ABSTRACT

GOVAN, ANJELA YURYEVNA. Ranking Theory with Application to Popular Sports. (Under the direction of Professor Carl D. Meyer).

The *rank* of an object is its relative importance to the other objects in a finite set of size n. Often a rank is an integer assigned from the set $\{1, 2, ..., n\}$. Ideally an assignment of available ranks ($\{1, 2, ..., n\}$) to n objects is one-to-one.

However in certain circumstances it is possible that more than one object is assigned the same rank. A *ranking model* is a method of determining a way in which the ranks are assigned. Typically a ranking model uses information available to determine a *rating* for each object. The ratings carry more information than the ranks; they provide us with the degree of relative importance of each object. Once we have the ratings the assignment of ranks can be as simple as sorting the objects in descending order of the corresponding ratings. Ranking models can be used for a number of applications such as sports, web search, literature search, etc.

The type of ranking investigated in this work has close ties with the Method of Paired Comparison. Oftentimes the information that is the easiest to obtain or naturally available is the relative preference of the objects taken two at a time. The information is then summarized in a weighted directed graph and hence as the corresponding matrix. A number of ranking models makes use of the matrix representation of paired comparisons to compute ratings of the individual objects.

Two ranking models proposed and investigated in this work start with forming nonegative matrices that do represent certain pairwise type comparisons. The models have different approaches to computing the rating scores. The Offense-Defense Model makes use of the Sinkhorn-Knopp Theorem on equivalence matrix balancing, whereas the Generalized Markov model is based on the theory of finite Markov chains. Both models are then used to compute the ratings of the National Football League teams, National Collegiate Athletic Association football and basketball teams. The ratings are used to perform game predictions. The proposed models are not specific to sports and can be applied to any situation consisting of a set of objects and a set of pairwise information. However, picking team sports as a ranking application allows for unrestricted access of free and abundant data.

The game predictions experiments consisted of both foresight and hindsight predictions. All the experiments included the proposed models as well as several current sports ranking methods.

Ranking Theory with Application to Popular Sports

by
Anjela Yuryevna Govan

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2008

APPROVED BY:

Dr. Mansoor Haider	Dr. Alun Lloyd
Dr. Carl D. Meyer Chair of Advisory Committee	Dr. E. L. Stitzinger

DEDICATION

To my husband Vincent, my family, and all my teachers.

BIOGRAPHY

Anjela Y. Govan has graduated with Bachelors of Science in Mathematics from the University of Michigan in 2000. She recieved Masters of Science in Mathematics from Colorado State University in 2004.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Carl Meyer for all the time, effort, advice, and support he has given me over the past three years. I would like to thank all my teachers for helping me to reach my potential. My special thanks go to Dr. Amy Langville for her advice and support, to Martin Meyer for starting me on my sports data scraping, to Danielle Freudenberg for persevering through my thesis, and to Becky Meyer for her kindness. Many thanks to my parents George and Galina, my sisters Lena, Diana and Alice, to my extended family, especially Ed, Beccie, Grandma Clare, Grandma Lena, Grandpa Frank, and all my friends. Finally, I owe an enormous debt of gratitude to my husband whose patience, love, and support made this work possible. Thank you Vincent for teaching me Perl, helping me debug my programs, proofreading my writing, and always being there for me every step of the way.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The *rank* of an object is its relative importance to the other objects in a finite set of size n. Often a rank is an integer assigned from the set $\{1, 2, ..., n\}$. Ideally an assignment of available ranks ($\{1, 2, ..., n\}$) to n objects is one-to-one. However in certain circumstances it is possible that more than one object is assigned the same rank. A *ranking model* is a method of determining a way in which the ranks are assigned. Typically a ranking model uses information available to determine a *rating* for each object. The ratings carry more information than the ranks because they provide us with the degree of relative importance of each object. Once we have the ratings the assignment of ranks can be as simple as sorting the objects in descending order of the corresponding ratings. Ranking models can be used for a number of applications such as sports, web search, literature search, etc.

The type of ranking investigated in this work has close ties with the Method of Paired Comparison. Oftentimes the information that is the easiest to obtain or naturally available is the relative preference of the objects taken two at a time. Within the Method of Paired Comparison there have been studies on inconsistencies that might be present when one considers using the set of preferences to arrive at a ranking. A simple example of such inconsistencies in competitive sports such as American football, baseball, soccor etc. is as follows. Throughout the season teams are "compared" two at a time with the result of the outcome usually based on game scores. Given that team A beat team B and team B beat team C it is not necessarily the rule that this relationship is transitive. It is not uncommon to encounter a situation in which team C beats team A. In the language of the Method of Paired Comparison this is referred

to as a circular triad [69]. There have been numerous studies that have focused on finding the "best" ranking given a set of preferences with just a few of the studies being [2, 26, 30, 41, 52, 65, 68, 116, 134]. One common approach is to use the set of preferences to form a matrix, oftentimes nonnegative, with the rating vector being a solution to some system of linear equations or a dominant eigenvector.

This work is then structured in the following manner. Chapter 2 is dedicated to the historical development and the basic terms used in the Method of Paired Comparison. Chapter 2 also includes some of the ranking models that arose directly from the Method of Paired Comparison. Chapters 3 and 4 deal with the two main applications of ranking, which are sports and web page ranking. The chapters include descriptions of the current ranking methods that use available pairwise information on teams and web pages to generate respective ratings and ranks. Chapters 5 and 6 are centered on the development of new ranking models that are then tested against current ranking models. The testing of all the models is done using National Football League (NFL), National Collegiate Athletic Association (NCAA) football, and NCAA basketball game data over the past several seasons (subject to availability). The data is used to produce ratings for the teams, and the ratings in turn are used to make game predictions. Acquisition and processing of game data and game prediction results were done using Perl scrips (included in Appendix B), whereas the numerical computations of the ratings were done using Matlab (code in Appendix A). The concluding chapters 7 and 8 contain the testing results and analysis and finally the conclusion.

Chapter 2

Historical Note on the Method of Paired Comparison and Ranking

2.1 The Method of Paired Comparison

The Method of Paired Comparison (aka the Method of Pairwise Comparison) consists of three basic elements - the objects (or persons, treatments, stimuli, and the like), the judges, and the preferences or comparison results of the pairs of objects decided by a judge. For a given pair of objects A and B in a set of n objects the preference may be A is preferred to B or B is preferred to A (the possibility of no preference between A and B is often ignored). In some cases all the possible preferences (done by a given judge) are available, this is referred to as a *balanced paired-comparison* [31] or a n-tournament [25], and we will refer to it as a complete set of preferences. In some papers the preferences are instead called pairwise comparisons [2]. If there are several judges then a set of preferences (complete or incomplete) is generated by each judge. Frequently applications are based on a single judge, i.e. a single set of preferences.

The basic use of the Method of Paired Comparison can be traced back as far as the 1860 publication on Psychophysics by Fechner [39, 40]. A more definite reference to the Method of Paired Comparison was made some years later by Thurstone also in Psychophysics [130, 131, 132]. Theoretical development of the method with applications outside of Psychology has been done since 1940, starting with an authoritative publication on the subject by Kendall and Babington Smith [69]. Using notation introduced by

Kendall and Babington Smith, if an object A is preferred to an object B in a given set of n objects and a single judge then write $A \rightarrow B$. One indication of an inconsistency in a complete set of preferences is a *circular triad*, i.e. $A \rightarrow B$, $B \rightarrow C$, and $C \rightarrow A$. The *coefficient of consistence* (ξ) of a given set of preferences was defined to depend on the number of objects and the number of triads among the preferences [69]. $\xi = 1$ if and only if there are no triads among the preferences. ξ decreases to 0 as the number of triads in the complete set of preferences increases. The number of circular triads, c, can also be interpreted as the number of preference reversals necessary to break all ties in the score vector $\mathbf{a}$ (a_i=number of times i is preferred to other objects). Once all the ties are removed the complete set of preferences represents a ranking, also called a *transitive n-tournament* [25], or a *linear ordering* [26] that is not necessarily unique. David calls the resulting ranking a *nearest adjoining order*. In 1961, Slater proposed a different measure of inconsistency he called $\mathbf{i}$ that is the minimum number of preference reversals needed to reach a nearest adjoining order, note that $\mathbf{i} \leq c$ [124, 61]. Another type of inconsistency has been studied by Gerard and Shapiro [47]. If a prior ordering of the objects has $A \rightarrow B \rightarrow C$, Gerard and Shapiro call the situation in which the complete set of preferences contains $A \rightarrow B$ and $C \rightarrow B$ a *separation*. Some attention has been given to the study of an incomplete set of preferences. Bauer, for example, proposed a method for computing the number of circular triads in a paired comparison when the set of preferences is incomplete [10, 13, 31].

A set of preferences can be represented using a directed graph where objects are nodes and a preference $A \rightarrow B$ is a directed edge from the node representing A to the node representing B. Hence a set of preferences can also be denoted using an adjacency matrix. This representation initially had been suggested by Kendall and Babington Smith [69]. The tournament matrix and tournament directed graph have been objects of study of their own since [25, 35, 100, 99].

2.2 Various Ranking Models Associated with the Method of Paired Comparison

A ranking can be thought of as a special case of a complete set of preferences for a set of n elements. As mentioned before, a ranking constitutes a transitive n-

tournament. The ranking of n objects can be used to form a complete set of preferences, however, a complete set of preferences does not always represent a ranking. A complete set of paired comparisons is more general than a ranking and allows for inconsistencies such as circular triads. There has been considerable interest in determining the "near" ranking to a given complete set of preferences. Ryser [114] and Fulkerson [43] have put forth methods that produce rankings that minimize the number of inconsistencies for *monotone tournaments* (the objects are rearranged so that the corresponding score vector is monotonically increasing, $a_1 \leq ... \leq a_n$).

In his later publication Kendall elaborated on the procedure proposed by Wei [135] to produce ratings using a complete set of preferences [68]. Kendall created a nonnegative matrix, $\mathbf{P}$, such that P_{ij} is the number of times i is preferred to j, with $P_{ii} = 1/2$. The ranking can be generated by using the relative order of the entries of $\mathbf{P}^n\mathbf{e}$ (where $\mathbf{e}$ is a vector of all 1's) as $n \to \infty$. The relative order of values converges, and the limit is the relative order of values in the Perron vector of the matrix $\mathbf{P}$. The matrix $\mathbf{P} \geq 0$ is (usually) irreducible and primitive by construction. In case the matrix $\mathbf{P}$ is reducible the method is applied to the irreducible, primitive diagonal blocks of $\mathbf{QPQ}^T$. Kendall's publication was followed by numerous works that were aimed at producing ratings and consequently rankings from a given complete or incomplete set of preferences [2, 26, 30, 41, 52, 65, 68, 116, 124, 134].

A notable paper in the development of ranking models with reference to paired comparisons is by Bradley and Terry [15]. In it the authors suppose that each of the t objects has true ratings $\pi_1, ..., \pi_t$, s.t. $\pi_i \geq 0$ and $\sum \pi_i = 1$. Then the probability that i is preferred to j is defined to be $\pi_i/(\pi_i + \pi_j)$. The paper continues with the development of the probability of the observed ranking within the kth comparison (kth judge) of i and j and uses this probability to obtain the likelihood function for the vector $\boldsymbol{\pi}$. The preference ratio proposed by Bradley and Terry has been used in consequent ranking models such as the one developed by Ford [41].

Another noteworthy method for ranking is the Analytic Hierarchy Process created by Saaty [115]. Given the objects $C_1, ..., C_n$ and the corresponding preferences, Saaty denoted a_{ij} to be the amount of preference of C_i over C_j and set $a_{ji} = 1/a_{ij}$. The matrix $\mathbf{A}$ is referred to as a *reciprocal matrix*. If the preferences were perfect then $a_{ik} = a_{ij}a_{jk}$ for all i, j, k, i.e. there is consistency in the comparison resulting in what is called a *consistent* reciprocal matrix. If the reciprocal matrix $\mathbf{A}$ is consistent then all

its eigenvalues are zero except for the dominant one which is n. The rating score, w_i, for the object C_i is the ith entry in the Perron vector of $\mathbf{A}$. If $\mathbf{A}$ is inconsistent the Perron vector is still used to generate the rating scores for the objects $C_1, ..., C_n$. The consistency index ("closeness to consistency") is defined as

$$\frac{\rho(\mathbf{A}) - n}{n-1},$$

with $\rho(\mathbf{A})$ being the spectral radius and consequently the dominant eigenvalue of $\mathbf{A}$. An application of the Analytic Hierarchy Process to the problem of creating a best method for admissions of students to colleges was done by Barbeau[9].

In a later publication Saaty addressed the issue of rank order preservation. He examined three methods for computing the rating vector each using the reciprocal matrix $\mathbf{A}$. The models are the eigenvector method (EM), the least squares method (LSM) and the logarithmic least squares method (LLSM). The vector $\mathbf{x}$ containing the rating scores can in fact be used to approximate entries of $\mathbf{A}$, i.e., $a_{ij} \approx x_i/x_j$. When matrix $\mathbf{A}$ is consistent ($a_{ij} = a_{ik}a_{kj}$ for all i, j, k) then all three methods produce the same rating vector $\mathbf{x}$. However, when $\mathbf{A}$ is inconsistent Saaty claimed that the EM is the best at preserving the relative order of the values in the resulting rating vectors. Saaty defines rank preservation as follows. A method of solution is said to preserve rank weakly if $a_{ij} > 1$ implies $x_i \geq x_j$. A method of solution is said to preserve rank strongly if $a_{ik} \geq a_{jk}$ for all k implies $x_i \geq x_j$. He further states that entries of $\mathbf{A}^m$ can be used to measure the consistency relation between a given row and each column, which is the notion of relative dominance of one object over other objects. In the final theorem Saaty stated that the EM preserves this dominance whereas the other two methods can fail. Saaty revisited his Analytic Hierarchy Process in 1987 and observed that the ith value of the Perron vector $\mathbf{w}$, used for ranking, can also be viewed as a dominance of i over other objects along all k-walks (all k length paths from i in the directed graph representation of the preference set) as $k \to \infty$ [117]. More recently Gass showed how to determine the number of triads that exist within a reciprocal matrix using the results from tournaments and graph theory. The paper further provided linear programming procedures to find the triads [45].

The interest in ranking has been increasing in the past decade with the growth of the World Wide Web. With increasing numbers of web pages, the search for relevant information becomes difficult. One of the essential tasks of an internet search engine

(such as Google, Yahoo! Search, Ask.com etc.) is to be able to determine the relative quality of the web pages, i.e. ranks. A number of new ranking models, for example [18, 138], have a "transpose" relation to the preference relation in the Method of Paired Comparison. The web pages are identified as the objects, and the hyperlinks could be used to determine the order of preference. In many web ranking models the hyperlinks are likened to endorsements that are opposite of preference. For example a hyperlink from a web page P_i to a web page P_j can be interpreted as P_i endorses P_j or P_j is preferred to P_i. Note that to determine a preference between two web pages linked by a hyperlink, it makes more sense that the preferred web page is the one we move to, hence hyperlink from P_i to P_j would be $P_j \rightarrow P_i$. Hence the adjacency matrix constructed from a directed web graph in which the hyperlinks are endorsements is the transpose of the adjacency matrix constructed using the hyperlinks to define preferences.

Sports ranking can be related to the Method of Paired Comparison. In terms of sports an object is a team (or a player) and the game statistics can be used to determine the preferences of a judge. Each game usually has a number of measurements associated with each team, such as scores, total yards, or first downs in the National Football League (NFL). We refer to these measurements as *game statistics*. What constitutes a judge in sports can be vague. If during a season each team plays other teams no more then once, then for each game statistic we can construct a set of preferences. For example for a given game if team A scored more points than team B then $A \rightarrow B$. This indicates that the number of judges can be as many as the number of statistical categories. However, there are team sports such that during a single season some teams play each other more than once. This might be interpreted as the same judge making several sets of preferences, but separating which games are used for which set is left to the reader. Furthermore, the game statistics could be used to express endorsements of the winning teams by the losing teams. A balanced paired-comparison corresponds to a round-robin tournament. In a number of sports the set of preferences is often incomplete. In chapters 3 and 4 we introduce some of the most recent web ranking and sports ranking models. In both cases the hyperlinks and the game statistics are used to form the endorsements rather than preferences.

Chapter 3

Current Web Ranking Models

3.1 Introduction

The origins of web page ranking can be traced to the ranking methods for scientific journals via citations. Kleinberg [71] gives a well-rounded analysis of this fact. One of the major developments in ranking methods for scientific journals was a Markov chains based method proposed by Pinski and Narin [46, 107]. Both of the web ranking models, HITS developed by Kleinberg [71] and PageRank by Brin and Page [103, 18], presented below have elements that resemble the Pinski and Narin method. A thorough analysis of both of these models as well as several others can be found in [94]. The rating vector in the PageRank model is the left eigenvector of an irreducible stochastic matrix produced using the "endorsement" approach. The HITS model starts with an adjacency matrix also corresponding to the "endorsement" interpretation of hyperlinks.

3.2 PageRank Method

In 1998 future founders of the company Google, Larry Page and Sergey Brin, developed a ranking algorithm they named PageRank [18]. PageRank produces a rating score for each of the identified web pages on the World Wide Web. These rating scores then are used to rank the web pages. The efficiency of the algorithm is in the fact that all the ranking computations are done before and independently from the user's query. The basis of the algorithm lays in the elegant theory of graphs and the theory of finite Markov chains [93].

The following is a brief outline of PageRank. The initial step is to represent the world wide web using a directed graph where web pages are the nodes and hyperlinks between the web pages are the directed edges. The resulting directed graph is extremely large, as of 2005 Yahoo reported indexing 19.2 billion web pages [86]. To visualize the web-to-graph transition consider the following graph representation of a tiny web consisting of only six web pages, labeled 1 through 6. Each directed link represents a hyperlink from one web page to another.

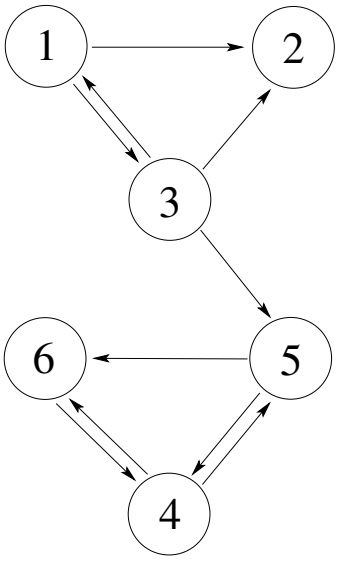

Figure 3.1: Tiny web example courtesy of Langville & Meyer [94].

The link structure of the web is thought to contain the "natural" ranks of the web pages. The premise of the algorithm is that "good" web pages will be pointed to by many "good" web pages. The next step in discovering the web page ranks is to use an adjacency matrix, call it $\mathbf{A}$, to summarize the web graph structure

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if there exists an edge from node } P_i \text{ to node } P_j, \\ 0 & \text{otherwise.} \end{cases}$$

To build the endorsement amount into the model let each directed edge from a web page P_i have a weight equal to $1/deg^{-}(i)$. The $deg^{-}(i)$ is the number of the directed edges originating in the node P_i, also known as an outdegree of the node. Thus all the endorsements (hyperlinks) of the web page P_i amount to 1. This new rescaled matrix is referred to as the hyperlink matrix $\mathbf{H}$

$$\mathbf{H}_{ij} = \begin{cases} 1/deg^{-}(i) & \text{if there exists an edge from node } P_i \text{ to node } P_j, \\ 0 & \text{otherwise.} \end{cases}$$

In order to fully utilize the Markov chains theory, Page and Brin propose two more adjustments to the matrix $\mathbf{H}$. The first deals with dangling nodes, vertices in the digraph that do not have outlinks. The other adjustment adds connectivity to the original web structure. Usually the original web structure exhibits both dangling nodes and the lack of connectivity. The modifications proposed by Brin and Page do change the original structure of the web and for further explanations and justifications of these changes see [94].

There is more than one approach to modifying the web graph to eliminate the dangling nodes. Many methods are based on adding one or more directed edges from node P_i to other node(s) or completely removing the dangling node along with the hyperlinks pointing to it. In the matrix $\mathbf{H}$ dangling node P_i corresponds to a zero row. A simple solution is to replace the ith row of $\mathbf{H}$ with a row $(1/n)\mathbf{e}^T$, where n is the total number of web pages and $\mathbf{e}$ is a vector of all 1's. The result is a stochastic matrix $\mathbf{S}$. A nonnegative matrix is *stochastic* if its row sums are equal to 1. Replacing the ith zero row of $\mathbf{H}$ with $(1/n)\mathbf{e}$ is equivalent to adding directed edges, each with a weight $1/n$, from the node P_i to P_i and every other node in the directed web graph. Let $\mathbf{a}$ be a vector such that a_i is zero if row i of $\mathbf{A}$ is nonzero and a_i is 1 otherwise.

$$a_i = \begin{cases} 1 & \text{if } \mathbf{H}_i^T = \mathbf{0}, \\ 0 & \text{otherwise.} \end{cases}$$

Then the matrix $\mathbf{S}$ can be written in terms of $\mathbf{a}$ and $\mathbf{H}$

$$\mathbf{S} = \mathbf{H} + (1/n)\mathbf{a}\mathbf{e}^T.$$

The last adjustment is to make the matrix $\mathbf{S}$ irreducible. A nonnegative matrix is called *irreducible* if and only if the corresponding directed graph $\mathcal{G}(\mathbf{A})$ is strongly connected. There are numerous ways of producing an irreducible matrix from $\mathbf{S}$. The simplest is to make a rank 1 update to $\mathbf{S}$ and ensure that the result stays stochastic by making the sum into a convex combination

$$\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)\mathbf{e}\mathbf{v}^T,$$

where $\mathbf{v}$ is a positive probability distribution vector and $0 < \alpha < 1$. The matrix $\mathbf{G}$ is called the Google matrix and the vector $\mathbf{v}$ is referred to as a personalization vector. A basic personalization vector is $(1/n)\mathbf{e}$, more sophisticated personalization vector choices and their significance to PageRank are discussed at length by Langville and Meyer [94]. According to Markov chains theory the dominant eigenvalue of the resulting matrix $\mathbf{G}$ is 1 and the corresponding left eigenvector, $\boldsymbol{\pi}$, is unique (up to a scalar multiplication) and positive

$$\boldsymbol{\pi}^T = \boldsymbol{\pi}^T \mathbf{G}.$$

The ith entry of $\boldsymbol{\pi}$ is the rating score of the ith web page. Another desirable property which the matrix $\mathbf{G}$ can have is primitivity. A nonnegative irreducible matrix is *primitive* if it has only one eigenvalue on its spectral circle. Due to the size of the matrix $\mathbf{G}$, computation of the rank vector $\boldsymbol{\pi}$ is a daunting task [98]. Primitivity of the matrix $\mathbf{G}$ ensures the convergence of the simple and effective method for computing the dominant eigenpair, the power method. The convergence of the power method is also affected by the value of α. It is reported that Google originally used $\alpha = 0.85$ [18]. Assign ranks to the web pages according to the descending order of the respective rating values.

Summary of the PageRank model:

1. Form an adjacency matrix $\mathbf{A}$

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if there is a link from web page } P_i \text{ to web page } P_j, \\ 0 & \text{otherwise.} \end{cases}$$

2. Form the hyperlink matrix $\mathbf{H}$

$$\mathbf{H}_{ij} = \begin{cases} 1/\sum_{k=1}^{n} \mathbf{A}_{ik} & \text{if there is a link from } P_i \text{ to } P_j, \\ 0 & \text{otherwise.} \end{cases}$$

3. Form the Google matrix $\mathbf{G}$

$$\mathbf{G} = \alpha[\mathbf{H} + (1/n)\mathbf{a}\mathbf{e}^T] + (1-\alpha)\mathbf{e}\mathbf{v}^T,$$

where $\mathbf{a}$ is the vertical vector such that $\mathbf{a}_i$ is 1 if row i of $\mathbf{H}$ is zero and $\mathbf{a}_i$ is 0 otherwise, $\mathbf{e}$ is a vector of all 1's, $0 < \alpha < 1$, and $\mathbf{v} > 0$ is a probability distribution vector ($\mathbf{v} \geq 0$ and sums to 1).

4. The rating vector $\boldsymbol{\pi}$ is a positive left eigenvector of $\mathbf{G}$ corresponding to eigenvalue 1, i.e. solve

$$\boldsymbol{\pi}^T = \boldsymbol{\pi}^T \mathbf{G}.$$

Use rating scores in $\boldsymbol{\pi}$ to rank web pages.

3.3 Hyperlink-Induced Topic Search model (HITS)

The ranking algorithm HITS was created by Jon Kleinberg in 1998 also for web page ranking [71]. According to HITS a web page P_i has two rating scores associated with it: an authority, a_i, and a hub h_i. The World Wide Web is again represented as a directed graph with web pages being nodes and hyperlinks as directed edges. The rating scores are computed as

$$a_i = l_{1i}h_1 + ... + l_{ni}h_n,$$
$$h_i = l_{i1}a_1 + ... + l_{in}a_n,$$

where $l_{ij} = 1$ if P_i has a hyperlink to P_j and $l_{ij} = 0$ otherwise. Thus the authority rating score of a web page P_i depends on the hub rating score of the web pages endorsing P_i,

and the hub rating score of the web page P_i depends on the authority rating scores of the web pages endorsed by P_i. The process of computing authority and hub ratings is therefore iterative and can be written using the adjacency matrix $\mathbf{L}$ of the directed web graph

$$\begin{aligned}\mathbf{a}^{(k)} &= \mathbf{L}^T\mathbf{h}^{(k-1)},\\ \mathbf{h}^{(k)} &= \mathbf{L}\mathbf{a}^{(k)},\end{aligned}$$

which simplify to

$$\begin{aligned}\mathbf{a}^{(k)} &= \mathbf{L}^T\mathbf{L}\mathbf{a}^{(k-1)},\\ \mathbf{h}^{(k)} &= \mathbf{L}\mathbf{L}^T\mathbf{h}^{(k-1)}.\end{aligned} \tag{3.1}$$

Matrices $\mathbf{L}^T\mathbf{L}$ and $\mathbf{L}\mathbf{L}^T$ are nonnegative and provided that they are primitive then equations 3.1 converge to eigenvectors corresponding to the dominant eigenvalue of each matrix. Equations 3.1 are in fact the power method, an iterative technique for computing the dominant eigenpair. However, if the matrices $\mathbf{L}^T\mathbf{L}$ and $\mathbf{L}\mathbf{L}^T$ are irreducible but not primitive the power method cannot be used to solve for the eigenvectors although the dominant eigenvectors of $\mathbf{L}^T\mathbf{L}$ and $\mathbf{L}\mathbf{L}^T$ exist and are unique (up to multiplication by a scalar).

Summary of the HITS model:

1. Form an adjacency matrix $\mathbf{L}$.

   $$\mathbf{L}_{ij} = \begin{cases} 1 & \text{if there is a link from web page } P_i \text{ to web page } P_j,\\ 0 & \text{otherwise.}\end{cases}$$

2. Form $\mathbf{L}^T\mathbf{L}$ and $\mathbf{L}\mathbf{L}^T$.

3. The authority ratings vector $\mathbf{a}$ is the dominant eigenvector of $\mathbf{L}^T\mathbf{L}$ and the hub ratings vector $\mathbf{H}$ is the dominant eigenvector of $\mathbf{L}\mathbf{L}^T$. Use ratings to assign ranks to the web pages.

Chapter 4

Current Popular Sports Ranking Models

4.1 Introduction

Many sports ranking models make use of graph and matrix theory to generate the ratings for the sports teams. The models relying on the Perron-Frobenius Theorem include [66]. The sports ranking models that utilize Markov chains theory are [21, 49, 76, 109], etc. Another type is the Sinkhorn-Knopp based ranking models such as [125] and Offense-Defense Model (ODM) [50]. An example of a direct application of paired comparison to obtaining ratings is [3]. An example of a sports ranking model using inverse equal paths was published by Hochbaum [57]. The Massey ranking model is an example of linear regression [90] whereas ratings produced by the Colley Matrix method [28] are the solution to a linear system. The following sections present details of the three different ranking models [28, 66, 90]. These three models were chosen due to their current popularity and ease of implementation. Both the Colley Matrix method and the Massey Least Squares Ratings are used to compute the computer average for the Bowl Championship Series. In this thesis we used the last publicly known ranking methods presented by Colley and Massey.

4.2 Small Example

In this section we introduce a small example consisting of a few games from the 2007-2008 National Football League regular season. This example will be used throughout to illustrate various ranking models. The example is summarized in Table 4.1

Table 4.1: Select NFL games from the 2007-2008 season

Team	Score	Team	Score
Carolina	16	New Orleans	13
Dallas	38	Philadelphia	17
Dallas	28	Washington	23
Houston	34	Carolina	21
Houston	23	New Orleans	10
New Orleans	31	Carolina	6
Philadelphia	33	Washington	25
Philadelphia	38	New Orleans	23
Washington	27	Dallas	6
Washington	20	Philadelphia	12

4.3 Colley Matrix Model

Wesley Colley created the Colley Method in 2002 for computing ratings of teams in competitive sports. The only information used by this model are wins, losses, and game counts for each team, assuming no ties. The resulting ratings are bias free, that is a team can not boost its ranking by running up points since they are not included in the computation.

Colley's ranking algorithm utilizes an idea from probability, known as Laplace's rule of succession [110, p. 108]. It can be stated in terms of football competition as follows:

Consider picking an NFL team T_i from k available teams (currently 32 teams) at random. Suppose that the first n_i games played by team T_i result in w_i wins, independent of each other and with probability i/k. Then the probability that the $(n_i + 1)th$ game is a win is $(w_i + 1)/(n_i + 2)$.

Assuming that there are no ties, define the rating of the team T_i to be

$$r_i = \frac{w_i + 1}{n_i + 2}, \tag{4.1}$$

where w_i is the number of wins of team T_i and n_i is the total number of games played by a team T_i. To add the strength of schedule into the rating equation Colley rewrites w_i in the following fashion. Let l_i designate the number of losses by team T_i then $n_i = w_i + l_i$ and

$$w_i = \frac{w_i + n_i - l_i}{2} = \frac{w_i - l_i}{2} + \frac{n_i}{2}. \tag{4.2}$$

Colley proposed that the second term of the preceding equation, $n_i/2$, incorporates the ratings of team T_i's opponents. In the simplest case the ratings of teams that lost to team T_i are all the same $1/2$

$$\frac{n_i}{2} = \sum_{j=1}^{n_i} \frac{1}{2}.$$

To generalize, the summation is rewritten as

$$\frac{n_i}{2} = \sum_{j=1}^{n_i} r_j^{(i)}, \tag{4.3}$$

where $r_j^{(i)}$ is the rating of the team T_i opponent, team T_j. Expression 4.3 is what Colley referred to as the *adjustment for strength of schedule*. Substitute 4.3 into 4.2 and the wins of team T_i with incorporated strength of schedule are

$$w_i = \frac{w_i - l_i}{2} + \sum_{j=1}^{n_i} r_j^{(i)}. \tag{4.4}$$

Substitute the rewritten wins (4.4) into the original definition of the team T_i rating (4.1)

$$r_i = \frac{w_i + 1}{n_i + 2} = \frac{[(w_i - l_i)/2 + \sum\limits_{j=1}^{n_i} r_j^{(i)}] + 1}{n_i + 2}.$$

After simplification and transfer of the terms with ratings to the left of the equality sign

$$(2 + n_i)r_i - \sum_{j=1}^{n_i} r_j^{(i)} = 1 + (w_i - l, i)/2. \tag{4.5}$$

Provided that there are n teams the system of equations of the form 4.5 can be rewritten using matrix-vector multiplication

$$\mathbf{C}_{n\times n}\mathbf{r} = \mathbf{b},$$

where

$$\mathbf{C}_{ij} = \begin{cases} -n_{ji} & \text{if} \quad i \neq j, \\ 2 + n_i & \text{if} \quad i = j, \end{cases}$$
$$b_i = 1 + (w_i - l_i)/2,$$

given that n_{ji} is the number of times team T_i played team T_j. The matrix $\mathbf{C}$ is called the *Colley matrix*.

