Relieving and Readjusting Pythagoras

Improving the Pythagorean Formula

by

Victor Luo

Steven J. Miller, Advisor

A thesis submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Arts with Honors
in Mathematics

Williams College
Williamstown, Massachusetts

May 20, 2014

Abstract

The Pythagorean expectation was invented by Bill James in the late 70's as a way of calculating how many wins a baseball team should have by utilizing just runs scored and runs allowed. His original formula predicts a winning percentage of $\frac{RS^2}{RS^2+RA^2}$, where RS stands for runs scored and RA stands for runs allowed. Although the simplicity of the formula is a thing of beauty, with the development of more advanced baseball statistics it should be possible to enhance the formula such that it gives a more accurate prediction of a team's wins. Implementing statistics such as ballpark effect as well as accounting for game state factors, we will test to see if it is indeed the case that adjusting the Pythagorean expectation formula gives a statistically significantly better prediction for a team's wins than the unadjusted formula.

In order to test these adjusted formulas, we will be culling data from the internet, specifically from `http://www.retrosheet.org/gamelogs/index.html` and `espn.com`. We will then import this data into R and use our code to manipulate the data, calculating the new adjusted Pythagorean expectation and old Pythagorean expectation by year for each team. Then, using different regression models, we will determine which expectation formula is more accurate.

In addition, it has been shown that we can use a Weibull distribution in order to model run production. The versatility of the distribution is due to the fact that it accounts for three parameters that can be varied to adjust the spread, shift, and scale of the distribution. We will explore whether a linear combination of Weibulls is able to more accurately determine a team's run production.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis advisor, Professor Steven J. Miller. Without his assistance and enduring support throughout the time of research and writing of this thesis, I would not have been able to complete this thesis. I could not have asked for a better and more involved advisor for my undergraduate thesis.

I would also like to thank my second reader, Professor Qing Wang, for her encouragement and time in reading my thesis, as well as her meticulous and insightful comments.

Last but not least, I would like to thank my friends and family: my parents Tie Luo and Ching Hua, my brother Lawrence Luo, my teammates on WUFO, and those that I have met and become close with in my four years at Williams. Their support has meant the world to me, and without it, the road would have been that much tougher in completion of this thesis.

Contents

1. Introduction

1.1. **The History of the Pythagorean Win/Loss Formula.** Bill James first started writing about baseball statistics as a security guard at a cannery. His writings contrasted to contemporary baseball pieces, which instead recounted games in meticulous detail. In order to expand his audience, James published his *Baseball Abstract* in 1977, which had nearly one hundred pages of comprehensive analysis of data and statistics James compiled from the 1976 season. One of these statistics was the Pythagorean Win/Loss Formula. The Pythagorean Win/Loss Formula, also known as the Pythagorean formula or Pythagorean expectation, was invented by James in order to predict how many wins a baseball team should have won, given their runs scored and runs allowed. The original formula is given as

$$\text{Win/Loss Percentage} = \frac{\text{runs scored}^\gamma}{\text{runs scored}^\gamma + \text{runs allowed}^\gamma}.$$

To find the expected number of wins, one only need multiply the team's Win/Loss Percentage by the number of games the team played in the season. γ was initially taken to be 2, but through analysis on observed runs scored and allowed, a γ of 1.83 produces a more accurate Win/Loss Percentage. One is taken back by how simple the formula is, requiring only the runs scored and allowed by a team in a season, and the calculation can be done on any calculator. This simple, closed form expression of a team's Win/Loss Percentage of only three parameters allows for easy calculations. However, having such a simple formula certainly has its drawbacks, most notably it cannot address all the subtleties and issues that come with a complicated game like baseball.

1.2. **Previous Findings.** [MIL] and [MCGLP] provided theoretical justification for the Pythagorean W/L Formula by modeling runs scored and allowed by independent Weibull distributions. It was established that the single Weibull distribution did a very good job of mapping runs scored and allowed, and lead to a predicted win-loss percentage of

$$\frac{(RS-\beta)^\gamma}{(RS-\beta)^\gamma+(RA-\beta)^\gamma},$$

where RS and RA are the means of the Weibull random variable corresponding to runs scored and allowed, respectively, while $RS-\beta$ and $RA-\beta$ are estimators of the observed runs scored and allowed. Analysis on the 14 American League teams from the 2004 season showed that runs scored and allowed were statistically independent and that the single Weibull with best fit parameters obtained from the maximum likelihood method and least squares method provide good fits for the observed data.

1.3. **Thesis Results.** Adjusting runs for certain sabermetric statistics that are discussed in Section 3 does not seem to increase the accuracy of the Pythagorean W/L Formula. Accuracy testing was done in R, and the code used for each statistic is displayed in the Appendices. However, it was found that using a linear combination of Weibulls rather than a single Weibull increases the prediction accuracy of a team's W/L percentage. More specifically, using Mathematica on the years from 2004-2012, we saw that the single Weibull's predictions for a team's wins were on average 4.22 games off (with a standard deviation of 3.03), while the linear combination of Weibull's predictions for a team's wins were from 2004-2012 were on average 3.11 games off (with a standard deviation of 2.33), producing about a 25% increase in prediction accuracy. We also compare our results to ESPN's Expected WL and comment. Finally, we perform χ^2 goodness of fit tests for the linear combination of Weibulls and test the statistical independence of runs scored and allowed (a necessary assumption), and see that in fact the linear combination of Weibulls with properly estimated parameters obtained from least squares analysis closely maps the observed runs and that runs scored and allowed are in fact statistically independent.

1.4. **Outline of the Thesis.** In this paper, we will first present preliminaries needed, notably the Weibull distribution and the method of culling data from the internet. We will then analyze whether the prediction accuracy of the Pythagorean W/L Formula when the runs scored and allowed are adjusted for various sabermetric statistics increases as these statistics are added in. Finally, we extend the work done in [MIL]; namely, we will see if a linear combination of Weibulls is a more precise model in mapping runs.

2. Preliminaries

In this section, we present the Weibull distribution, which will be utilized in great amounts later on. In addition, we also analyze tools that were used in order to test these statistics, and give a glimpse into the code that was written to experiment with these statistics.

2.1. **The Weibull Distribution.**

The Pythagorean Formula is given by $\frac{RS_{obs}^{\gamma}}{RS_{obs}^{\gamma}+RA_{obs}^{\gamma}}$, where γ is constant throughout the league. To give a theoretical justification for the Pythagorean Formula and a value of γ, the number of runs scored and allowed in baseball games can be modeled as independent random variables taken from Weibull distributions with the same β and γ by different α; the probability density of the Weibull distribution is given by

$$f(x;\alpha,\beta,\gamma) = \begin{cases} \frac{\gamma}{\alpha}((x-\beta)/\alpha)^{\gamma-1}e^{-((x-\beta)/\alpha)^{\gamma}} & \text{if } x \geq \beta \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Using a single Weibull model leads (see [MIL]) to a predicted won-loss percentage of

$$\text{Won-Loss Percentage}(RS,RA,\beta,\gamma) = \frac{(RS-\beta)^{\gamma}}{(RS-\beta)^{\gamma}+(RA-\beta)^{\gamma}}.$$

It is important to note that we assume that runs scored and allowed are taken from continuous, not discrete, distributions. This allows for us to deal with continuous integrals rather than discrete sums, which most of the time leads to easier calculations. While a discrete distribution would probably more effectively map runs in baseball, the assumption of drawing runs from a continuous distribution allows for more manageable calculations, and is a very sensible estimate of those runs observed. It also will lead to closed form expressions, which are much easier to work with. Weibulls lead to significantly easier calculations because if we have a random variable X chosen from a Weibull distribution with parameters α, β, and γ, it is the case that $X^{1/\gamma}$ is exponentially distributed with parameter α^{γ}; thus, a change of variables will lead to a much simpler integral of exponentials, which will be able to be done in closed form (see Appendix 9.1 in [MCGLP]).

The choice of the Weibull distribution is natural, as it takes on three flexible parameters which allow for wide and various shapes of the distribution, and given the almost random and erratic behavior of runs, it is fitting to choose a distribution with such a dynamic and malleable nature. α determines the spread of the distribution, namely as α increases, the distribution becomes more spread out. In addition, γ, arguably the most important of the parameters, causes the distribution to have many varying shapes. This is demonstrated in Figure 1. Finally, β shifts the distribution along the real line.

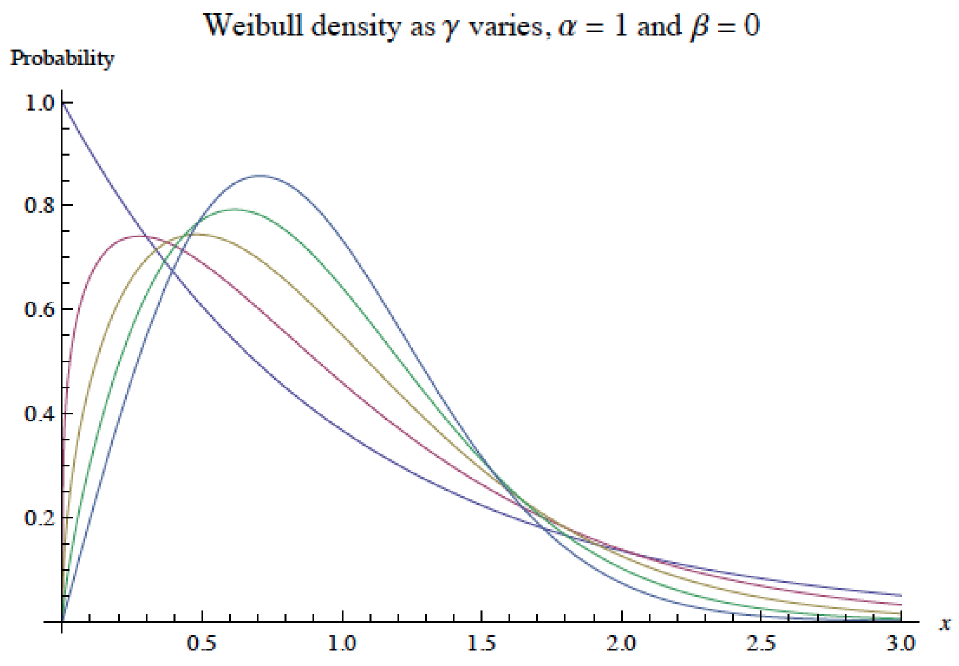


FIGURE 1. The varying distributions of the Weibull family with $\alpha = 1$ and $\beta = 0$.

One other important assumption we will make is that $\beta = -1/2$ for a particular Weibull. Fixing β, along with γ, we can determine α so that the mean of our Weibull closely resembles those of the observed average runs scored/allowed per game (see (2.3)). Using the Method of Least Squares, we can find the best fit parameters α, β, and γ for the observed data. The choice of $\beta = -1/2$ breaks the data into bins

$$\left[-\frac{1}{2}, \frac{1}{2}\right), \left[\frac{1}{2}, \frac{3}{2}\right), \left[\frac{3}{2}, \frac{5}{2}\right), \cdots \tag{2.2}$$

so that the centers of the bins are integer values 0,1, 2, and so on. These are the possible scores in a baseball game, taking $\beta = -1/2$ causes the possible integers scores to be in the middle of each bin, which will lead to the least amount of issues in determining the best fit values.

Our final assumption is that runs scored and runs allowed are independent. This obviously cannot be true, as a baseball game never ends in a tie. So, if the Rangers and Angels are playing, if the Angels scored 4 runs, then the Rangers must have scored something other than 4 runs. However, analyzing the data shows that these issues will cancel each other out on average, and that runs score and allowed will tend to behave like they are statistically independent if the runs scored and allowed are different. This can be shown by constructing an independence test with structural zeros, namely values in the contingency table that are not accessible (see Appendix 9.2 in [MCGLP] for a more rigorous description of the method).

To end this section, we state the mean and variance of the Weibull distribution in a lemma, which will be used heavily later on in the thesis:

Lemma 2.1. *The mean, $\mu_{\alpha,\beta,\gamma}$, is given by*

$$\mu_{\alpha,\beta,\gamma} = \alpha\Gamma(1+\gamma^{-1}) + \beta \tag{2.3}$$

while the variance, $\sigma^2_{\alpha,\beta,\gamma}$, is given by

$$\sigma^2_{\alpha,\beta,\gamma} = \alpha^2\Gamma(1+2\gamma^{-1}) - \alpha^2\Gamma(1+\gamma^{-1})^2 \tag{2.4}$$

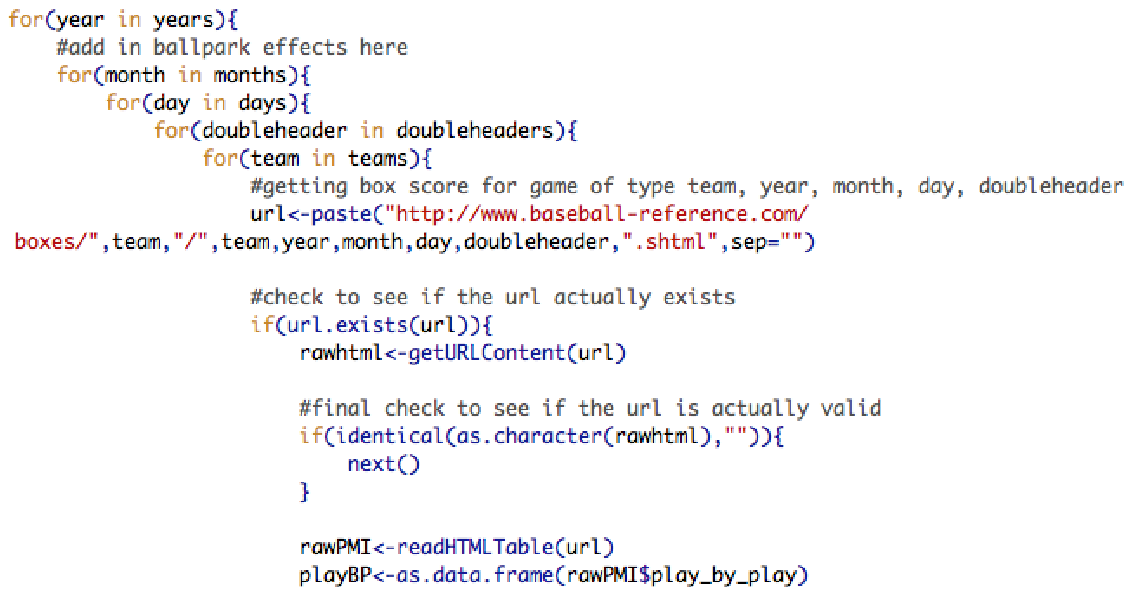
where $\Gamma(x)$ is the Gamma function, defined as $\Gamma(x) = \int_0^\infty e^{-u}u^{x-1}du$.

A derivation of the mean can be seen in Appendix 9.1 in [MCGLP]; the derivation of the variance follows in a similar fashion.

2.2. **Culling Data from the Internet.** An important component of this thesis was having the ability to cull data from the internet, namely bringing in many large data sets from the internet of player and team statistics. Without this power, it would nearly impossible to be able to test different formulas. Two main programs were used, namely R and a program written by Kevin Dayaratna.

In R, the main packages used were “XML" and “RCurl". First, the years were put into their own vector, as was the three letter abbreviation of each team. Then, for each team, in a given year, the url from baseball-reference was assigned. For example, if we were considering the Red Sox in the year 2008, the url would be `http://www.baseball-reference.com/teams/BOS/2008-pitching.shtml`. Then, using the command *rawPMI* <- *readHTMLTable*(url), the various tables on the page of the url are assigned to the variable *rawPMI*. Each table is given it's own name, and to call a table, say the table *players.pitching*, we need only enter *rawPMI$players.pitching*. This process can be seen in Figure 2. After this data is culled, it is very easy to manipulate it in R game by game, which allows for a wider range of sabermetric statistics to be used.

The second program, written by Kevin Dayaratna, only requires the three letter abbreviation of a team and a year.



```
for(year in years){
    #add in ballpark effects here
    for(month in months){
        for(day in days){
            for(doubleheader in doubleheaders){
                for(team in teams){
                    #getting box score for game of type team, year, month, day, doubleheader
                    url<-paste("http://www.baseball-reference.com/
boxes/",team,"/",team,year,month,day,doubleheader,".shtml",sep="")

                    #check to see if the url actually exists
                    if(url.exists(url)){
                        rawhtml<-getURLContent(url)

                        #final check to see if the url is actually valid
                        if(identical(as.character(rawhtml),"")){
                            next()
                        }

                        rawPMI<-readHTMLTable(url)
                        playBP<-as.data.frame(rawPMI$play_by_play)
```

FIGURE 2. Culling data from the internet using R.

Previewed in Figure 3, it writes that team's runs scored and runs allowed for that year to a text document in two columns, where the first column is runs scored and the second is runs allowed, taking the data from the baseball-almanac website. This was important in bringing in runs scored and allowed into Mathematica; we will discuss the importance of Mathematica later on in the paper.

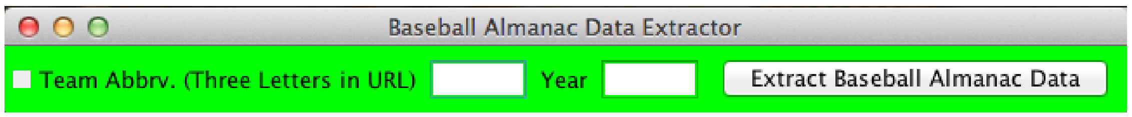

Baseball Almanac Data Extractor
Team Abbrv. (Three Letters in URL)
Year
Extract Baseball Almanac Data

FIGURE 3. Kevin Dayaratna's Baseball Almanac program.

It cannot be stressed enough how important the ability to cull data from the internet is. Without it, there would be much more time spent collecting data rather than analyzing data.

3. Adjusting for Sabermetric Statistcs

In this section, we analyze possible sabermetric statistics that can be used to adjust the Pythagorean W/L Formula to potentially increase its accuracy.

