Randomized Projection Methods for Linear Systems

摘要：

在医学成像、纠错和传感器网络等应用中，人们需要解决大规模的线性系统，这些系统可能会受到少量但是任意大的破坏。我们考虑解决这样的大规模线性方程组Ax=b，这些方程组由于测量向量b的损坏而不一致。为了解决这类问题，我们开发了一种方法，可以检测出被破坏的条目，从而收敛到原始系统的"真实 "解，并且为该方法提供了分析性的证明，以及在真实和合成系统上的实验证据。

1.实验简介

我们考虑解决的大规模线性方程系统，形式为$Ax=b, A\in R^{m*n}, b\in R^{m}$，其中$m\gg n$。这意味着该系统不一定有一个解决方案，我们可以寻求最小二乘法的解决方案x_{LS}，该解能满足最小化$||Ax-b||$，此处范数$||\cdot||$为二范数。

对于真实向量$b^*, b^*\in R^m$我们无法得到，假设满足$Ax=b^*$的伪解为x^*。由于观测向量b^*损坏，我们只能观测到被损坏的向量b，其中$b=b^*+b_c$。而b_c向量中非零的个数，应该远小于m，我们依然希望能够求解出真实的x^*。

这种具有稀疏性损坏的模型在生活中很常见，从医学成像到传感器网络和纠错码。例如少量的传感器可能发生故障，导致结果向量中出现少量，但是灾难性的损坏。由于报告错误本身可能是任意大的，最小二乘法求解离理想的解很远，但是由于这种灾难性的错误数量较少，我们仍有希望恢复未被破坏的系统的真实解。

我们实验了一些方法，试图识别b中的损坏条目，然后收敛到伪解。这些方法由Random Kaczmarz(RK)算法的多次迭代 “回合”组成。该方法直观的理由是如果只有少数被破坏的方程和许多一致的方程，那么RK的迭代将高概率地选择一致的方程，在伪解附近产生一个迭代，然后最大的残余条目将对应于被破坏的方程。我们给出了一个单轮检测到被破坏方程的概率下限。我们可以运行许多独立的回合，增加检测到这些被破坏的约束的概率。

2.Random Kaczmarz method

RK方法是一种流行的线性方程组的迭代求解器，特别是对于具有极大行数的系统来说，是首选的方法。该方法包括对单个方程的解集进行连续的正交投射。给定系统Ax=b，RK方法通过投影来计算迭代结果$a_i^Tx=b_i$，其中a_i^T是随机选取矩阵A中的某行，b_i是b中的相应的行的值，进行迭代：

$$x_{k+1}=x_k+\frac{b_i-a_i^Tx_k}{||a_i||^2}a_i$$

行a_i以概率$||a_i||^2/||A||_F^2$进行选取。

RK 方法的收敛性证明：

x_0是算法选取的初始值，x是方程$Ax=b$的真实解，先考虑x_k-x

$$\begin{aligned} x_k - x &= x_{k-1} - x - \frac{a_i^T x_{k-1} - b_i}{||a_i||^2} a_i \\ &= I(x_{k-1} - x) - \frac{a_i^T (x_{k-1} - x)}{||a_i||^2} a_i \\ &= (I - \frac{a_i a_i^T}{||a_i||^2})(x_{k-1} - x) \end{aligned}$$

其中$\frac{a_i a_i^T}{||a_i||^2}$是正交投影矩阵，满足

$$(I - \frac{a_i a_i^T}{||a_i||^2})^T (I - \frac{a_i a_i^T}{||a_i||^2}) = (I - \frac{a_i a_i^T}{||a_i||^2})$$

将该性质代入下面的式子：

$$\begin{aligned} ||x_k - x||^2 &= (x_k - x)^T (x_k - x) \\ &= (x_{k-1} - x)^T (I - \frac{a_i a_i^T}{||a_i||^2})(x_{k-1} - x) \end{aligned}$$

接下来计算$||x_k-x||^2$的数学期望值：

$$\mathbb{E}\left[||x_k - x||^2\right] = \mathbb{E}\left[\mathbb{E}\left[||x_k - x||^2 | x_{k-1}\right]\right]$$

$$\begin{aligned} \mathbb{E}\left[||x_k - x||^2 | x_{k-1}\right] &= (x_{k-1} - x)^T [\sum (I - \frac{a_i a_i^T}{||a_i||^2}) \frac{||a_i||^2}{||A||_F^2}](x_{k-1} - x) \\ &= (x_{k-1} - x)^T (I - \frac{\sum a_i a_i^T}{||A||_F^2})(x_{k-1} - x) \\ &= (x_{k-1} - x)^T (I - \frac{A^T A}{||A||_F^2})(x_{k-1} - x) \\ &\leq \rho ||x_{k-1} - x||^2 \end{aligned}$$

其中$\rho = 1 - \frac{\sigma_{min}^2(A)}{||A||_F^2}$，迭代后得到：

$$\mathbb{E}[\mathbb{E}\left[||x_k - x||^2 | x_{k-1}\right]] \leq \rho^k ||x_0 - x||^2$$

