AN EXPERIMENTAL STUDY OF THE TRIANGLE ALGORITHM WITH EMPHASIS ON SOLVING A LINEAR SYSTEM

BY HAO SHEN

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Computer Science

Written under the direction of

Dr. Bahman Kalantari

and approved by

_____

_____

_____

New Brunswick, New Jersey

January, 2015

ABSTRACT OF THE THESIS

An Experimental Study of the Triangle Algorithm with Emphasis on Solving a Linear System

by Hao Shen

Thesis Director: Dr. Bahman Kalantari

The triangle algorithm, Kalantari [7], is designed to solve the convex hull membership problem. It can also solve LP, and as shown in Kalantari [6] solve a square linear system. In this thesis we carry out some experimentation with the triangle algorithm both for solving convex hull problem and a linear system, however, with more emphasis on the latter problem.

We first tested the triangle algorithm on the convex hull problem and made comparison with the Frank-Wolfe algorithm. The triangle algorithm outperformed the Frank-Wolfe for large scale problems, up to 10,000 points in dimensions up to 500. The triangle algorithm takes fewer iterations than the Frank-Wolfe algorithm.

For linear systems, we implemented the incremental version of the triangle algorithm in [6] and made some comparison with SOR and Gauss-Seidel methods for

systems of dimension up to 1000. The triangle algorithm is more efficient than these algorithms taking fewer iterations.

We also tested the triangle algorithm for solving the PageRank matrix by converting it into a convex hull membership problem. We solved the problem in dimensions ranging from 200 to 2200. We made comparisons with the power method. The triangle algorithm took less iterations to reach the same accuracy.

Additionally, we tested a large scale PageRank matrix problem of size of 281,903 due to S. Kamvar. Surprisingly, the triangle algorithm took only 1 iteration to obtain a solution with the accuracy of 10^{-10}.

Acknowledgements

Foremost, I would like to express my deep gratitude to my thesis advisor Professor Bahman Kalantari for the continuous support during my Master study and research. His patience, motivation, enthusiasm and immense knowledge really helped me during the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master study.

My sincere thanks also goes to my friends Thomas Gibson and Meng Li for sharing their codes and technical experiences with the triangle algorithm.

Finally, thanks Prof. Richter and Prof. Saraf for their comments during my defense.

Dedication

In my life, several persons have always been there during those difficult and tiring times. I would like to dedicate this thesis and everything I do to my parents and my wife, Ping Wang. I would not be who I am today without the love and support of my family.

Table of Contents

Abstract ii

Acknowledgements iv

Dedication v

List of Tables viii

List of Figures ix

1. Introduction 1

2. Review of The Triangle Algorithm 4

2.1. Basic Concepts of Triangle Algorithm 4

3. Numerical Experiment for Solving The Convex Hull Problem and Comparison with The Frank-Wolfe Algorithm 9

4. Solving Ax=b Via The Triangle Algorithm 12

4.1. Solving A Linear System with Nonnegative Solution 12

4.2. Solving A General Linear System 14

4.3. The Incremental Triangle Algorithm 15

5. Numerical Experiment for Solving $Ax = b$ Via The Triangle Algorithm and Comparison with SOR and Gauss-Seidel 18

5.1. Review of Gauss-Seidel 18

5.2. Review of SOR method 19

5.3. Numerical Comparison with Triangle Algorithm, Gauss-Seidel and SOR 20

5.4. More Numerical Experiment with t_{min} 23

6. Solving PageRank Problem via Triangle Algorithm 29

7. Numerical Comparison with Power Method 32

8. Conclusion and Future Work 36

9. Appendix for Matlab Codes 38

9.1. Triangle Algorithm for LP Feasibility 38

9.2. Incremental Triangle Algorithm 45

9.3. Power Method 52

9.4. Generated Test Data for LP Feasibility 53

9.5. Generated Data for Page Rank Problem 55

9.6. Get Optimal T 56

9.7. Frank Wolf Algorithm 59

9.8. SOR and Gauss Seidel 61

References 62

List of Tables

5.1. Iterations with Increasing t value 20

5.2. Ex.1 Triangle VS SOR VS Gauss Seidel 21

5.3. Time Comparison: Triangle vs Optimum SOR vs Gauss-Seidel 22

5.4. Iterations Comparison: Triangle vs Optimum SOR vs Gauss-Seidel . 23

5.5. Iterations with Increasing t value 25

5.6. Iterations with Increasing t value 26

7.1. Triangle Algorithm VS Power Method 33

7.2. Triangle Algorithm's performance for practical data method 34

List of Figures

2.1. $W_p = \phi$ (left) and $W_p \neq \phi$ (gray area)[7] 5

2.2. Depiction of gaps $\delta = \|p - p'\|$, $\delta' = \|p'' - p\|$ 7

3.1. n=10,000, Iterations Comparison Triangle Algorithm VS Frank-Wolfe 10

3.2. n=10,000, Time Comparison of Triangle Algorithm VS Frank-Wolfe . 10

5.1. Distribution of iterations with different t values 27

Chapter 1
Introduction

Given a set $S = \{v_1, ..., v_n\} \subset \Re^m$ and a point $p \in \Re^m$, testing if $p \in conv(S)$, the convex hull of S, is called the *convex hull decision problem* (or the *convex hull membership problem*). It is a fundamental problem in computational geometry and linear programming. One application is the *irredundancy problem*, that compute all the vertices of $conv(S)$, [5]. We can also consider the convex hull membership problems as a special case of linear programming (LP) feasibility problem:

$$\begin{aligned} Ax &= b, \\ e^T x &= 1, \\ x &\geq 0. \end{aligned} \tag{1.1}$$

where the column vectors of A are $\{v_1, ...v_n\}$ and e is the vector of ones. Conversely, we can convert an LP feasibility to a convex hull decision problem.

To solve the convex hull decision problem, a simple and natural geometric method, *the triangle algorithm*, has been proposed in [7]. In each iteration of the triangle algorithm, we have a current approximation to p, a point $p' \in conv(S)$. Using this, we select a *pivot* point $v_j \in S$, i.e. such that $d(p', v_j) \geq d(p, v_j)$, where $d(.,.)$ is the Euclidean distance. Then approache p greedily along the direction from current iterate point to the pivot. If a pivot does not exists we can deduce that $p \notin conv(S)$.

In next Chapter we describe the triangle algorithm in more detail.

In the experimental section in solving convex hull problem, we implemented the triangle algorithm and made comparison of its performance with the Frank-Wolfe algorithm, a gradient decent method when applied to a convex quadratic function over a simplex. According to our computational results, the triangle algorithm has better performance than the Frank-Wolfe algorithm when the number of points n is much larger than the dimension m [9].

Next, we implemented the triangle algorithm for solving a linear system $Ax = b$ with an invertible square matrix. Solving general linear system of equations is undoubtedly one of the most practical problems in numerous aspects of scientific computing. Iterative methods are preferred for large systems, such as Gauss-Seidel and Successive Overrelaxion(SOR) method. In our studies, we implemented the incremental version of the triangle algorithm, Kalantari [6], to solve $Ax = b$, where A is an invertible matrix, and it is not known that the solution $x = A^{-1}b$ is nonnegative. The justification for doing so lies in the fact that solving the linear system is equivalent to solving $A(x + te) = b + tAe$, where t is any nonnegative scalar and e is the vector of ones. Equivalently, the latter system can be written as $Ax(t) = b + tAe = b(t)$, and for appropriate t the solution $x(t)$ is nonnegative. Then for such value t we have, $0 \in conv(\{a_1, ..., a_n, -b(t)\})$, where a_i is the i-th column of A. In our experimental work, we implemented the triangle algorithm and solved different linear systems with dimensions up to 1000 and made comparisons with the Gauss-Seidel and SOR. We showed that the incremental triangle algorithm can solve the general linear system $Ax = b$ and outperform the two classical algorithms with less iterations.

Finally, we implemented the triangle algorithm to solve a very popular problem,

PageRank problem. A regular PageRank problem can be represented as solving:

$$\begin{aligned} Ax &= x, \\ e^T x &= 1, \\ x &\geq 0. \end{aligned} \tag{1.2}$$

We converted the PageRank problem to the following system:

$$\begin{aligned} (A - I)x &= 0, \\ e^T x &= 1, \\ x &\geq 0. \end{aligned} \tag{1.3}$$

Set $A' = A - I$ and $b = 0$, rewrote the above equations as

$$\begin{aligned} A'x &= b, \\ e^T x &= 1, \\ x &\geq 0. \end{aligned} \tag{1.4}$$

In the experimental work, we implemented the algorithm by generating some random sparse matrix with dimension ranging from 200 to 2200. Moreover, we implemented the triangle algorithm to solve a practical web data due to S. Kamvar with a large matrix A that is of size $281,903 * 281,903$. We compared the result with classic Power method, a popular method for solving the PageRank problem. From the results, we can conclude that the triangle algorithm takes fewer iterations than the Power method. We believe that the triangle algorithm offers another option to solve the PageRank problem.