3.1. **Ballpark Effect Trends.** One statistic that is easily accessible and has an obvious effect on runs is ballpark factors. Ballpark factor is calculated as

$$\frac{\text{(Runs Scored at Home + Runs Allowed at Home)/(Number of Home Games)}}{\text{(Runs Scored Away + Runs Allowed Away)/(Number of Away Games)}}.$$

A stadium with a ballpark factor over 1 (labeled as "hitter friendly"), such as Coors Field, home of the Colorado Rockies, indicates that the team has more overall runs at home than away, while a stadium with a ballpark factor under 1 (labeled as "pitcher friendly") indicates that the team has less overall runs at home than away. It is apparent that ballpark factor can be a misleading statistic, as it does not account for pitching. If a team has a below average pitching staff, it could seem like the team's ballpark is "hitter friendly", as the team's pitchers are increasing the overall runs, namely by allowing more runs at home, causing the ballpark factor to increase. Figure 4 details the trends of ballpark factors from 2005 to 2012 (as listed on `espn.com`):

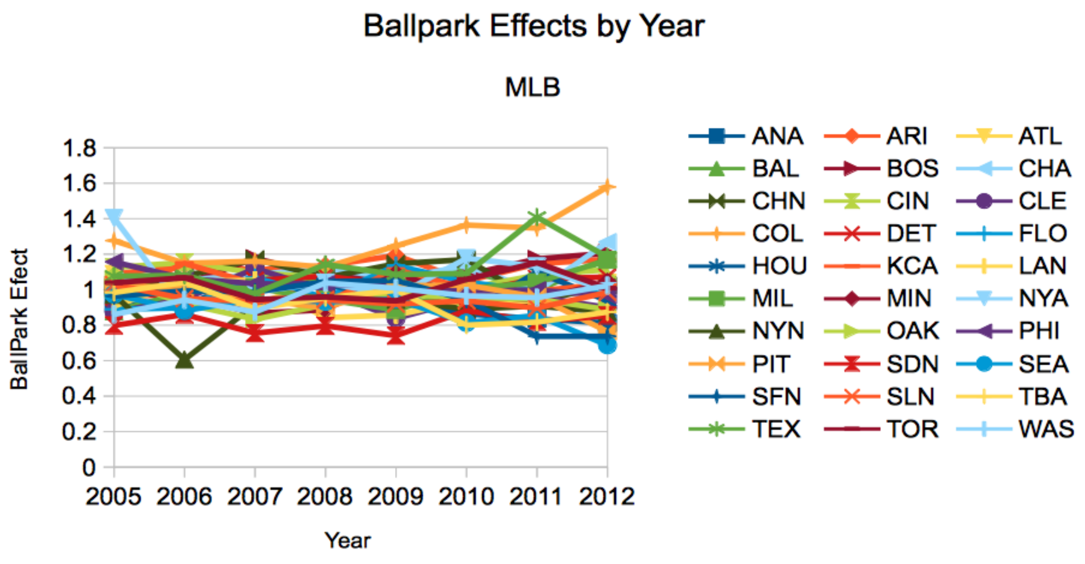


FIGURE 4. Ballpark factors from 2005-2012.

While a team's ballpark factor should be fairly constant over the years, there are a few teams that seem to defy this logic. Taking a closer look, four teams stand out, namely the Colorado Rockies (COL), New York Yankees (NYA), New York Mets (NYN), and Texas Rangers (TEX). Mapping their ballpark factor trends produces Figure 5.

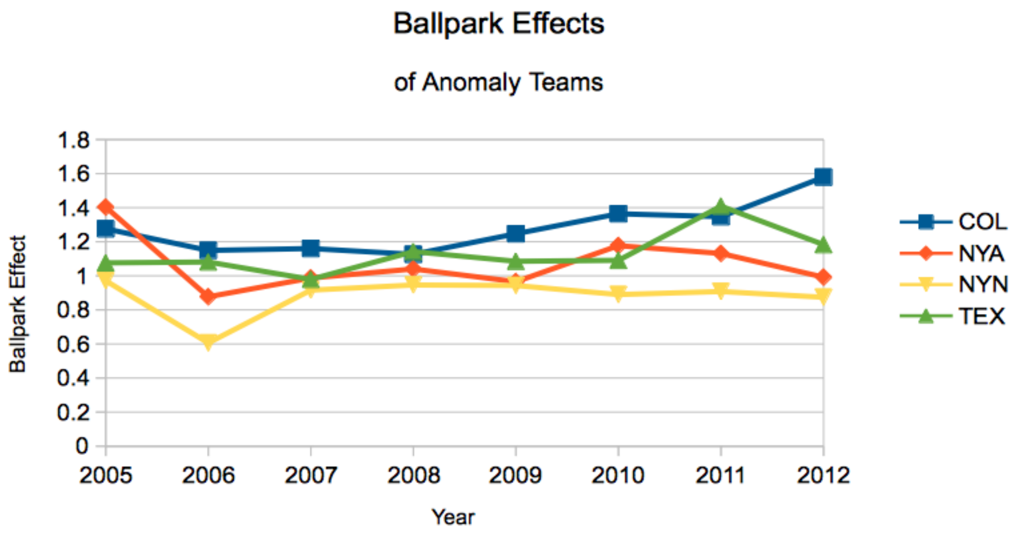


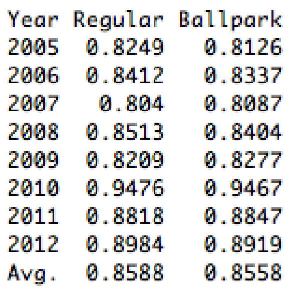
FIGURE 5. Anomaly ballpark factors from 2005-2012.

Each anomaly team's peculiar ballpark factor behavior can be attributed to a cause. In 2012, the Rockies allowed an absurd 890 runs, over 100 more than they did in 2011. With their already high ballpark factor, it should come as a surprise that a majority of the influx of runs allowed were at home, thus inflating the ballpark factor even more. From 2005 to 2006, the sharp decrease in ballpark factor for the Yankees was due to an increase in runs scored away and decrease in runs scored at home. 2005 was an out of ordinary year for the Yankees, in which they scored an abnormally large number of runs at home, thus leading to a very inflated ballpark factor. In 2006, the Mets saw a 112 run increase in runs scored and an 83 run increase in runs allowed. However, the majority of these runs scored/allowed were away from home, thus leading to an atypically low ballpark factor in 2006. Finally, the Rangers had an absurd year in 2011 in terms of offensive production, seeing a 68 run increase in runs scored, the majority of which came at home, so it isn't surprising that their ballpark factor in 2011 is so inflated.

To test whether ballpark factors could increase the predictive power of the Pythagorean W/L formula, we looked at the scores of each game in an MLB season. We then divided the scores by the ballpark factor of whichever stadium the teams were playing in. So, if the

Angels scored 4 runs and the Rangers scored 6 runs at Globe Life Park in Arlington (Texas Rangers Stadium), which has a ballpark factor of 1.183, then their adjusted runs scored are $4/1.183 = 3.38$ and $6/1.183 = 5.07$, respectively. It makes the most sense to divide by the ballpark factor, as if a stadium is hitter friendly, then the scores are inflated, and since the ballpark factor is greater than 1 when the stadium is hitter friendly, dividing will deflate the scores. Similarly, if the stadium is “pitcher friendly", then the scores are deflated, and since the ballpark factor is less than 1 when the stadium is pitcher friendly, dividing will inflate the scores.

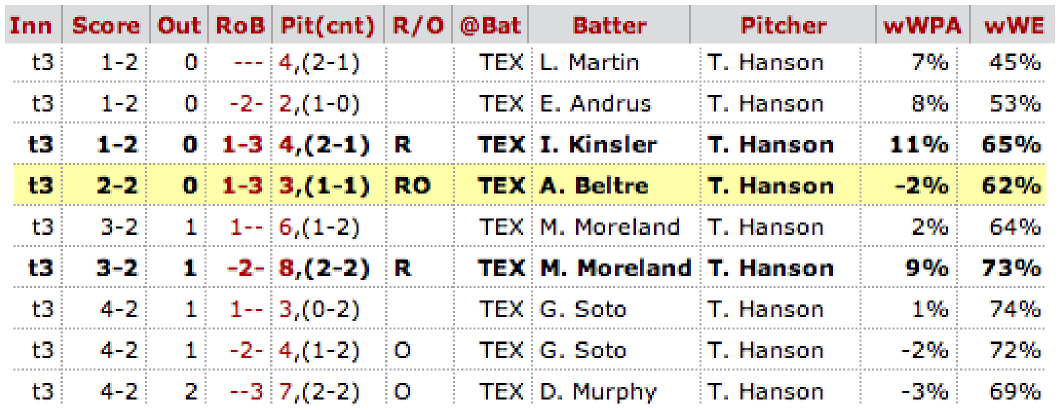
Figure 6 details the R-squared values given when estimating teams’ W/L percentage by the Pythagorean W/L formula and Pythagorean W/L formula adjusted for ballpark factors for each year from 2005 to 2012 using a gamma of 1.83, where the Pythagorean W/L is the formula defined by taking runs scored squared over the sum of runs scored squared and runs allowed squared. Interestingly enough, it seems that ballpark factors only slightly increase the predictive power of the Pythagorean W/L formula in some years; however, the minuscule increases observed in some years cannot be considered statistically significant by any means. While at first this may seem strange, as the ballpark that a team is playing in should play a large role in determining the runs scored and allowed, we must remember that there are 2,430 games in an MLB season. In addition, the home and away team in a game are both equally hit by the ballpark factor of the home team’s stadium. Taking these two factors into account, eventually it should make sense that the factors wash out over the entire season, thus leading to a predictive power at least on par with that of the original Pythagorean W/L formula.



```
Year Regular Ballpark
2005  0.8249   0.8126
2006  0.8412   0.8337
2007   0.804   0.8087
2008  0.8513   0.8404
2009  0.8209   0.8277
2010  0.9476   0.9467
2011  0.8818   0.8847
2012  0.8984   0.8919
Avg.  0.8588   0.8558
```

FIGURE 6. R-squared values of normal Pythagorean W/L (Original Pythag) and Pythagorean W/L adjusted for ballpark factors (Adjusted Pythag) from 2005-2012 using a gamma of 1.83.

3.2. **Game State.** The next sabermetric statistic considered was game state. Game state essentially takes the situation that the game is in (number of outs, which inning, players on bases, and current score) and calculates the percentage that the team at bat has of winning the game in that certain game state, using past game results that encountered the same game state in those games. The site `baseball-reference.com` does exactly this, labeling it as Winning Team Win Expectancy (wWE), as seen in Figure 7 when one looks at the box score for a particular game.



Inn	Score	Out	RoB	Pit(cnt)	R/O	@Bat	Batter	Pitcher	wWPA	wWE
t3	1-2	0	---	4,(2-1)		TEX	L. Martin	T. Hanson	7%	45%
t3	1-2	0	-2-	2,(1-0)		TEX	E. Andrus	T. Hanson	8%	53%
t3	**1-2**	**0**	**1-3**	**4,(2-1)**	**R**	**TEX**	**I. Kinsler**	**T. Hanson**	**11%**	**65%**
t3	**2-2**	**0**	**1-3**	**3,(1-1)**	**RO**	**TEX**	**A. Beltre**	**T. Hanson**	**-2%**	**62%**
t3	3-2	1	1--	6,(1-2)		TEX	M. Moreland	T. Hanson	2%	64%
t3	**3-2**	**1**	**-2-**	**8,(2-2)**	**R**	**TEX**	**M. Moreland**	**T. Hanson**	**9%**	**73%**
t3	4-2	1	1--	3,(0-2)		TEX	G. Soto	T. Hanson	1%	74%
t3	4-2	1	-2-	4,(1-2)	O	TEX	G. Soto	T. Hanson	-2%	72%
t3	4-2	2	--3	7,(2-2)	O	TEX	D. Murphy	T. Hanson	-3%	69%

FIGURE 7. Top of the 3rd inning from box score for Texas Rangers at Los Angeles Angels on August 7, 2013. `http://www.baseball-reference.com/boxes/ANA/ANA201308070.shtml`.

Accounting for game state is reasonable, as if a score progresses from say 9-0 in the bottom of the seventh to a final score of 12-0, those additional three runs are only inflating that team's total runs scored. For all intensive purposes, when the game is at 9-0 at the bottom of the seventh (or even sooner in the game), the game should be considered over and that score should be used when calculated a team's total runs scored and allowed at the end of the year.

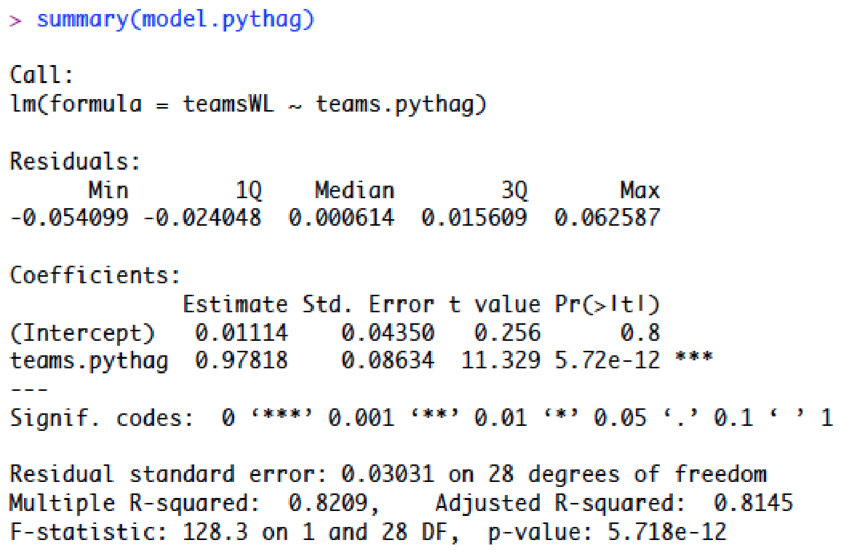
In order to implement this, we decided to pick a “threshold" wWE. Thus, if the game progressed to that threshold percentage (or 100 minus that threshold percentage, as wWE only represents the winning team), then we would consider the game over and use the scores at that state in the game as the final score. Two thresholds were used, namely 95% and 98%. In R, this required bringing in each box score of every game in a season (so 2430 urls to go through), and then bringing in the play by play table from the url, and finally searching through the table to find the required threshold and recording the scores of each team at that exact point. Going through so many urls and searches required a lot of computational capacity, and in order to get multiple years without spending weeks waiting for the data, we used the Amazon EC2 cloud. This allowed us to run code for different years at the same time in the cloud, which cut the culling time by about a quarter. We display the code used in Appendix A.

After the data was culled, we used it in the original Pythagorean W/L Formula and the formula accounting for ballpark factors. Unfortunately, the game state adjusted data does not have better predictive power than the original data; in fact, it is notably worse, as we display in Figure 8 for a threshold of 98% in 2009. For the regular Pythagorean W/L Formula, we have an adjusted R-squared value of 0.8145, while for the Pythagorean W/L Formula adjusted for game state at a 98% threshold and ballpark factors, the adjusted R-squared value is 0.7261.

This significant decrease in predictive power seems strange, as a threshold of 98% should give values almost similar to the original scores, with the exception of a few cases (the same decrease in power is evident in the case of a 95% threshold as well). This can probably be attributed to teams coming back more often than expected and the fact that we are throwing away good values that could evaluate a team. So, for example, if a team is down 0-6 in the bottom of the seventh, so the game is considered almost surely over, but comes back to win 7-6, those 7 runs thrown out are good values that are important in evaluating the team. So, we have problems with not looking at the entire game.

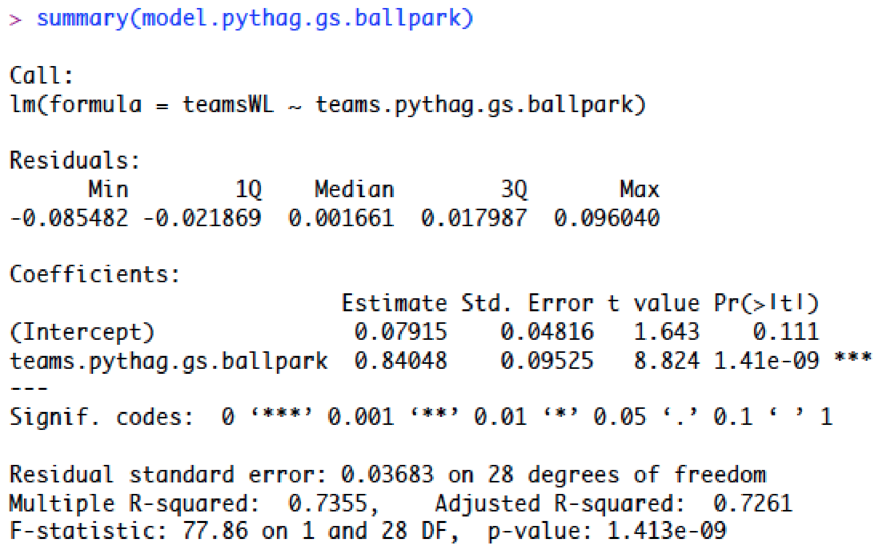
We also tried a weighted game state approach, where we used the approach above, but afterwards, we multiplied the runs scored and runs allowed in a game by a factor of 9/(inning stopped in) so that we could get the runs for a complete, nine inning game. The (inning stopped in) factor is the inning in which the team last batted in. For example, if we cut the game off at the top of the seventh inning, we multiply the cut off score of the current team at bat by 9/7, and the other team's cut off score by 9/6. If the game is cut off at the bottom of the fifth inning, we multiply the cut off score of both of the teams by 9/5. We display the results in Figure 9.