显然ρ是在0-1范围中的常数，当k趋近于正无穷，x_k会趋于真实值，RK算法收敛性得证。不论初始值的选取如何，算法都能有效的将解收敛到真实值。

RK方法数值实验：

选取800*41的矩阵A进行实验，矩阵A中元素都是服从高斯分布的处于01区间的值，向量x中元素也是服从高斯分布，范围01的浮点数值。同时进行20组实验，每组实验求解不同的方程，都进行10000次投影过程，记录每次投影的x值与真实值的二范数大小。最后将20组实验得到的数据求平均，并画出图像直观观察，图像如下：

很明显该RK算法求解方程非常有效，平均不到4000次的投影，成功将解收敛到真实值。

但是真实情况下，向量b由少量但是灾难性的误差。为了表示误差，我在上述条件下，在向量b中每隔150行便加入一个随机误差值。该误差值服从100倍的标准高斯分布，将带有误差的向量代入上述RK算法中，设定的参数也与上文相同，我们得到：

非常明显的，我们可以观察到数值的收敛停滞了，收敛到一个较大的误差的最小二乘解上，由于方程组中具有灾难性的误差，所以此时的最小二乘解，距离真实的值距离较远。此时传统的RK方法对具有少量灾难性误差的方程失效了，需要进行改进。接下来我介绍改进的RK算法。

3.Random Kaczmarz with Removal

该算法在RK算法的前提上，补充了删去错误项的步骤，使得要求解的方程的根，能尽可能靠近真实值。直观的来说，RK算法的少量灾难性误差数量，远远少于矩阵的行数值，所以在随机投影的过程中，对解的影响次数非常有限。因此我们可以在迭代一定的数量后，观察此时的解与每行方程所展开的超平面的距离。根据概率收敛原理，在相当数量投影次数后，得到的解应该距离出错的超平面较远，因此我们有机会发现具体出错的行，根据距离二范数的大小排序，去除相距最远的超平面（相当于去除掉最有可能是错误行）。在逐次的去除后，相当于把原来具有误差的方程组进行挑错，去除错误行后的方程，再运用RK算法，就能有效收敛到真实解。

优化RK算法与传统RK算法的对比

数值实验条件与上文相同，同样是20次实验，每组实验10000次投影。在改进的算法中，增加了每400次投影后，测量$|b_i - A_i x|$的值，并得到最大值相应的索引k，在方程中去除第k行，得到新的方程投入后续迭代。数值实验的结果如下：

实验的结果非常成功，二十次随机化的方程，在改进后都有效的去除了错误行，并且收敛到真实值。

为了验证该方法是否真的有效，我们再进行一次更大规模的实验，将原来$800 * 41$的矩阵扩展到$2000 * 101$矩阵，进行20次随机化实验，并且将每次去除方程的行的步数间隔设定为100，200，400，800，观察去除的间隔对收敛速度的影响。

每100次投影去除一次最大误差行，得到下图：

每200次投影去除一次最大误差行，得到下图：

每400次投影去除一次最大误差行，得到下图：

每800次投影去除一次最大误差行，得到下图：

4.实验总结

综上，有去除的RK方法能够克服原方程出错，导致无法收敛到真实值的问题，是传统Rk方法更好的改进。在有去除的RK方法中，相隔某个次数删去出错行的过程中，当次数间隔大于相当的值之后，虽然都能收敛，但是需要更多的运算次数。所以我们的相隔次数应尽可能的少，特别是误差行数远远小于总行数的时候，能够有效提高运算效率。

代码附录（python）

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import plotly


def build_normal_matrix(m, n):
    np.random.seed(200)
    matrix_a = np.random.normal(0, 1, (m, n))
    matrix_a_norm = np.linalg.norm(matrix_a, axis=1, ord=2)
    return matrix_a, np.array(matrix_a_norm).reshape(m, 1)


def classical_K(A, b, x_acc, x0, iter_max):
    x = x0
    res = x - x_acc
    res_norm_record = [np.linalg.norm(res, 2)]

    for k in range(iter_max):
        r = k % m
        a = A[r].reshape((1, n))
        v = b[r] - np.dot(a, x)
        v_norm = np.linalg.norm(a, 2) ** 2
        v = v / v_norm
        x = x + v * a.T
        r_norm2 = np.linalg.norm(x - x_acc, 2)
        res_norm_record.append(float(r_norm2))

    return res_norm_record


def RK(A, b, x_acc, x0, iter_max, A_norm):
    x = x0
    res = x - x_acc
    res_norm_record = [np.linalg.norm(res, 2)]

    A_norm = A_norm ** 2 / np.sum(A_norm ** 2)
    A_csum = np.cumsum(A_norm)

    np.random.seed()
    for k in range(iter_max):
        rand_num = np.random.uniform(0, 1)
        r = np.min(np.where(A_csum > rand_num))
        a = A[r].reshape((1, n))
        v = b[r] - np.dot(a, x)
        v_norm = np.linalg.norm(a, 2) ** 2
        v = v / v_norm
        x = x + v * a.T
```

