矩阵论及其应用实习题

陈轶洲 MF20330010

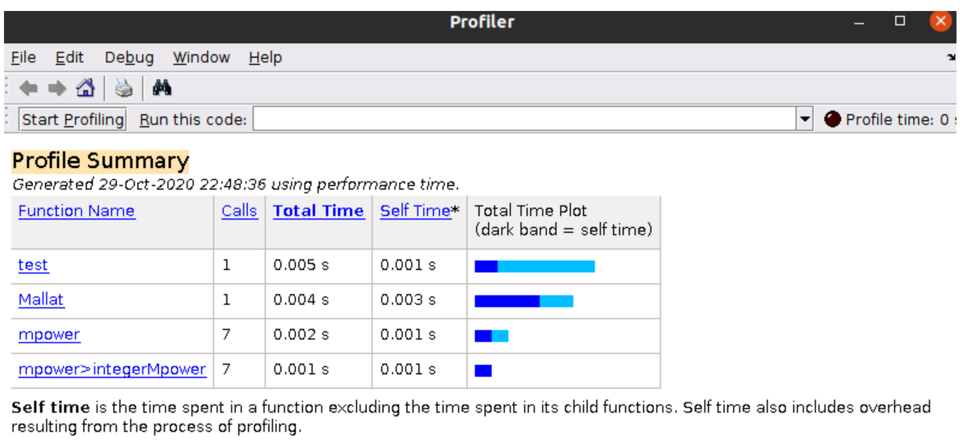
November 12, 2020

1 实验环境

计算机型号：MacBook Air
操作系统：Mac os
编程语言：Matlab

2 CPU 时间

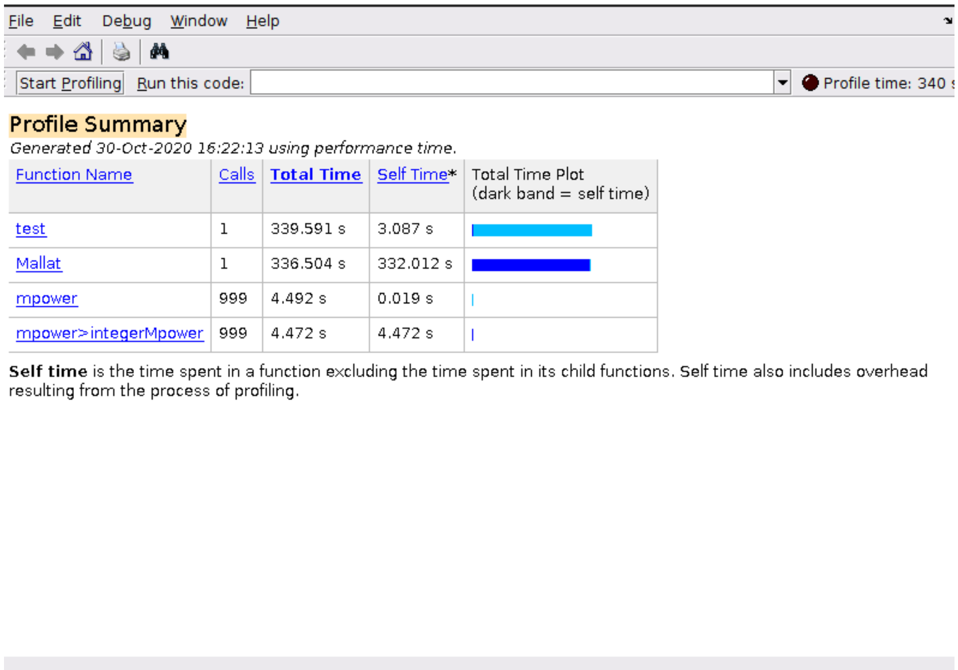
输入第一组数据：$K = 8, M = 50, N = 100$
由于数据规模不大，所以该算法消耗的 CPU 时间只有几毫秒



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
test	1	0.005 s	0.001 s	
Mallat	1	0.004 s	0.003 s	
mpower	7	0.002 s	0.001 s	
mpower>integerMpower	7	0.001 s	0.001 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