In next chapter we will review the basic concepts of triangle algorithm from Kalantari [7].

Chapter 2
Review of The Triangle Algorithm

2.1 Basic Concepts of Triangle Algorithm

In this section, we make a basic introduction to the triangle algorithm. The triangle algorithm is a simple iterative algorithm due to Kalantari [7], for testing if the convex of hull of a set of points in a Euclidean space contains a particular point. We let $\|\cdot\|$ denote the Euclidean norm. We begin the theories as follows:

Theorem 2.1.1 (Distance Duality[7]). *Let $S = \{v_1, ..., v_n\} \subset \Re^m$, $p \in \Re^m$.*
(i):$p \in conv(S)$ if and only if given any $p' \in conv(S)$, there exists v_j such that $\|p' - v_j\| \geqslant \|p - v_j\|$. Such v_j is called a p-pivot (or simply pivot).
(ii): $p \notin conv(S)$ if and only if there exists $p' \in conv(S)$ such that $\|p'-v_i\| < \|p-v_i\|$, $\forall i$. Such p' is called a witness.

Each witness certifies that $p \notin conv(S)$. In [7], it is shown that each witness actually induces a separating hyperplane. The set W_p of all such witnesses is the intersection of $conv(S)$ and open balls B_i of radius $\|p - v_i\|$ at v_i, where $i = 1, ..., n$, and forms a convex subset of $conv(S)$.

The following result shows that when we have a witness, we have an approximation to the distance of p of the convex hull of S:

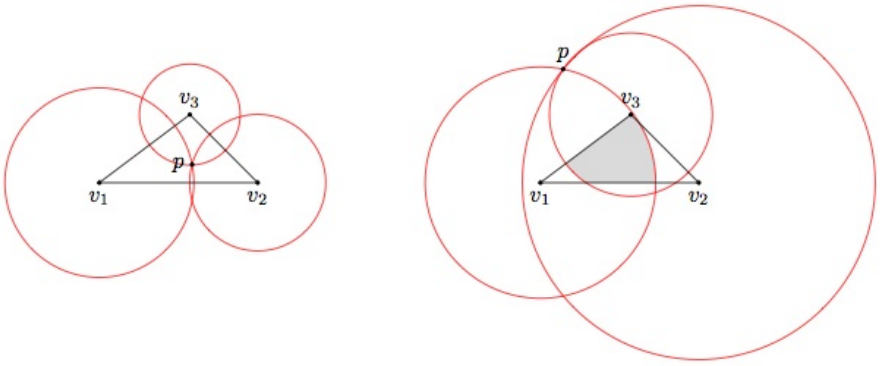

Figure 2.1: $W_p = \phi$ (left) and $W_p \neq \phi$ (gray area)[7]

Theorem 2.1.2. *Suppose* $p \notin conv(S)$. *Let* $\Delta = min\{\|p-x\| : x \in conv(S)\}$. *Then any* $p' \in W_p$ *satisfies* $0.5\|p-p'\| \leq \Delta \leq \|p-p'\|$.

Definition 1. *Given* $\epsilon \in (0,1)$, $p' \in conv(S)$ *is an* $\epsilon-$*approximate solution if* $\|p'-p\| \leq \epsilon R$, *where* $R = max\{\|p-v_1\|, ..., \|p-v_n\|\}$.

Using the characterization theorem (Theorem 2.1.1), [7] described a simple algorithm, called the triangle algorithm. Given a desired tolerance $\epsilon \in (0,1)$, and a current iterate $p' \in conv(S)$, in each iteration the triangle algorithm searches for a triangle $\triangle pp'v_j$ where $v_j \in S$ satisfies $\|p'-v_j\| \geq \|p-v_j\|$. Given that such triangle exists, the algorithm uses v_j as a *pivot* to "pull" the current iterate p' closer to p to get a new iterate $p'' \in conv(S)$. If no such a triangle exists by Theorem 2.1.1, p' is a witness certifying that p is not in $conv(S)$. The triangle algorithm consists of iterating two steps:

Triangle Algorithm ($S = \{v_1, ..., v_n\}$, p, $\epsilon \in (0,1)$)

Step 0. (Initialization) Let $p' = v = argmin\{\|p - v_i\| : v_i \in S\}$.

Step 1. If $\|p - p'\| \leq \epsilon\|p - v\|$, output p' as $\epsilon-$approximate solution, stop. Otherwise, if there exists a pivot v_j, replace v with v_j. If no pivot exists, then output p' as a witness, stop.

Step 2. Compute the *step-size*

$$\alpha_* = \frac{(p - p')^T(v_j - p')}{\|v_j - p'\|^2} \tag{2.1}$$

Given the current iterate $p' = \Sigma_{i=1}^n \alpha_i v_i$, set the new iterate as:

$$p'' = (1 - \alpha_*)p' + \alpha_* v_j = \sum_{i=1}^{n} \alpha'_i v_i, \tag{2.2}$$

$$\begin{aligned} \alpha'_j &= (1 - \alpha_*)\alpha_j + \alpha_*, \\ \alpha'_i &= (1 - \alpha_*)\alpha_i, \forall i \neq j \end{aligned} \tag{2.3}$$

Replace p' with p'', α_i with α'_i, for all $i = 1, ..., n$. Go to Step 1.

It can be shown that the point p'' in Step 2 is the closest point to p on the line $p'v_j$. Since p'' is a convex combination of p' and v_j it will remain in $conv(S)$. The algorithm replaces p' with p'' nd repeats the above iterative step. Note that a p-pivot v_j may or may not be a vertex of $conv(S)$. Figure 2.2 demonstrates two consecutive iterates.

Next, we state the estimate of complexity of the triangle algorithm.

Theorem 2.1.3. *(i) Suppose $p \in conv(S)$. Given $\epsilon \geq 0$, $p_0 \in conv(S)$, with $\delta_0 = \|p - p_0\| \leq R_0 = min\{\|p - v_i\| : i = 1, ..., n\}$. The number of iterations K_ϵ to compute*

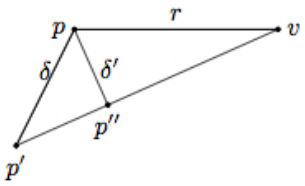

Figure 2.2: Depiction of gaps $\delta = \|p - p'\|$, $\delta' = \|p'' - p\|$

a point p_ϵ *in* $conv(S)$ *so that* $\|p - p_\epsilon\| \leq \epsilon\|p - v_i\|$, *for some* $v_i \in S$ *satisfies*

$$K_\epsilon = O(\frac{1}{\epsilon^2}) \tag{2.4}$$

(ii) Suppose $p \notin conv(S)$. *Let* $\Delta = min\{\|x - p\| : x \in conv(S)\}$. *The number of iterations* K_Δ *to compute a p-witness, a point* p_Δ *in* $conv(S)$ *so that* $\|p_\Delta - v_i\| \leq \|p - v_i\|$ *for all* $v_i \in S$, *satisfies*

$$K_\Delta = O(\frac{R^2}{\Delta^2}), \;\; R = max\{\|p - v_i\|, i = 1, \ldots, n\}. \tag{2.5}$$

Note that each iteration of triangle algorithm takes $O(mn)$ arithmetic operations. Next let us give a stronger version of the distance duality. First, here is the definition of strict pivot.

Definition 2. *Given* $p' \in conv(S)$, *we say* $v_j \in S$ *is a strict pivot relative to* p *(or strict p-pivot, or simply strict pivot) if* $\angle p'pv_j \geq \pi/2$

Theorem 2.1.4 (Strict Distance Duality[7])**.** *Assume* $p \notin S$, *Then* $p \in conv(S)$ *if and only if for each* $p' \in conv(S)$ *there exists strict p-pivot,* v_j.

Then, an alternative complexity bound can be stated.

Theorem 2.1.5 ([7]). *Assume $p \in conv^{\circ}(S)$, the relative interior of $conv(S)$. Let ρ_p be the supreme of radii of the balls centered at p in this relative interior. Suppose the triangle algorithm uses a strict pivot in each iteration. Given $\epsilon \in (0,1)$, the number of iterations of the algorithm to compute $p\epsilon \in conv(S)$ such that $\|p - p_\epsilon\| \leq \epsilon R$. $R = max\{\|p - v_i\|, i = 1, ..., n\}$, satisfies*

$$O(\frac{R^2}{\rho_p^2} \ln \frac{1}{\epsilon}).$$

In next chapter, we will make a numerical experiment comparison in solving the convex hull problem with triangle algorithm and Frank-Wolfe algorithm.

