数值计算实验报告

宋振华

2018 年 12 月 31 日

学院	泰山学堂
年级	2016 级
专业	计算机科学与技术
学号	201605301357
指导老师	刘保东

目录

1 第一章 **3**

1.1 基本概念 3

1.2 上机实验 6

2 第二章：求解方程的根 **9**

2.1 基本概念 9

2.2 上机实验 10

3 第三章：向量与矩阵范数 **15**

3.1 基本概念 15

3.2 上机实验 22

4 第四章：拉格朗日插值 **31**

4.1 基本概念 31

4.2 上机实验 31

5 第五章：最小二乘拟合 **34**

5.1 基本概念 34

5.2 上机实验 36

6 第六章数值微分 **38**

6.1 基本概念 38

6.2 上机实验 38

7 第七章：数值积分 **40**

7.1 基本概念 40

7.2 上机实验 42

8 常微分方程数值解 **44**

8.1 基本概念 44

8.2 上机实验 46

9 总结 **52**

简介

数值计算指有效使用数字计算机求数学问题近似解的方法与过程, 主要研究如何利用计算机更好的解决各种数学问题, 包括连续系统离散化和离散形方程的求解, 并考虑误差、收敛性和稳定性等问题.

从数学类型来分, 数值运算的研究领域包括数值逼近、数值微分和数值积分、数值代数、最优化方法、常微分方程数值解法、积分方程数值解法、偏微分方程数值解法、计算几何、计算概率统计等. 随着计算机的广泛应用和发展, 许多计算领域的问题, 如计算物理、计算力学、计算化学、计算经济学等都可归结为数值计算问题. 数值计算有以下 4 个特点:

1. 面向计算机, 要根据计算机的特点提供切实可行的有效算法;
2. 有可靠的理论分析, 包括算法的收敛性、稳定性、误差分析;
3. 要有好的计算复杂性, 包括时间复杂性、空间复杂性. 时间复杂性好是指节省时间, 空间复杂性好是指节省存储量.
4. 要有数值实验.

本学期实验对常用算法进行总结, 并编程实现.

1 第一章

1.1 基本概念

误差

1. 误差的来源、类型

   模型误差: 从实际问题转化为数学问题, 即建立数学模型时, 对被描述的实际问题进行了抽象和简化, 忽略了一些次要因素, 这样建立的数学模型只是对客观现象的一种近似, 这种数学模型与实际问题之间出现的误差称为模型误差.

   观测误差: 数学问题中一般会包含一些参量, 它们的值往往是由观测或实验得到的, 而观测或实验不可能绝对准确, 由此产生的误差称为观测误差.

截断误差: 通常指的是, 用一个基本表达式替换一个相当复杂的算术表达式时, 所引入的误差. 这个术语从用截断泰勒级数替换一个复杂表达式的技术衍生而来.

舍入误差: 计算机表示的示数受限于尾数的固定精度, 因此有时并不能确切地表示真实值, 这一类型的误差称为舍入误差.

2. 误差度量方法: 设 $\hat{p}$ 是 p 的近似值,

相对误差

$$R_p = \frac{|p - \hat{p}|}{p}, p \neq 0$$

绝对误差

$$E_p = |p - \hat{p}|$$

当 $|p|$ 远离 1 时 (大于或小于), 相对误差 R_p 比误差 E_p 能更好地表示近似值的精确程度.

迭代序列收敛性 迭代法是一种逐次逼近的方法, 首先给定一个粗糙的初值, 然后用同一个迭代公式, 反复校正这个初值, 直到满足预先给出的精度要求为止.

误差的收敛阶 设 $\lim\limits_{n\to\infty} x_n = x$, 有序列 $\{r_n\}_{n=1}^{\infty}$, 且 $\lim\limits_{n\to\infty} r_n = 0$. 如果存在常量 $K > 0$, 满足 $\frac{|x_n - x|}{|r_n|} \leq K$, n 足够大, 则称 $\{x_n\}_{n=1}^{\infty}$ 以收敛阶 $\boldsymbol{O}(r_n)$ 收敛于 x. 可以将其表示为 $x_n = x + \boldsymbol{O}(r_n)$, 或表示为 $x_n \to x$, 收敛阶为 $\boldsymbol{O}(r_n)$.

其他概念

1. 误差传播途径

加法运算: 考虑数 p 和 q(真实值) 的加法运算, 它们的近似值分别为 $\hat{p}$ 和 $\hat{q}$, 误差分别为 ϵ_p 和 ϵ_q. 从 $p = \hat{p} + \epsilon_p$ 和 $q = \hat{q} + \epsilon_q$ 开始, 它们的和为

$$p + q = (\hat{p} + \epsilon_p) + (\hat{q} + \epsilon_q) = (\hat{p} + \hat{q}) + (\epsilon_p + \epsilon_q)$$

因此对于加法运算, 和的误差是每个加数误差的和.

乘法运算: 乘积表达式如下所示:

$$pq = (\hat{p} + \epsilon_p)(q + \epsilon_q) = \hat{p}\hat{q} + \hat{p}\epsilon_q + \hat{q}\epsilon_p + \epsilon_p\epsilon_q$$

. 因此, 如果 $|\hat{p}| > 1, |\hat{q}| > 1$, 则原来的误差 ϵ_p 和 ϵ_q 会被放大成 $\hat{p}\epsilon_q$ 和 $\hat{q}\epsilon_p$.
假设 $p \neq 0, q \neq 0$, 则相对误差

$$R_{pq} = \frac{pq - \hat{p}\hat{q}}{pq} = \frac{\hat{p}\epsilon_p}{pq} + \frac{\hat{q}\epsilon_p}{pq} + \frac{\epsilon_p\epsilon_q}{pq}$$

进一步假设 $\hat{p}, \hat{q}$ 是 p, q 的好的近似, 则 $\frac{\hat{p}}{p} \approx 1, \frac{\hat{q}}{q} \approx 1, R_pR_q = \left(\frac{\epsilon_p}{p}\right)\left(\frac{\epsilon_q}{q}\right) \approx 0$, R_p, R_q 是 $\hat{p}, \hat{q}$ 的相对误差. 将它们替换到上式, 可以得到如下简化关系式:

$$R_{pq} = \frac{pq - \hat{p}\hat{q}}{pq} \approx \frac{\epsilon_q}{q} + \frac{\epsilon_p}{p} + 0 = R_q + R_p$$

这表明乘积 pq 的相对误差大致等于 $\hat{p}, \hat{q}$ 的相对误差之和.

初始误差通过一系列计算进行传播. 对于任何数值计算而言, 都要尽量减少初始误差, 因为初始条件下的小误差对最终结果产生的影响较小. 这样的算法称为稳定算法, 否则称为不稳定算法.

2. 误差增长: 设 ϵ 表示初始误差, $\epsilon(n)$ 表示第 n 步计算后的误差增长. 如果 $|\epsilon(n)| \approx n\epsilon$, 则称误差按线性增长. 如果 $|\epsilon(n)| \approx K^n\epsilon$, 则称误差按指数增长. 如果 $K > 1$, 当 $n \to \infty$ 时, 指数误差增长无界; 如果 $0 < K < 1$, 则当 $n \to \infty$ 时, 指数误差的增长趋于 0.

3. 误差的累积
当利用递推公式对各部分计算结果进行积分 (或累加) 时, 其误差也随之累加, 最后所得到误差总和称为累积误差.

4. 误差的改善

    选择数值稳定的算法;

    避免两个相近的数相减;

    避免大数吃小数现象;

    避免绝对值太小的数作除数;

    简化计算步骤, 减少运算次数.

1.2 上机实验

题目 根据习题 12 和习题 13 构造算法和程序，以便精确计算所有情况下的二次方程的根，包括 $|b| \approx \sqrt{b^2-4ac}$ 的情况.

习题 12 改进二次根公式. 设 $a \neq 0, b^2-4ac>0, ax^2+bx+c=0$. 通过如下二次根公式可解出方程的根:

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \quad x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a} \tag{1}$$

证明这些根可通过下列等价公式解出:

$$x_1 = \frac{-2c}{b+\sqrt{b^2-4ac}} \quad x_2 = \frac{-2c}{b-\sqrt{b^2-4ac}} \tag{2}$$

批注: 当 $|b| \approx \sqrt{b^2-4ac}$ 时, 必须小心处理, 以避免其值过小引起的巨量消失而带来的精度损失.

证明:

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} = \frac{\left(-b+\sqrt{b^2-4ac}\right)\times\left(b+\sqrt{b^2-4ac}\right)}{2a\times\left(b+\sqrt{b^2-4ac}\right)} = \frac{-2c}{b+\sqrt{b^2-4ac}}$$

$$x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a} = \frac{\left(-b-\sqrt{b^2-4ac}\right)\times\left(b-\sqrt{b^2-4ac}\right)}{2a\times\left(b-\sqrt{b^2-4ac}\right)} = \frac{-2c}{b-\sqrt{b^2-4ac}}$$

习题 13 利用习题 12 中求解 x_1, x_2 的公式, 计算下列二次方程的根.

1. $x^2 - 1000.001x + 1 = 0$
2. $x^2 - 10000.0001x + 1 = 0$
3. $x^2 - 100000.00001x + 1 = 0$
4. $x^2 - 1000000.000001x + 1 = 0$

求解:

1. $x_1 = 1000.0, x_2 = 10^{-3}$;
2. $x_1 = 10000.0, x_2 = 10^{-4}$;
3. $x_1 = 100000.0, x_2 = 10^{-5}$;

4. $x_1 = 1000000.0, x_2 = 10^{-6}$;

代码如下:

```
from math import sqrt
def calc(a,b,c):
    t = sqrt(b**2-4*a*c)
    if abs(b-t) < 1e-7:
        x1 = (-2*c)/(b+t)
        x2 = (-b-t)/(2*a)
    else:
        x1 = (-b+t)/(2*a)
        x2 = (-2*c)/(b-t)
    return x1, x2
```

2 参考例 1.25, 对系列 3 个差分方程计算出前 10 个数值近似值. 在每种情况下, 引入一个小的初始误差. 如果没有初始误差, 每个差分方程将生成序列 $\{1/2^n\}_{n=1}^{\infty}$.

1. $r_0 = 0.994, r_n = \frac{1}{2}r_{n-1}, n = 1, 2, \cdots$
2. $p_0 = 1, p_1 = 0.497, p_n = \frac{3}{2}p_{n-1} - \frac{1}{2}p_{n-2}, n = 2, 3, \cdots$
3. $q_0 = 1, q_1 = 0.497, q_n = \frac{5}{2}q_{n-1} - q_{n-2}, n = 2, 3, \cdots$

n	x_n	r_n	p_n	q_n
0	1	0.994	1	1
1	$\frac{1}{2}$	0.497	0.497	0.497
2	$\frac{1}{4}$	0.2485	0.24249999999999994	0.24550000000000005
3	$\frac{1}{8}$	0.12425	0.11975000000000008	0.10924999999999985
4	$\frac{1}{16}$	0.062125	0.05687500000000009	0.03062499999999968
5	$\frac{1}{32}$	0.0310625	0.0254375000000001	−0.03268750000000065
6	$\frac{1}{64}$	0.01553125	0.009718750000000102	−0.1123437500000013
7	$\frac{1}{128}$	0.007765625	0.001859375000000104	−0.2481718750000026
8	$\frac{1}{256}$	0.0038828125	−0.002070312499999895	−0.5080859375000052
9	$\frac{1}{512}$	0.00194140625	−0.004035156249999895	−1.0220429687500103
10	$\frac{1}{1024}$	0.000970703125	−0.005017578124999895	−2.0470214843750210

n	$x_n - r_n$	$x_n - p_n$	$x_n - q_n$
0	0.006000000000000005	0.0	0.0
1	0.003000000000000003	0.003000000000000003	0.003000000000000003
2	0.0015000000000000013	0.004499999999999949	0.007500000000000062
3	0.0007500000000000007	0.005249999999999921	0.015750000000000153
4	0.0003750000000000003	0.005624999999999908	0.03187500000000032
5	0.0001875000000000002	0.005812499999999901	0.06393750000000065
6	$9.3750000000000e-05$	0.005906249999999898	0.1279687500000013
7	$4.6875000000000e-05$	0.005953124999999896	0.2559843750000026
8	$2.3437500000000e-05$	0.005976562499999895	0.5119921875000052
9	$1.1718750000000e-05$	0.005988281249999895	1.0239960937500103
10	$5.8593750000000e-06$	0.005994140624999895	2.047998046875021

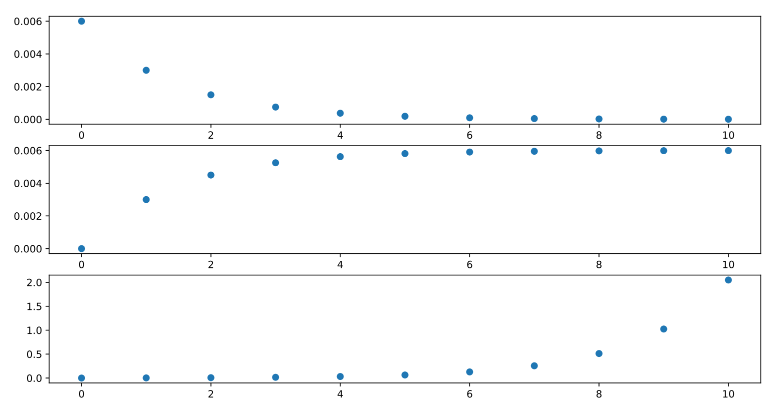

图 1: 上图为 $\{x_n - t_n\}$, 中图为 $\{x_n - p_n\}$, 下图为 $\{x_n - q_n\}$

代码:

```
r,p,q = [0.994],[1, 0.497],[1, 0.497]
for i in range(0, 10):
    r.append(r[-1] * 0.5)
for i in range(0,9):
    p.append(1.5*p[-1]-0.5*p[-2])
    q.append(2.5*q[-1]-q[-2])
```

2 第二章: 求解方程的根

2.1 基本概念

1. 方程的根: 方程的根是使方程左、右两边相等的未知数的取值.