输入第二组数据：$K = 1000, M = 15000, N = 23000$
第二组数据规模远远超过第一组，耗时 332.012s



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
test	1	339.591 s	3.087 s	
Mallat	1	336.504 s	332.012 s	
mpower	999	4.492 s	0.019 s	
mpower>integerMpower	999	4.472 s	4.472 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

3 算法说明

3.1 算法步骤

Algorithm 1 正交匹配追踪算法 $Mallat(y, A, K)$

Input: 观测数据向量 $y \in R^{M\times 1}$，字典矩阵 $A \in R^{M\times N}$，稀疏性指标 $K \in N$
Output: 稀疏的信号向量 $x \in R^{N\times 1}$

初始化：$\Omega_0 = \phi, r_0 = y, k = 1$
while $k \leq K$ **do**
    $j_k \in \arg\max_j |(r_{k-1}, a_j)|$
    $\Omega_k = \Omega_{k-1} \cup \{j_k\}$
    $x_k = (A^T_{\Omega_k} A_{\Omega_k})^{-1} A^T_{\Omega_k} y$
    $r_k = y - A_{\Omega_k} x_k$
    $k+ = 1$
$\hat{x}(i) = \begin{cases} x_K(i) & i \in \Omega_K \\ 0 & others \end{cases}$
return $\hat{x}$

3.2 变量说明

使用 Matlab 设计的正交匹配追踪算法中，对变量做如下定义:

1.A 为测度矩阵，规模为 $M \times N$;

2.x 为待重构向量，规模为 $N \times 1$;

3.y 为观测所得向量，规模为 $M \times 1$;

4.K 为稀疏性指标，为正整数;

5.r 为残差向量，规模为 $M \times 1$;

6.At，每轮迭代时被选中的列;

7.x_l 为最小二乘解;

8.x_r 为经过算法重构的向量。

3.3 程序清单

首先在 Mallat.m 中实现正交匹配追踪算法:

```
function [ x ] = Mallat(y,A,K)
          [M,N] = size(A);%字典矩阵的规模
          x = zeros(N, 1);%需要重构的解向量x
          At = zeros(M, K);%存储每轮迭代时A中被选中的列
          x_pos = zeros(1, K);%存储每轮迭代时A中被选中列的序号
          r = y;%残差向量，初始化为y

          for ii = 1:K%迭代K轮
                    product = A'*r;%计算矩阵各列与残差的点积
                    [val, pos] = max(abs(product));%找到与残差最相关的列
                    At(:,ii) = A(:,pos);%将被选中的列存储到At中
                    x_pos(ii) = pos;%记录被选中列的序号
                    A(:,pos) = zeros(M,1);
                    x_l = (At(:,1:ii)'*At(:,1:ii))^(-1)*At(:,1:ii)'*y;%最小二乘解
                    r = y - At(:,1:ii)*x_l;%更新残差
          end

          x(x_pos) = x_l;
end
```

接着在 test.m 中对两组规模的数据进行测试

```
          K=8;
          M=50;
          N=100;
          A=randn(M,N);
          x=randn(N,1);
          kk=randperm(N);
          x(kk(1:N-K))=0;
          y=A*x;
          x_r1 = Mallat(y,A,K);
          stem([x,x_r1]);
```

```
K=1000;
M=15000;
N=23000;
A=randn(M,N);
x=randn(N,1);
kk=randperm(N);
x(kk(1:N-K))=0;
y=A*x;
x_r2 = Mallat(y,A,K);
stem([x,x_r2]);
```

4 运行结果及分析

4.1 $K=8, M=50, N=100$

以第一组数据为例，当 $M=50, N=100$ 时，取稀疏性系数 $K=8$，通过随机生成的系数向量 x 生成观测所得向量，并通过正交匹配追踪算法得到重构向量 x_r，其输出如下：

