

# 计算物理学作业4

许嘉琪 1500011417

January 1, 2018

## 1 常微分方程的初值问题

### 1.1 重新定义参数

通过观察方程的形式, 令  $x' = \frac{\delta}{\gamma}x$ ,  $y' = \frac{\beta}{\alpha}y$ ,  $t' = \gamma t$  之后, 方程组变为:

$$\begin{cases} \dot{x}' = \frac{\alpha}{\gamma}x'(1-y') \\ \dot{y}' = y'(x'-1) \end{cases}$$

### 1.2 固定点解

#### 1.2.1 求解

首先,  $x$  和  $y$  都不随时间变化的解一定满足  $\dot{x}' = \dot{y}' = 0$ 。这样考虑方程组:

$$\begin{cases} x'(1-y') = 0 \\ y'(x'-1) = 0 \end{cases}$$

得到两组解:  $x' = y' = 0$  或  $x' = y' = 1$ 。也就是对于原来的物理量  $x, y$  来说:  $\begin{cases} x' = 0 \\ y' = 0 \end{cases}$  或者  $\begin{cases} x = \frac{\gamma}{\delta} \\ y = \frac{\alpha}{\beta} \end{cases}$ 。

#### 1.2.2 稳定性

先分析  $(0,0)$  点, 分析约化后的  $(x', y')$ 。假设  $x$  从原点附近的  $x_0$  开始, 按照方程我们有  $\frac{dx}{x} = \frac{\alpha}{\gamma} dt$ , 于是积分后:  $x(t) = x_0 e^{(\frac{\alpha}{\gamma})t}$  可以看出  $x(t)$  是不稳定的。

再分析  $(1,1)$ , 同假设  $x$  从原点附近的  $x_0$  开始, 按照原始的方程我们有

$$\begin{aligned} \dot{x} &= \alpha(\frac{\gamma}{\delta} + \Delta x) - \beta(\frac{\gamma}{\delta} + \Delta x)(\frac{\alpha}{\beta} + \Delta y) \approx -\frac{\beta\gamma}{\delta} \Delta y \\ \dot{y} &= \delta(\frac{\gamma}{\delta} + \Delta x)(\frac{\alpha}{\beta} + \Delta y) - \gamma(\frac{\alpha}{\beta} + \Delta y) \approx \frac{\delta\alpha}{\beta} \Delta x \end{aligned}$$

求导进行消元, 得:  $\begin{cases} \ddot{x} + \alpha\gamma x = 0 \\ \ddot{y} + \alpha\gamma y = 0. \end{cases}$  可见这个点是稳定的。

### 1.2.3 系统守恒量

将两个方程相除,获得关于x 和y 的微分方程 $\frac{\alpha-\beta y}{y}dy = \frac{\delta x-\gamma}{x}dx$  积分得:

$$\frac{\exp(\delta x + \beta y)}{y^\alpha x^\gamma} = Const.$$

### 1.3 RK4数值解

对于这个问题,我定义了结构体vec, 包含 $x'$ 和 $y'$ , 并定义一个update()函数, 用来实现一个步长h下对vec进行更新, 其中RK4算法体现在update()中:

```
void update(type h,type par){
    type ky1=h*(x-1)*y;
    type kx1=h*par*(1-y)*x;
    type ky2=h*(x+0.5*kx1-1)*(y+0.5*ky1);
    type kx2=h*par*(1-y-0.5*ky1)*(x+0.5*kx1);
    type ky3=h*(x+0.5*kx2-1)*(y+0.5*ky2);
    type kx3=h*par*(1-y-0.5*ky2)*(x+0.5*kx2);
    type ky4=h*(x+kx3-1)*(y+ky3);
    type kx4=h*par*(1-y-ky3)*(x+kx3);
    x = x+ kx1/6+kx2/3+kx3/3+kx4/6;
    y = y+ ky1/6+ky2/3+ky3/3+ky4/6;
}
```

参数中的h代表步长, par等于 $\frac{\alpha}{\gamma}$ 。进行反复迭代后, 我利用root中的MultiGraph类进行绘图, 得到的结果见下图:

