问题描述

我们研究了从大量测度中进行相位恢复的问题。特别的，我们希望重建一个复值信号 $\mathbf{x} in C^n$，我们对其进行了一个无相采样：$y_r = |\langle \mathbf{a}_r, \mathbf{x} \rangle|^2, r = 1, \ldots, m$（已知这些相的样本可以生成一个线性系统。）这篇文章提出了一个非凸的相位恢复问题，和它的一个紧致解。

简而言之，这个算法开始于一个基于谱方法生成的初始值，然后通过迭代低复杂度的初始更新规则，提炼一个初始估计。这个过程跟梯度下降是很像的。这个算法的主要贡献是证明了相位恢复问题可以用几乎是最少的随机度量得到。实际上，可以证明一系列的迭代可以以几何速度收敛到真实解。理论上，这个算法可以导出一个接近线性时间的模型实现。

相变

信号模型

随机低通信号

这里，$\mathbf{x}$ 由下式给定：

$$x[t] = \sum_{k=-(M/2-1)}^{M/2} (X_k + iY_k) e^{2\pi i(k-1)(t-1)/n},$$

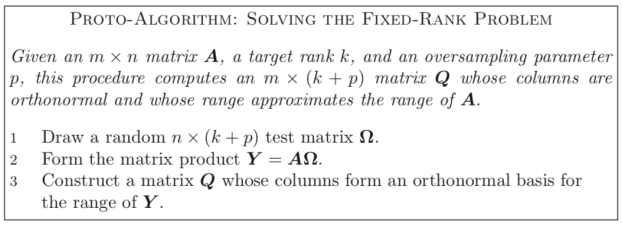
其中 $M = n/8, X_k, Y_k$ 是服从 $\mathcal{N}(0,1)$ 的独立同分布的。

随机高斯信号

这个模型中，$\mathbf{x} \in C^n$ 是一个复随机高斯向量。每一个维度的形式都是 $x[t] = X + iY$, 其中 X 和 Y 都是服从 $\mathcal{N}(0,1)$; this can be expressed as

第二种算法的思路如下图1所示：

我在这次作业中，选择了第一种算法，用 python 语言对其进行实现。



PROTO-ALGORITHM: SOLVING THE FIXED-RANK PROBLEM

Given an $m \times n$ matrix $\boldsymbol{A}$, a target rank k, and an oversampling parameter p, this procedure computes an $m \times (k+p)$ matrix $\boldsymbol{Q}$ whose columns are orthonormal and whose range approximates the range of $\boldsymbol{A}$.

1 Draw a random $n \times (k+p)$ test matrix $\boldsymbol{\Omega}$.
2 Form the matrix product $\boldsymbol{Y} = \boldsymbol{A\Omega}$.
3 Construct a matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of $\boldsymbol{Y}$.

图 1: Proto-Algorithm: Solving the Fixed-Rank Problem

实验

随机生成数据

通过下面的设置取得一个矩阵，并求出其前 $r \in \{5, 10, 15, 20\}$ 大的奇异值，以及其奇异向量。

```
m = 2048
n = 512
p = 20
A = randn(m,p)*randn(p,n)
```

得到的结果如下：

r	exp1: t	exp2: t	exp3: t
5	0.08126497268676758	0.06409597396850586	0.07390093803405762
10	0.03443193435668945	0.025016069412231445	0.03003096580505371
15	0.03320908546447754	0.035800933837890625	0.03665590286254883
20	0.034197092056274414	0.03660106658935547	0.040490150451660156

我对 $r = \{5, 10, 15, 20\}$ 进行了 3 组实验，得到的运行时间如上表所示。可见运行时间较为稳定，可以在线性时间内运行完毕。

rSVD-single-pass 数据集

这个部分里，我复现了https://github.com/WenjianYu/rSVD-single-pass中生成随机数据集的算法，写在 *generatedata.py* 文件中。并测试了对于 $m = n = 2 \times 10^3$. 的矩阵的计算

时间的三次试验结果[1]。

r	exp1-timing	exp2-timing	exp3-timing
50	0.4229423999786377	0.4193081855773926	0.4090569019317627
100	0.4109618663787842	0.5056757926940918	0.4087069034576416
150	0.451876163482666	0.4589710235595703	0.43385791778564453
200	0.5109848976135254	0.5014669895172119	0.4837648868560791
500	0.994359016418457	0.9467556476593018	0.9239499568939209
1000	2.2344188690185547	2.214768886566162	2.300198793411255
1500	4.6173412799835205	4.693850994110107	4.715956926345825

可见，运行时间比较稳定，算法是有效的。

[1]我尝试生成了 2×10^5 阶矩阵的数据集，但是内存超了，所以没有呈现在报告上。特此说明。