It seems that we run into the problems stated in the original game state case; using a weighted game state at 95% threshold gives an R-squared of 0.568, and that of a 98% threshold gives an R-squared of 0.574, a significant decrease from the 0.882 using the original Pythagorean W/L Formula. Again, this can probably be attributed to teams coming back more often than expected and the fact that we are throwing away good values that could evaluate a team.



```
> summary(model.pythag)

Call:
lm(formula = teamsWL ~ teams.pythag)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.054099 -0.024048  0.000614  0.015609  0.062587 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)   0.01114    0.04350   0.256      0.8    
teams.pythag  0.97818    0.08634  11.329 5.72e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 0.03031 on 28 degrees of freedom
Multiple R-squared:  0.8209,    Adjusted R-squared:  0.8145 
F-statistic: 128.3 on 1 and 28 DF,  p-value: 5.718e-12
```

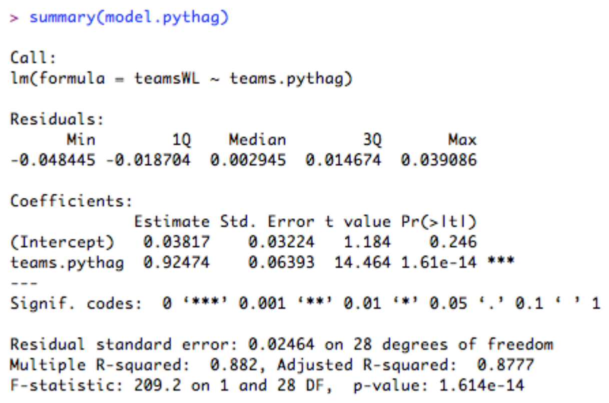
(a)



```
> summary(model.pythag.gs.ballpark)

Call:
lm(formula = teamsWL ~ teams.pythag.gs.ballpark)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.085482 -0.021869  0.001661  0.017987  0.096040 

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)    
(Intercept)               0.07915    0.04816   1.643    0.111    
teams.pythag.gs.ballpark  0.84048    0.09525   8.824 1.41e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 0.03683 on 28 degrees of freedom
Multiple R-squared:  0.7355,    Adjusted R-squared:  0.7261 
F-statistic: 77.86 on 1 and 28 DF,  p-value: 1.413e-09
```

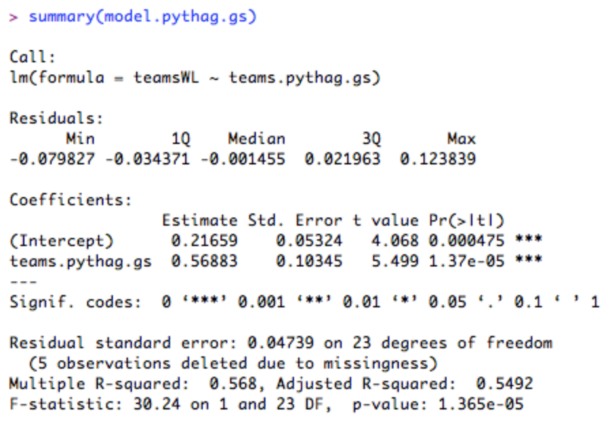
(b)

FIGURE 8. 2009 Teams' W/L percentage modeled by (a) Pythagorean W/L formula with original scores and (b) Teams' W/L percentage formula modeled by Pythagorean W/L adjusted for game state with 98% threshold and ballpark factors.



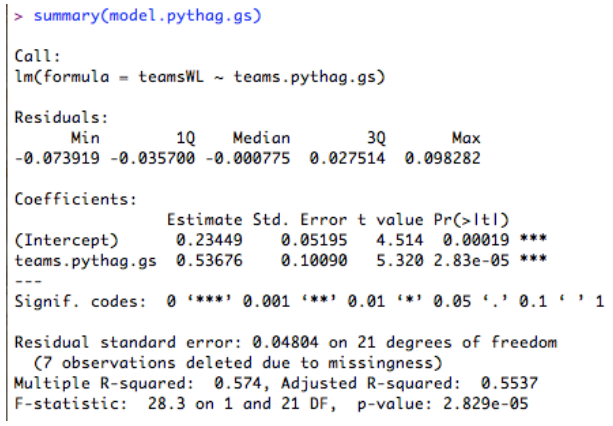
```
> summary(model.pythag)

Call:
lm(formula = teamsWL ~ teams.pythag)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.048445 -0.018704  0.002945  0.014674  0.039086 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)   0.03817    0.03224   1.184    0.246    
teams.pythag  0.92474    0.06393  14.464 1.61e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02464 on 28 degrees of freedom
Multiple R-squared:  0.882, Adjusted R-squared:  0.8777 
F-statistic: 209.2 on 1 and 28 DF,  p-value: 1.614e-14
```

(a)



```
> summary(model.pythag.gs)

Call:
lm(formula = teamsWL ~ teams.pythag.gs)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.079827 -0.034371 -0.001455  0.021963  0.123839 

Coefficients:
                Estimate Std. Error t value Pr(>|t|)    
(Intercept)      0.21659    0.05324   4.068 0.000475 ***
teams.pythag.gs  0.56883    0.10345   5.499 1.37e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.04739 on 23 degrees of freedom
  (5 observations deleted due to missingness)
Multiple R-squared:  0.568, Adjusted R-squared:  0.5492 
F-statistic: 30.24 on 1 and 23 DF,  p-value: 1.365e-05
```

(b)



```
> summary(model.pythag.gs)

Call:
lm(formula = teamsWL ~ teams.pythag.gs)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.073919 -0.035700 -0.000775  0.027514  0.098282 

Coefficients:
                Estimate Std. Error t value Pr(>|t|)    
(Intercept)      0.23449    0.05195   4.514  0.00019 ***
teams.pythag.gs  0.53676    0.10090   5.320 2.83e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.04804 on 21 degrees of freedom
  (7 observations deleted due to missingness)
Multiple R-squared:  0.574, Adjusted R-squared:  0.5537 
F-statistic:  28.3 on 1 and 21 DF,  p-value: 2.829e-05
```

(c)

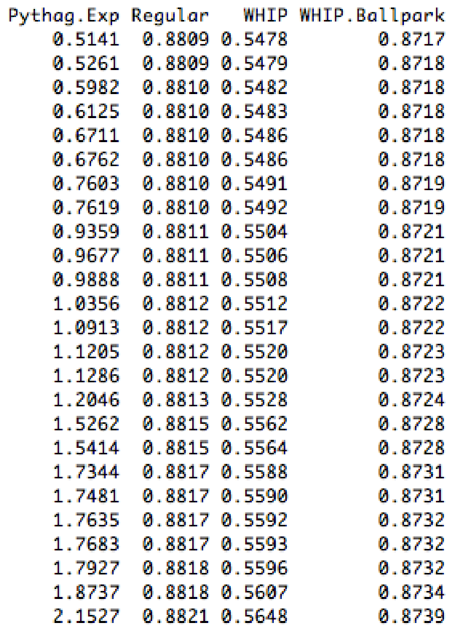
FIGURE 9. 2011 Teams' W/L percentage modeled by (a) Pythagorean W/L formula with original scores, (b) Teams' W/L percentage formula modeled by Pythagorean W/L adjusted for weighted game state with 95% threshold, and (c) Teams' W/L percentage formula modeled by Pythagorean W/L adjusted for weighted game state with 98% threshold.

The fact that the predictive power of the original Pythagorean W/L formula with the original scores is at least on par or better than the adjusted formulas really demonstrates the robustness of the formula, and that in some cases, the simplest formula is in fact the best one.

3.3. **Sabermetric Statistics Considered.** It seems most natural to consider sabermetric statistics of pitchers, as pitchers have arguably the greatest impact on the runs allowed, and in turn the runs scored of the opposing team. With this intuition, we proceeded to seek sabermetric statistics that measured the effectiveness of a pitcher. Three came to mind, namely (1) Walks plus Hits per Innings Pitched (WHIP), (2) Adjusted ERA+ (ERA+), and (3) Wins Above Replacement (WAR).

WHIP, given as (BB+H)/IP, where BB is walks, H is hits, and IP is innings pitched, is natural to consider as it is never negative, so there is no need to scale it, and it is readily available on many websites. In addition, it is relatively easy to evaluate a pitcher based on their WHIP; the lower the WHIP, the better the pitcher is usually. So, the way that we adjusted the runs in each game was to divide a team's runs scored in a game by the opposing pitcher's WHIP, since the higher the pitcher's WHIP, the worse the pitcher is, which means the score is inflated. In Figure 10, we compare the R-squared values of the Pythagorean W/L Formula (Regular) to that of the formula where runs are adjusted for WHIP (WHIP), and that of the formula where runs are adjusted for by WHIP and ballpark factors (WHIP.Ballpark) for the 2011 season. In addition, we vary the γ exponent, denoted by Pythag.Exp in the table.

In calculating WHIP.Ballpark, we increased the magnitude of the ballpark factor and decreased that of the WHIP, since it did not seem that WHIP played an important role in the predictive power of the Pythagorean W/L Formula, while we had showed earlier that ballpark factors could slightly increase the predictive power of the Pythagorean W/L Formula. The powers by which we increased and decreased the magnitudes of the respective statistics can be seen in the code, which we provide in Appendix B. As can be seen from the table, the regular form of the Pythagorean W/L Formula is still a much better predictor of a team's W/L percentage (this is true over all years). While WHIP may seem like an appropriate statistic to consider, as it basically evaluates a pitcher based on how effectively he keeps a batter off the bags, the low R-squared values may be due to the fact that pitchers have very little control over what happens after the ball is hit; more specifically, they do



```
Pythag.Exp Regular   WHIP WHIP.Ballpark
    0.5141  0.8809 0.5478        0.8717
    0.5261  0.8809 0.5479        0.8718
    0.5982  0.8810 0.5482        0.8718
    0.6125  0.8810 0.5483        0.8718
    0.6711  0.8810 0.5486        0.8718
    0.6762  0.8810 0.5486        0.8718
    0.7603  0.8810 0.5491        0.8719
    0.7619  0.8810 0.5492        0.8719
    0.9359  0.8811 0.5504        0.8721
    0.9677  0.8811 0.5506        0.8721
    0.9888  0.8811 0.5508        0.8721
    1.0356  0.8812 0.5512        0.8722
    1.0913  0.8812 0.5517        0.8722
    1.1205  0.8812 0.5520        0.8723
    1.1286  0.8812 0.5520        0.8723
    1.2046  0.8813 0.5528        0.8724
    1.5262  0.8815 0.5562        0.8728
    1.5414  0.8815 0.5564        0.8728
    1.7344  0.8817 0.5588        0.8731
    1.7481  0.8817 0.5590        0.8731
    1.7635  0.8817 0.5592        0.8732
    1.7683  0.8817 0.5593        0.8732
    1.7927  0.8818 0.5596        0.8732
    1.8737  0.8818 0.5607        0.8734
    2.1527  0.8821 0.5648        0.8739
```

FIGURE 10. Table of R-squared values of Pythagorean W/L Formula, Formula adjusted for WHIP, and for WHIP and ballpark over varying gammas for 2011.

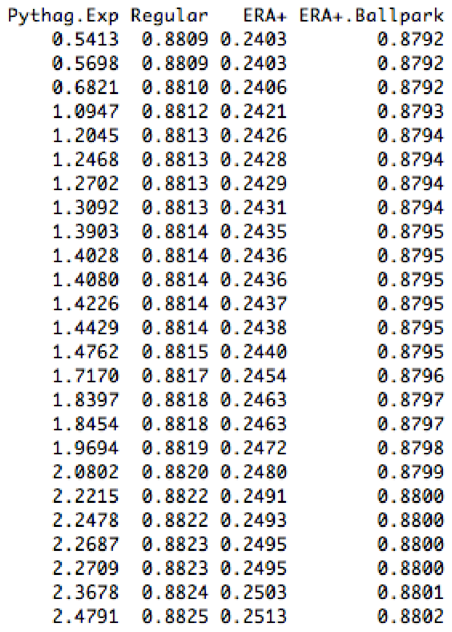
not control whether a ball hit in play actually goes for a hit, the rest of the team on the field does.

ERA+ adjusts a pitcher's earned run average (ERA) based on the ERA of the pitcher's league as well as the ballpark which the pitcher pitches in. It is calculated as

$$((\text{League ERA})*(\text{home ballpark factor})*100)/(\text{player's ERA}).$$

ERA+ is calculated so that the league average is 100, and is never negative. Similar to WHIP, this makes it easy to implement in terms of adjusting runs. The higher an ERA+ a pitcher has, the better the pitcher is considered. Thus, it makes sense to adjust the runs by multiplying by (ERA+)$*$(1/100), including a scaling factor of (1/100) in order to bring the league average to 1. We display the R-squared values of the Pythagorean W/L Formula (Regular), those of the formula where runs are adjusted for ERA+ (ERA+), and those of the formula where runs are adjusted for by ERA+ and ballpark factors (ERA+.Ballpark) for the 2011 season in Figure 11. In addition, we vary the γ exponent, denoted by Pythag.Exp in the table.

Similarly to the WHIP.Ballpark case, in calculating ERA+.Ballpark, we increased the magnitude of the ballpark factor and decreased that of the ERA+, since it did not seem that



```
Pythag.Exp Regular   ERA+ ERA+.Ballpark
    0.5413  0.8809 0.2403        0.8792
    0.5698  0.8809 0.2403        0.8792
    0.6821  0.8810 0.2406        0.8792
    1.0947  0.8812 0.2421        0.8793
    1.2045  0.8813 0.2426        0.8794
    1.2468  0.8813 0.2428        0.8794
    1.2702  0.8813 0.2429        0.8794
    1.3092  0.8813 0.2431        0.8794
    1.3903  0.8814 0.2435        0.8795
    1.4028  0.8814 0.2436        0.8795
    1.4080  0.8814 0.2436        0.8795
    1.4226  0.8814 0.2437        0.8795
    1.4429  0.8814 0.2438        0.8795
    1.4762  0.8815 0.2440        0.8795
    1.7170  0.8817 0.2454        0.8796
    1.8397  0.8818 0.2463        0.8797
    1.8454  0.8818 0.2463        0.8797
    1.9694  0.8819 0.2472        0.8798
    2.0802  0.8820 0.2480        0.8799
    2.2215  0.8822 0.2491        0.8800
    2.2478  0.8822 0.2493        0.8800
    2.2687  0.8823 0.2495        0.8800
    2.2709  0.8823 0.2495        0.8800
    2.3678  0.8824 0.2503        0.8801
    2.4791  0.8825 0.2513        0.8802
```

FIGURE 11. Table of R-squared values of Pythagorean W/L Formula, Formula adjusted for ERA+, and for ERA+ and ballpark over varying gammas for 2011.

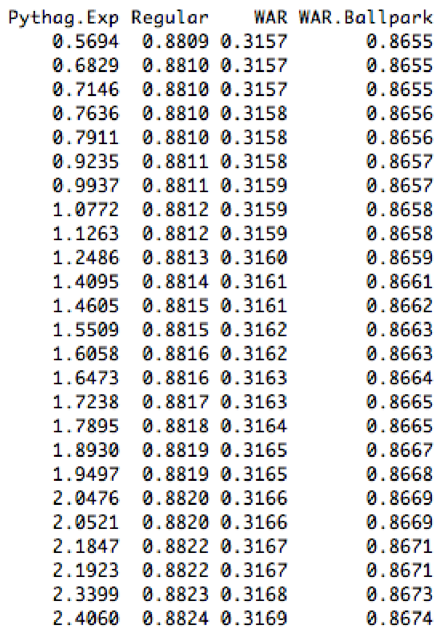
ERA+ played an important role in the predictive power of the Pythagorean W/L Formula, while we had showed earlier that ballpark factors could slightly increase the predictive power of the Pythagorean W/L Formula. The powers by which we increased and decreased the magnitudes of the respective statistics can be seen in the code, which is very similar to the code in Appendix B. As can be seen from the table, again the regular form of the Pythagorean W/L Formula is still a much better predictor of a team's W/L percentage than the formula with added in sabermetric statistics over all years. We note in the code that if a pitcher does not have an ERA+, then we give the pitcher an ERA+ of 100 and proceed with the calculations. The very low R-squared values produced from the adjusted formula with ERA+ could be due to the underlying inaccuracies of ERA, and namely what it means to be an earned run, which is an abstract and arbitrary term in and of itself (see `http://en.wikipedia.org/wiki/Earned_run`).

3.4. **Standardizing WAR.** We also decided to consider WAR, as a pitcher's WAR is directly related to their team's W/L percentage (see `http://www.baseball-reference.com/about/war_explained_pitch.shtml` on how `baseball-reference.`

`com` computes WAR). WAR is interpreted as the number of wins a player adds (or subtracts) compared to a replacement level player (a team of replacement level players is expected to average 40-50 wins in a 162 game season). However, in order to adjust runs with respect to WAR, we needed to scale WAR so that it can't be negative. This was done by first compiling the WAR for every pitcher in the league in a vector called NEW.WAR. We then got the the mean and standard deviation of NEW.WAR; call them new.war.mean and new.war.std. We then standardize the elements of NEW.WAR by the mean and std, namely subtracting new.war.mean and dividing by new.war.std for each element. Call this new vector of standardized values WAR.standard. We next find the minimum of WAR.standard, and call it WARstand.min. Finally, for each element of WAR.standard, we subtract WARstand.min so that the smallest element of the vector WAR.standard is 0, and then add 0.5 so that all the values will be positive.

The implementation of this standardization can be seen in the code in Appendix C. The way that we tested the effectiveness of WAR is the same as how we tested that of ERA+, without the scaling factor of 1/100, i.e. multiplying the runs scored of a team by the opposing pitcher's WAR. For the 2011 season, we show the R-squared values of the Pythagorean W/L Formula (Regular), those of the formula where runs are adjusted for WAR (WAR), and those of the formula where runs are adjusted for by WAR and ballpark factors (WAR.Ballpark) in Figure 12. In addition, we vary the γ exponent, denoted by Pythag.Exp in the table.