2. 不动点: 指被这个函数映射到其自身一个点.

3. 迭代:

   重复执行一系列运算步骤, 从前面的量依次求出后面的量的过程. 此过程的每一次结果, 都是由对前一次所得结果施行相同的运算步骤得到. 例如利用迭代法求某一数学问题的解.

   对计算机特定程序中需要反复执行的子程序一组指令, 进行一次重复, 即重复执行程序中的循环, 直到满足某条件为止, 亦称为迭代.

迭代法收敛条件

设函数 $\phi(x)$ 在区间 $[a,b]$ 上满足条件:

1. $\forall x \in [a,b]$, 都有 $a \le \phi(x) \le b$,

2. $\exists L \in (0,1)$, st $\forall x, y \in [a,b]$, $|\phi(x) - \phi(y)| \le L|x-y|$.

则有如下结论:

1. $x = \phi(x)$ 在 $[a,b]$ 上有唯一的根 x^*;

2. $\forall x_0 \in [a,b]$, 迭代序列 $x_{n+1} = \phi(x)$ 收敛于 x^*;

3. $|x^* - x_n| \le \frac{L}{1-L}|x_n - x_{n-1}|$;

4. $|x^* - x_n| \le \frac{L^n}{1-L}|x_1 - x_0|$.

一般来说, 构造迭代函数, 使其在预先指定的较大区间上满足上述定理的条件比较困难, 但若取初值 x_0 充分接近根 x^*, 则收敛性的讨论可在根的附近进行.

局部收敛定理:

设 x^* 为方程 $x = \phi(x)$ 的根, 如果函数 $\phi(x)$ 在 x^* 的某一邻域 $O(x^*, \delta^*)$ 连续可微, 且 $\left|\phi'(x)\right| < 1$, 则 $\exists \delta \in (0, \delta^*]$, st $\forall x_0 \in [x^* - \delta, x^* + \delta]$, 迭代序列 $x_{n+1} = \phi(x_n)$ 收敛于 x^*.

不动点迭代法收敛速度:
由于 $\lim\limits_{n\to\infty}\frac{p_{n+1}-p}{p_n-p}=\lim\limits_{n\to\infty}g^{'}(\xi_n)=g^{'}(p)$, 即 $\lim\limits_{n\to\infty}\frac{|p_{n+1}-p|}{|p_n-p|}=\left|g^{'}(p)\right|$, 即不动点迭代法线性收敛.

牛顿法推导 :
假设 $f\in C^2[a,b]$, 并且 x^* 是 $f(x)=0$ 的一个解.

令 $\overline{x}\in[a,b]$ 是对 x_* 的一个近似, 使得 $f'(\overline{x})\neq 0$ 且 $|\overline{x}-x^*|$ 比较小.

考虑 $f(x)$ 在 $\overline{x}$ 处展开的一阶泰勒多项式 $f(x)=f(\overline{x})+(x-\overline{x})f'(\overline{x})+\frac{(x-\overline{x})^2}{2}f^{''}(\xi(x))$, 其中 $\xi(x)$ 在 x 和 $\overline{x}$ 之间.

因为 $f(x^*)=0$, 令 $x=x^*$, 此时有 $0=f(x^*)=f(\overline{x})+(x^*-\overline{x})f'(\overline{x})+\frac{(x^*-\overline{x})^2}{2}f^{''}(\xi(x))$.

忽略余项, 得到 $0=f(x^*)\approx f(\overline{x})+(x^*-\overline{x})f'(\overline{x})$

求得 $x^*\approx\overline{x}-\frac{f(\overline{x})}{f'(\overline{x})}$

因此定义迭代序列为: $x_n=x_{n-1}-\frac{f(x_{n-1})}{f'(x_{n-1})},\forall n\geq 1$.

割线法推导 用割线近似代替牛顿法中的切线. 得到公式 $x_{k+1}=x_k-f(x_k)\frac{x_k-x_{k-1}}{f(x_k)-f(x_{k-1})}$.

错位法推导 要在区间 $[p_0,p_1]$ 上寻找 $f(x)=0$ 的解, 其中 $f(p_0)f(p_1)<0$.

使用与切线法相同的方式, 找到近似点 p_2.

为确定使用哪一条线计算 p_3, 计算 $f(p_2)f(p_1)$ 和 $f(p_2)f(p_0)$.

如果 $f(p_2)f(p_1)<0$, 说明 p_1,p_2 中间包含了一个根, 那么取 $(p_1,f(p_1)),(p_2,f(p_2))$ 线段的斜率, 作为切线法中的斜率.

2.2 上机实验

实验 1 利用牛顿法求解方程

$$\frac{1}{2}+\frac{1}{4}x^2-x\sin x-\frac{1}{2}\cos 2x=0 \tag{3}$$

分别取 $x_0=\frac{\pi}{2},5\pi,10\pi$, 使得精度不超过 10^{-5}. 比较初值对计算结果的影响.

求解 代码:

```python
import math
fx = lambda x: 0.5 + 0.25 * x**2 - x * math.sin(x) - 0.5*math.cos(2*x)
dfx = lambda x: 0.5*x - x*math.cos(x) - math.sin(x) + math.sin(2*x)
def newton(x0):
    meps, max_iter = 1e-5, 200
    for i in range(0, max_iter):
        fx0, dfx0 = fx(x0), dfx(x0)
        if abs(fx0) < meps:
            break;
        x0 = x0 - fx0 / dfx0;
    return x0, fx0, i
```

结果			
初始值	迭代次数	最终 x	$f(x)$
$\pi/2$	5	1.892489	6.03e-06
5π	9	1.892790	4.89e-06
10π	不收敛	-	-

比较:
给定的函数, 在较大的范围内, 具有多个极值点. 当某一次迭代跳到极值点附近时, 将会获得较小的切线斜率, 从而会跳转到距离解较远的地方, 甚至会不收敛. 因此, 对于迭代法而言, 选择一个合适的初值, 非常重要.

展示图:

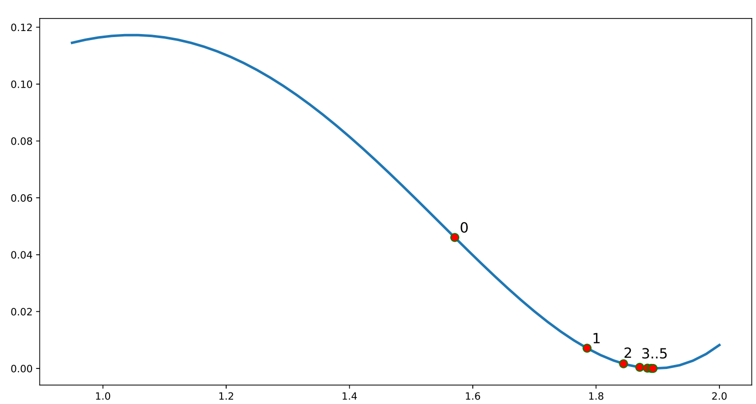

图 2: $x_0 = \pi/2$ 的情况, 由于初始值选取位置较好, 很容易收敛

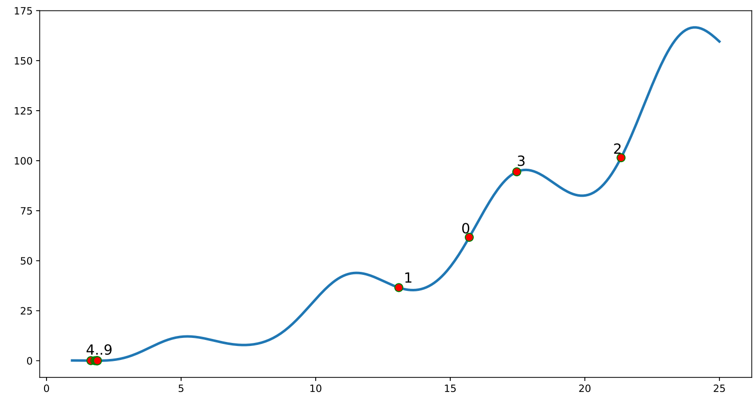

图 3: $x_0 = 5\pi$ 的情况, 迭代过程经过数个斜率较小点, 在反复增大、减小后, 收敛至解. 在这种情况下, 迭代有可能不收敛

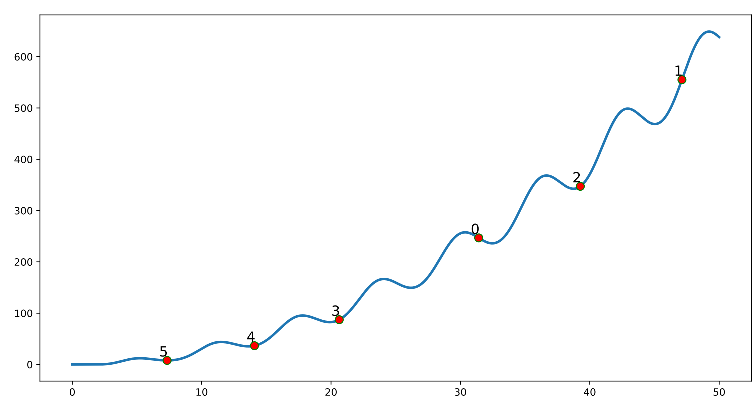

图 4: $x_0 = 10\pi$ 的情况. 迭代过程经历若干个极值点附近, 波动范围极大

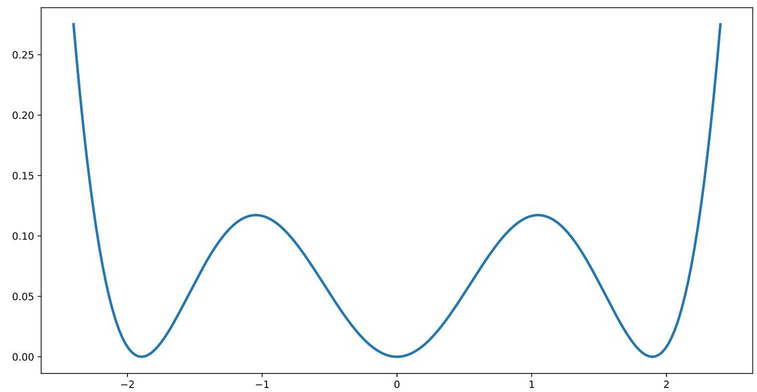

图 5: 函数图像

观察图像可以发现, $\lim_{x\to\infty} f(x) = +\infty$, 函数的零点在 $x=0$ 附近. 可以发现, $f(x)$ 的一个零点位于 $x=0$ 处, 一个零点位于 $(1.5,2)$, 一个零点位于 $(-2,-1.5)$. 根据初始值不同, 牛顿法迭代可能收敛到不同的零点, 也可能不收敛.

补充 证明方程 $2-3x-\sin(x)=0$ 在 $(0,1)$ 内有且仅有一个实根, 使用二分法求误差不大于 0.0005 的根, 以及需要的迭代次数.

证明:

令 $f(x) = 2-3x-\sin(x)$.

则 $f'(x) = -3-\cos(x) < 0$.

而 $f(0) = 2 > 0, f(1) = 1-\sin(1) < 0$

因此方程 $2-3x-\sin(x)=0$ 在 $(0,1)$ 内有且仅有一个实根.

代码:

```
def bisection(left, right):
    tol, N = 0.0005,100
    fl,cnt,mid,A = f(left),0,0,[]
    for cnt in range(0, N):
        mid = (left+right)/2
        fmid = f(mid)
        A.append(fmid)
        if(fl * fmid < 0):
            right = mid
```

```
        elif fl * fmid == 0:
            break
        else:
            fl, left = fmid, mid
        if right - left < tol:
            break
    return A, mid
```

运行结果:$x = 0.5053$, $f(x) = -0.00011$, 迭代次数 14.

实验 2 :

已知 $f(x) = 5x - e^x$ 在 $(0,1)$ 之间有一个实根, 试分别利用二分法、牛顿法、割线法、错位法设计相应的计算格式, 并编程求解 (精确到 4 位小数).

分析:

当 $x \in [0,1]$ 时, $f'(x) = 5 - e^x > 0$, 而 $f(0) = -1 < 0, f(1) = 5 - e > 0$. 因此 $f(x)$ 在 $[0,1]$ 上有且仅有一个零点. 二分法、牛顿法程序, 前面实验已经出现, 此处不再赘述.

