代法:

$$
\begin{aligned}
&[L+D+U]x=b\\
&Dx=-Lx-Ux+b\\
&Dx^{(k+1)}=-Lx^{(k+1)}-Ux^{(k)}+b\\
&\Leftrightarrow (D+L)\vec{x}^{(k+1)}=-U\vec{x}^{(k)}+\vec{b}\\
&\Leftrightarrow \vec{x}^{(k+1)}=-(D+L)^{-1}U\vec{x}^{(k)}+(D+L)^{-1}\vec{b}\\
&\quad x^{(k+1)}=B_{\mathrm{s}}x^{(k)}+\vec{g}_{\mathrm{s}}
\end{aligned}
$$

4.6.3 问题求解与分析

Jacobi 迭代法:

迭代次数	x_1	x_2	x_3
0	0.0	0.0	0.0
1	0.27272727	1.2	-0.36842105
2	0.53301435	1.03349282	0.41818182
3	0.63062201	1.55751196	0.34041803

$Gauss-Seidel$ 迭代法:

迭代次数	x_1	x_2	x_3
0	0.0	0.0	0.0
1	0.27272727	1.25454545	0.45263158
2	0.69717268	1.61101348	0.72244775
3	0.84344872	1.80215839	0.85856832

分析: 该系数矩阵用 *Jacobi* 迭代法和 $Gauss-Seidel$ 迭代法均收敛。且该系数矩阵 $Gauss-Seidel$ 迭代法比 *Jacobi* 迭代法收敛速度更快。

4.6.4 算法源代码

```
1  import numpy as np
2
3  A = np.mat([[11, -3, -2], [-1, 5, -3], [-2, -12, 19]])
4  b = np.mat([3, 6, -7]).reshape(-1, 1)
5
6
7  def judge_convergence(B): # 利用谱半径来判断是否收敛
```

```
    eig, _ = np.linalg.eig(B)
    if np.max(abs(eig)) < 1:
        return True
    else:
        return False


def Jacobi(A, b):
    D = np.mat(np.diag(np.diag(A)))
    L = np.tril(A, -1)
    U = np.triu(A, 1)
    B_j = np.dot(-D.I, L + U)
    g_j = np.dot(D.I, b)
    if judge_convergence(B_j):
        x = np.mat(np.zeros(shape=(A.shape[0], 1)))
        print('初始值:', x.T)
        for i in range(3):
            x = np.dot(B_j, x) + g_j
            print('第{:d}次迭代: '.format(i+1), x.T)
    else:
        print("此系数矩阵Jacobi迭代法不收敛！")


def Gauss_Seidel(A, b):
    D = np.mat(np.diag(np.diag(A)))
    L = np.tril(A, -1)
    U = np.triu(A, 1)
    B_s = np.dot(-(D + L).I, U)
    g_s = np.dot((D + L).I, b)
    if judge_convergence(B_s):
        x = np.mat(np.zeros(shape=(A.shape[0], 1)))
        print('初始值:', x.T)
        for i in range(3):
            x = np.dot(B_s, x) + g_s
            print('第{:d}次迭代: '.format(i+1), x.T)
    else:
        print("此系数矩阵Gauss_Seidel迭代法不收敛！")


Jacobi(A, b)
print()
Gauss_Seidel(A, b)
```

4.7 综合实验：直接法和迭代法求解病态方程组

4.7.1 实验题目

系数矩阵为 $Hilbert$ 阵，对解全为 1 的方程组，随着 $n = 2, 3, \cdots\cdots$ 的增加，编写程序，测试和分析利用直接法和迭代法求解方程组的结果差别。

$$H_n = \begin{bmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \cdots & \frac{1}{2n-1} \end{bmatrix}$$

4.7.2 实验准备

直接法采用 $Gauss$ 消元法进行求解。

迭代法采用 $Jacobi$ 迭代法，$Gauss_Seidel$ 迭代法求解。

x 的准确解为 $x = (1, 1, 1, \cdots, 1)^{\mathrm{T}}$

利用下式计算各方法所求出的解与准确解之间的误差。

$$\delta\bar{x} = \vec{x}^* - \vec{x} \Rightarrow \frac{\|\delta\vec{x}\|_\infty}{\|\vec{x}\|_\infty}$$

4.7.3 实验结果与分析

选取 $n = 2, 5, 10, 20$ 进行分析。

	解	误差
$Gauss$	$x = (1.0, 1.0)^{\mathrm{T}}$	1.1102e-15
$Jacobi$	$x = (1.0, 1.0)^{\mathrm{T}}$	1.1339e-07
$Gauss - Seidel$	$x = (1.0, 1.0)^{\mathrm{T}}$	7.0790e-07
准确解	$x = (1, 1)^{\mathrm{T}}$	