Similarly to the cases of WHIP.Ballpark and ERA+.Ballpark, in calculating WAR.Ballpark, we increased the magnitude of the ballpark factor and decreased that of the WAR, since it did not seem that WAR played an important role in the predictive power of the Pythagorean W/L Formula, while we had showed earlier that ballpark factors could slightly increase the predictive power of the Pythagorean W/L Formula. The powers by which we increased and decreased the magnitudes of the respective statistics can be seen in the code, which we provide in Appendix C. As can be seen from the table, again the regular form of the Pythagorean W/L Formula is still a much better predictor of a team's W/L percentage than the formula with added in sabermetric statistics over all years. The low R-squared values produced from the adjusted formula with WAR can probably be attributed to the way that we standardized WAR; unfortunately, there is not a good, clean way of standardizing WAR. So, while it may seem that using WAR in adjusted runs is possibly the right statistic to consider, the lack of a true standardization hurts this realization. Another standardization method we tried was first standardizing WAR (so that the vector of values had mean

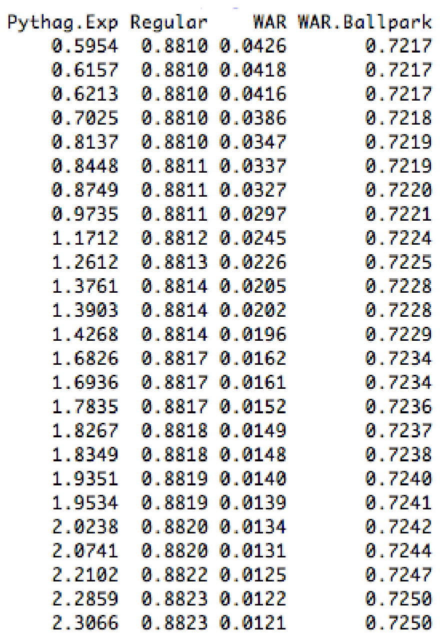


```
Pythag.Exp Regular    WAR WAR.Ballpark
    0.5694  0.8809 0.3157       0.8655
    0.6829  0.8810 0.3157       0.8655
    0.7146  0.8810 0.3157       0.8655
    0.7636  0.8810 0.3158       0.8656
    0.7911  0.8810 0.3158       0.8656
    0.9235  0.8811 0.3158       0.8657
    0.9937  0.8811 0.3159       0.8657
    1.0772  0.8812 0.3159       0.8658
    1.1263  0.8812 0.3159       0.8658
    1.2486  0.8813 0.3160       0.8659
    1.4095  0.8814 0.3161       0.8661
    1.4605  0.8815 0.3161       0.8662
    1.5509  0.8815 0.3162       0.8663
    1.6058  0.8816 0.3162       0.8663
    1.6473  0.8816 0.3163       0.8664
    1.7238  0.8817 0.3163       0.8665
    1.7895  0.8818 0.3164       0.8665
    1.8930  0.8819 0.3165       0.8667
    1.9497  0.8819 0.3165       0.8668
    2.0476  0.8820 0.3166       0.8669
    2.0521  0.8820 0.3166       0.8669
    2.1847  0.8822 0.3167       0.8671
    2.1923  0.8822 0.3167       0.8671
    2.3399  0.8823 0.3168       0.8673
    2.4060  0.8824 0.3169       0.8674
```

FIGURE 12. Table of R-squared values of Pythagorean W/L Formula, Formula adjusted for WAR, and for WAR and ballpark over varying gammas for 2011.

0), then taking e to the values (so that all the values are positive), and finally dividing by the mean of the new exponentiated values in order to have the vector centered around 1. However, we obtain R^2 values (displayed in Figure 13) worse than the ones we obtained in Figure 12; again, the lack of an accepted standardization most likely hurts the realization.

Throughout the different statistics considered, the robustness of the regular form of the formula is truly startling; it is amazing that a formula devised in the 1970s can consistently beat out formulas with added in whistles and bells that intuitively should make the formula more accurate. With more time, it would have been interesting to consider more complicated pitcher statistics that were more defense-independent, such as Fielding Independent Pitching (FIP) or tRA (an extension of FIP).



```
Pythag.Exp Regular    WAR WAR.Ballpark
    0.5954  0.8810 0.0426       0.7217
    0.6157  0.8810 0.0418       0.7217
    0.6213  0.8810 0.0416       0.7217
    0.7025  0.8810 0.0386       0.7218
    0.8137  0.8810 0.0347       0.7219
    0.8448  0.8811 0.0337       0.7219
    0.8749  0.8811 0.0327       0.7220
    0.9735  0.8811 0.0297       0.7221
    1.1712  0.8812 0.0245       0.7224
    1.2612  0.8813 0.0226       0.7225
    1.3761  0.8814 0.0205       0.7228
    1.3903  0.8814 0.0202       0.7228
    1.4268  0.8814 0.0196       0.7229
    1.6826  0.8817 0.0162       0.7234
    1.6936  0.8817 0.0161       0.7234
    1.7835  0.8817 0.0152       0.7236
    1.8267  0.8818 0.0149       0.7237
    1.8349  0.8818 0.0148       0.7238
    1.9351  0.8819 0.0140       0.7240
    1.9534  0.8819 0.0139       0.7241
    2.0238  0.8820 0.0134       0.7242
    2.0741  0.8820 0.0131       0.7244
    2.2102  0.8822 0.0125       0.7247
    2.2859  0.8823 0.0122       0.7250
    2.3066  0.8823 0.0121       0.7250
```

FIGURE 13. Table of R-squared values of Pythagorean W/L Formula, Formula adjusted for exponentiated WAR, and for exponentiated WAR and ballpark over varying gammas for 2011.

4. Linear Combinations of Weibulls

We now state and prove our main result for a linear combination of two Weibulls, and leave the straightforward generalization to combinations of more Weibulls to the reader. The reason such an expansion is advantageous and natural is that, following [MIL], we can integrate pairs of Weibulls in the regions needed and obtain simple closed form expressions. The theorem below also holds if $\gamma < 0$; however, in that situation the more your runs scored exceeds your runs allowed, the worse your predicted record due to the different shape of the Weibull (in all applications of Weibulls in survival analysis, the shape parameter γ must be positive).

4.1. **Linear Combination of Weibulls.**

Theorem 4.1. *Let the runs scored and allowed per game be two independent random variables drawn from linear combinations of independent Weibull distributions with the same β's and γ's. Specifically, if $W(t;\alpha,\beta,\gamma)$ represents a Weibull distribution with parameters (α,β,γ), and we choose non-negative weights*[1] $0 \le c_i, c_j' \le 1$ *(so $c_1 + c_2 = 1$ and $c_1' + c_2' = 1$), then the density of runs scored, X is*

$$f(x;\alpha_{\mathrm{RS}_1},\alpha_{\mathrm{RS}_2},\beta,\gamma,c_1,c_2) \;=\; c_1 W(x;\alpha_{\mathrm{RS}_1},\beta,\gamma) + c_2 W(\alpha_{\mathrm{RS}_2},\beta,\gamma) \tag{4.1}$$

and runs allowed, Y, is

$$f(y;\alpha_{\mathrm{RA}_1},\alpha_{\mathrm{RA}_2},\beta,\gamma,c_1',c_2') \;=\; c_1' W(y;\alpha_{\mathrm{RA}_1},\beta,\gamma) + c_2' W(\alpha_{\mathrm{RA}_2},\beta,\gamma). \tag{4.2}$$

In addition, we choose α_{RS_1} and α_{RS_2} so that the mean of X is $\mathrm{RS}_{\mathrm{obs}}$ and choose α_{RA_1} and α_{RA_2} such that the mean of Y is $\mathrm{RA}_{\mathrm{obs}}$. For $\gamma > 0$, we have

$$\begin{aligned}
&\mathrm{Won-Loss\ Percentage}(\alpha_{\mathrm{RS}_1},\alpha_{\mathrm{RS}_2},\alpha_{\mathrm{RA}_1},\alpha_{\mathrm{RA}_2},\beta,\gamma,c_1,c_2,c_1',c_2')\\
&= c_1c_1'\frac{\alpha_{\mathrm{RS}_1}^{\gamma}}{\alpha_{\mathrm{RS}_1}^{\gamma}+\alpha_{\mathrm{RA}_1}^{\gamma}} + c_1c_2'\frac{\alpha_{\mathrm{RS}_1}^{\gamma}}{\alpha_{\mathrm{RS}_1}^{\gamma}+\alpha_{\mathrm{RA}_2}^{\gamma}}\\
&\quad + c_2c_1'\frac{\alpha_{\mathrm{RS}_2}^{\gamma}}{\alpha_{\mathrm{RS}_2}^{\gamma}+\alpha_{\mathrm{RA}_1}^{\gamma}} + c_2c_2'\frac{\alpha_{\mathrm{RS}_2}^{\gamma}}{\alpha_{\mathrm{RS}_2}^{\gamma}+\alpha_{\mathrm{RA}_2}^{\gamma}}\\
&= \sum_{i=1}^{2}\sum_{j=1}^{2} c_ic_j'\frac{\alpha_{\mathrm{RS}_i}^{\gamma}}{\alpha_{\mathrm{RS}_i}^{\gamma}+\alpha_{\mathrm{RA}_j}^{\gamma}}.
\end{aligned} \tag{4.3}$$

[1] If we had more terms in the linear combination, we would simply choose non-negative weights summing to 1.

Proof. Since the means of X (runs scored) and Y (runs allowed) are $\mathrm{RS_{obs}}$ and $\mathrm{RA_{obs}}$, respectively, and the random variables are drawn from linear combinations of independent Weibulls, by Lemma 2.1, we have that

$$\begin{aligned}\mathrm{RS_{obs}} &= c_1(\alpha_{\mathrm{RS}_1}\Gamma(1+\gamma^{-1})+\beta)+(1-c_1)(\alpha_{\mathrm{RS}_2}\Gamma(1+\gamma^{-1})+\beta)\\ \mathrm{RA_{obs}} &= c_1(\alpha_{\mathrm{RA}_1}\Gamma(1+\gamma^{-1})+\beta)+(1-c_1)(\alpha_{\mathrm{RA}_2}\Gamma(1+\gamma^{-1})+\beta). \qquad (4.4)\end{aligned}$$

We now calculate the probability that X exceeds Y. We constantly use the fact that the integral of a probability density is 1. We need the two β's and the two γ's to be equal in order to obtain closed form expressions.[2] We find

$$\begin{aligned}\mathrm{Prob}(X>Y) &= \int_{x=\beta}^{\infty}\int_{y=\beta}^{x} f(x;\alpha_{\mathrm{RS}_1},\alpha_{\mathrm{RS}_2},\beta,\gamma,c_1,c_2)f(y;\alpha_{\mathrm{RA}_1},\alpha_{\mathrm{RA}_2},\beta,\gamma,c_1',c_2')dydx\\
&= \sum_{i=1}^{2}\sum_{j=1}^{2}\int_{x=0}^{\infty}\int_{y=0}^{x} c_ic_j'\frac{\gamma}{\alpha_{\mathrm{RS}_i}}\left(\frac{x}{\alpha_{\mathrm{RS}_i}}\right)^{\gamma-1} e^{-(\frac{x}{\alpha_{\mathrm{RS}_i}})^\gamma}\frac{\gamma}{\alpha_{\mathrm{RA}_j}}\left(\frac{x}{\alpha_{\mathrm{RA}_j}}\right)^{\gamma-1} e^{-(\frac{x}{\alpha_{\mathrm{RA}_j}})^\gamma}dydx\\
&= \sum_{i=1}^{2}\sum_{j=1}^{2}c_ic_j'\int_{x=0}^{\infty}\frac{\gamma}{\alpha_{\mathrm{RS}_i}}\left(\frac{x}{\alpha_{\mathrm{RS}_i}}\right)^{\gamma-1} e^{-(\frac{x}{\alpha_{\mathrm{RS}_i}})^\gamma}\left[\int_{y=0}^{x}\frac{\gamma}{\alpha_{\mathrm{RA}_j}}\left(\frac{y}{\alpha_{\mathrm{RA}_j}}\right)^{\gamma-1} e^{-(\frac{y}{\alpha_{\mathrm{RA}_j}})^\gamma}dy\right]dx\\
&= \sum_{i=1}^{2}\sum_{j=1}^{2}c_ic_j'\int_{x=0}^{\infty}\frac{\gamma}{\alpha_{\mathrm{RS}_i}}\left(\frac{x}{\alpha_{\mathrm{RS}_i}}\right)^{\gamma-1} e^{-(\frac{x}{\alpha_{\mathrm{RS}_i}})^\gamma} * \left[1-e^{-(\frac{x}{\alpha_{\mathrm{RA}_j}})^\gamma}\right]dx\\
&= \sum_{i=1}^{2}\sum_{j=1}^{2}c_ic_j'\left[1-\int_{x=0}^{\infty}\frac{\gamma}{\alpha_{\mathrm{RS}_i}}\left(\frac{x}{\alpha_{\mathrm{RS}_i}}\right)^{\gamma-1} e^{-(\frac{x}{\alpha_{\mathrm{RS}_i}})^\gamma-(\frac{x}{\alpha_{\mathrm{RA}_j}})^\gamma}dx\right]. \qquad (4.5)\end{aligned}$$

We set

$$\frac{1}{\alpha_{ij}^\gamma} = \frac{1}{\alpha_{\mathrm{RS}_i}^\gamma}+\frac{1}{\alpha_{\mathrm{RA}_j}^\gamma} = \frac{\alpha_{\mathrm{RS}_i}^\gamma+\alpha_{\mathrm{RA}_j}^\gamma}{\alpha_{\mathrm{RS}_i}^\gamma\alpha_{\mathrm{RA}_j}^\gamma}$$

[2]If the β's are differen then in the integration below we might have issues with the bounds of integration, while if the γ's are unequal we get incomplete Gamma functions, though for certain rational ratios of the γ's these can be done in closed form.

for $1 \leq i, j \leq 2$, so that (4.5) becomes

$$\begin{aligned}
&\sum_{i=1}^{2}\sum_{j=1}^{2} c_i c'_j \left[1 - \int_{x=0}^{\infty} \frac{\gamma}{\alpha_{\mathrm{RS}_i}} \left(\frac{x}{\alpha_{\mathrm{RS}_i}}\right)^{\gamma-1} e^{-(\frac{x}{\alpha_{ij}})^{\gamma}} dx\right] \\
&= \sum_{i=1}^{2}\sum_{j=1}^{2} c_i c'_j \left[1 - \frac{\alpha_{ij}^{\gamma}}{\alpha_{\mathrm{RS}_i}^{\gamma}} \int_{x=0}^{\infty} \frac{\gamma}{\alpha_{ij}} \left(\frac{x}{\alpha_{ij}}\right)^{\gamma-1} e^{-(\frac{x}{\alpha_{ij}})^{\gamma}} dx\right] \\
&= \sum_{i=1}^{2}\sum_{j=1}^{2} c_i c'_j \left[1 - \frac{\alpha_{ij}^{\gamma}}{\alpha_{\mathrm{RS}_i}^{\gamma}}\right] \\
&= \sum_{i=1}^{2}\sum_{j=1}^{2} c_i c'_j \left[1 - \frac{1}{\alpha_{\mathrm{RS}_i}^{\gamma}} * \frac{\alpha_{\mathrm{RS}_i}^{\gamma}\alpha_{\mathrm{RA}_j}^{\gamma}}{\alpha_{\mathrm{RS}_i}^{\gamma} + \alpha_{\mathrm{RA}_j}^{\gamma}}\right] \\
&= \sum_{i=1}^{2}\sum_{j=1}^{2} c_i c'_j \left[\frac{\alpha_{\mathrm{RS}_i}^{\gamma}}{\alpha_{\mathrm{RS}_i}^{\gamma} + \alpha_{\mathrm{RA}_j}^{\gamma}}\right] \\
&= c_1 c'_1 \frac{\alpha_{\mathrm{RS}_1}^{\gamma}}{\alpha_{\mathrm{RS}_1}^{\gamma} + \alpha_{\mathrm{RA}_1}^{\gamma}} + c_1 c'_2 \frac{\alpha_{\mathrm{RS}_1}^{\gamma}}{\alpha_{\mathrm{RS}_1}^{\gamma} + \alpha_{\mathrm{RA}_2}^{\gamma}} \\
&\quad + c_2 c'_1 \frac{\alpha_{\mathrm{RS}_2}^{\gamma}}{\alpha_{\mathrm{RS}_2}^{\gamma} + \alpha_{\mathrm{RA}_1}^{\gamma}} + c_2 c'_2 \frac{\alpha_{\mathrm{RS}_2}^{\gamma}}{\alpha_{\mathrm{RS}_2}^{\gamma} + \alpha_{\mathrm{RA}_2}^{\gamma}},
\end{aligned} \tag{4.6}$$

completing the proof of Theorem 4.1. □

Observation 4.2. *We set γ to be the same for both distributions in the linear combination of Weibulls of runs scored and allowed as it allows for a closed form solution (one less parameter to worry about). It also really doesn't make sense for there to be two different gammas over the entire league. In addition, we have $\gamma > 0$. Consider the simple case where $c_1 = 1$, we are left with $\alpha_{\mathrm{RS}_1}^{\gamma}/(\alpha_{\mathrm{RS}_1}^{\gamma} + \alpha_{\mathrm{RA}_1}^{\gamma})$, which simplifies to $(RS-\beta)^{\gamma}/((RS-\beta)^{\gamma} + (RA-\beta)^{\gamma})$ (see* [MIL]*). Let $\beta = 0$ and $\gamma = -1/2 < 0$. If a teams scores 36 runs (RS=36) and allows 20 runs (RA=20), they are expected to have a winning percentage. However, with $\gamma < 0$, we obtain a winning percentage of $36^{-1/2}/(36^{-1/2} + 20^{-1/2}) = 0.427 < 0.5$, which intuitively doesn't make sense. So, we only consider $\gamma > 0$.*

4.2. **Moments Analysis.**

The first attempt at estimating values for the parameters was by utilizing the moments. We looked at the moments of the distribution that runs scored was drawn from, namely

$$c_1 \frac{\gamma}{\alpha_{\mathrm{RS}_1}} \left(\frac{x-\beta_1}{\alpha_{\mathrm{RS}_1}}\right)^{\gamma-1} e^{-(\frac{x-\beta_1}{\alpha_{\mathrm{RS}_1}})^{\gamma}} + (1-c_1)\frac{\gamma}{\alpha_{\mathrm{RS}_2}} \left(\frac{x-\beta_2}{\alpha_{\mathrm{RS}_2}}\right)^{\gamma-1} e^{-(\frac{x-\beta_2}{\alpha_{\mathrm{RS}_2}})^{\gamma}}.$$