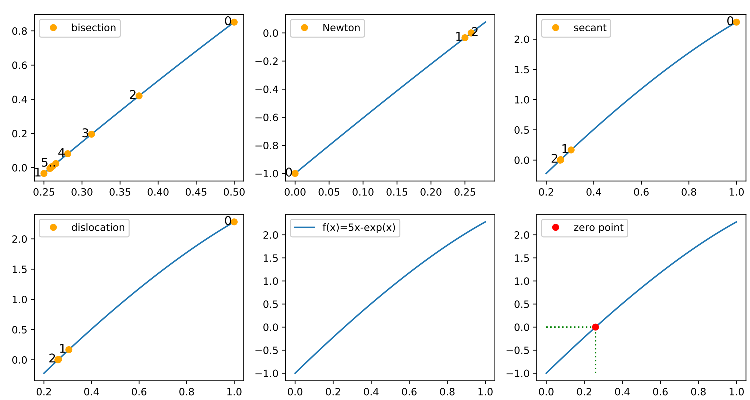

图 6: 函数图像

割线法程序:

```
def secant(left, right):
    tol, N = 5e-5, 100
    x0, x1, A = right, left, []
    for i in range(1, N):
        x = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
```

```
            A.append(x)
            if abs(x-x1) < tol:
                return f(x),A,i
            x0 ,x1 = x1, x
    return None, None, None
```

错位法程序:

```
def dislocation(left ,right):
    tol,N = 5e-5, 100
    p0,p1 = left,right
    q0,q1,A = f(p0),f(p1),[p1]
    for cnt in range(0,N):
        p = p1 - q1 * (p1 - p0) / (q1 - q0)
        A.append(p)
        q = f(p)
        if abs(p - p1) < tol:
            return q, A, cnt
        if q * q1 < 0:
            p0,q0 = p,q
        else:
            p1,q1 = p,q
    return None, None, None
```

	二分法	牛顿法	割线法	错位法
迭代次数	14	2	4	4
计算结果	0.259185	0.259171	0.259171	0.259171
函数值误差	$5.4e-05$	$-5.4e-05$	$1.8e-08$	$4.4e-07$

3 第三章: 向量与矩阵范数

3.1 基本概念

向量与矩阵范数 :

在泛函分析中, 它定义在赋范线性空间中, 并满足一定的条件, 即非负性、齐次性、三角不等式. 它常常被用来度量某个向量空间 (或矩阵) 中的每个向量的长度或大小用数学语言描述如下:

若 X 是数域上的线性空间, 泛函 $\|\cdot\|: X \to \mathbb{R}$ 满足:

1. 正定性: $\|x\| \geq 0, \|x\| = 0 \leftrightarrow x = 0$;
2. 正齐次性: $\|cx\| = |c|\,\|x\|$

3. 次可加性 (三角不等式): $\|x+y\| \le \|x\| + \|y\|$

那么, 称 $\|\cdot\|$ 为 X 上的一个范数.

向量范数 在 $\mathbb{C}^n$ 空间内, 最常用的范数是 p 范数. 若 $x = (x_1, x_2, \cdots, x_n)^T$, 那么 $\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$ 可以验证, p 范数确实满足范数的定义. 当 p 取 $1, 2, \infty$ 时, 分别是以下几种最简单的情形:

1. 1 范数: $\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$
2. 2 范数: $\|x\|_2 = \left(|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2\right)^{\frac{1}{2}}$
3. ∞ 范数: $\max\left(|x_1|, |x_2|, \cdots, |x_n|\right)$

其中 2 范数就是通常意义下的距离. 对于这些范数有以下不等式:

$$\|x\|_\infty \le \|x\|_2 \le \|x\|_1 \le \sqrt{n}\,\|x\|_2 \le n\,\|x\|_\infty$$

矩阵范数 矩阵范数还应满足矩阵运算的性质:

1. $\|\alpha A\| = \alpha \|A\|$
2. $\|AB\| \le \|A\| \|B\|$

矩阵距离: 对于 $n \times n$ 的矩阵 A, B, 距离为 $\|A - B\|$ 如果 $\|\cdot\|$ 是在 $\mathbb{R}^n$ 上的向量范数, 那么 $\|A\| = \max\limits_{\|x\|=1}$ 是一种矩阵范数.

1. ∞ 范数: $\|A\|_\infty = \max\limits_{\|x\|=1}$.

   对于 $A = (a_{ij})$, $\|A\|_\infty = \max\limits_{1 \le i \le n} \sum\limits_{j=1}^{n} |a_{ij}|$
2. 2 范数: 定义谱半径 $\rho(A) = \max|\lambda|$, 其中 λ 是 A 的特征值 (如果 $\lambda \in \mathbb{C}$ 且 $\lambda = \alpha + \beta i, |\lambda| = \sqrt{\alpha^2 + \beta^2}$). 则 $\|A\|_2 = \left[\rho\left(A^T A\right)\right]^{1/2}$

1. 特殊矩阵

   对称正定矩阵: 设 $A \in \mathbb{R}^{n \times n}$, 若 $A = A^T, \forall 0 \ne x \in \mathbb{R}^n$, st $x^T A x > 0$

   对角占优矩阵: 矩阵中每个主对角元素的模都大于与它同行的其他元素的模的总和.

2. 求解算法

   直接求解算法: LU 分解

   对称矩阵的 LL^T,LDL^T 分解

LU 分解形式 设 $\mathbf{A} = (a_{i,j})_{n\times n}$, 分解为 $\mathbf{A} = \mathbf{LU}$ 的形式, 其中 $\mathbf{L}$ 为对角元为 1 的下三角矩阵, $\mathbf{U}$ 为上三角矩阵. 即

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix}$$

按照矩阵乘法规则, 比较系数, 可得

$$\begin{cases} u_{1j} = a_{1j} & (j = 1, 2, \cdots, n) \\ l_{i1} = \dfrac{a_{i1}}{u_{11}} & (i = 2, 3, \cdots, n) \\ u_{ij} = a_{ij} - \displaystyle\sum_{k=1}^{i-1} l_{ik}u_{kj} & (i = 2, \cdots, n, j = i, \cdots, n) \\ l_{ij} = \left(a_{ij} - \displaystyle\sum_{k=1}^{j-1} l_{ik}u_{kj} \right) & (j = 1, 2, \cdots, n, i = j+1, \cdots, n) \end{cases}$$

迭代法推导

Jacobi 迭代法推导 假设方程组

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \qquad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_1 \end{cases}$$

的系数矩阵 $\mathbf{A}$ 非奇异, 不妨设 $a_{ii} \neq 0\,(i=1,2,\cdots,n)$. 将方程组变形为:

$$\begin{cases} x_1 = \dfrac{1}{a_{11}}\left(-a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n + b_1\right) \\ x_2 = \dfrac{1}{a_{22}}\left(-a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n + b_2\right) \\ \qquad\vdots \\ x_n = \dfrac{1}{a_{nn}}\left(-a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1} + b_n\right) \end{cases}$$

建立迭代公式:

$$\begin{cases} x_1^{(k+1)} = \dfrac{1}{a_{11}}\left(-a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \cdots - a_{1n}x_n^{(k)} + b_1\right) \\ x_2^{(k+1)} = \dfrac{1}{a_{22}}\left(-a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \cdots - a_{2n}x_n^{(k)} + b_2\right) \\ \qquad\vdots \\ x_n^{(k+1)} = \dfrac{1}{a_{nn}}\left(-a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} - \cdots - a_{n,n-1}x_{n-1}^{(k)} + b_n\right) \end{cases}$$

选定初始向量 $\mathbf{x}^{(0)}$ 后, 反复迭代可以得到向量序列 $\left\{\mathbf{x}^{(k)}\right\}$ 迭代公式为:

$$\begin{cases} \mathbf{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0),\cdots,x_n^{(0)}}\right)^T \\ x_i^{(k+1)} = \dfrac{1}{a_{ii}}\left(b_i - \displaystyle\sum_{j=1,j\neq i}^{n} a_{ij}x_j^{(k)}\right) \end{cases}$$

Jacobi 迭代法也可以写为向量递推的形式

$$\mathbf{x}^{(k)} = \mathbf{T}\mathbf{x}^{(k)} + \mathbf{c}$$

设 $\mathbf{D}$ 是对角元与 $\mathbf{A}$ 相同的对角阵; $-\mathbf{L}$ 是严格下三角矩阵, 下三角元素与 $\mathbf{A}$ 相同; $-\mathbf{U}$ 是严格上三角矩阵, 上三角元素与 $\mathbf{A}$ 相同. 因此 $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$.

对于方程 $\mathbf{A}\mathbf{x} = \mathbf{b}$, 或 $(\mathbf{D} - \mathbf{L} - \mathbf{U})\,\mathbf{x} = \mathbf{b}$,

可以变形为

$$\mathbf{D}\mathbf{x} = (\mathbf{L} + \mathbf{U})\,\mathbf{x} + \mathbf{b}$$

也就是

$$\mathbf{x} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\,\mathbf{x} + \mathbf{D}^{-1}\mathbf{b}$$

写成递推式的形式:

$$\mathbf{x}^{(k)} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\,\mathbf{x}^{(k-1)} + \mathbf{D}^{-1}\mathbf{b}$$

Gauss-Seidel 迭代法推导 ：如果把 Jacobi 迭代公式改成以下形式

$$\begin{cases} x_1^{(k+1)} = \dfrac{1}{a_{11}}\left(-a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \cdots - a_{1n}x_n^{(k)} + b_1\right) \\ x_2^{(k+1)} = \dfrac{1}{a_{22}}\left(-a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \cdots - a_{2n}x_n^{(k)} + b_2\right) \\ \qquad\qquad \vdots \\ x_n^{(k+1)} = \dfrac{1}{a_{nn}}\left(-a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - \cdots - a_{n,n-1}x_{n-1}^{(k+1)} + b_n\right) \end{cases}$$

选取初始向量 $\mathbf{x}^{(0)}$, 用迭代公式:

$$\begin{cases} \mathbf{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0),\cdots,x_n^{(0)}}\right)^T \\ x_i^{(k+1)} = \dfrac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) \end{cases}$$

Gauss-Seidel 迭代法也可以写为向量递推的形式:

$$\mathbf{x}^{(k)} = \mathbf{T}_g\mathbf{x}^{(k)} + \mathbf{c}_g$$

或

$$(\mathbf{D} - \mathbf{L})\,\mathbf{x}^{(k)} = \mathbf{U}\mathbf{x}^{(k-1)} + \mathbf{b}$$

即

$$\mathbf{x}^{(k)} = (\mathbf{D} - \mathbf{L})^{-1}\,\mathbf{U}\mathbf{x}^{k-1} + (\mathbf{D} - \mathbf{L})^{-1}\,\mathbf{b}$$

特征值求解

幂法 假定 $n \times n$ 的矩阵 $\mathbf{A}$ 有 n 个特征值 $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$, 对应线性无关的特征向量 $\left\{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \cdots, \mathbf{v}^{(n)}\right\}$,

$\forall \mathbf{x} \in \mathbb{R}^n, \exists \beta_1, \beta_2, \cdots, \beta_n, st\ \mathbf{x} = \beta_1\mathbf{v}^{(1)} + \beta_2\mathbf{v}^{(2)} + \cdots + \beta_n\mathbf{v}^{(n)} = \sum_{j=1}^{n}\beta_j\mathbf{v}^{(j)}$

即

$$\begin{aligned}
\mathbf{A}\mathbf{x} &= \sum_{j=1}^{n} \beta_j \mathbf{A}\mathbf{v}^{(j)} = \sum_{j=1}^{n} \beta_j \lambda_j \mathbf{v}^{(j)} \\
\mathbf{A}^2\mathbf{x} &= \sum_{j=1}^{n} \beta_j \mathbf{A}\mathbf{v}^{(j)} = \sum_{j=1}^{n} \beta_j \lambda_j^2 \mathbf{v}^{(j)} \\
\mathbf{A}^2\mathbf{x} &= \sum_{j=1}^{n} \beta_j \mathbf{A}\mathbf{v}^{(j)} = \sum_{j=1}^{n} \beta_j \lambda_j^2 \mathbf{v}^{(j)} \\
&\vdots \\
\mathbf{A}^k\mathbf{x} &= \sum_{j=1}^{n} \beta_j \mathbf{A}\mathbf{v}^{(j)} = \sum_{j=1}^{n} \beta_j \lambda_j^k \mathbf{v}^{(j)} \\
&= \lambda_1^k \left(\beta_1 \mathbf{v}^{(1)} + \sum_{j=2}^{n} \beta_j \left(\frac{\lambda_j}{\lambda_1} \right)^k \mathbf{v}^{(j)} \right)
\end{aligned}$$

由于 $\forall j \in \{2, \cdots, n\}, |\lambda_1| > |\lambda_j|$, 因此 $\lim\limits_{k\to\infty} \left(\frac{\lambda_j}{\lambda_1}\right)^k = 0$. 即

$$\lim_{k\to\infty} \mathbf{A}^k\mathbf{x} = \lim_{k\to\infty} \lambda_1^k \beta_1 \mathbf{v}^{(1)}$$

用 $x_i^{(k)}$ 表示 $\mathbf{x}^{(k)}$ 的第 i 个分量. 由于

$$\frac{x_i^{(k+1)}}{x_i^{(k)}} \approx \frac{\lambda_1^{k+1}\beta_1 \mathbf{v}_i^{(1)}}{\lambda_1^k \beta_1 \mathbf{v}_i^{(k)}} = \lambda_1$$

需要注意, 当 $|\lambda_1| > 1$ 时, $\mathbf{x}^{(k)}$ 的各分量趋于无穷, 当 $|\lambda_1| < 1$ 时, $\mathbf{x}^{(k)}$ 的各分量趋于 0. 为了克服这一缺点, 需要将迭代向量规范化. 采用如下方式进行迭代:

$$\begin{cases}
\mathbf{x}^{(0)} = \mathbf{y}^{(0)} \neq \mathbf{0} \\
\mathbf{x}^{(k)} = \mathbf{A}\mathbf{y}^{(k-1)} \\
\mathbf{y}^{(k)} = \dfrac{\mathbf{x}^{(k)}}{||\mathbf{x}^{(k)}||_\infty}
\end{cases}$$

$$\lim_{k\to\infty} \mathbf{y}^{(k)} = \frac{v^{(1)}}{||\mathbf{v}^{(1)}||}, \lim_{k\to\infty} ||\mathbf{x}||_\infty = \lambda_1$$

.