表 3: $n = 2$

从表 1 可以看出，n=2 时，三种方法求出的解都比较精确，误差极小。

	解	误差
$Gauss$	$x = (1.0, 1.0, 1.0, 1.0, 1.0)^{\mathrm{T}}$	6.23424e-12
$Jacobi$	不收敛	
$Gauss - Seidel$	$x = (0.99978, 1.00437, 0.98045, 1.03016, 0.98504)^{\mathrm{T}}$	0.069257
准确解	$x = (1, 1, 1, 1, 1)^{\mathrm{T}}$	

表 4: $n = 5$

从表 2 可以看出，n=5 时，Jacobi 不收敛，而 Gauss-Seidiel 方法误差上升到 0.069257，Gauss 方法误差仍较小。

	解	误差
$Gauss$	$x = (1.0, 1.0, \cdots, 0.99991, 1.00004, 0.99999)^{\mathrm{T}}$	3.11177e-4
$Jacobi$	不收敛	
$Gauss - Seidel$	$x = (0.99688, 1.06466, \cdots, 1.01739, 0.95543, 0.88949)^{\mathrm{T}}$	0.64467
准确解	$x = (1, 1, \cdots, 1, 1, 1)^{\mathrm{T}}$	

表 5: $n = 10$

从表 3 可以看出，n=10 时，Jacobi 方法不收敛，而 Gauss-Seidel 与 Gauss 方法所求出的解的误差继续上升。

	解	误差
$Gauss$	$x = (1.0, 0.9999, \cdots, 1.23974, 0.99395, 1.03248)^{\mathrm{T}}$	39.541
$Jacobi$	不收敛	
$Gauss - Seidel$	不收敛	
准确解	$x = (1, 1, \cdots, 1, 1, 1)^{\mathrm{T}}$	

表 6: $n = 20$

从表 4 可以看出，n=20 时，Jacobi 方法，Gauss-Seidel 方法不收敛，而 Gauss 方法所求出解已经上升至 39.541。

4.7.4 算法源代码

```
import numpy as np


def Gauss(aug_matrix):
    for k in range(0, n - 1):
        for i in range(k + 1, n):
            l = aug_matrix[i][k] / aug_matrix[k][k]
            for j in range(k + 1, n + 1):
                aug_matrix[i][j] = aug_matrix[i][j] - l * aug_matrix[k][j]
            aug_matrix[i][k] = 0
```

```
11
12      x = np.ones(n)
13
14      x[n - 1] = aug_matrix[n - 1][n] / aug_matrix[n - 1][n - 1]
15
16      for k in range(n - 1, -1, -1):
17          sum = 0
18          for j in range(k + 1, n):
19              sum += aug_matrix[k][j] * x[j]
20          x[k] = (aug_matrix[k][n] - sum) / aug_matrix[k][k]
21      print(np.round(x, 5))
22      print('误差:', np.linalg.norm(x - x_exact, ord=np.inf) / np.linalg.norm(x_exact, ord=np.inf)
            )
23
24
25  def judge_convergence(B):
26      eig, _ = np.linalg.eig(B)
27      if np.max(abs(eig)) < 1:
28          return True
29      else:
30          return False
31
32
33  def Jacobi(A, b):
34      D = np.mat(np.diag(np.diag(A)))
35      L = np.tril(A, -1)
36      U = np.triu(A, 1)
37      B_j = np.dot(-D.I, L + U)
38      g_j = np.dot(D.I, b)
39      if judge_convergence(B_j):
40          limit = 1e-7
41          x0 = np.mat(np.zeros(shape=(A.shape[0], 1)))
42          x = np.dot(B_j, x0) + g_j
43          while np.min(abs(x - x0)) > limit:
44              x0 = x
45              x = np.dot(B_j, x) + g_j
46          print(np.round(x.T, 5))
47          print('误差:', np.linalg.norm(x.reshape(1, -1) - x_exact, ord=np.inf) / np.linalg.norm(
                x_exact, ord=np.inf))
48      else:
49          print("该矩阵不收敛！")
50
51
52  def Gauss_Seidel(A, b):
53      D = np.mat(np.diag(np.diag(A)))
54      L = np.tril(A, -1)
55      U = np.triu(A, 1)
56      B_s = np.dot(-(D + L).I, U)
57      g_s = np.dot((D + L).I, b)
58      if judge_convergence(B_s):
```

```
        limit = 1e-7
        x0 = np.mat(np.zeros(shape=(A.shape[0], 1)))
        x = np.dot(B_s, x0) + g_s
        while np.min(np.fabs(x - x0)) > limit:
            x0 = x
            x = np.dot(B_s, x) + g_s
        print(np.round(x.T, 5))
        print('误差:', np.linalg.norm(x.reshape(1, -1) - x_exact, ord=np.inf) / np.linalg.norm(
            x_exact, ord=np.inf))
    else:
        print("该矩阵不收敛！")


nums = [2, 5, 10, 20]
for n in nums:
    print('n =', n)
    x_exact = np.ones(n).reshape(-1, 1)
    Hilbert = np.empty((n, n))
    x = np.ones(n).reshape(-1, 1)

    for i in range(0, n):
        for j in range(0, n):
            Hilbert[i][j] = 1 / (j + 1 + i)
    b = np.dot(Hilbert, x)

    # Gauss
    Gauss(np.concatenate((Hilbert, b), axis=-1))

    # 迭代法
    x = np.mat(np.ones(n).reshape(-1, 1))
    b = np.dot(Hilbert, x)
    print('Jacobi:')
    Jacobi(Hilbert, b)
    print('Gauss_Seidel:')
    Gauss_Seidel(Hilbert, b)
    print()
```