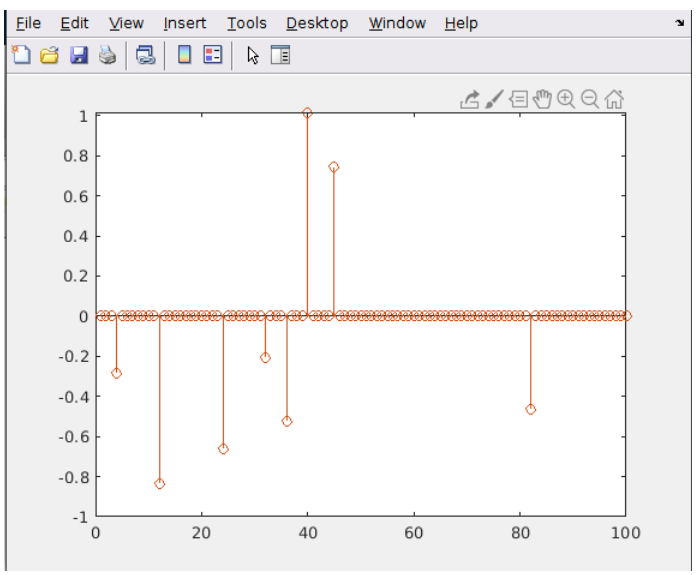
x：

```
Columns 1 through 13

       0    0.4454    1.3161         0         0         0         0         0         0         0         0         0         0

Columns 14 through 26

       0         0         0         0         0         0         0         0         0         0         0    0.8258         0

Columns 27 through 39

       0         0         0         0         0         0         0         0         0         0         0         0         0

Columns 40 through 52

  0.7915         0         0         0         0         0         0         0         0         0         0         0         0

Columns 53 through 65

 -2.6094         0         0   -0.3713         0         0         0   -0.5605         0         0         0         0         0

Columns 66 through 78

       0         0         0   -0.2236         0         0         0         0         0   -1.0343         0         0    0.3266

Columns 79 through 91

       0         0         0         0         0         0         0         0         0         0         0         0         0

Columns 92 through 100

       0         0         0         0         0         0         0         0         0
```

x_r：

```
Columns 1 through 13

     0    0.4454    1.3161         0         0         0         0         0         0         0         0         0         0

Columns 14 through 26

     0         0         0         0         0         0         0         0         0         0         0    0.8258         0

Columns 27 through 39

     0         0         0         0         0         0         0         0         0         0         0         0         0

Columns 40 through 52

0.7915         0         0         0         0         0         0         0         0         0         0         0         0

Columns 53 through 65

-2.6094        0         0   -0.3713         0         0         0   -0.5605         0         0         0         0         0

Columns 66 through 78

     0         0         0   -0.2236         0         0         0         0         0   -1.0343         0         0    0.3266

Columns 79 through 91

     0         0         0         0         0         0         0         0         0         0         0         0         0

Columns 92 through 100

     0         0         0         0         0         0         0         0         0
```