Theorem 4.3.1 *The vector* $\mathbf{r}$*, solution to the* $\mathbf{Cr} = \mathbf{b}$ *exists, is unique, and nonnegative.*

Proof. The Colley matrix $\mathbf{C}$ is diagonally dominant

$$|C_{ii}| = |2 + n_i| > \sum_{j=1, j\neq i}^{n} |C_{ij}| = n_i,$$

therefore Gerschgorin's theorem [93, p.498] guarantees that $0 \notin \rho(\mathbf{C})$, thus ensuring that $\mathbf{C}$ is nonsingular. Furthermore, since $\mathbf{C}$ is a symmetric matrix all its eigenvalues are real. Hence also by Gerschgorin's theorem all eigenvalues of $\mathbf{C}$ must be real and positive making $\mathbf{C}$ a Stieltjes matrix, which is a symmetric nonsingular M-matrix. A matrix $\mathbf{A}$ is called a nonsingular *M-matrix* if $\mathbf{A} \in \{\mathbf{M} = (M_{ij}) \in \mathbb{R}^{n\times n} | M_{ij} \leq 0, i \neq j\}$ and every eigenvalue of $\mathbf{A}$ is positive (in fact Berman and Plemmons list 50 conditions equivalent to $\mathbf{A}$ being a nonsingular M-matrix) [11, p.134-137]. Thus the matrix $\mathbf{C}$ is inverse-positive, that is $\mathbf{C}^{-1} \geq 0$. This guarantees that the vector containing ratings is nonnegative, $\mathbf{r} = \mathbf{C}^{-1}\mathbf{b} \geq 0$.

To illustrate the model we use our NFL example 4.1. The corresponding Colley matrix and $\mathbf{b}$ vector are

$$\mathbf{C} = \begin{pmatrix} 5 & 0 & -1 & -2 & 0 & 0 \\ 0 & 5 & 0 & 0 & -1 & -2 \\ -1 & 0 & 4 & -1 & 0 & 0 \\ -2 & 0 & -1 & 6 & -1 & 0 \\ 0 & -1 & 0 & -1 & 6 & -2 \\ 0 & -2 & 0 & 0 & -2 & 6 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 1/2 \\ 3/2 \\ 2 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

The resulting rating scores vector for our example is

$$\mathbf{r} \approx \begin{pmatrix} 0.3597 & 0.616 & 0.6687 & 0.3149 & 0.5015 & 0.5392 \end{pmatrix}^T.$$

According to these rating scores the list of ranked teams (from first to last) is

Houston Dallas Washington Philadelphia Carolina New Orleans.

Summary of the Colley Matrix Model:

1. Form the Colley matrix $\mathbf{C}$

$$\mathbf{C}_{ij} = \begin{cases} -n_{ji} & \text{if} \quad i \neq j, \\ 2 + n_i & \text{if} \quad i = j, \end{cases}$$

where n_i is the total number of games played by team T_i and n_{ji} is the number of times T_i played T_j.

2. Form the vector $\mathbf{b}$

$$b_i = 1 + (w_i - l_i)/2,$$

where w_i is the number of wins by team T_i and l_i is the number of loses by team T_i.

3. Solve

$$\mathbf{Cr} = \mathbf{b},$$

the vector $\mathbf{r}$ contains rating scores of each team. Use the ratings to assign the ranks.

4.4 Keener Ranking Model

James P. Keener proposed his ranking method based on the theory of nonnegative matrices in 1993 [66]. Keener's approach forms a smoothed score matrix, and the ith entry of the Keener matrix depends on the score ratio that utilizes Laplace's rule of succession

$$\frac{S_{ij} + 1}{S_{ij} + S_{ji} + 2}, \tag{4.6}$$

where S_{ij} is the score produced by team T_i against team T_j. The use of Laplace's rule of succession ratio in 4.6 is to make sure that in case one team gets a score of 0 the other team does not get the entirety of the credit. Furthermore, 4.6 can be thought of as a probability of team T_i beating T_j in their next encounter.

Since Keener uses game scores to compute ratings the resulting algorithm is susceptible to some manipulation (or bias), e.g. the teams can affect their ranking by running up scores. To minimize the possibility of manipulation Keener created a score "smoothing" function. He suggested several possibilities for smoothing functions, but the one with the best empirical results is

$$h(x) = 1/2 + (1/2)\text{sgn}(x - 1/2)\sqrt{|2x - 1|}.$$

The ith entry in the Keener matrix, $\mathbf{K}$, is then

$$\mathbf{K}_{ij} = \begin{cases} h\left(\dfrac{S_{ij}+1}{S_{ij}+S_{ji}+2}\right) & \text{if teams } T_i \text{ and } T_j \text{ played each other} \\ 0 & \text{otherwise.} \end{cases}$$

The result is a nonnegative matrix that is irreducible provided that there have been enough games played among the teams. Keener made use of the properties of existence and uniqueness of the Perron vector guaranteed by Perron-Frobenius Theorem [93, p.673]. Provided that $\mathbf{K}$ is irreducible, the corresponding Perron vector contains positive ratings of all the teams.

Some teams in our small NFL example 4.1 play each other more than once. In this case we add the corresponding game scores for each of the games between the same two teams to produce one cumulative score. For example, Dallas (Dal) and Washington (Was) played twice where Dallas won 28-23 and Washington won 27-6. The cumulative score between Dallas and Washington is therefore Washington won 60-34. This is the simplest approach for dealing with multiple games between two teams. The Keener matrix for our example is

$$\mathbf{K} \approx \begin{pmatrix} 0 & 0 & 0.2612 & 0.2156 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8035 & 0.2843 \\ 0.7388 & 0 & 0 & 0.8047 & 0 & 0 \\ 0.7844 & 0 & 0.1953 & 0 & 0.256 & 0 \\ 0 & 0.1965 & 0 & 0.744 & 0 & 0.5 \\ 0 & 0.7157 & 0 & 0 & 0.5 & 0 \end{pmatrix}.$$

The corresponding ratings vector is

$$\mathbf{r} \approx \begin{pmatrix} 0.1044 & 0.5252 & 0.2437 & 0.2375 & 0.5158 & 0.5757 \end{pmatrix}^T.$$

Sorting our six teams according to their rating scores we obtain the following ranking from first to last

Washington Dallas Philadelphia Houson New Orleans Carolina.

Summary of the Keener model:

1. Form the Keener matrix $\mathbf{K}$

$$\mathbf{K}_{ij} = \begin{cases} h\left(\dfrac{S_{ij}+1}{S_{ij}+S_{ji}+2}\right) & \text{if team } T_i \text{ played team } T_j, \\ 0 & \text{otherwise,} \end{cases}$$

   where S_{ij} is number of points scored by team T_i against team T_j and

$$h(x) = 1/2 + (1/2)\text{sgn}(x - 1/2)\sqrt{|2x-1|}.$$

2. The positive vector $\mathbf{r}$ contains each team's rating scores and is the Perron vector of matrix $\mathbf{K}$, i.e. solve

$$\mathbf{Kr} = \lambda\mathbf{r},$$

   where λ is the spectral radius (dominant eigenvalue) of $\mathbf{K}$. Use rating scores in $\mathbf{r}$ to rank teams.

4.5 Massey Least Squares Ratings

A different type of ranking model has been suggested by Kenneth Massey in 1997 [90]. In his honors project Massey expresses the relationship between the team ratings and the margin of victory, a.k.a. point spread, via a system of linear equations. The least squares approach is then used to solve for the ratings. The initial assumption made by Massey is that the expected margin of victory of a game is directly proportional to the difference in the participants ratings. Let Y_i be the random variable representing the margin of victory for the kth game between two teams, T_i and T_j and r_i is the rating of the team T_i assume that

$$E(Y_k) = r_i - r_j.$$

The observed margin of victory for the game k is y_k and

$$y_k = E(Y) + e_k = r_i - r_j + \varepsilon_k,$$

where ε_k is the error term for the game k. Let x_{ki} be an indicator variable for the outcome of the kth game for team T_i

$$x_{ki} = \begin{cases} 1 & \text{if team } T_i \text{ won } k\text{th game,} \\ -1 & \text{if team } T_i \text{ lost } k\text{th game,} \\ 0 & \text{otherwise.} \end{cases}$$

Therefore the observed margin of victory for game k, y_k, is

$$y_k = r_{ki} - r_{kj} = x_{k1}r_1 + ... + x_{ki}r_i + ... + x_{kj}r_j + ...x_{kn}r_n + \varepsilon_k.$$

If there have been m games among n teams then the resulting system of equations can be written as

$$\mathbf{y} = \mathbf{X}_{m\times n}\mathbf{r} + \boldsymbol{\varepsilon},$$

where $\mathbf{y}$ is the vector of observed margins of victory for the m games, $\mathbf{X}$ is the $m \times n$ matrix of indicator variables, $\mathbf{r}$ is the vector of ratings for the n teams, and $\boldsymbol{\varepsilon}$ is the error term.

The objective is to solve for the vector $\mathbf{r}$ to minimize the error, the natural choice being the least squares method. The problem lies in the fact that the matrix $\mathbf{X}$ does not have full rank since $\sum_{j=1}^{n} \mathbf{X}_j = \mathbf{0}$, that is the sum of the rows of $\mathbf{X}$ is $\mathbf{0}$. However, since every team plays at least once there are no zero columns in $\mathbf{X}$. Furthermore, the observation matrix $\mathbf{X}$ is *saturated*, i.e. the directed graph with n nodes corresponding to the teams and m edges corresponding to the games played is strongly connected. This ensures that $dim(N(\mathbf{X})) = 1$, i.e. the null space of $\mathbf{X}$ is then spanned by the vector $\mathbf{e} = (\ 1 \quad 1 \quad ... \quad 1\ )^T$.
Massey suggested several modifications to the system $\mathbf{X}^T\mathbf{X}\mathbf{r} = \mathbf{X}^T\mathbf{y}$ to force the uniqueness of the solution:

- Impose a condition on a solution $\mathbf{r}$, such as $\sum_{j=1}^{n} r_j = c$, a typical choice is $c = 0$. In this case the matrix $\mathbf{X}$ can be appended with a row of all 1's and the vector $\mathbf{y}$ with the value c. The effect on the system of normal equations is

$$\left[\begin{pmatrix} \mathbf{X}^T & \mathbf{e}^T \end{pmatrix} \begin{pmatrix} \mathbf{X} \\ \mathbf{e} \end{pmatrix} \right] \mathbf{r} = \begin{pmatrix} \mathbf{X}^T & \mathbf{e}^T \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ c \end{pmatrix},$$

$$\left[\mathbf{X}^T\mathbf{X} + \mathbf{e}^T\mathbf{e}\right]\mathbf{r} = \mathbf{X}^T\mathbf{y} + c\mathbf{e}^T.$$

- Replace one of the rows of $\mathbf{X}^T\mathbf{X}$ with $\mathbf{e} = (1, 1, ..., 1)$ and the corresponding entry in $\mathbf{X}^T\mathbf{y}$ with c.
- Eliminate one of the variables, k. Every other team's rating will be scaled relative to k. After the ratings are calculated, the desired scale, such as $\sum_{j=1}^{n} r_j = c$ can be achieved by simply adding an appropriate constant to each rating parameter.

It might be computationally prohibitive to use the matrix $\mathbf{X}$ due to its size. Note that the normal equations for the least squares solution is

$$\mathbf{X}^T\mathbf{X}\mathbf{r} = \mathbf{X}^T\mathbf{y},$$

such that

$$[\mathbf{X}^T\mathbf{X}]_{ij} = \begin{cases} -n_{j,i} & \text{if} \quad i \neq j, \\ n_i & \text{if} \quad i = j, \end{cases}$$

where n_i is the total number of games played by team T_i and $n_{j,i}$ is the number of times team T_i played team T_j and let $d_i = [\mathbf{X}^T\mathbf{y}]_i$ be the total difference in scores for team T_i.

To illustrate the elements of the Massey method consider the small NFL example 4.1 and observe that

$$\mathbf{X} = \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}, \qquad \mathbf{y} = \begin{pmatrix} 3 \\ 21 \\ 5 \\ 13 \\ 13 \\ 25 \\ 8 \\ 15 \\ 21 \\ 8 \end{pmatrix}.$$

Therefore

$$\mathbf{M} = \mathbf{X}^T\mathbf{X} = \begin{pmatrix} 3 & 0 & -1 & -2 & 0 & 0 \\ 0 & 3 & 0 & 0 & -1 & -2 \\ -1 & 0 & 2 & -1 & 0 & 0 \\ -2 & 0 & -1 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -2 \\ 0 & -2 & 0 & 0 & -2 & 4 \end{pmatrix}, \qquad \mathbf{d} = \mathbf{X}^T\mathbf{y} = \begin{pmatrix} -35 \\ 5 \\ 26 \\ -6 \\ -6 \\ 16 \end{pmatrix}.$$

To ensure that $\mathbf{Mr} = \mathbf{d}$ has a unique solution replace the last row of $\mathbf{M}$ with $\mathbf{e}$, vector of all 1's, and the corresponding entry in $\mathbf{d}$ with 0.

$$\mathbf{M} = \mathbf{X}^T\mathbf{X} = \begin{pmatrix} 3 & 0 & -1 & -2 & 0 & 0 \\ 0 & 3 & 0 & 0 & -1 & -2 \\ -1 & 0 & 2 & -1 & 0 & 0 \\ -2 & 0 & -1 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \qquad \mathbf{d} = \mathbf{X}^T\mathbf{y} = \begin{pmatrix} -35 \\ 5 \\ 26 \\ -6 \\ -6 \\ 0 \end{pmatrix}.$$

Then the unique solution to $\mathbf{Mr} = \mathbf{d}$ is

$$\mathbf{r} \approx \begin{pmatrix} -18.5583 & 11.7417 & -1.1583 & -9.7583 & 5.2417 & 12.4917 \end{pmatrix}^T.$$

Sort the six teams according to their rating scores from first to last

Washington Dallas Philadelphia Houson New Orleans Carolina.

Summary of the Massey Least Squares Ratings model:

1. Form the Massey matrix $\mathbf{M} = \mathbf{X}^T\mathbf{X}$

$$\mathbf{M}_{ij} = \begin{cases} -n_{j,i} & \text{if} \quad i \neq j, \\ n_i & \text{if} \quad i = j, \end{cases}$$

   where n_i is the total number of games played by team T_i and $n_{j,i}$ is the number of times team T_i played team T_j.

2. Form the vector $\mathbf{d} = \mathbf{X}^T\mathbf{y}$, d_i =total difference in scores for team T_i.

3. Force $\mathbf{M}$ to have full rank by doing ONE of the following

   a. Replace $\mathbf{M}$ with $\mathbf{M} + \mathbf{e}^T\mathbf{e}$, where $\mathbf{e}$ is a vector of all 1's.

   b. Replace one of the rows of $\mathbf{M}$ with $\mathbf{e}$ and the corresponding entry in $\mathbf{d}$ with c.

4. The ratings vector $\mathbf{r}$ is the solution to the system resulting from the previous step (3). Use ratings in $\mathbf{r}$ to rank teams.

Chapter 5

The Offense-Defense Model (ODM)

5.1 Introduction

This chapter is centered on the development of a non-linear iterative ranking model called the Offense-Defense Model. The convergence of this model is based on the theory of a special type of matrix balancing, the Sinkhorn-Knopp balancing method. The following sections include a brief introduction to matrix balancing and the historical development of the Sinkhorn-Knopp method and its convergence.

5.2 Matrix Balancing

In general a square matrix is *balanced* if it satisfies a given set of linear restrictions on its columns and rows [120]. The linear restrictions may depend on a given problem's requirements. The matrix balancing can be divided into two basic categories [120]

Problem 1. *Given an $m \times n$ nonnegative matrix* $\mathbf{A}$ *and positive vectors* $\mathbf{u}$ *and* $\mathbf{v}$ *of dimensions m and n, respectively, determine a nearby $m \times n$ nonnegative matrix* $\mathbf{X}$ *such that*

$$\sum_{j=1}^{n} X_{ij} = u_i \qquad for \quad i = 1, 2, ..., n,$$

$$\sum_{i=1}^{n} X_{ij} = v_j \qquad for \quad j = 1, 2, ..., n,$$

and $X_{ij} > 0$ *only if* $A_{ij} > 0$.

Problem 2. *Given an* $m \times n$ *nonnegative matrix* $\mathbf{A}$ *determine a nearby* $m \times n$ *nonnegative matrix* $\mathbf{X}$ *such that*

$$\sum_{j=1}^{n} x_{ij} = \sum_{j=1}^{n} x_{ji} \qquad for \quad i = 1, 2, ..., n,$$

and $X_{ij} > 0$ *only if* $A_{ij} > 0$.

What constitutes a *nearby* matrix is determined by the balancing method applied, one being a diagonal equivalence. Problem 1 has been referred to as an (r, c)-scaling [111].

The resulting matrix $\mathbf{X}$ in Problem 2 is frequently called a *balanced matrix* [51, 62, 118] or a *line-sum-symmetric* matrix [33]. Matrix balancing using two different diagonal matrices to scale $\mathbf{A}$ has been called the equivalence scaling [118, 112] and is used to address Problem 1. Whereas matrix balancing via similarity transformation has been referred to as symmetric scaling [118, 112] and is a method for solving Problem 2. Osborne initially proved the convergence of matrix balancing via similarity transformations for irreducible matrices [102]. Later Grad generalized the convergence of matrix balancing that uses similarity transformations to real $n \times n$ matrices [51]. Both Grad and Osborne used balancing of matrices to improve the computational accuracy of the eigenvalues. The same year Hartfiel showed that if an $n \times n$ matrix $\mathbf{A}$ is irreducible then there exists a unique diagonal matrix $\mathbf{D}$ with positive main diagonal such that $\mathbf{DAD}^{-1}$ is a balanced matrix [53]. Kalantari et al. proved that if a nonnegative square matrix $\mathbf{A}$ is irreducible then we can compute a positive diagonal matrix $\mathbf{X} = diag(e^{w_1}, ..., e^{w_n})$ so that $\mathbf{XAX}^{-1}$ is balanced to relative error of ε and absolute error of $e\sigma\varepsilon$. Set $A_{min} = \min A_{ij}$, $\sigma = \sum A_{ij}$, and $\nu = A_{min}/\sigma$, for any given accuracy $\varepsilon \in (0, 1)$.

Some of the generalizations of matrix balancing, such as truncated matrix balancing has been developed by [118]. The technique of analyzing matrix scaling

problems by studying equivalent optimization problems has been done in [16, 33, 36, 77, 87]. The relationship between entropy minimization (maximization) and matrix scaling has been studied by [23, 34, 36, 37]. Matrix balancing applications can be found in economics, transportation, statistics, demography, stochastic modeling [74, 83, 120] and eigenvalue computation [51, 102, 105].

5.3 Historical Note on the Sinkhorn-Knopp Matrix Balancing

A special type of matrix balancing converts a nonnegative matrix $\mathbf{A}$ into a doubly stochastic matrix. This corresponds to Problem 1, such that $\mathbf{u} = \mathbf{e}$, and $\mathbf{v} = \mathbf{e}$. To restate the problem

Given an $m \times n$ nonnegative matrix $\mathbf{A}$ determine a nearby $m \times n$ nonnegative matrix $\mathbf{X}$ such that

$$\sum_{j=1}^{n} X_{ij} = 1 \qquad for \quad i = 1, 2, ..., n$$

$$\sum_{i=1}^{n} X_{ij} = 1 \qquad for \quad j = 1, 2, ..., n$$

and $X_{ij} > 0$ only if $X_{ij} > 0$

Applications of the problem include estimation of transition probabilities in Markov chains, network optimization, planning of telephone traffic, transportation, social accounting, matrix preconditioning, and algebraic image reconstruction, see Sinkhorn [121], Bacharach [4], Schneider and Zenios [120], Rothblum an Schneider [111].

Several methods for solving this problem have been investigated so far. A method of alternative scaling of rows and columns of the matrix was used as early as 1937 by Kruithof [75], which he calls the *method of twin factors*. Later in 1940 Deming and Stephan called this the *method of iterative proportions*. Marcus and Newman [85] in 1961 showed that if $\mathbf{A}$ is symmetric and positive then there exists a diagonal matrix $\mathbf{D}$ with positive main diagonal such that $\mathbf{B} = \mathbf{DAD}$ is doubly stochastic. In 1964 Sinkhorn generalized the Marcus and Newman result to any square positive matrix $\mathbf{A}$ [121]. Sinkhorn then used the iterative renormalization of rows and columns of a positive

matrix **A** to demonstrate constructively the existence of the diagonal scaling matrices **D** and **E**. This result is summarized in the following Theorem

Theorem 5.3.1 *To a given square matrix* **A** *with strictly positive elements there corresponds exactly one doubly stochastic matrix* **B** *which can be expressed in the form* **B**=**DAE**, *where* **D** *and* **E** *are diagonal matrices with strictly positive diagonal elements. The matrices* **D** *and* **E** *are themselves unique up to nonzero scalar multiplication.*

An alternative proof of Theorem 5.3.1 was published in 1998 by Borobia [14] and an alternative existence proof by Menon [91] in 1967. Later (1967) Sinkhorn generalized Theorem 5.3.1 and proved the convergence of the matrix scaling method for any given $m \times n$ positive matrix **A** [122]. In this case the resulting diagonally equivalent matrix $\mathbf{B} = \mathbf{DAE}$ will have row and column sums equal to some specified positive numbers. Sinkhorn also proved the existence and uniqueness of the resulting diagonal scaling matrices **D** and **E** for this generalized case.

A *diagonal* of the matrix **A** is a set of elements $\{a_{1,\sigma(1)}, ..., a_{n,\sigma(n)}\}$,where σ is a permutation of $\{1, ..., n\}$. A matrix **A** is *fully indecomposible* if if there do not exist permutation matrices **P** and **Q** such that

$$\mathbf{PAQ} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{B} \\ \mathbf{0} & \mathbf{A}_2 \end{pmatrix}$$

where $\mathbf{A}_1$ is a $k \times k$ matrix, $1 \leq k \leq n-1$. A nonnegative square matrix **A** is said to have *total support* if $\mathbf{A} \neq 0$ and if every positive element of **A** lies on a positive diagonal. Perfect and Minsky showed in 1965 that for every fully indecomposable square nonnegative matrix there exists a doubly stochastic matrix with the same zero pattern [106]. Furthermore Perfect and Minsky proved that being a fully indecomposable matrix **A** is equivalent to having total support.

In 1966 Brualdi et al. generalized the result of Theorem 5.3.1 to fully indecomposable nonnegative matrices, their result stated that the scaling matrices **D** and **E** exist if **A** is a direct sum of fully indecomposable matrices [19]. Almost simultaneously Sinkhorn and Knopp published their proof for the existence of the scaling matrices **R** and **C** under the assumption that a nonnegative matrix **A** has total support [123].

The following theorem 5.3.2 by Sinkhorn and Knopp states the conditions for convergence of the iterative renormalization scaling method and existence of the scaling

matrices when **A** is nonnegative.

Theorem 5.3.2 *(Sinkhorn-Knopp) For each nonnegative matrix* **A** *with total support there exists a unique doubly stochastic matrix* **S** *of the form* **DAE** *where* **D** *and* **E** *are unique (up to a scalar multiple) diagonal matrices with positive main diagonal. Matrices* **D** *and* **E** *are obtained by alternately normalizing columns and rows of* **A**.

If a matrix **A** *has support (has at least one positive diagonal), then the iterative procedure of alternately normalizing columns and rows of* **A** *converges. However, the sequences of the diagonal matrices, produced for this normalization, do not necessarily converge.*

Henceforth many subsequent papers in the mathematical community refer to the iterative process of alternately normalizing columns and rows of a matrix as a Sinkhorn-Knopp algorithm (method).

An optimization procedure for computing **D** and **E** was proposed by Marshall and Olkin in 1968 [87]. Matrices **D** and **E** are obtained by minimizing $\mathbf{x}^T\mathbf{A}\mathbf{y}$ subject to the constraints $\prod x_i = \prod y_j = 1$. Another procedure was formulated by Djokovic in 1970 [32]. In his paper Djokovic computes **D** and **E** by minimizing

$$f(x_1, ..., x_n) = \frac{\prod_{k=1}^{n} \left(\sum_{i=1}^{n} A_{ki} x_i \right)}{\prod_{k=1}^{n} x_k},$$

subject to the constraints $x_k > 0$, $k = 1, ..., n$, and $\sum_{k=1}^{n} x_k = 1$. Djokovic proved that the function $f(x_1, ..., x_n)$ achieves the minimum at **c** in the specified region. Diagonal matrices **D** and **E** are then formed using **c**. Other developments in optimization techniques for matrix scaling can be found in [120, 6], although optimization methods tend to have a complex implementation.

In 1982 Parlett and Landis proposed three new algorithms for iteratively computing the scaling matrices **D** and **E** [104]. All three algorithms perform some selective row or column or row and column scaling at each iteration, reducing the standard deviation between row sums of the iterates.

Scaling of multidimensional positive matrices was solved independently by Bapat [8] and Raghavan [108]. Bapat and Raghavan later jointly extended their approach to nonnegative multidimensional matrices [7]. Franklin and Lorenz (1989) provided an

alternative proof to the Bapat and Raghavan theorem on nonnegative multidimensional matrix scaling, with Theorems of Sinkhorn [121] and of Brualdi et al. [19] derived as corollaries [42].

Sinkhorn [122] has shown that for a positive matrix $\mathbf{A}$ the iterative normalization procedure converges geometrically. In 1991 Soules [126] proved that the convergence is geometric if a nonnegative matrix $\mathbf{A}$ has total support. Furthermore in 1993 Achilles [1] proved that if convergence of the iterative normalization is geometric then matrix $\mathbf{A}$ has total support.

If the matrix $\mathbf{A}$ has support which is not total, then there is a positive entry in $\mathbf{A}$ that converges to 0. In such a case, Sinkhorn and Knopp concluded that the iteration on $\mathbf{A} = (A_{ij})$ converges to the same limit as does the iteration on the matrix $\bar{\mathbf{A}} = (\bar{A}_{ij})$, where $\bar{A}_{ij} = A_{ij}$ if A_{ij} lies on at least one positive diagonal and $\bar{A}_{ij} = 0$ otherwise [123]. Soules proved in 1989 that Sinkhorn balancing always converges linearly, provided the starting matrix has total support [126]. Over the years numerous algorithms for computing the Sinkhorn-Knopp matrix balancing (matrix scaling) have been proposed, recent ones include [83, 6, 74].

5.4 Deriving the Offense-Defense Model.

For a set of teams engaged in a competitive sport, the objective of this section is to present a model for rating the overall strength of each team relative to the others. While there are numerous factors that might be taken into account, our approach is to characterize "strength" by combining each team's relative offensive and defensive prowess in a non-linear fashion.

To compute offensive and defensive ratings we start with the assumption that larger offensive ratings correspond to greater offensive strength, i.e. the capability of producing larger scores. On the other hand, smaller defensive ratings will correspond to greater defensive strength, i.e., a low defensive rating indicates that it is hard for the opposition to run up a large score.

If teams T_i and T_j compete, then let $\mathbf{A} = [A_{ij}]$ be such that A_{ij} is the score that team T_j generated against team T_i (set $A_{ij} = 0$ if the two teams did not play each other). Alternatively, A_{ij} can be thought of as the number of points that team T_i held team T_j to. In other words, depending on how it is viewed, A_{ij} simultaneously reflects

relative offensive and defensive capability. To utilize this feature we define the *offensive rating* of team T_j to be the combination

$$o_j = A_{1j}(1/d_1) + ... + A_{nj}(1/d_n),$$

where d_i is the *defensive rating* of team T_i that is defined to be

$$d_i = A_{i1}(1/o_1) + ... + A_{in}(1/o_n).$$

Since o_j's and d_i's are interdependent, these values will have to be determined by a successive refinement technique that is described below. For example, consider a league consisting of three teams, called team 1, team 2, and team 3. Suppose that initially team 1 has the strongest offense and defense, team 2 has next strongest offense and defense, and team 3 has the worst offense and defense. Then $o_1 \geq o_2 \geq o_3$, and $d_1 \leq d_2 \leq d_3$. Suppose that some initial estimates of these quantities are made (or guessed). Given that each team played every other team exactly once let us examine refined offensive and defensive ratings for team 3. The new offensive rating for team 3 is

$$o_3 = A_{13}(1/d_1) + A_{23}(1/d_2).$$

Since team 1 has strong defense then d_1 is relatively small and therefore team 3's offense will be rewarded for scoring high against a strong opponent, i.e., points A_{13} produced against team 1 will be divided by a smaller number thus boosting the offensive rating of team 3.

The new defensive rating of team 3 is

$$d_3 = A_{31}(1/o_1) + A_{32}(1/o_2).$$

Team 1 has the strongest offense, so the defense of team 3 is less penalized for allowing team 1 to score more points. On the other hand, team 2's offense is less impressive, and therefore defense 3 must hold offense 2 to less points to avoid increase in d_3 rating.

This intuition leads to the following general rule for successive refinement. Given $\mathbf{A} = [A_{ij}]$, initialize $d_i = 1$ for all i so that $\mathbf{d}^{(0)} = (\ 1 \quad 1 \quad ... \quad 1\ )^T$. Define $\mathbf{o}^{(1)} = \mathbf{A}^T(1/\mathbf{d}^{(0)})$, where $1/\mathbf{d}^{(0)}$ denotes the column vector $1/\mathbf{d}^{(0)} = (\ 1/d_1 \quad 1/d_2 \quad ... \quad 1/d_n\ )^T$. Now successively refine the defensive and offensive values by the following iterative procedure

$$\begin{aligned} \mathbf{o}^{(k)} &= \mathbf{A}^T \frac{1}{\mathbf{d}^{(k-1)}}, \\ \mathbf{d}^{(k)} &= \mathbf{A} \frac{1}{\mathbf{o}^{(k)}}. \end{aligned} \tag{5.1}$$

5.5 The Offence-Defense Aggregation

The Offense-Defense model assigns two rating scores, offensive and defensive, to each team. In most ranking applications the preference is to have a single rating score for each team. This can be accomplished by aggregating the corresponding offensive and defensive rating scores.

Rank aggregation is a function which uses several ratings (or ranks) obtained using various models as an input to produce a single rating (or rank) of each team as an output. The simplest aggregation function that can be applied to the Offense-Defense model is $r_i = o_i/d_i$, i.e., the overall rating score of team T_i is its offensive rating divided by its defensive rating. This allows us to retain the "large value is better" interpretation of the ratings. This rating score is a reciprocal of the Sinkhorn rating score proposed by Smith [125].

5.6 Convergence of the Offense-Defense Model

Equations (5.1) for computing the offensive and defensive ratings are in fact equivalent to a row-column scaling of the matrix $\mathbf{A}$. That is, provided that the iterative procedure converges, the entries of $1/\mathbf{o}$ normalize columns of $\mathbf{A}$ and entries of $1/\mathbf{d}$ normalize rows of $\mathbf{A}$ so that $\mathbf{A}(1/\mathbf{o}) = \mathbf{e}$ and $\mathbf{A}^T(1/\mathbf{d}) = \mathbf{e}$, where $\mathbf{e}$ is a vector of all 1's. In other words, $\mathfrak{D}(1/\mathbf{d})\mathbf{A}\mathfrak{D}(1/\mathbf{o})$ is a doubly stochastic matrix where $\mathfrak{D}(\mathbf{x})$ denotes the diagonal matrix

$$\mathfrak{D}(\mathbf{x}) = \begin{pmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_n \end{pmatrix}.$$

Scaling of a square nonnegative matrix $\mathbf{A}$ so that row and column sums of $\mathbf{A}$ are 1 is a special case of matrix balancing. In a general case the rows and columns of $\mathbf{A}$ do not

have to add to the same scalar, see [120] for details.