5　插值法的实验与分析

5.1　综合实验：物体运动轨迹的插值预测

5.1.1　实验题目

采用拉格朗日插值、分段线性插值、样条插值等方法进行插值，估计 $x=-3.75$ 和 0.25 位置的 y 坐标值，并绘制插值函数的图形，根据结果，确定一种最好的插值方法。

5.1.2　算法介绍

拉格朗日插值法:

构造 $x_0,\cdots,x_n$ 处的插值基函数。

对于 x_i 的插值基函数 $l_i(x)$, 它在 $x_0,\cdots,x_{i-1},x_{i+1},\cdots,x_n$ 上都是零点, 其次数不高于 n, 因此可以假设

$$l_i(x)=A\left(x-x_0\right)\cdots\left(x-x_{i-1}\right)\left(x-x_{i+1}\right)\cdots\left(x-x_n\right)$$

又由于 $l_i(x)$ 在 x_i 点上的函数值是 1 , 可以解得

$$A=\frac{1}{\left(x_i-x_0\right)\cdots\left(x_i-x_{i-1}\right)\left(x_i-x_{i+1}\right)\cdots\left(x_i-x_n\right)}$$

因此, 有

$$l_i(x)=\frac{\left(x-x_0\right)\cdots\left(x-x_{i-1}\right)\left(x-x_{i+1}\right)\cdots\left(x-x_n\right)}{\left(x_i-x_0\right)\cdots\left(x_i-x_{i-1}\right)\left(x_i-x_{i+1}\right)\cdots\left(x_i-x_n\right)},\quad i=1,2,\cdots,n$$

以 $y_0,y_1,\cdots,y_n$ 作为基函数的组合系数, 得到 n 次拉格朗日插值多项式

$$L_n(x)=\sum_{i=0}^{n}y_il_i(x)$$

n 次插值的截断误差公式

$$R(x)=f(x)-\varphi_n(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}\left(x-x_0\right)\left(x-x_1\right)\cdots\left(x-x_n\right)$$

牛顿多项式插值法:

$$f\left[x_0,x_1,\cdots,x_n\right]=\frac{f\left[x_0,x_1,\cdots,x_{n-2},x_n\right]-f\left[x_0,x_1,\cdots,x_{n-2},x_{n-1}\right]}{x_n-x_{n-1}}$$

为函数 $f(x)$ 在节点 $x_0, x_1, \ldots, x_n$ 处的 n 阶差商

1) 通过已知插值节点的函数值建立差商表

2) 以相应的差商作为系数就可以得到插值多项式

构造差商表

x_i	$f(x_i)$	一阶	二阶	三阶
x_0	$f(x_0)$	$f[x_0,x_1]$	$f[x_0,x_1,x_2]$	$f[x_0,x_1,x_2,x_3]$
x_1	$f(x_1)$	$f[x_1,x_2]$	$f[x_1,x_2,x_3]$	
x_2	$f(x_2)$	$f[x_2,x_3]$		
x_3	$f(x_3)$	...		
...	...			

$$\varphi_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

$$a_i = f[x_0, x_1, \cdots, x_i]$$

分段线性插值法:

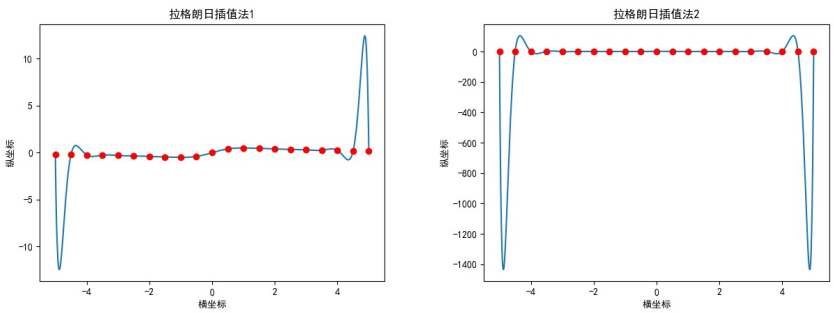
给定 $n+1$ 个插值节点 $x_0 < x_1 < \cdots < x_n$ 以及在这些插值节点上的函数值 $y_0, y_1, \cdots, y_n$, 构造一个分段插值函数 $P(x)$, 使得对任意 $i = 0, 1, 2, \cdots, n$, 有 $P(x_i) = y_i$, 并且在每个子区间 $[x_i, x_{i+1}]$ 内, $P(x)$ 都是线性函数。称为对插值点 $(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)$ 的分段线性插值函数