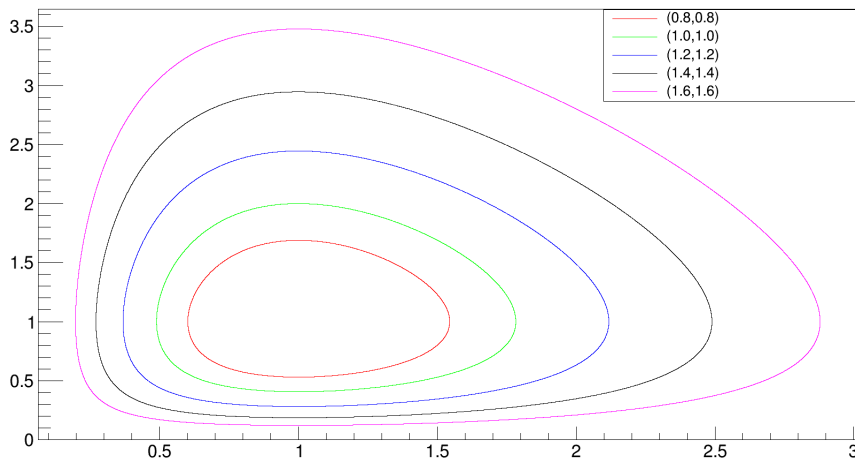


Figure 1: trajectories of  $(x', y')$  for different starting points

## 2 迭代法求解偏微分方程

### 2.1 Jacobi方法

#### 2.1.1 矩阵实现

虽然这是一个稀疏矩阵的求解，我们不妨先利用上一次作业中的matrix类尝试求解：这里，提供了另外几种matrix的构造函数：

1. matrix(int N) 只有一个参数N，用来生成大小为 $N^2 * N^2$ 的矩阵A。其形式为对角线上是4，其余位置分布着一些零散的-1。需要注意的是，在某些边界位置，不应多写-1！
2. matrix(int N, string s="GetD-1") 得到 $D^{-1}$ 矩阵，这在Jacobi方法中，就是一个对角元均为0.25的矩阵。
3. matrix(int N, string s="GetB") 得到一个 $N^2 * 1$ 的列矩阵b，其元素值利用 $2 * \pi^2 * h^2 * \sin(\pi(i+1)h) \sin(\pi(j+1)h)$ 计算，之所以括号内有+1，是因为N个点不包括边界点(包括边界后共N+2个点)。

定义好了这些矩阵后，就可以按照公式： $u^{i+1} = (I - B^{-1}A)u^i + B^{-1}b$ ，简洁紧凑的进行迭代：代码为：

```
u=((I-(D_inv^A))^u)+(D_inv^B);
```

其中hat符号是matrix类重载的运算符，代表矩阵乘法。

#### 2.1.2 非矩阵实现

考虑到 $N = 50$ 的时候需要开一个大小为 $50^4$ 的数组，而其是极其稀疏的，没有必要这么操作，于是，我直接手动进行迭代是的更新操作，具体如下：

```
for (int i=0;i<N;++i){
    for (int j=0;j<N;++j){
        if (j!=0)
            e[i*N+j]+=0.25*u.ele[i*N+j-1];
        if (j!=N-1)
            e[i*N+j]+=0.25*u.ele[i*N+j+1];
        if (i!=0)
            e[i*N+j]+=0.25*u.ele[i*N+j-N];
        if (i!=N-1)
            e[i*N+j]+=0.25*u.ele[i*N+j+N];
        e[i*N+j]+=0.5*PI*PI*h*h*sin(PI*(i+1)*h)*sin(PI*(j+1)*h);
        temp=4*e[i*N+j]-4*u.ele[i*N+j];
        norm=max(norm,temp);
    }
}
```

```
}
```

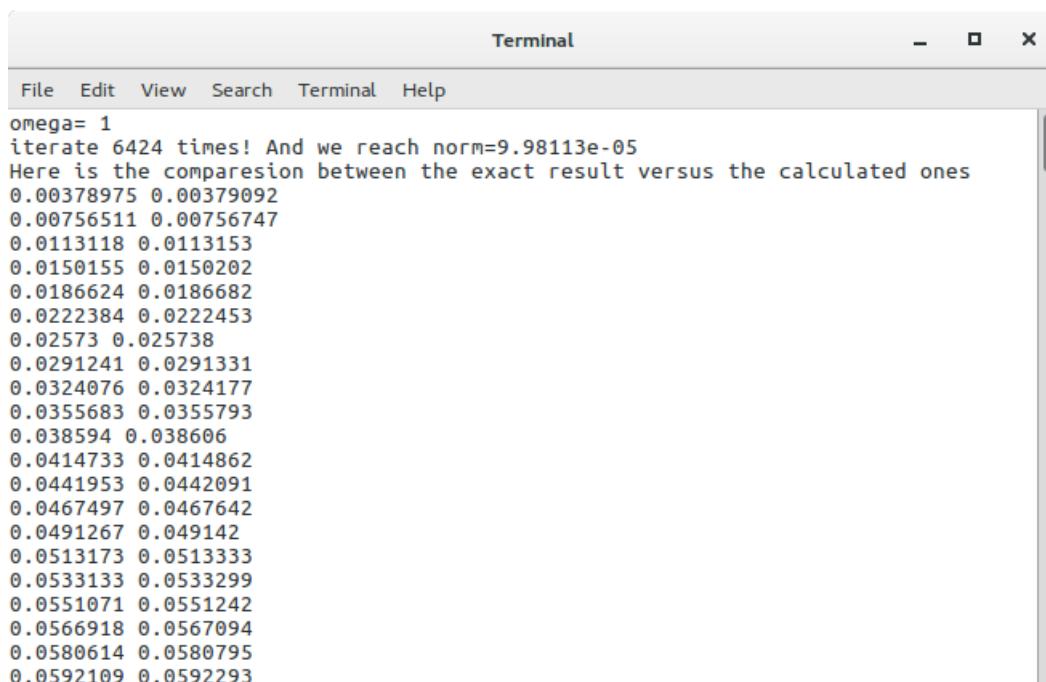
主要的想法就是，每次判断所处理的点是否在边界上，如果不在边界上，就加上相应的它四周的值。最后统计无穷模为所有 $Au-b$ 的元素的 $\max$ 值。