易知最终的 $\mathbf{x}$ 为所求特征向量.

对称幂法 假设 $\mathbf{A}$ 是对称矩阵, $\mathbf{A}$ 有 n 个实数特征向量 $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, 对应 n 个标准正交特征向量 $\left\{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \cdots, \mathbf{v}^{(n)}\right\}$.

$\forall \mathbf{x}_0 \in \mathbb{R}^n, \mathbf{x}_0 = \beta_1 \mathbf{v}^{(1)} + \beta_2 \mathbf{v}^{(2)} + \cdots + \mathbf{v}^{(n)}$.

对于 $\mathbf{x}_k = \mathbf{A}^k \mathbf{x}_0$,$\mathbf{x}_k = \beta_1 \lambda_1^k \mathbf{v}^{(1)} + \beta_2 \lambda_2^k \mathbf{v}^{(2)} + \cdots + \beta_n \lambda_n^k \mathbf{v}^{(n)}$. 因为 n 个特征向量标准正交, 因此

$$\mathbf{x}_k^T \mathbf{x}_k = \sum_{j=1}^{n} \beta_j^2 \lambda_j^{2k} = \beta_1^2 \lambda_1^{2k} \left[1 + \sum_{j=2}^{n} \left(\frac{\beta_j}{\beta_1}\right)^2 \left(\frac{\lambda_j}{\lambda_n}\right)^{2k}\right]$$

且

$$\mathbf{x}_k^T \mathbf{A} \mathbf{x}_k = \sum_{j=1}^{n} \beta_j^2 \lambda_j^{2k+1} = \beta_1^2 \lambda_1^{2k+1} \left[1 + \sum_{j=2}^{n} \left(\frac{\beta_j}{\beta_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k+1}\right]$$

于是

$$\lim_{k\to\infty} \frac{\mathbf{x}_k^T \mathbf{A} \mathbf{x}_k}{\mathbf{x}_k^T \mathbf{x}_k} = \lambda_1, \lim_{k\to\infty} \frac{\mathbf{x}_k}{||\mathbf{x}_k|_2|} = \frac{\mathbf{v}^{(1)}}{||\mathbf{v}^{(1)}||_2}$$

.

反幂法 设 $\mathbf{A}$ 为 $n \times n$ 阶非奇异矩阵, λ 和 $\mathbf{v}$ 为对应的特征向量, 即 $\mathbf{A}\mathbf{u} = \lambda \mathbf{v}$.

由于 $\mathbf{A}^{-1}\mathbf{v} = \frac{1}{\lambda}\mathbf{v}$. 如果 $\mathbf{A}$ 的特征值的顺序为 $|\lambda_1\ | \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$, 则 $\mathbf{A}^{-1}$ 的特征值 $\left|\frac{1}{\lambda_n}\right| \geq \left|\frac{1}{\lambda_{n-1}}\right| \geq \cdots \geq \left|\frac{1}{\lambda_1}\right|$.

因此, 若对矩阵 $\mathbf{A}^{-1}$ 用幂法, 即可计算出 $\mathbf{A}^{-1}$ 的最大特征值 $\frac{1}{\lambda_n}$, 从而求得 $\mathbf{A}$ 按模最小的特征值 λ_n.

因为 $\mathbf{A}^{-1}$ 的计算比较麻烦, 因此在实际运算时, 以求解方程组 $\mathbf{A}\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$ 替代.

在本题中, 需要求解最大特征值. 根据圆盘定理, 矩阵 $\mathbf{A}$ 的特征值有上界 Ma. 设 λ 为 $\mathbf{A}$ 的特征值, $k \in \mathbb{R}$. 则 $(\mathbf{A} - k\mathbf{I})\,\mathbf{x} = (\lambda - k)\,\mathbf{x}$. 此时令 $k = Ma + 2$, 则 $(\mathbf{A} - (Ma+2)\,\mathbf{I})$ 的特征值均小于 0. 设用反幂法求出 $(\mathbf{A} - (Ma+2)\,\mathbf{I})^{-1}$ 的绝对值最小特征值为 $|\lambda^*|$, 则 $\mathbf{A}$ 的最大特征值为 $Ma + 2 - \frac{1}{|\lambda^*|}$.

注: 圆盘定理: 设 $\mathbf{A}$ 是一个 $n \times n$ 的矩阵, $\mathbb{R}_i$ 表示以 a_{ii} 为圆心, $\sum_{j=1,j\neq i}^{n} |a_{ij}|$ 的圆. 令

$$\mathbb{R}_i = \left\{ z \in \mathbb{C} \middle| \, |z - a_{ii}| \leq \sum_{j=1,j\neq i}^{n} |a_{ij}| \right\}$$

$\mathbf{A}$ 的特征值包含在 $\bigcup_{i=1}^{n} \mathbb{R}_i$ 中.

对幂法进行改造, 反幂法计算的主要步骤如下:

1. 对 $\mathbf{A}$ 进行三角形分解 $\mathbf{A} = \mathbf{LU}$

2. 做如下迭代:

$$\begin{cases} Let\ \mathbf{x}^{(0)} = \mathbf{y}^{(0)} \neq 0 \\ Solve\ :\mathbf{A}\mathbf{x}^{(k)} = \mathbf{y}^{(k-1)} \\ Then\ :\mathbf{y}^{(k)} = \dfrac{\mathbf{x}^{(k)}}{||\mathbf{x}^{(k)}||_\infty} \end{cases}$$

在反幂法中, 最终得到的 $\mathbf{x}$ 即为所求特征向量.

3.2 上机实验

1 求解线性方程组

$$\begin{cases} 4x - y + z = 7 \\ 4x - 8y + z = -21 \\ -2x + y + 5z = 15 \end{cases} \tag{4}$$

1. 试用 LU 分解求解此方程组;

2. 分别用 Jacobi, Gauss-Seidel 方法求解此方程组

LU 分解代码:

```python
import numpy as np
def my_LU(B):
    n = len(B)
    L,U = np.zeros(shape=(n, n)),np.array(B)
    for k in range(n - 1):
        gauss_vector = U[:, k]
        gauss_vector[k + 1:] = gauss_vector[k + 1:] / gauss_vector[k]
        gauss_vector[0:k + 1] = np.zeros(k + 1)
        L[:, k] = gauss_vector
        L[k][k] = 1.0
        for l in range(k + 1, n):
            B[l, :] = B[l, :] - gauss_vector[l] * B[k, :]
        U = np.array(B)
    L[k + 1][k + 1] = 1.0
    return L, U

```

令 $y = Ux$, 则 $LUx = Ly = b$. 求解 $Ly = b$:

```
def Lyb(L,b):
    n,y = len(L), np.zeros(shape=(len(L)))
    for i in range(0,n):
        t = b[i]
        for j in range(0,i):
            t -= L[i,j] * y[j]
        y[i] = t / L[i,i]
    return y

```

求解 $Ux = y$:

```
def Uxy(U,y):
    n, x = len(U), np.zeros(shape=(len(U)))
    for i in range(0,n):
        t = y[n-1-i]
        for j in range(0,i):
            t -= U[n-1-i,n-1-j] * x[n-1-j]
        x[n-1-i] = t / U[n-1-i,n-1-i]
    return x

```

求解结果: $(x_1, x_2, x_3)^T = (2, 4, 3)^T$.

Jacobi 迭代求解:

```
import numpy as np
def jacobi(A,B):
    x0,x = np.zeros(shape=(len(B))),np.zeros(shape=(len(B)))
    meps, miter = 1e-6, 200
    for times in range(0,miter):
        for i in range(len(B)):
            temp = 0
            for j in range(len(B)):
                if i != j:
                    temp += x0[j] * A[i][j]
            x[i] = (B[i] - temp) / A[i][i]
        if max(abs(x - x0)) < meps:
            break
        x0 = x.copy()
    return x,times
```

求解结果: $(x_1, x_2, x_3)^T = (1.99999985, 3.9999997, 2.99999978)^T$, 迭代次数 14

Gauss-Seidel 迭代

```python
def gauss(A,b):
    x,lx = np.zeros(shape=(len(b))), np.zeros(shape=(len(b)))
    meps, miter = 1e-6, 200
    for count in range(0, miter):
        for i in range(len(x)):
            nxi = b[i]
            for j in range(len(A[i])):
                if j != i:
                    nxi = nxi + (-A[i][j]) * x[j]
            nxi = nxi / A[i][i]
            x[i] = nxi
        if max(abs(x - lx)) < meps:
            break
        lx = np.array(x)
    return x, count
```

求解结果: $(x_1, x_2, x_3)^T = (1.99999996, 3.99999997, 2.99999999)^T$, 迭代次数 8. 注意到, Gauss-Seidel 此时收敛速率较 Jacobi 快.

2 3.6.5 算法与程序 (P118), 3,4 3. 设有如下三角线性方程组, 而且系数矩阵具有严格对角优势:

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 \\ 0 & 0 & 0 & 0 & a_6 & b_6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{pmatrix}$$

1. 设计一个算法来求解上述方程组. 算法必须有效地利用系数矩阵的稀疏性.

2. 根据上面算法, 构造一个程序, 并求解下列三角线性方程组.

$$\begin{array}{ccccccc||ccccccc}
4m_1 & + & m_2 & & & = & 3 & 4m_1 & + & m_2 & & & = & 1 \\
m_1 & + & 4m_2 & + & m_3 & = & 3 & m_1 & + & 4m_2 & + & m_3 & = & 2 \\
m_2 & + & 4m_3 & + & m_4 & = & 3 & m_2 & + & 4m_3 & + & m_4 & = & 1 \\
m_3 & + & 4m_4 & + & m_5 & = & 3 & m_3 & + & 4m_4 & + & m_5 & = & 2 \\
\vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\
m_{48} & + & 4m_{49} & + & m_{50} & = & 3 & m_{48} & + & 4m_{49} & + & m_{50} & = & 1 \\
 & & 4m_{49} & + & 4m_{50} & = & 3 & & & 4m_{49} & + & 4m_{50} & = & 2
\end{array}$$

求解三对角矩阵:

1. 将系数矩阵变为一个上三角矩阵. 第一行:

$$b_1x_1 + c_1x_2 = r_1 \leftrightarrow x_1 + \frac{c_1}{b_1}x_2 = \frac{r_1}{b_1}$$

$$x_1 + \gamma_1 x_2 = \rho_1, \gamma_1 = \frac{c_1}{b_1}, \rho_1 = \frac{r_1}{b_1}$$

第二行:

$$(b_2 - a_2\gamma_1)\, x_2 + c_2x_3 = r_2 - a_2\rho_1$$

同理可推: 第三行:

$$x_3 + \gamma_3x_4 = \rho_3, \gamma_3 = \frac{c_3}{b_3 - a_3\gamma_2}, \rho_3 = \frac{r_3 - a_3\rho_2}{b_3 - a_3\gamma_2}$$

第四行:

$$x_4 + \gamma_4x_5 = \rho_4, \gamma_4 = \frac{c_4}{b_4 - a_4\gamma_3}, \rho_4 = \frac{r_4 - a_4\rho_3}{b_4 - a_4\gamma_3}$$

以此类推, 可以得到一般性迭代公式:

$$c_i^{'} = \begin{cases} \dfrac{c_i}{b_i}, i = 1 \\ \dfrac{c_i}{b_i - c_{i-1}^{'}a_i}, i = 2, 3, \cdots, n-1 \end{cases}$$

$$r_i^{'} = \begin{cases} \dfrac{r_i}{b_i}, i = 1 \\ \dfrac{r_i - r_{i-1}'a_i}{b_i - c_{i-1}^{'}a_i} \end{cases}$$

$$x_i = d_i^{'} - c_i^{'}x_{i+1}, i = n-1, n-2, \cdots, 1$$

代码:

```
import numpy as np
def tridig(a,b,c,r):
    n = len(b)
    c1,r1,x = np.array(c), np.array(r), np.array(r)
    c1[0] = c[0] / b[0]
    for i in range(1, n-1):
        c1[i] = c[i] / (b[i] - c1[i-1] * a[i])
    r1[0] = r[0] / b[0]
    for i in range(1, n):
        r1[i] = (r[i] - r1[i-1]*a[i]) / (b[i] - c1[i-1]*a[i])
    x[n-1] = r1[n-1]
    for i in range(n-2, -1, -1):
        x[i] = r1[i] - c1[i] * x[i+1]
    return x