利用线性插值函数的构造方法，就可以得到分段线性插值函数在区间内的公式

$$P(x) = y_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i}$$

5.1.3 实验结果及分析

拉格朗日插值法:



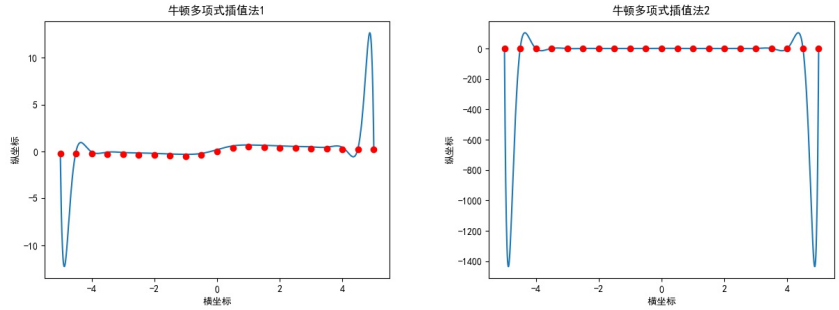
数据 1:

$\phi(-3.75) = -0.3856, \phi(0.25) = 0.2300, R(-3.75) = 0.1367, R(0.25) = 0.0053.$

数据 2:

$\phi(-3.75) = -12.6479, \phi(0.25) = 0.7044, R(-3.75) = 12.6507, R(0.25) = -0.3142.$

牛顿多项式插值法:



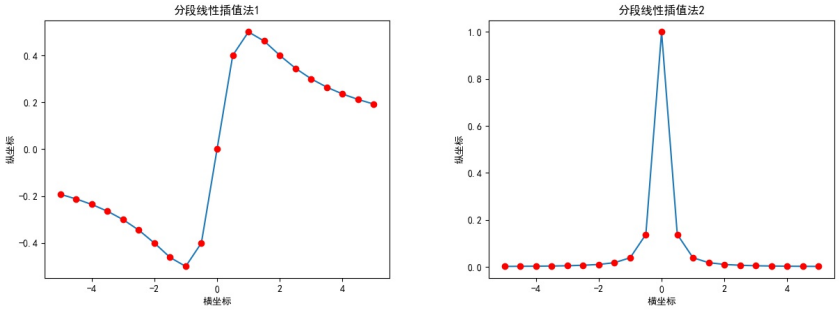
数据 1:

$\phi(-3.75) = -0.1933, \phi(0.25) = 0.4223, R(-3.75) = -0.05565, R(0.25) = -0.1870.$

数据 2:

$\phi(-3.75) = -12.6495, \phi(0.25) = 0.7028, R(-3.75) = 12.6523, R(0.25) = -0.3126.$

分段线性插值法:



数据 1:

$\phi(-3.75) = -0.2498, \phi(0.25) = 0.2000, R(-3.75) = 0.0007873, R(0.25) = 0.03529.$

数据 2:

$\phi(-3.75) = 0.0029, \phi(0.25) = 0.5690, R(-3.75) = -6.3623 * 10^{-5}, R(0.25) = -0.1787.$