The convergence of the Offense-Defense model is guaranteed by the Sinkhorn-Knopp Theorem [123]. In order for the set of equations (5.1) to converge, the matrix $\mathbf{A}$ has to have total support. Theorem 5.3.2 states the conditions for convergence of the scaling method if $\mathbf{A}$ is nonnegative. We make use of the Sinkhorn-Knopp convergence result for the nonnegative matrices in the following theorem.

Theorem 5.6.1 *Given that the score matrix* $\mathbf{A}$ *has total support, the entries in the vectors* $\mathbf{o}$ *and* $\mathbf{d}$ *are the reciprocals of the main diagonal elements of the matrices* $\mathbf{D}$ *and* $\mathbf{E}$*.*

Proof. The iterative method used to obtain matrices $\mathbf{D}$ and $\mathbf{E}$ using $\mathbf{D}_0 = \mathbf{E}_0 = \mathbf{I}$ is

$$\mathbf{c}_k^T = \mathbf{e}^T \mathbf{D}_{k-1} \mathbf{A} \mathbf{E}_{k-1},$$
$$\mathbf{r}_k = \mathbf{D}_{k-1} \mathbf{A} \mathbf{E}_k \mathbf{e},$$

where $\mathbf{D}_k = [\mathfrak{D}(\mathbf{r}_k)]^{-1}$, and $\mathbf{E}_k = [\mathfrak{D}(\mathbf{c}_k)]^{-1}$. Given that matrix $\mathbf{A}$ has total support, this iteration converges, and the final matrices $\mathbf{D} = [\mathfrak{D}(\mathbf{r})]^{-1}$ and $\mathbf{E} = [\mathfrak{D}(\mathbf{c})]^{-1}$ are unique. Consequently $\mathbf{S} = \mathbf{DAE} = [\mathfrak{D}(\mathbf{r})]^{-1}\mathbf{A}[\mathfrak{D}(\mathbf{c})]^{-1}$, where $\mathbf{S}$ is doubly stochastic. Using $\mathfrak{D}(\mathbf{x})\mathbf{e} = \mathbf{x}$ and $\mathbf{e}^T\mathfrak{D}(\mathbf{x}) = \mathbf{x}^T$ with $\mathbf{Se} = \mathbf{e}$ and $\mathbf{e}^T\mathbf{S} = \mathbf{e}^T$ produces

$$\mathbf{c} = \mathbf{A}^T[\mathfrak{D}(\mathbf{r})]^{-1}\mathbf{e},$$
$$\mathbf{r} = \mathbf{A}[\mathfrak{D}(\mathbf{c})]^{-1}\mathbf{e}.$$

By setting $\mathbf{c} = \mathbf{o}$, $\mathbf{r} = \mathbf{d}$, and noting that the inverse of a diagonal matrix with positive diagonal is obtained by taking the element-wise reciprocals of the diagonal entries, the above equation set is rewritten as

$$\mathbf{o} = \mathbf{A}^T \frac{1}{\mathbf{d}},$$
$$\mathbf{d} = \mathbf{A} \frac{1}{\mathbf{o}},$$

which is the limit of the original definition of the Offense-Defense Model (5.1) with $\mathbf{A}$ having total support. Thus the theorem is proven.

In practice there is no guarantee that the score matrix $\mathbf{A}$ will have total support. To compensate, a rank-one perturbation can be incorporated to define a new matrix

$$\mathbf{P} = \mathbf{A} + \epsilon \mathbf{e}\mathbf{e}^T.$$

This is used in place of $\mathbf{A}$ so that convergence of (5.1) is guaranteed. This perturbation adds ϵ to all elements of $\mathbf{A}$, and if ϵ is sufficiently small, then its effect on the model should not be significant. However, adding ϵ may affect convergence rates. The method is finalized below

Definition 5.6.2 *The Offense-Defense Model (ODM) The offensive ratings o_i of a team T_i and the defensive ratings of a team T_j are entries of the vectors* $\mathbf{o}$ *and* $\mathbf{d}$ *that are limits of* $\mathbf{o}^{(k)}$ *and* $\mathbf{d}^{(k)}$ *as $k \to \infty$, in which*

$$\begin{aligned} \mathbf{o}^{(k)} &= \mathbf{P}^T \frac{1}{\mathbf{d}^{(k-1)}}, \quad \textit{where } \mathbf{d}^{(0)} = \mathbf{e}, \\ \mathbf{d}^{(k)} &= \mathbf{P} \frac{1}{\mathbf{o}^{(k)}}, \end{aligned} \tag{5.2}$$

where

$$P_{ij} = \begin{cases} A_{ij} + \epsilon & \textit{if team } T_i \textit{ played team } T_j, \\ \epsilon & \textit{team } T_i \textit{ and } T_j \textit{ did not play}, \end{cases}$$

with $\mathbf{e}$ *is a vector of all 1's.*

The second part of the Sinkhorn-Knopp Theorem states that if a matrix $\mathbf{A}$ has support, then the sequences of the diagonal matrices produced for the matrix normalization do not necessarily converge. Almost always matrices created using sports team data for a season will have support (this happens when each team plays at least once or until each team acquires a nonzero score), but it is not given that enough games played translates to total support of the matrix $\mathbf{A}$.

It is straightforward to check whether a matrix $\mathbf{A}$ has support — all row and column sums have to be positive. Checking for existence of total support in a large matrix is a nontrivial issue. One may check that each diagonal of $\mathbf{A}$ is either zero or positive. Alternatively one can check whether there is a positive diagonal to which each nonzero element of $\mathbf{A}$ belongs, as each element of $\mathbf{A}$ belongs to multiple diagonals. This is the reason why it is computationally simpler to perturb each element of $\mathbf{A}$ to force total support rather than perturbing select elements.

Summary of the Offense-Defense Model:

1. Form the adjacency matrix $\mathbf{A}$

$$A_{ij} = \begin{cases} \text{score generated by } T_j \text{ against } T_i & \text{if } T_i \text{ played } T_j, \\ 0 & T_i \text{ and } T_j \text{ did not play.} \end{cases}$$

2. Form the matrix $\mathbf{P}$, $0 < \epsilon < 1$

$$P_{ij} = \begin{cases} A_{ij} + \epsilon & \text{if } T_i \text{ played } T_j, \\ \epsilon & T_i \text{ and } T_j \text{ did not play.} \end{cases}$$

3. Compute $\mathbf{o}$ and $\mathbf{d}$ vectors

$$\mathbf{o}^{(k)} = \mathbf{P}^T \frac{1}{\mathbf{d}^{(k-1)}} \quad k = 1, 2, ..., \text{ where } \mathbf{d}^{(0)} = \mathbf{e},$$

$$\mathbf{d}^{(k)} = \mathbf{P} \frac{1}{\mathbf{o}^{(k)}} \qquad k = 1, 2, ...$$

4. Use rank aggregation to produce the overall rating vector $\mathbf{r}$ (e.g. $\mathbf{r} = \mathbf{o}/\mathbf{d}$).

5.7 Sensitivity of the ODM Ratings to tolerance and ϵ

Given that $\mathbf{A}$ has support, the matrix $[\mathfrak{D}(\mathbf{d}^{(k)})]^{-1}\mathbf{A}[\mathfrak{D}(\mathbf{o}^{(k)})]^{-1}$ will converge to a stochastic matrix as k tends to infinity, but the entries in $\mathbf{o}$ and $\mathbf{d}$ will either converge to zero or grow without bound. Since the ranking is assigned based on the relative order of the vector values we include the following investigation of the rank dependence on the tolerance (tol) and ϵ. First comes a study of the amount of iterations it takes for the ODM to converge depending on the values of tol and ϵ, see Table 5.1. The number of iterations performed by ODM has been only considered for $\epsilon \leq tol$ since the goal is to compute the ratings with a minimum alteration of the data. The study was done using the largest application, NCAA basketball. During 2007 year NCAA basketball Division I contained 341 teams. The ratings were computed for a single game day in the 2007-2008 season.

Table 5.1: The total number of iterations performed by ODM for the given values of tolerance and ϵ.

	tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=0.00001
ϵ =0	27	208	2009	20010	200011
ϵ =0.1	10	-	-	-	-
ϵ =0.01	13	18	-	-	-
ϵ =0.001	16	24	34	-	-
ϵ =0.0001	22	44	68	96	-
ϵ =1e-05	26	77	131	198	291
ϵ =1e-06	27	126	242	365	586
ϵ =1e-07	27	177	438	683	1005
ϵ =1e-08	27	203	765	1282	1818
ϵ =1e-09	27	207	1242	2389	3480
ϵ =1e-10	27	208	1737	4356	6684
ϵ =1e-11	27	208	1965	7630	12727

Next are the Tables 5.2 and 5.3 that contains the pairs of numbers a, b such that

a = the number of teams that were assigned ranks according to the ratings computed by the ODM using **A** (without total support) different from the ranks assigned according to the ratings computed by the ODM using **P** (with total support).

b = the largest difference between ranks assigned to each team using ratings computed by the ODM using **A** and ODM using **P**.

Table 5.2: Sensitivity of the offensive ratings to changes in tolerance and ϵ.

	tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=0.00001
ϵ =0.1	319,49	-	-	-	-
ϵ =0.01	277,14	273,1	-	-	-
ϵ =0.001	140,9	159,9	164,10	-	-
ϵ =0.0001	43,3	69,4	70,5	74,5	-
ϵ =1e-05	10,1	22,1	27,2	31,2	31,2
ϵ =1e-06	0,0	6,1	14,1	18,1	18,1
ϵ =1e-07	0,0	0,0	8,1	10,1	10,1
ϵ =1e-08	0,0	0,0	4,1	6,1	6,1
ϵ =1e-09	0,0	0,0	0,0	4,1	4,1
ϵ =1e-10	0,0	0,0	0,0	0,0	0,0
ϵ =1e-11	0,0	0,0	0,0	0,0	0,0

Table 5.3: Sensitivity of the defensive ratings to changes in tolerance and ϵ.

	tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=0.00001
ϵ =0.1	317,46	-	-	-	-
ϵ =0.01	272,16	265,16	-	-	-
ϵ =0.001	151,4	144,6	150,6	-	-
ϵ =0.0001	46,2	58,4	65,3	67,3	-
ϵ =1e-05	6,1	24,2	27,1	27,1	27,1
ϵ =1e-06	0,0	4,1	8,1	10,1	10,1
ϵ =1e-07	0,0	2,1	4,1	6,1	6,1
ϵ =1e-08	0,0	0,0	4,1	4,1	4,1
ϵ =1e-09	0,0	0,0	0,0	2,1	2,1
ϵ =1e-10	0,0	0,0	0,0	0,0	0,0
ϵ =1e-11	0,0	0,0	0,0	0,0	0,0

As expected the decrease in values of the tolerance and ϵ increases the number of iteration while decreasing the assigned rank discrepancy between the ODM using matrix with total support and the ODM using the matrix without the total support. The conclusion is that changes in ϵ do have an effect on the ratings and decreasing the ϵ to the point at which the discrepancy disappears negates the advantage of fast computation associated with the total support in the matrix.

Table 5.4 contains the pairs a, b as previously defined for the aggregated rating.

Table 5.4: Sensitivity of the aggregated ratings to changes in tolerance and ϵ.

	tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=0.00001
ϵ =0.1	322,97	-	-	-	-
ϵ =0.01	285,80	283,82	-	-	-
ϵ =0.001	165,40	178,55	183,60	-	-
ϵ =0.0001	46,10	102,37	124,47	125,49	-
ϵ =1e-05	6,3	43,19	67,30	76,36	77,37
ϵ =1e-06	0,0	19,6	35,17	43,22	43,25
ϵ =1e-07	0,0	6,1	20,10	23,12	26,14
ϵ =1e-08	0,0	0,0	10,5	12,7	14,8
ϵ =1e-09	0,0	0,0	7,2	8,5	9,6
ϵ =1e-10	0,0	0,0	2,1	5,2	6,3
ϵ =1e-11	0,0	0,0	0,0	2,1	2,1

In the case of aggregated ratings the discrepancy in assigned ranks increases more than for the offensive and defensive ratings. The issue is the propagation of error

from offensive and defensive ratings to the overall ratings. Another study of interest to us is the effect of tolerance used to limit the number of iterations performed by the ODM. Again we are using one day of game data for NCAA basketball from the 2007-2008 season. The entries a, b in the $(tol = i, tol = j)$ cell of the following three tables Table 5.5, Table 5.6, and Table 5.7 are

a = the number of teams that were assigned ranks according to the ratings computed by the ODM using $tol = i$ different from the ranks assigned according to the ratings computed by the ODM using $tol = j$, with ϵ fixed.

b = the largest difference between ranks assigned to each team using ratings computed by the ODM using $\mathbf{A}$ and ODM using $\mathbf{P}$.

Table 5.5: Sensitivity of the offensive ratings to changes in tolerance for a fixed ϵ.

		tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=1e-05
$\epsilon = 0$	tol=0.1	0,0	62,2	70,3	74,3	74,3
	tol=0.01	-	0,0	12,1	16,1	16,1
	tol=0.001	-	-	0,0	4,1	4,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.01$	tol=0.1	0,0	42,2	48,2	48,2	48,2
	tol=0.01	-	0,0	6,1	6,1	6,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.001$	tol=0.1	0,0	18,3	22,3	22,3	22,3
	tol=0.01	-	0,0	4,1	4,1	4,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.0001$	tol=0.1	0,0	36,2	42,2	42,2	42,2
	tol=0.01	-	0,0	6,1	6,1	6,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-05$	tol=0.1	0,0	51,2	57,2	57,2	57,2
	tol=0.01	-	0,0	6,1	6,1	6,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-06$	tol=0.1	0,0	58,2	60,2	60,2	60,2
	tol=0.01	-	0,0	4,1	4,1	4,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0

Table 5.5 continued.

		tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=1e-05
$\epsilon = 1e-07$	tol=0.1	0,0	62,2	65,3	66,3	66,3
	tol=0.01	-	0,0	4,1	6,1	6,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-08$	tol=0.1	0,0	62,2	68,3	69,3	69,3
	tol=0.01	-	0,0	8,1	10,1	10,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-09$	tol=0.1	0,0	62,2	70,3	70,3	70,3
	tol=0.01	-	0,0	12,1	12,1	12,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-10$	tol=0.1	0,0	62,2	70,3	74,3	74,3
	tol=0.01	-	0,0	12,1	16,1	16,1
	tol=0.001	-	-	0,0	4,1	4,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-11$	tol=0.1	0,0	62,2	70,3	74,3	74,3
	tol=0.01	-	0,0	12,1	16,1	16,1
	tol=0.001	-	-	0,0	4,1	4,1
	tol=0.0001	-	-	-	0,0	0,0

Table 5.6: Sensitivity of the defensive ratings to changes in tolerance for a fixed ϵ.

		tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=1e-05
$\epsilon = 0$	tol=0.1	0,0	52,4	58,4	60,4	60,4
	tol=0.01	-	0,0	6,1	8,1	8,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.1$	tol=0.1	0,0	31,2	32,2	34,2	34,2
	tol=0.01	-	0,0	4,1	6,1	6,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.01$	tol=0.1	0,0	54,3	59,3	59,3	59,3
	tol=0.01	-	0,0	6,1	6,1	6,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.001$	tol=0.1	0,0	43,2	43,2	43,2	43,2
	tol=0.01	-	0,0	0,0	0,0	0,0
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0

Table 5.6 continued.

		tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=1e-05
$\epsilon = 0.0001$	tol=0.1	0,0	40,2	42,2	42,2	42,2
	tol=0.01	-	0,0	2,1	2,1	2,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-05$	tol=0.1	0,0	34,2	38,2	40,2	40,2
	tol=0.01	-	0,0	4,1	6,1	6,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-06$	tol=0.1	0,0	49,3	51,3	51,3	51,3
	tol=0.01	-	0,0	2,1	2,1	2,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-07$	tol=0.1	0,0	51,3	54,4	54,4	54,4
	tol=0.01	-	0,0	4,1	4,1	4,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-08$	tol=0.1	0,0	52,4	54,4	56,4	56,4
	tol=0.01	-	0,0	2,1	4,1	4,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-09$	tol=0.1	0,0	52,4	58,4	58,4	58,4
	tol=0.01	-	0,0	6,1	6,1	6,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-10$	tol=0.1	0,0	52,4	58,4	60,4	60,4
	tol=0.01	-	0,0	6,1	8,1	8,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 1e-11$	tol=0.1	0,0	52,4	58,4	60,4	60,4
	tol=0.01	-	0,0	6,1	8,1	8,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0

Table 5.7: Sensitivity of the defensive ratings to changes in tolerance for a fixed ϵ.

		tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=1e-05
$\epsilon = 0$	tol=0.1	0,0	57,2	64,2	65,2	65,2
	tol=0.01	-	0,0	12,1	13,1	13,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.1$	tol=0.1	0,0	74,2	87,3	91,3	91,3
	tol=0.01	-	0,0	16,2	20,2	22,2
	tol=0.001	-	-	0,0	4,1	6,1
	tol=0.0001	-	-	-	0,0	2,1
$\epsilon = 0.01$	tol=0.1	0,0	107,4	111,4	111,4	111,4
	tol=0.01	-	0,0	12,1	14,1	14,1
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.001$	tol=0.1	0,0	79,14	82,19	83,20	83,20
	tol=0.01	-	0,0	13,5	14,6	14,6
	tol=0.001	-	-	0,0	2,1	2,1
	tol=0.0001	-	-	-	0,0	0,0
$\epsilon = 0.0001$	tol=0.1	0,0	57,23	71,32	73,33	74,33
	tol=0.01	-	0,0	22,9	24,10	25,10
	tol=0.001	-	-	0,0	4,1	5,2
	tol=0.0001	-	-	-	0,0	2,1
$\epsilon = 1e-05$	tol=0.1	0,0	36,8	44,19	55,25	56,25
	tol=0.01	-	0,0	21,11	32,17	33,17
	tol=0.001	-	-	0,0	13,6	14,6
	tol=0.0001	-	-	-	0,0	2,1
$\epsilon = 1e-06$	tol=0.1	0,0	48,4	44,5	43,8	42,8
	tol=0.01	-	0,0	11,6	18,10	21,13
	tol=0.001	-	-	0,0	9,4	12,7
	tol=0.0001	-	-	-	0,0	4,3
$\epsilon = 1e-07$	tol=0.1	0,0	52,2	53,3	53,4	52,5
	tol=0.01	-	0,0	13,4	15,5	18,7
	tol=0.001	-	-	0,0	4,1	7,3
	tol=0.0001	-	-	-	0,0	5,2
$\epsilon = 1e-08$	tol=0.1	0,0	57,2	58,2	57,2	57,2
	tol=0.01	-	0,0	6,1	8,1	10,2
	tol=0.001	-	-	0,0	2,1	5,2
	tol=0.0001	-	-	-	0,0	4,1

Table 5.7 continued.

		tol=0.1	tol=0.01	tol=0.001	tol=0.0001	tol=1e-05
$\epsilon = 1e-009$	tol=0.1	0,0	57,2	61,2	58,2	57,2
	tol=0.01	-	0,0	10,1	10,1	8,1
	tol=0.001	-	-	0,0	5,2	6,3
	tol=0.0001	-	-	-	0,0	2,1
$\epsilon = 1e-10$	tol=0.1	0,0	57,2	63,2	61,2	60,2
	tol=0.01	-	0,0	11,1	13,1	12,1
	tol=0.001	-	-	0,0	2,1	4,1
	tol=0.0001	-	-	-	0,0	2,1
$\epsilon = 1e-11$	tol=0.1	0,0	57,2	64,2	64,2	64,2
	tol=0.01	-	0,0	12,1	12,1	12,1
	tol=0.001	-	-	0,0	0,0	0,0
	tol=0.0001	-	-	-	0,0	0,0

The apparent conclusion of the tolerance experiments summarized in tables 5.5, 5.6, and 5.7 is that $tol = 0.1$ results in the highest rank discrepancies with other tolerance values. Since number of iterations performed by ODM increases by a multiple of 10 with every decrease of ϵ by 10^{-1}, then $tol < 0.001$ is not practical. For the game predictions illustrated in the chapter 7 the values $tol = 0.01$ and $\epsilon = 0.00001$ were chosen.

5.8 Score Matrix

It is not a certainty that a better team will always accumulate the highest score during a given game. Hence we can take into account other statistics as alternative strength indicators of a given team. For example, in football each team accumulates total yardage in each game. We can form a score matrix $\mathbf{A}$ using the total yards statistic and compute the offensive and defensive strength of a team according to this statistic. One can compute multiple offensive and defensive ratings of a given team based on several statistics (e.g. game scores, total rushing yards, total first downs, etc.) and aggregate them into a single offensive and a single defensive score.

5.9 Relation to the Hyperlink-Induced Topic Search Model

The Offense-Defense model is in part inspired by the ranking algorithm HITS created by Jon Kleinberg in 1999 for web ranking [71]. Analogous to the Offence-Defense model, HITS assumes each object has two ratings. However, unlike the offense and defense ratings both authority and hub values are "good" when they are high. Hence there are no reciprocals in the computation, and, providing that the iterative procedure (3.1) converges, vectors $\mathbf{a}$ and $\mathbf{h}$ are eigenvectors of $\mathbf{L}^T\mathbf{L}$ and $\mathbf{L}\mathbf{L}^T$ respectively. In other words HITS is a linear model while our ODM formulation is non-linear.

5.10 Ranking via Diagonal Similarity Matrix Scaling (DSS)

A different way to balance a nonnegative matrix is to use a similarity scaling. Given $\mathbf{A} \geq 0$ find a diagonal matrix $\mathbf{D}$ with positive main diagonal such that $\mathbf{DAD}^{-1}$ is balanced, that is sums of row i and column i are equal. As mentioned before Hartfiel has proved that if $\mathbf{A} \geq 0$ is irreducible then such diagonal matrix $\mathbf{D}$ exists and is unique (up to a scalar multiple) [53]. Although this method bears some similarity to the Sinkhorn-Knopp method the result of the similarity scaling is not necessarily a doubly stochastic matrix.

Here we define another ranking method. Let matrix $\mathbf{A}$ contain the game scores such that

$$A_{ij} = \begin{cases} \text{score generated by } T_j \text{ against } T_i & \text{if team } T_i \text{ and } T_j \text{ played,} \\ 0 & \text{team } T_i \text{ and } T_j \text{ did not play.} \end{cases}$$

Then the rating of a team T_i is $r_i = D_{ii}$, where $\mathbf{D}$ is a diagonal matrix with positive main diagonal such that $\mathbf{DAD}^{-1}$ is balanced (i.e. row sum i is equal to column sum i). The requirement for the convergence of this method is the irreducibility of the matrix $\mathbf{A}$. Typically a few weeks into the season teams (in both NFL and NCAA) have played enough games to ensure that the score matrix $\mathbf{A}$ is irreducible.

Summary of the Diagonal Similarity Scaling Model:

1. Form an adjacency matrix $\mathbf{A}$

$$A_{ij} = \begin{cases} \text{score generated by } T_j \text{ against } T_i & \text{if } T_i \text{ and } T_j \text{ played,} \\ 0 & T_i \text{ and } T_j \text{ did not play.} \end{cases}$$

2. Compute $\mathbf{r}$

$$\mathbf{A}^{(k+1)} = \mathfrak{D}(\mathbf{v}^{(k)})\mathbf{A}^{(k)}[\mathfrak{D}(\mathbf{v}^{(k)})]^{-1},$$

$$\mathbf{v}^{(k+1)} = \sqrt{\frac{[\mathbf{A}^{(k+1)}]^T\mathbf{e}}{\mathbf{A}^{(k+1)}\mathbf{e}}},$$

where division is taken elementwise and $\mathbf{A}^{(0)} = \mathbf{A}$ and $\mathbf{v}_0 = \sqrt{\mathbf{A}^T\mathbf{e}/\mathbf{A}\mathbf{e}}$. The rating vector $\mathbf{r} = [\mathfrak{D}(\mathbf{v}^{(0)})\mathfrak{D}(\mathbf{v}^{(1)})...]\mathbf{e}$

3. Use the ratings in $\mathbf{r}$ to rank the teams.

Chapter 6

The Generalized Markov Method (GeM)

6.1 Introduction

The Generalized Markov (GeM) ranking model has its beginning in the PageRank method. As PageRank does, GeM uses the basics of graph theory and the theory of Markov chains to generate ratings of n objects in a finite set. The GeM model is not limited to sports and can be used on any problem that can be represented as a weighted directed graph.

6.2 Feature matrices GeM

A sport season with team competition can always be represented as a weighted directed graph, where nodes symbolize teams and the weighted directed edges describe the interactions, or games. The approach is to use the statistical data generated by the sport teams throughout a given season to form a stochastic irreducible matrix $\mathbf{G}$. The theory of finite Markov chains guarantees the existence of a unique positive left eigenvector $\boldsymbol{\pi}$ corresponding to the eigenvalue 1 such that $\sum \pi_i = 1$. This vector $\boldsymbol{\pi}$ is a discrete probability distribution whose ith entry represents the eventual proportion of time that the Markov chain is in state i. The ith entry of the vector $\boldsymbol{\pi}$ can be interpreted as a rating score corresponding to the ith team, and any function on the rating scores can be used to assign ranks to the teams. As previously, we use the most basic function

that assigns rank 1 to the team with the highest rating score, rank 2 to the team with the next highest rating score and so on.

The matrix $\mathbf{G}$ is defined as

$$\mathbf{G} = \alpha_0\mathbf{S}_0 + \alpha_1\mathbf{S}_1 + ... + \alpha_p\mathbf{S}_p, \tag{6.1}$$

where $\mathbf{S}_i$ is a stochastic matrix, called a *feature matrix*, and $0 \leq \alpha_i \leq 1$, with $\sum \alpha_i = 1$. The challenging part is to decide how to best form each stochastic matrix $\mathbf{S}_i$.

For our experiments we use data from NFL and NCAA football and basketball seasons. To form a directed graph, teams (in both NFL and NCAA) become nodes and each game is a directed edge from loser to winner, hence loser endorsing the winner. Which team is considered to be a winner depends on what game statistic is being considered. The basic game statistic in football is game points accumulated by each team during a game. These define the winner of each game in the conventional way. However, there are other game statistics such as the rushing yardage each team accumulates by carrying the football, or passing yardage by throwing the ball. The Dallas Cowboys may score more points to beat the Washington Redskins, but the Redskins may accumulate more total yards. Points scored by a team may not always be an absolute indication of the team's "quality," or rank, relative to its opponent, so we want to consider other statistics or indicators when computing each team's rating. Since game points are used to assign winners of each game we designate the first feature matrix $\mathbf{S}_0$ to be formed using game points. The remaining feature matrices are similarly formed from other game statistics.

Using the small example 4.1 introduced earlier the associated directed graph is shown in Fig 6.2. The digraph has six nodes and 10 edges. Each edge is from the loser to the winner of the game with associated weight being the positive score difference. For example the first game is represented by a directed edge from NO (New Orleans Saints) to Car (Carolina Panthers) with the associated weight $|16 - 13| = 3$.

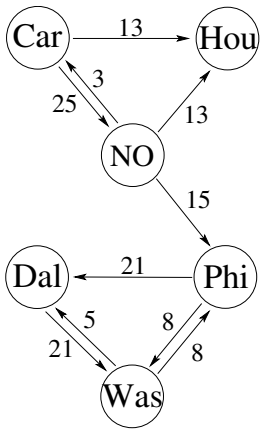

Figure 6.1: Small NFL example from the 2007 regular season.

The corresponding adjacency matrix $\mathbf{A}$ is defined by

$$A_{ij} = \begin{cases} w_{ij} & \text{if team } T_i \text{ lost to } T_j, \\ 0 & \text{otherwise,} \end{cases}$$

where w_{ij} is the sum of positive point score differences of the games that team T_i lost to team T_j. For this example

$$\mathbf{A} = \begin{array}{c} \\ \text{Car} \\ \text{Dal} \\ \text{Hou} \\ \text{NO} \\ \text{Phi} \\ \text{Was} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Car} & \text{Dal} & \text{Hou} & \text{NO} & \text{Phi} & \text{Was} \end{array} \\ \begin{pmatrix} 0 & 0 & 13 & 25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 21 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 13 & 0 & 15 & 0 \\ 0 & 21 & 0 & 0 & 0 & 8 \\ 0 & 5 & 0 & 0 & 8 & 0 \end{pmatrix} \end{array}$$

A stochastic matrix $\mathbf{S}$ is obtained from the adjacency matrix $\mathbf{A}$ by setting $\mathbf{S} = \mathbf{H} + \mathbf{a}\mathbf{u}^T$, where

$$\mathbf{H} = [\mathfrak{D}(\mathbf{A}\mathbf{e} + \mathbf{a})]^{-1}\mathbf{A},$$

$\mathbf{u}$ is any positive probability vector, and

$$\mathbf{a}_i = \begin{cases} 1 & \text{if } i\text{th row of } \mathbf{A} \text{ is 0,} \\ 0 & \text{otherwise.} \end{cases}$$

The matrix $[\mathfrak{D}(\mathbf{Ae}+\mathbf{a})]^{-1}$ is diagonal such that the diagonal entry (i,i) is 1 if the ith row of $\mathbf{A}$ is zero and is the reciprocal of the ith row sum of $\mathbf{A}$ otherwise.

In our example the only undefeated team is Houston, and hence the 4th row of $\mathbf{A}$ is zero. We use a uniform probability distribution vector $\mathbf{u}$. The resulting stochastic matrix is

$$\mathbf{S} = \begin{array}{c} \\ \text{Car} \\ \text{Dal} \\ \text{Hou} \\ \text{NO} \\ \text{Phi} \\ \text{Was} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Car} & \text{Dal} & \text{Hou} & \text{NO} & \text{Phi} & \text{Was} \end{array} \\ \begin{pmatrix} 0 & 0 & 13/38 & 25/38 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 3/31 & 0 & 13/31 & 0 & 15/31 & 0 \\ 0 & 21/29 & 0 & 0 & 0 & 8/29 \\ 0 & 5/13 & 0 & 0 & 8/13 & 0 \end{pmatrix} \end{array}.$$

To make use of existence and uniqueness of the left eigenvector, corresponding to the eigenvalue 1 guaranteed by Markov chains theory, the matrix in question has to be irreducible. The matrix $\mathbf{S}$ is stochastic but is not necessarily irreducible. Our small NFL example is a simple demonstration of this fact. To complete the method form the matrix $\mathbf{G}$ as described above

$$\mathbf{G} = \alpha_0\mathbf{S}_0 + \alpha_1\mathbf{S}_1 + ... + \alpha_p\mathbf{S}_p,$$

where each $\mathbf{S}_i$ is constructed using ith game statistic and the technique described above. The resulting matrix $\mathbf{G}$ is both stochastic and in almost all cases irreducible. There is a slight possibility that if the stochastic matrices $\mathbf{S}_0, ..., \mathbf{S}_p$ are very sparse, then the convex combination of these matrices will still be reducible. However, when applying this method to sport competition such as NFL there is usually enough data to guarantee the irreducibility after three weeks of games. From experiments during the first three weeks of the NFL season teams usually have not played enough games to make the corresponding directed graph for game scores strongly connected, and therefore matrix $\mathbf{S}_0$ will fail to be irreducible.