```

求解结果-1:					
$m_1 - m_5$	0.633974	0.464101	0.509618	0.497422	0.500690
$m_6 - m_{10}$	0.499814	0.500049	0.499986	0.500000	0.499999
$m_{11} - m_{15}$	0.500000	0.499999	0.500000	0.499999	0.500000
$m_{16} - m_{20}$	0.499999	0.500000	0.499999	0.500000	0.499999
$m_{21} - m_{25}$	0.500000	0.499999	0.500000	0.49999	0.500000
$m_{26} - m_{30}$	0.500000	0.499999	0.500000	0.499999	0.500000
$m_{31} - m_{35}$	0.499999	0.500000	0.499999	0.500000	0.499999
$m_{36} - m_{40}$	0.500000	0.499999	0.500000	0.499999	0.500000
$m_{41} - m_{45}$	0.499999	0.500003	0.499986	0.500049	0.499814
$m_{46} - m_{50}$	0.500690	0.497422	0.509618	0.464101	0.633974
求解结果-2:					
$m_1 - m_5$	0.133975	0.464102	0.009619	0.497423	0.000691
$m_6 - m_{10}$	0.499815	0.000050	0.499987	0.000004	0.499999
$m_{11} - m_{15}$	0.000000	0.500000	0.000000	0.500000	0.000000
$m_{16} - m_{20}$	0.500000	0.000000	0.500000	0.000000	0.500000
$m_{21} - m_{25}$	0.000000	0.500000	0.000000	0.500000	0.000000
$m_{26} - m_{30}$	0.500000	0.000000	0.500000	0.000000	0.500000
$m_{31} - m_{35}$	0.000000	0.500000	0.000000	0.500000	0.000000
$m_{36} - m_{40}$	0.500000	0.000000	0.500000	0.000000	0.500000
$m_{41} - m_{45}$	0.000000	0.500000	0.000000	0.500000	0.000000
$m_{46} - m_{30}$	0.500000	0.000000	0.500000	0.000000	0.500000

3 利用 Gauss-Seidel 迭代法求下列带状方程:

$$
\begin{array}{ccccccccccccc}
12x_1 & - & 2x_2 & + & x_3 & & & & & & & = & 5 \\
-2x_1 & + & 12x_2 & - & 2x_3 & + & x_4 & & & & & = & 5 \\
x_1 & - & 2x_2 & + & 12x_3 & - & 2x_4 & + & x_5 & & & = & 5 \\
x_2 & - & 2x_3 & + & 12x_4 & - & 2x_5 & + & x_6 & & & = & 5 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & & & \vdots \\
x_{46} & - & 2x_{47} & + & 12x_{48} & - & 2x_{49} & + & x_{50} & & & = & 5 \\
 & & x_{47} & - & 2x_{48} & + & 12x_{49} & - & 2x_{50} & & & = & 5 \\
 & - & & & 12x_{48} & - & 2x_{49} & + & 12x_{50} & & & = & 5
\end{array}
$$

Gauss-Seidel 迭代法前面已经说明, 此处不再赘述.
结果:

$x_1 - x_5$	0.463796	0.537285	0.509023	0.498222	0.498942
$x_6 - x_{10}$	0.499985	0.500089	0.500015	0.499995	0.499998
$x_{11} - x_{15}$	0.500000	0.500000	0.500000	0.500000	0.500000
$x_{16} - x_{20}$	0.500000	0.500000	0.500000	0.500000	0.500000
$x_{21} - x_{25}$	0.500000	0.500000	0.500000	0.500000	0.500000
$x_{26} - x_{30}$	0.500000	0.500000	0.500000	0.500000	0.500000
$x_{31} - x_{35}$	0.500000	0.500000	0.500000	0.500000	0.500000
$x_{36} - x_{40}$	0.500000	0.500000	0.500000	0.500000	0.500000
$x_{41} - x_{45}$	0.499998	0.499995	0.500015	0.500089	0.499985
$x_{46} - x_{50}$	0.498942	0.498222	0.509023	0.537285	0.463796

4 已知矩阵

$$\mathbf{A} = \begin{pmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{pmatrix}$$

是一个对称矩阵, 且其特征值为 $\lambda_1 = 6, \lambda_2 = 3, \lambda_3 = 1$. 分别利用幂法则, 对称幂法, 反幂法求其最大特征值和特征向量.

注意: 可取初始向量 $x^{(0)} = (1, 1, 1)^T$

幂法:

```python
import numpy as np
def powermethod(A,x0):
    N, tol = 100, 1e-7
    y0, mx0,n = x0, np.max(x0), len(A)
    for i in range(0, N):
        x = A*y0
        mx = np.max(x)
        y0 = x / mx
        if abs(mx - mx0) < tol:
            return mx, x, i
        x0, mx0 = x, mx
    return None, None
```

Answer: Max Eigen Value: 6.000000, Iteration time 27.
Eigen Vector: $\mathbf{x} = (6.0000, -6.0000, 6.0000)^T$

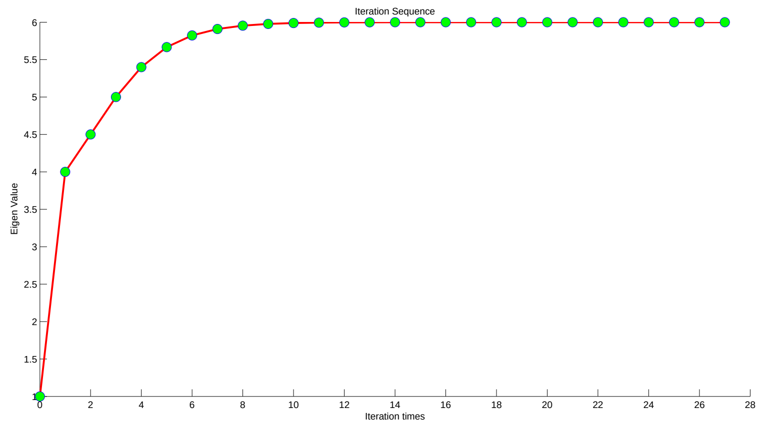

图 7: 幂法特征值迭代序列

对称幂法:

```
def symPowerMethod(A,x0):
    N, TOL = 100, 1e-7
    x0 = x0 / np.linalg.norm(x0, 2);
    for i in range(0, N):
        y = A * x0
        u = x0.T*y
        if np.linalg.norm(y, 2) < 1e-10:
            print('A has eigen value 0. select new vector x and restart.')
            return None
        norm_y = np.linalg.norm(y)
        err = np.linalg.norm(x0 - y / norm_y)
        x0 = y / norm_y;
        if err < TOL:
            return u, x0, i
    return None
```

Max Eigen Value: 6.000000, Iteration time 24

特征向量:

$$\mathbf{x} = (0.5774, -0.5774, 0.5774)$$

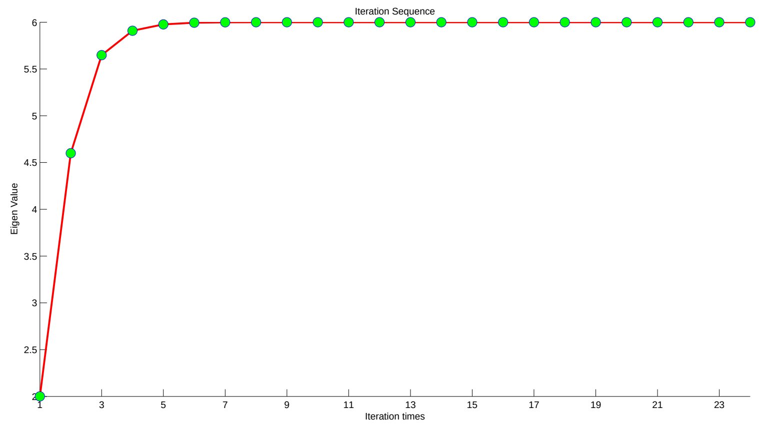

图 8: 对称幂法特征值迭代序列

反幂法:

```
A = [4,-1,1;-1,3,-2;1,-2,3]; n = length(A); TOL = 1e-7;
Ma = zeros(1,n);
for i=1:n
    for j=1:n
        if i==j
            Ma(i) = Ma(i) + A(i,i);
        else
            Ma(i) = Ma(i) + abs(A(i,j));
        end
    end
end
change = max(Ma) + 2; A = A - change*eye(n);
L=eye(n,n); U=zeros(n,n);
for k=1:n
    for j=k:n
        U(k,j)=A(k,j)-sum(L(k,1:k-1).*U(1:k-1,j)');
    end
    for i=k+1:n
        L(i,k)=(A(i,k)-sum(L(i,1:k-1).*U(1:k-1,k)'))/U(k,k);
    end
end
x0 = [1;1;1]; MAXN = 100; mx0 = max(x0);y0 = x0/mx0;
for CNT = 1:MAXN
    b = y0;
    X=zeros(1,3);Y=zeros(1,3);
```

```
    Y(1)=b(1);
    for i=2:n
        for j=1:i-1
            b(i)=b(i)-L(i,j)*Y(j);
        end
        Y(i)=b(i);
    end
    X(n)=Y(n)/U(n,n);
    for i=(n-1):-1:1
        for j=n:-1:i+1
            Y(i)=Y(i)-U(i,j)*X(j);
        end
        X(i)=Y(i)/U(i,i);
    end
    x = X';mx = max(x); y0 = x/mx;
    if abs(mx-mx0) < TOL
        fprintf('Answer %f Iteration time %d\n',change-1/mx,CNT)
        disp(x)
        return
    end
    x0 = x; mx0 = mx;
end
fprintf('No solution')
```

Max Eigen Value: 6.000000, Iteration time 20

Eigen Vector: $\mathbf{x} = (-0.5000, 0.5000, -0.5000)^T$

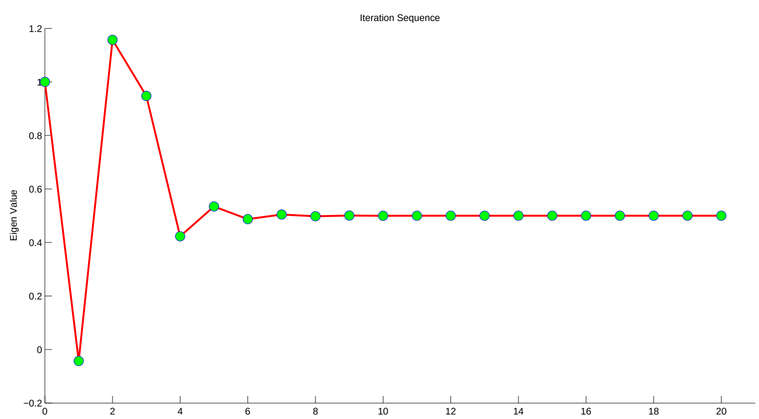

图 9: 反幂法特征值迭代序列

4 第四章: 拉格朗日插值

4.1 基本概念

拉格朗日插值多项式 对于给定的 $n+1$ 个点, n 次拉格朗日插值多项式以此通过这 $n+1$ 个点.

拉格朗日多项式形式如下:

$$P\left(x\right)=\sum_{k=0}^{n}L_{n,k}\left(x\right)f\left(x\right)$$

.

其中 $L_{n,k}\left(x\right)$ 为基函数, 满足这样的性质:

$$L_{n,k}\left(x_i\right)=\begin{cases}0,\ i\neq k\\ 1,\ i=k\end{cases}$$

取

$$L_{n,k}\left(x\right)=\prod_{i=0,i\neq k}^{n}\frac{x-x_i}{x_k-x_i}$$

.

拉格朗日余项 $f\left(x\right)=P\left(x\right)+R\left(x\right)$, 其中 $P\left(x\right)$ 为 n 次多项式,

$$R\left(x\right)=\frac{f^{(n+1)}\left(\xi\left(x\right)\right)}{(n+1)!}\prod_{i=0}^{n}\left(x-x_i\right)$$

为余项. 用于误差估计.

4.2 上机实验

在区间 $[-5,5]$ 上, 生成 11 个等距插值节点 $x_i,i=0,1,\cdots,10$. 在相应插值节点上计算函数

$$y\left(x\right)=\frac{1}{1+x^2} \tag{5}$$

的函数值作为观测值, $y\left(x_i\right),i=0,1,2,\cdots,10$.

1. 利用这 11 个数据点, 生成一个 10 次拉格朗日插值多项式 $P_{10}\left(x\right)$, 并做出插值函数与原函数的对比结果图.

2. 利用此多项式近似计算
$$\int_{-5}^{5} \frac{1}{1+x^2} dx \approx \int_{-5}^{5} P_{10}(x)\, dx \tag{6}$$
与解析解比较, 分析误差产生的原因.

3. 利用 $\{(x_i, y(x_i))\}_{i=0}^{10}$, 构造分片线性插值多项式 $P(x)$, 并利用此分片插值多项式近似计算积分
$$\int_{-5}^{5} \frac{1}{1+x^2} dx \approx \int_{-5}^{5} P_{10}(x)\, dx \tag{7}$$

4. 若希望提高积分的计算精度, 试提出你自己的建议, 并进行实验测试验证.

求解:

分析: 当插值点比较多的时候, 拉格朗日插值多项式的次数可能会很高, 因此具有数值不稳定的特点. 这类现象称为龙格现象. 因此, 要尝试分段, 用较低次数的插值多项式.

由于函数具有对称性, 因此可以分成左右两个五次多项式进行插值, 代价是牺牲了一部分连续性.

