Numerical Analyze: homework 01

Due on Tuesday, Febrary 28, 2017

102061149 Fu-En Wang

1 Introduction

When we are given an n x n squre matrix A, then an n x n orthogonal matrix G can also be found by Gram-Schmidt process.

1.1 Gram-Schmidt process

When

$$A = \begin{bmatrix} A_1 & A_2 & ..... & A_n \end{bmatrix} \tag{1}$$

and

$$G = \begin{bmatrix} G_1 & G_2 & ..... & G_n \end{bmatrix} \tag{2}$$

Each A_i and G_i is a column of squre matrix A and G, respectively. And the algorithm of **Gram-Schmidt process** is as the following:

$$G_1 = A_1 \tag{3}$$

$$G_k = A_k - \sum_{i=1}^{k-1} \frac{(A_k^T G_i) G_i}{G_i^T G_i} \tag{4}$$

1.2 Sigma calculation

Because G is an orthogonal matrix, so $M = G^T G$ will be a diagonal matrix; in other words, each non-diagonal element

$$M[i][j], i \neq j \tag{5}$$

has to be zero. To prove this property, we calculate the error of non-diagonal elements by Sigma:

$$\sigma = \sqrt{\sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} d_{i,j}^2} \tag{6}$$

Theorectically, σ should be very closed to zeros.

In this homework, we will implement three algorithms to compare their runtime.

2 C++ Implementation

2.1 Algorithm-1

```
MAT A_t = A.transpose();
MAT G(n, n);
G[0] = A_t[0];
for(int k=1; k<n; k++){
    G[k] = 0;
    for(int i=0; i<k; i++){
        G[k] += (A[k] * G[i]).sum() * G[i] / (G[i] * G[i]).sum();
    }
    G[k] = A_t[k] - G[k];
}
G = G.T();
```

This is the most straightforward method for **Gram-Schmidt process**. In the innermost for-loop, we calculate $\sum_{i=1}^{k-1} \frac{(A_k^T G_i)G_i}{G_i^T G_i}$ first and subtracted by $A[k]$, and then replace $G[k]$ by the final vector.

2.2 Algorithm-2

```
MAT A_t = A.transpose();
MAT G(n, n);
G[0] = A_t[0];
for(int k=1; k<n; k++){
    G[k] = A_t[k];
    for(int i=0; i<k; i++){
        G[k] -= (G[k] * G[i]).sum() * G[i] / (G[i] * G[i]).sum();
    }
}
G = G.T();
```

Algo2 simplify algo1 by modifying three terms and removing $G[k] = A[k]$ - $G[k]$. Because there is one more subtraction in algo1 than algo2, so algo2 should be faster than algo1.

2.3 Algorithm-3

```
MAT A_t = A.transpose();
MAT G(n, n);
G[0] = A_t[0];
for(int k=1; k<n; k++){
    G[k] = A_t[k];
    for(int i=0; i<k; i++){
        G[k] -= (G[k] * G[i]).sum() / (G[i] * G[i]).sum() * G[i];
    }
}
G = G.T();
```

Compared to algo2, algo3 only move the multiplication of $(G_k^T G_i)$ and G_i to the last position. By such manipulation, two problems show up:

1. Will the result change? (observed by $\boldsymbol{\sigma}$)

2. Will the speed change? (observed by **RUNTIME**)

We will discuss these two questions in next section.

3 Discussion

3.1 Performance Evaluation

In this project, we focus on two numbers for performance:

1. $\boldsymbol{\sigma}$ (accuracy)

2. **RUNTIME** (speed)

σ can be regarded as the loss or error of our system, so we must make it small as possible as we can. As for RUNTIME, there are two way to deal with it.

1. Total runtime (total time of execution)

2. Algorithm runtime (total time of algorithm part)

Because total runtime consists of the time that program parses the **mX.dat**; when the matrix dimension is large, it will consume much more time. But what we really care about is the time our algorithms consume, so I also track the algorithm time in each experiment.

Table 1: Add caption

A	B
1	4
2	5
3	6