6.3 Personalization Vector GeM, GeMPV

A more basic version of our model is similar to the method used by the Google search engine, called PageRank [18]. Only two stochastic matrices are used, and the second has rank 1

$$\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)\mathbf{e}\mathbf{v}^T \tag{6.2}$$

where $0 < \alpha < 1$, $\mathbf{S}$ is stochastic, $\mathbf{e}$ is a vector of all 1's, and $\mathbf{v}$ is a positive probability distribution vector. Larry Page and Sergey Brin, creators of PageRank, refer to the vector $\mathbf{v}$ as a personalization vector. Within a sports context entry i of vector $\mathbf{v}$ is a probability of a loss to team T_i, or equivalently an overall "measure" of strength of team T_i. This vector can be computed using some combination of game statistics. A very simple personalization vector is a uniform distribution $(1/n)\mathbf{e}$.

Using $\alpha = 0.85$, $\mathbf{v} = (1/6)\mathbf{e}$, and $\mathbf{S}$ formed using game scores the matrix $\mathbf{G}$ is

$$\mathbf{G} = 0.85\mathbf{S} + 0.15(1/6)\mathbf{e}\mathbf{e}^T =$$

$$\begin{array}{c} \\ \text{Car} \\ \text{Dal} \\ \text{Hou} \\ \text{NO} \\ \text{Phi} \\ \text{Was} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Car} & \text{Dal} & \text{Hou} & \text{NO} & \text{Phi} & \text{Was} \end{array} \\ \left(\begin{array}{cccccc} 1/40 & 1/40 & 6/19 & 111/190 & 1/40 & 1/40 \\ 1/40 & 1/40 & 1/40 & 1/40 & 1/40 & 7/8 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 133/1240 & 1/40 & 473/1240 & 1/40 & 541/1240 & 1/40 \\ 1/40 & 743/1160 & 1/40 & 1/40 & 1/40 & 301/1160 \\ 1/40 & 183/520 & 1/40 & 1/40 & 57/104 & 1/40 \end{array} \right) \end{array}.$$

The rating vector for this example is the stationary distribution for $\mathbf{G}$, which is

$$\boldsymbol{\pi}^T \approx \begin{pmatrix} 0.0389 & 0.2824 & 0.0656 & 0.056 & 0.2289 & 0.3281 \end{pmatrix}.$$

The list of the teams in the order of rating scores (ranked best to worst) is

Washington Dallas Philadelphia Houston New Orleans Carolina.

6.4 Offense-Defense Vectors GeM, GeMODF

The second version of our method involves three stochastic matrices

$$\mathbf{G} = \alpha\mathbf{S} + \beta_1\mathbf{e}\mathbf{o}^T + \beta_2\mathbf{e}\mathbf{d}^T, \tag{6.3}$$

where $0 \leq \alpha, \beta_1, \beta_2 \leq 1$, $\alpha + \beta_1 + \beta_2 = 1$, $\mathbf{S}$ is stochastic, and $\mathbf{o}$, $\mathbf{d}$ are probability distribution vectors. Vector $\mathbf{o}$ is called the offense vector, with entries being offensive ratings of each team. Vector $\mathbf{d}$ is called the defense vector and its entries are the defensive ratings of the teams. Vectors $\mathbf{o}$ and $\mathbf{d}$ are computed using the ODM method, see Chapter 5, on a statistic of user's choice.

6.5 Feature Vectors GeM, GeMNMF

The final version of GeM presented in this paper uses multiple rank one updates to the stochastic matrix $\mathbf{S}$

$$\mathbf{G} = \alpha_0 \mathbf{S} + \alpha_1 \mathbf{e}\mathbf{u}_1^T + ... + \alpha_p \mathbf{e}\mathbf{u}_p^T, \tag{6.4}$$

where $0 < \alpha_i < 1$, $\sum_{i=0}^{p} \alpha_i = 1$, and $\mathbf{u}_i$ is a probability distribution vector. Vectors $\mathbf{u}_i$, $i = 1, .., p$, are produced using a team's season statistics. One possibility is to use all of the "raw" normalized team-by-statistic vectors that are available. However, another way is to collect all the original statistical data into a team-by-statistic matrix, call it $\mathbf{M}$, and decompose it. GeM algorithm requires the vectors $\mathbf{u}_i$ to be nonnegative, and hence to be useful, the decomposition of $\mathbf{M}$ has to yield some set of nonnegative results. The natural candidate is the nonnegative matrix decomposition

$$\mathbf{M}_{n \times m} \approxeq \mathbf{W}_{n \times p} \mathbf{H}_{p \times m}$$

where $\mathbf{H}$ and $\mathbf{W}$ are nonnegative matrices, n is the number of teams, m is the total number of statistics, and p is the number of statistical vectors chosen to update $\mathbf{S}$.

Nonnegative matrix factorization was initially introduced by Lee and Seung in 1999 [80]. The nonnegativity of the decomposition is its greatest advantage. A nonnegative matrix $\mathbf{M}$ can be approximated as a sum of nonnegative matrices of rank one $\mathbf{W}_i \mathbf{H}_i^T$. In the initial publication Lee and Seung used their method to decompose matrices representing images of human faces. The resulting rank 1 matrices represented face features. Since then nonnegative matrix factorization has been used in numerous applications where nonnegative matrices naturally arise [12]. One of the main applications is data clustering. There are some drawbacks associated with this method. The known decomposition methods do not guarantee the most optimal decomposition and depend on the initialization. The number p, the size of the decomposition, has to be

specified by the user. Other studies have been done in advancing, accelerating, and increasing sparsity in the resulting matrices **H** and **W** of the original decomposition method proposed by Lee and Seung [48, 58, 81, 95]

In the GeM application of the nonnegative matrix decomposition we use columns of **W** to form vectors $\mathbf{u}_i$ and rows of **H** to obtain corresponding α_i. Observe that

$$\mathbf{M}_{n\times m} \approxeq \mathbf{W}_{n\times p}\mathbf{H}_{p\times m} = \mathbf{W}_1\mathbf{H}_1^T + ... + \mathbf{W}_p\mathbf{H}_p^T,$$

with $\mathbf{W}_i$ being the ith column of **W** and $\mathbf{H}_i^T$ the ith row of **H**. Let the feature vector $\mathbf{u}_i = (1/||\mathbf{W}_i||_1)\mathbf{W}_i$, and $\alpha_i = [(1-\alpha_0)/\sum_{i,j} W_{ij}]\sum_{j=1}^{m} W_{ij}$, $i = 1, ..., p$.

6.6 Computing Parameters $\alpha_0, ...\alpha_p$

This section contains an investigation of a method for determining the values of the parameters $\alpha_0, ...\alpha_p$ for the initially proposed model 7.3

$$\mathbf{G} = \alpha_0\mathbf{S}_0 + \alpha_1\mathbf{S}_1 + ... + \alpha_p\mathbf{S}_p.$$

Later the ratings produced by the various versions of the GeM model are used to predict game outcomes for NFL and NCAA football. We presume that the "better" the computed ratings the higher the accuracy of our predictions. The problem of estimating $\alpha_0, ...\alpha_p$ therefore consists of two aspects. One is to determine which game statistics are the better predictors of the team's ability to win. The other is to determine the approximation of the $\alpha_0, ...\alpha_p$ that produces the "better" ratings.

Let us consider the issue of determining which game statistics have the greatest influence on the team's wins. Assume that various score differences obtained by team T_i against team T_j have a linear relation to the score differences of wins team T_i produces against team T_j. Given n teams and $p+1$ game statistics let $t_{(ij)_k}$ be the total difference of a kth game statistic produced by the team T_i against team T_j. In case teams T_i and T_j did not play set $t_{(ij)_k} = 0$ for all k. The game score differences have a direct correspondence to the wins of team T_i over team T_j, however, the score differences convey more information. Thus given that team T_i beat team T_j by w_{i_j} points suppose that

$$w_{i_j} = \alpha_0 t_{(ij)_0} + \alpha_1 t_{(ij)_1} + ... + \alpha_p t_{(ij)_p},$$

where $0 < \alpha_i < 1$, $\sum_{i=0}^{p} \alpha_i = 1$. The resulting system of linear equations can be written as $\mathbf{T}\boldsymbol{\alpha} = \mathbf{w}$ subject to $\boldsymbol{\alpha} \geq 0$, $\sum_{i=0}^{p} \alpha_i = 1$. Where

$$\mathbf{T} = \begin{pmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} & \dots & \mathbf{T}_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{T}_{n1} & \mathbf{T}_{n2} & \dots & \mathbf{T}_{np} \end{pmatrix}_{n(n-1)\times p} \qquad \mathbf{T}_{ik} = \begin{array}{l} \\ \text{team } 1 \\ \vdots \\ \text{team } i-1 \\ \text{team } i+1 \\ \vdots \\ \text{team } n \end{array} \begin{array}{c} \text{Stat } k \\ \begin{pmatrix} t_{(i1)_k} \\ \vdots \\ t_{(i[i-1])_k} \\ t_{(i[i+1])_k} \\ \vdots \\ t_{(in)_k} \end{pmatrix} \end{array}$$

and $\mathbf{T}_{ik}$ is an $(n-1) \times 1$ vector.

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}_{n(n-1)\times 1} \qquad \mathbf{w}_i = \begin{array}{l} \\ \text{team } 1 \\ \vdots \\ \text{team } i-1 \\ \text{team } i+1 \\ \vdots \\ \text{team } n \end{array} \begin{array}{c} \text{team } T_i \\ \begin{pmatrix} w_{i_1} \\ \vdots \\ w_{i_{i-1}} \\ w_{i_{i+1}} \\ \vdots \\ w_{i_n} \end{pmatrix}_{(n-1)\times 1} \end{array}$$

and w_{i_j} is the total amount by which team T_i outscored team T_j.

The computational experiment in Table 6.1 is based on the 2007 NFL season and playoffs. The game statistics considered were scores, total yards, rushing yards, passing yards, first downs, and return yards. Matrix $\mathbf{T}$ and vector w were updated weekly and $\alpha_1, ...\alpha_5$ recomputed. The solution to $\mathbf{T}\boldsymbol{\alpha} = \mathbf{w}$ was computed using the linear least squares with nonnegativity constraints Matlab function lsqnonneg($\mathbf{T}$,$\mathbf{w}$). This function solves

$$\mathbf{T}\boldsymbol{\alpha} = \mathbf{w}, \qquad \alpha_i \geq 0,$$

and is based on the algorithm by Lawson and Hanson [79]. The computed nonnegative values $\alpha_0, ...\alpha_5$ are normalized so that $\sum_{i=0}^{5} \alpha_i = 1$.

Table 6.1: Values of $\alpha_1, ..., \alpha_5$ during the 2007 NFL season and playoffs.

Week played	α_1 (Total Yards)	α_2 (Rushing Yards)	α_3 (Passing Yards)	α_4 (First Downs)	α_5 (Return Yards)
0	0.0000	0.1614	0.0492	0.7429	0.0466
1	0.1907	0.4370	0.0712	0.1677	0.1335
2	0.1448	0.3529	0.0303	0.3800	0.0920
3	0.1824	0.3537	0.0000	0.3850	0.0789
4	0.4257	0.5437	0.0000	0.0000	0.0306
5	0.4186	0.5512	0.0000	0.0000	0.0302
6	0.4836	0.5065	0.0000	0.0000	0.0098
7	0.4680	0.4966	0.0000	0.0000	0.0354
8	0.4807	0.4843	0.0000	0.0000	0.0350
9	0.2884	0.2279	0.0000	0.4501	0.0336
10	0.1938	0.1411	0.0030	0.6406	0.0216
11	0.2617	0.1482	0.0000	0.5379	0.0522
12	0.1707	0.1247	0.0000	0.6656	0.0390
13	0.1609	0.1230	0.0000	0.6858	0.0303
14	0.2146	0.1529	0.0031	0.5889	0.0405
15	0.1783	0.1378	0.0061	0.6383	0.0395
16	0.1879	0.1519	0.0000	0.6219	0.0383
17	0.2629	0.1967	0.0000	0.4849	0.0555
18	0.2816	0.2026	0.0000	0.4616	0.0542
19	0.3085	0.2246	0.0000	0.4093	0.0576
20	0.3159	0.2264	0.0000	0.4012	0.0565

Table 6.1 appears to suggest that by the end of week 20 (third week of playoffs) the statistics contributing the most to score differences are total yards and first downs. However, even though there is a significant fluctuation of values from week to week there is some tendency of convergence through the later part of the season. There is also a considerable fluctuation from year to year. Consider the Table 6.2 where we include the values of $\alpha_1, ...\alpha_5$ from week 20 for the years 2001 through 2007.

Table 6.2: Values of $\alpha_1, ..., \alpha_5$ after week 20 during the 2001-2007 NFL seasons.

Year played	α_1 (Total Yards)	α_2 (Rushing Yards)	α_3 (Passing Yards)	α_4 (First Downs)	α_5 (Return Yards)
2001	0.0138	0.2418	0.0912	0.6021	0.0510
2002	0.4043	0.4606	0.0404	0.0000	0.0946
2003	0.2668	0.2995	0.0665	0.2521	0.1151
2004	0.0000	0.0751	0.0308	0.8797	0.0144
2005	0.1353	0.3057	0.0096	0.5426	0.0067
2006	0.1025	0.1162	0.0204	0.7417	0.0192
2007	0.3159	0.2264	0.0000	0.4012	0.0565

The fluctuations of the values of $\alpha_1, ...\alpha_5$ may suggest that the NFL goes through certain trends during which playing styles propel various statistics be the tools responsible for producing greater score differences. For example, perhaps during some years teams that were most successful had moved the ball consistently but short distances thus increasing the number of first downs while having a relatively small number of passing yards. Whereas in other years the strategy of outscoring opponents was in prodigious passing. In any case a general conclusion from the experiments is that the passing yards and the return yards appear to be least significant among the five statistics considered. Furthermore the total yards appear to be somewhat less important than the rushing yards and the first downs. With this in mind the upcoming game prediction experiments use these two statistics to compute the ratings.

The second issue to be addressed is the actual computation of the selected parameters, α_2 and α_4. The simplest approach is to recompute and normalize the solution to $\mathbf{T}\boldsymbol{\alpha} = \mathbf{w}$, where $\boldsymbol{\alpha}$=(α_2 α_4) ≥ 0 and the matrix $\mathbf{T}$ contains two columns corresponding to rushing yards and first down differences. The result for the 2007 NFL season is summarized in Table 6.3

Table 6.3: Values of α_2 and α_4 during the 2007 NFL season.

Week played	α_2 (Rushing Yards)	α_4 (First Downs)
0	0.0846	0.9154
1	0.1087	0.8913
2	0.1274	0.8726
3	0.1257	0.8743
4	0.1108	0.8892
5	0.1346	0.8654
6	0.1102	0.8898
7	0.1076	0.8924
8	0.0957	0.9043
9	0.0765	0.9235
10	0.0646	0.9354
11	0.0610	0.9390
12	0.0649	0.9351
13	0.0652	0.9348
14	0.0681	0.9319
15	0.0654	0.9346
16	0.0714	0.9286
17	0.0771	0.9229
18	0.0764	0.9236
19	0.0796	0.9204
20	0.0794	0.9206

For the 2007 season first downs appear to decidedly outweigh the importance of rushing yards. Next consider a larger example, which is NCAA football competition. For NCAA football the statistics considered were total yards, rushing yards, passing yards, and first downs. Unlike results using the NFL data the results using the NCAA football data exhibit more consistency from year to year. As shown in Table 6.4 the two most relevant statistics are total yards and rushing yards.

Table 6.4: Values of $\alpha_1, ..., \alpha_4$ during the 2007 NCAA football seasons.

Week played	α_1 (Total Yards)	α_2 (Rushing Yards)	α_3 (Passing Yards)	α_4 (First Downs)
5	0.6778	0.3222	0.0000	0.0000
6	0.6314	0.3686	0.0000	0.0000
7	0.6407	0.3516	0.0077	0.0000
8	0.6031	0.3653	0.0315	0.0000
9	0.6263	0.3523	0.0214	0.0000
10	0.6638	0.3125	0.0237	0.0000
11	0.6905	0.2860	0.0235	0.0000
12	0.6817	0.2955	0.0229	0.0000
13	0.6728	0.3114	0.0158	0.0000
14	0.6549	0.3270	0.0181	0.0000
15	0.6549	0.3270	0.0181	0.0000
16	0.6549	0.3270	0.0181	0.0000

In view of the results from Table 6.4 we henceforth exclude the passing yards and the first downs statistics from our ratings computation using GeM. The recomputed values of the α_1 and α_2 during the 2007 NCAA season are given in Table 6.5.

Table 6.5: Values of α_1 and α_2 during the 2007 NCAA football season.

Week played	α_1 (Total Yards)	α_2 (First Downs)
5	0.6778	0.3222
6	0.6314	0.3686
7	0.6506	0.3494
8	0.6426	0.3574
9	0.6530	0.3470
10	0.6940	0.3060
11	0.7208	0.2792
12	0.7107	0.2893
13	0.6927	0.3073
14	0.6775	0.3225
15	0.6775	0.3225
16	0.6775	0.3225

Chapter 7

Game Predictions

7.1 Data

Now we proceed to the game predictions part of this work which is a validation of the proposed Offense-Defense and Generalized Markov Models. Game predictions could amount to periodically computing the overall team ratings and use them to predict the outcomes of upcoming games. The process could be as simple as comparing current team ratings and assigning the team with the higher rating the winner of an upcoming match. The predictions are then compared to the actual results. Refer to this approach as "foresight predictions." A somewhat different way of game prediction is to use the entire season of data to compute the ratings score and then use these rating scores to predict the outcomes of the games of that season. In other words, if all the information about a given season is available, what is the maximum accuracy that we can achieve? This approach will be referred to as "hindsight predictions".

The complexity of the rating model may require substantial effort in data gathering and processing. If the model uses only win-loss counts and scores of the individual games played then the data is relatively easy to acquire. Many sports websites contain lists of NFL or NCAA games along with scores for a number of seasons, see for example [89].

If there is a need for individual game statistics such as rushing or passing yards, then the data gathering process may involve parsing data files of the game box scores such as ones found on [133, 38]. These files have a great structure geared toward delivering the information to human observers and are not easily transfered to a format friendly

to the MATLAB (a numerical computing environment and programming language created by The MathWorks). Furthermore the number of files that has to be processed is large enough to prohibit manual data entry. For the following experiments we have used NFL data from 2001 to 2007. The statistics collected were individual game scores, total yards, rushing yards, passing yards, number of first downs, and return yards for each of the team. The number of files (each corresponding to a single game) processed for the NFL alone was close to two thousand. The number of files pulled and parsed increased for NCAA football and basketball even though not as many years of game files were available.

In any case, data gathering and traversing methods had to automated. Two main attributes are required from the potential data website sources. One is the reliability of the information, and the second is the stability of the file format used to showcase the information. Both attributes are essential if one uses software to extract the data from these files. Each file had to be pulled from the web and traversed by parsing software that had to determine which teams played the given game and their respective game statistics.

Since we chose to use only Division I-A (Div I-A) teams then each parsed NCAA game had to be identified as valid (between two Div I-A teams) or invalid (one of the teams is not from Div I-A). For this reason a careful account of the valid team names for each season had to be kept. Both NFL and NCAA Div I-A have experienced expansions, and the NFL in particular had several of its teams change names over the past decade. Since each team is associated with a row in the team-by-team matrices formed by various models it was imperative to keep track of the changes in the overall number of teams as well as in their names. For all three sports the alphabetical order of the valid team names is the order in which the teams are associated with the matrix rows. MATLAB was used to perform the numerical computations of the ratings for all the models so far introduced. However, MATLAB does not posses the flexibility of fast and efficient data processing. Therefore all the data files were downloaded and parsed, using Perl scripts, from two sources, [133] for NFL and [38] for NCAA football and basketball. Since parsing of NFL and NCAA data is similar only the NFL parsing scripts are included in the Appendix B.

7.2 Game Predictions

This section contains the results of game predictions for three sports; football (NFL), college football (NCAA football), and college basketball (NCAA basketball). Different team sports present different challenges to game predictions. For example, the NFL tends to have a very regular schedule with a sufficient interplay within divisions as well as without. The NFL currently consists of only 32 teams that are relatively close to each other in "quality." This makes for a narrow difference in the ratings values. A small number of teams corresponds to a small matrix and hence fast MATLAB computations for the game predictions. In contrast, NCAA football currently has 120 Division I-A teams, and the team quality ranges wildly. Game data for the NCAA is very sparse since there are relatively few games played in comparison to the number of teams. There is little interplay between divisions, and Division I-A teams tend to play Division I-AA teams during the beginning of the season. Finally, NCAA basketball currently has 341 teams in its Div I-A. On average each Division I-A team tends to play about 30 games per season. Although during the 2007-2008 season teams played more than 5000 games the resulting matrices are sparse. There is also a great fluctuation in relative strength of the teams.

7.3 Game Prediction Results for NFL

Both foresight and hindsight game prediction results are presented for the NFL based experiments. The first type, foresight, recomputes the rating scores weekly and uses them to predict the winners of the upcoming games. These predictions for the NFL are done for regular season games from week 1 through the Super Bowl. In order to ensure that some models (Keener, Massey and GeM) have enough game data the pre-season game results are taken into account during the first three weeks of the regular season and used for all the models. The previous season data was not used to supplement the first three weeks of regular season since there are significant changes in the makeup of the teams, names of the teams, and the number of the teams in the past ten years. Enough games have been played to discard pre-season data after three weeks. The number of games predicted correctly by each of the models was converted into percentages, since over the years the total number of games played by NFL teams

has changed. The second type of game predictions is more theoretical in nature than practical because it employs hindsight. Hindsight predictions give us an idea of just how well the ranking methods can do in an ideal case. There is an expected and substantial increase in the accuracy of predictions in the case of hindsight as evidenced in Table 7.1, Table 7.2, Table 7.3 and 7.4.

The ratings used to do game predictions are computed using the game outcomes for the entire season. The prediction results are broken into three tables. Tables 7.1 and 7.2 contain foresight and hindsight predictions using the Offense-Defense and the Diagonal Similarity Scaling Models. Three other popular sports ranking models by Colley, Massey, and Keener are included for comparison. Since different values of the ODM parameter ϵ did not produce significant changes in the predictions, all the predictions were computed assuming a tolerance $tol = 0.01$ and $\epsilon = 0.00001$. The DSS model uses tolerance $tol = 0.01$.

Table 7.1: Foresight game prediction percentages for NFL (with DSS and ODM).

	Colley	DSS	Keener	Massey	ODM
2001	57.92	59.85	58.69	60.23	60.62
2002	59.18	62.17	58.43	60.30	63.30
2003	63.30	64.79	58.05	64.04	61.05
2004	61.80	60.67	59.93	62.17	58.43
2005	61.80	64.79	62.55	65.17	64.04
2006	58.80	59.55	57.68	60.30	58.05
2007	66.67	67.79	62.55	68.16	68.91

Table 7.2: Hindsight game prediction percentages for NFL (with DSS and ODM).

	Colley	DSS	Keener	Massey	ODM
2001	72.97	69.50	72.97	69.88	69.88
2002	68.16	68.16	68.91	67.04	68.54
2003	75.66	73.03	73.78	71.91	72.28
2004	74.16	68.16	70.79	67.42	68.54
2005	73.03	74.91	75.66	75.28	76.40
2006	72.66	72.66	69.29	71.16	70.04
2007	75.66	72.28	76.03	73.41	72.28

The following two tables 7.3 and 7.4 contain foresight and hindsight predictions using four versions of GeM. The first one is the simple Personalization Vector version

(GeMPV)

$$\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)\mathbf{e}\mathbf{v}^T,$$

with $\alpha = 0.6$, $\mathbf{v} = (1/n)\mathbf{e}$, and $\mathbf{e}$ is a vector of all 1's. The second GeM version (GeMODM) uses two normalized vectors $\mathbf{o}$ and $\mathbf{d}$ computed using ODM and the rushing yards statistic

$$\mathbf{G} = \alpha_0\mathbf{S} + \alpha_1\mathbf{e}\mathbf{o}^T + \alpha_2\mathbf{e}\mathbf{d}^T,$$

and setting $\alpha_0 = 0.6$, $\alpha_1 = 0.2$, and $\alpha_2 = 0.2$. The third version (GeMNMF) uses the matrix of the form

$$\mathbf{G} = \alpha_0\mathbf{S} + \alpha_1\mathbf{e}\mathbf{u}_1^T + ... + \alpha_p\mathbf{e}\mathbf{u}_p^T,$$

where $\alpha_0 = 0.6$, $p = 2$, and vectors $\mathbf{u}_1, \mathbf{u}_2$ and α_1, α_2 are formed using the Nonnegative Matrix Factorization. The final version of GeM (GeMFM) uses the matrix $\mathbf{G}$ of the form

$$\mathbf{G} = \alpha_0\mathbf{S}_0 + \alpha_1\mathbf{S}_1 + ... + \alpha_p\mathbf{S}_p,$$

where $p = 2$, and matrices $\mathbf{S_1}$ and $\mathbf{S}_2$ are feature matrices formed using rushing yards and first down game statistics. The value of $\alpha = 0.6$ ($\alpha_0 = 0.6$) was chosen based on empirical experiments.

Table 7.3: Foresight game prediction percentages for NFL (with GeM).

	Colley	GeMPV	GeMODM	GeMNMF	GeMFM	Keener	Massey
2001	57.92	57.92	59.46	55.60	58.30	58.69	60.23
2002	59.18	56.18	58.80	55.81	55.06	58.43	60.30
2003	63.30	54.68	53.56	53.18	55.81	58.05	64.04
2004	61.80	61.42	61.80	64.42	58.43	59.93	62.17
2005	61.80	65.54	62.92	65.54	62.17	62.55	65.17
2006	58.80	57.68	60.30	58.43	56.93	57.68	60.30
2007	66.67	62.92	65.54	64.17	61.80	62.55	68.16

The two matrix scaling based models (DSS and ODM) are very close in the prediction results with the ODM outperforming DSS in general. In both hindsight experiments the Colley method tends to outperform other models. However in the foresight experiments the Massey model exhibits better predictions closely followed by the DSS and the ODM. There are two empirical observations first related to the GeM and the other related to the ODM that should be mentioned here. First, NFL game predictions via the GeM model are not sensitive to small perturbations of α's. Second,

Table 7.4: Hindsight game prediction percentages for NFL (with GeM).

	Colley	GeMPV	GeMODM	GeMNMF	GeMFM	Keener	Massey
2001	72.97	70.27	70.66	70.66	70.66	72.97	69.88
2002	68.16	66.67	67.42	69.29	66.67	68.91	67.04
2003	75.66	69.66	69.29	68.54	65.17	73.78	71.91
2004	74.16	69.66	70.41	69.66	67.04	70.79	67.42
2005	73.03	75.66	73.78	73.78	70.04	75.66	75.28
2006	72.66	64.79	64.42	65.92	63.67	69.29	71.16
2007	75.66	71.91	72.28	73.03	68.91	76.03	73.41

NFL game predictions via ODM are not sensitive to small perturbations in ϵ. Hence the choices of α (or α_0) and k for GeM versions are based on empirical experiments and the chosen values appear to produce higher prediction results.

7.4 Game Prediction Results for NCAA Football

Both foresight and hindsight predictions are done for the NCAA football. Foresight game predictions are done for the regular season of NCAA football starting with week 6 all the way through the Bowl games. Since NCAA football does not have a pre-season the game predictions cannot start until week 6 of the regular season. Only Div I-A teams are considered. The games played between I-A and I-AA are discarded. Both foresight and hindsight prediction results for NCAA football are in Tables 7.5 and 7.6. In both experiments the tolerance used for DSS and ODM is 0.01 and for the ODM $\epsilon = 0.00001$.

Table 7.5: Foresight game prediction percentages for NCAA football (with DSS and ODM).

	Colley	DSS	Keener	Massey	ODM
2003	66.30	67.85	70.29	69.62	68.96
2004	66.14	69.28	63.68	67.71	66.82
2005	67.34	66.44	64.21	67.79	64.43
2006	68.74	71.95	65.74	73.23	71.73
2007	67.10	68.17	68.82	69.89	68.60

Table 7.6: Hindsight game prediction percentages for NCAA football (with DSS and ODM).

	Colley	DSS	Keener	Massey	ODM
2003	82.04	77.61	76.72	77.38	76.05
2004	81.17	77.13	78.03	76.91	79.15
2005	81.66	74.94	77.63	76.06	74.27
2006	82.23	76.23	78.37	77.09	77.30
2007	79.35	76.13	76.13	77.42	75.48

The game predictions using versions of GeM appear in tables 7.7 and 7.8. The α (α_0) corresponding to the score differences was set to 0.6. For the GeMODM we set $\alpha_1 = \alpha_2 = 0.2$. The GeMNMF used $k = 2$ and GeMFM used the total yards and the rushing yards.

Table 7.7: Foresight game prediction percentages for NCAA football (with GeM).

	Colley	GeMPV	GeMODM	GeMNMF	GeMFM	Keener	Massey
2003	66.30	66.30	68.29	68.51	66.96	70.29	69.62
2004	66.14	65.02	62.56	61.66	66.14	63.68	67.71
2005	67.34	67.34	66.67	64.21	66.89	64.21	67.79
2006	68.74	67.24	67.45	67.88	64.88	65.74	73.23
2007	67.10	64.30	64.09	62.80	64.30	68.82	69.89

Table 7.8: Hindsight game prediction percentages for NCAA football (with GeM).

	Colley	GeMPV	GeMODM	GeMNMF	GeMFM	Keener	Massey
2003	82.04	77.38	73.61	73.39	72.95	76.72	77.38
2004	81.17	79.15	76.46	76.91	78.03	77.80	76.91
2005	81.66	75.39	74.94	74.50	74.27	77.63	76.06
2006	82.23	78.16	77.30	76.23	72.38	78.37	77.09
2007	79.35	74.84	72.47	73.12	69.03	76.77	77.42

The Colley Method appears to benefit the most from the hindsight game knowledge. So does the simple version of GeM, GeMPV. However, in the case of foresight predictions Massey outperforms the other models. For both experiments all models appear to produce results that are relatively close to one another.

7.5 Game Prediction Results for NCAA Basketball

The largest example of the game prediction is done with NCAA basketball. Since teams may play more than once in any one week the game predictions have to be done daily. As with previous sports, since the total number of games played varies from year to year the predictions are converted to percentages. Only the games between the Division I teams were used. The games of the Division I teams with teams outside of the Division I were disregarded. Since there is no pre-season then foresight predictions wait until game day 26 to start the predictions and are predicted through the tournament. The viable starting point may be earlier for some seasons depending on the number of games we have to disregard in the initial game days. The hindsight predictions used the entire season together with the tournament to compute the ratings. These ratings were then used to predict games daily for the same season. Finally, given the tolerance=0.01 for ODM convergence the NCAA basketball foresight and hindsight game prediction results are shown in tables 7.9 and 7.10.

Table 7.9: Foresight game prediction percentages for NCAA basketball.

	Colley	DSS	GeMPV	Keener	Massey	ODM
2001	68.60	70.11	66.85	64.60	69.65	70.03
2002	69.02	69.72	67.67	64.79	70.13	70.03
2003	68.92	69.95	68.89	64.92	70.19	70.22
2004	68.65	70.40	68.09	65.27	70.50	70.12
2005	66.98	69.48	67.13	64.44	68.95	69.52
2006	68.37	69.76	67.63	64.84	70.02	69.69
2007	68.28	70.58	67.84	64.91	70.13	70.09

Table 7.10: Hindsight game prediction percentages for NCAA basketball.

	Colley	DSS	GeMPV	Keener	Massey	ODM
2001	75.41	73.96	72.53	70.51	74.20	73.80
2002	76.25	74.59	73.35	71.61	74.59	74.62
2003	76.04	74.90	73.43	70.41	74.77	74.57
2004	75.39	74.48	73.59	71.16	74.12	74.43
2005	75.72	74.07	73.71	69.91	74.36	74.09
2006	75.73	73.98	73.09	69.57	74.03	73.81
2007	75.38	74.13	73.04	70.23	74.11	74.08

There is a clear and expected increase in the prediction accuracy if we use the

entire season's data to compute the ratings. Prediction results for all three sports yield an interesting observation that ODM does better as a foresight predictor relative to the other ranking models than it does as a hindsight predictor. Furthermore both Colley and Keener excel in hindsight but not foresight predictions. Another point of interest for us is the computation time. Given that foresight predictions of NCAA basketball games is the largest of the examples and hence it is used to compute the total cpu time expended and the number of iterations used by ODM. Provided that the tolerance for the ODM is set to be 0.01 the Table 7.11 illustrates total amount of cpu time expended for each of the methods and the Table 7.12 considers the total number of iterations performed by ODM for different values of ϵ.

