STAT243 Lecture 10.3 Computational Issues

1 Storing matrices

矩阵在内存中的存储方式包括**列优先 (column-major)** 和 **行优先 (row-major)**。

- **列优先**语言（R、Fortran）中，相邻列元素在内存中连续，所以访问同一列或整列更快
- **行优先**语言（Python、C）中，相邻行元素在内存中连续，所以访问同一行或整行更快

某些优化的矩阵分解算法会**将输出结果直接覆盖输入矩阵**以节省内存并提升速度（如某些 in-place factorizations）。

---

2 Algorithms

> **Logic**
>
> 优秀算法能将效率提升几个数量级，现代计算速度的主要提升往往来自算法进步而非硬件本身。

2.1 一个例子: 向量乘法的两种实现

对于向量乘法 $b = Ax$，可使用两种伪代码实现：

2.1.1 按行进行内积 (row-wise)

$$b_i = \sum_{j=1}^{m} a_{ij} x_j$$

```
# initialize b[1:n] = 0
for(i = 1:n){
    for(j = 1:m){
        b_i = b_i + a_{ij} * x_j
    }
}
```

2.1.2 按列进行线性组合 (column-wise)

$$b = \sum_{j=1}^{m} x_j A_{\cdot j}$$

```
# initialize b[1:n] = 0
for(j = 1:m){
    for(i = 1:n){
        b_i = b_i + a_{ij} * x_j
    }
}
```

两种方法的计算量相同，但：

- 方法 1 按行访问 A，适合 **row-major**（Python、C）
- 方法 2 按列访问 A，适合 **column-major**（R、Fortran）

2.2 General computational issues

许多前面讨论过的计算机浮点限制在矩阵计算中同样适用：

- 小数的舍入误差
- catastrophic cancellation
- 分母接近 0 的不稳定情况

优秀的线性代数库（如 LAPACK）会内部处理这些数值稳定性问题。

---

3 Ill-conditioned problems

> **Logic**
>
> 对于矩阵 ill-conditioned problem 的详细论述, 见 DDA3005 Lecture 6

3.1 Basics

若输入的微小扰动导致输出显著变化，则问题是**病态 (ill-conditioned)** 的, 一种衡量是否病态的方式为 **condition number**（条件数）

3.2 例子：矩阵求解对扰动的敏感性

设

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}, \qquad b = (32, 23, 33, 31)^T$$

求解 $Ax = b$ 得到 $x = (1, 1, 1, 1)$

若扰动为

$$\delta b = (0.1, -0.1, 0.1, -0.1)$$

新的解变为

$$x + \delta x = (9.2, -12.6, 4.5, -1.1)$$

可见非常敏感

3.3 条件数的公式

对于欧几里得范数 L_2 (实际上可以选择任何范数), 定义 condition number 为：

$$\text{cond}(A) = ||A||\,||A^{-1}||$$

并且 (仅对于 L_2 norm) 有:

$$\text{cond}(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$$

其中 $\lambda_{\max}$ 和 $\lambda_{\min}$ 为最大和最小特征值的绝对值

并且有近似不等式：

$$\frac{||\delta x||}{||x||} \le \text{cond}(A)\frac{||\delta b||}{||b||}$$

解释：
若 $\text{cond}(A) = 10^8$，我们在双精度 10^{-16} 的基础上会丢失 8 位有效数字

3.4 代码验证（略作整理）

```python
e = np.linalg.eig(A)
evals = e[0]
print(evals)
# [3.02886853e+01 3.85805746e+00 1.01500484e-02 8.43107150e-01]

## relative perturbation in x much bigger than in b
norm2(delta_x) / norm2(x)
# np.float64(8.19847546803699)

norm2(delta_b) / norm2(b)
# np.float64(0.0033319453118976702)

## ratio of relative perturbations
(norm2(delta_x) / norm2(x)) / (norm2(delta_b) / norm2(b))
# np.float64(2460.567236431514)

## ratio of largest and smallest magnitude eigenvalues
## confusingly evals[2] is the smallest, not evals[3]
(evals[0]/evals[2])
# np.float64(2984.092701676269)
```

3.5 Improving conditioning

Logic

统计问题中病态常由自变量共线性引起。改善方式通常是**重新建模**或对变量 **中心化、标准化**。

核心思想：

- **避免输入数值量级差距过大**
- 尽量让变量接近数量级 1

3.5.1 以二次回归为例

原始变量 $t = 1990, \ldots, 2010$ 导致 $X = (1, t, t^2)$ 中：

- 1 的量级 ≈ 1
- t 的量级 ≈ 2000
- t^2 的量级 $\approx 4 \times 10^6$

因此 $X^\top X$ 条件数巨大，OLS 不稳定。

3.6 步骤 1：中心化

定义

$$t_2 = t - 2000$$

3.7 步骤 2：进一步缩放

$$t_3 = \frac{t_2}{10}$$

通过中心化和缩放：

- 特征值范围变窄
- 条件数明显下降
- 回归系数更精确

3.8 结果总结

- 简单的中心化和标准化往往足以改善病态
- 应始终关注变量的量级
- 最好使每列变量都在数量级 $O(1)$