1. 绘图:

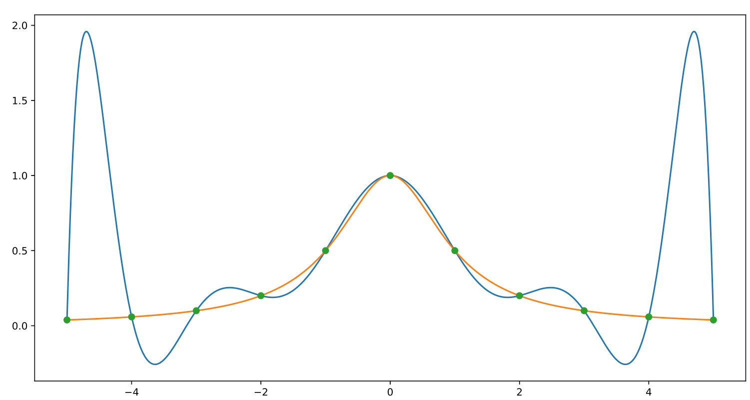

图 10: 10 次拉格朗日插值多项式与原函数比较图

代码:

```
import numpy as np
import matplotlib.pyplot as plt
def get_li(xi, x_set = []):
    def li(Lx):
        W,c = 1., 1.
        for each_x in x_set:
            if each_x != xi:
                W = W * (Lx - each_x)
                c = c * (xi - each_x)
        return W / c
    return li
def get_Lxfunc(x = [], fx = []):
    set_of_lifunc = []
    for each in x:
        lifunc = get_li(each, x)
        set_of_lifunc.append(lifunc)

    def Lxfunc(Lx):
        res = 0
        for i in range(len(x)):
            res = res + fx[i]*set_of_lifunc[i](Lx)
        return res

    return Lxfunc
l,r,cnt = -5.,5.,11
x = np.linspace(l,r,cnt)
y0 = 1/(1+x**2)
Lx = get_Lxfunc(x,y0)
tmpx = np.linspace(l,r,100)
tmpy0 = 1/(1+tmpx**2)
tmpy = [Lx(i) for i in tmpx]
plt.plot(tmpx, tmpy, tmpx, tmpy0, x, y0, 'o')
plt.show()

```

2. 利用拉格朗日插值函数近似积分、分片插值

   此处采用梯形法进行积分. 由于梯形法代码在积分章节已经出现, 此处不再赘述. 误差产生的原因:

   拉格朗日插值多项式具有误差, 且剧烈抖动 (龙格现象).

   梯形积分公式具有误差. 结果:

10 次拉格朗日插值	分片线性插值	两个拉格朗日 5 次插值	真实值
4.673301	2.756109	2.661199	2.746802

3. 提高积分精确度:

   此时应当采取辛普森积分等, 以提高精确度.

   设置合适的拉格朗日插值多项式次数.

5 第五章: 最小二乘拟合

5.1 基本概念

一次函数拟合推导 Cost Function:

$$Q=\sum_{i=1}^{m}\left(y_i-\beta_0-\beta_1 x\right)^2$$

为使 Cost Function 最小, 对 β_0,β_1 分别求偏导数, 有

$$\begin{cases}\dfrac{\partial Q}{\partial \beta_0}=-2\sum\limits_{i=1}^{m}\left(y_i-\beta_0-\beta_1 x_i\right)=0\\ \dfrac{\partial Q}{\partial \beta_1}=-2\sum\limits_{i=1}^{m}\left(y_i-\beta_0-\beta_1 x_i\right)x_i=0\end{cases}$$

求解方程组, 可得

$$\begin{cases}\beta_0=\dfrac{\sum\limits_{i=1}^{m}x_i^2\sum\limits_{i=1}^{m}y_i-\sum\limits_{i=1}^{m}x_iy_i\sum\limits_{i=1}^{m}x_i}{m\left(\sum\limits_{i=1}^{m}x_i^2\right)-\left(\sum\limits_{i=1}^{m}x_i\right)^2}\\ \beta_1=\dfrac{m\sum\limits_{i=1}^{m}x_iy_i-\sum\limits_{i=1}^{m}x_i\sum\limits_{i=1}^{m}y_i}{m\left(\sum\limits_{i=1}^{m}x_i^2\right)^2-\left(\sum\limits_{i=1}^{m}x_i\right)^2}\end{cases}$$

一般多项式推导 拟合多项式

$$P_n\left(x\right)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$$

目标:

$$\min E=\sum_{i=1}^{m}\left(y_i-P_n\left(x_i\right)\right)^2=\sum_{i=1}^{m}\left(y_i-\sum_{k=1}^{n}a_kx_i^k\right)^2$$

令

$$0=\frac{\partial E}{\partial a_j}=-2\sum_{i=1}^{m}y_ix_i^j+2\sum_{k=1}^{n}a_k\sum_{i=1}x_i^{j+k}$$

即

$$\sum_{k=1}^{n}a_k\sum_{i=1}x_i^{j+k}=\sum_{i=1}^{m}y_ix_i^j$$

令

$$\mathbf{R}=\begin{pmatrix} x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_m^n & x_m^{n-1} & \cdots & x_m & 1 \end{pmatrix},\mathbf{a}=\begin{pmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \\ a_0 \end{pmatrix},\mathbf{y}=\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \\ y_m \end{pmatrix}$$

则 $\mathbf{R}^T\mathbf{Ra}=\mathbf{R}^T\mathbf{y}$, 即 $\mathbf{a}=\left(\mathbf{R}^T\mathbf{R}\right)^{-1}\mathbf{R}^T\mathbf{y}$

三次样条 样条是一种特殊的分段函数. 由于样条插值可以使用低阶多项式样条实现较小的插值误差, 这样就避免了使用高阶多项式所出现的龙格现象.

对于 $n+1$ 个给定点的数据集, 我们可以使用 n 段三次多项式在数据点之间构建一个三次样条. 如果

$$S\left(x\right)=\begin{cases} S_0\left(x\right),x\in\left[x_0,x_1\right] \\ S_1\left(x\right),x\in\left[x_1,x_2\right] \\ \vdots \\ S_{n-1}\left(x\right),x\in\left[x_{n-1},x_n\right] \end{cases}$$

表示对函数 f 进行插值的样条函数, 那么需要插值特性 $S\left(x_1\right)=f\left(x_1\right)$
样条可相互连接:

$$S_{i-1}\left(x_i\right)=S_i\left(x_i\right)$$

两次连续可导:

$$S_{i-1}^{'}\left(x_i\right)=S_i^{'}\left(x_i\right),i=1,\cdots,n$$
$$S_{i-1}^{''}\left(x_i\right)=S_i^{''}\left(x_i\right),i=1,\cdots,n$$

5.2 上机实验

P184 算法与程序:2

题目 根据下列数据, 使用例 5.3 中的幂曲线拟合, 写一个程序求解重力常量 g.

表 1: a

t_k	d_k
0.200	0.1960
0.400	0.7835
0.600	1.7630
0.800	3.1345
1.000	4.8975

表 2: b

x_k	d_k
0.2	0.1965
0.4	0.7855
0.6	1.7675
0.8	3.1420
1.0	4.9095

求解 由物理学公式知, 运动方程为

$$d_k = v_0 t + \frac{1}{2} g t^2$$

运行结果:$g = 9.80699693564862$

代码:

```
import numpy as np
import math
def cal(x,y):
    a = np.sum(np.power(x,2)*y.T)/np.sum(np.power(x,4))
    return a
x1 = np.array([0.2,0.4,0.6,0.8,1.0])
y1 = np.array([0.196,0.7835,1.7630,3.1345,4.8975])
x2 = np.array([0.2,0.4,0.6,0.8,1.0])
y2 = np.array([0.1965,0.7855,1.7675,3.1420,4.9095])
a = (cal(x1,y1)+cal(x2,y2))/2
g = 2*a print(g)
```

P207 算法与程序:3

题目 使用上题的程序, 根据点 $(0,1), (1,0), (2,0), (3,1), (4,2), (5,2), (6,1)$, 求 5 种不同的三次样条插值, 其中 $S'(0) = -0.6, S'(6) = -1.8, S''(0) = 1, S''(6) = -1$. 在同一坐标系下, 画出这 5 个三次样条插值和这些数据点.

求解:

五种三次样条插值, 只有端点约束不同. 根据不同约束条件修改参数.

1. 三次压紧样条:
$$s'(0) = -0.6, s'(6) = -1$$
2. Natural 三次样条:
$$s''(0) = 0, s''(6) = -0$$
3. 外推三次样条: 通过对点 x_1 和 x_2 外推得到 $s''(0)$, 对点 x_{n-1} 和 x_{n-2} 外推得到 $s''(6)$
4. 抛物线终结样条:
$$s'''(0) = 0, s'''(6) = -0$$
5. 端点曲率调整样条
$$s''(0) = 1, s''(6) = -1$$

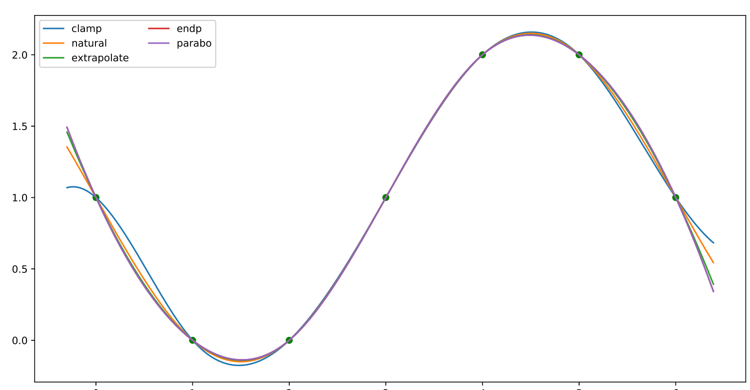

图 11: 5 种样条曲线

代码:

```
import numpy as np
import scipy.interpolate as intp
import matplotlib.pyplot as plt
x = np.array([0,1,2,3,4,5,6])
```

```
y = np.array([1,0,0,1,2,2,1])
# clamp:
clamp = intp.CubicSpline(x,y,bc_type=((1,-0.6),(1,-1)))
#natural
natural = intp.CubicSpline(x,y,bc_type=((2,0.0),(2,0.0)))
#extrapolated
extrap = intp.CubicSpline(x,y,extrapolate=True)
#parabo
defa = intp.CubicSpline(x,y).derivative(2)
secondss = defa([x[1],x[-2]])
parabo = intp.CubicSpline(x,y,bc_type=((2,secondss[0]),(2,secondss[1])))

endp = intp.CubicSpline(x,y,bc_type=((2,1),(2,-1)))

t = np.arange(-0.3, 6.4, 0.01)
plt.plot(x,y,'go')
plt.plot(t,clamp(t),label='clamp')
plt.plot(t,natural(t),label='natural')
plt.plot(t,extrap(t),label='extrapolate')
plt.plot(t,endp(t),label='endp')
plt.plot(t,parabo(t),label='parabo')
plt.legend(loc='top left', ncol=2)
plt.show()
```

求解

6 第六章数值微分

6.1 基本概念

6.2 上机实验

6.1.6 算法与程序 (P235):1(b)

题目 用程序 6.1 求解下列函数在 x 处的导数近似值, 精度为小数点后 13 位. 注: 有必要改写程序中的 $max1$ 的值和 h 的初值值. $\tan\left(\cos\left(\frac{\sqrt{5}+\sin(x)}{1+x^2}\right)\right)$; $x = \frac{1+\sqrt{5}}{3}$

求解 实验分析: 使用精度为 $O\left(h^2\right)$ 的中心差分公式求, 在给定点的导数近似值. 由于精度量级为 $O\left(h^2\right)$, 取 $h=1e-7$.

$$f'\left(x\right)\approx\frac{f\left(x+10^{-k}h\right)-f\left(x-10^{-k}h\right)}{2\left(10^{-k}h\right)}$$

运行结果: $f'\left(x\right)=1.2285974236581065$

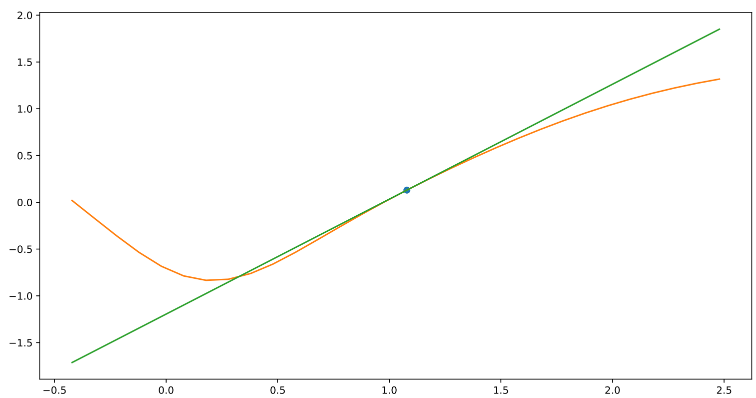

图 12: 微分割线示意图

代码:

```
import numpy as np
import scipy.interpolate as intp
import matplotlib.pyplot as plt
import scipy.misc as misc
import math
def fx(x):
    return math.tan(math.cos((5**0.5+math.sin(x))/(1+x**2)))
def ffx(xx):
    global dx,x,y
    return dx*(xx-x)+y
x = (1+5**0.5)/3.0
y = fx(x)
t = np.arange(x-1.5,x+1.5,0.1)
yf = np.array(list(map(fx, t)))
dx = misc.derivative(fx,x,dx=1e-7)
print(dx)
yff = np.array(list(map(ffx, t)))
plt.plot(x,y,'o')
```

```
plt.plot(t,yf,label='fx')
plt.plot(t,yff,label='ffx')
plt.show()
```

7 第七章: 数值积分

7.1 基本概念

梯形法简介 把被积区间划分成若干小区间, 对于每一小区间, 用函数两个端点与 x 轴围成的梯形面积, 近似代替函数图像与 x 轴围成的面积. 将所有小梯形面积累加, 即为结果.(或者说, 每一个小区间都用线性拉格朗日插值多项式来代替).