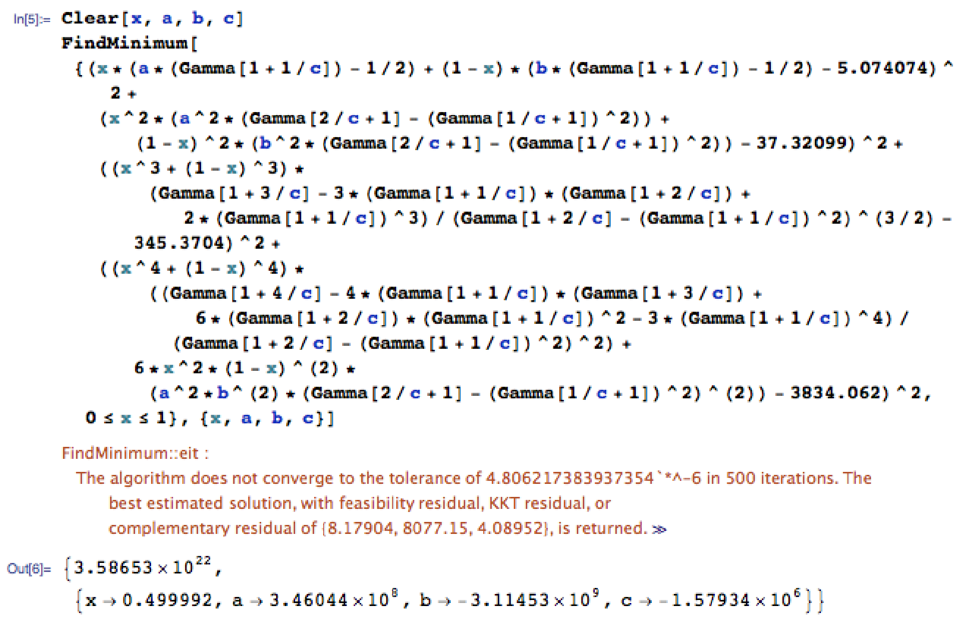
From [MUR], and using the fact that the two Weibulls in the linear combination are independent, we obtained the following moments:

$$\begin{aligned}
\text{First Moment} &= c_1(\alpha_{\mathrm{RS}_1}\Gamma(1+\gamma^{-1})+\beta_1)+(1-c_1)(\alpha_{\mathrm{RS}_2}\Gamma(1+\gamma^{-1})+\beta_2)\\
\text{Second Moment} &= c_1^2\left[\alpha_{\mathrm{RS}_1}^2\left(\Gamma(\frac{2}{\gamma}+1)-(\Gamma(\frac{1}{\gamma}+1))^2\right)\right]\\
&\quad+(1-c_1)^2\left[\alpha_{\mathrm{RS}_2}^2\left(\Gamma(\frac{2}{\gamma}+1)-(\Gamma(\frac{1}{\gamma}+1))^2\right)\right]\\
\text{Third Moment} &= (c_1^3+(1-c_1)^3)*\frac{g_3-3g_1g_2+2g_1^3}{(g_2-g_1^2)^{3/2}}\\
\text{Fourth Moment} &= (c_1^4+(1-c_1)^4)*\frac{g_4-4g_1g_3+6g_2g_1^2-3g_1^4}{(g_2-g_1^2)^2}\\
&\quad+6*c_1^2(1-c_1)^2*\left[\alpha_{RS_1}^2\alpha_{RS_2}^2\left(g_2-g_1^2\right)^2\right]
\end{aligned}$$

where $g_i=\Gamma(1+\frac{i}{\gamma})$, and $\Gamma(x)$ is the Gamma function defined as $\Gamma(x)=\int_0^\infty e^{-u}u^{x-1}du$. We next used R to find the observed moments for teams' runs scored from 2007. The code can be seen in Appendix D. In order to find estimates for the parameters, we wanted to minimize the sum of the squares of the difference in observed and expected values. To implement this, we used Mathematica; the code is displayed in Figure 14. In the code, x represents c_1, a represents α_{RS_1}, b represents α_{RS_2}, c represents γ, and we assume that $\beta_1=-1/2=\beta_2$.

The code in Figure 14 uses the observed moments of a team and attempts to minimize the sum of the squares of the difference in observed and expected moments using FindMinimum. As can be seen from the output (and error message), Mathematica was unable to find a reasonable minimum, and the parameter estimates do not seem feasible in the slightest. We also tried setting starting values for the parameters, but this did not help. This deterred us from utilizing moments analysis, and we instead turned our attention to least squares.

4.3. **Least Squares.** We looked at the 30 teams of the entire league from the 2004 to 2012 season. We display results from the 2011, but the results from any other season are readily available. We implemented the method of least squares using the bins in (2.2). The method of least squares involved minimizing the sum of squares of the error of the runs scored data plus the sum of squares of the error of the runs allowed data; using the bins as in (2.2), there were seven free parameters, namely α_{RS_1}, α_{RS_2}, α_{RA_1}, α_{RA_2}, γ, c_1, and c_1'. Letting $\mathrm{Bin}(k)$ be the kth bin of 2.2, $RS_{obs}(k)$ and $RA_{obs}(k)$ represent the observed number of games with



```
In[5]:= Clear[x, a, b, c]
        FindMinimum[
         {(x * (a * (Gamma[1 + 1 / c]) - 1 / 2) + (1 - x) * (b * (Gamma[1 + 1 / c]) - 1 / 2) - 5.074074) ^
             2 +
            (x ^ 2 * (a ^ 2 * (Gamma[2 / c + 1] - (Gamma[1 / c + 1]) ^ 2)) +
                (1 - x) ^ 2 * (b ^ 2 * (Gamma[2 / c + 1] - (Gamma[1 / c + 1]) ^ 2)) - 37.32099) ^ 2 +
            ((x ^ 3 + (1 - x) ^ 3) *
                  (Gamma[1 + 3 / c] - 3 * (Gamma[1 + 1 / c]) * (Gamma[1 + 2 / c]) +
                      2 * (Gamma[1 + 1 / c]) ^ 3) / (Gamma[1 + 2 / c] - (Gamma[1 + 1 / c]) ^ 2) ^ (3 / 2) -
                345.3704) ^ 2 +
            ((x ^ 4 + (1 - x) ^ 4) *
                  ((Gamma[1 + 4 / c] - 4 * (Gamma[1 + 1 / c]) * (Gamma[1 + 3 / c]) +
                       6 * (Gamma[1 + 2 / c]) * (Gamma[1 + 1 / c]) ^ 2 - 3 * (Gamma[1 + 1 / c]) ^ 4) /
                     (Gamma[1 + 2 / c] - (Gamma[1 + 1 / c]) ^ 2) ^ 2) +
                6 * x ^ 2 * (1 - x) ^ (2) *
                  (a ^ 2 * b ^ (2) * (Gamma[2 / c + 1] - (Gamma[1 / c + 1]) ^ 2) ^ (2)) - 3834.062) ^ 2,
          0 ≤ x ≤ 1}, {x, a, b, c}]

        FindMinimum::eit :
         The algorithm does not converge to the tolerance of 4.806217383937354`*^-6 in 500 iterations. The
             best estimated solution, with feasibility residual, KKT residual, or
             complementary residual of {8.17904, 8077.15, 4.08952}, is returned. >>

Out[6]= {3.58653 × 10^22,
         {x → 0.499992, a → 3.46044 × 10^8, b → -3.11453 × 10^9, c → -1.57934 × 10^6}}
```

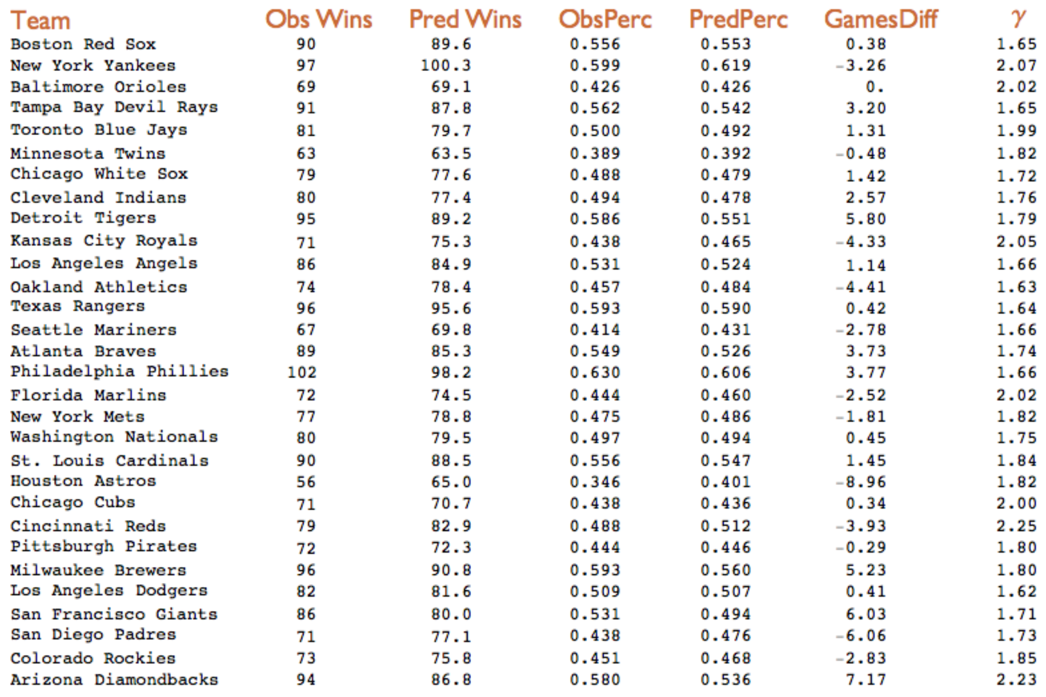
FIGURE 14. Mathematica code used in moments analysis.

number of runs scored and allowed in Bin(k), and $A(\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma, c_1, k)$ denote the area under the Weibull with parameters $(\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma, c_1)$ in Bin(k), then for each team we found the values of $(\alpha_{\mathrm{RS}_1}, \alpha_{\mathrm{RS}_2}, \alpha_{\mathrm{RA}_1}, \alpha_{\mathrm{RA}_2}, \gamma, c_1, c_1')$ that minimized

$$\sum_{k=1}^{\text{Num. Bins}} (RS_{obs}(k) - \#\text{Games} * A(\alpha_{\mathrm{RS}_1}, \alpha_{\mathrm{RS}_2}, -.5, -.5, \gamma, c_1, k))^2 + \sum_{k=1}^{\text{Num. Bins}} (RA_{obs}(k) - \#\text{Games} * A(\alpha_{\mathrm{RA}_1}, \alpha_{\mathrm{RA}_2}, -.5, -.5, \gamma, c_1', k))^2. \quad (4.7)$$

For each team, we found the best fit Weibulls with forms $(\alpha_{\mathrm{RS}_1}, \alpha_{\mathrm{RS}_2}, -.5, -.5, \gamma, c_1)$ and $(\alpha_{\mathrm{RA}_1}, \alpha_{\mathrm{RA}_2}, -.5, -.5, \gamma, c_1')$. In Figure 15, we compared the predicted wins, losses, and won-loss percentage with the observed ones.

The code used can be seen in Appendix E. Using the method of least squares, the mean γ over all 30 teams is 1.83 with a standard deviation of 0.18 (the median is 1.79). We can see that the exponent 1.83, considered as the best exponent, is within the region of one standard deviation from the mean γ. Considering the absolute value of the difference between observed and predicted wins, we have a mean of 2.89 with a standard deviation of 2.34 (median is 2.68). Without considering the absolute value, the mean is 0.104 with a



Team	Obs Wins	Pred Wins	ObsPerc	PredPerc	GamesDiff	γ
Boston Red Sox	90	89.6	0.556	0.553	0.38	1.65
New York Yankees	97	100.3	0.599	0.619	-3.26	2.07
Baltimore Orioles	69	69.1	0.426	0.426	0.	2.02
Tampa Bay Devil Rays	91	87.8	0.562	0.542	3.20	1.65
Toronto Blue Jays	81	79.7	0.500	0.492	1.31	1.99
Minnesota Twins	63	63.5	0.389	0.392	-0.48	1.82
Chicago White Sox	79	77.6	0.488	0.479	1.42	1.72
Cleveland Indians	80	77.4	0.494	0.478	2.57	1.76
Detroit Tigers	95	89.2	0.586	0.551	5.80	1.79
Kansas City Royals	71	75.3	0.438	0.465	-4.33	2.05
Los Angeles Angels	86	84.9	0.531	0.524	1.14	1.66
Oakland Athletics	74	78.4	0.457	0.484	-4.41	1.63
Texas Rangers	96	95.6	0.593	0.590	0.42	1.64
Seattle Mariners	67	69.8	0.414	0.431	-2.78	1.66
Atlanta Braves	89	85.3	0.549	0.526	3.73	1.74
Philadelphia Phillies	102	98.2	0.630	0.606	3.77	1.66
Florida Marlins	72	74.5	0.444	0.460	-2.52	2.02
New York Mets	77	78.8	0.475	0.486	-1.81	1.82
Washington Nationals	80	79.5	0.497	0.494	0.45	1.75
St. Louis Cardinals	90	88.5	0.556	0.547	1.45	1.84
Houston Astros	56	65.0	0.346	0.401	-8.96	1.82
Chicago Cubs	71	70.7	0.438	0.436	0.34	2.00
Cincinnati Reds	79	82.9	0.488	0.512	-3.93	2.25
Pittsburgh Pirates	72	72.3	0.444	0.446	-0.29	1.80
Milwaukee Brewers	96	90.8	0.593	0.560	5.23	1.80
Los Angeles Dodgers	82	81.6	0.509	0.507	0.41	1.62
San Francisco Giants	86	80.0	0.531	0.494	6.03	1.71
San Diego Padres	71	77.1	0.438	0.476	-6.06	1.73
Colorado Rockies	73	75.8	0.451	0.468	-2.83	1.85
Arizona Diamondbacks	94	86.8	0.580	0.536	7.17	2.23

FIGURE 15. Results for the 2011 season using Method of Least Squares.

standard deviation of 3.75 (and a median of 0.39). We will only concern ourselves with the absolute value of the difference, as this really tells how accurate our predicted values are.

These values are improvements on those obtained when using a single Weibull distribution to predict runs, which produces a mean number of games off of 4.43 with standard deviation 3.23 (and median 3.54) in the absolute value case. We display the results over seasons from 2004 to 2012 in Figure 16. It is apparent that the linear combination of Weibulls better estimates teams' win/loss percentage; in fact, it is over one game better at estimating than the single Weibull! The mean number of games off for a single Weibull from 2004 to 2012 was 4.22 (with a standard deviation of 3.03), while that of the linear combination of Weibulls was 3.11 (with a standard deviation of 2.33). In addition, there is less standard deviation in the estimates. Thus, it appears that the linear combination of Weibulls provides a much tighter, better estimate than the single Weibull does. To further demonstrate how accurate the quality of the fit is, we compare the best fit linear combination of Weibulls of runs scored and allowed with those observed of the 2011 Seattle Mariners in Figure 18; we can see that the fit is visually very good.

We conduct an independent two-sample t-test with unequal variances in R using the t.test command to see if the difference between the games off determined by the single Weibull

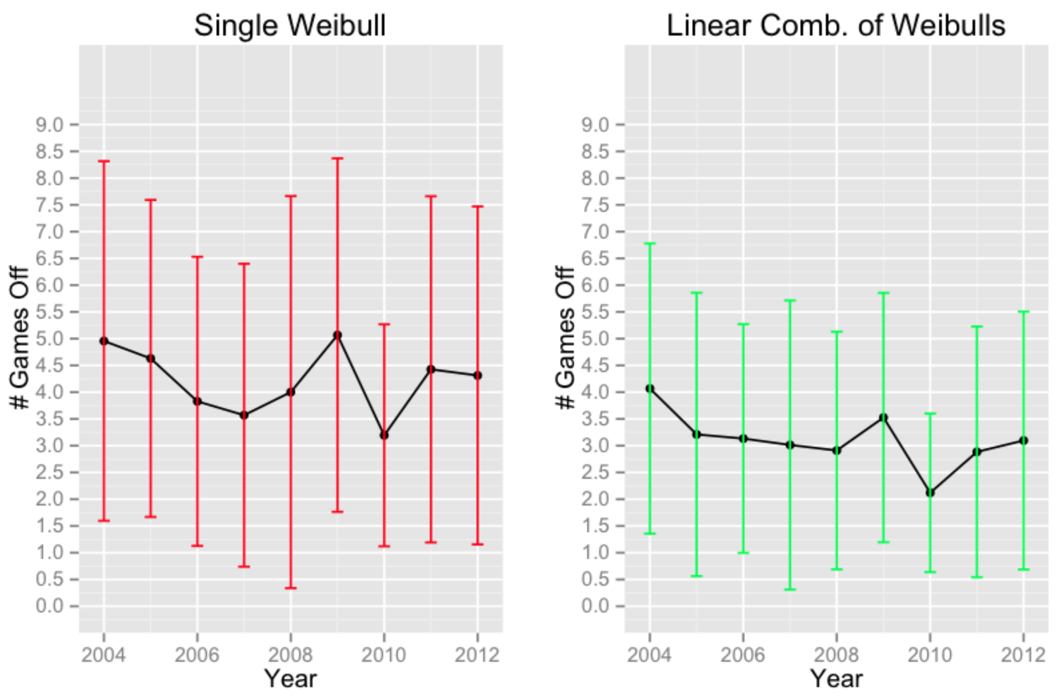


FIGURE 16. Mean number of games off (with standard deviation) for single Weibull and linear combination of Weibulls from 2004-2012.

and those by linear combinations of Weibulls is statistically significant in Figure 17. With a p-value less than 0.01 and a 95% confidence interval that does not contain 0, we can see that the difference is in fact statistically significant.