Chapter 3

Numerical Experiment for Solving The Convex Hull Problem and Comparison with The Frank-Wolfe Algorithm

We will implement the triangle algorithm for the convex hull decision problem and compare it with the classical algorithm, the Frank-Wolfe algorithm. We let A to be the matrix whose column vectors are $\{v_1, ..., v_n\}$, e is the vector of ones, and p the query point. The Frank-Wolfe algorithm for the convex hull decision problem is quite straightforward, it is based on the following quadratic programming, see [3]:

$$min\{f(x) = (Ax - p)^T(Ax - p) \ ; \ e^T x = 1, \ x \geq 0\} \tag{3.1}$$

In each iteration, Frank-Wolfe uses $O(mn)$ operations to compute the gradient of f and chooses a direction to progress, using partial derivatives, see [1]. In contrast, the triangle algorithm has many options. In one option it goes through all the column vectors. It moves to next iterations as long as it finds a "perfect" v_j. We do this by checking the angle $\angle pp_k v_j$ and select the smallest one at a current iterate p_k. Therefore, triangle algorithm usually spends much less iteration steps than the Frank-Wolfe algorithm.

To compare the above two algorithms, we implemented them in Matlab. In the experiment, we randomly and uniformly generated n points to form S, a query point

p, all in m-dimensional unit ball, giving a very dense matrix. We set $\epsilon = 10^{-4}$

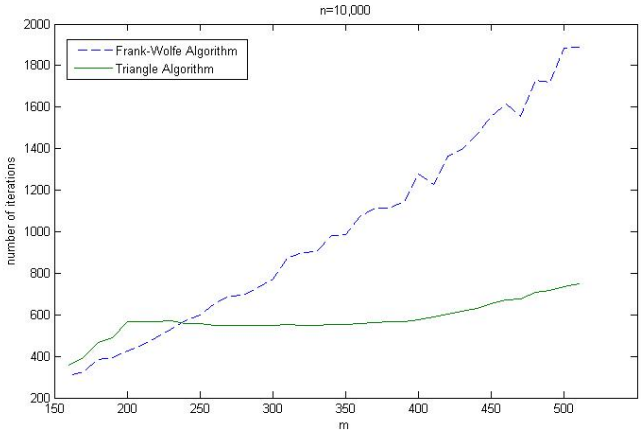

Figure 3.1: n=10,000, Iterations Comparison Triangle Algorithm VS Frank-Wolfe

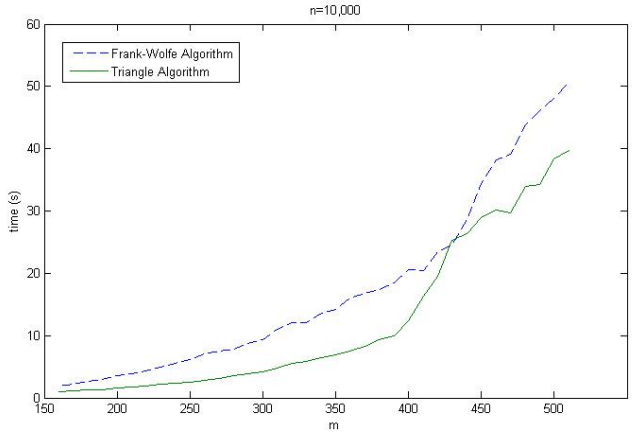

Figure 3.2: n=10,000, Time Comparison of Triangle Algorithm VS Frank-Wolfe

As we can see in Figure 3.1 and Figure 3.2 , when dimension m grows, the iteration steps of Frank-Wolfe algorithm increase significant, while triangle algorithm performs very well with only a slight increase in the iterations steps. This can be explained by

the fact that the triangle algorithm always find the "perfect" pivot, which minimize the $\angle pp_k v$ to make the iteration steps smallest. And since it need visit all the column of vectors, the running time does not performed much better than Frank-Wolfe with the grow of dimension m. In [9], they introduced another improvement on reducing the running time of each iteration that triangle algorithm moves to the next iterations as long as it finds a "good" v_j. The triangle algorithm does not need to visit all the n points and thus spends less time than Frank-Wolfe algorithm in each iteration. And the triangle algorithm has a better chance for finding a "good" pivot v_j to approach p efficiently when n increases, see [9]

In summary, the triangle algorithm does well for large scale data set, especially when the number of points n is much larger than m. And based on our experiments it is more efficient than Frank-Wolfe algorithm, both in running time the number of iterations.

Chapter 4
Solving Ax=b Via The Triangle Algorithm

In this chapter, we consider solving a linear system via the triangle algorithm as described in Kalantari [6]. Consider solving $Ax = b$ with A an invertible $n \times n$ real matrix. We let a_i denote the i-th column of A. Given a matrix B write $conv(B)$ for the convex hull of the columns of B.

Definition 3. *We say x_0 is an ϵ-approximate solution of $Ax = b$ if*

$$\|Ax_0 - b\| \leq \epsilon\rho, \ \ \rho = max\{\|a_1\|, ..., \|a_n\|, \|b\|\} \tag{4.1}$$

4.1 Solving A Linear System with Nonnegative Solution

First, suppose that $x = A^{-1}b \geq 0$. We show how to solve this as a convex hull problem. Next, we solve the general case to relax this condition. Since A is invertible, $Ax = 0$ has only trivial solution. In particular, $0 \notin conv(A)$.

In [7] Kalantari described an application of the triangle algorithm in computing an approximate solution of the linear programming feasibility problem. In Kalantari [6] he describes application of the triangle algorithm in solving a linear system.

Here we explain Kalantari's approach to solve $Ax = b$ via the triangle algorithm to compute for any $\epsilon \in (0, 1)$, an $\epsilon-$approximate solution.

Proposition 4.1.1. *$x = A^{-1}b \geq 0$ if and only if $0 \in conv([A, -b])$.*

It follows that solving $Ax = b$ approximately is equivalent to finding an approximation to 0 in the set $conv(\{a_1, ..., a_n, -b\})$

Theorem 4.1.2 (Sensitivity Theorem [7])**.** *Let $\Delta_0 = min\{\|AX\| : \sum_{i=1}^{n} x_i = 1, x_i \geq 0\}$, and $\rho = max\{\|a_1\|, ..., \|a_n\|, \|b\|\}$. Let Δ_0' be any number such that $0 < \Delta_0' \leq \Delta_0$. Suppose $\epsilon \in (0, 1)$ satisfies $\epsilon \leq \Delta_0'/2\rho$ and suppose we have computed*

$$p' = A\alpha - \alpha_{n+1}b \in conv([A, -b]), \ \|p'\| \leq \epsilon\rho. \tag{4.2}$$

Let $x_0 - \alpha/\alpha_{n+1}$. Then, $x_0 \geq 0$, and if $\epsilon' = 2(1 + \|b\|/\Delta_0')\epsilon$, we have

$$\|Ax_0 - b\| \leq \epsilon'\rho \tag{4.3}$$

We can now describe a Two-Phase algorithm to solve Ax=b as a convex hull problem. Phase 1 of the algorithm attempts to find a witness $p' \in conv(\{a_1, ..., a_n\})$ that proves 0 is not in this convex hull. Any such witness p' gives rise to a lower bound to Δ_0 which in turn can be used in Phase 2 of the algorithm an ϵ-approximate point in $conv([A, -b])$.

Two-Phase Triangle Algorithm$(A, b, \epsilon_0 \in (0, 1))$

Phase 1. Call triangle algorithm $(A, 0, \epsilon)$ to get a witness $p' \in conv(A)$.

Phase 2. Starting with p', call triangle algorithm $([A, -b], 0, \epsilon)$.

Theorem 4.1.3 ([7]). *Given any $\epsilon_0 \in (0,1)$, in order to compute an ϵ_0-approximate solution (i.e. a solution $x_0 \geq 0$ such that $\|Ax_0 - b\| \leq \epsilon_0\rho$) it suffices to set $\Delta_0' = 0.5\|p'\|$, where p' is the witness computed in Phase 1 of the Two-Phase triangle algorithm. Then in Phase 2 of the algorithm it suffices to compute a point $p' \in conv(\{a_1, ..., a_n, -b\})$ so that*

$$\|p'\| \leq \epsilon\rho, \quad \epsilon \leq \frac{\Delta_0'}{2} min\{\frac{1}{\rho}, \frac{\epsilon_0}{(\Delta_0' + \|b\|)}\} \tag{4.4}$$

Then the number of iterations in Phase 2, K_ϵ, each of cost $O(n^2)$ arithmetic operations, satisfies

$$K_\epsilon = O(\frac{1}{\epsilon_0^2}\frac{\rho^2}{\Delta_0'^2}) \tag{4.5}$$

4.2 Solving A General Linear System

In this section we describe Kalantari's method from [6] for solving the general case of solving square $Ax = b$ with A an invertible matrix, where it is not known if the solution $x = A^{-1}b$ is nonnegative. Let $e = (1, ..., 1)^T \in \Re^m$. Let $u = Ae$, then $u \neq 0$ (Since A is invertible). Let

$$\begin{aligned} t_* &= min\{t : A(x - te) = b, x \geq 0\} \\ &= min\{t : Ax = b(t) = b + tu), x \geq 0\} \end{aligned} \tag{4.6}$$

If a value $t \geq t_*$ is know we can apply the Two-Phase triangle algorithm to solve $Ax = b(t)$, $x \geq 0$. See [6] In the complexity analysis we use bounds that also depend upon t. We may restate the complexity result, Theorem 4.1.3, where $\|b\|$ and ρ are replaced with $\|b(t)\|$ and $\rho(t) = max\{\|a_1\|, ..., \|a_n\|, \|b(t)\|\}$, respectively.