分析: 结合图像，以及误差分析，显然分段线性插值方法更好。

5.1.4 附录：源代码

```
import numpy as np
from matplotlib import pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei'] # 中文字体设置-黑体
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题

x = np.linspace(-5, 5, 21)
y1 = [-0.1923, -0.2118, -0.2353, -0.2642, -0.3, -0.3448, -0.4000, -0.4615, -0.5000, -0.4000, 0,
     0.4000, 0.5000, 0.4615, 0.4000, 0.3448, 0.3000, 0.2642, 0.2353, 0.2118, 0.1923]
y2 = [0.0016, 0.002, 0.0025, 0.0033, 0.0044, 0.0064, 0.0099, 0.0175, 0.0385, 0.1379, 1.0000,
     0.1379, 0.0385, 0.0175, 0.0099, 0.0064, 0.0044, 0.0033, 0.0025, 0.0020, 0.0016]
find = [-3.75, 0.25]

data1 = []
for i in range(len(x)):
    data1.append([x[i], y1[i]])

data2 = []
for i in range(len(x)):
    data2.append([x[i], y2[i]])


# 拉格朗日
def Lg(data, testdata):
    predict = 0
    data_x = [data[i][0] for i in range(len(data))]
    data_y = [data[i][1] for i in range(len(data))]
    if testdata in data_x:
        return data_y[data_x.index(testdata)]
    for i in range(len(data_x)):
        af = 1
        for j in range(len(data_x)):
            if j != i:
                af *= (1.0 * (testdata - data_x[j]) / (data_x[i] - data_x[j]))
        predict += data_y[i] * af
    return predict


def plot_Lg(data, nums, cnt):
    data_x = [data[i][0] for i in range(len(data))]
    data_y = [data[i][1] for i in range(len(data))]

    X = np.linspace(min(data_x), max(data_x), nums)
    Y = [Lg(data, x) for x in X]

    plt.plot(X, Y, label='result')
    for i in range(len(data_x)):
```

```
48          plt.plot(data_x[i], data_y[i], 'ro', label='point')
49
50      plt.xlabel('横坐标')
51      plt.ylabel('纵坐标')
52      plt.title('拉格朗日插值法' + str(cnt))
53      plt.savefig('Lg' + str(cnt) + '.jpg')
54      plt.show()
55
56
57  # 分段线性插值
58  def DivideLine(data, testdata):
59      data_x = [data[i][0] for i in range(len(data))]
60      data_y = [data[i][1] for i in range(len(data))]
61
62      if testdata in data_x:
63          return data_y[data_x.index(testdata)]
64      else:
65          index = 0
66          for j in range(len(data_x)):
67              if data_x[j] < testdata < data_x[j + 1]:
68                  index = j
69                  break
70          predict = 1.0 * (testdata - data_x[index]) * \
71                    (data_y[index + 1] - data_y[index]) / (data_x[index + 1] - data_x[index]) +
                          data_y[index]
72          return predict
73
74
75  def plot_DL(data, nums, cnt):
76      data_x = [data[i][0] for i in range(len(data))]
77      data_y = [data[i][1] for i in range(len(data))]
78
79      X = np.linspace(min(data_x), max(data_x), nums)
80      Y = [DivideLine(data, x) for x in X]
81
82      # print('数据' + str(cnt) + ':')
83      # for i in find:
84      # print(i, Y[X.tolist().index(i)])
85
86      plt.plot(X, Y, label='result')
87
88      for i in range(len(data_x)):
89          plt.plot(data_x[i], data_y[i], 'ro', label='point')
90
91      plt.xlabel('横坐标')
92      plt.ylabel('纵坐标')
93      plt.title('分段线性插值法' + str(cnt))
94      plt.savefig('DL' + str(cnt) + '.jpg')
95      plt.show()
96
```

```
97
98  # 牛顿插值
99  def calF(data):
100     data_x = [data[i][0] for i in range(len(data))]
101     data_y = [data[i][1] for i in range(len(data))]
102
103     F = [1 for _ in range(len(data))]
104     FM = []
105     for i in range(len(data)):
106         FME = []
107         if i == 0:
108             FME = data_y
109         else:
110             for j in range(len(FM[len(FM) - 1]) - 1):
111                 delta = data_x[i + j] - data_x[j]
112                 value = 1.0 * (FM[len(FM) - 1][j + 1] - FM[len(FM) - 1][j]) / delta
113                 FME.append(value)
114         FM.append(FME)
115     F = [fme[0] for fme in FM]
116     # print(FM)
117     return F
118
119
120 def NT(data, testdata, F):
121     predict = 0
122     data_x = [data[i][0] for i in range(len(data))]
123     data_y = [data[i][1] for i in range(len(data))]
124     if testdata in data_x:
125         return data_y[data_x.index(testdata)]
126     else:
127         for i in range(len(data_x)):
128             Eq = 1
129             if i != 0:
130                 for j in range(i):
131                     Eq = Eq * (testdata - data_x[j])
132                 predict += (F[i] * Eq)
133     return predict
134
135
136 def plot_NT(data, nums, cnt):
137     data_x = [data[i][0] for i in range(len(data))]
138     data_y = [data[i][1] for i in range(len(data))]
139
140     X = np.linspace(min(data_x), max(data_x), nums)
141
142     F = calF(data)
143     Y = [NT(data, x, F) for x in X]
144
145     plt.plot(X, Y, label='result')
146     for i in range(len(data_x)):
```

```
147             plt.plot(data_x[i], data_y[i], 'ro', label='point')
148
149         plt.xlabel('横坐标')
150         plt.ylabel('纵坐标')
151         plt.title('牛顿多项式插值法' + str(cnt))
152         plt.savefig('NT' + str(cnt) + '.jpg')
153         plt.show()
154
155
156     def func1(x):
157         return x / (1 + x * x)
158
159
160     def func2(x):
161         return 1 / (1 + 25 * x * x)
162
163
164     plot_Lg(data1, 1000, 1)
165     plot_Lg(data2, 1000, 2)
166
167     plot_NT(data1, 1000, 1)
168     plot_NT(data2, 1000, 2)
169
170     plot_DL(data1, 1000, 1)
171     plot_DL(data2, 1000, 2)
172
173     print('拉格朗日插值法:')
174     print('数据1:')
175     print(Lg(data1, -3.75), Lg(data1, 0.25))
176     print('R(-3.75)=', func1(-3.75) - Lg(data1, -3.75))
177     print('R(0.25)=', func1(0.25) - Lg(data1, 0.25))
178
179     print('数据2:')
180     print(Lg(data2, -3.75), Lg(data2, 0.25))
181     print('R(-3.75)=', func2(-3.75) - Lg(data2, -3.75))
182     print('R(0.25)=', func2(0.25) - Lg(data2, 0.25))
183
184     print()
185
186     print('牛顿多项式插值法:')
187     F1 = calF(data1)
188     F2 = calF(data2)
189     print('数据1:')
190     print(NT(data1, -3.75, F1), NT(data1, 0.25, F1))
191     print('R(-3.75)=', func1(-3.75) - NT(data1, -3.75, F1))
192     print('R(0.25)=', func1(0.25) - NT(data1, 0.25, F1))
193
194     print('数据2:')
195     print(NT(data2, -3.75, F2), NT(data2, 0.25, F2))
196     print('R(-3.75)=', func2(-3.75) - NT(data2, -3.75, F2))
```

```
197 print('R(0.25)=', func2(0.25) - NT(data2, 0.25, F2))
198
199 print()
200
201 print('分段线性插值法:')
202 print('数据1:')
203 print(DivideLine(data1, -3.75), DivideLine(data1, 0.25))
204 print('R(-3.75)=', func1(-3.75) - DivideLine(data1, -3.75))
205 print('R(0.25)=', func1(0.25) - DivideLine(data1, 0.25))
206
207 print('数据2:')
208 print(DivideLine(data2, -3.75), DivideLine(data2, 0.25))
209 print('R(-3.75)=', func2(-3.75) - DivideLine(data2, -3.75))
210 print('R(0.25)=', func2(0.25) - DivideLine(data2, 0.25))
211
212 print()
213
214 for x in find:
215     print(x, func1(x), end=' ')
216 print()
217 for x in find:
218     print(x, func2(x), end=' ')
```