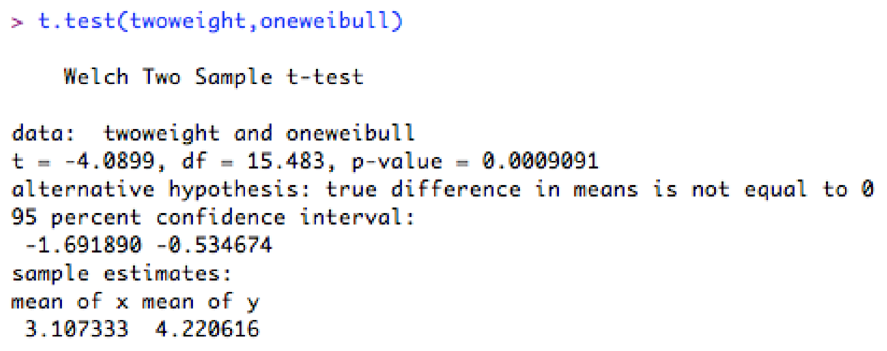
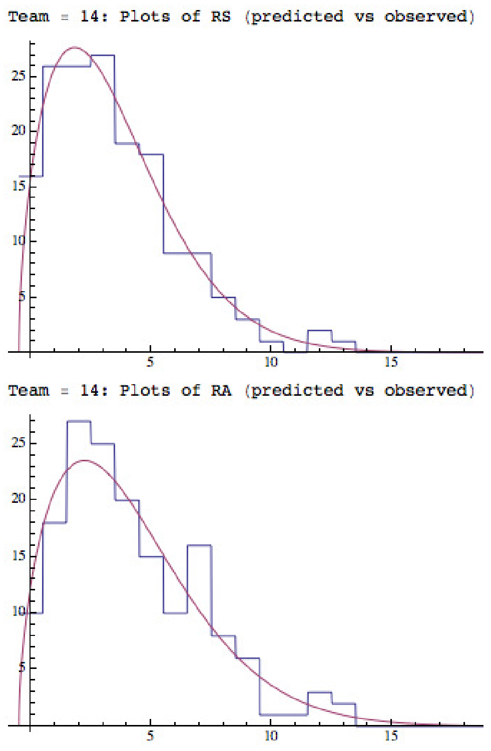
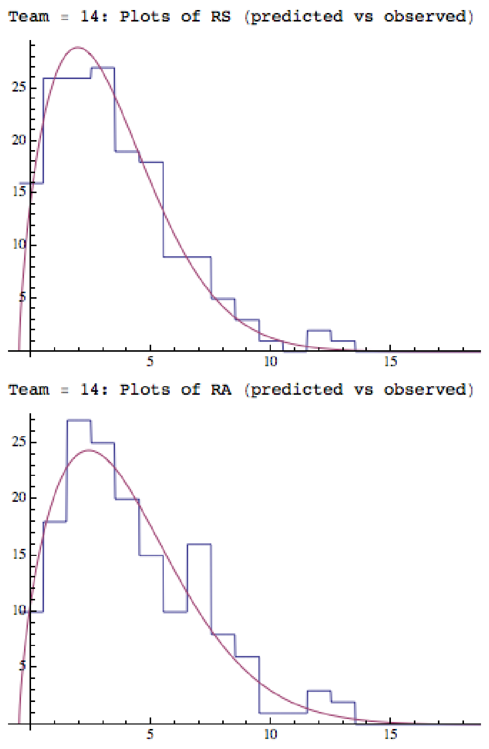

```
> t.test(twoweight,oneweibull)

	Welch Two Sample t-test

data:  twoweight and oneweibull
t = -4.0899, df = 15.483, p-value = 0.0009091
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.691890 -0.534674
sample estimates:
mean of x mean of y
 3.107333  4.220616
```

FIGURE 17. t-test to determine whether the difference between the games off determined by the single Weibull and those by linear combinations of Weibulls is statistically significant.



(A) Single Weibull mapping runs scored and allowed.



(B) Linear Combination of Weibulls mapping runs scored and allowed.

FIGURE 18. Comparison of best fit linear combination of Weibulls versus single Weibull for runs scored (top) and allowed (bottom) for the 2011 Seattle Mariners against the observed distribution of scores.

In addition, we compared the mean number of games off of ESPN's calculated Expected WL (ExWL) and those of the linear combination of Weibulls from 1979 to 2013. For the years from 2002 to 2013, we utilize ESPN's ExWL, which calculates a team's expected WL as $(\text{runs scored}^2)/(\text{runs scored}^2 + \text{runs allowed}^2)$.[3] Since ESPN only had this data available down to the year 2002, we used `baseball-reference.com`'s Pythagorean WL (pythagWL) in order to obtain more data (`baseball-reference.com` allowed us to go all the way down to 1979). The pythagWL statistic is calculated as $(\text{runs scored}^{1.83})/(\text{runs scored}^{1.83} + \text{runs allowed}^{1.83})$.[4]

[3]see bottom of page: `http://espn.go.com/mlb/stats/rpi/_/year/2011`.

[4]see section "What is Pythagorean Winning Percentage?": `http://www.sports-reference.com/blog/baseball-reference-faqs/`.

We display the results in Figure 19. The mean number of games off for ESPN's ExWL was 3.09 with a mean standard deviation of 2.29, numbers slightly worse than those of the linear combination of Weibulls (mean of 3.03 with standard deviation of 2.21). So, we can see that the linear combination of Weibulls is doing, on average, about .06 of a game better than ESPN's ExWL. We performed an independent two-sample t-test with unequal variances in R using the t.test command to see if the difference between the games off determined by ESPN's ExWL and the linear combination of Weibulls is statistically significant in Figure 20. With a very large p-value, we fail to reject the null hypothesis, suggesting that the difference in mean number of games off is not in fact statistically significant. We also display a plot (Figure 21) that models the difference in the number of games between ESPN's ExWL and the linear combination of Weibulls; it seems to be a constant positive value for the most part, suggesting that the linear combination of Weibulls is doing slightly better than ESPN's ExWL.

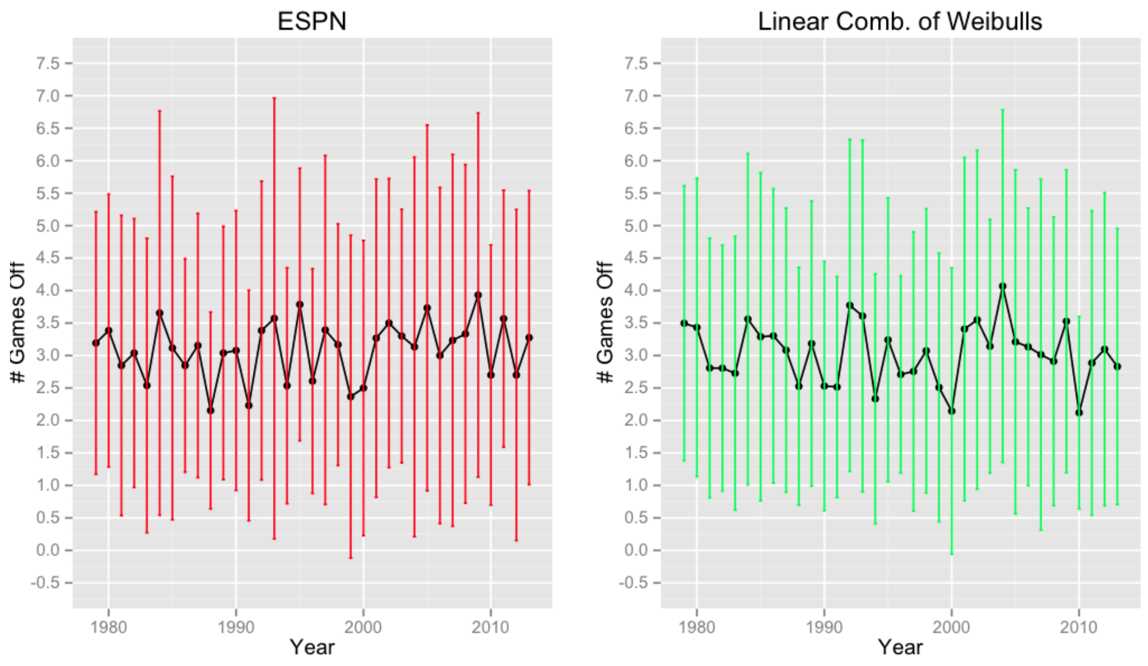
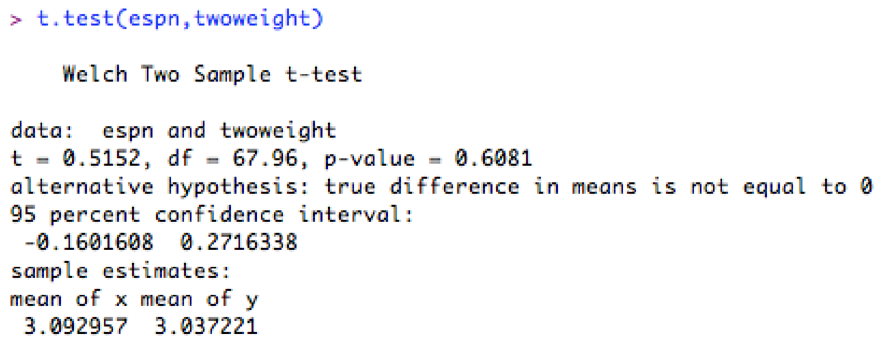


FIGURE 19. Mean number of games off (with standard deviation) for ESPN ExWL and linear combination of Weibulls from 1979-2013.

Looking at Figure 21 more closely, we can see that there are blocks of the graph where ESPN does better, and blocks where linear combination of Weibulls does better. For example, it is clear in the era from 1979-1989 that ESPN's ExWL is the more accurate statistic, beating the linear combination of Weibulls in 7 out of the 11 years. However, from 1990 to 2013, the linear combination of Weibulls beats ESPN's ExWL 14 out of the 24 years, and by around 0.3/0.35 games in those years. When ESPN's ExWL does beat the linear combination of Weibulls in the years from 1990 to 2013, it does so by around 0.25 games, including the point at 2004, which seems very out of the ordinary; without this point, ESPN's ExWL wins by about .2 games in the years between 1990 and 2013 that it does beat the linear combination of Weibulls. Thus, in more recent years, it may make more sense to use the linear combination of Weibulls. In addition, with respect to the standard deviation of number of games off of ESPN's ExWL (2.29) and the linear combination of Weibulls (2.21), we can see that the linear combination of Weibulls provides on average a tighter fit, i.e. there is less fluctuation in the mean number of games off for each team in each year (from 1990 to 2013, ESPN's ExWL standard deviation in games off is 2.34 while that of the linear combination of Weibulls is 2.22, so we again see that the linear combination does noticeably better in recent years).

It is important to note that ESPN just takes the functional form of the Pythagorean Win/Loss Formula with an exponent (γ) of 2, while we give theoretical justification for our formula by utilizing some nice properties of the Weibull distribution and *Mathematica*. We also note that in the years from 1979 to 2001, when we are comparing against `baseball-reference.com`'s Pythagorean Win/Loss Formula with an exponent of 1.83 rather than 2 (which ESPN's ExWL uses), it seems that the difference in games off is more centered around zero than in the years when we are comparing against ESPN's ExWL (2002-2013). Thus, it might make sense for ESPN to use an exponent of 1.83 when computing their ExWL rather than 2.



```
> t.test(espn,twoweight)

	Welch Two Sample t-test

data:  espn and twoweight
t = 0.5152, df = 67.96, p-value = 0.6081
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.1601608  0.2716338
sample estimates:
mean of x mean of y 
 3.092957  3.037221 
```

FIGURE 20. t-test to determine whether the difference between the games off determined by ESPN ExWL and those by linear combinations of Weibulls is statistically significant.

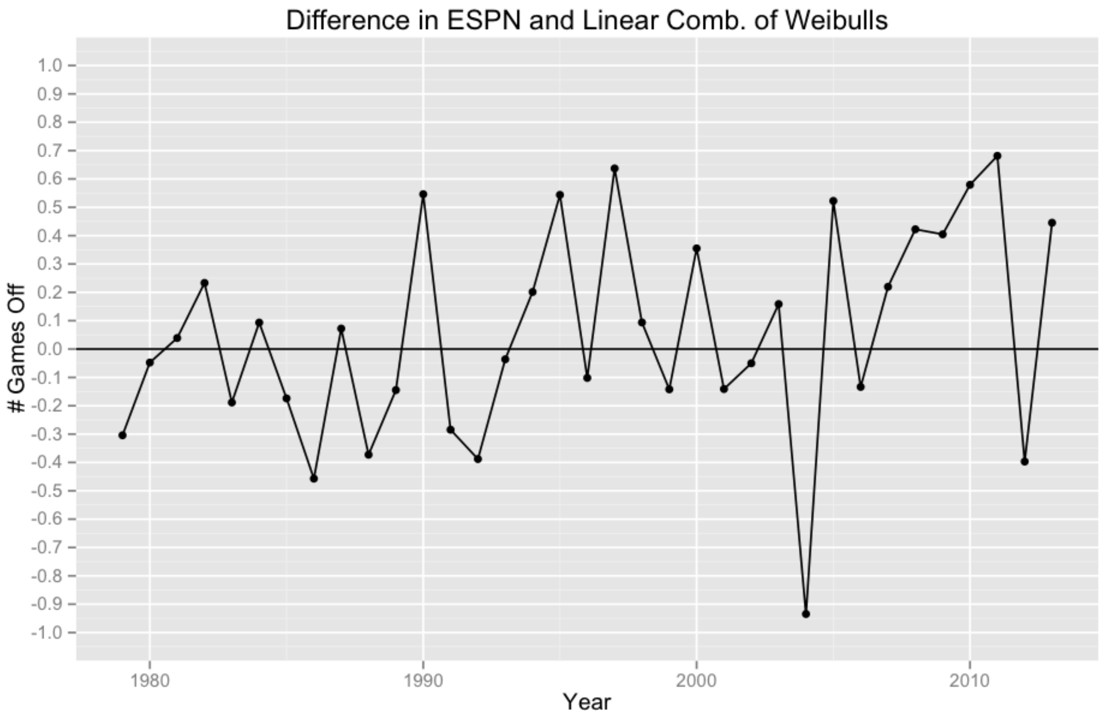


FIGURE 21. Difference in mean number of games off for ESPN ExWL and linear combination of Weibulls from 1979-2013.

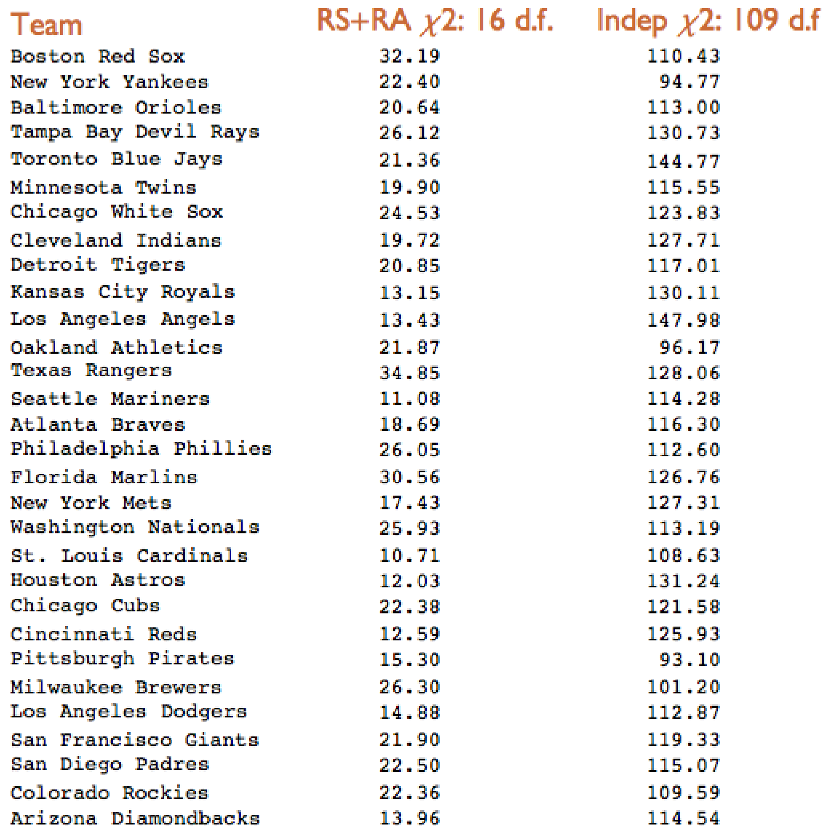
We also performed χ^2 tests to determine the goodness of fit to see how well the linear combination of Weibulls maps the observed data, and whether runs scored and allowed are independent. We used the bins as in (2.2), namely

$$\left[-\frac{1}{2}, \frac{1}{2}\right), \left[\frac{1}{2}, \frac{3}{2}\right), \left[\frac{3}{2}, \frac{5}{2}\right), \cdots$$

and test statistic

$$\sum_{k=1}^{\#\text{ Bins}} \frac{(RS_{obs}(k) - \#\text{Games} * A(\alpha_{\text{RS}_1}, \alpha_{\text{RS}_2}, -.5, -.5, \gamma, c_1, k))^2}{\#\text{Games} * A(\alpha_{\text{RS}_1}, \alpha_{\text{RS}_2}, -.5, -.5, \gamma, c_1, k)} + \sum_{k=1}^{\#\text{ Bins}} \frac{(RA_{obs}(k) - \#\text{Games} * A(\alpha_{\text{RA}_1}, \alpha_{\text{RA}_2}, -.5, -.5, \gamma, c_1', k))^2}{\#\text{Games} * A(\alpha_{\text{RA}_1}, \alpha_{\text{RS}_2}, -.5, -.5, \gamma, c_1', k)} \tag{4.8}$$

for the goodness of fit tests, with $2*(\#\text{Bins}-1)-1-7 = 16$ degrees of freedom, the factor of 7 coming from estimating 7 parameters, namely α_{RS_1}, α_{RS_2}, α_{RA_1}, α_{RA_2}, γ, c_1, and c_1'. We did not estimate β_1 or β_2, as we took them to be -.5. Having 16 degrees of freedom gives critical threshold values of 26.3 (at the 95% level) and 32.0 (at the 99% level). However, since there are multiple comparisons being done (namely 30 for the different teams), we use a Bonferroni adjustment ($\alpha_{new} = (1 - (1 - \alpha)^c)$, where c is the number of comparisons, in our case the number of teams) and obtain critical thresholds of 37.7 (95%) and 42.5 (99%). From the first column of Figure 22, all the teams fall within the unadjusted 99% threshold, with the exception of the Texas Rangers (just barely!), who easily fall into the Bonferroni adjusted 95% threshold. Therefore, the observed data closely follows a linear combination of Weibulls with the proper estimated parameters. A more in depth discussion of the justification behind the tests can be seen in [MIL].