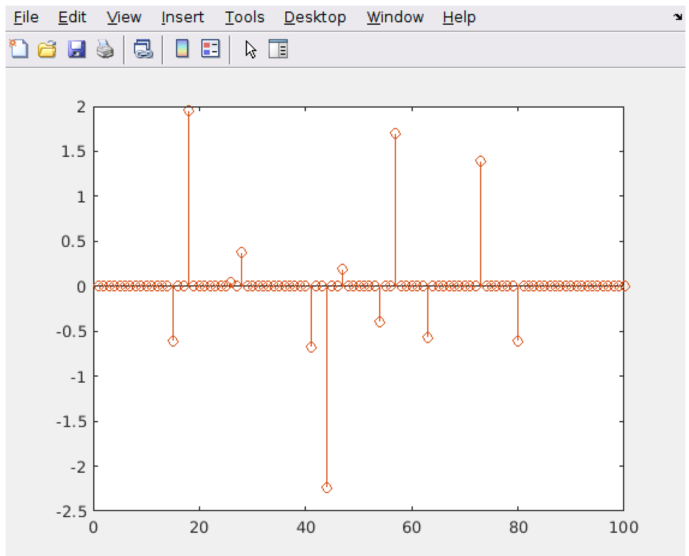
通过 matlab 自带的绘图函数 stem () 绘制两个向量的针状图



结合输出向量和图表可知重构向量与原始向量相同，在该条件下正交匹配追踪算法能根据观测向量完美重构向量

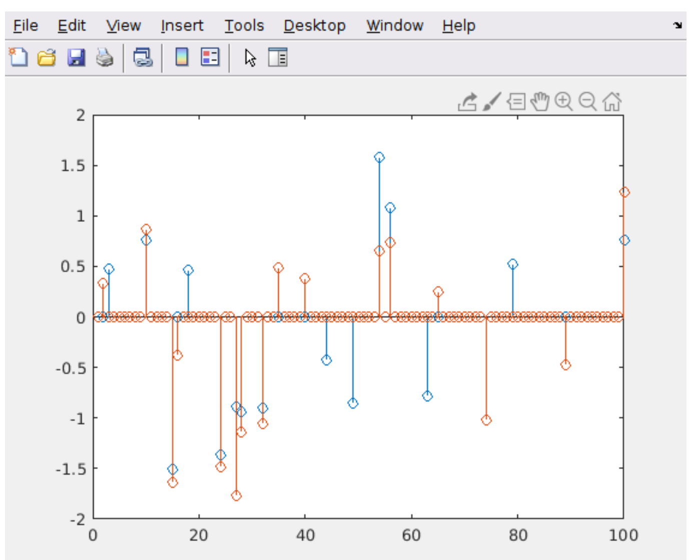
4.2 $K = 12, M = 50, N = 100$

仍使用第一组数据，将稀疏性系数 K 增大到 12，由图可知重构向量与原向量几乎没有误差



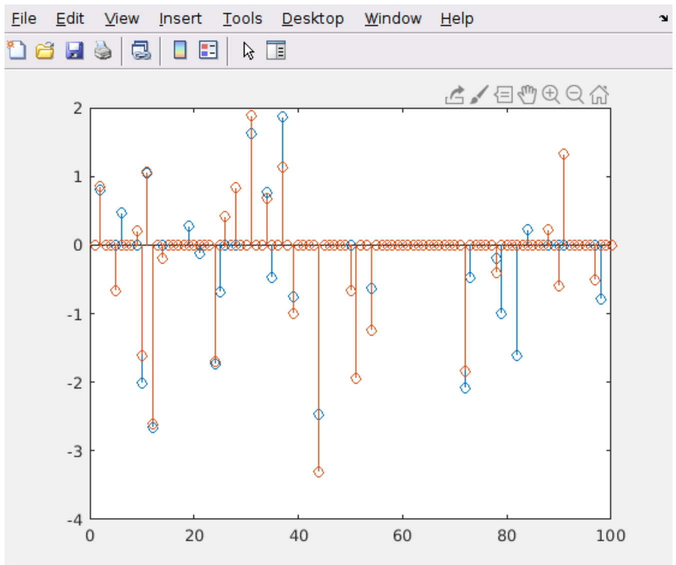
4.3 $K = 16, M = 50, N = 100$

将稀疏性系数 K 继续增大到 16，由图可知此时重构向量与原向量之间出现误差



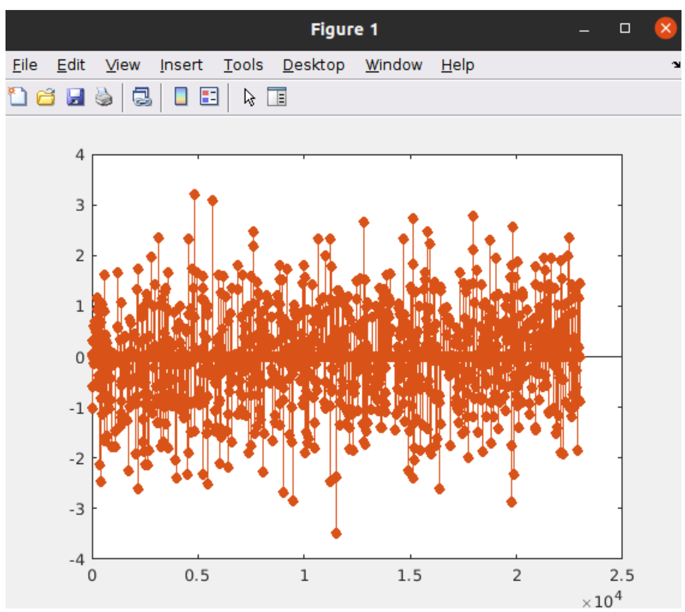
4.4 $K = 24, M = 50, N = 100$

当稀疏性系数 K 继续增大到 24 后，重构向量与原向量之间的误差进一步增大



4.5 $K = 1000, M = 15000, N = 23000$

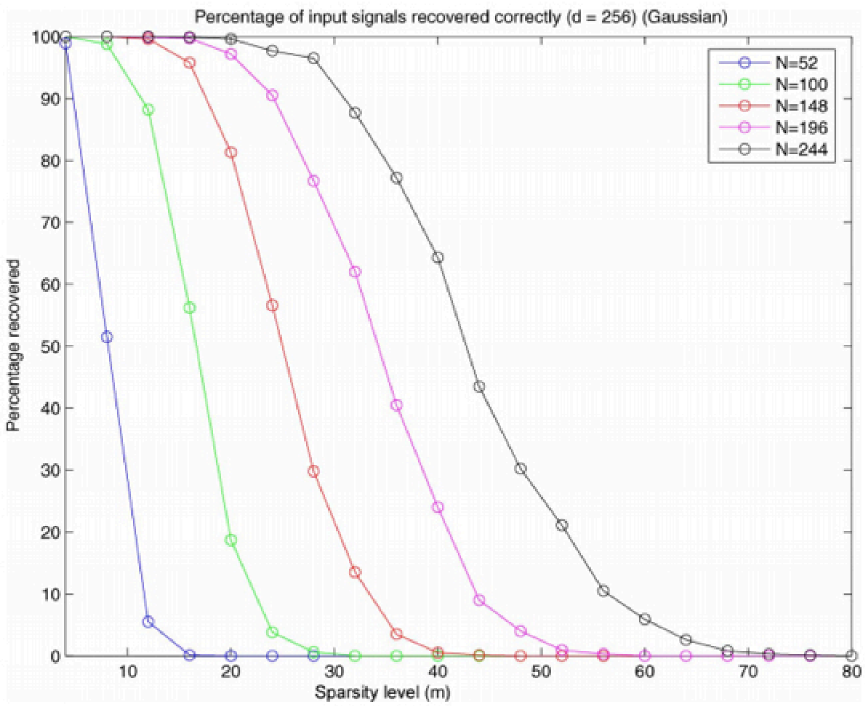
考虑第二组数据，由图可知重构向量与原向量几乎没有误差



4.6 结果分析

1) 对于两组输入数据，分析其 CPU 运行时间, 可以发现：对于小规模输入, 正交匹配追踪算法可以在几毫秒内求出结果; 而对稍大规模数据, 该算法的求解效率依然很可观;