6 最小二乘拟合的实验与分析

6.1 综合实验：石油产量预测

6.1.1 实验题目

比较不同拟合函数的预测精度。

世界石油产量以每天百万桶计，如下表所示，求最佳最小二乘法数值估计：

年	桶/天 ($\times 10^6$)	年	桶/天 ($\times 10^6$)
1994	67.052	1999	72.063
1995	68.008	2000	74.669
1996	69.803	2001	74.487
1997	72.024	2002	74.065
1998	73.400	2003	76.777

1. 分别分析采用 (a) 直线，(b) 抛物线，(c) 立方曲线拟合 10 个数据点的结果，写出拟合后的具体函数表达式。

2. 进行如表 1 的残差分析。就残差分析结果而言，哪一种拟合最好的代表了这些数据？

3. 利用上面的每一种拟合来估计 2010 年的石油生产水平，讨论结果。

6.1.2 算法介绍

假设“逼近”规律的近似函数为 $y = f(x)$, 即有

$$y_i^* = f(x_i) \quad i = 1, 2, \ldots, m$$

它与观测值 y_i 之差 $\delta_i = y_i^* - y_i = f(x_i) - y_i$ 称为残差。

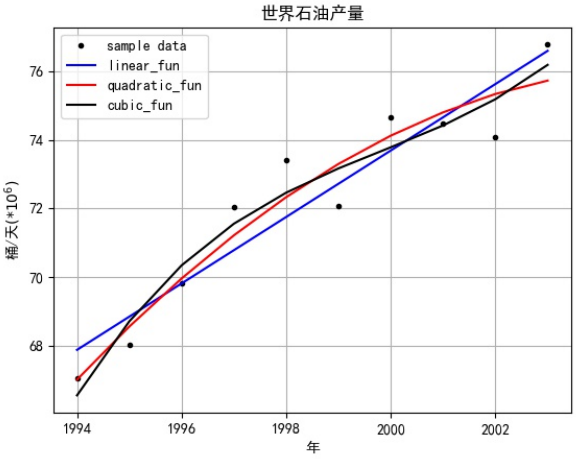
残差大大小可以作为衡量近似函数好坏的标准。按照使残差的平方和 $\sum \delta_i^2$ 最小的规则求得近似函数 $y = f(x)$ 的方法称为最佳平方逼近, 也称为曲线拟合的最小二乘法。

6.1.3 实验结果及分析

直线：$f_1(x) = 0.9693(x - 1994) + 67.8727$

抛物线: $f_2(x) = -0.0723(x - 1994)^2 + 1.6203(x - 1994) + 67.0048$

立方曲线：$f_3(x) = 0.0182(x - 1994)^3 - 0.3184(x - 1994)^2 + 2.4604(x - 1994) + 66.5455$



多项式类型	二范数的平方	残差的标准差	残差绝对值的均值
直线	9.528	0.976	0.811
抛物线	6.766	0.823	0.704
立方曲线	5.740	0.758	0.691

从数值上看，立方曲线的拟合效果最好。

估计 2010 年的石油生产水平，$f_1(2010) = 83.382, f_2(2010) = 74.413, f_3(2010) = 99.061$。从之前的拟合效果来看应当立方曲线的预测值较为准确，为 99.061。