Team	RS+RA χ2: 16 d.f.	Indep χ2: 109 d.f
Boston Red Sox	32.19	110.43
New York Yankees	22.40	94.77
Baltimore Orioles	20.64	113.00
Tampa Bay Devil Rays	26.12	130.73
Toronto Blue Jays	21.36	144.77
Minnesota Twins	19.90	115.55
Chicago White Sox	24.53	123.83
Cleveland Indians	19.72	127.71
Detroit Tigers	20.85	117.01
Kansas City Royals	13.15	130.11
Los Angeles Angels	13.43	147.98
Oakland Athletics	21.87	96.17
Texas Rangers	34.85	128.06
Seattle Mariners	11.08	114.28
Atlanta Braves	18.69	116.30
Philadelphia Phillies	26.05	112.60
Florida Marlins	30.56	126.76
New York Mets	17.43	127.31
Washington Nationals	25.93	113.19
St. Louis Cardinals	10.71	108.63
Houston Astros	12.03	131.24
Chicago Cubs	22.38	121.58
Cincinnati Reds	12.59	125.93
Pittsburgh Pirates	15.30	93.10
Milwaukee Brewers	26.30	101.20
Los Angeles Dodgers	14.88	112.87
San Francisco Giants	21.90	119.33
San Diego Padres	22.50	115.07
Colorado Rockies	22.36	109.59
Arizona Diamondbacks	13.96	114.54

FIGURE 22. χ^2 test results of the 2011 season from least squares of goodness of fit and independence of runs score and allowed.

Since the test for independence of runs scored and allowed requires that the row and column of the contingency table have at least one non-zero entry, the bins used to bin the runs score and allowed were

$$[0, 1) \cup [1, 2) \cup \cdots \cup [9, 10) \cup [11, \infty). \tag{4.9}$$

We use integer endpoints because we are using the observed runs from games. So, we have a 12 by 12 contingency table with zeroes along the diagonal, since runs scored and allowed can never be equal. This leads to an incomplete 12 by 12 contingency table with $(12 - 1)^2 - 12 = 109$ degrees of freedom; constructing a test requires the use of structural zeroes. The theory behind tests using structural zeroes can be seen in Appendix 9.2 of [MCGLP]. We observe that 109 degrees of freedom give critical threshold values of 134.37 (at the 95$\%$ level) and 146.26 (at the 99$\%$ level). Again, since we are doing multiple comparisons, we use a Bonferroni adjustment, obtaining critical thresholds of 157.68 (95%) and 166.45 (99%). From the second column of Figure 22, all the teams fall within the 99% threshold, with the exception of the Los Angeles Angels (just barely!), who easily fall into the Bonferroni adjusted 95% threshold. Thus, runs scored and allowed are acting as though they are statistically independent.

4.4. **A Simpler Formula?** While the one game improvement in prediction is very promising, the formula and curve fitting/parameter estimating is not very easily implemented. So, we tried to simplify the formula, even giving up some accuracy, in order to devise a formula that could be easily implemented using just a team's runs scored and allowed (and the variance of each of these) in order to determine the team's W/L percentage. Unfortunately, the weight parameters c_1 and c_1' plays too much of a factor; for example, in 2011, the mean of the parameter c_1 is 0.21 with a standard deviation of 0.39 (and a median of 0.21). With such large fluctuations in the weight parameters from team to team, the task of finding a simpler formula was almost impossible, as creating a uniform formula that every team could use was not feasible when two of the key parameters were so volatile.

Appendix A. 2011 Game State Code

```
#install.packages(c("XML","RCurl"))

##########################################################
##########################################################
#using try function .... not sure what it is doing.

require(XML)
require(RCurl)

#clear all variables before evaluating code
rm(list=ls(all=TRUE))

#variables

#years=c("2006","2007","2008","2009","2010","2011","2012","2013")
years=c("2011")
months=c("03","04","05","06","07","08","09","10","11")
days=c("01","02","03","04","05","06","07","08","09","10","11","12","13","14","15","16","17","18","19","20","21","22","
 23","24","25","26","27","28","29","30","31")
doubleheaders=c("0","1","2")
teams=c("ANA", "ARI", "ATL", "BAL", "BOS", "CHA", "CHN", "CIN", "CLE", "COL", "DET", "FLO", "HOU", "KCA", "LAN",
 "MIL", "MIN", "NYA", "NYN", "OAK", "PHI", "PIT", "SDN", "SEA", "SFN", "SLN", "TBA", "TEX", "TOR", "WAS")

#data frame that will store everything
seasonGameSt<-data.frame(Date=character(),
 Home_Team=character(),Home_ScoreGS=character(),Away_Team=character(),Away_ScoreGS=character())

for(year in years){
    #add in ballpark effects here
    for(month in months){
        for(day in days){
            for(doubleheader in doubleheaders){
                for(team in teams){
                    #getting box score for game of type team, year, month, day, doubleheader
                    url<-paste("http://www.baseball-reference.com/
 boxes/",team,"/",team,year,month,day,doubleheader,".shtml",sep="")

                    #check to see if the url actually exists
                    if(url.exists(url)){
                        rawhtml<-getURLContent(url)

                        #final check to see if the url is actually valid
                        if(identical(as.character(rawhtml),"")){
                            next()
                        }

                        rawPMI<-readHTMLTable(url)
                        playBP<-as.data.frame(rawPMI$play_by_play)

                        #attach(playBP)  probably don't need
                        colnames(playBP)<-c("Inn","Score","Out", "RoB", "Pitch","RO","AtBat","Batter", "Pitcher",
 "wWPA", "wWE", "Play")
                        #attach(playBP)   probably don't need

                        #get rid of rows with NA in them for wWE
                        newplayBP<- subset(playBP,! wWE %in% c(NA))
                        attach(newplayBP)

                        #get the home and away team, and if necessary, convert away teams to original team names
                        HomeTeam=team
                        AwayTeam=as.character(AtBat[1])
                        if(identical(AwayTeam,"NYY")==TRUE){
                            AwayTeam<-"NYA"
                        }
                        if(identical(AwayTeam,"TBD")==TRUE){
                            AwayTeam<-"TBA"
                        }
                        if(identical(AwayTeam,"NYM")==TRUE){
```

```
                                    AwayTeam<-"NYN"
                                }
                                if(identical(AwayTeam,"LAA")==TRUE){
                                    AwayTeam<-"ANA"
                                }
                                if(identical(AwayTeam,"SFG")==TRUE){
                                    AwayTeam<-"SFN"
                                }
                                if(identical(AwayTeam,"KCR")==TRUE){
                                    AwayTeam<-"KCA"
                                }
                                if(identical(AwayTeam,"STL")==TRUE){
                                    AwayTeam<-"SLN"
                                }
                                if(identical(AwayTeam,"LAD")==TRUE){
                                    AwayTeam<-"LAN"
                                }
                                if(identical(AwayTeam,"CHW")==TRUE){
                                    AwayTeam<-"CHA"
                                }
                                if(identical(AwayTeam,"CHC")==TRUE){
                                    AwayTeam<-"CHN"
                                }
                                if(identical(AwayTeam,"SDP")==TRUE){
                                    AwayTeam<-"SDN"
                                }
                                if(identical(AwayTeam,"TBR")==TRUE){
                                    AwayTeam<-"TBA"
                                }
                                if(identical(AwayTeam,"WSH")==TRUE){
                                    AwayTeam<-"WAS"
                                }
                                if(identical(AwayTeam,"WSN")==TRUE){
                                    AwayTeam<-"WAS"
                                }
                                if(identical(AwayTeam,"MIA")==TRUE){
                                    AwayTeam<-"FLO"
                                }
                                if(identical(AwayTeam,"FLA")==TRUE){
                                    AwayTeam<-"FLO"
                                }

                                ###mlb site to get scores

                                #correcting for site errors
                                holderdouble<-""
                                if(identical(doubleheader,"0")||identical(doubleheader,"1")){
                                    holderdouble<-"1"
                                }
                                else{
                                    holderdouble<-"2"
                                }

                                url2<-paste("http://mlb.mlb.com/news/boxscore.jsp?
gid=",year,"_",month,"_",day,"_",tolower(AwayTeam),"mlb_",tolower(HomeTeam),"mlb_",holderdouble,"&mode=box",sep="")
                                rawPMI2<-try(readHTMLTable(url2))
                                #catch error in writing rawPMI2
                                if(class(rawPMI2)=="try-error"){
                                    seasonGameSt<-rbind(seasonGameSt,
data.frame(Date=paste(year,"/",month,"/",day,sep=""),Home_Team=HomeTeam, Home_ScoreGS="NA", Away_Team=AwayTeam,
Away_ScoreGS="NA"))
                                    next;
                                }

                                #initialize variables... really dumb
                                HomeScore<-""
                                AwayScore<-""
```

```
#getting the adjusted for game state scores
dummy<- FALSE
for(i in 1:nrow(newplayBP)){


    winExpect<-as.character(newplayBP$wWE[i])
    winExpectNoP<-gsub("%","",winExpect)
    numericalWinEx<-as.integer(winExpectNoP)
    rankBoxScore<-i+1

    #if goes past and numericalWinEx still not past threshold
    ###just added in november 25, 2013
    if(rankBoxScore>nrow(newplayBP)){
            HomeScore<-as.character(rawPMI2$`print-linescore`$R[2])
            AwayScore<-as.character(rawPMI2$`print-linescore`$R[1])
            dummy<-TRUE
            break
    }

    #98% threshold
    if((numericalWinEx>97) || (numericalWinEx<3)){
        batTeam<-as.character(newplayBP$AtBat[i+1])
        boxScore<-as.character(newplayBP$Score[i+1])
        placeInn<-as.character(substring(newplayBP$Inn[i+1],1,1))
        numInn<-as.character(substring(newplayBP$Inn[i+1],2))
        minusLocation<-which(strsplit(boxScore, "")[[1]]=="-")
        batScore<-substr(boxScore,1,minusLocation-1)
        pitchScore<-substring(boxScore,minusLocation+1)
        HolderHomeTeam<-""
        HolderHomeTeam2<-""

        if(identical(HomeTeam,"NYA")==TRUE){
            HolderHomeTeam<-"NYY"
        }
        if(identical(HomeTeam,"NYN")==TRUE){
            HolderHomeTeam<-"NYM"
        }
        if(identical(HomeTeam,"ANA")==TRUE){
            HolderHomeTeam<-"LAA"
        }
        if(identical(HomeTeam,"SFN")==TRUE){
            HolderHomeTeam<-"SFG"
        }
        if(identical(HomeTeam,"KCA")==TRUE){
            HolderHomeTeam<-"KCR"
        }
        if(identical(HomeTeam,"SLN")==TRUE){
            HolderHomeTeam<-"STL"
        }
        if(identical(HomeTeam,"LAN")==TRUE){
            HolderHomeTeam<-"LAD"
        }
        if(identical(HomeTeam,"CHA")==TRUE){
            HolderHomeTeam<-"CHW"
        }
        if(identical(HomeTeam,"CHN")==TRUE){
            HolderHomeTeam<-"CHC"
        }
        if(identical(HomeTeam,"SDN")==TRUE){
            HolderHomeTeam<-"SDP"
        }
        if(identical(HomeTeam,"TBA")==TRUE){
            HolderHomeTeam<-"TBR"
        }
        if(identical(HomeTeam,"WAS")==TRUE){
            HolderHomeTeam<-"WSH"
            HolderHomeTeam<-"WSN"
        }
```

```
                                        if(identical(HomeTeam,"TBA")==TRUE){
                                            HolderHomeTeam<-"TBD"
                                        }
                                        if(identical(HomeTeam,"FLO")==TRUE){
                                            HolderHomeTeam<-"MIA"
                                            HolderHomeTeam2<-"FLA"
                                        }


                                        if(identical(batTeam,HomeTeam)||identical(batTeam,HolderHomeTeam)||
identical(batTeam,HolderHomeTeam2)){
                                            if(identical("t",placeInn)){
                                                HomeScore<-as.character(as.numeric(batScore)*(9/as.numeric(numInn)))
                                                AwayScore<-as.character(as.numeric(pitchScore)*(9/(as.numeric(numInn)-1)))
                                            }
                                            else{
                                                HomeScore<-as.character(as.numeric(batScore)*(9/as.numeric(numInn)))
                                                AwayScore<-as.character(as.numeric(pitchScore)*(9/as.numeric(numInn)))
                                                }
                                        }
                                        else{
                                            if(identical("t",placeInn)){
                                                HomeScore<-as.character(as.numeric(pitchScore)*(9/(as.numeric(numInn)-1)))
                                                AwayScore<-as.character(as.numeric(batScore)*(9/(as.numeric(numInn))))
                                            }
                                            else{
                                                HomeScore<-as.character(as.numeric(pitchScore)*(9/as.numeric(numInn)))
                                                AwayScore<-as.character(as.numeric(batScore)*(9/as.numeric(numInn)))
                                                }
                                        }

                                    #need to exit the for loop now
                                    dummy<- TRUE
                                    break
                                    }

                                    #exits for loop if dummy is true
                                    if(dummy){break}
                                }

                                #insert things into the data frame
                                seasonGameSt<-rbind(seasonGameSt, data.frame(Date=paste(year,"/",month,"/",day,sep=""),
Home_Team=HomeTeam, Home_ScoreGS=HomeScore, Away_Team=AwayTeam, Away_ScoreGS=AwayScore))

                                print(seasonGameSt)

                                #clearing things to potentially increase run speed?
                                rm(list=setdiff(ls(),c("seasonGameSt","years","months","days", "doubleheaders", "teams",
"year", "month", "day","doubleheader","team", "i")))

                            }
                        }
                    }
                }
            }
}
```