积分公式可以表述为:

$$f(x)\,dx = \frac{h}{2}\left[f(a) + 2\sum_{j}^{n-1} f(x_j) + f(b)\right] - \frac{b-a}{12}h^2 f''(\mu)$$

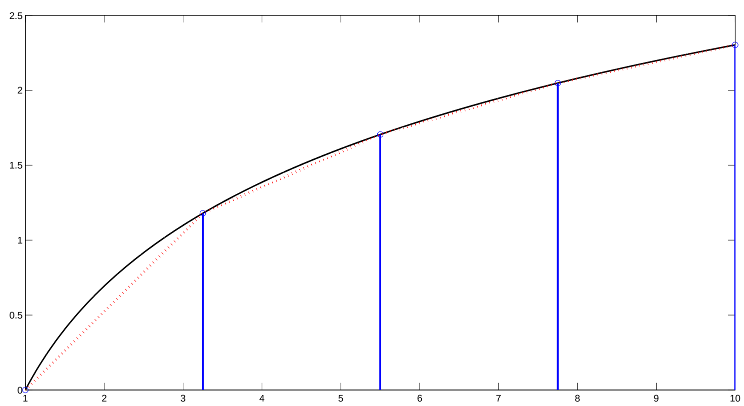

图 13: 梯形法图示

辛普森公式推导 设 $x_1 = x_0+h, x_2 = x_0+2h\,(h>0)\ \forall x \in [x_0, x_2]$,$\exists \xi(x) \in [x_0, x_2]$, 使得

$$f(x) = f(x_1) + f'(x_1)(x-x_1) + \frac{f''(x_1)}{2}(x-x_1)^2 +$$

$$\frac{f^{'''}(x_1)}{6}(x-x_1)^3+\frac{f^{(4)}(\xi(x))}{24}(x-x_1)^4$$

. 积分, 得 $\int_{x_0}^{x_2} f(x)\,dx=$

$$\left[f(x_1)(x-x_1)+\frac{f^{'}(x_1)}{2}(x-x_1)^2+\frac{f^{''}(x_1)}{6}(x-x_1)^3+\frac{f^{'''}(x_1)}{24}(x-x_1)^4\right]_{x_0}^{x_2}$$

$$+\frac{1}{24}\int_{x_0}^{x_2} f^{(4)}(\xi(x))(x-x_1)^4\,dx$$

由于 $\forall x\in[x_0,x_2],(x-x_1)^4\geq 0$, 因此

$$\frac{1}{24}\int_{x_0}^{x_2} f^{(4)}(\xi(x))(x-x_1)^4\,dx=\frac{f^{(4)}(\xi_1)}{24}\int_{x_0}^{x_2}(x-x_1)^4\,dx=\frac{f^{(4)}(\xi_1)}{120}(x-x_1)^5\,|_{x_0}^{x_2}$$

其中 $\xi_1\in(x_0,x_2)$.

利用 $h=x_2-x_1=x_1-x_0$, 上述积分表述为

$$\int_{x_0}^{x_2} f(x)\,dx=2hf(x_1)+\frac{h^3}{3}f^{''}(x_1)+\frac{f^{(4)(\xi_1)}}{60}h^5$$

.

对于 $f^{''}(x_1)$, 用如下方式处理:

$$f(x_0+h)=f(x_0)+f^{'}(x_0)h+\frac{1}{2}f^{''}(x_0)h^2+\frac{1}{6}f^{(3)}(x_0)h^3+\frac{1}{24}f^{(4)}(\xi_1)h^4$$

.

$$f(x_0+h)=f(x_0)-f^{'}(x_0)h+\frac{1}{2}f^{''}(x_0)h^2-\frac{1}{6}f^{(3)}(x_0)h^3+\frac{1}{24}f^{(4)}(\xi_1)h^4$$

. 两式相加,

$$f(x_0+h)+f(x_0-h)=2f(x_0)+f^{''}(x_0)h^2+\frac{h^4}{24}\left[f^{(4)}(\xi_1)+f^{(4)}(\xi_{-1})\right]$$

取 $f^{(4)}(\xi)=\frac{1}{2}\left[f^{(4)}(\xi_1)+f^{(4)}(\xi_{-1})\right]$, 把 $f^{''}(x_1)$ 的值代入, 可以求得

$$\int_{x_0}^{x_2} f(x)\,dx=\frac{h}{3}\left[f(x_0)+4f(x_1)+f(x_2)\right]-\frac{h^5}{12}\left[\frac{1}{3}f^{(4)}(\xi_2)-\frac{1}{5}f^{(4)}(\xi_1)\right]$$

余项可以写成 $\frac{h^5}{90}f^{(4)}(\xi)$ 的形式.

对于复合辛普森公式,

$$\int_a^b f(x)\,dx = \sum_{j=1}^{n/2} \int_{x_{2j-2}}^{x_{2j}} f(x)\,dx =$$

$$\frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f\left(x_{2j-1} + f(x_n)\right)\right] + E(f)$$

其中 $E(f) = -\frac{h^5}{90}\sum_{j=1}^{n/2} f^{(4)}(\xi_j) = -\frac{b-a}{180}h^4 f^{(4)}(\mu)$

辛普森公式实质: 相邻的 3 个点用二次多项式进行插值, 误差项可以用上述方式推导.

辛普森公式图示:

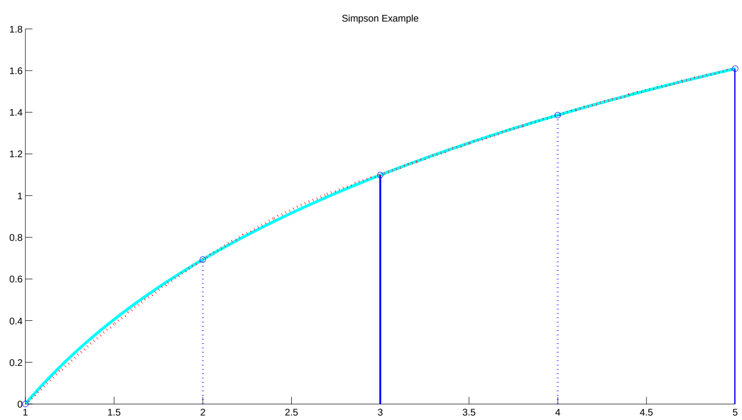

图 14: 辛普森公式图示

7.2 上机实验

7.2.3 算法与程序 (P262) 2

题目 用程序 7.2(组合辛普森公式) 求习题 2 中的定积分, 精确到小数点后 11 位.

$$Length = \int_a^b \sqrt{1 + (f'(x))^2}\,dx$$

1. $f(x) = x^3, 0 \le x \le 1$

2. $f(x) = \sin(x), 0 \le x \le \pi/4$

3. $f(x) = \exp^{-x}, 0 \le x \le 1$

求解 组合辛普森公式代码:

```
def simpson(M,a,b):
    x = np.linspace(a,b,2*M+1)
    h = x[1] - x[0]
    ss0,ss1 = 0.0, 0.0
    for i in range(1,M):
        ss0 += f(x[2*i])
    for i in range(1,M+1):
        ss1 += f(x[2*i-1])
    return h/3*(f(a)+f(b)) + 2.0*h/3.0 * ss0 + 4.0*h/3.0 * ss1
```

函数编号	1	2	3
计算值	1.5478656546836094	1.311442498215542	1.1927014019721596

7.2.3 算法与程序 (P262) 3

题目 修改组合梯形公式, 使之可以求只有若干点函数值已知的函数积分. 将程序 7.1 修改为求区间 $[a, b]$ 上过 M 个给定点的函数 $f(x)$ 的积分逼近. 注意节点不需要等距.

注: 程序 7.1 为组合梯形公式, 代码如下:

```
function s=trap1(f,a,b,M)
h = (b-a)/M;
s = 0;
for k=1:(M-1)
  x = a+h*k;
  s = s + feval(f,x);
end
s = h * (feval(f,a)+feval(f,b))/2+h*s;
```

修改结果:

```
def trap2(x):
    s = 0.0
    for k in range(1, len(x)):
        s += (x[k] - x[k-1]) * (f(x[k]) + f(x[k-1])) / 2.0
    return s
```

假定端点 a,b 均在数组 x 内.
利用该程序求过点 $\left\{\left(\sqrt{k^2+1},k^{1/3}\right)\right\}_{k=0}^{13}$ 的函数积分逼近. 求解结果: $ans=21.84106920647963$

8 常微分方程数值解

8.1 基本概念

欧拉显式格式 :
公式推导: 根据泰勒展开, 有

$y\left(t_{j+1}\right)+hf\left(t_j,y\left(t_j\right)\right)+\frac{h^2}{2}y^{''}\left(\xi_j\right)$

忽略余项, 可以得到欧拉显式格式:

$$\begin{cases} w_0=\alpha \\ w_{j+1}=w_j+hf\left(t_j,w_j\right) \end{cases}$$

误差分析:

假设 f 在 $D=\left\{(t,y)\left|a\le t\le b,-\infty<y<\infty\right.\right\}$ 连续,

满足 Lipschitz 条件 (Lipschitz 常数为 L),

且满足 $\left|y^{''}(t)\right|\le M,\forall t\in[a,b]$.

则 $\left|y\left(t_j\right)-w_j\right|\le\frac{hM}{2L}\left[e^{L(t_j-a)}-1\right]$.

在某些情况下, M,L 较难确定. 但可以发现, 随着向后递推, 欧拉显式格式的误差逐渐增大.

欧拉法误差分析:

对于 $\left|y\left(x_j\right)-w_j\right|\le\frac{hM}{2L}\left[e^{L(x_j-a)}-1\right]$, 需要确定 M,L 的值.

由于 $M=\max\left|y^{''}(x)\right|,x\in[a,b]$,

而 $y(x)=\sqrt{2x+1}$,$y^{''}(x)=\frac{-1}{(2x+1)^{3/2}}$.

有 $M=\max\left|y^{''}(x)\right|=\left|f^{''}(-1)\right|=1$,

$\frac{\partial f}{\partial y}=1+\frac{2x}{y^2}=1+\frac{2x}{1+2x}$, $L=\max\left|\frac{\partial f}{\partial y}\right|=\left|\frac{\partial f}{\partial y}\Big|_{x=1}\right|\approx 1.6667$.

因此 $\left|y(1)-w_n\right|\le\frac{hM}{2L}\left[e^L-1\right]$.

休恩方法 对于 $y'(t)=f(t,y(t))$, $y(t_0)=y_0$.

要得到解 (t_1, y_1), 可以用微积分基本定理, 在 $[t_0, t_1]$ 上对 $y'(t)$ 积分得:

$$\int_{t_0}^{t_1} f(t, y(t))\, dt = \int_{t_0}^{t_1} y'(t)\, dt = y(t_1) - y(t_0)$$

其中 $y'(t)$ 的不定积分为待求函数 $y(t)$. 可得:

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(t, y(t))\, dt$$

如果采用步长为 $h = t_1 - t_0$ 的梯形公式, 则结果为

$$y(t_1) \approx y(t_0) + \frac{h}{2} f(t_0, y(t_0)) + f(t_1, y(t_1))$$

要继续求解, 需要 $y(t_1)$ 的一个估计值, 欧拉方法的解能够满足这一目的, 将它代入后, 得到求解 (t_1, y_1) 的公式, 称为休恩 (Heun) 方法.

$$y_1 = y(t_0) + \frac{h}{2}(f(t_0, y_0) + f(t_1, y_0 + hf(t_0, y_0)))$$

每一步都用欧拉方法作为预报, 然后用梯形公式进行校正, 得到最终的值. 休恩方法的一般步骤为:

$$\begin{cases} p_{k+1} = y_k + hf(t_k, y_k) \\ t_{k+1} = t_k + h \\ y_{k+1} = y_k + \dfrac{h}{2}(f(t_k, y_k) + f(t_{k+1}, p_{k+1})) \end{cases} \tag{8}$$

龙格-库塔 一般地, 设近似公式为

$$\begin{cases} y_{n+1} = y_n + h\displaystyle\sum_{i=1}^{p} c_i K_i \\ K_1 = f(x_n, y_n) \\ K_i = f\left(x_n + a_i h, y_n + h\displaystyle\sum_{j=1}^{i-1} b_{ij} K_j\right),\ i = 2, 3, \cdots, p \end{cases} \tag{9}$$

其中 a_i, b_{ij}, c_i 都是参数. 若 $p = 2, c_1 + c_2 = \frac{1}{2}, a_2 = b_{21} = 1$, 近似公式为:

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{2}(K_1 + K_2) \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + h, y_n + hK_1) \end{cases} \tag{10}$$

这就是改进的 Euler 公式.
若取 $c_1=0, c_2=1, a_2=b_{21}=\frac{1}{2}$, 则得:

$$\begin{cases} y_{n+1}=y_n+hK_2 \\ K_1=f\left(x_n,y_n\right) \\ K_2=f\left(x_n+\dfrac{h}{2}, y_n+\dfrac{h}{2}K_1\right) \end{cases} \tag{11}$$

则为中点公式.
取 $p=3$, 通过较复杂的计算, 可得

$$\begin{cases} y_{n+1}=y_n+\dfrac{h}{6}\left(K_1+4K_2+2K_3\right) \\ K_1=f\left(x_n,y_n\right) \\ K_2=f\left(x_n+\dfrac{h}{2}, y_n+\dfrac{h}{2}K_1\right) \\ K_3=f\left(x_n+h, y_n-hK_1+2hK_2\right) \end{cases} \tag{12}$$

取 $p=4$, 通过更复杂的计算, 可得

$$\begin{cases} y_{n+1}=y_n+\dfrac{h}{6}\left(K_1+2K_2+2K_3+K_4\right) \\ K_1=f\left(x_n,y_n\right) \\ K_2=f\left(x_n+\dfrac{h}{2}, y_n+\dfrac{h}{2}K_1\right) \\ K_3=f\left(x_n+\dfrac{h}{2}, y_n+\dfrac{h}{2}K_2\right) \\ K_4=f\left(x_n+h, y_n+hK_3\right) \end{cases} \tag{13}$$

上式称为经典的 4 阶 R-K 公式, 是常用的四阶 R-K 公式.

