

# Chap4 线性方程组的迭代解法实验报告

李晨昊 2017011466

2019-3-9

## 目录

<b>1</b>	<b>运行代码</b>	<b>2</b>
<b>2</b>	<b>q4.2</b>	<b>2</b>
2.1	任务 . . . . .	2
2.2	解题思路 . . . . .	2
2.3	实验结果 . . . . .	2
2.4	心得体会 . . . . .	4

## 1 运行代码

环境要求: rust, gnuplot

运行代码

```
cargo run
```

## 2 q4.2

### 2.1 任务

实现 jacobi, gs, sor 三种迭代解法, 利用它们分别解一个由微分方程导出的差分方程组

### 2.2 解题思路

jacobi, gs, sor 三种迭代解法均按照课本上描述的解法即可; 精度控制由用户传入 `eps` 参数, 每次更新解时检查相对变化是否超过 `eps`, 如果某轮迭代没有一个分量超过即迭代完成。

关于初值, 我采用了两种方式: 全部赋 0 和按照  $y = x$  赋值 (主要是考虑到本题的函数的大致图像情况)。

具体在实现时, 因为本题的矩阵十分稀疏, 我采用了稀疏矩阵的存储方式, 具体来说每行只存储对角线元素 (因为本节的算法都用到了对角线元素) 和其他非零元素及其对应列数。

差分方程组的  $b$  向量没有在课本上描述清楚。经过分析, 矩阵  $A$  和向量  $b$  的大小都应该是  $n-1$ ,  $b$  除了最后一个分量之外都为  $ah^2$ , 最后一个分量因为缺少  $y_n$  项, 值应为  $ah^2 - 1.0 * (\epsilon + h)$  (第一个分量也缺少  $y_0$ , 但是  $y_0 = 0$ , 所以没有影响)

### 2.3 实验结果

输出如下

```
# init: all 0
eps = 1
jacobi: iter = 10858, inf norm dist = 0.003354437050578696
gs: iter = 5442, inf norm dist = 0.003404823354075326
sor(w = 1.1): iter = 4623, inf norm dist = 0.0028298141381259967

eps = 0.1
jacobi: iter = 5098, inf norm dist = 0.009761705106593743
gs: iter = 2578, inf norm dist = 0.009817518298027128
```

```
sor(w = 1.1): iter = 2168, inf norm dist = 0.009631073453557848
```

```
eps = 0.01
```

```
jacobi: iter = 540, inf norm dist = 0.06608628042120523
```

```
gs: iter = 320, inf norm dist = 0.0660841108742467
```

```
sor(w = 1.1): iter = 270, inf norm dist = 0.06607949954560188
```

```
eps = 0.0001
```

```
jacobi: iter = 116, inf norm dist = 0.004952583362066498
```

```
gs: iter = 108, inf norm dist = 0.004950747300598746
```

```
sor(w = 1.1): iter = 141, inf norm dist = 0.004950008892115387
```

```
# init: y = x
```

```
eps = 1
```

```
jacobi: iter = 4578, inf norm dist = 0.006589778549244585
```

```
gs: iter = 2991, inf norm dist = 0.0034038771970676396
```

```
sor(w = 1.1): iter = 2613, inf norm dist = 0.002830033772390106
```

```
eps = 0.1
```

```
jacobi: iter = 3546, inf norm dist = 0.010870718610929586
```

```
gs: iter = 2021, inf norm dist = 0.009821361136785833
```

```
sor(w = 1.1): iter = 1710, inf norm dist = 0.009632574673256844
```

```
eps = 0.01
```

```
jacobi: iter = 450, inf norm dist = 0.06611831557737685
```

```
gs: iter = 280, inf norm dist = 0.06608721442076454
```

```
sor(w = 1.1): iter = 238, inf norm dist = 0.06608082591598069
```

```
eps = 0.0001
```

```
jacobi: iter = 110, inf norm dist = 0.0049552795115767045
```

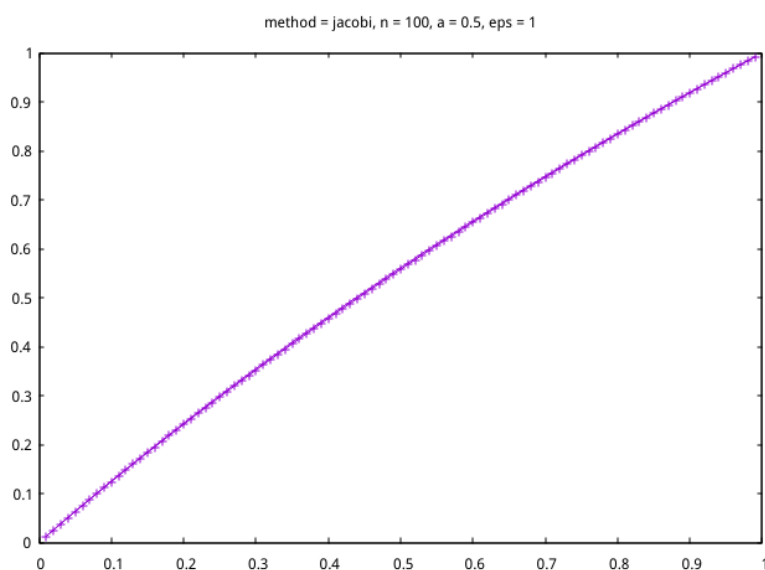
```
gs: iter = 105, inf norm dist = 0.00495114521122475
```

```
sor(w = 1.1): iter = 137, inf norm dist = 0.004950070249054805
```

可见这个简单的初始值选择策略对收敛速度是有一定帮助的，最好情况下可以把收敛速度提高一倍多。

经过一些尝试我发现，对于  $\text{eps} = 0.0001$  的情况，sor 解法取  $w$  尽量接近于 1 会收敛较快；但对于其他情况，sor 解法取  $w$  稍大于 1 会收敛较快。最终代码里就统一取 1.1 了。

我还绘制了对应的图形，可以更加直观地看出误差情况



## 2.4 心得体会

其实这几个迭代方法的收敛速度都比我期望的慢一些，但是如考虑到稀疏矩阵和向量乘法的时间开销很小，那么这几种方法都还算是比较高效的。另外，一些简单的启发式的初始值选择方式可以显著加快收敛速度。