Table 7.11: Total cpu time (sec) for each of the methods on NCAA basketball.

	Colley	Keener	Massey	ODM $\epsilon = 0.001$	ODM $\epsilon = 0.0001$	ODM $\epsilon = 0.00001$
2001	1.78125	162.90625	2.0625	0.90625	0.953125	0.9375
2002	1.5625	161.234375	2.015625	0.84375	0.921875	1.1875
2003	1.671875	167.609375	2.125	1.03125	1.0625	1.125
2004	1.796875	170.046875	2.09375	1.03125	0.90625	0.875
2005	1.84375	174.96875	2.03125	1.109375	1.03125	1.140625
2006	1.96875	181.5	2.1875	0.90625	0.953125	1.03125
2007	2.03125	207.09375	2.453125	1.140625	1.1875	1.203125

Table 7.12: Total number of iterations done by ODM for each of the seasons on NCAA basketball.

	$\epsilon = 0.001$	$\epsilon = 0.0001$	$\epsilon = 0.00001$
2001	1576	1577	1577
2002	1690	1866	2174
2003	1622	2000	2619
2004	1515	1605	1745
2005	1748	1997	2402
2006	1421	1465	1532
2007	1555	1638	1776

Since there is no significant increase in the number of iterations it would make sense to decrease ϵ, however, as mentioned before, the empirical experiments showed that game predictions are not sensitive perturbations in ϵ.

7.6 Game Predictions Using Aggregation

In section 5.5 we used a very simple aggregation method to produce an overall team rating from the offense and defense ratings produced by the ODM. Another aggregation method called the Graph Theory Method of Rank Aggregation has been suggested by Dr. Amy Langville. As an input it takes several ranked lists of n teams. We then form a directed graph where all the teams are represented by the graph nodes. The edges point from lower ranked teams to higher ranked teams and each weight can be defined as

$$w_{ij} = \text{number of ranked lists having } i \text{ below } j,$$

or

$$w_{ij} = \text{sum of rank differences from lists having } i \text{ below } j,$$

Then apply any ranking method to compute ratings and form an aggregated ranked list. In essence each ranked list can represent a round robin tournament where the rank of the team is a score and the team with the lowest score wins. In general the goal of rank aggregation is to minimize the effect of the outliers, i.e. the teams that are assigned a rank by one method that is significantly different from the ranks assigned by the majority of the other methods.

We use rank differences for the weights and the aggregated ratings are produced using the ODM. If we were to be concerned with game predictions only, then there is another simple aggregation approach that is worth mentioning. Given six ranked lists (Aggregated, Colley, GeMPV, Keener, Massey, and ODM) we predict the game outcomes using each of the lists and then use these prediction to form an overall prediction. If a majority of the ranked lists predicts team A to beat team B in a given game then the overall prediction is team A wins. If there is a tie in predictions (two of the ranked lists predict A to win and one predicts a tie) then we can chose which method breaks the tie. In our experiments we chose the ODM to be the tie breaker. Both aggregation methods use a subset of models. For example, there are four versions of GeM, but the predictions produced by them are fairly similar. If all four versions are used in aggregation they can overwhelm the contributions by the other models. The predictions using the DSS model are also excluded since they are similar to the predictions done using

the ODM. The foresight game prediction results for the aggregated methods as well as other five ranking models of the NFL, the NCAA football, and the NCAA basketball appear in the Tables 7.13, 7.14, 7.15.

Table 7.13: Foresight game prediction percentages for the NFL.

	Aggregated Predictions	Aggregated Rank	Colley	GeMPV	Keener	Massey	ODM
2001	61.39	61.78	57.92	57.92	58.69	60.23	60.62
2002	60.30	59.55	59.18	56.18	58.43	60.30	63.30
2003	61.42	60.67	63.30	54.68	58.05	64.04	61.05
2004	60.67	61.80	61.80	61.42	59.93	62.17	58.43
2005	64.79	65.17	61.80	65.54	62.55	65.17	64.04
2006	57.30	56.55	58.80	57.68	57.68	60.30	58.05
2007	66.67	66.67	66.67	62.92	62.55	68.16	68.91

The aggregations for the NFL do not tend to consistently outperform the predictions done by individual aggregated models. This could be due to the fact that NFL teams are fairly even in "quality" and there is small discrepancy among the models as to picking the winners.

Table 7.14: Foresight game prediction percentages for NCAA football.

	Aggregated Predictions	Aggregated Rank	Colley	GeMPV	Keener	Massey	ODM
2003	71.40	70.95	66.30	66.30	70.29	69.62	68.96
2004	67.26	67.04	66.14	65.02	63.68	67.71	66.82
2005	67.79	68.01	67.34	67.34	64.21	67.79	64.43
2006	71.31	71.31	68.74	67.24	65.74	73.23	71.73
2007	68.82	69.89	67.10	64.30	68.82	69.89	68.60

The aggregation for NCAA football is more successful and outperforms the best of the predictions by individual models. The "quality" of NCAA football teams (Div I-A) varies significantly, which leads to higher prediction discrepancy among the models. Thus there are higher chances of outliers which are teams picked to be winners by a small number of models. If the team that is an outlier should not be picked a winner then aggregation produces a better result.

NCAA basketball aggregations include only three models: Colley, Massey, and ODM. Since both Keener and GeM ratings produced lower predictions they were

excluded from the aggregations. As before, the tolerance for ODM is set to be 0.01 and predictions start with the game day 26 and continue until the end of the NCAA basketball tournaments.

Table 7.15: Foresight game prediction percentages for NCAA basketball.

	Aggregated Predictions	Aggregated Rank	Colley	Massey	ODM
2001	70.06	69.90	68.60	69.65	70.03
2002	70.16	70.24	69.02	70.13	70.03
2003	69.97	70.46	68.92	70.19	70.22
2004	70.40	70.07	68.65	70.50	70.12
2005	69.33	69.07	66.98	68.95	69.52
2006	70.00	69.54	68.37	70.02	69.69
2007	70.09	69.55	68.28	70.13	70.09

The aggregated prediction results for NCAA basketball do not appear to be much better than the individual predictions. NCAA basketball is the largest example in terms of the number of teams and the number of games played. There is greater disparity in quality among college teams versus pro basketball teams. However, increased number of games in NCAA basketball compared to NCAA football allows for less discrepancy in the computed ranks, hence less outliers.

Chapter 8

Conclusion

This work had two main goals. The first was to introduce new ranking methods and the second was to apply them to sports and perform model verification via game predictions. The assumption is that better ranks of sports teams produce more accurate game predictions.

In the first place two ranking models were introduced, the Offense-Defense Model and the Diagonal Similarity Scaling model, both derived from a nonnegative matrix balancing. Both models are easy to implement, tend to be fast, and perform well in game predictions. The ODM has an advantage over the DSS model in that it provides the user with more information. The ODM model computes two rating scores for each team. The combination of these ratings provides the overall score, but having offensive and defensive scores available for each team may be useful in the point spread computation.

The Generalized Markov ranking model proposed in this work uses the theory of Markov Chains. The model has a built in flexibility that allows for inclusion of game information other than scores. With flexibility comes an additional need of investigation and computation of the parameters corresponding to the information to be included. The constrained least squares method proposed allows for both determining the statistics relevant to the score differences (which correspond to wins) and computing the values of the chosen parameters. There are four versions of the GeM model that were proposed and tested. All four versions appeared to produce predictions within a close range of each other.

Finally the game prediction experiments included collection and parsing of

game data as well as foresight and hindsight game predictions. The new ranking models fared well against current sports models(Colley, Keener, and Massey). However, the new models ODM and GeM include flexibility that was not completely explored in this work and could potentially be used to improve the predictions.

REFERENCES

[1] Eva Achilles. Implications of Convergence Rates in Sinkhorn Balancing. *Linear Algebra and Its Applications*, 187:109–112, 1993.

[2] Iqbal Ali, Wade D. Cook, and Moshe Kress. On the Minimum Violations Ranking of a Tournament. *Management Science*, 32(6):660–672, 1986.

[3] David H. Annis and Bruce A. Craig. Hybrid Paired Comparison Analysis, with Applications to the Ranking of College Football Teams. *Journal of Quantitative Analysis in Sports*, 1(1), 2005.

[4] M. Bacharach. *Biproportional Matrices and Input-Output Change*. Cambridge University Press, 1970.

[5] H. Balakrishnan, I. Hwang, and CJ Tomlin. Polynomial approximation algorithms for belief matrix maintenance in identity management. In *Proceedings of the 43rd International IEEE Conference on Decision and Control*, volume 5, pages 4874–4879, 2004.

[6] Hamsa Balkrishnan, Inseok Hwang, and Claire J. Tomlin. Polynomial approximation algorithms for belief matrix maintenance in identity management. In *43rd IEEE Conference on Decision and Control*, volume 5, pages 4874–4879, 2004.

[7] R. B. Bapat and T. E. S. Raghavan. An extension of a theorem of Darroch and Ratcliff in loglinear models and its application to scaling multidimensional matrices. *Linear Algebra and its Applications*, 114/115:705–715, 1989.

[8] Ravindra Bapat. $\mathbf{D}_1\mathbf{A}\mathbf{D}_2$ theorems for multidimensional matrices. *Linear Algebra and its Applications*, 48:437–442, 1982.

[9] Ed Barbeau. Perron's Result and a Decision on Admissions Tests. *Mathematics Magazine*, 59:12–22, 1986.

[10] D.F. Bauer. Circular Triads When not all Paired Comparisons are Made. *Biometrics*, 34:458–461, 1978.

[11] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press Inc., 1979.

[12] Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics and Data Analysis.*, 52(1):155–173, 2007.

[13] T.G.G. Bezembinder. Circularity and consistency in paired comparisons. *The British journal of mathematical & statistical psychology*, 34:16–37, 1981.

[14] Alberto Borobia. Matrix Scaling: A Geometric Proof of Sinkhorn's Theorem. *Linear Algebra and its Applications*, 268:1–8, 1998.

[15] Ralph Allan Bradley and Milton E. Terry. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[16] L. M. Bregman. Proof of the convergence of Sheleikhovskii's method for a problem with transportation constraints. *USSR Computational Mathematics and Mathematical Physics*, 7(1):191–204, 1967.

[17] L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

[18] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 33:107–117, 1998.

[19] Richard A. Brualdi, Seymour V. Parter, and Hans Schneider. The diagonal equivalence of a nonnegative matrix to a stochastic matrix. *Journal of Mathematical Analysis and Applications*, 16:31–50, 1966.

[20] Richard A. Brualdi and Li Qiao. Upsets in round robin tournaments. *Journal of Combinatorial Theory, Series B*, 35(1):62–77, 1983.

[21] Thomas Callaghan, Peter J. Mucha, and Mason A. Porter. Random walker ranking for NCAA Division I-A football. *American Mathematical Monthly*, 114:761–777, 2007.

[22] Jeromy Carriere and Rick Kazman. WebQuery: Searching and Visualizing the Web through Connectivity. In *6th International World-Wide Web Conference*, 1996.

[23] Yair Censor. On linearly constrained entropy maximization. *Linear Algebra and Its Applications*, 80:191–195, 1986.

[24] P. Chang, D. Mendona, X. Yao, and M. Raghavachari. An Evaluation of Ranking Methods for Multiple Incomplete Round-Robin Tournaments. In *35th Annual Meeting of Decision Sciences Institute*, 2004.

[25] Iréne Charon and Olivier Hudry. Links between the Slater Index and the Ryser Index of Tournaments. *Graphs and Combinatorics*, 19(3):309–322, 2003.

[26] Iréne Charon and Olivier Hudry. A survey on the linear ordering problem for weighted or unweighted tournaments. *A Quarterly Journal of Operations Research*, 5(1):5–60, 2007.

[27] A.T.W. Chu, R.E. Kalaba, and K. Spingarn. A Comparison of Two Methods for Determining the Weights of Belonging to Fuzzy Sets. *Optimization Theory and Applications*, 27(4):531–538, 1979.

[28] Wesley N. Colley. Colleys Bias Free College Football Ranking Method: The Colley Matrix Explained, 2002.
Colley matrix ranking method. Uses Laplace Rule of succession and strength of scheulde adjustment to derive the linear equation Cr=b, where C is the Colley matrix, r is the ratings vector and b is formed using wins and losses.

[29] Timothee Cour, Praveen Srinivasan, and Jianbo Shi. Balanced graph matching. In B. Schölkopf, J.C. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

[30] H.A. David. Ranking from unbalanced paired-comparison data. *Biometrika*, 74:432–436, 1987.

[31] Herbert A. David. *The Method of Paired Comparisons*. Oxford University Press, 1988.

[32] D. Z. Djokovic. Note on nonnegative matrices. In *Proceedings of the American Mathematical Society*, volume 25, pages 80–82, 1970.

[33] B. Curtis Eaves, Alan J. Hoffman, Uriel G. Rothblum, and Hans Schneider. Line-Sum-Symmetric Scaling of Square Nonnegative Matrices. *Mathematical Programming Study*, 25:124–141, 1985.

[34] Tommy Elfving. On some methods for entropy maximization and matrix scaling.

[35] P. Erdos and J. W. Moon. On Sets of Consistent Arcs in a Tournament. *Canadian Mathematical Bulletin*, 8(3):269–271, 1965.

[36] Jan Eriksson. A note on solution of large sparse maximum entropy problems with linear equality constraints. *Mathematical Programming*, 18(1):146–154, 1980.

[37] Sven Erlander. Entropy in linear programs. *Mathematical Programming*, 21(1):137–151, 1981.

[38] ESPN. http://scores.espn.go.com/ncf/scoreboard
Website holds stable NCAA football Box Scores files for each game from 2003 to present.

[39] G.T. Fechner. *Elemente der Psychophysik*. Breitkopf and Härtel, 1860.

[40] G.T. Fechner. *Elements of Psychophysics, Vol. 1, transl. H.E. Adler*. Holt, Rinehart & Winston, 1965.

[41] L. R. Ford. Solution of a Ranking Problem from Binary Comparisons. *The American Mathematical Monthly*, 64(8):28–33, 1957.

[42] Joel Franklin and Jens Lorenz. On the Scaling of Multidimensional Matrices. *Linear Algebra and its Applications*, 114/115:717–735, 1989.

[43] D.R. Fulkerson. Upsets in Round Robin Tournaments. *Canadian Journal of Mathematics*, 17:957–969, 1965.

[44] Eugene Garfield. Citation Analysis as a Tool in Journal Evaluation. *Science*, 178(4060):471–479, 1972.

[45] S.I. Gass. Tournamen, Transitivity and Pairwise Comparison Matrices. *Journal of the Operational Research Society*, 49(6):616–624, 1998.

[46] Nancy L. Geller. On the citation influence methodology of Pinski and Narin. *Information Processing & Management*, 14:93–95, 1978.

[47] Harold B. Gerard and Harold N. Shapiro. Determining the degree of inconsistency in a set of paired comparisons. *Psychometrika*, 23(1):33–46, 1958.

[48] Edward F. Gonzalez and Yin Zhang. Accelerating the Lee-Seung Algorithm for Nonnegative Matrix Factorization. Technical report, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005, 2005.

[49] Anjela Govan, Carl D. Meyer, and Russell Albright. Generalizing Google's PageRank to Rank National Football League Teams. In *Proceedings of the SAS Global Forum 2008*, 2008.

[50] Anjela Y. Govan, Amy N. Langville, and Carl D. Meyer. The Offense-defense approach to ranking team sports. *Journal of Quantitative Analysis in Sports*, 5, 2009.

[51] J. Grad. Matrix Balancing. *Computer Journal*, 14:280–284, 1971.

[52] Louis Guttman. An Approach for Quantifying Paired Comparisons and Rank Order. *The Annals of Mathematical Statistics*, 17(2):144–163, 1946.

[53] D.J. Hartfiel. Concerning Diagonal Similarity of Irreducible Matrices. *Proceedings of the American Mathematical Society*, 30:419–425, 1971.

[54] David Harville. Predictions for National Football League Games Via Linear-Model Methodology. *Journal of the American Statistical Association.*, 75(371):516–524, 1980.

[55] Taher Haveliwala and Sepandar Kamvar. The Second Eigenvalue of the Google Matrix. Technical report, Department of Computer Science, Stanford University, Stanford, CA 94305, Mar 2003.

[56] Taher Haveliwala, Sepandar Kamvar, and Glen Jeh. An Analytical Comparison of Approaches to Personalizing PageRank. Technical report, Department of Computer Science, Stanford University, Stanford, CA 94305, Jun 2003.

[57] Dorit S. Hochbaum. *Ranking Sports Teams and the Inverse Equal Paths Problem.* Springer Berlin / Heidelberg, 2006.

[58] Patrik O. Hoyer. Non-negative Matrix Factorization with Sparseness Constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

[59] Luke C. Ingram. Ranking NCAA sports teams with Linear algebra. Master's thesis, College of Charleston, Charleston, SC 29424, Apr 2007.

[60] Thomas Jech. The Ranking of Incomplete Tournaments: A Mathematician's Guide to Popular Sports. *The American Mathematical Monthly*, 90(4):246–266, 1983.

[61] Joseph B. Kadane. Some Equivalence Classes in Paired Comparisons. *The Annals of Mathematical Statistics*, 37(2):488–494, 1966.

[62] Bahman Kalantari and Leonid Khachiyan. On the rate of convergence of deterministic and randomized RAS matrix scaling algorithms. *Operations Research Letters*, 14:237–244, 1993.

[63] Bahman Kalantari and Leonid Khachiyan. On the Complexity of Nonnegative-Matrix Scaling. *Journal on Matrix Analysis and Applications*, 240:87–103, 1996.

[64] Bahman Kalantari, Leonid Khachiyan, and A. Shokoufandeh. On the Complexity of Matrix Balancing. *Journal on Matrix Analysis and Applications*, 18(2):450–463, 1997.

[65] Mikio Kano and Akio Sakamoto. Ranking the vertices of a paired comparison digraph. *SIAM Journal on Algebraic and Discrete Methods*, 6:79–92, 1985.

[66] James P. Keener. The Perron-Frobenius Theorem and the Ranking of Football Teams. *SIAM Review*, 35(1):80–93, 1993.

[67] M. G. Kendall. A New Measure of Rank Corelation. *Biometrika*, 30:81–89, 1938.

[68] M. G. Kendall. Further Contribution to the Theory of Paired Comparisons. *Biometrics*, 11(1):43–62, 1955.

[69] M. G. Kendall and B. Babington Smith. On the Method of Paired Comparisons. *Biometrika*, 31(3/4):324–345, 1940.

[70] Maurice Kendall and Jean Dickinson Gibbons. *Rank Correlation Methods*. Oxford University Press., 1990.

[71] Jon Kleinberg. Authoritative sources in a hyperlink environment. *Journal of the ACM*, 46(5), 1999.

[72] Philip A. Knight. The Sinkhorn-Knopp Algorithm: Convergence and Applications . Technical report, Department of Mathematics, University of Strathclyde, Glasgow G1 1XH Scotland, 2006.

[73] Philip A. Knight. The Sinkhorn–Knopp Algorithm: Convergence and Applications. *Journal on Matrix Analysis and Applications*, 30(1):261–275, 2008.

[74] Philip A. Knight and Daniel Ruiz. A fast algorithm for matrix balancing. In *Web Information Retrieval and Linear Algebra Algorithms*, 2007.

[75] J. Kruithof. Telefoonverkeersrekening. *Ingenieur*, 52:E15–E25, 1937.

[76] Paul Kvam and Joel S. Sokol. A Logistic Regression/Markov Chain Model for NCAA Basketball. *Naval Research Logistics*, 53:788–803, 2008.

[77] B. Lamond and N. F. Stewart. Bregman's balancing method. *Transportation Research Part B: Methodological*, 15(4):239–248, 1981.

[78] Hady Wirawan Lauw, Ee-Peng Lim, and Ke Wang. Summarizing review scores of "unequal" reviewers. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, 2007.

[79] Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, Inc., 1974.

[80] D. Lee and H. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[81] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2000.

[82] Jr. Lester R. Ford and Selmer M. Johnson. A Tournament Problem. *The American Mathematical Monthly*, 66(5):387–389, 1959.

[83] Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A Deterministic Strongly Polynomial Algorithm For Matrix Scaling and Approximate Permanents. *Combinatorica*, 20(4):545–568, 2000.

[84] Marvin Marcus and Henryk Minc. Some Results on Doubly Stochastic Matrices. *Proceedings of the American Mathematical Society*, 13(4):571–579, 1962.

[85] Marvin Marcus and Morris Newman. The permanent of a symmetric matrix, Abstract. *Notices of the American Mathematical Society*, 8:595, 1961.

[86] John Markoff. Debating the size of the web, Aug, 16 2005.

[87] Albert W. Marshall and Ingram Olkin. Scaling of matrices to achieve specified row and column sums. *Numerische Mathematik*, 12(1):83–90, 1968.

[88] Kahn Mason. *Detecting Colluders in PageRank: Finding Slow Mixing States in a Markov Chain.* PhD thesis, Stanford University, Stanford, CA 94305, Sep 2005.

[89] Kenneth Massey. http://www.mratings.com/.

[90] Kenneth Massey. Statistical models applied to the rating of sports teams, 1997.

[91] M. V. Menon. Reduction of a Matrix with Positive Elements to a Doubly Stochastic Matrix. *Proceedings of the American Mathematical Society*, 18(2):244–247, 1967.

[92] M. Merritt and Y. Zhang. Interior-Point Gradient Method for Large-Scale Totally Nonnegative Least Squares Problems. *Journal Of Optimization Theory And Applications.*, 126(1):191–202, 2005.

[93] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra.* SIAM, 2001.

[94] Carl D. Meyer and Amy N. Langville. *Google's PageRank and Beyond: The Science of Search Engine Rankings.* Princeton University Press, 2006.

[95] Carl D. Meyer, Amy N. Langville, and Russell Albright. Initializations for the Nonnegative Matrix Factorization. In *Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[96] H. Minc. *Permanents*. Addison-Wesley, 1978.

[97] Henryk Minc. A note on an Inequality of M. Marcus and M. Newman. *Proceedings of the American Mathematical Society*, 14(6):890–892, 1963.

[98] Cleve Moler. The world's largest matrix computation, 2002.

[99] J. W. Moon and N. J. Pullman. On Generalized Tournament Matrices. *SIAM Review*, 12(3):384–399, 1970.

[100] J.W. Moon. *Topics on tournaments*. Holt, 1968.

[101] Amy Oliver. Ranking NFL teams. Master's thesis, North Carolina State University, Raleigh, NC 27695, Mar 2006.

[102] E. E. Osborne. On Pre-Conditioning of Matrices. *Journal of the ACM*, 7(4):338–345, 1960.

[103] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Department of Computer Science, Stanford University, Stanford, CA 94305, Jan 1998.

[104] B. N. Parlett and T.L. Landis. Methods for Scaling to Doubly Stochastic Form. *Linear Algebra and Its Applications*, 48:53–79, 1982.

[105] B.N. Parlett and C. Reinsch. Balancing a Matrix for Calculation of Eigenvalues and Eigenvectors. *Numerical Mathematics*, 13(4):293–304, 1969.

[106] Hazel Perfect and L. Mirsky. The Distribution of Positive Elements in Doubly-stochastic Matrices. *Journal of the London Mathematical Society*, 40:689–698, 1965.

[107] Gabriel Pinski and Francis Narin. Citation influence For Journal Aggregates of Scientific Publications: Theory, with Applications to the Literature of Physics. *Information Processing & Management*, 12:297–312, 1976.

[108] T. E. S. Raghavan. On pairs of multidimensional matrices. *Linear Algebra and its Applications*, 62:263–268, 1984.

[109] Charles Redmond. A Natural Generilazation of the Win-Loss Rating System. *Mathematics Magazine*, 76(2):119–126, 2003.

[110] Sheldon Ross. *A First Course in Probability.* Pearson Prentice Hall, 2006.

[111] Uriel G. Rothblum. Scaling of matrices which have pre specified row sums and column sums via optimization. *Linear Algebra and Its Applications*, 114/115:737–764, 1989.

[112] Uriel G. Rothblum, Hans Schneider, and Michael H. Scneider. Scaling Matrices to Prescribed Row and Column Maxima. *Journal on Matrix Analysis and Applications*, 15(1):1–14, 1994.

[113] Daniel Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical report, Computational Science and Engineering Department, Science and Technology Facilities Council, Sep 2001.

[114] H.J. Ryser. Matrices of zeros and ones in combinatorial mathematics. In *Recent Advances in Matrix Theory (H. Scheider, Ed.).* University of Wisconsin Press, 1964.

[115] Thomas L. Saaty. *Analytic Hierarchy Process.* McGraw-Hill International Book Company, 1980.

[116] Thomas L. Saaty. Inconsistency and Rank Preservation. *Journal of Mathematical Psychology*, 28:205–214, 1984.

[117] Thomas L. Saaty. Rank According to Perron: A New Insight. *Mathematics Magazine*, 60(4):211–213, 1987.

[118] Michael H. Schneider. Matrix Scaling, Entropy Minimization, and Conjugate Duality. I. Existence Conditions. *Linear Algebra and Its Applications*, 114/115:785–813, 1989.

[119] Michael H. Schneider. Matrix Scaling, Entropy Minimization, and Conjugate Duality (II): The Dual Problem. *Mathematical Programming*, 48:103–124, 1990.

[120] Michael H. Schneider and Stavros A. Zenios. A Comparative Study of Algorithms for Matrix Balancing. *Operations Research*, 38(3):439–455, 1990.

[121] R. Sinkhorn. A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

[122] Richard Sinkhorn. Diagonal Equivalence to Matrices with Prescribed Row and Column Sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

[123] Richard Sinkhorn and Paul Knopp. Concerning Nonnegative Matrices and Doubly Stochastic Matrices. *Pacific Journal Of Mathematics.*, 21(2):343–348, 1967.

[124] Patrick Slater. Inconsistencies in a Schedule of Paired Comparisons. *Biometrika*, 48(3/4):303–312, 1961.

[125] Warren D. Smith. Sinkhorn ratings, and new strongly polynomial time algorithms for sinkhorn balancing, perron eigenvectors, and markov chains, 2005.

[126] George W. Soules. The Rate of Convergence of Sinkhorn Balancing. *Linear Algebra and its Applications*, 150:3–40, 1991.

[127] C. Spearman. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[128] Ray Stefani and Richard Pollard. Football Rating Systems for Top-Level Competition: A Critical Survey. *Journal of Quantitative Analysis in Sports*, 3(3), 2007.

[129] Michael Stob. A Supplement to "A Mathematician's Guide to Popular Sports". *The American Mathematical Monthly*, 91(5):277–282, 1984.

[130] L. L. Thurstone. A law of comparative judgment. *Psychological Review*, 34(4):273–286, 1927.

[131] L. L. Thurstone. Psychophysical Analysis. *The American Journal of Psychology*, 38(3):368–389, 1927.

[132] L. L. Thurstone. The method of paired comparisons for social values. *The Journal of Abnormal and Social Psychology*, 21(4):384–400, 1927.

[133] John M. Troan. http://www.jt-sw.com/football/boxes/index.nsf
Website holds stable NFL Box Scores files of each game from 1994 to present.

[134] Jr. W. A. Thompson and Jr. Russell Remage. Rankings from Paired Comparisons. *The Annals Of Mathematical Statistics*, 35(2):739–747, 1964.

[135] T.H. Wei. *The algebraic foundations of ranking theory.* PhD thesis, Cambridge University, England, 1952.

[136] Brady T. West. A Simple and Flexible Rating Method for Predicting Success in the NCAA Basketball Tournament. *Journal of Quantitative Analysis in Sports.*, 2(3), 2006.

[137] Stefan Wild, James Curry, and Anne Dougherty. Motivating non-negative matrix factorizations. In *Proceedings of the Eighth SIAM Conference on Applied Linear Algebra*, July 2003.

[138] Dell Zhang and Yisheng Dong. An efficient algorithm to rank Web resources. *Computer Networks*, 33:449–455, 2000.