Now we may state the following complexity bound.

Theorem 4.2.1. *[6] Given $\epsilon_0 > 0$, any $t \geq 0$, and any lower bound $0 < \Delta_0' \leq \Delta_0$, the Two-Phase triangle algorithm in*

$$O((\frac{\rho(t)}{\epsilon_0 \Delta_0'})^2) = O((\frac{(\rho + t\|u\|)}{\epsilon_0 \Delta_0'})^2)$$

iterations, each of cost $O(n^2)$ arithmetic operations, either determines that $Ax = b + tu$, $x \geq 0$ is infeasible, or computes x_0 satisfying $\|Ax_0 - b\| \leq \epsilon_0 \rho$.

4.3 The Incremental Triangle Algorithm

The *incremental triangle algorithm* works as follows. Assume that for a given $t_0 \geq 0$ (initially set to zero) we have attempted to compute an ϵ-approximate solution for $Ax = b$, i.e. a vector $x_0 \geq 0$ such that

$$\|A(x_0 - t_0 e) - b\| \leq \epsilon \times max\{\|a_1\|, ..., \|a_n\|, \|b\|\}.$$

If this satisfied, we are done. If not, then $conv([A, -b(t_0)])$ does not contain the origin, where $b(t_0) = b + t_0 u$. Thus by Theorem 2.1.1 the triangle algorithm computes a witness, i.e.

$$p'(t_0) \in conv([A, -b(t_0)]) \tag{4.7}$$

such that the following set of $n + 1$ strict inequalities are satisfied:

$$\|p'(t_0) - a_i\| < \|a_i\|, \ \forall i = 1, ...n \tag{4.8}$$

and

$$\|p'(t_0) + b(t_0)\| < \|b(t_0)\|. \tag{4.9}$$

Equivalently, after expanding and simplifying 4.8 and 4.9 we get

$$\|p'(t_0)\|^2 - 2p'(t_0)^T a_i < 0, \ \forall i = 1, ...n. \tag{4.10}$$

$$\|p'(t_0)\|^2 + 2p'(t+0)^T b(t_0) < 0 \tag{4.11}$$

From 4.7 we have

$$p'(t_0) = A\alpha - \alpha_{n+1}(b + t_0 u),\ \sum_{i=1}^{m} \alpha_1 = 1,\ \alpha_i \geq 0,\ \forall i \tag{4.12}$$

Letting $p' = A\alpha - \alpha_{n+1} b$, we may write

$$p'(t_0) = p' - t_0 \alpha_{n+1} u. \tag{4.13}$$

Thus,

$$p'(t_0)^T a_i = p'^T a_i - t_0 \alpha_{n+1} u^T \alpha_i. \tag{4.14}$$

For each t define

$$p'(t) = p' - t\alpha_{n+1} u. \tag{4.15}$$

For $i = 1, ...n$, define

$$g_i(t) = \|p'(t)\|^2 - 2p'(t)^T a_i. \tag{4.16}$$

Also, define

$$g_{n+1}(t) = \|p'(t)\|^2 + 2p'(t)^T b(t). \tag{4.17}$$

It is easy to verify that for $i = 1, ...n$ we have

$$g_i(t) = t^2 \alpha_{n+1}^2 \|u\|^2 - 2t\alpha_{n+1}(p' - a_i)^T u + \|p'\|^2 - 2p'^T a_i \tag{4.18}$$

The coefficient of t^2 in $g_{n+1}(t)$ can be shown as:

$$\alpha_{n+1}(\alpha_{n+1} - 2)\|u\|^2. \tag{4.19}$$

A formal description of incremental triangle algorithm is given in the following:

Incremental Triangle Algorithm $(A, b, \epsilon \in (0,1))$**[6]**

Step 0: (Initialization) Let $u = Ae$, $e = (1, ..., 1)^T \in \Re^m$, Let $C_t = conv([A, -(b + tu)])$. Set $t_0 = 0$. Select $p' = A\alpha - \alpha_{n+1}b \in C_0$.

Step 1: Given $p' = A\alpha - \alpha_{n+1}b$, set $x_0 = \frac{1}{\alpha_{n+1}}\alpha$, τ_0 according to

$$E(\tau_0) = \|Ax_0 - (b + \tau u)\| = min\{\|Ax_0 - (b + tu)\| : \ t \geq t_0\}. \tag{4.20}$$

Replace t_0 with τ_0. If $E(t_0) \leq \epsilon\rho$, set $x_0' = x_0 - t_0 e$, stop.

Step 2: If $p'(t_0) = p' - \alpha_{n+1}t_0 u$ is a witness with respect to C_t, go to Step 3. Otherwise, call triangle algorithm$([A, -(b + t_0 u)], 0, \epsilon)$ to compute a new iterate $p''(t_0)$:

$$p''(t_0) = p'' - \beta_{n+1}t_0 u, \ p'' = A\beta - \beta_{n+1}b \in C_t. \tag{4.21}$$

Replace p' with p'', α with β, and α_{n+1} with β_{n+1}. Go to Step 1.

Step 3: Compute t_0', the smallest value t such that $g_i(t) \geq 0$ for some $i = 1, ..., n$. Replace t_0 with t_0'. Go to Step 2.

In next chapter, we will make a numerical experiment in solving linear system problem $Ax = b$ and make a comparison with triangle algorithm, SOR and Gauss-Seidel.

Chapter 5

Numerical Experiment for Solving $Ax = b$ Via The Triangle Algorithm and Comparison with SOR and Gauss-Seidel

5.1 Review of Gauss-Seidel

In numerical linear algebra, the Gauss-Seidel method, also known as the Liebmann method, or the method of successive displacement, is an iterative method used to solve a linear system of equations. Consider a square system of n linear equations with unknown x

$$Ax = b$$

where the matrix A is decomposed into a lower triangular component L_* and a strictly upper triangular component U, so that $A = L_* + U$. The system of linear equations maybe written as

$$L_* x = b - Ux \tag{5.1}$$

The Gauss-Seidel method now solves the left hand side of [5.1] for x. Using previous value x on the right hand sid. Analytically, this maybe written as

$$x^{k+1} = L_*^{-1}(b - Ux^k). \tag{5.2}$$

However, by taking advantage of the triangular form of L_*, the element of x^{k+1} can be computed sequentially using forward substitution.

Next, we will introduce another iterative method, called the Successive Overrelaxation (SOR) method. Here is the idea:

5.2 Review of SOR method

We can derive the SOR Method from the Gauss-Seidel method. First, we can rewrite the Gauss-Seidel equation as

$$Dx^{k+1} = b - Lx^{k+1} - Ux^k \tag{5.3}$$

where D is the diagonal of A, and L is a strictly lower triangular component, U is defined as before.

So that

$$x^{k+1} = D^{-1}[b - Lx^{k+1} - Ux^k] \tag{5.4}$$

We can subtract x^k from both sides to get

$$x^{k+1} - x^k = D^{-1}[b - Lx^{k+1} - Dx^k - Ux^k] \tag{5.5}$$

The idea of SOR Method is to iterate

$$x^{k+1} = x^k + \omega(x^{k+1} - x^k)_{GS} \tag{5.6}$$

where

$$(x^{k+1} - x^k)_{GS} = D^{-1}[b - Lx^{k+1} - Dx^k - Ux^k] \tag{5.7}$$

and generally $1 < w < 2$. Notice that if $w = 1$ then this is the Gauss-Seidel Method.