8.2 上机实验

流行病模型: 流行病的数学模型描述如下: 设有 L 个成员构成的群落, 其中有 P 个感染个体, Q 为未感染个体. 令 $y(t)$ 表示时刻 t 感染个体数量. 对于温和的疾病, 如普通感冒, 每个个体保持存活, 流行病从感染者传播到未感染者. 由于两组间有 PQ 种可能的接触, $y(t)$ 的变化率正比于 PQ. 故该问题可以描述为初值问题:

$$y'=ky\left(L-y\right), y\left(0\right)=y_0$$

1. 用 $L=25000, k=0.000003, h=0.2$ 和初值 $y(0)=250$, 并用程序 9.1 计算 $[0,60]$ 上的欧拉近似解.
2. 画出 (a) 中的近似解
3. 通过求 (a) 中欧拉方法的纵坐标平均值来估计平均感染个体数目.
4. 通过用曲线拟合 (a) 中的数据, 并用定理 1.10(积分均值定理), 估计平均感染个体的数目.

   积分均值定理

   若 $f \in C[a,b]$, 则至少存在一点 $c \in (a,b)$, 满足

$$\frac{1}{b-a}\int_a^b f(x)\,dx = f(c)$$

   $f(c)$ 是函数 f 在区间 $[a,b]$ 内的平均值.

分析: 对于 $y' = ky(L-y)$, 经过分离变量积分可得

$$y = L\frac{C\exp^{kLt}}{1+C\exp^{kLt}}$$

. 注意到, 当 $kL>0$ 时, $\lim\limits_{t\to+\infty} y = L$

```python
def euler(x0, y0, h, N):
    n, xlist, ylist = 1,[x0], [y0]
    while (n <= N):
        x1,y1 = x0 + h, y0 + h* f(x0, y0)
        print("%.2f, %.6f" % (x1, y1))
        n,x0,y0 = n+1,x1,y1
        xlist.append(x1), ylist.append(y1)
    return xlist, ylist
```

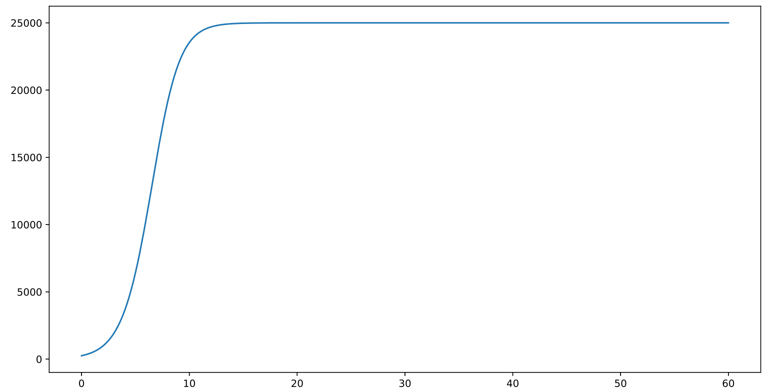

图 15: 欧拉法. 观察图像发现, 函数收敛于 25000, 与求解微分方程得到的结果吻合

计算 $t \in [0, 60]$ 时的平均值: 22301.559616

拟合: 函数的形状与 3 次函数局部相近, 考虑用 3 次多项式进行拟合.

考虑一阶积分-常微分方程

$$y' = 1.3y - 0.25y^2 - 0.0001y\int_0^t y(\tau)\,d\tau$$

1. 在区间 $[0, 20]$ 上, 用欧拉方法和 $h = 0.2, y(0) = 250$ 以及梯形公式求方程的近似解. 提示: 欧拉方法 (6) 的一般步长为:

$$y_{k+1} = y_k + h\left(1.3y_k - 0.25y_k^2 - 0.0001y_k\int_0^{t_k} y(\tau)\,d\tau\right)$$

   如果梯形公式用于逼近积分, 则该表达式为:

$$y_{k+1} = y_k + h\left(1.3y_k - 0.25y_k^2 - 0.0001y_k T_k(h)\right)$$

   其中 $T_0(h) = 0$ 且

$$T_k(h) = T_{k-1}(h) + \frac{h}{2}(y_{k-1} + y_k),\ k = 0, 1, \cdots, 99$$

2. 用初值 $y(0) = 200$ 和 $y(0) = 300$ 重复 (a) 的计算.

3. 在同一坐标系下, 画出 (a) 和 (b) 的近似解

求解: 原数据增长速率过快, 很快就导致浮点溢出. 因此, 使用 $2.50, 2.00, 3.00$ 这三组数据进行计算.

```
def f(t0,y0):
    return 1.3*y0-0.25*y0*y0-0.0001*t0*y0

def euler2(x0, y0,t0, h, N):
    n, xlist, ylist = 1,[x0], [y0]
    while (n <= N):
        x1 = x0 + h
        y1 = y0 + h * f(t0, y0)
        t1 = t0 + h/2 *(y0 + y1)
        n, x0, y0, t0 = n+1, x1, y1, t1
        xlist.append(x1), ylist.append(y1)
    return xlist, ylist
```

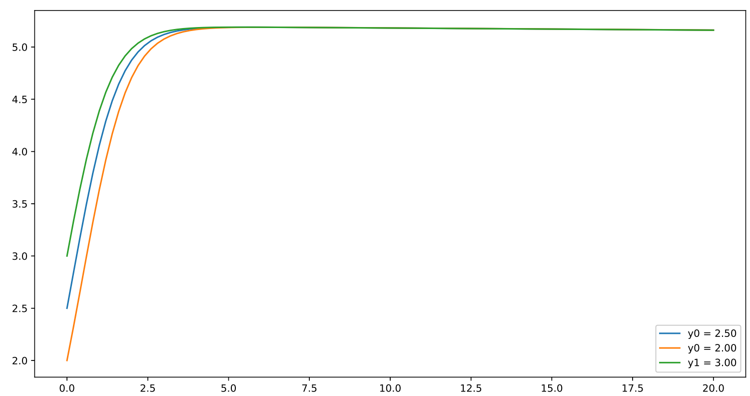

图 16: 观察发现, 这三个图像均有收敛的迹象. 收敛于 5.1 左右

$y(0) = 2.50$	$y(0) = 2.00$	$y(0) = 3.00$
5.161256	5.161618	5.160968

用休恩方法求解微分方程 $y' = 3y + 3t, y(0) = 1, y(t) = \frac{4}{3}e^{3t} - t - \frac{1}{3}$

1. 令 $h = 0.1$, 程序 9.2 执行 20 步, 然后令 $h = 0.05$, 程序 9.2 执行 40 步.

2. 比较 (a) 中两个近似解与精确解 $y(2)$

3. 当 h 减半时, (a) 中的全局误差是否和预期相符?

4. 将两个近似解和精确解画在同一坐标系中.

求解:

```
def Heun(x0, y0, h, N):
    n = 1
    xlist, ylist = [x0], [y0]
    while (n <= N):
        x1 = x0 + h
        f0 = f(x0, y0)
        p1 = y0 + h * f0
        y1 = y0 + h/2*(f0 + f(x1,p1))
        print("%.2f, %.6f" % (x1, y1))
        n,x0,y0 = n+1,x1,y1
        xlist.append(x1), ylist.append(y1)
    return xlist, ylist
```

结果:

情况	$h = 0.1$	$h = 0.05$	真值
y	$\hat{y}(2) = 539.589989$	$\hat{y}(2) = 536.577420$	$y(2) = 535.571724$
Err	4.018265	1.005696	0

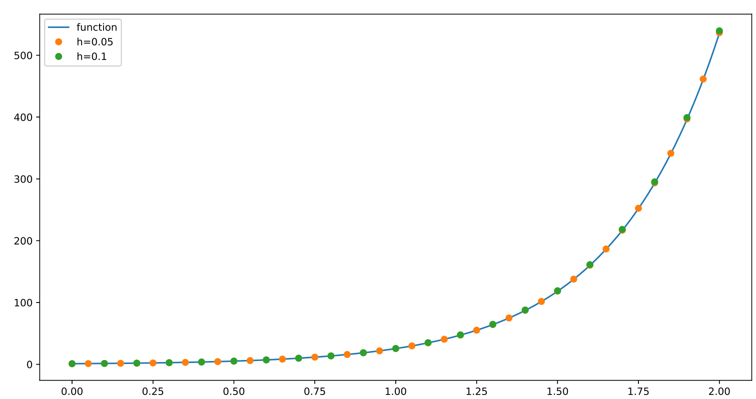

图 17: Heun 预估校正. 可以观察到, 随着 t 增长, y 存在少量误差

用 $N = 4$ 的龙格-库塔求解微分方程 $y' = 3y + 3t, y(0) = 1, y(t) = \frac{4}{3}e^{3t} - t - \frac{1}{3}$

1. 令 $h = 0.1$, 程序 9.4 执行 20 步, 然后令 $h = 0.05$, 程序 9.4 执行 40 步

2. 比较 (a) 中两个近似解与精确解 $y(2)$
3. 当 h 减半时, (a) 中的全局误差是否和预期相符?
4. 在同一坐标系中画出两个近似解和精确解.

求解:

```
import math
def caly(x, y):
    return 4.0/3.0*math.exp(3*x)-x-1.0/3.0
def f(x, y):
    return 3*caly(x,y)+3*x
def rf4(x0, y0, h, N):
    n = 1
    while (n <= N):
        x1 = x0 + h
        k1 = f(x0, y0)
        k2 = f(x0 + h / 2, y0 + h * k1 / 2)
        k3 = f(x0 + h / 2, y0 + h * k2 / 2)
        k4 = f(x1, y0 + h * k3)
        y1 = y0 + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6
        print("%.2f, %.6f" % (x1, y1))
        n,x0,y0 = n + 1, x1, y1
  return x0,y0
```

对于该微分方程, 其解析解为: $y = \frac{4}{3}e^{3t} - t - \frac{1}{3}$

情况	$h = 0.1$	$h = 0.05$	真值
y	$\hat{y}(2) = 535.573230$	$\hat{y}(2) = 535.571819$	$y(2) = 535.571724$
Err	0.0015	9.5e-05	0

观察表格, 可发现, $Err_1/Err_2 \approx 15.78 \approx 16$. 而 4 阶龙格-库塔方法中, 误差项是 ch^5. 当步长减半时, 误差近似变成原来的 $\frac{1}{2^4}$. 可能存在截断误差与舍入误差等.

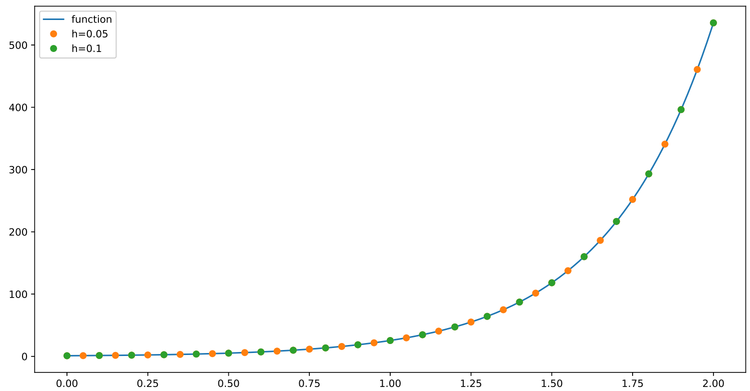

图 18: 4 阶龙格-库塔图像

9 总结

在本学期数值计算实验中, 我掌握了如下技能:

1. 二分法、牛顿法;
2. 多项式插值, 包括拉格朗日插值、Hermite 插值等;
3. 根据插值多项式进行数值积分;
4. 求给定初始条件的常微分方程组;
5. 求解矩阵特征值;
6. 求解线性方程组;
7. 通过最小二乘进行多项式拟合;
8. 加深对数学分析、高等代数的理解;
9. Latex, python 画图工具的使用;
10. 误差分析.

最后, 感谢刘保东老师, 在本学期的数值计算课程中, 让我对数学、计算有了新的认识.