6.1.4 附录：源代码

```
import math
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.optimize import leastsq
import numpy as np

plt.rcParams['font.sans-serif'] = ['SimHei'] # 中文字体设置-黑体
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题

x0 = np.arange(1994, 2004, 1, dtype='float')
x = np.asarray([i-1994 for i in x0])
y = np.array([67.052, 68.008, 69.803, 72.024, 73.400, 72.063, 74.6669, 74.487, 74.065, 76.777])
plt.figure()
plt.title('世界石油产量')
plt.xlabel('年')
plt.ylabel('桶/天$(*10^6)$')
plt.grid(True)
```

```
18  plt.plot(x0, y, 'k.')
19
20  param1 = [0, 0]
21
22
23  def linear_fun(s, x):
24      k, b = s
25      return k * x + b
26
27
28  param2 = [0, 0, 0]
29
30
31  def quadratic_fun(s, x):
32      k1, k2, b = s
33      return k1 * x ** 2 + k2 * x + b
34
35
36  param3 = [0, 0, 0, 0]
37
38
39  def cubic_fun(s, x):
40      k1, k2, k3, b = s
41      return k1 * x ** 3 + k2 * x ** 2 + k3 * x + b
42
43
44  def dist(a, fun, x, y):
45      return fun(a, x) - y
46
47
48  funs = [linear_fun, quadratic_fun, cubic_fun]
49  params = [param1, param2, param3]
50  colors = ['blue', 'red', 'black']
51  fun_name = ['linear_fun', 'quadratic_fun', 'cubic_fun']
52
53  for i, (func, param, color, name) in enumerate(zip(funs, params, colors, fun_name)):
54      var = leastsq(dist, param, args=(func, x, y))
55      print('[%s] 二范数: %.4f, abs(bias): %.4f, bias_std: %.4f' % (name,
56                                                                  ((y-func(var[0], x))**2).sum(),
57                                                                  (y-func(var[0], x)).std(),
58                                                                  (abs((y-func(var[0], x))).mean())))
59      print(var[0])
60      print(func(var[0], 2010-1994))
61      plt.plot(x0, func(var[0], x), color)
62  plt.legend(['sample data', 'linear_fun', 'quadratic_fun', 'cubic_fun'], loc='upper left')
63  plt.savefig('predict.jpg')
64  plt.show()
```

7　数值积分、微分和常微分方程的数值解法

7.1　数值积分

7.1.1　题目一

用梯形公式计算积分 $\int_0^1 x^2\ \mathrm{d}x$。

解：

$$\int_0^1 x^2\ \mathrm{d}x = \int_0^1 x\ \mathrm{d}x \approx \frac{1-0}{2}[f(1)+f(0)] = \frac{1}{2}$$

7.1.2　题目二

用辛普森公式计算积分 $\int_1^2 \sqrt{x}dx$。

解：

$$\begin{aligned}\int_1^2 \sqrt{x}\mathrm{d}x &\approx \frac{h}{3}\left[f(1)+4f\left(\frac{1+2}{2}\right)+f(2)\right]\\ &= \frac{1}{6}\left[1+2\sqrt{6}+\sqrt{2}\right]\\ &\approx 1.219\end{aligned}$$

7.1.3　题目三

分别用 4 次、6 次牛顿—柯特斯公式计算以下积分并与精确值作比较。

$$\int_1^2 \left(7x^6-4x^3+1\right)\mathrm{d}x$$

解:

①4 次:　4 阶对应系数 $\frac{7}{90}, \frac{32}{90}, \frac{12}{90}, \frac{32}{90}, \frac{7}{90}$, 对区间 [1, 2] 四等分

$$\begin{aligned}\int_1^2 \left(7x^6-4x^3+1\right)\mathrm{d}x = (2-1)\times(&\frac{7}{90}\cdot(7-4+1)+\frac{32}{90}\cdot(7\cdot 1.25^6-4\cdot 1.25^3+1)\\ &+\frac{12}{90}\cdot(7\cdot 1.5^6-4\cdot 1.5^3+1)\\ &+\frac{7}{90}\cdot(7\cdot 1.75^6-4\cdot 1.75^3+1)\\ &+\frac{7}{90}\cdot(7\cdot 2^6-4\cdot 2^3+1))\end{aligned}$$

$$\approx 0.3111 + 7.0721 + 8.9646 + 64.2214 + 32.4333$$
$$\approx 113.0026$$

②6 次:　6 阶对应系数 $\frac{41}{840}, \frac{216}{840}, \frac{27}{840}, \frac{272}{840}, \frac{27}{840}, \frac{216}{840}, \frac{41}{840}$, 对区间 [1, 2] 六等分

$$\begin{aligned}\int_1^2 \left(7x^6 - 4x^3 + 1\right) \mathrm{d}x = (2-1) \times (&\frac{41}{840} \cdot (7 \cdot 1^6 - 4 \cdot 1^3 + 1) + \frac{216}{840} \cdot (7 \cdot (\frac{7}{6})^6 - 4 \cdot (\frac{7}{6})^3 + 1) \\ &+ \frac{27}{840} \cdot (7 \cdot (\frac{4}{3})^6 - 4 \cdot (\frac{4}{3})^3 + 1) \\ &+ \frac{272}{840} \cdot (7 \cdot (\frac{3}{2})^6 - 4 \cdot (\frac{3}{2})^3 + 1) \\ &+ \frac{27}{840} \cdot (7 \cdot (\frac{5}{3})^6 - 4 \cdot (\frac{5}{3})^3 + 1) \\ &+ \frac{216}{840} \cdot (7 \cdot (\frac{11}{6})^6 - 4 \cdot (\frac{11}{6})^3 + 1) \\ &+ \frac{41}{840} \cdot (7 \cdot 2^6 - 4 \cdot 2^3 + 1)) \\ \approx 112.9997\end{aligned}$$