We can write the SOR iterates as

$$x^{k+1} = x^k + \omega D^{-1}[b - Lx^{k+1} - Dx^k - Ux^k] \tag{5.8}$$

When we solve for x^{k+1}, we get

$$x^{k+1} = (L + \frac{1}{\omega}D)^{-1}[(\frac{1}{\omega}D - D - U)x^k + b]. \tag{5.9}$$

5.3 Numerical Comparison with Triangle Algorithm, Gauss-Seidel and SOR

In this section, we will solve general linear system $Ax = b$ with incremental triangle algorithm, Gauss-Seidel and SOR. First, let us consider the following example:

Example 1. *Consider the* 3×3 *linear system.*

$$\begin{pmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -24 \\ -30 \\ -24 \end{pmatrix} \tag{5.10}$$

Its solution is $x = [3, -12, -9]$. *If you directly implement the incremental triangle algorithm with* $t = 0$, *it only takes 2 iterations to find the witness that* $(0, 0)$ *is not in* $conv([A, -b])$. *From the solution,* t_* *should be located in* $t_* \geq 12$. *The following table show the iterations with increasing t value.*

Table 5.1: Iterations with Increasing t value

t value	*Iterations*

$t = 12.8$	*77819*
$t = 12.9$	*55791*
$t_{min} = 13$	*5*
$t = 13.5$	*205*
$t = 13.7$	*262*
$t = 14$	*384*

Based on the result, incremental triangle algorithm only need take 5 iteration step when t =13, let us describe this t as t_{min}*. The definition of* t_{min} *will be given in generality in 4 in next section.*

The following table shows the comparison results with triangle algorithm, SOR and Gauss-Seidel method with tol $= 10^{-4}$*, and here initial* x_0*'s of three methods start from zeros(n,1):*

Table 5.2: Ex.1 Triangle VS SOR VS Gauss Seidel

ϵ *value*	*Triangle Algorithm*	*SOR (Yang's Formula)*	*Gauss-S*
$\epsilon = 10^{-4}$	*5*	*10*	*20*

From the above table, the triangle algorithm does less iteration than both SOR and Gauss-Seidel.

Next let us consider an example of a larger linear system:

Example 2. *Consider the linear tridiagonal system:*
A is defined as below:

$$A(i,i) = 2;\ A(i,i+1) = -1;\ A(i+1,i) = -1,\ i = 1,...,n-1;\ A(n,n) = 2$$

and b is defined as $b = [1, 0, ...0, 1]'$

$$\begin{pmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ ... \\ ... \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ ... \\ 1 \end{pmatrix}$$

Comparison of the incremental triangle algorithm with t_{min}*, SOR with optimum w and Gaussian Seidel, for tol* $= 10^{-6}$ *and* $n = 100$ *and* 1000*. Here, the optimum w of SOR was obtained from Young's formula [10],* $t_{min} = 10^{-3}$ *for incremental triangle algorithm and the initial guess* x_0 *of three methods equal to zeros(n,1). Here are the result:*

Table 5.3: Time Comparison:
Triangle vs Optimum SOR vs Gauss-Seidel

Matrix Size(N)	*Incremental Triangle*	*SOR (Young's Formula)*	*Gauss-S* ($w = 1$)
$N = 100$	*0.0033*	*0.0762*	*0.7972*
$N = 1000$	*0.0038*	*24.572*	≥ 1250.793

Table 5.4: Iterations Comparison:
Triangle vs Optimum SOR vs Gauss-Seidel

Matrix Size(N)	*Incremental Triangle*	*SOR (Young's Formula)*	*Gauss-S* ($w = 1$)
$N = 100$	*2*	*236*	*7004*
$N = 1000$	*2*	*2004*	*230417*

According to the above two tables, we conclude that the running time and number of iterations of the triangle algorithm is much less than both the SOR method and Gauss Seidel method.

From the above examples, we can conclude that the incremental triangle algorithm outperforms both SOR and Gauss-Seidel methods in some linear systems, and the value t_{min} selected in the the triangle algorithm can significantly improve the performance of the algorithm. Next, we will give more examples to show that how different t values can affect the performance of triangle algorithm.

5.4 More Numerical Experiment with t_{min}

Let us start from a simple example to observe how different t values can improve the performance of the triangle algorithm. Consider testing if $p = (0, 0)^T$ lies in the triangle $conv((1, 0)^T, (-1, 0)^T, (0, 1)^T)$. Solving this via the triangle when $t = 0$ takes

22 iterations to get an approximation with its absolute error less than $\epsilon = 0.1$, see [4]. This large number of iterations is due to the fact that p is a boundary point of convex hull. If however p is replaced with a point so that a circle of radius ρ centered at p would lie inside the triangle, the complexity of the triangle would improve to $O(\rho^{-2} \ln \epsilon^{-1})$. To solve this, first we convert the convex hull problem into $Ax = b$, $x \geq 0$.

$$\begin{pmatrix} 1 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{5.11}$$

Now consider adding tu to both sides of $Ax = b$, where t is a positive scalar, $u = Ae$, $e = (1, 1, 1)^T$, this gives a new LP feasibility problem, $Ax_t = b + tu$, where $x_t = x + tu \geq 0$. Its corresponding convex hull problem is to test if for any positive t, $(0, 0, 0)^T$ is interior to

$$conv([A, -(b + tu)]) = conv(\{(1, 0, 1)^T, (-1, 0, 1)^T, (0, 1, 1)^T, -(0, t, 1 + 3t)^T\})$$

Now we transfer the original problem $Ax = b$ satisfies $x + te \geq 0$, rather than $x \geq 0$. For example, if $t = 1$, solving the above problem via the triangle algorithm gives an approximate solution within the same error $\epsilon = 0.1$, in only takes 2 iterations. In fact, when A is invertible it can be shown that getting an approximate solution for any positive value t results in a corresponding approximate solution for $Ax = b$ itself.

Definition 4. *Given ϵ, let t_{min} be the value of t such that the number of iterations of incremental triangle algorithm is the smallest.*

Next, let us see more examples with how t_{min} improves the performance of triangle algorithm.

Example 3. *Consider an* 3×3 *linear system*

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 4 \\ 1 & -1 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -14 \\ -14 \\ 10 \end{pmatrix} \tag{5.12}$$

Its solution is $x = [-1, -2, -3]^T$. *Similar to Example 1 we also show the iterations with corresponding* t *value.*

Table 5.5: Iterations with Increasing t *value*

t *value*	*Iterations*
$t = 3.1$	*5651*
$t = 3.5$	*628*
$t_{min} = 3.98$	*71*
$t = 4.0$	*75*
$t = 5.0$	*143*

We observe that the iteration steps decreases significantly when $t = 3.98$, *and start to increase slowly when* t *keep increasing.*

Example 4. *Consider* 4×4 *linear system*

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 6 \\ -1 \\ 4 \end{pmatrix} \tag{5.13}$$

The solution to the system is $x = [2, -3, 4, -1]^T$. *Similar to previous examples, we can choose* t_* *in the range* $t \geq 3$. *The iterations with corresponding t value are shown in the following table:*

Table 5.6: Iterations with Increasing t value

t value	*Iterations*
$t = 3.70$	*4613*
$t = 3.72$	*2922*
$t = 3.725$	*1983*
$t_{min} = 3.73$	*78*
$t = 3.74$	*84*
$t = 3.75$	*91*
$t = 3.8$	*96*
$t = 3.9$	*100*
$t = 4.0$	*102*

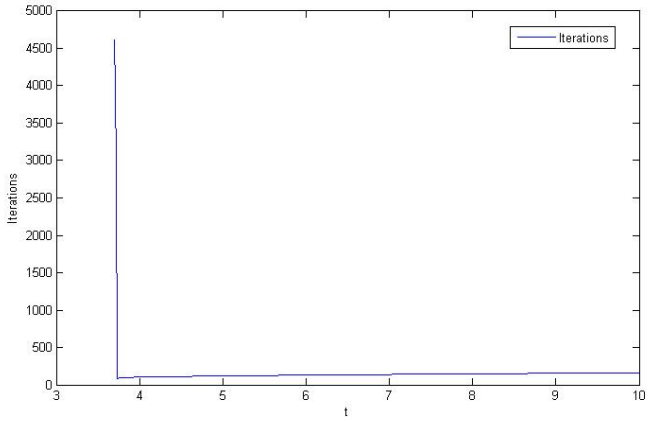

Figure 5.1: Distribution of iterations with different t values

From the table above, we see similar trend as in Example 3: the iteration steps decreases significantly when t is around 3.73, and increases slowly when keep increasing t.

Through all examples above and other experimental results, we can conclude that different t values will influence iterations. The value t_{min} will decrease the number of iteration to the minimum.

From Figure 5.1, we find that iterations will be huge when the point is on the margin of convex hull, and it will decrease hugely when t is getting close to the optimal value, and when t passed the t_{min}, it increases smoothly. We can use this property to find try to compute the value t_{min} in applications. See details in the algorithm in FindOptimalt.m in the Appendix for Matlab codes.

In addition, we can find an initial value t_0 by getOptT.m, then we can make use this property to find a solution by keep doubling t_0, because the iterations will converge to some number when we keep increasing t value.