2) 根据第一组实验，在确定 M 和 N 后，K 的取值决定了最后重构向量的表现: $K << N$ 时重构向量与原向量几乎无误差，而随着 K 的增大，重构向量与原向量之间的误差也越来越大，根据文献 [1] 可以看出信号稀疏度 K 与重构成功概率之间的关系：



5 实验总结

1) 通过计算程序运行 CPU 时间可以发现，正交匹配追踪算法的时间复杂度并不高，这是由于它本质上是一个贪心算法，通过稀疏性系数 K 来决定迭代次数，这样就避免了对整个系数矩阵进行求逆等耗时的操作；

2) 在进行算法设计过程中，我考虑利用残差向量寻找到 A 中与其内积绝对值最大的列后，将该列置为零。后来发现这一步是不必要的，因为在之后的轮次中，残差向量与该列正交，所以不会出现重复选择某一列的情况；

3) 对于“增大稀疏性系数 K 后误差变大”这一现象，我认为可以类比主成分分析 (principal component analysis, PCA)，正交匹配算法每轮选择与残差向量内

积绝对值最大的列，可以认为这些列最大程度地保留了向量的信息，而随着 K 的增大，所选择的列越多，这样势必会加入一些与向量相关性不大的列，从而给信号带来噪声，使得重构误差增加。

6 参考文献

[1] Joel A. Tropp and Anna C. Gilbert. Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit[J]. IEEETransactions on Information Theory, VOL. 53, NO. 12, DECEMBER 2007.