## 2.2 Gauss-Seidel方法

只需要在上面Jacobi算法的迭代的过程中修改两处即可。对于矩阵的 $u$ 的左下区域，每次利用本轮刚刚更新的值去迭代。即只有这两句有所调整：

```
if (j!=0)
    e [ i*N+j ] += 0.25 * e [ i*N+j - 1 ];
if (i!=0)
    e [ i*N+j ] += 0.25 * e [ i*N+j -N ];
```

可以看到由 $u\_ele[i*N+j-1]$ 变为了 $e[i*N+j-1]$ ，即使用刚刚更新的值。可以看到，最后的 $r_i = 9.969 \times$



```
Terminal
File Edit View Search Terminal Help
omega= 1
iterate 6424 times! And we reach norm=9.98113e-05
Here is the comparesion between the exact result versus the calculated ones
0.00378975 0.00379092
0.00756511 0.00756747
0.0113118 0.0113153
0.0150155 0.0150202
0.0186624 0.0186682
0.0222384 0.0222453
0.02573 0.025738
0.0291241 0.0291331
0.0324076 0.0324177
0.0355683 0.0355793
0.038594 0.038606
0.0414733 0.0414862
0.0441953 0.0442091
0.0467497 0.0467642
0.0491267 0.049142
0.0513173 0.0513333
0.0533133 0.0533299
0.0551071 0.0551242
0.0566918 0.0567094
0.0580614 0.0580795
0.0592109 0.0592293
```

Figure 2: G-S at  $N=50$

$10^{-5}$ 。

## 2.3 SOR方法

这里需要计算使得矩阵 $J$ 的谱半径取得最小的 $\omega$ 值。利用正确的公式(讲义公式符号有误)后得到：

$$\omega_b = \frac{2}{1 + \sqrt{1 - \rho(J)^2}}$$

其中 $\rho(J) = \cos(\pi/(N+1))$ 。相应的在迭代程序段再多乘以 $\omega$ 得到的效果如图所示：

```
Terminal
File Edit View Search Terminal Help
omega= 1
iterate 3124 times! And we reach norm=9.96729e-05
Here is the comparesion between the exact result versus the calculated ones
0.00378975 0.00379093
0.00756511 0.00756749
0.0113118 0.0113153
0.0150155 0.0150203
0.0186624 0.0186682
0.0222384 0.0222453
0.02573 0.0257381
0.0291241 0.0291332
0.0324076 0.0324178
0.0355683 0.0355794
0.038594 0.0386061
0.0414733 0.0414863
0.0441953 0.0442092
0.0467497 0.0467643
0.0491267 0.0491421
0.0513173 0.0513334
0.0533133 0.05333
0.0551071 0.0551243
0.0566918 0.0567096
0.0580614 0.0580796
0.0592109 0.0592294
```

Figure 3: SOR method at N=50

## 2.4 迭代次数对比

结果见Table 1.

Method	# of recursion	$r_i/10^{-6}$
Jacobi(10)	296	0.9736
Jacobi(20)	1087	0.9907
Jacobi(50)	6433	0.9982
G-S(10)	150	0.9314
G-S(20)	545	0.9782
G-S(50)	3124	0.996
SOR(10)	32	0.5528
SOR(20)	60	0.9678
SOR(50)	143	0.959

Table 1: comparing three methods