In general, we have shown that incremental triangle algorithm can solve a general linear system $Ax = b$, outperforming SOR and Gauss-Seidel methods. In addition, with changing parametric values t, we can decrease the number of iterations and improve the performance. In next chapter, we describe the implementation of the triangle algorithm for solving the PageRank problem. The triangle algorithm has a good performance in solving large scale linear systems.

Chapter 6

Solving PageRank Problem via Triangle Algorithm

In this chapter, we will solve the PageRank Problem via the triangle algorithm. Imagining surfing the Web, going from page to page by randomly choosing an outgoing link from one page to get to the next. This can lead to dead ends at pages with no outgoing links, or cycles around cliques of interconnected pages with simply choose a random page from the Web. This theoretical random walk is known as a *Markov chain* or *Markov process*[2]. The limiting probability that an infinitely dedicated random surfer visits any particular page is its Page-Rank. A page has high rank if other pages with high rank link to it.

Let W be the set of Web pages that can be reached by following a chain of hyper-links starting at some root page, and let n be the number of pages in W. For Google, the set W actually varies with time, but by June, 2004, n was over 4 billion. Let G be the n-by-n connectivity matrix of a portion of the Web, that is, $g_{ij} = 1$ if there is a hyper-link to page i from page j and $g_{ij} = 0$ otherwise. The matrix G can be huge, but it is very sparse. Its jth column shows the links on the jth page. The number of non-zeros in G is the total number of hyper-links in W, see [2].

Let r_i and c_j be the row and column sums of G:

$$r_i = \sum_j g_{ij}, \;\; c_j = \sum_i g_{ij} \tag{6.1}$$

The quantities r_j and c_j are the in-degree and out-degree of the j-th pages. Let p be the probability that the random walk follows a link. A typical value is $p = 0.85$. The $1 - p$ is the probability that some arbitrary page is chosen and $\delta = (1 - p)/n$ is the probability that a particular random page is chosen. Let A be the n-by-n matrix whose elements are, see [8]:

$$a_{ij} = \begin{cases} pg_{ij}/c_j + \delta & : \;\; c_j \neq 0 \\ 1/n & : \;\; c_j = 0 \end{cases} \tag{6.2}$$

Notice that A comes from scaling the connectivity matrix by its column sums. The jth column is the probability of jumping from the jth page to the other pages on the Web. If the jth page is a dead end, that is has no out-links, then we assign a uniform probability of $1/n$ to all the elements in its column. Most of the elements of A are equal to δ, the probability of jumping from one page to another without following a link.

Here, the matrix A is the transition probability matrix of the Markov chain. Its elements are all strictly between zero and one. Its column sums are all equal to one. It concludes that a nonzero solution of the equation

$$x = Ax \tag{6.3}$$

exists and is unique to within a scaling factor. If this scaling factor is chosen so that $\sum_i x_i = 1$, then x is the *state vector* of the Markov chain and is Google Page Rank. The elements x are all positive and less than one.

We can rewrite the equation as

$$(I - A)x = 0$$

Set $A' = (I - A)$ and $b' = 0$, we can also regard this problem as convex hull problem.

$$\begin{aligned} A'x &= b', \\ e^T x &= 1, \\ x &\geq 0. \end{aligned} \tag{6.4}$$

We can also use the incremental triangle algorithm to solve this problem. In the next experimental chapter, we randomly generated the random matrix from dimension 200 to 2200, and make a result comparison with classic power method, which shows that triangle algorithm is competed to solve this problem. Finally, we use the practical web data from Prof. Kamvar, which is $281,903 \times 281,903$ matrix. The experiment shows that triangle algorithm only need take 1 iteration to find the solution, which is an amazing result.

Chapter 7
Numerical Comparison with Power Method

In this section, we will implement triangle algorithm into PageRank problem and make a comparison with the Power Method. We randomly generate the data based on the **convex combination**:

$$G = \alpha S + (1 - \alpha)ev^T \tag{7.1}$$

where S is the Stochastic Matrix and Damping factor $0 \leq \alpha < 1$, here we set $\alpha = 0.85$. e is column vector of all ones. Personalization vector $v \geq 0$, $\|v\|_1 = 1$.

From 7.1 and G is Stochastic, with eigenvalues:

$$1 > \alpha|\lambda_2(S)| \geq \alpha|\lambda_3(S)| \geq ... \tag{7.2}$$

We can get the unique dominant left eigenvector:

$$\pi^T G = \pi^T,\ \pi \geq 0\ and\ \|\pi\|_1 = 1 \tag{7.3}$$

Where π_i is PageRank of Web page i.

The following is a brief introduction to Power Method:

Power Method

Want: π such that $\pi^T G = \pi^T$

Power Method: Pick an initial guess x_0. Repeat

$$[x_{k+1}]^T := [x_k]^T G \tag{7.4}$$

until "termination criterion satisfied".

More information can be available in code **pagerankpow.m** in the Appendix.

Then, we generated the datasets randomly with the dimension from 200 to 2200. We implemented the both triangle algorithm and Power Method in Matlab with tolerance 10^{-4}. In each dimension, we repeated 10 times and get average of that. The experiment results are showed as follows:

Table 7.1: Triangle Algorithm VS Power Method

Dimension n	Triangle Iter (Avg)	Power Iter (Avg)	Triangle Time (s)	Power Time (s)
200	1	4.2	0.013261	0.0036568
400	1	4.0	0.027061	0.0078675
600	1	4.0	0.017507	0.0053561
800	1	3.6	0.03037	0.0013188
1000	1	3.0	0.044809	0.0046782
1200	1	3.0	0.062895	0.0080632
1400	1	3.0	0.081302	0.011503
1600	1	3.0	0.10477	0.014961
1800	1	3.0	0.13136	0.018843

2000	1	3.0	0.16327	0.023479
2200	1	3.0	0.19548	0.028181

From table 7.1, we observed that triangle algorithm need less iterative steps to find the solution compared with the Power Method. So we believed that triangle algorithm is also a good way to solve the Google page rank problem.

Next, to show that triangle algorithm is also a very good method to solve the practical Google Page Rank problem. we will use the real web data from Prof. Kamvar's personal research website(http://kamvar.org/personalized_search) with the dimension $281,903 \times 281,903$. In the following table, it showed the computation result for ϵ from 10^{-4} to 10^{-10}.

Table 7.2: Triangle Algorithm's performance for practical data method

ϵ	Iterative Steps	Time (s)
$\epsilon = 10^{-4}$	1	3765.0395
$\epsilon = 10^{-5}$	1	3786.4572
$\epsilon = 10^{-6}$	1	3812.4369
$\epsilon = 10^{-7}$	1	3812.7181
$\epsilon = 10^{-8}$	1	3812.3995
$\epsilon = 10^{-9}$	1	3823.0131
$\epsilon = 10^{-10}$	1	4905.1076

From table 7.2, we found that triangle algorithm had a suprising good performance, which only need 1 iterative step to find the solution. This can be explained by the fact that we always find the optimal pivot for each iteration step so that it can find the solution in just 1 iteration.

In sum, we concluded that we had found another way to compute the PageRank problem. The tirangle algorithm is a competed algorithm to compute the large scale sparse matrix.

Chapter 8
Conclusion and Future Work

In this thesis, we reviewed the basic conceptions of the triangle algorithm, which was written by Prof.Kalantari. Then, we made some implementations of triangle algorithm into convex hull problem and a linear system problem. Firstly, we implemented the triangle algorithm into solving convex hull problem and made a comparison with Frank-Wolfe algorithm. From the result, the triangle algorithm outperformed the Frank-Wolfe on large scale problem. Secondly, we implemented the incremental version of triangle algorithm to solving the linear system $Ax = b$, and made some comparison with SOR and Gauss-Seidel methods. The triangle algorithm is more efficient taking fewer iterations than these algorithms. Finally, we implemented the triangle algorithm into solving the PageRank problem and made the comparison with Power method. The triangle algorithm took less iterations to reach the same accuracy. Surprising, the triangle algorithm only took 1 iteration to solve the Kamvar datasets for large scale PageRank problem whose dimension is 281,903.

We have showed that the triangle algorithm was a competed algorithm to solve some linear systems. In future, we definitely need implement the triangle algorithm into more linear system problems. And we can optimize each iteration step to improve

the calculation speed. Moreover, we could prove and find the existed the relationship between the optimal t_{min} and pivots. Finding the optimal t_{min} can reduced the time and iterations significantly via our experiment in this paper.