APPENDICES

Appendix A

Matlab code

A.1 Ranking Models

```
%%%%%%%%%%%%%%%%%%%%%%%%%% colley.m %%%%%%%%%%%%%%%%%%%%%%%%%%
%% Colley Ranking Model
%%
%% INPUT:  T  = n-by-n symmetric matrix
%%              T(i,j)=-number of games played by i and j
%%                     =0, otherwise
%%              T(i,i)=0
%%          N = n-ny-3 vector
%%              N(i,1) = number of wins by team i
%%              N(i,2) = number of loses by team i
%%              N(i,3) = total number of games played by team i
%%
%% OUTPUT:  r = vector of ratings
%%

function [r] = colley(T,N)
teams=size(T,1);
b=ones(teams,1);
r=zeros(teams,1);

%% SET UP COLLEY MATRIX C
C=T+2*eye(teams,teams)+diag(N(:,3));

%% SET UP VECTOR b
b=b+0.5*(N(:,1)-N(:,2));

r=inv(C)*b;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%% DSS.m %%%%%%%%%%%%%%%%%%%%%%%%%%%
%% INPUT:    A = n-by-n nonsymmetric matrix of point scores;
%%               A(i,j)=# of points that team j scored on team i;
%%     epsilon = used to force total support
%%         tol = tolerance for convergence criterion of ODM
%%               algorithm;
%%
%% OUTPUT:   r = overall ratings vector of DSS
%%        iter = number of iterations required by ODM

function [r,iter]=DSS(A,tol)
r=ones(size(A,2),1);
col_row_diff=1;
iter=1;

while col_row_diff>tol
u=sum(A,2); %% ROW SUMS
v=sum(A',2); %% COLUMN SUMS

d=sqrt(v./u);
r=r.*d;

A=A*inv(diag(d)); %% RESCALE COLUMNS
A=diag(d)*A;      %% RESCALE ROWS

col_row_diff=max(abs(sum(A,2)-sum(A',2)));
iter=iter+1;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%% gem.m %%%%%%%%%%%%%%%%%%%%%%%%%%
%% The Generalized Markov Ranking Model
%%
%% INPUT:  SD = n-by-n nonsymmetric matrix of point scores;
%%               SD(i,j)=amount of points by which team i lost
%%               to team j;
%%       rows =  rows used to replace zero rows of SD
%%     alphas =  values of alpha_0, alpha_1, ..., alpha_k
%%per_vectors =  either feature vectors or feature matrices
%%               [FV_2 FV_2 ... FV_k] or
%%               [FM_1 FM_2 ... FM_k]
%%
%% OUTPUT: pi = vector of ratings

function [pi]=gem(SD,rows,alphas,per_vectors)
format long

%%---------------------------
%% INITIALIZE
teams=size(SD,1);
e=ones(teams,1);

nalphas=size(alphas,2);
npv=size(per_vectors,2);


%%---------------------------
%% TURN A SCORE DIFFERENCE MATRIX INTO A STOCHASTIC MATRIX
%% REPLACE EACH ZERO ROW OF SD BY THE CORRESPONDING ROW IN rows
zeroi=find(sum(SD,2)==0);
for i=1:size(zeroi)
    SD(zeroi(i),:)=rows(i,:);
end
%% NORMALIZE EACH ROW OF MATRIX SD SO IT SUMS TO 1
St=SD.*((1./sum(SD,2))*ones(1,teams));


%%---------------------------
%%  FORM THE GeM MATRIX G
G=alphas(1)*St;
if(size(per_vectors,2)<teams)
    for i=2:nalphas
      G=G+alphas(i)*e*per_vectors(:,i-1)';
    end
```

```
else
    for i=1:size(per_vectors,2)/teams
      S=zeros(teams,teams);
      alphai=alphas(i+1);

      Temp=per_vectors(:,((i-1)*teams+1):(i*teams));
      zeroi=find(sum(Temp,2)==0);
      for i=1:size(zeroi)
          Temp(zeroi(i),:)=ones(1,teams);
      end
      S=Temp.*((1./sum(Temp,2))*ones(1,teams));

      G=G+alphai*S;
    end

end

%%---------------------------
%%  COMPUTE THE RATINGS
pi=null(G'-eye(teams,teams));
pi=(1/sum(abs(pi)))*abs(pi);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%% keener.m %%%%%%%%%%%%%%%%%%%%%%%%%
%% Keener Ranking Model
%%
%% INPUT:  S  = n-by-n nonsymmetric matrix of point scores;
%%              S(i,j)=amount of points team i scored
%%              against team j;
%%
%% OUTPUT:  r = vector of ratings

function [r] = keener(S)
m=size(S,1);

%%---------------------------
%% CREATE KEENER MATRIX K
K=zeros(m,m);
for i=1:m
    for j=1:m
        if S(i,j)~=0|S(j,i)~=0
        K(i,j)=h((S(i,j)+1)/(S(i,j)+S(j,i)+2));
        end
    end
end

[V,D]=eig(K);
lambda=0;
index=0;

%%---------------------------
%% DETERMINE THE MAXIMUM EIGENVALUE OF K
[lambda,index]=max(max(D,[],2));

r=abs(V(:,index));

%%%%%%%%%%%%%%%%%%%%%%%%%% h.m %%%%%%%%%%%%%%%%%%%%%%%%%%
%% Keener smoothing function
%%
%% INPUT:  x = ratio
%%
%% OUTPUT: y = value between 0 and 1

function [y] = h(x)
y=0.5+0.5*sign(x-0.5)*sqrt(abs(2*x-1));
```

```
%%%%%%%%%%%%%%%%%%%%%%%% massey.m %%%%%%%%%%%%%%%%%%%%%%%%
%% Massey Ranking Model
%%
%% INPUT:  X  = n-by-n symmetric matrix
%%              X(i,j)=-number of games played by i and j
%%                     =0, otherwise
%%              X(i,i)=0
%%         SD = n-by-1 vector
%%              SD(i) = total score difference accumulated
%%                        by team i
%%          N = n-ny-1 vector
%%              N(i) = total number of games played by team i
%%
%% OUTPUT:  r = vector of ratings

function [r]=massey(X,SD,N)
t=size(X,1);
MM=diag(N)+X;


%% MAKE MASSEY MATRIX FULL RANK BY REMOVING ONE OF THE ROWS
%% (ONE GAME),IN THIS CASE THE FIRST ROW, AND REPLACING IT
%% WITH [1 1 ... 1 0]
MM(1,:)=ones(1,t);
SD(1,1)=0;

r=MM\SD;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%% ODM.m %%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The Offense-Defense Ranking Model
%%
%% INPUT:    A = n-by-n nonsymmetric matrix of point scores;
%%               A(i,j)=# of points that team j scored on team i;
%%     epsilon = used to force total support
%%         tol = tolerance for convergence criterion of ODM
%%               algorithm;
%%
%% OUTPUT: def = defensive ratings vector produced by ODM
%%         off = offensive ratings vector produced by ODM
%%           r = o./d = overall ratings vector of ODM
%%        iter = number of iterations required by ODM

function [def,off,r,iter] = ODM(A,epsilon,tol);
n=size(A,1);   %% NUMBER OF TEAMS

P=epsilon*ones(n,n); %% FORCE TOTAL SUPPORT
P=P+A;
e=ones(n,1);
off=e;
error=1;
iter=1;

while error > tol
    oldobar=off;
    def=P*(1./off);
    off=P'*(1./def);
    error=norm(oldobar.*(1./off)-e,1);
    iter=iter+1;
end
r=off./def;
```

A.2 Game Prediction Functions

```
%%%%%%%%%%%%%%%%%% RatingsVectorsNFL.m %%%%%%%%%%%%%%%%%%%%
%% MAIN function for computing NFL game predictions
%%
%% INPUT: beginweek = begin predictions starting beginweek+1
%%          endweek = end predictions endweek+1
%%
%%

function []=RatingsVectorsNFL(beginweek,endweek)
%%endweek-LAST WEEK PLAYED
%%
%%-------------------------------------------------------------------
%%        SET PARAMETERS, INITIALIZE FILES
%%-------------------------------------------------------------------
%%
beginseason=2007;
endseason=2007;

beginepsilon=5;
endepsilon=5;
tol=0.01;

perl('InitializeFilesNFL.pl',num2str(beginseason),num2str(endseason),
       num2str(beginepsilon),num2str(endepsilon),num2str(beginweek));

%%
%%-------------------------------------------------------------------
%%        PREDICTIONS, LOOP OVER SEASONS
%%-------------------------------------------------------------------
for season=beginseason:endseason
    y=num2str(season)
    fidalphas = fopen(strcat('C:\DataProcessing\NFL\GamePredictions\'
        ,y,'\GeMFMalphas.txt'), 'w');
    fclose(fidalphas);

    for ce=beginepsilon:endepsilon
       epsilon=10^(-ce);
       eps='0';
       for j=2:ce-1
           eps=strcat(eps,'0');
       end
       eps=strcat(eps,'1')
```

```
 %% ACCESS THE DIRECTORY CONTANINING THE CURRENT SEASON DATA FILES
 newpath=
    strcat('C:\MATLAB7\RESEARCH\NFL\RankingPrograms\NFL_DATA\',y);
 addpath(newpath);
 cd(newpath);

 %% INITIALIZE PICKS
 agr=[];
 c=[];
 g=[];
 k=[];
 m2=[];
 odm=[];

 %% UNCOMMENT IF DOING HINDSIGHT PREDICTIONS
 %perl('NFL_data.pl',y,'21');
 %pause(1);


for week=beginweek:endweek
    week
    w=num2str(week);
    pw=num2str(week+1); %% PREDICTION DAY IS ONE AHEAD OF THE DAY
                        %% WE ARE CURRENTLY PROCESSING

    %% UNCOMMENT IF DOING FORESIGHT PREDICTIONS
    perl('NFL_data.pl',y,w);
    pause(1);

%%
%%------------------------------------------------------------------------
%%        LOAD DATA
%%------------------------------------------------------------------------
if week>0 %% PLAYED AT LEAST ONE WEEK OF REGULAR SEASON
   S=GameScores;
   R=RushingYards;

   [C,N]=colley_matrix;
   [Mas,PD]=Massey_matrix;

   SD=GeM_Score_Diff;
   Diff=LastWeekScoreDiff;
   Stats=GameStats;
```

```
   YD=GameYards;
   RYD=GameRushingYards;
   PYD=GamePassingYards;
   ReYD=GameReturnYards;
   FD=GameFirstDowns;

   teams=size(S,1);

   if week<4   %% IF TOO FEW WEEKS PLAYED PADD ALL MATRICES WITH
               %% PRESEASON DATA;
               %% COMMENT OUT IF DOING HINDSIGHT PREDICTIONS
      preSD=preGeM_Score_Diff;
      preS=preGameScores;
      preR=preRushingYards;
      preStats=preGameStats;
      S=S+preS;
      R=R+preR;
      SD=SD+preSD;
      Stats=Stats+preStats;

      [preMas,prePD]=preMassey_matrix;
      Mas=Mas+preMas;
      PD=PD+prePD;

      [preC,preN]=precolley_matrix;
      C=C+preC;
      N=N+preN;

      preYD=preGameYards;
      YD=YD+preYD;
      preRYD=preGameRushingYards;
      RYD=RYD+preRYD;
      prePYD=preGamePassingYards;
      PYD=PYD+prePYD;
      preReYD=preGameReturnYards;
      ReYD=ReYD+preReYD;
      preFD=preGameFirstDowns;
      FD=FD+preFD;
   end
else   %% NO GAMES PLAYED IN THE REGULAR SEASON, USE PRESEASON
       %% DATA COMMENT OUT IF DOING HINDSIGHT PREDICTIONS
   S=preGameScores;
   R=preRushingYards;
```

```
   SD=preGeM_Score_Diff;
   Diff=preLastWeekScoreDiff;
   Stats=preGameStats;

   [C,N]=precolley_matrix;
   [Mas,PD]=preMassey_matrix;

   YD=preGameYards;
   RYD=preGameRushingYards;
   PYD=preGamePassingYards;
   ReYD=preGameReturnYards;
   FD=preGameFirstDowns;

   teams=size(S,1);
end
%%
%%-------------------------------------------------------------------------
%%       COMPUTE RATINGS
%%-------------------------------------------------------------------------
e=ones(teams,1);
rows=(1/teams)*ones(teams-nnz(sum(SD,2)),teams); %% SET UP REPLACEMENT
                                                 %% FOR GEM ZERO ROWS

%%---------------------
%%'Computing DSS'
t = cputime;
[dss,iter_dss]=DSS(S',tol);
dsstime=cputime-t;
iter_dss

%%---------------------
%%'Computing Colley'
t = cputime;
c=colley(C,N);
colleytime = cputime-t;

%%---------------------
%%'Computing GeM1 - Simple Version'
t = cputime;
[pi1]=gem(SD,rows,[0.6,0.4],(1/teams)*e);
gemtime1=cputime-t;

%%---------------------
%%'Computing GeM2 - Offense-Defense Vectors'
```

```
t = cputime;
[rd,ro,rodm,riter]=ODM(R',epsilon,tol);
[pi2]=gem(SD,rows,[0.6,0.2,0.2],[(1/sum(rd))*rd,(1/sum(ro))*ro]);
gemtime2=cputime-t;

%%---------------------
%%'Computing GeM3 - NMF'
[W,H]=LeeSeungNMF(Stats,2);    %% NOTE: k=2
vectors=W*inv(diag(sum(W,1))); %% NORMALIZE THE ROWS OF W, SO THEY
                               %% ADD TO 1

alpha=0.6;                     %% SET UP alpha_0
as=sum(H,2);                   %% COLUMN SUMS OF H WILL BE
totalas=sum(as);               %% alpha_1, ... ,alpha_k
for i=1:size(as)
    as(i)=(1-alpha)*(as(i)/totalas);
end

t = cputime;
[pi3]=gem(SD,rows,[alpha,as'],vectors);
gemtime3=cputime-t;

%%---------------------
%%'Computing GeM4 - Feature Matrices'
vectors=[RYD,FD];     %% STORE THE DESIRED STATS
Total{1}=SD;
%Total{}=YD;
Total{2}=RYD;
%Total{}=PYD;
Total{3}=FD;
%Total{}=ReYD;
alphas=AlphasLS(Total);

t = cputime;
[pi4]=gem(SD,rows,[0.6,0.4*alphas'],vectors);
gemtime4=cputime-t;

%%---------------------
%%'Computing Keener'
t = cputime;
k=keener(S);
keenertime=cputime-t;

%%---------------------
```

```
%%'Computing Massey'
t = cputime;
[m2]=massey(Mas,PD,N(:,3));
masseytime=cputime-t;

%%--------------------
%%'Computing ODM'
t = cputime;
[d,o,odm,iter]=ODM(S',epsilon,tol);
odmtime=cputime-t;

%%
%%-----------------------------------------------------------------------
%%     PRINT COMPUTED RATINGS INTO A FILE, PERL SCRIPTS USE THE
%%     COMPUTED RATINGS TO PERFORM GAME PREDICTIONS AND PRINT
%%     THE RESULTS INTO FILES
%%-----------------------------------------------------------------------
fid = fopen(strcat('C:\DataProcessing\NFL\GamePredictions\Ratings\
                                          ratings_',y,'.txt'), 'w');
fidagr = fopen(strcat('C:\DataProcessing\NFL\GamePredictions\Ratings\
                                       agr_ratings_',y,'.txt'), 'w');
fidalphas = fopen(strcat('C:\DataProcessing\NFL\GamePredictions\'
                                        ,y,'\GeMFMalphas.txt'), 'a');
%fidcpu = fopen(strcat('C:\DataProcessing\NFL\GamePredictions\
                                    Epsilon\cputime_',y,'.txt'), 'a');
%fiditer = fopen(strcat('C:\DataProcessing\NFL\GamePredictions\
                               Epsilon\ODMiterations_',y,'.txt'), 'a');

Temp=[c,dss,pi1,pi2,pi3,pi4,k,m2,odm];
fprintf(fid,'Colley,DSS,GeMSim,GeMODM,GeMNMF,GeMFM,Keener,Massey,
                                                              ODM\n');
fprintf(fid,'%18.15f,%18.15f,%18.15f,%18.15f,%18.15f,%18.15f,%18.15f,
                                          %18.15f,%18.15f\n', Temp');
fclose(fid);

Temp=[season,week,alphas'];
fprintf(fidalphas,'%g&%g&%5.4f&%5.4f\n', Temp');
fclose(fidalphas);


%% RANK AGGREGATION
perl('RankAggregationNFL.pl',y,strcat('0.',eps));
pause(1);
```

```
   %% RANK AGGREGATION PERFORMED BY ODM
   A=aggregate_matrix;

   t = cputime;
   [agrd,agro,agr,agiter]=ODM(A,epsilon,tol);
   agrtime=cputime-t;

   fprintf(fidagr,'Aggregated\n');
   fprintf(fidagr,'%18.15f,\n', agr');

   %% PRINT alphas COMPUTED FOR GeM
   %Temp2=[colleytime,gemtime1,keenertime,masseytime,odmtime];
   %fprintf(fidcpu,'%18.16f,%18.16f,%18.16f,%18.16f,%18.16f\n', Temp2');
   %fprintf(fiditer,'%g\n', iter);

   %fclose(fidcpu);
   %fclose(fiditer);
   fclose(fidagr);

   %% PRINT OUT THE MATCHUPS AND RESULTS OF THE GAMES TO BE PREDICTED
   perl('NFL_Results.pl',y,pw);

   %% PREDICT GAME OUTCOMES COMPARE TO THE TRUE RESULTS
   perl('NFLGamePredictions.pl',y,pw,strcat('0.',eps));
   pause(1);
   end %% WEEK LOOP

   perl('PrintTotalPrecentages.pl',y,strcat('0.',eps));
   %%perl('PrintTotalCPU_iter.pl',y,eps);
   rmpath(newpath);

   end % EPSILON LOOP
end % SEASON LOOP

%% SOUND EFFECT - END OF THE RUN
load chirp
sound(y,Fs)
```

```
%%%%%%%%%%%%%%%%%%%%%%%% AlphaLS.m %%%%%%%%%%%%%%%%%%%%%%%%
%% Computes alphas for GeM using least squares method.
%%
%% INPUT:  Total = list of k n-by-n nonsymmetric matrices
%%                 Total{p}(i,j) = positive difference of the
%%                 pth stat that i gave up to j
%%
%% OUTPUT: alphas = nonegative values of alphas for GeM

function [alphas]=AlphasLS(Total)
teamcount=size(Total{1},1);
statcount=size(Total,2);

%% ELIMINATE THE DIAGONAL ZEROS IN EACH OF THE AGGREGATED MATRICES
for i=1:statcount
    for j=1:teamcount
        T{i}(:,j)=[Total{i}(1:(j-1),j);Total{i}((j+1):teamcount,j)];
    end
end

%% TURN EACH STATISTIC MATRIX INTO A VECTOR, THAT IS THE jth COLUMN
%% IN THE MATRIX C
for j=1:statcount
    C(:,j)=reshape(T{j},[],1);
end

b=C(:,1);
C(:,1)=[];

%% CONSTRAINED LEAST SQUARES COMPUTATION OF alphas

%% ELIMINATE ZERO ROWS
Cbar=C(find(sum(C,2)),:);
bbar=b(find(sum(C,2)),:);

alphas = lsqnonneg(Cbar,bbar);
alphas=(1/sum(alphas))*alphas;
```

```
%%%%%%%%%%%%%%%%%%%%%% LeeSeungNMFm %%%%%%%%%%%%%%%%%%%%%%%%%
%% Computes a nonnegative matrix decomposition using Lee and
%% Seung algorithm.
%%
%% INPUT:  V = nonnegative n-by-n matrix
%%         k = dimention of the decomposition
%%
%% OUTPUT: W = n-by-k nonegative matrix
%%         H = k-by-n nonegative matrix

function [W H]=LeeSeungNMF(V,k)

[n m]=size(V);
W=abs(randn(n,k));
H=abs(randn(k,m));

for i=1:100000
    H=H.*(W'*V)./(W'*W*H+10^(-9));
    W=W.*(V*H')./(W*H*H'+10^(-9));
end
```

Appendix B

Perl scripts

B.1 Notes

Due to the page margin constraints the following Perl scripts had a number of lines broken into two or more. Some effort at reassembling the code maybe required before it works properly.

All the scripts assume a certain file structure. For the most part the expected data structure is summarized in the $DATAPATHNAME and $PATHNAME variables.

B.2 Acquiring NFL Data Files

```
#####################################################################
##
## EXECUTE
## Get game links form the given NFL website and writes
## them into a file
##
#####################################################################
#!/usr/bin/perl
use HTML::LinkExtor;
use LWP::Simple;

## SET UP PARAMETERS
my $year=2008;
my $totalweeks=4;
my $regularseason=17;
my $startweek=1;
```

```
## DIRECTORY WHERE THE FILE WITH LINKS IS TO BE WRITTEN
my $main_dir = sprintf ("c:/DataProcessing/NFL/Game_Links_NFL");


   $year=$year;
   print "processing year $year ...";
   open(FOUT,">$main_dir/game_links_$year.pl")
   or die "Could not open $main_dir/game_links_$year.pl";
   print FOUT "#!/usr/bin/perl\n\n";

   for($week=$startweek;$week<=$totalweeks;$week++){
       print FOUT "\@{\$games[$week]} = qw (";

       if ($week<10){
           $base_url="http://www.jt-sw.com/football/boxes/index.nsf/
        By/Week?OpenDocument&Season=$year&WeekType=Reg&Week=0$week";
       }
       elsif($week>=10 && $week<=$regularseason){
           $base_url="http://www.jt-sw.com/football/boxes/index.nsf/
        By/Week?OpenDocument&Season=$year&WeekType=Reg&Week=$week";
        }
elsif($week==$regularseason+1) {
           $base_url="http://www.jt-sw.com/football/boxes/index.nsf/
        By/Week?OpenDocument&Season=$year&WeekType=Post&Week=wc";
       }
elsif($week==$regularseason+2) {
           $base_url="http://www.jt-sw.com/football/boxes/index.nsf/
        By/Week?OpenDocument&Season=$year&WeekType=Post&Week=div";
       }
elsif($week==$regularseason+3) {
           $base_url="http://www.jt-sw.com/football/boxes/index.nsf/
        By/Week?OpenDocument&Season=$year&WeekType=Post&Week=conf";
       }
       else{
   $base_url="http://www.jt-sw.com/football/boxes/index.nsf/
        By/Week?OpenDocument&Season=$year&WeekType=Post&Week=sb";
       }

        $parser=HTML::LinkExtor->new(undef, $base_url);
        $parser->parse(get($base_url))->eof;
        @links=$parser->links;

     foreach $linkarray (@links){
```

```
my @element=@$linkarray;
my $elt_type=shift @element;
while (@element){
my ($attr_name,$attr_value)=splice(@element,0,2);
if($attr_value=~/boxes\/index.nsf\/8/)
                        {
      print FOUT "$attr_value\n"
                        }
}
      }
print FOUT ");\n\n";
  }
  print "Done\n";
close(FOUT);
```

```
#!/usr/bin/perl
#####################################################################
##
## Access the BoxScore games from the web and save them as text
## files. The links to the files were acuired using the previous
## Perl script
##
#####################################################################

#####################################################################
######################### getWeeklyGames ############################
sub getWeeklyGames () {
   use LWP::UserAgent;
   use HTML::FormatText;
   use HTML::Parse;

   my $week = $w;
   my $dir = $main_dir;
   my $totalweeks= $g;
   my $game=0;

   my $baseUrl = "http://www.jt-sw.com/football/boxes/index.nsf/";
   my $ext = ".txt";

   my @time = localtime();
   my $j=0;

   open(FOUT,">>c:/DataProcessing/NFL/Statistics$year/filepaths.txt")
        or die
   "Could not open/DataProcessing/NFL/Statistics$year/filepaths.txt";
   print "$week\n\n";

   if($week<10){
      print FOUT "week0$week\n";
   }
   else {
      print FOUT "week$week\n";
   }


   foreach $game (@{$games[$week]}) {
 $j++;

 if($week<10){
```

```
          $file = $dir . "/match0${week}_$j$ext";
          print FOUT "match0${week}_$j.txt\n";
  }
 else {
  $file = $dir . "/match${week}_$j$ext";
          print FOUT "match${week}_$j.txt\n";
 }


         my $url = $game;
         print "Getting $url . . . ";

         my $request = HTTP::Request->new(GET => $url);
         my $ua = LWP::UserAgent->new;

         my $response = $ua->request($request);
         print "Done\n";

         if ($response->is_success) {
            open (OUT, "> $file")
               or die "Can't open original $file: $!\n\n";
   print OUT $response->content;
            close OUT;
         }
        else {
            print $response->error_as_HTML;
            exit 1;
         }
 open (FIN, "+< $file") or die "Can't open saved $file: $!\n\n";
 $html=parse_htmlfile($file);
 $formatter=HTML::FormatText->new(leftmargin=>0,rightmargin=> 50);
 $ascii=$formatter->format($html);
 print FIN $ascii;
         close(FIN);
  } # end foeach loop
  if($week<$totalweeks){
  print FOUT "end\n";}
  else{
  print FOUT "end";}
  close(FOUT);
}  # end of subroutine

#####################################################################
############################ EXECUTE ################################
```

```
## SET UP PARAMTERS
$year=2008;
$g=4;
my $startweek=4;

require "c:/DataProcessing/NFL/Game_Links_NFL/game_links_$year.pl";
$main_dir = sprintf ("c:/DataProcessing/NFL/Statistics%4d", $year);

for($w=$startweek;$w<=$g;$w++){
getWeeklyGames();
}

#####################################################################
##
## Acquire the list of the NFL team names for each season, save
## as a text file and create arrays with valid team names and
## abbreviations. These arrays of names will allow the extraction
## of the data form the BoxScores.
##
#####################################################################

#####################################################################
############################## Abbreviate ############################
sub Abbreviate()
{
my $name = shift @_;
my $abbriviation="NONE";

  if($name=~/St. Louis Rams/){
    $abbriviation="STL";
    print "$name --> $abbriviation\n";
  }
  elsif($name=~/Arizona/){
    $abbriviation="ARZ";
    print "$name --> $abbriviation\n";
  }
  elsif($name=~/(\w+)\s(\w+)\s(\w+)/){
    $part1=$1;
    $part2=$2;
    $part3=$3;

    if($part2=~/York/){
    $abbriviation="NY".substr("$part3",0,1);
    print "$name --> $abbriviation\n";
```

```
    }
    elsif($part2=~/Angeles/){
    $abbriviation="LA".substr("\U$part3",2,1);
    print "$name --> $abbriviation\n";
    }
    else{
    $abbriviation=substr("\U$part1",0,1).substr("\U$part2",0,1);
    print "$name --> $abbriviation\n";
    }
  }
  else{
    #print "$name[1], $name[2], $name[3]\n";
    $abbriviation=substr("\U$name",0,3);
    print "$name --> $abbriviation\n";
  }

return $abbriviation;
}
#####################################################################
############################ EXECUTE ################################
#!/usr/bin/perl -w
use LWP::UserAgent;
use HTML::FormatText;
use HTML::Parse;

## SET UP PARAMETERS
my $startyear=2007;
my $endyear=2007;
my ($url,$request,$ua,$response);
my @names=();
my @abbr=();
my $count=0;
my $main_dir ="c:/DataProcessing/NFL/NFL_Roster";
mkdir $main_dir unless (-d $main_dir);

## PULL THE WEB PAGES WITH THE VALID TEAM NAMES FOR EACH SEASON
## AND SAVE THEM AS TEXT FILES
for($i=$startyear;$i<=$endyear;$i++){
   print "processing year $i ...";
   open(FOUT,">${main_dir}/TextRosters/roster_$i.txt")
or die "Could not open ${main_dir}/TextRosters/roster_$i.txt";

   $url="http://www.jt-sw.com/football/pro/rosters.nsf/
                            By/Season?OpenDocument&Season=$i";
```

```
   print "Getting $url . . . ";

   $request = HTTP::Request->new(GET => $url);
   $ua = LWP::UserAgent->new;

   $response = $ua->request($request);
   print "Done\n";

   if ($response->is_success) {
           $file = "${main_dir}/TextRosters/roster_$i.txt";
    print FOUT $response->content;
             close FOUT;
         }
        else {
            print $response->error_as_HTML;
            exit 1;
         }
 open (FIN, "+< $file") or die "Can't open saved $file:$!\n\n";
 $html=parse_htmlfile($file);
 $formatter=HTML::FormatText->new(leftmargin=>0,rightmargin=>
                                                                    50);
 $ascii=$formatter->format($html);
 print FIN $ascii;
         close(FIN);

print "Done\n";
close(FOUT);
}

## PARSE THE TEXT FILES WITH TEAM NAMES AND WRITE OUT THE TEAM NAMES IN
## THE FORM OF THE PERL ARRAY (AND MATLAB ARRAY), ALSO WRITE AN ARRAY
## OF ABBREVIATIONS
for($i=$startyear;$i<=$endyear;$i++){
   open(FOUT,">$main_dir/roster$i.pl")
            or die "Could not open $main_dir/roster$i.pl";
   print FOUT "#!/usr/bin/perl\n\n";
   open(FOUTM,">$main_dir/teams$i.m")
            or die "Could not open $main_dir/teams$i.m";

   open(FIN,"<$main_dir/TextRosters/roster_$i.txt")
           or die "Could not open $main_dir/TextRosters/roster_$i.txt";
   @fp=<FIN>;
   close(FIN);
   chomp(@fp);
```

```
foreach $line (@fp) {
   if($line=~/\*(\s+?)/){
$names[$count]=$';
$abbr[$count]=&Abbreviate($names[$count]);
$count++;
   }
}
## PRINT OUT PERL ARRAY OF TEAM NAMES
print FOUT "\@names=(";
my $begin=0;
foreach $name (@names) {
if($begin==0){
   print FOUT "\"$name\"";
   $begin=1;
        }
else{
   print FOUT ",\"$name\"";
}
}
print FOUT ");\n\n";

## PRINT OUT PERL ARRAY OF ABBREVIATIONS
print FOUT "\@abbr=(";
$begin=0;
foreach $a (@abbr) {
if($begin==0){
   print FOUT "\"$a\"";
   $begin=1;
        }
else{
   print FOUT ",\"$a\"";
}
}
print FOUT ");";

## PRINT OUT A MATLAB LIST OF TEAM NAMES
print FOUTM "function [T]=teams$i()\n";
print FOUTM "T={";
my $endarray=0;
foreach $name (@names) {
if($endarray==$#names){
   print FOUTM "'$name'};";
        }
```

```
   else{
      print FOUTM "'$name',\n";
      $endarray++;
   }
   }
close(FOUT);
close(FOUTM);
}
```