③精确解:

$$\int_1^2 \left(7x^6 - 4x^3 + 1\right) \mathrm{d}x = (x^7 - x^4 + x)|_1^2 = 113$$

经过对比，六次牛顿－柯特斯公式更加精确。

7.2　数值微分

7.2.1　题目一

用三点插值微分公式求各点的一阶和二阶导数，函数 $f(x)$ 由下表给出

x_i	1.0	1.1	1.2
$f(x_i)$	0.2500	0.2268	0.2066

解：

$$\begin{aligned} f^{'}(x_0) &\approx \frac{1}{2h}\cdot[-3f(x_0)+4f(x_1)-f(x_2)] \\ &= \frac{1}{0.2}\times(-3\times 0.25+4\times 0.2268-0.2066) \\ &= -0.247 \end{aligned}$$

$$\begin{aligned} f^{'}(x_1) \approx \frac{1}{2h}\cdot[-f(x_0)+f(x_2)] &= \frac{1}{0.2}\times(-0.25+0.2066) \\ &= -0.217 \end{aligned}$$

$$\begin{aligned} f^{'}(x_2) &\approx \frac{1}{2h}\cdot[f(x_0)-4f(x_1)+3f(x_2)] \\ &= \frac{1}{0.2}\times(0.25-4\times 0.2268+3\times 0.2066) \\ &= -0.187 \end{aligned}$$

$$\begin{aligned} f^{''}(x_0) &\approx \frac{1}{h^2}\cdot[f(x_0)-2f(x_1)+f(x_2)] \\ &= \frac{1}{0.1^2}\times(0.25-2\times 0.2268+0.2066) \\ &= 0.3 \end{aligned}$$

$$\begin{aligned} f^{''}(x_1) &\approx \frac{1}{h^2}\cdot[f(x_0)-2f(x_1)+f(x_2)] \\ &= \frac{1}{0.1^2}\times(0.25-2\times 0.2268+0.2066) \\ &= 0.3 \end{aligned}$$

$$\begin{aligned} f^{''}(x_2) &\approx \frac{1}{h^2}\cdot[f(x_0)-2f(x_1)+f(x_2)] \\ &= \frac{1}{0.1^2}\times(0.25-2\times 0.2268+0.2066) \\ &= 0.3 \end{aligned}$$

7.3 常微分方程的数值解法

7.3.1 题目一

分别用欧拉方法、改进欧拉法求解初值问题

$$y' = \frac{t-y}{2}, \quad y(0) = 1$$

在区间 [0, 3] 上的数值解，取步长为 $h = 0.5$，比较它们与准确解的误差（已知该初值问题的解析解为 $y(t) = 3e^{-t/2} - 2 + t$）。

解：

①欧拉方法:

$$\begin{cases} y(0) = 1 \\ y_{i+1} = y_i + h \cdot \frac{t-y}{2} \end{cases}$$

故:

表 7: 欧拉方法

t_k	y_k	y_{t_k}
0	1.0	1
0.5	0.75	0.836405
1	0.6875	0.819592
1.5	0.765625	0.917100
2	0.949219	1.103638
2.5	1.211914	1.359514
3	1.533936	1.669390

①改进欧拉方法:

$$\begin{cases} y(0) = 1 \\ y_{i+1} = \frac{4-h}{2h+4} y_i + \frac{h}{2h+4} (t_i + t_{i+1}) \end{cases}$$

故:

表 8: 改进欧拉方法

t_k	y_k	y_{t_k}
0	1.0	1
0.5	0.85	0.836402

1	0.845	0.819592
1.5	0.9415	0.917100
2	1.10905	1.103638
2.5	1.326335	1.359514
3	1.5784345	1.669390

8 总结与心得体会

8.1 总结

我们这门课学习了计算误差、非线性方程求根、线性方程组的解法、矩阵的特征值与特征向量、插值、拟合与最小二乘法、数值积分与数值微分。

8.2 心得体会

这门课程是一个十分重视算法和原理的学科，同时它能够将人的思维引入数学思考的模式，在处理问题的时候，可以合理适当的提出方案和假设。他的内容贴近实际，像数值分析，数值微分，求解线性方程组的解等，使数学理论更加有实际意义。

在课后，我们使用计算机编程实践这些算法，让我们对这些算法更加深刻理解，也培养了我们数值分析核心素养，用有穷取代无穷，变不可解为可解，变不可编程为可编程的方法，对数学问题近似求解，逐步求精。