Chapter 9
Appendix for Matlab Codes

9.1 Triangle Algorithm for LP Feasibility

```
function [time,iteration,average] = Triangle_v2(A,b,epsilon)
%Solve convex hull problem via Triangle Algorithm

%Given P, a set of points a1,a2,...,an in m-dimension space
%find whether the target point b is in the convex hull of P
%If yes, return the convex combination of a1,a2,...,an to represent b.
format long;
[~,n]=size(A);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 1
%Find R = max{norm(ai-b,2)}, the scality of the point set.
dist = zeros(1,n);
R = 0;
for i=1:n
    if R<=norm(A(:,i)-b,2) R = norm(A(:,i)-b,2); end
    dist(1,i) = norm(A(:,i)-b,2);
```

```
end

for i=1:n-1
    k = i;
    for j=i+1:n
        if dist(1,j)<dist(1,k) k = j; end
    end
    temp = dist(1,i);
    dist(1,i) = dist(1,k);
    dist(1,k) = temp;
    tempvector = A(:,i);
    A(:,i) = A(:,k);
    A(:,k) = tempvector;
end
A0 = A;
n0 = n;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 2
%Set the center point of P as the initial iterate
tic;
coefficient = zeros(1,n);
for i=1:n
    coefficient(1,i) = 1/n;
end
```

```
p = A*coefficient';
gap = norm(p-b,2);
iteration = 0;
sumoperation = 0;
sumauxilary = 0;
sumneedhelp = 0;
index = 0;
unitvector = zeros(1,n);
bnorm = b'*b;

frequency = zeros(1,n);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 3
%Start iteration untill gap is small enough
while (gap>epsilon*R)
    index = 0;  bestcos = 0;
    pnorm = p'*p;
    diff = (bnorm-pnorm)/2;
    gapvector = b-p;
    flag = 0;
    tempinner = p'*gapvector;
    threhold = 0.14*gap;
```

```
    for i=1:n;
        sumoperation = sumoperation+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 4
%Find a pivot, judge the pivot is good enough or not
        temp = gapvector'*A(:,i);
        if diff<temp
            cosine = (temp-tempinner)/(norm(A(:,i)-p,2));
            if (bestcos<cosine)
                index = i;
                bestcos = cosine;
            end
            if (cosine>threhold)
                flag = 1;
                break;
            end
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 5
%No pivot, report NO
%Otherwise, calculate the next iterate, the projection of b lies on the
%segment from p to ai
%Update the covex combination for next iterate
```

```
    if (index == 0)
        break;
    else
       if ((flag == 0)&&(iteration>20))
%Using auxiliary pivot based on the history feedback data
%Associate each point a counter to count the number of times chosen to be
%pivot.
        sumneedhelp = sumneedhelp+1;
        pivot = A0*(frequency'/iteration);
        temp = norm(pivot-p,2);
        if norm(pivot-b,2)<temp
        cosine = (pivot-p)'*gapvector/temp;
        if (bestcos<cosine)
           sumauxilary = sumauxilary+1;
           alpha = bestcos/temp;
           p = p+alpha*(pivot-p);
           gap = norm(p-b,2);
           coefficient = (1-alpha)*coefficient+alpha*
           (frequency/iteration);

           A = horzcat(A,pivot);
           if (n==n0)
               n = n+1;
               auxcoefficient = frequency/iteration;
```

```
            else
              n = n+1;
              auxcoefficient = [auxcoefficient;frequency/iteration];
              end
              frequency = frequency + frequency/iteration;
              iteration = iteration+1;
              continue;
            end
        end
end

iteration = iteration+1;
temp = norm(A(:,index)-p,2);
alpha = bestcos/temp;
p = p+alpha*(A(:,index)-p);
gap = norm(p-b,2);
if (index<=n0)
        unitvector(1,index) = 1;
        coefficient = (1-alpha)*coefficient+alpha*unitvector;
        unitvector(1,index) = 0;
        frequency(1,index) = frequency(1,index)+1;
 else
        coefficient = (1-alpha)*coefficient+alpha
        *auxcoefficient(index-n0,:);
```

```
            frequency = frequency+auxcoefficient(index-n0,:);
        end

    end
end
time = toc;
average = sumoperation/iteration;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 6
%If gap is small enought, report YES
if (index == 0)
    disp('Not in convex hull!');
else
    disp(['The target point is in the convex hull!
    The number of iteration is ', num2str(iteration),
    ' and the running time is ', num2str(time),
    ' and # need help times ', num2str(sumneedhelp),
    ' and # of auxilary points is ', num2str(sumauxilary)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Verification for correctness
correct = 1;
```

```
xnorm = norm(coefficient,1);
if (abs(xnorm-1)>1e-10) correct = 0; end
[~,temp] = size(find(coefficient<-1e-10));
if (temp >0) correct = 0; end
if (norm(A0*coefficient'-b,2)>epsilon*R) correct = 0; end
if (correct == 1)
    disp('The solution is verified to be correct! ');
else
    disp('The solution is false! ');
end
```

9.2 Incremental Triangle Algorithm

```
function [pprime, error, iter, alpha, x0, solError,xold,flag]
= incTriangle2(A, b, t, tol)
%
% input   A         REAL matrix
%         pprime    REAL starting guess to p
%         p         REAL right hand side vector
%         tol       REAL error tolerance
%
% output  pprime    REAL approximation to p
%         error     REAL error norm
%         iter      INTEGER number of iterations performed
%         alpha     REAL vector of convex coefficients
```

```
%          x_0        solution to Linear system
%          xold       orignal solution
%          Flag       0: not in convex hull,
%                     1: in convex hull
format long;
[~,w]= size(A);
c = ones(w,1);
u = A*c;
bnew = b+t*u;
S = [A -bnew];

[m,n]=size(S);
%E = eye(n);
E = sparse(1:n,1:n,1);

p = zeros(m,1);
pprime = (sum(S')/n)';
counterTol = 1.0e10;
S0 = S;                                          % Fixing S0

iter = 0;                                        % initialization
% t=1;

% step 0: initial guess
```

```
error = norm(p - pprime);
alpha = (1/n)*ones(n,1);
[z,~] = size(alpha);

x0 = alpha(1:z-1,1)/alpha(z);

solError = norm(A*x0-(b+t*u));

counter = 0;
tic;
while error > tol && solError > tol          % iterative steps

    flag = 0;
    beta = []; index = [];
    betai = 1; indexi = 1;
    [~,n] = size(S);

    for i=1:n                                % step 1: find pivot
%
%   Using the law of cosines, this block of code will generate a set
%   of angles beta_i corresponding to each of the candidate pivots
%
        if norm(pprime - S(:,i)) >= norm(p - S(:,i))
            % Law of Cosines
```

```
            a = norm(pprime - S(:,i));
            B = norm(p - S(:,i));
            c = norm(p - pprime);
            % Generating the set of beta_i and marking the indicies
            % in the set index
            beta(betai) = acos((a^2 + c^2 - B^2)/(2*a*c));
            index(indexi) = i;

            flag = 1;
            % incrementing position in index and beta arrays
            betai = betai+1;
            indexi = indexi+1;

        elseif norm(pprime - S(:,i)) < norm(p - S(:,i))
            %ignore vertex that does not satisfy condition
            continue;
        end
    end
                                        % Check for witness
    if flag == 0;
        disp('Witness was found. P is not in the convex set of S')
        break;
    end
%
```

```
%   The selection of the ideal pivot per iteration will be based
%   on the angle which is both the minimum of the set {beta}
%
    [~, cindex] = min(beta);
%
%   This checks the previous and previous-previous verticies
%   If there is a pattern, this code will create an auxillary
%   vertex defined to be the average of the two verticies the
%   iterate is oscillating between
%
%%
%%   This calculates and moves the iterate to a new location
%%   step 2: update alpha
%%
    ic = index(cindex);

    if iter == 0
        ipp = ic;
        v = S(:,ic);
        e = E(:,ic);
    elseif iter == 1
        ip = ic;
        v = S(:,ic);
        e = E(:,ic);
```

```
    else
        if ic == ipp
            ipp = ip;
            ip = ic;
            counter = counter + 1;
        else
            counter = 0;
            ipp = ip;
            ip = ic;
        end
        if counter > counterTol
%
%   This code executes the auxillary pivot points by taking the average
%   of two pivots that the iterate oscillates between cycles. It also
%   updates the standard basis vector matrix to properly update the
%   alpha coefficients vector
%
            v = (S(:,ip) + S(:,ipp))/2;
            S = [S v];
            e = (E(:,ip) + E(:,ipp))/2;
            E = [E e];
        else
            v = S(:,ic);
            e = E(:,ic);
```

```
        end
    end
%
%   Updates the alpha vector and generates a new iterate
%
    beta = (p-pprime)'*(v-pprime)/(norm(v-pprime)^2);
    alpha = (1-beta)*alpha + beta*e;
    pprime = S0*alpha;
%
%   Updates the approximate solution to the system of equations
%   given the alpha coefficients of the augmented matrix [A -b]
%
    x0 = alpha(1:z-1,1)/alpha(z);

    solError = norm(A*x0 - (b+t*u));
    error = norm(p - pprime);                    % update error
    iter = iter + 1;                             % update iterations
%     t = t+1;
end

time = toc;
e = ones(w,1);
xold=x0-t*e;
% if flag == 0;
```

```
%     disp('Witness was found. P is not in the convex set of S')
% elseif flag == 1;
%     disp('P is in the convex set of S')
% end
% END incTriangle.m
```