B.3 NFL Preseason Data

```
#!/usr/bin/perl

######################################################################
###                SUBROUTINES
######################################################################

######################################################################
##########  SETTING UP HASH TABLES FOR TEAM INDEX LOOK UP ######
sub SetUpNameHash()
{
require "${DATAPATHNAME}NFL_Roster/roster$year.pl";

my %team_names;
my $j=0;
my @entry=();

   for($i=0;$i<=$teamcount-1;$i++){
       $j=$i+1;
       $entry[0]=$j;
       $entry[1]=$abbr[$i];
       $team_names{$names[$i]} = [$j, $abbr[$i],$names[$i]];
    }
return %team_names;
}

######################################################################
### PARSING THROUGH A BOX SCORE FILE TO GET SCORES AND YARDS ###
sub ParseBoxScore
{
my %teams=SetUpNameHash();
my ($firstline, $n1, $n2, $s1, $s2, $index1, $index2, $abbr1,
$abbr2, $found, $ftf, $yards1, $yards2, $rushing_yards1,
$rushing_yards, $passing_yards1, $passing_yards2, $first_downs1,
$first_downs2, $return_yards1, $return_yards2);

## READ IN THE STAT FILE AND STORE IT IN AN ARRAY, CLOSE THE
## INPUT FILE
open(FIN,"<${DATAPATHNAME}Statistics$year/${filename}")
or die "Could not open ${DATAPATHNAME}Statistics$year/${filename}";
my @array=();


## TAKE OUT EMPTY LINES
```

```
while(<FIN>){
  chomp;
  if(/^$/){ ## SKIP EMPTY LINE
  }
  elsif(/Last Modifed: (\d+)\/(\d+)\/(\d+)/){
                                          ## AFTER THIS LINE THE FILE
                                          ## IS JUNK, HENCE END
    last;
  }
  else{
    push(@array, $_);  ## SAVE THE NON EMPTY LINE
  }
}
close(FIN);

## GET RID OF THE NEW LINE CHARACTER
chomp(@array);

## INITIALIZE FLAG FOR FINDING THE ABBREVIATION STATING THE ORDER
## IN WHICH STATS ARE LISTED
$found=0;

## INITIALIZE FLAG WHICH MARKS WHETHER team1 STATS LISTED FIRST
## OR SECOND
$ftf=0;

## INITIALIZE FLAG FOR FINDING THE TEAM NAMES AND SCORES
## (GAME MATCH INFO)
$linecount=0;

for($i=0;$i<$#array;$i++){ ## PARSE THROUGH EACH LINE OF A GAME FILE
 $x=$array[$i]; ##CURRENT LINE

 if($linecount<2){ ## FIND THE GAME MATCH INFORMATION
                   ## (I.E. TEAM NAMES AND GAME SCORES)
   if($x=~/San Francisco 49ers (\d+)/) ## SPECIAL CASE, TEAM NAME
                                       ## CONTAINS LETTERS AND NUM
   {
      if($linecount==0){
         $n1 = "San Francisco 49ers";
         $s1 = $1;
         $linecount=1;
      }
      else{
```

```
      $n2 = "San Francisco 49ers";
      $s2 = $1;
      $linecount=2;
      $abbr1= ${$team_names{$n1}}[1];
      $abbr2= ${$team_names{$n2}}[1];
      $index1= ${$team_names{$n1}}[0];
      $index2= ${$team_names{$n2}}[0];
    }
 }
 elsif($x=~/(.*?)\s+(\d+)(.*?)/){   ## FOUND LINE CONTAINING TEAM
                                    ## NAME AND SCORE
    if($linecount==0){
      $n1 = $1;
      $s1 = $2;
 if(exists($teams{$n1})){
           $linecount=1;
        }
    }
    else{
      $n2 = $1;
      $s2 = $2;
 if(exists($teams{$n1})){
           $linecount=2;
           $abbr1= ${$team_names{$n1}}[1];
           $abbr2= ${$team_names{$n2}}[1];
           $index1= ${$team_names{$n1}}[0];
           $index2= ${$team_names{$n2}}[0];
        }
    }
 }
} ## END if($linecount<2)
else{ ## NOW CAN SEARCH FOR STATS
   if($x=~/A--(\d+)/||$x=~/A--na/){  ## TEAMS STATS START AFTER
                                     ## THIS MARKER
 $linecount=3;
 next;
   }
   elsif($linecount==3){
       if($found==0){   ## HAVE NOT FOUND THE ABBREVIATION FOR
                        ## ORDER IN WHICH TOTAL YARDS ARE LISTED
          if($x=~/$abbr1/){ ## team1 HAS ITS STATS LISTED FIRST
    print "Found abbr1 first: $x\n";
         $found=1;
            $ftf=1;
```

```
                }
                elsif($x=~/$abbr2/){ ## team2 HAS ITS STATS LISTED FIRST
        print "Found abbr2 first: $x\n";
             $found=1;
                     $ftf=0;
        }
        else{
             next;
        }
           }
           elsif($found==1){ ## ALREADY FOUND THE ABBREVIATION, NOW
                             ## LOOKING FOR THE STATS
           if($x=~/TOTAL NET YARDS/){
              if($ftf==1){
                 $yards1=$array[$i+1];
                 $yards2=$array[$i+2];
              }
              elsif($ftf==0){
                 $yards1=$array[$i+2];
                 $yards2=$array[$i+1];
             }
                 }
elsif($x=~/NET YARDS RUSHING/){
              if($ftf==1){
                 $rushing_yards1=$array[$i+1];
                 $rushing_yards2=$array[$i+2];
              }
              elsif($ftf==0){
                 $rushing_yards1=$array[$i+2];
                 $rushing_yards2=$array[$i+1];
             }
                 }
elsif($x=~/NET YARDS PASSING/){
              if($ftf==1){
                 $passing_yards1=$array[$i+1];
                 $passing_yards2=$array[$i+2];
              }
              elsif($ftf==0){
                 $passing_yards1=$array[$i+2];
                 $passing_yards2=$array[$i+1];
             }
                 }
elsif($x=~/FIRST DOWNS/){;
              if($ftf==1){
```

```
                $first_downs1=$array[$i+1];
                $first_downs2=$array[$i+2];
             }
             elsif($ftf==0){
                $first_downs1=$array[$i+2];
                $first_downs2=$array[$i+1];
            }
                }
elsif($x=~/RETURN YARDAGE/){
             if($ftf==1){
                $return_yards1=$array[$i+1];
                $return_yards2=$array[$i+2];
             }
             elsif($ftf==0){
                $return_yards1=$array[$i+2];
                $return_yards2=$array[$i+1];
            }
                }
          }
      else{
          die "Never found abbreviation $abbr1 or $abbr2 in
               ${PATHNAME}${filename}";
          }
      }
      else{
      next;
      }
   } ## END if($linecount<2)->else
 } ## END OF FOR LOOP

if($linecount==0){
die "Never found team names or scores in ${PATHNAME}${filename}";
}

##
## PARSE THE DOCUMENT FOR THE TOTAL YARDS FOR EACH TEAM AND
## THE ORDER IN WHICH THEY ARE LISTED.

##
my @output=();

## team 1 CORRESPONDS TO FIRST INDEX EQUAL 0
$output[0][0]=$index1;
$output[0][1]=$s1;
```

```
$output[0][2]=$yards1;
$output[0][3]=$rushing_yards1;
$output[0][4]=$passing_yards1;
$output[0][5]=$first_downs1;
$output[0][6]=$return_yards1;

## team 2 CORRESPONDS TO FIRST INDEX EQUAL 1
$output[1][0]=$index2;
$output[1][1]=$s2;
$output[1][2]=$yards2;
$output[1][3]=$rushing_yards2;
$output[1][4]=$passing_yards2;
$output[1][5]=$first_downs2;
$output[1][6]=$return_yards2;

for($i=0;$i<2;$i++){
for($j=2;$j<7;$j++){
if($output[$i][$j]=~/\@ERROR/){
print "FOUND AN ERROR in the $filename \n";
$output[$i][$j]=10;
}
}
}

return @output;
}
#####################################################################
############### CREATE THE GAME DATA MATRICES FOR MATLAB #######
sub CreateGameData()
{
   require "${DATAPATHNAME}NFL_Roster/roster$year.pl";
   my ($function_name,$function_header,$length_g,$i,$j,$w,$k);
   my $week=0;
   my $length_weeks =0;
   $length_g=0;

   ## CREATE .m FILE CONTAINING MATRICES WITH CUMULATIVE SCORES
   $function_name="preGameScores";
   $function_header= "%%\n%% Game scores matrix $year
   \n%% entry (i,j) - score of team team i against team j.
   If both (i,j) and (j,i) \n%% entries are zero them teams
   (most likely) did not play each other this week.\n%%\n";
   print "$function_name\n";
   open(FOUTS,">${PATHNAME}${function_name}.m")
```

```
    or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                        \@Scores_List,"0");
close(FOUTS);

## CREATE .m FILE CONTAINING MATRICES WITH RUSHING YARDS
$function_name="preRushingYards";
$function_header= "%%\n%% Game scores matrix $year
\n%% entry (i,j) - score of team team i against team j.
If both (i,j) and (j,i) \n%% entries are zero them teams
(most likely) did not play each other this week.\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                     \@Rushing_Yards_List,"0");
close(FOUTS);

## CREATE .m FILE CONTAINING MATRICES WITH TEAMS-TEAMS WINS
$function_name="preTeamAgainstTeamWins";
$function_header= "%%\n%% (i,j) = number of wins of team i
against team j by playing $week weeks during $year \n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                       \@Team_TeamWins,"0");

## CREATE .m FILE CONTAINING MATRICES WITH CUMULATIVE
## YARDAGE DIFFERENCES
$function_name="preGameYards";
$function_header= "%%\n%% Total yards $year.\n%%
Y(i,j)=total game yards of team i against team j\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                        \@Yards_Diff,"0");
close(FOUTS);

## CREATE .m FILE CONTAINING MATRICES WITH CUMULATIVE
## RUSHING YARDAGE DIFFERENCES
$function_name="preGameRushingYards";
$function_header= "%%\n%% Rushing yards $year.\n%%
```

```
R(i,j)=rushing yards of team i against team j\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m") or die
"Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                \@Rushing_Yards_Diff,"0");
close(FOUTS);

## CREATE .m FILE CONTAINING MATRICES WITH CUMULATIVE
## PASSING YARDAGE DIFFERENCES
$function_name="preGamePassingYards";
$function_header= "%%\n%% Passing yards $year.\n%%
P(i,j)=passing yards of team i against team j\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                \@Passing_Yards_Diff,"0");
close(FOUTS);

## CREATE .m FILE CONTAINING MATRICES WITH CUMULATIVE
## FIRST DOWN DIFFERENCES
$function_name="preGameFirstDowns";
$function_header= "%%\n%% First Downs $year.\n%%
F(i,j)=number of first downs of team i against team j\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                   \@First_Downs_Diff,"0");
close(FOUTS);

## CREATE .m FILE CONTAINING MATRICES WITH CUMULATIVE
## RETURN YARDAGE DIFFERENCES
$function_name="preGameReturnYards";
$function_header= "%%\n%% Returned Yards $year.\n%%
R(i,j)=returned yards of team i against team j\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                   \@Return_Yards_Diff,"0");
close(FOUTS);
```

```
## CREATE .m FILE CONTAINING MATRICES WITH CUMULATIVE
## SCORES DIFFERENCES
$function_name="preGeM_Score_Diff";
$function_header= "%%\n%% Score Difference Matrices for NFL
                                          season $year \n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                         \@Score_Diff,"0");
close(FOUTS);

## CREATE .m FILE CONTAINING LAST WEEK WORTH OF SCORE
## DIFFERENCE MATRIX
$function_name="preLastWeekScoreDiff";
$function_header= "%%\n%% Week $weekcount score differences
                                               for NFL\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                      \@Last_Score_Diff,"0");
close(FOUTS);


## CREATE .m FILE CONTAINING COLLEY MATRIX (WITHOUT DIAGONAL)
$function_name="precolley_matrix";
$function_header= "%%\n%% Colley Matrix for NFL\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                        \@Colley_Matrix,"N");
print FOUTS "\n\nN=[";
   for($i=1;$i<=$teamcount;$i++){
      if($i>1){
         print FOUTS ";\n";
      }
    for($j=1;$j<=3;$j++){
       if($j<3){
            print FOUTS "$Colley_WinLoss[$i][$j] ";
       }
       else{
```

```
            print FOUTS "$Colley_WinLoss[$i][$j]";
        }
    }
 }
print FOUTS "];\n\n";
close(FOUTS);


## CREATE .m FILE CONTAINING MASSEY MATRIX (WITHOUT DIAGONAL)
$function_name="preMassey_matrix";
$function_header= "%%\n%% Massey Matrix for NFL\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                        \@Colley_Matrix,"SD");
print FOUTS "\n\nSD=[";
   for($i=1;$i<=$teamcount;$i++){
      if($i>1){
         print FOUTS ";\n";
      }
      print FOUTS "$Massey_SD[$i]";
   }
print FOUTS "];\n\n";
close(FOUTS);

## CREATE .m FILE CONTAINING MATRIX WITH CUMULATIVE STATS
print FOUTST "%%\n";
print FOUTST "%% STATS $year: total game yards, rushing yards,
                 passing yards, first downs, return yards\n";
print FOUTST "%%\n";

print FOUTST "function [Stats]=preGameStats()\n";
print FOUTST "Stats=[";
for($i=1;$i<=$teamcount;$i++){
   if($i>1){
       print FOUTST ";\n";
   }
 for($k=1;$k<=$stats_num;$k++){
    if($k<$stats_num){
       print FOUTST "$Stats_List[$i][$k] ";
    }
    else{
        print FOUTST "$Stats_List[$i][$k]";
```

```
        }
       }
     }
     print FOUTST "];";

   ## CREATE .M FILE CONTAINING MATRICES WITH WEEKLY
   ## MATCH UPS AND SCORES
   print FOUTM "%%\n";
   print FOUTM "%% Weekly match ups, $year: team 1 index,
                  team 1 score, team 2 index, team 2 score\n";
   print FOUTM "%%\n";


   print FOUTSch "%%\n";
   print FOUTSch "%% Weekly match up schedule, $year: team 1 index,
                     team 2 index\n";
   print FOUTSch "%%\n";

   print FOUTM "function [M]=preMatchUps()\n";
   print FOUTSch "function [Sch]=NFL${year}schedule()\n";
   print FOUTGAMES "#!/usr/bin/perl\n";
   print FOUTGAMES "\@gamecount=(";
   $week=1;
   $length_weeks = scalar(@Matchup_list);
   while($week<$length_weeks)
   {
        print FOUTM "M{$week}=[";
        print FOUTSch "Sch{$week}=[";
$g=0;

$length_g = scalar(@{$Matchup_list[$week]});
if($week<$length_weeks-1){
            print FOUTGAMES "$length_g,";
        }
else{
    print FOUTGAMES "$length_g);";
}

while($g<$length_g)
        {
   if($g<$length_g-1)
   {
           print FOUTM "$Matchup_list[$week][$g][0],
                        $Matchup_list[$week][$g][1],
```

```
                        $Matchup_list[$week][$g][2],
                        $Matchup_list[$week][$g][3];\n";
            print FOUTSch "$Matchup_list[$week][$g][0],
                          $Matchup_list[$week][$g][2];\n"
   }
   else
            {
            print FOUTM "$Matchup_list[$week][$g][0],
                        $Matchup_list[$week][$g][1],
                        $Matchup_list[$week][$g][2],
                        $Matchup_list[$week][$g][3]];\n\n";
            print FOUTSch "$Matchup_list[$week][$g][0],
                          $Matchup_list[$week][$g][2]];\n\n";
   }
        $g++;
}
$week++;
    }


   ## CREATE .txt FILE CONTAINING MATRICES WITH WEEKLY MATCH UPS WITH
   ## RESULTS FOR GamePredictions PERL SCRIPT
   $week=1;
   my $temp=0;
   while($week<$length_weeks){
$g=0;
$length_g = scalar(@{$Matchup_list[$week]});

print FOUTPL "Begin Week $week\n";
while($g<$length_g){
        print FOUTPL "$Matchup_list[$week][$g][0],
                      $Matchup_list[$week][$g][1],
                      $Matchup_list[$week][$g][2],
                      $Matchup_list[$week][$g][3]\n";

        $g++;
}
print FOUTPL "End Week $week\n";
$week++;
    }

}
#################################################################
############ PRINT TEAM-BY-TEAM MATRIX OF GIVEN DATA ############
```

```
sub TeamByTeamMatrix()
{
   my $function_name = shift @_;
   my $function_header = shift @_;
   my $DataRef = shift @_;
   my @Data = @{$DataRef};
   my $second_matrix = shift @_;
   my $matrix_name=substr("$function_name",3,1);

   print "function name=$function_name, matrix name=$matrix_name\n";

   print FOUTS "$function_header";
   if($second_matrix eq "0"){
     print FOUTS "function [$matrix_name]=$function_name()\n";
   }
   else{
     print FOUTS "function [$matrix_name,$second_matrix]=
                                                $function_name()\n";
   }


   print FOUTS "${matrix_name}=[";
         for($i=1;$i<=$teamcount;$i++){
            if($i>1){
            print FOUTS ";\n";
            }
            for($j=1;$j<=$teamcount;$j++)
               {
               if($j<$teamcount){
                  print FOUTS "$Data[$i][$j] ";
               }
               else{
                  print FOUTS "$Data[$i][$j]";
               }
            }
         }
   print FOUTS "];\n\n";
}

#####################################################################
########### UPDATE THE GIVEN DIFFERENCE MATRIX ##################
sub Data_To_DiffMat()
{
   my $team1stat = shift @_;
```

```
   my $team2stat = shift @_;
   my $team1index = shift @_;
   my $team2index = shift @_;
   my $DataRef = shift @_;
   my @Data = @{$DataRef};


   if($team1stat>$team2stat){ ## TEAM 1 SCORED MORE POINTS THEN TEAM 2
      ## TEAM 2 LOST TO TEAM 1, HENCE TEAM 2 "votes" for TEAM 1,
      ##ADD SCORE DIFFERENCE TO (team 2, team1) ENTRY
      $Data[$team2index][$team1index]=
      $Data[$team2index][$team1index]+$team1stat-$team2stat;
   }
   elsif($team1stat<$team2stat){ ## TEAM 2 SCORED MORE POINTS THEN TEAM 1
      ## TEAM 1 LOST TO TEAM 2, HENCE TEAM 1 "votes" for TEAM 2,
      ## ADD SCORE DIFFERENCE TO (team 1, team2) ENTRY
      $Data[$team1index][$team2index]=
      $Data[$team1index][$team2index]+$team2stat-$team1stat;
      }
  else{
      print "WARNING: teams tied, can not do week $weekcount, teams -
            $team1index and $team2index  . \n";
  }


}
#####################################################################
######################### EXECUTE ###################################

## SET UP PARAMETERS
$year=2008;
$lastweek=5;

$DATAPATHNAME="c:/DataProcessing/NFL/";
$FORMATEDPATHNAME="c:/DataProcessing/FormatedDataFiles/";
$PATHNAME="c:/MATLAB7/Research/NFL/RankingPrograms/NFL_Data/$year/";
require "${DATAPATHNAME}NFL_Roster/roster$year.pl";

## SET UP TEAM NAMES HASH
my @temp=@names;
$teamcount=$#temp+1; ## NUMBER OF TEAMS IN THE ROSTER
                     ## (ARRAY COUNTS FROM 0)

%team_names=SetUpNameHash();
```

```
## READ IN THE FILE CONTAINING THE LIST OF FILE NAMES
## CONTAINING WEEKLY DATA
open(FIN,"<${DATAPATHNAME}Statistics$year/prefilepaths.txt")
or die
"Could not open ${DATAPATHNAME}Statistics$year/prefilepaths.txt";
@fp=<FIN>;
close(FIN);
chomp(@fp);

@Massey_SD=();
@Colley_Matrix=();
@Colley_WinLoss=();

@Team_TeamWins=();
@Matchup_list=();

@Stats_List=();
@Scores_List=();
@Rushing_Yards_List=();

@Score_Diff=();
@Last_Score_Diff=();
@Yards_Diff=();
@Rushing_Yards_Diff=();
@Passing_Yards_Diff=();
@First_Downs_Diff=();
@Return_Yards_Diff=();

$weekcount=0;
$stats_num=5; ## NUMBER OF STATS I CURRENTLY ACQUIRE,
              ## NOTE: THIS MIGHT CHANGE!!
my $count=0;
my $game=0;
my ($team1index,$team2index,$score1,$score2);

for($i=1;$i<=$teamcount;$i++){
    $Massey_SD[$i]=0;

    for($j=1;$j<=3;$j++){
      $Colley_WinLoss[$i][$j]=0;
    }

    for($j=1;$j<=$teamcount;$j++){
```

```
      $Colley_Matrix[$i][$j]=0;
      $Last_Score_Diff[$i][$j]=0;

      $Team_TeamWins[$i][$j]=0;

      $Scores_List[$i][$j]=0;
      $Rushing_Yards_List[$i][$j]=0;

      $Score_Diff[$i][$j]=0;
      $Yards_Diff[$i][$j]=0;
      $Rushing_Yards_Diff[$i][$j]=0;
      $Passing_Yards_Diff[$i][$j]=0;
      $First_Downs_Diff[$i][$j]=0;
      $Return_Yards_Diff[$i][$j]=0;
    }
    for($k=1; $k<=$stats_num; $k++){
      $Stats_List[$i][$k]=0;
    }
}

foreach $line (@fp) ## LOOPING OVER ALL THE BOX SCORE FILES
                    ## FOR THE GIVEN YEAR
{
   if($line=~/week(\d\d)/) ## FOUND BEGINNING OF A WEEK MARKER
   {
   if($1>$lastweek){ ## PARSE FILES UP TO AND INCLUDING
                             ## THE SPECIFIED WEEK
   last;
   }
       print "Found beginning of week$1\n";
       $weekcount++;
       $count=0;
       $game=1;

   }
   elsif($line=~/end/)
   {
      #print "Found end of week $weekcount\n";
   }
   else ## Parse the data file for the given game
   {
      $filename=$line;

      ## PARSE THE CURRENT GAME FILE AND OBTAIN THE INDICES OF THE
```

```
## TEAMS THEIR SCORES AND STATS
@out=&ParseBoxScore();
$team1index=$out[0][0];
$team2index=$out[1][0];
$score1=$out[0][1];
$score2=$out[1][1];


## SET UP THE CURRENT GAME MATCH UP FOR THE WEEK
$Matchup_list[$weekcount][$count][0]= $team1index;
$Matchup_list[$weekcount][$count][1]= $score1;
$Matchup_list[$weekcount][$count][2]= $team2index;
$Matchup_list[$weekcount][$count][3]= $score2;

## ACCUMULATE TOTAL AMOUNT OF SCORES AND RUSHING
## YARDS FOR EACH TEAM
$Scores_List[$team1index][$team2index] =
     $Scores_List[$team1index][$team2index]+$score1;
$Scores_List[$team2index][$team1index] =
     $Scores_List[$team2index][$team1index]+$score2;

$Rushing_Yards_List[$team1index][$team2index]=
     $Rushing_Yards_List[$team1index][$team2index]+$out[0][3];
$Rushing_Yards_List[$team2index][$team1index]=
     $Rushing_Yards_List[$team2index][$team1index]+$out[1][3];

for($k=1; $k<=$stats_num;$k++){
    $Stats_List[$team1index][$k] =
            $Stats_List[$team1index][$k]+ $out[1][$k+1];
    $Stats_List[$team2index][$k] =
            $Stats_List[$team2index][$k]+ $out[0][$k+1];
}
$count++;

$Massey_SD[$team1index]=$Massey_SD[$team1index]+$score1-$score2;
$Massey_SD[$team2index]=$Massey_SD[$team2index]+$score2-$score1;

$Colley_Matrix[$team1index][$team2index]=
       $Colley_Matrix[$team1index][$team2index]-1;
$Colley_Matrix[$team2index][$team1index]=
       $Colley_Matrix[$team2index][$team1index]-1;

## SET UP MASSEY MATRIX, MASSEY GAME MATRIX ROW
## FOR THE CURRENT GAME OF THE WEEK
```

```
    if($score1>$score2){ ## Team 1 scored more points then Team 2
        ## INCREASE WINS OF TEAM 1
$Colley_WinLoss[$team1index][1]=
             $Colley_WinLoss[$team1index][1]+1;

         ## INCREASE TOTAL OF TEAM 1
         $Colley_WinLoss[$team1index][3]=
             $Colley_WinLoss[$team1index][3]+1;

        ## INCREASE LOSSES OF TEAM 2
$Colley_WinLoss[$team2index][2]=
             $Colley_WinLoss[$team2index][2]+1;
        ## INCREASE TOTAL OF TEAM 2
$Colley_WinLoss[$team2index][3]=
             $Colley_WinLoss[$team2index][3]+1;

        ## TEAM 1 BEAT TEAM 2, INCREASE NUMBER OF WINS
        ## BY TEAM 1 AGAINST TEAM 2 BY 1
        $Team_TeamWins[$team1index][$team2index]=
             $Team_TeamWins[$team1index][$team2index]+1;

## TEAM 2 LOST TO TEAM 1, HENCE TEAM 2 "votes" for
        ## TEAM 1, ADD SCORE DIFFERENCE TO (team 2, team1) ENTRY
if($weekcount==$lastweek){
     $Last_Score_Diff[$team2index][$team1index]=
         $Last_Score_Diff[$team2index][$team1index]+$score1-$score2;
}
    }
    elsif($score1<$score2){ ## Team 2 scored more points then Team 1
        ## INCREASE LOSSES OF TEAM 1
$Colley_WinLoss[$team1index][2]=$Colley_WinLoss[$team1index][2]+1;
        ## INCREASE TOTAL OF TEAM 1
$Colley_WinLoss[$team1index][3]=$Colley_WinLoss[$team1index][3]+1;

        ## INCREASE WINS OF TEAM 2
$Colley_WinLoss[$team2index][1]=$Colley_WinLoss[$team2index][1]+1;
        ## INCREASE TOTAL OF TEAM 2
$Colley_WinLoss[$team2index][3]=$Colley_WinLoss[$team2index][3]+1;

## TEAM 2 BEAT TEAM 1, INCREASE NUMBER OF WINS BY
        ## TEAM 2 AGAINST TEAM 1 BY 1
        $Team_TeamWins[$team2index][$team1index]=
             $Team_TeamWins[$team2index][$team1index]+1;
```

```
  ## TEAM 1 LOST TO TEAM 2, HENCE TEAM 1 "votes" for
          ## TEAM 2, ADD SCORE DIFFERENCE TO (team 1, team2) ENTRY
  if($weekcount==$lastweek){
      $Last_Score_Diff[$team1index][$team2index]=
        $Last_Score_Diff[$team1index][$team2index]+$score2-$score1;
  }
      }
      else{
          print "WARNING: teams tied, can not do week $weekcount,
                 game $game for Massey matrix. \n";
      }
      ## UPDATE ALL SCORE DIFFERENCE MATRICES
      &Data_To_DiffMat($score1,$score2,$team1index,$team2index,
                                                      \@Score_Diff);
      &Data_To_DiffMat($out[0][2],$out[1][2],$team1index,$team2index,
                                                      \@Yards_Diff);
      &Data_To_DiffMat($out[0][3],$out[1][3],$team1index,$team2index,
                                                  \@Rushing_Yards_Diff);
      &Data_To_DiffMat($out[0][4],$out[1][4],$team1index,$team2index,
                                                  \@Passing_Yards_Diff);
      &Data_To_DiffMat($out[0][5],$out[1][5],$team1index,$team2index
                                                   ,\@First_Downs_Diff);
      &Data_To_DiffMat($out[0][6],$out[1][6],$team1index,$team2index,
                                                   \@Return_Yards_Diff);
      $game++;

  }
  #print "DONE\n";
  #print "now weekcount=$weekcount\n";
}

## OPEN THE DATA FILES FOR OUTPUT

print "$weekcount\n";

open(FOUTST,">${PATHNAME}preGameStats.m")
or die "Could not open ${PATHNAME}preGameStats.m";

open(FOUTM,">${PATHNAME}preMatchUps.m")
or die "Could not open ${PATHNAME}preMatchUps.m";

open(FOUTSch,">${PATHNAME}preNFL${year}schedule.m")
or die "Could not open ${PATHNAME}preNFL${year}schedule.m";
```

```
open(FOUTGAMES,">${DATAPATHNAME}Statistics$year
                                    /preGameCount$year.pl")
or die "Could not open ${DATAPATHNAME}Statistics$year
                                    /preGameCount$year.pl";

open(FOUTPL,"> ${DATAPATHNAME}GamePredictions/${year}/
                                   prematches_${year}.txt")
or die "Could not open ${DATAPATHNAME}preGamePredictions/
                             ${year}/prematches_${year}.txt";


CreateGameData();

## CLOSE ALL OUTPUT FILES
close(FOUTST);
close(FOUTM);
close(FOUTSch);
close(FOUTGAMES);
close(FOUTPL);
```