```python
        r_norm2 = np.linalg.norm(x - x_acc, 2)
        res_norm_record.append(float(r_norm2))

    return res_norm_record


def RK_delete(matrix_a, vector_b, x_round):
    # matrix_a=np.array(matrix_a)
    # vector_b=np.array(vector_b)
    # x_round=np.array(x_round)
    error = abs(vector_b - np.dot(matrix_a, x_round))
    row_del = np.argmax(error)
    matrix_a = np.delete(matrix_a, row_del, axis=0)
    vector_b = np.delete(vector_b, row_del, axis=0)
    return matrix_a, vector_b


def RK_Removal(matrix_a, vector_b, x_acc, x0, iter_max, matrix_a_norm):
    x = x0
    res = x - x_acc
    res_norm_record = [np.linalg.norm(res, 2)]

    matrix_a_norm = matrix_a_norm ** 2 / np.sum(matrix_a_norm ** 2)
    A_csum = np.cumsum(matrix_a_norm)

    np.random.seed()
    for k in range(iter_max):
        rand_num = np.random.uniform(0, 1)
        r = np.min(np.where(A_csum > rand_num))
        a = matrix_a[r].reshape((1, n))
        v = vector_b[r] - np.dot(a, x)
        v_norm = np.linalg.norm(a, 2) ** 2
        v = v / v_norm
        x = x + v * a.T
        r_norm2 = np.linalg.norm(x - x_acc, 2)
        res_norm_record.append(float(r_norm2))
        if k % 800 == 0:
            matrix_a, vector_b = RK_delete(matrix_a, vector_b,
np.array(x).reshape((n, 1)))
            matrix_a_norm = np.linalg.norm(matrix_a, axis=1, ord=2)
            matrix_a_norm = matrix_a_norm ** 2 / np.sum(matrix_a_norm ** 2)
            A_csum = np.cumsum(matrix_a_norm)
    return res_norm_record


def upper_bound(A, x, x_acc, iter_max, A_norm):
    _, sigma, _ = np.linalg.svd(A)
    r0_norm = np.linalg.norm(x - x_acc)
    alpha = 1 - sigma[-1] ** 2 / np.sum(A_norm ** 2)
    k = np.arange(0, iter_max + 1)
    r_norm_bd = r0_norm * alpha ** (k / 2)

    return r_norm_bd


if __name__ == '__main__':

    iter_max = 20000
```

```python
    run_max = 20
    figcont = 0
    cont = np.arange(0, iter_max + 1)

    m = 2000
    n = 101

    A_nm, A_nm_norm = build_normal_matrix(m, n)
    x_true = np.ones((n, 1))
    x = np.random.uniform(-100, 100, (n, 1))
    b_star = np.dot(A_nm, x_true)
    b_c = np.zeros((m, 1), dtype='float')
    for i in range(m):
        if i % 150 == 0:
            b_c[i] = 100 * np.random.random()
    print(b_c)
    b = b_star + b_c
    r_norm_RK = []
    r_norm_RKR = []
    x_experiment1 = x.copy()
    x_experiment2 = x.copy()

    for i in range(run_max):
        r_norm_RK.append(RK(A_nm, b, x_true, x_experiment1, iter_max, A_nm_norm))
        if i % 5 == 0:
            print(str(i))
    r_norm_RK = np.mean(np.array(r_norm_RK), axis=0)

    for i in range(run_max):
        r_norm_RKR.append(RK_Removal(A_nm, b, x_true, x_experiment2, iter_max, A_nm_norm))
        if i % 5 == 0:
            print(str(i))
    r_norm_RKR = np.mean(np.array(r_norm_RKR), axis=0)

    # Plot Figure
    plt.figure(figcont, figsize=(9, 7))
    plt.semilogy(cont, r_norm_RK,
                 linewidth='4', linestyle='-',
                 color='r', label="Random Kaczmarz")
    plt.semilogy(cont, r_norm_RKR,
                 linewidth='4', linestyle=':',
                 color='b', label="RK with Removal")

    ax = plt.gca()
    ax.spines['left'].set_linewidth(1.5)
    ax.spines['bottom'].set_linewidth(1.5)
    ax.spines['top'].set_linewidth(1.5)
    ax.spines['right'].set_linewidth(1.5)

    plt.xticks(np.arange(0, 10) * 2000)
    plt.yticks(np.logspace(-16, 4, num=5))
    plt.xlim(0, iter_max)
    plt.ylim(1e-16, 1e+4)
    legend = plt.legend(fontsize=17)
    plt.ylabel(r"$E||x_{k}-x||_2 $", fontsize="22")
    plt.xlabel("Number of Projection" + r"$k$", fontsize="22")
```

```python
    plt.tick_params(labelsize=15)
    plt.title("Gaussian " + str(m) + " by " + str(n), fontsize=24)
    ax.xaxis.grid(True, which='major')
    ax.yaxis.grid(True, which='major')
    plt.savefig("bigsize800.jpg")
    plt.show()

```