9.3 Power Method

```
function [time,iteration] = pagerankpow(A,b,epsilon)

% PAGERANKPOW PageRank by power method.
% x = pagerankpow(G) is the PageRank of the graph G.
% [time,iteration] = pagerankpow(A,b,epsilon)
% counts the number of iterations.
% Link structure

[~,n] = size(A);



R = 0;
for i=1:n
    if R<norm(A(:,i)-b,2)
        R = norm(A(:,i)-b,2);
    end
end
```

```
tic;
% Power method
x = ones(n,1)/n;
iteration = 0;
gap = norm(A*x-x,2);

while (gap>epsilon*R)
    gap = norm(A*x-x);
    x = A*x;
    iteration = iteration+1;
end

time = toc;


if (gap<epsilon*R)
    disp(['The number of iteration is ', num2str(iteration), ' and the run
else
    disp('No solution!');
end
```

9.4 Generated Test Data for LP Feasibility

```
function [A,b] = TestData(m,n)
```

```
A = randsphere(n,m,2500);
A = A';
A = A+2500*ones(m,n);
alpha = zeros(1,n);
for i=1:n
    alpha(1,i) = random('unif',1,n);
end

sum = 0;
count = 0;
for i=1:n
    if (alpha(1,i)>0)&&(alpha(1,i)<(m+1))
        sum = sum+alpha(1,i);
        count = count+1;
    else
        alpha(1,i) = 0;
    end
end

for i=1:n
    alpha(1,i) = alpha(1,i)/sum;
end
```

```
b = A*alpha';
```

9.5 Generated Data for Page Rank Problem

```
function [A,G,b] = Gmatrix_v4(n)

%Randomly generate adjacent matrix with value 0 or 1
%X = sprand(n,n,0.05);
X = sprand(n,n,0.5);

for i=1:n
    for j=1:n
        X(i,j) = round(X(i,j));
    end;
end;

%Normalize the adjacent matrix by column
for i=1:n
    norm1 = sum(X(:,i));
    if (norm1>0)
        X(:,i) = X(:,i)/norm1;
    else
        X(:,i) = ones(n,1)/n;
    end;
end
```

```
alpha = 0.85;
G = alpha*X+(1-alpha)/n*ones(n,n);
b = zeros(n,1);
A = G-eye(n);
```

9.6 Get Optimal T

```
function t=getOptT(A,b)
%How to get t
%method1 max
t=max(abs(b))/max(max(abs(A)));

%method 2 norm
% t=norm(b)/norm(A);
t=ceil(t);

%End getT.m
format long;
%Get the intial t
told=getOptT(A,b);
[pprime1, error1, iter1, alpha1, x01, solError1,xold1,flag1]
= incTriangle2(A, b, told, 1e-1);

%Get the new t and gap = 1
tnew = told+1;
```

```
[pprime2, error2, iter2, alpha2, x02, solError2,xold2,flag2]
= incTriangle2(A, b, tnew, 1e-1);

if flag1==0 || flag2==0
    disp('not in convex hull');
end
n=3;
while n>0 && iter1 > iter2 && flag1==flag2

    told = tnew;
    [pprime1, error1, iter1, alpha1, x01, solError1,xold1,flag1]
= incTriangle2(A, b, told, 1e-1);
    tnew = tnew + 1;
    [pprime2, error2, iter2, alpha2, x02, solError2,xold2,flag2]
= incTriangle2(A, b, tnew, 1e-1);


    n=n-1;
end
tnew = told;
told = tnew-1/2;
[pprime1, error1, iter1, alpha1, x01, solError1,xold1,flag1]
= incTriangle2(A, b, told, 1e-2);
```

```
[pprime2, error2, iter2, alpha2, x02, solError2,xold2,flag2]
= incTriangle2(A, b, tnew, 1e-2);

k=2;
error =1;
while k>0 || error>1e-4
    if iter1 >= iter2
        told=(told+tnew)/2;
        [pprime1, error1, iter1, alpha1, x01, solError1,xold1,flag1]
= incTriangle2(A, b, told, 1e-3);
    elseif iter1 < iter2
        diff = tnew-told;
        tnew=told;
        told=tnew-diff;
        [pprime1, error1, iter1, alpha1, x01, solError1,xold1,flag1]
= incTriangle2(A, b, told, 1e-3);
        [pprime2, error2, iter2, alpha2, x02, solError2,xold2,flag2]
= incTriangle2(A, b, tnew, 1e-3);
    end
    k=k-1;
    error = abs(told-tnew);
end
```

```
[pprime1, error1, iter1, alpha1, x01, solError1,xold1,flag1]
= incTriangle2(A, b, told, 1e-3);
[pprime2, error2, iter2, alpha2, x02, solError2,xold2,flag2]
= incTriangle2(A, b, tnew, 1e-3);
```

9.7 Frank Wolf Algorithm

```
function [time iteration] = FrankWolf(A,b,epsilon)

[~,n]=size(A);

R = 0;
for i=1:n
    if R<norm(A(:,i)-b,2) R = norm(A(:,i)-b,2); end
end

tic;
iteration = 0;
unitvector = zeros(n,1);
x = ones(n,1)/n;
gap = norm(A*x-b,2);

c = A'*b;
while (gap>epsilon*R)
    iteration = iteration+1;
```

```
    temp = A'*(A*x)-c;
    [~,i] = min(temp);
    unitvector(i,1) = 1;
    temp = unitvector-x;
    tempvector = A*temp;
    tempnorm = tempvector'*tempvector;
    if (tempnorm<1e-10)
        break;
    end
    alpha = (b-A*x)'*tempvector/tempnorm;
    if (alpha >1) alpha = 1;
    else if (alpha<1e-10)
            break;
        end
    end
    x = x + alpha*temp;
    unitvector(i,1) = 0;
    gap = norm(A*x-b,2);
end

time = toc;
if (gap<epsilon*R)
    disp(['The target point is in the convex hull!
    The number of iteration is ', num2str(iteration),
```

```
    ' and the running time is ', num2str(time)]);
else
    disp('Not in convex hull');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Verification for correctness
correct = 1;
xnorm = norm(x,1);
if (abs(xnorm-1)>1e-10) correct = 0; end
[temp,~] = size(find(x<-1e-10));
if (temp >0) correct = 0; end
if (norm(A*x-b,2)>epsilon*R) correct = 0; end
if (correct == 1)
    disp('The solution is verified to be correct! ');
else
    disp('The solution is false! ');
end
```

9.8 SOR and Gauss Seidel

```
function [ x ,result] = sor( A, b,w,max_ite ,tol)
D=diag(diag(A));
L=tril(A)-D;
U=triu(A)-D;
```

```
[m,n]=size(A);
x=zeros(n,1);
for iter=1:max_ite
xprev=x;
x=(D+w*L)\(w*b-(w*U+(w-1)*D)*x);
error=norm( x - xprev) / norm( x );
result(iter,:)= [iter error ];
if ( error <= tol )
    break;
end ;

%Yong's optimal formula
D=diag(diag(A));
J = D\(D - A); e = eig(J);
r=max(abs(e));
w = 2/(1 + sqrt(1 - r^2));
```

References

[1] K. L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. July 2008.

[2] Department of Methematics, North Carolina State University, Raleigh, USA. *The Mathematics Behind Google's PageRank*.

[3] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Res. Logist Quart.*, 3(1956):95–110.

[4] T. Gibson and B. Kalantari. Experiments with the triangle algorithm for linear systems. *23rd Annual Fall Workshop on Computational Geometry, City College of New York*, October 2013.

[5] J. E. Goodman and J. O'Rourke, editors. *Discrete Mathematics and its Applications*. Chapman Hall Boca Raton, 2004.

[6] B. Kalantari. Solving linear system of equations via a convex hull algorithm. *arxiv.org/pdf/1210.7858v1.pdf*, 2012.

[7] B. Kalantari. A characterization theorem and an algorithm for a convex hull problem. *Annals of Operations Research*, 2014.

[8] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for the computation of pagerank. July 2012.

[9] M. Li and B. Kalantari. Experimental study of the convex hull decision problem via a new geometric algorithm. *23rd Annual Fall Workshop on Computational Geometry, City College of New York*, October 2013.

[10] D. M. Young. Iterative solution of large linear systems. *Academic Press*, 1971.