B.4 NFL Regular Season and Postseason Data Parsing

```
#!/usr/bin/perl
#####################################################################
###
### Parse NFL data files.
###
#####################################################################

#####################################################################
#########  SETTING UP HASH TABLES FOR TEAM INDEX LOOK UP #######
sub SetUpNameHash()
{
require "${DATAPATHNAME}NFL_Roster/roster$year.pl";

my %team_names;
my $j=0;
my @entry=();

   for($i=0;$i<=$teamcount-1;$i++){
       $j=$i+1;
       $entry[0]=$j;
       $entry[1]=$abbr[$i];
       $team_names{$names[$i]} = [$j, $abbr[$i],$names[$i]];
    }
return %team_names;
}

#####################################################################
## PARSING THROUGH A BOX SCORE FILE TO GET SCORES, YARDS, ETC ##
sub ParseBoxScore
{
## DECLARE LOCAL VARIABLES
my %teams=SetUpNameHash();
my ($firstline, $n1, $n2, $s1, $s2, $index1, $index2, $abbr1, $abbr2,
$found, $ftf, $yards1, $yards2, $rushing_yards1, $rushing_yards,
$passing_yards1, $passing_yards2, $first_downs1, $first_downs2,
$return_yards1, $return_yards2);

## READ IN THE STAT FILE AND STORE IT IN AN ARRAY,
## CLOSE THE INPUT FILE
open(FIN,"<${DATAPATHNAME}Statistics$year/${filename}")
or die "Could not open ${DATAPATHNAME}Statistics$year/${filename}";
my @array=();
```

```
## TAKE OUT EMPTY LINES
while(<FIN>){
  chomp;
  if(/^$/){ ## SKIP EMPTY LINE
  }
  elsif(/Last Modifed: (\d+)\/(\d+)\/(\d+)/){ ## AFTER THIS LINE THE
                                              ## FILE IS JUNK, DONE
    last;
  }
  else{
    push(@array, $_);  ## SAVE THE LINE
  }
}
close(FIN);

## GET RID OF THE NEW LINE CHARACTER
chomp(@array);

## INITIALIZE FLAG FOR FINDING THE ABBREVIATION STATING THE ORDER IN
## WHICH STATS ARE LISTED
$found=0;

## INITIALIZE FLAG WHICH MARKS WHETHER team1 STATS LISTED FIRST OR
## SECOND
$ftf=0;

## INITIALIZE FLAG FOR FINDING THE TEAM NAMES AND SCORES
## (GAME MATCH INFO)
$linecount=0;

for($i=0;$i<$#array;$i++){ ## PARSE THROUGH EACH LINE OF A GAME FILE
 $x=$array[$i]; ## CURRENT LINE
 #print "Current line: $x\n";
 #print "linecount=$linecount\n";

 if($linecount<2){ ## FIND THE GAME MATCH INFORMATION
                   ## (I.E. TEAM NAMES AND GAME SCORES)
   if($x=~/San Francisco 49ers (\d+)/) ## SPECIAL CASE, TEAM NAME
                                       ## CONTAINS LETTERS
                                       ## AND NUMERALS
   {
      if($linecount==0){
         $n1 = "San Francisco 49ers";
```

```
      $s1 = $1;
      $linecount=1;
   }
   else{
      $n2 = "San Francisco 49ers";
      $s2 = $1;
      $linecount=2;
      $abbr1= ${$team_names{$n1}}[1];
      $abbr2= ${$team_names{$n2}}[1];
      $index1= ${$team_names{$n1}}[0];
      $index2= ${$team_names{$n2}}[0];
    }
 }
 elsif($x=~/(.*?)\s+(\d+)(.*?)/){ ## FOUND LINE CONTAINING
                                   ## TEAM NAME AND SCORE
    if($linecount==0){
      $n1 = $1;
      $s1 = $2;
 if(exists($teams{$n1})){
            $linecount=1;
         }
    }
    else{
      $n2 = $1;
      $s2 = $2;
 if(exists($teams{$n1})){
            $linecount=2;
            $abbr1= ${$team_names{$n1}}[1];
            $abbr2= ${$team_names{$n2}}[1];
            $index1= ${$team_names{$n1}}[0];
            $index2= ${$team_names{$n2}}[0];
         }
    }
 }
} ## end if($linecount<2)
else{ ## NOW CAN SEARCH FOR STATS
    if($x=~/A--(\d+)/||$x=~/A--na/){ ## TEAMS STATS START
                                     ## AFTER THIS MARKER
 $linecount=3;
 next;
    }
    elsif($linecount==3){
        if($found==0){ ## HAVE NOT FOUND THE ABBREVIATION IN
                       ## ORDER THAT TOTAL YARDS ARE LISTED
```

```
             if($x=~/$abbr1/){ ## team1 HAS ITS STATS LISTED FIRST
            $found=1;
               $ftf=1;
               }
               elsif($x=~/$abbr2/){ ## team2 HAS ITS STATS
                                    ## LISTED FIRST
            $found=1;
                    $ftf=0;
       }
       else{
            next;
       }
          }
          elsif($found==1){ ## ALREADY FOUND THE ABBREVIATION,
                            ## NOW LOOKING FOR THE STATS
          if($x=~/TOTAL NET YARDS/){
             if($ftf==1){
                $yards1=$array[$i+1];
                $yards2=$array[$i+2];
             }
             elsif($ftf==0){
                $yards1=$array[$i+2];
                $yards2=$array[$i+1];
            }
                }
elsif($x=~/NET YARDS RUSHING/){
             if($ftf==1){
                $rushing_yards1=$array[$i+1];
                $rushing_yards2=$array[$i+2];
             }
             elsif($ftf==0){
                $rushing_yards1=$array[$i+2];
                $rushing_yards2=$array[$i+1];
            }
                }
elsif($x=~/NET YARDS PASSING/){
             if($ftf==1){
                $passing_yards1=$array[$i+1];
                $passing_yards2=$array[$i+2];
             }
             elsif($ftf==0){
                $passing_yards1=$array[$i+2];
                $passing_yards2=$array[$i+1];
            }
```

```
                }
elsif($x=~/FIRST DOWNS/){
             if($ftf==1){
                $first_downs1=$array[$i+1];
                $first_downs2=$array[$i+2];
             }
             elsif($ftf==0){
                $first_downs1=$array[$i+2];
                $first_downs2=$array[$i+1];
            }
                }
elsif($x=~/RETURN YARDAGE/){
             if($ftf==1){
                $return_yards1=$array[$i+1];
                $return_yards2=$array[$i+2];
             }
             elsif($ftf==0){
                $return_yards1=$array[$i+2];
                $return_yards2=$array[$i+1];
            }
                }
          }
      else{
          die "Never found abbreviation $abbr1 or $abbr2
                                in ${PATHNAME}${filename}";
          }
      }
      else{
      next;
      }
   } ## END if($linecount<2)->else
 } ## END OF FOR LOOP

if($linecount==0){
die "Never found team names or scores in ${PATHNAME}${filename}";
}



## PARSE THE DOCUMENT FOR THE TOTAL YARDS FOR EACH TEAM AND THE
## ORDER IN WHICH THEY ARE LISTED.

my @output=();
```

```
## team 1 CORRESPONDS TO FIRST INDEX EQUAL  0
$output[0][0]=$index1;
$output[0][1]=$s1;
$output[0][2]=$yards1;
$output[0][3]=$rushing_yards1;
$output[0][4]=$passing_yards1;
$output[0][5]=$first_downs1;
$output[0][6]=$return_yards1;

## team 2 CORRESPONDS TO FIRST INDEX EQUAL  1
$output[1][0]=$index2;
$output[1][1]=$s2;
$output[1][2]=$yards2;
$output[1][3]=$rushing_yards2;
$output[1][4]=$passing_yards2;
$output[1][5]=$first_downs2;
$output[1][6]=$return_yards2;

for($i=0;$i<2;$i++){
for($j=2;$j<7;$j++){
if($output[$i][$j]=~/\@ERROR/){
print "FOUND AN ERROR in the $filename \n";
$output[$i][$j]=10;
}
}
}

return @output;
}
######################################################################
############ CREATE THE GAME DATA MATRICES FOR MATLAB ############
sub CreateGameData()
{
   require "${DATAPATHNAME}NFL_Roster/roster$year.pl";
   my ($function_name,$function_header,$length_g,$i,$j,$w,$k);
   my $week=0;
   my $length_weeks =0;
   $length_g=0;

   ## CREATE .M FILE CONTAINING MATRICES WITH CUMULATIVE SCORES
   $function_name="GameScores";
   $function_header= "%%\n%% Game scores matrices $year
      \n%% entry (i,j) - score of team i against team j.
      If both (i,j) and (j,i) \n%% entries are zero them teams
```

```
   (most likely) did not play each other this week.\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
     or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                    \@Scores_List,"0");
close(FOUTS);

## CREATE .M FILE CONTAINING MATRICES WITH CUMULATIVE
##RUSHING YARDS
$function_name="RushingYards";
$function_header= "%%\n%% Game scores matrices $year
\n%% entry (i,j) - amount of rushing yards by team i
 against team j. If both (i,j) and (j,i) \n%% entries are zero
  them teams (most likely) did not play each other this week.
  \n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
        or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                     \@Rushing_Yards_List,"0");
close(FOUTS);

## CREATE .M FILE CONTAINING MATRICES WITH TEAM-BY-TEAM WINS
$function_name="TeamAgainstTeamWins";
$function_header= "%%\n%% (i,j) = number of wins of team i
                  against team j by playing $week weeks during
                  $year \n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
         or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                           \@Team_TeamWins,"0");
close(FOUTS);

## CREATE .M FILE CONTAINING MATRICES WITH CUMULATIVE YARDAGE
## DIFFERENCES
$function_name="GameYards";
$function_header= "%%\n%% Total yards $year.\n%%
 Y(i,j)=total game yards difference team j out gained i\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
         or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
```

```
                                                \@Yards_Diff,"0");
close(FOUTS);

## CREATE .M FILE CONTAINING MATRICES WITH CUMULATIVE RUSHING
## YARDAGE DIFFERENCES
$function_name="GameRushingYards";
$function_header= "%%\n%% Rushing yards $year.\n%%
   R(i,j)=rushing yards difference team j out gained i\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
        or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                     \@Rushing_Yards_Diff,"0");
close(FOUTS);

## CREATE .M FILE CONTAINING MATRICES WITH CUMULATIVE PASSING
## YARDAGE DIFFERENCES
$function_name="GamePassingYards";
$function_header= "%%\n%% Passing yards $year.\n%%
   P(i,j)=passing yards difference team j out gained i\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
        or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                     \@Passing_Yards_Diff,"0");
close(FOUTS);

## CREATE .M FILE CONTAINING MATRICES WITH CUMULATIVE
## FIRST DOWNS DIFFERENCES
$function_name="GameFirstDowns";
$function_header= "%%\n%% First Downs $year.\n%%
F(i,j)=number of first downs difference team j out gained i\n%%\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
          or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                       \@First_Downs_Diff,"0");
close(FOUTS);

## CREATE .M FILE CONTAINING MATRICES WITH CUMULATIVE RETURN
## YARDAGE DIFFERENCES
$function_name="GameReturnYards";
$function_header= "%%\n%% Returned Yards $year.\n%%
R(i,j)=returned yards difference team j out gained i\n%%\n";
```

```
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
      or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                    \@Return_Yards_Diff,"0");
close(FOUTS);


## CREATE .M FILE CONTAINING MATRICES WITH CUMULATIVE
## SCORE DIFFERENCES
$function_name="GeM_Score_Diff";
$function_header= "%%\n%% Weekly Score Difference
                   Matrices for NFL season $year \n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
      or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                         \@Score_Diff,"0");
close(FOUTS);

## CREATE .M FILE CONTAINING LAST WEEK WORTH OF SCORE
## DIFFERENCE MATRIX
$function_name="LastWeekScoreDiff";
$function_header= "%%\n%% Week $weekcount score
                                     differences for NFL\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
      or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                     \@Last_Score_Diff,"0");
close(FOUTS);


## CREATE .M FILE CONTAINING COLLEY MATRIX (WITHOUT DIAGONAL)
$function_name="colley_matrix";
$function_header= "%%\n%% Colley Matrix for NFL\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
       or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                       \@Colley_Matrix,"N");

## PRINT N VECTOR CONTAINING EACH TEAMS WINS, LOSSES,
## AND TOTAL NUMBER OF GAMES PLAYED
```

```
print FOUTS "\n\nN=[";
   for($i=1;$i<=$teamcount;$i++){
      if($i>1){
         print FOUTS ";\n";
      }
    for($j=1;$j<=3;$j++){
       if($j<3){
            print FOUTS "$Colley_WinLoss[$i][$j] ";
       }
       else{
            print FOUTS "$Colley_WinLoss[$i][$j]";
       }
    }
 }
print FOUTS "];\n\n";
close(FOUTS);


## CREATE .M FILE CONTAINING MASSEY MATRIX (WITHOUT DIAGONAL)
$function_name="Massey_matrix";
$function_header= "%%\n%% Massey Matrix for NFL\n";
print "$function_name\n";
open(FOUTS,">${PATHNAME}${function_name}.m")
      or die "Could not open ${PATHNAME}${function_name}.m";
TeamByTeamMatrix($function_name,$function_header,
                                        \@Colley_Matrix,"SD");
print FOUTS "\n\nSD=[";
   for($i=1;$i<=$teamcount;$i++){
      if($i>1){
         print FOUTS ";\n";
      }
      print FOUTS "$Massey_SD[$i]";
   }
print FOUTS "];\n\n";
close(FOUTS);

## CREATE .M FILE CONTAINING MATRICES WITH WEEKLY STATS
print FOUTST "%%\n";
print FOUTST "%% STATS $year: total game yards, rushing yards,
                  passing yards, first downs, return yards\n";
print FOUTST "%%\n";

print FOUTST "function [Stats]=GameStats()\n";
print FOUTST "Stats=[";
```

```
   for($i=1;$i<=$teamcount;$i++){
      if($i>1){
          print FOUTST ";\n";
      }
    for($k=1;$k<=$stats_num;$k++){
       if($k<$stats_num){
          print FOUTST "$Stats_List[$i][$k] ";
       }
       else{
           print FOUTST "$Stats_List[$i][$k]";
       }
      }
    }
    print FOUTST "];";

   ## CREATE .M FILE CONTAINING MATRICES WITH WEEKLY
   ## MATCH UPS AND SCORES
   print FOUTM "%%\n";
   print FOUTM "%% Weekly match ups, $year: team 1 index,
                    team 1 score, team 2 index, team 2 score\n";
   print FOUTM "%%\n";


   print FOUTSch "%%\n";
   print FOUTSch "%% Weekly match up schedule,
                  $year: team 1 index, team 2 index\n";
   print FOUTSch "%%\n";

   print FOUTM "function [M]=MatchUps()\n";
   print FOUTSch "function [Sch]=NFL${year}schedule()\n";
   print FOUTGAMES "#!/usr/bin/perl\n";
   print FOUTGAMES "\@gamecount=(";
   $week=1;
   $length_weeks = scalar(@Matchup_list);
   while($week<$length_weeks)
   {
        print FOUTM "M{$week}=[";
        print FOUTSch "Sch{$week}=[";
$g=0;

$length_g = scalar(@{$Matchup_list[$week]});
if($week<$length_weeks-1){
            print FOUTGAMES "$length_g,";
        }
```

```
else{
    print FOUTGAMES "$length_g);";
}

while($g<$length_g)
        {
   if($g<$length_g-1)
   {
           print FOUTM "$Matchup_list[$week][$g][0],
                        $Matchup_list[$week][$g][1],
                        $Matchup_list[$week][$g][2],
                        $Matchup_list[$week][$g][3];\n";
           print FOUTSch "$Matchup_list[$week][$g][0],
                          $Matchup_list[$week][$g][2];\n"
   }
   else
           {
           print FOUTM "$Matchup_list[$week][$g][0],
                        $Matchup_list[$week][$g][1],
                        $Matchup_list[$week][$g][2],
                        $Matchup_list[$week][$g][3]];\n\n";
           print FOUTSch "$Matchup_list[$week][$g][0],
                          $Matchup_list[$week][$g][2]];\n\n";
   }
        $g++;
}
$week++;
    }


   ## CREATE .TXT FILE CONTAINING MATRICES WITH WEEKLY MATCH
   ## UPS WITH RESULTS FOR GAMEPREDICTIONS PERL PROGRAM
   $week=1;
   my $temp=0;
   while($week<$length_weeks){
$g=0;
$length_g = scalar(@{$Matchup_list[$week]});

print FOUTPL "Begin Week $week\n";
while($g<$length_g){
        print FOUTPL "$Matchup_list[$week][$g][0],
                      $Matchup_list[$week][$g][1],
                      $Matchup_list[$week][$g][2],
                      $Matchup_list[$week][$g][3]\n";
```

```
        $g++;
}
print FOUTPL "End Week $week\n";
$week++;
    }

}
#####################################################################
#####################################################################
sub TeamByTeamMatrix()
{
   my $function_name = shift @_;
   my $function_header = shift @_;
   my $DataRef = shift @_;
   my @Data = @{$DataRef};
   my $second_matrix = shift @_;
   my $matrix_name=substr("$function_name",0,1);

   print "function name=$function_name, matrix name=$matrix_name\n";

   print FOUTS "$function_header";
   if($second_matrix eq "0"){
     print FOUTS "function [$matrix_name]=$function_name()\n";
   }
   else{
     print FOUTS "function [$matrix_name,$second_matrix]=
                                                $function_name()\n";
   }


   print FOUTS "${matrix_name}=[";
         for($i=1;$i<=$teamcount;$i++){
            if($i>1){
            print FOUTS ";\n";
            }
            for($j=1;$j<=$teamcount;$j++)
               {
               if($j<$teamcount){
                  print FOUTS "$Data[$i][$j] ";
               }
               else{
                  print FOUTS "$Data[$i][$j]";
               }
```

```
            }
          }
   print FOUTS "];\n\n";
}

#####################################################################
#####################################################################
sub Data_To_DiffMat()
{
   my $team1stat = shift @_;
   my $team2stat = shift @_;
   my $team1index = shift @_;
   my $team2index = shift @_;
   my $DataRef = shift @_;
   my @Data = @{$DataRef};


   if($team1stat>$team2stat){ ## TEAM 1 SCORED MORE POINTS THAN TEAM 2
      ## TEAM 2 LOST TO TEAM 1, HENCE TEAM 2 "votes" for TEAM 1,
      ## ADD SCORE DIFFERENCE TO (team 2, team1) ENTRY
      $Data[$team2index][$team1index]=$Data[$team2index][$team1index]+
                                                $team1stat-$team2stat;
   }
   elsif($team1stat<$team2stat){ ## TEAM 2 SCORED MORE POINTS THAN TEAM 1
      ## TEAM 1 LOST TO TEAM 2, HENCE TEAM 1 "votes" for TEAM 2,
      ## ADD SCORE DIFFERENCE TO (team 1, team2) ENTRY
      $Data[$team1index][$team2index]=$Data[$team1index][$team2index]+
                                                $team2stat-$team1stat;
      }
  else{
      print "WARNING: teams tied, can not do week $weekcount, teams-
                                    $team1index and $team2index  . \n";
  }


}

#####################################################################
################### EXECUTION #######################################
$year=$ARGV[0];
#$year=2007;
$lastweek=$ARGV[1];
#$lastweek=21;
```

```
#print "Processing the year $year ... ";
$DATAPATHNAME="c:/DataProcessing/NFL/";
$FORMATEDPATHNAME="c:/DataProcessing/FormatedDataFiles/";
$PATHNAME="c:/MATLAB7/Research/NFL/RankingPrograms/NFL_Data/$year/";
require "${DATAPATHNAME}NFL_Roster/roster$year.pl";

## SET UP TEAM NAMES HASH
my @temp=@names;
$teamcount=$#temp+1; ## NUMBER OF TEAMS IN THE ROSTER
                     ## (ARRAY COUNTS FROM 0)

%team_names=SetUpNameHash();

## READ IN THE FILE CONTAINING THE LIST OF FILE NAMES
## CONTAINING WEEKLY DATA
open(FIN,"<${DATAPATHNAME}Statistics$year/filepaths.txt")
or die "Could not open ${DATAPATHNAME}Statistics$year/filepaths.txt";
@fp=<FIN>;
close(FIN);
chomp(@fp);

@Massey_SD=();
@Colley_Matrix=();
@Colley_WinLoss=();

@Team_TeamWins=();
@Matchup_list=();

@Stats_List=();
@Scores_List=();
@Rushing_Yards_List=();

@Score_Diff=();
@Last_Score_Diff=();
@Yards_Diff=();
@Rushing_Yards_Diff=();
@Passing_Yards_Diff=();
@First_Downs_Diff=();
@Return_Yards_Diff=();

$weekcount=0;
$stats_num=5;     ## NUMBER OF STATS I CURRENTLY ACQUIRE,
```

```
                  ## NOTE: THIS MIGHT CHANGE!!
my $count=0;
my $game=0;
my ($team1index,$team2index,$score1,$score2);

for($i=1;$i<=$teamcount;$i++){
    $Massey_SD[$i]=0;

    for($j=1;$j<=3;$j++){
      $Colley_WinLoss[$i][$j]=0;
    }

    for($j=1;$j<=$teamcount;$j++){
      $Colley_Matrix[$i][$j]=0;
      $Last_Score_Diff[$i][$j]=0;

      $Team_TeamWins[$i][$j]=0;

      $Scores_List[$i][$j]=0;
      $Rushing_Yards_List[$i][$j]=0;

      $Score_Diff[$i][$j]=0;
      $Yards_Diff[$i][$j]=0;
      $Rushing_Yards_Diff[$i][$j]=0;
      $Passing_Yards_Diff[$i][$j]=0;
      $First_Downs_Diff[$i][$j]=0;
      $Return_Yards_Diff[$i][$j]=0;
    }
    for($k=1; $k<=$stats_num; $k++){
      $Stats_List[$i][$k]=0;
    }
}

foreach $line (@fp) ## LOOPING OVER ALL THE BOX SCORE
                    ## FILES FOR THE GIVEN YEAR
{
   if($line=~/week(\d\d)/) ## FOUND BEGINNING OF A
                           ## WEEK MARKER
   {
       if($1>$lastweek){ ## PARSE FILES UP TO AND
                         ## INCLUDING THE SPECIFIED WEEK
       last;
       }
       print "Found begining of week$1\n";
```

```
    $weekcount++;
    $count=0;
    $game=1;

}
elsif($line=~/end/)
{
   #print "Found end of week $weekcount\n";
}
else ## PARSE THE DATA FILE FOR THE GIVEN GAME
{
   $filename=$line;

   ## PARSE THE CURRENT GAME FILE AND OBTAIN THE INDICES OF
   ## THE TEAMS THEIR SCORES AND STATS
   @out=&ParseBoxScore();
   $team1index=$out[0][0];
   $team2index=$out[1][0];
   $score1=$out[0][1];
   $score2=$out[1][1];


   ## SET UP THE CURRENT GAME MATCH UP FOR THE WEEK
   $Matchup_list[$weekcount][$count][0]= $team1index;
   $Matchup_list[$weekcount][$count][1]= $score1;
   $Matchup_list[$weekcount][$count][2]= $team2index;
   $Matchup_list[$weekcount][$count][3]= $score2;


   ## ACCUMULATE TOTAL AMOUNT OF SCORES AND RUSHING YARDS
   ## FOR EACH TEAM
   $Scores_List[$team1index][$team2index] =
                  $Scores_List[$team1index][$team2index]+$score1;
   $Scores_List[$team2index][$team1index] =
                  $Scores_List[$team2index][$team1index]+$score2;

   $Rushing_Yards_List[$team1index][$team2index]=
        $Rushing_Yards_List[$team1index][$team2index]+$out[0][3];
   $Rushing_Yards_List[$team2index][$team1index]=
        $Rushing_Yards_List[$team2index][$team1index]+$out[1][3];

   for($k=1; $k<=$stats_num;$k++){
         $Stats_List[$team1index][$k] =$Stats_List[$team1index][$k]+
                                                    $out[1][$k+1];
```

```
          $Stats_List[$team2index][$k] =$Stats_List[$team2index][$k]+
                                                        $out[0][$k+1];
    }
    $count++;

   $Massey_SD[$team1index]=$Massey_SD[$team1index]+$score1-$score2;
   $Massey_SD[$team2index]=$Massey_SD[$team2index]+$score2-$score1;

   $Colley_Matrix[$team1index][$team2index]=
                         $Colley_Matrix[$team1index][$team2index]-1;
   $Colley_Matrix[$team2index][$team1index]=
                         $Colley_Matrix[$team2index][$team1index]-1;

    ## SET UP MASSEY NORMAL MATRIX, MASSEY GAME MATRIX ROW FOR THE
    ## CURRENT GAME OF THE WEEK
    if($score1>$score2){ ## team 1 SCORED MORE POINTS THEN team 2
        ## INCREASE WINS OF team 1
$Colley_WinLoss[$team1index][1]=
                                    $Colley_WinLoss[$team1index][1]+1;
        ## INCREASE TOTALS OF team 1
$Colley_WinLoss[$team1index][3]=
                                    $Colley_WinLoss[$team1index][3]+1;

        ## INCREASE LOSSES OF team 2
$Colley_WinLoss[$team2index][2]=
                                    $Colley_WinLoss[$team2index][2]+1;
        ## INCREASE TOTALS OF team 2
$Colley_WinLoss[$team2index][3]=
                                    $Colley_WinLoss[$team2index][3]+1;

        ## TEAM 1 BEAT TEAM 2, INCREASE NUMBER OF WINS BY TEAM 1
        ##AGAINST TEAM 2 BY 1
        $Team_TeamWins[$team1index][$team2index]=
                         $Team_TeamWins[$team1index][$team2index]+1;

## TEAM 2 LOST TO TEAM 1, HENCE TEAM 2 "votes" for TEAM 1,
        ## ADD SCORE DIFFERENCE TO (team 2, team1) ENTRY
if($weekcount==$lastweek){
   $Last_Score_Diff[$team2index][$team1index]=
        $Last_Score_Diff[$team2index][$team1index]+$score1-$score2;
}
    }
    elsif($score1<$score2){ ## team 2 SCORED MORE POINTS THEN team 1
        ## INCREASE LOSSES OF Team 1
```

```
$Colley_WinLoss[$team1index][2]=
                                    $Colley_WinLoss[$team1index][2]+1;
        ## INCREASE TOTAL OF Team 1
$Colley_WinLoss[$team1index][3]=
                                    $Colley_WinLoss[$team1index][3]+1;

        ## INCREASE WINS OF Team 2
$Colley_WinLoss[$team2index][1]=
                                    $Colley_WinLoss[$team2index][1]+1;
        ## INCREASE TOTAL OF Team 2
$Colley_WinLoss[$team2index][3]=
                                    $Colley_WinLoss[$team2index][3]+1;

## TEAM 2 BEAT TEAM 1, INCREASE NUMBER OF WINS BY TEAM 2
        ## AGAINST TEAM 1 BY 1
        $Team_TeamWins[$team2index][$team1index]=
                          $Team_TeamWins[$team2index][$team1index]+1;

## TEAM 1 LOST TO TEAM 2, HENCE TEAM 1 "votes" for TEAM 2,
        ## ADD SCORE DIFFERENCE TO (team 1, team2) ENTRY
if($weekcount==$lastweek){
   $Last_Score_Diff[$team1index][$team2index]=
         $Last_Score_Diff[$team1index][$team2index]+$score2-$score1;
}
    }
    else{
        print "WARNING: teams tied, can not do week $weekcount,
                                    game $game for Massey matrix. \n";
    }
    &Data_To_DiffMat($score1,$score2,$team1index,$team2index,
                                                        \@Score_Diff);
    &Data_To_DiffMat($out[0][2],$out[1][2],$team1index,$team2index,
                                                        \@Yards_Diff);
    &Data_To_DiffMat($out[0][3],$out[1][3],$team1index,$team2index,
                                                   \@Rushing_Yards_Diff);
    &Data_To_DiffMat($out[0][4],$out[1][4],$team1index,$team2index,
                                                   \@Passing_Yards_Diff);
    &Data_To_DiffMat($out[0][5],$out[1][5],$team1index,$team2index,
                                                     \@First_Downs_Diff);
    &Data_To_DiffMat($out[0][6],$out[1][6],$team1index,$team2index,
                                                    \@Return_Yards_Diff);
    $game++;

}
```

```
  #print "DONE\n";
  #print "now weekcount=$weekcount\n";
}

## OPEN THE DATA FILES FOR OUTPUT

print "$weekcount\n";

open(FOUTST,">${PATHNAME}GameStats.m")
or die "Could not open ${PATHNAME}GameStats.m";

open(FOUTM,">${PATHNAME}MatchUps.m")
or die "Could not open ${PATHNAME}MatchUps.m";

open(FOUTSch,">${PATHNAME}NFL${year}schedule.m")
or die "Could not open ${PATHNAME}NFL${year}schedule.m";

open(FOUTGAMES,">${DATAPATHNAME}Statistics$year/GameCount$year.pl")
or die "Could not open ${DATAPATHNAME}Statistics$year
                                              /GameCount$year.pl";

open(FOUTPL,"> ${DATAPATHNAME}GamePredictions/${year}
/matches_${year}.txt") or die "Could not open
${DATAPATHNAME}GamePredictions/${year}/matches_${year}.txt";


CreateGameData();

## CLOSE ALL OUTPUT FILES
close(FOUTST);
close(FOUTM);
close(FOUTSch);
close(FOUTGAMES);
close(FOUTPL);

#print "DONE\n";
#print "Number of teams in the year $year is $teamcount.\n";
```

B.5 Game Prediction Scripts

```
#!/usr/bin/perl
#####################################################################
############################## EXECUTION ############################
```

```
$season=  $ARGV[0];
$week= $ARGV[1];
$epsilon =   $ARGV[2];

   my $e=substr($epsilon,2,length($epsilon));
   my $count=1;
   my @ft=();
   my @rating=();
   my ($team1,$team2,$score1,$score2,$games);
   my @result=();
   my @predresult=();
   my @aggregresult=();
   my ($i,$j,$k,$p);

   $PATHNAME="c:/DataProcessing/NFL/GamePredictions/";


   ### LOAD IN THE RATING VECTORS
   open(FIN,"<${PATHNAME}Ratings/ratings_${season}.txt")
or die "Could not open ${PATHNAME}Ratings/ratings_${season}.txt";
   @fp=<FIN>;
   close(FIN);
   chomp(@fp);

   foreach $line (@fp){
   #print "Line: $line\n";
       if($line=~/\s*(\D+),\s*(\S+),\s*(\S+),\s*(\S+),\s*(\S+),
                            \s*(\S+),\s*(\S+),\s*(\S+),\s*(\S+)/){
       $alg[0]='Total Games';
       $alg[1]='Aggregated Predictions';
       $alg[2]='Aggregated Ranks';
       $alg[3]=$1;
       $alg[4]=$2;
       $alg[5]=$3;
       $alg[6]=$4;
       $alg[7]=$5;
       $alg[8]=$6;
       $alg[9]=$7;
       $alg[10]=$8;
       $alg[11]=$9;
       }
       elsif($line=~/\s*(\S+),\s*(\S+),\s*(\S+),\s*(\S+),\s*(\S+),
                           \s*(\S+),\s*(\S+),\s*(\S+),\s*(\S+)/){
       #print "What a f is there: $1,$2,$3,$4\n";
```

```
        $rating[3][$count]=$1;
        $rating[4][$count]=$2;
        $rating[5][$count]=$3;
        $rating[6][$count]=$4;
        $rating[7][$count]=$5;
        $rating[8][$count]=$6;
        $rating[9][$count]=$7;
        $rating[10][$count]=$8;
        $rating[11][$count]=$9;
        $count++;

        }
    }

    open(FIN,"<${PATHNAME}Ratings/agr_ratings_${season}.txt")
or die "Could not open ${PATHNAME}Ratings/agr_ratings_${season}.txt";
    @fp=<FIN>;
    close(FIN);
    chomp(@fp);

    $count=1;
    foreach $line (@fp){
        if($line=~/\s*(\S+),/){
        $rating[2][$count]=$1;
        $count++;

        }
       elsif($line=~/\s*(\D+)/){
        $alg[2]=$1;        ## AGGREGATED RATINGS

        }
    }

    ### LOAD IN THIS YEARS MATCHUPS
    open(FIN,"<${PATHNAME}${season}/matches_${season}.txt")
or die "Could not open ${PATHNAME}${season}matches_${season}.txt";
    @fp=<FIN>;
    close(FIN);
    chomp(@fp);
    $flag=0;

    foreach $line (@fp){
        if($line=~/Begin Week $week/){
        $flag=1;
```

```
        print "Found week $week\n";
        $count=0;               ## GAME COUNTER
        for($a=0;$a<=$#alg;$a++){
    $predresult[$a]=0;
}

        }
        elsif($line=~/End Week $week/){
        print "Found End of Week $week\n";
        #$count--;
        $predresult[0]=$count;
        $flag=0;
                last;

        }
        elsif($flag){

for($i=0;$i<4;$i++){
    $aggregresult[$count][$i]=0;
         }
         if($line=~/(\d*),(\S+),(\S+),(\S+)/){
        $team1=$1;
        $score1=$2;
        $team2=$3;
        $score2=$4;

        if($score1>$score2){ ## TRUE RESULT: TEAM 1 WON
           $result[0][0]=$team1;
           $result[0][1]=$team2;
           $result[0][2]=1;

                    $aggregresult[$count][0]=1;
         }
        elsif($score1<$score2){## TRUE RESULT: TEAM 2 WON
          $result[0][0]=$team2;
          $result[0][1]=$team1;
          $result[0][2]=1;
                  $aggregresult[$count][0]=2;
         }
         else{                  ## TRUE RESULT: TEAMS TIED
          $result[0][0]=$team1;
          $result[0][1]=$team2;
          $result[0][2]=0;
```

```
                $aggregresult[$count][0]=3;
        }
       for($a=2;$a<=$#alg;$a++){  ## PREDICTIONS
                ## TEAM 1 PREDICTED TO WIN
        if($rating[$a][$team1]>$rating[$a][$team2]){
               $result[$a][0]=$team1;
               $result[$a][1]=$team2;
               $result[$a][2]=1;
                       $aggregresult[$count][1]=
                                    $aggregresult[$count][1]+1;
}
                ## TEAM 2 PREDICTED TO WIN
elsif($rating[$a][$team1]<$rating[$a][$team2]){
$result[$a][0]=$team2;
$result[$a][1]=$team1;
$result[$a][2]=1;

                       $aggregresult[$count][2]=
                                    $aggregresult[$count][2]+1;
 }
else{            ## TEAMS ARE PREDICTED TO TIE
$result[$a][0]=$team1;
$result[$a][1]=$team2;
$result[$a][2]=0;
 }

if(($result[$a][0]==$result[0][0])&&
                              ($result[$a][2]==$result[0][2])){
$result[$a][3]=1;
$predresult[$a]=$predresult[$a]+1;
}
else {
$result[$a][3]=0;
        }
}
               ## STORE THE ODM PREDICTION RESULT FOR THE GIVEN GAME
               $aggregresult[$count][3]=$result[$#alg][3];
       $count++;

       }}
   }
   print "Total number of games-> $count\n";

   for($i=0;$i<$count;$i++){
```

```
        ## CUMULATIVE PREDICTION FOR TEAM 1 TO WIN
  if($aggregresult[$i][1]>$aggregresult[$i][2]){
                ## REAL RESULT TEAM 1 WON
  if($aggregresult[$i][0]==1){
                    $predresult[1]=$predresult[1]+1;
  }
  }
        ## CUMULATIVE PREDICTION FOR TEAM 2 TO WIN
  elsif($aggregresult[$i][2]>$aggregresult[$i][1]){
                ## REAL RESULT TEAM 2 WON
  if($aggregresult[$i][0]==2){
                    $predresult[1]=$predresult[1]+1;
  }
  }
  else{ ## CUMULATIVE PREDICTION IS SPLIT
              ## (USE ODM AS A TIE BREAKER)
                    $predresult[1]=$predresult[1]+
                                        $aggregresult[$i][3];

  }
  }

  ## APPEND THE RESULTS OF THE PREDICTIONS INTO A
  ## FILE IN THE CURRENT SEASON
  open(FOUTFIN,">>${PATHNAME}${season}/Daily_NFL_Pred.tex")
   or die "Could not open ${PATHNAME}${season}/Daily_NFL_Pred.tex";

     print FOUTFIN "$season&$week&$predresult[0] ";
     print "games-->$predresult[0],Algorithms-->$#alg\n";
     for($a=1;$a<=$#alg;$a++){
$games=sprintf("%.0f",$predresult[$a]);
        print FOUTFIN "&$games";

        print  "&$games";
     }
        print  "\n";
        print FOUTFIN "\n";
  close(FOUTFIN);

#!/usr/bin/perl
#####################################################################
############################ EXECUTION ##############################
my $season=  $ARGV[0];
my $epsilon=  "$ARGV[1]";
```

```
my $e=substr($epsilon,2,length($epsilon));
my $PATHNAME="c:/DataProcessing/NFL/GamePredictions/";
my ($year,$day);
my $totals=();

open(FIN,"<${PATHNAME}$season/Daily_NFL_Pred.tex")
or die "Could not open ${PATHNAME}$season/Daily_NFL_Pred.tex";
@fp=<FIN>;
close(FIN);
chomp(@fp);

$totals[0] =0;
$totals[1] =0;
$totals[2] =0;
$totals[3] =0;
$totals[4] =0;
$totals[5] =0;
$totals[6] =0;
$totals[7] =0;
$totals[8] =0;
$totals[9] =0;
$totals[10]=0;
$totals[11]=0;


   foreach $line (@fp){
   #print "Processing: $line\n";
       if($line=~/(\d+)&(\d+)&(\d+)\s*&(\d+)&(\d+)&(\d+)
     &(\d+)&(\d+)&(\d+)&(\d+)&(\d+)&(\d+)&(\d+)&(\d+)/){
       $year=$1;
       $day=$2;
       $totals[0] =$totals[0]+$3;
       $totals[1] =$totals[1]+$4;
       $totals[2] =$totals[2]+$5;
       $totals[3] =$totals[3]+$6;
       $totals[4] =$totals[4]+$7;
       $totals[5] =$totals[5]+$8;
       $totals[6] =$totals[6]+$9;
       $totals[7] =$totals[7]+$10;
       $totals[8] =$totals[8]+$11;
       $totals[9] =$totals[9]+$12;
       $totals[10]=$totals[10]+$13;
       $totals[11]=$totals[11]+$14;
       }
```

```
   }

   open(FOUTFIN,">>${PATHNAME}FullSeasonTotalsNFL.tex")
  or die "Could not open ${PATHNAME}FullSeasonTotalsNFL.tex";
   print FOUTFIN "$season ";

   for($i=1;$i<=$#totals;$i++){
   $games=sprintf("%.2f",100*$totals[$i]/$totals[0]);
   print FOUTFIN "&$games";
   }
print FOUTFIN "\n";
close(FOUTFIN);
```