Appendix B. 2011 WHIP Code

```
#install.packages(c("XML","RCurl"))

##########################################################
##########################################################
#DON'T FORGET TO CHANGE YEAR IN URL (LINE 25), TEAMS (LINE 45), AND IN DATA (LINE 123)
#marlins and rays change over years

require(XML)
require(RCurl)

#clear all variables before evaluating code
rm(list=ls(all=TRUE))

#variables
pitcher.stats<-data.frame()

years=c("2006","2007","2008","2009","2010","2011","2012","2013")
teams=c("LAA", "ARI", "ATL", "BAL", "BOS", "CHW","CHC", "CIN", "CLE", "COL", "DET", "FLA", "HOU", "KCR", "LAD","MIL",
 "MIN", "NYY", "NYM", "OAK", "PHI", "PIT", "SDP","SEA", "SFG", "STL", "TBD","TEX", "TOR", "WSN")
Team=c()

for(team in teams){
    #getting box score for team and year
    #IMPORTANT HERE
    #change year accordingly
    #REALLY IMPORTANT HERE
    url<-paste("http://www.baseball-reference.com/teams/",team,"/2011.shtml",sep="")

    rawhtml<-getURLContent(url)

    #final check to see if the url is actually valid
    if(identical(as.character(rawhtml),"")){
        next()
    }

    rawPMI<-readHTMLTable(url)
    pitching<-as.data.frame(rawPMI$team_pitching)
    colnames(pitching)[3]<-"name"
    attach(pitching)

    pitcher.stats<-rbind(pitcher.stats,pitching)

    #get team
    #don't forget to change these for different years
    for(j in 1:nrow(pitching)){
        if(identical(as.character(pitching$name[j]),"")==FALSE){
            if(identical(team,"LAA")){
                Team=c(Team,"ANA")
            }
            else if(identical(team,"CHW")){
                Team=c(Team,"CHA")
            }
            else if(identical(team,"CHC")){
                Team=c(Team,"CHN")
            }
            else if(identical(team,"FLA")){
                Team=c(Team,"FLO")
            }
            else if(identical(team,"KCR")){
                Team=c(Team,"KCA")
            }
            else if(identical(team,"LAD")){
                Team=c(Team,"LAN")
            }
            else if(identical(team,"NYY")){
                Team=c(Team,"NYA")
            }
            else if(identical(team,"NYN")){
                Team=c(Team,"NYM")
            }
            else if(identical(team,"SDP")){
                Team=c(Team,"SDN")
            }
            else if(identical(team,"SFG")){
                Team=c(Team,"SFN")
            }
            else if(identical(team,"STL")){
                Team=c(Team,"SLN")
            }
            else if(identical(team,"TBD")){
                Team=c(Team,"TBA")
            }
            else if(identical(team,"WSN")){
                Team=c(Team,"WAS")
            }else{
                Team=c(Team,team)
            }
        }
    }
}

#cleaning up the pitcher stats

#first get rid of rows where there is no name
numrows=nrow(pitcher.stats)
for(i in 1:numrows){
    if(identical(as.character(pitcher.stats$name[i]),"")){
        pitcher.stats<-pitcher.stats[-i,]
    }
}
#update row indices
rownames(pitcher.stats) <- 1:nrow(pitcher.stats)


#getting rid of all the stars in the names
Names.updated=c()
for(i in 1:nrow(pitcher.stats)){
    name.new <- as.character(pitcher.stats$name[i])

    if(identical("*",substr(name.new,nchar(name.new),nchar(name.new)))){
        Names.updated<-c(Names.updated,substr(name.new,0,nchar(name.new)-1))
    }
    else{
        Names.updated<-c(Names.updated,name.new)

    }
}

pitcher.stats<-pitcher.stats[,c(-3)]
pitcher.stats<-cbind(Names.updated,Team,pitcher.stats)
colnames(pitcher.stats)[28]<-"ERA.plus"

#########################################################################################

#pythag exponents table
pythag.table <-data.frame()

#now bring in data
#depends on the year
data<-read.csv("/Users/victorluo/Documents/Williams College/Senior Year/thesis/Complete Season Data/
2011completedata.csv")
ballpark<-read.csv("/Users/victorluo/Documents/Williams College/Senior Year/thesis/Ballpark/Outdated Ballpark Effects/
2011ballparkeffects.csv")
```

```
teams.retro=c("ANA", "ARI", "ATL", "BAL", "BOS", "CHA", "CHN", "CIN", "CLE", "COL", "DET", "FLO", "HOU", "KCA", "LAN",
 "MIL", "MIN", "NYA", "NYN", "OAK", "PHI", "PIT", "SDN", "SEA", "SFN", "SLN", "TBA", "TEX", "TOR", "WAS")


#loop to find best pythag exponent power
for(l in 1:25){
    #pythag exponent power
    alpha <- runif(1,0.5,2.5)

    #intialize vectors
    teams.pythag<-c()
    teams.pythag.WHIP=c()
    teams.pythag.WHIP.park=c()
    teamsWL<-c()

#loop over teams
for(team in teams.retro){
    RS<-c()
    RS.WHIP<-c()
    RS.WHIP.park<-c()
    wins <- 0
    losses <- 0
    RA<-c()
    RA.WHIP<-c()
    RA.WHIP.park<-c()

    #get runs scored for pythag WL
    #also get runs scored for WHIP pythag WL
    for (i in 1:nrow(data)){
        #if the team is the home team
        if(data$Home_Team[i]==team){
            RS=c(RS, data$Home_Score[i])
            #get WHIP PLUS data
            indices <- match(as.character(data$Away_Pitcher[i]),pitcher.stats$Names.updated)
            for(index in indices){
                if(identical(as.character(pitcher.stats$Team[index]),as.character(data$Away_Team[i]))){
                    pitcher.plus<-as.character(pitcher.stats$WHIP[index])
                }
            }
            #HERE IS WHIP PLUS DATA
            ##############
            RS.WHIP = c(RS.WHIP, data$Home_Score[i]*(as.numeric(pitcher.plus)*(1/100)))
            RS.WHIP.park = c(RS.WHIP.park, data$Home_Score[i]*((as.numeric(pitcher.plus)*(1/100))^(1/8)/(ballpark
 $BallparkEffect[data$Home_Team[i]])^(1.5)))

            #get w/l
            if(data$Home_Score[i]>data$Away_Score[i]){
                wins <- wins+1
            }
            else{
                losses <- losses+1
            }
        }

        #if the team is the away team
        if(data$Away_Team[i]==team){
            RS=c(RS, data$Away_Score[i])
            #get WHIP PLUS data
            indices <- match(as.character(data$Home_Pitcher[i]),pitcher.stats$Names.updated)
            for(index in indices){
                if(identical(as.character(pitcher.stats$Team[index]),as.character(data$Home_Team[i]))){
                    pitcher.plus<-as.character(pitcher.stats$WHIP[index])
                }
            }
            #HERE IS WHIP PLUS DATA
            ##############
            RS.WHIP = c(RS.WHIP, data$Away_Score[i]*(as.numeric(pitcher.plus)*(1/100)))
            RS.WHIP.park = c(RS.WHIP.park, data$Away_Score[i]*((as.numeric(pitcher.plus)*(1/100))^(1/8)/(ballpark
$BallparkEffect[data$Home_Team[i]])^(1.5)))

            #get w/l
            if(data$Home_Score[i]<data$Away_Score[i]){
                wins <- wins+1
            }
            else{
                losses <- losses+1
            }
        }
    }

    #get runs allowed for pythag WL
    for (i in 1:nrow(data)){
        #if team is the home team
        if(data$Home_Team[i]==team){
            RA=c(RA, data$Away_Score[i])
            #get WHIP PLUS data
            indices <- match(as.character(data$Home_Pitcher[i]),pitcher.stats$Names.updated)
            for(index in indices){
                if(identical(as.character(pitcher.stats$Team[index]),as.character(data$Home_Team[i]))){
                    pitcher.plus<-as.character(pitcher.stats$WHIP[index])
                }
            }
            #HERE IS THE WHIP PLUS
            ##############
            RA.WHIP = c(RA.WHIP, data$Away_Score[i]*(as.numeric(pitcher.plus)*(1/100)))
            RA.WHIP.park = c(RA.WHIP.park, data$Away_Score[i]*((as.numeric(pitcher.plus)*(1/100))^(1/8)/(ballpark
$BallparkEffect[data$Home_Team[i]])^(1.5)))
        }
        #if team is the away team
        if(data$Away_Team[i]==team){
            RA=c(RA, data$Home_Score[i])
            #get WHIP PLUS data
            indices <- match(as.character(data$Away_Pitcher[i]),pitcher.stats$Names.updated)
            for(index in indices){
                if(identical(as.character(pitcher.stats$Team[index]),as.character(data$Away_Team[i]))){
                    pitcher.plus<-as.character(pitcher.stats$WHIP[index])
                }
            }
            #HERE IS WHIP PLUS DATA
            ##############
            RA.WHIP = c(RA.WHIP, data$Home_Score[i]*(as.numeric(pitcher.plus)*(1/100)))
            RA.WHIP.park = c(RA.WHIP.park, data$Home_Score[i]*((as.numeric(pitcher.plus)*(1/100))^(1/8)/(ballpark
$BallparkEffect[data$Home_Team[i]])^(1.5)))
        }
    }

    #take sums over runs scored in every category#
    #regular RS and RA
    runs.score=sum(RS)
    runs.allow=sum(RA)
    #WHIP
    runs.score.WHIP=sum(RS.WHIP)
    runs.allow.WHIP=sum(RA.WHIP)
    #WHIP + park
    runs.score.WHIP.park=sum(RS.WHIP.park)
    runs.allow.WHIP.park=sum(RA.WHIP.park)

    #compute the pythag formula for regular
    PythagWL=(runs.score^(alpha))/(runs.score^(alpha)+runs.allow^(alpha))
    #WHIP
    PythagWL.WHIP=(runs.score.WHIP^(alpha))/(runs.score.WHIP^(alpha)+runs.allow.WHIP^(alpha))
    PythagWL.WHIP.park=(runs.score.WHIP.park^(alpha))/(runs.score.WHIP.park^(alpha)+runs.allow.WHIP.park^(alpha))

    #add in the pythags into their vectors
    teams.pythag=c(teams.pythag,PythagWL)
    teams.pythag.WHIP=c(teams.pythag.WHIP,PythagWL.WHIP)
    teams.pythag.WHIP.park=c(teams.pythag.WHIP.park,PythagWL.WHIP.park)

    #compute teams W/L for the season
    WL <- wins/(wins + losses)
    teamsWL=c(teamsWL, WL)
}
######### end of loop over teams #################

model.pythag=lm(teamsWL~teams.pythag)
model.pythag.WHIP=lm(teamsWL~teams.pythag.WHIP)
model.pythag.WHIP.park = lm(teamsWL~teams.pythag.WHIP.park)


reg <- summary(model.pythag)$r.squared
plus <- summary(model.pythag.WHIP)$r.squared
plus.park <- summary(model.pythag.WHIP.park)$r.squared

#summary(fit)$r.squared

pythag.table <- rbind(pythag.table, c(alpha,reg,plus,plus.park))

print(pythag.table)
}
####### end of loop over pythag powers #################
colnames(pythag.table)<- c("Pythag.Exp", "Regular", "WHIP", "WHIP.Ballpark")

sorted.table<-pythag.table[order(pythag.table$Regular),]

#write.csv(sorted.table,"sorted.WHIP.ballpark2007")

##################################################################################
#interesting.... raising the WHIP to a power affects it
#TRY 2007 NEXT (1/8, 1.5)
#2009 (0,2.2)
#2010 (0, 0.5)
#2011 (0,1.5)
#2012(0,0.5)
```

APPENDIX C. 2011 WAR CODE

```
#install.packages(c("XML","RCurl"))

##########################################################
##########################################################
#DON'T FORGET TO CHANGE YEAR IN URL (LINE 25), TEAMS (LINE 45), AND IN DATA (LINE 123)
#marlins and rays change over years

require(XML)
require(RCurl)

#clear all variables before evaluating code
rm(list=ls(all=TRUE))

#variables
pitcher.stats<-data.frame()
pitcher.stats1<-data.frame()

years=c("2006","2007","2008","2009","2010","2011","2012","2013")
teams=c("LAA", "ARI", "ATL", "BAL", "BOS", "CHW","CHC", "CIN", "CLE", "COL", "DET", "FLA", "HOU", "KCR", "LAD","MIL",
 "MIN", "NYY", "NYM", "OAK", "PHI", "PIT", "SDP","SEA", "SFG", "STL", "TBD","TEX", "TOR", "WSN")
Team=c()

for(team in teams){
    #getting box score for team and year
    #IMPORTANT HERE
    #change year accordingly
    #REALLY IMPORTANT HERE
    url<-paste("http://www.baseball-reference.com/teams/",team,"/2011.shtml",sep="")
    url1<-paste("http://www.baseball-reference.com/teams/",team,"/2011-pitching.shtml",sep="")

    rawhtml<-getURLContent(url)

    #final check to see if the url is actually valid
    if(identical(as.character(rawhtml),"")){
        next()
    }

    rawPMI<-readHTMLTable(url)
    rawPMI1<-readHTMLTable(url1)
    pitching<-as.data.frame(rawPMI$team_pitching)
    pitching1<-as.data.frame(rawPMI1$players_value_pitching)
    colnames(pitching)[3]<-"name"
    attach(pitching)

    pitcher.stats<-rbind(pitcher.stats,pitching)
    pitcher.stats1<-rbind(pitcher.stats1,pitching1)

    #get team
    #don't forget to change these for different years
    for(j in 1:nrow(pitching)){
        if(identical(as.character(pitching$name[j]),"")==FALSE){
            if(identical(team,"LAA")){
                Team=c(Team,"ANA")
            }
            else if(identical(team,"CHW")){
                Team=c(Team,"CHA")
            }
            else if(identical(team,"CHC")){
                Team=c(Team,"CHN")
            }
            else if(identical(team,"FLA")){
                Team=c(Team,"FLO")
            }
            else if(identical(team,"KCR")){
                Team=c(Team,"KCA")
            }
            else if(identical(team,"LAD")){
                Team=c(Team,"LAN")
            }
            else if(identical(team,"NYY")){
                Team=c(Team,"NYA")
            }
            else if(identical(team,"NYN")){
                Team=c(Team,"NYM")
            }
            else if(identical(team,"SDP")){
                Team=c(Team,"SDN")
            }
            else if(identical(team,"SFG")){
                Team=c(Team,"SFN")
            }
            else if(identical(team,"STL")){
                Team=c(Team,"SLN")
            }
            else if(identical(team,"TBD")){
                Team=c(Team,"TBA")
            }
            else if(identical(team,"WSN")){
                Team=c(Team,"WAS")
            }else{
                Team=c(Team,team)
            }
        }
    }

}

#cleaning up the pitcher stats

#first get rid of rows where there is no name
numrows=nrow(pitcher.stats)
for(i in 1:numrows){
    if(identical(as.character(pitcher.stats$name[i]),"")){
        pitcher.stats<-pitcher.stats[-i,]
    }
}
#update row indices
rownames(pitcher.stats) <- 1:nrow(pitcher.stats)


#getting rid of all the stars in the names
Names.updated=c()
for(i in 1:nrow(pitcher.stats)){
    name.new <- as.character(pitcher.stats$name[i])

    if(identical("*",substr(name.new,nchar(name.new),nchar(name.new)))){
        Names.updated<-c(Names.updated,substr(name.new,0,nchar(name.new)-1))
    }
    else{
        Names.updated<-c(Names.updated,name.new)

    }
}

pitcher.stats<-pitcher.stats[,c(-3)]
pitcher.stats<-cbind(Names.updated,Team,pitcher.stats)

####standard WAR####
#############
NEW.WAR<- c()
for(j in 1:length(pitcher.stats1$WAR)){
    if(as.character(pitcher.stats1$WAR[j])==""){
        NEW.WAR<-c(NEW.WAR,"NA")
        next}
    else{
        NEW.WAR<-c(NEW.WAR,as.character(pitcher.stats1$WAR[j]))}
```

```
}
#getting mean and variance
dummy.WAR<-c()
for(k in NEW.WAR){
    if(k=="NA"){
        next}
    else{
        dummy.WAR<-c(dummy.WAR,k)}
}
WAR.mean<-mean(as.numeric(dummy.WAR))
WAR.var<-sqrt(var(as.numeric(dummy.WAR)))

#STANDARDIZATION
WAR.standard<-(as.numeric(dummy.WAR)-WAR.mean)/(WAR.var)
WARstand.min<-min(WAR.standard)
final.WAR<-c()
for(i in 1:length(NEW.WAR)){
    if(identical(NEW.WAR[i],"NA")==TRUE){
        final.WAR<-c(final.WAR, 1)}
    else{
        stand<-(as.numeric(NEW.WAR[i])-WAR.mean)/(WAR.var)
        what.add<-(stand-WARstand.min+0.5)
        final.WAR<-c(final.WAR,what.add)}
}
final.WAR<-final.WAR/mean(final.WAR)
pitcher.stats<-cbind(pitcher.stats,final.WAR)

print(final.WAR)




#########################################################################################

#pythag exponents table
pythag.table <-data.frame()

#now bring in data
#depends on the year
data<-read.csv("/Users/victorluo/Documents/Williams College/Senior Year/thesis/Complete Season Data/
 2011completedata.csv")
ballpark<-read.csv("/Users/victorluo/Documents/Williams College/Senior Year/thesis/Ballpark/Outdated Ballpark Effects/
 2011ballparkeffects.csv")



teams.retro=c("ANA", "ARI", "ATL", "BAL", "BOS", "CHA", "CHN", "CIN", "CLE", "COL", "DET", "FLO", "HOU", "KCA", "LAN",
 "MIL", "MIN", "NYA", "NYN", "OAK", "PHI", "PIT", "SDN", "SEA", "SFN", "SLN", "TBA", "TEX", "TOR", "WAS")


#loop to find best pythag exponent power
for(l in 1:25){
    #pythag exponent power
    alpha <- runif(1,0.5,2.5)

    #intialize vectors
    teams.pythag<-c()
    teams.pythag.WAR=c()
    teams.pythag.WAR.park=c()
    teamsWL<-c()

    #loop over teams
    for(team in teams.retro){
        RS<-c()
        RS.WAR<-c()
        RS.WAR.park<-c()
        wins <- 0
        losses <- 0
        RA<-c()
        RA.WAR<-c()
        RA.WAR.park<-c()

        #get runs scored for pythag WL
        #also get runs scored for WAR pythag WL
        for (i in 1:nrow(data)){
            #if the team is the home team
            if(data$Home_Team[i]==team){
                RS=c(RS, data$Home_Score[i])
                #get WAR data
                indices <- match(as.character(data$Away_Pitcher[i]),pitcher.stats$Names.updated)
                for(index in indices){
                    if(identical(as.character(pitcher.stats$Team[index]),as.character(data$Away_Team[i]))){
                        pitcher.plus<-pitcher.stats$final.WAR[index]
                    }
                }
                #HERE IS WAR DATA
                ##############
                RS.WAR = c(RS.WAR, (data$Home_Score[i])*pitcher.plus)
                RS.WAR.park = c(RS.WAR.park, (data$Home_Score[i])*pitcher.plus^(1/8)/(ballpark$BallparkEffect[data
$Home_Team[i]])^(1))

                #get w/l
                if(data$Home_Score[i]>data$Away_Score[i]){
                    wins <- wins+1
                }
                else{
                    losses <- losses+1
                }
            }

            #if the team is the away team
            if(data$Away_Team[i]==team){
                RS=c(RS, data$Away_Score[i])
                #get WAR PLUS data
                indices <- match(as.character(data$Home_Pitcher[i]),pitcher.stats$Names.updated)
                for(index in indices){
                    if(identical(as.character(pitcher.stats$Team[index]),as.character(data$Home_Team[i]))){
                        pitcher.plus<-pitcher.stats$final.WAR[index]
                    }
                }
                #HERE IS WAR DATA
                ##############
                RS.WAR = c(RS.WAR, data$Away_Score[i]*(pitcher.plus))
                RS.WAR.park = c(RS.WAR.park, data$Away_Score[i]*(pitcher.plus)^(1/8)/((ballpark$BallparkEffect[data
$Home_Team[i]])^(1)))

                #get w/l
                if(data$Home_Score[i]<data$Away_Score[i]){
                    wins <- wins+1
                }
                else{
                    losses <- losses+1
                }
            }
        }

        #get runs allowed for pythag WL
        for(i in 1:nrow(data)){
            #if team is the home team
            if(data$Home_Team[i]==team){
                RA=c(RA, data$Away_Score[i])
                #get WAR PLUS data
                indices <- match(as.character(data$Home_Pitcher[i]),pitcher.stats$Names.updated)
                for(index in indices){
                    if(identical(as.character(pitcher.stats$Team[index]),as.character(data$Home_Team[i]))){
                        pitcher.plus<-pitcher.stats$final.WAR[index]
                    }
                }
                #HERE IS THE WAR DATA
                ##############
                RA.WAR = c(RA.WAR, data$Away_Score[i]*(pitcher.plus))
                RA.WAR.park = c(RA.WAR.park, data$Away_Score[i]*(pitcher.plus)^(1/8)/((ballpark$BallparkEffect[data
$Home_Team[i]])^(1)))
            }

            #if team is the away team
            if(data$Away_Team[i]==team){
                RA=c(RA, data$Home_Score[i])
                #get WAR PLUS data
                indices <- match(as.character(data$Away_Pitcher[i]),pitcher.stats$Names.updated)
                for(index in indices){
                    if(identical(as.character(pitcher.stats$Team[index]),as.character(data$Away_Team[i]))){
                        pitcher.plus<-pitcher.stats$final.WAR[index]
                    }
                }
                #HERE IS WAR DATA
                ##############
                RA.WAR = c(RA.WAR, data$Home_Score[i]*(pitcher.plus))
                RA.WAR.park = c(RA.WAR.park, data$Home_Score[i]*(pitcher.plus)^(1/8)/((ballpark$BallparkEffect[data
$Home_Team[i]])^(1)))
            }
        }

        #take sums over runs scored in every category#
        #regular RS and RA
        runs.score=sum(RS)
        runs.allow=sum(RA)
        #WAR
        runs.score.WAR=sum(RS.WAR)
        runs.allow.WAR=sum(RA.WAR)
        #WAR + park
        runs.score.WAR.park=sum(RS.WAR.park)
        runs.allow.WAR.park=sum(RA.WAR.park)

        #compute the pythag formula for regular
        PythagWL=(runs.score^(alpha))/(runs.score^(alpha)+runs.allow^(alpha))
        #WAR
        PythagWL.WAR=(runs.score.WAR^(alpha))/(runs.score.WAR^(alpha)+runs.allow.WAR^(alpha))
        PythagWL.WAR.park=(runs.score.WAR.park^(alpha))/(runs.score.WAR.park^(alpha)+runs.allow.WAR.park^(alpha))

        #add in the pythags into their vectors
        teams.pythag=c(teams.pythag,PythagWL)
        teams.pythag.WAR=c(teams.pythag.WAR,PythagWL.WAR)
        teams.pythag.WAR.park=c(teams.pythag.WAR.park,PythagWL.WAR.park)

        #compute teams W/L for the season
        WL <- wins/(wins + losses)
        teamsWL=c(teamsWL, WL)
    }

######### end of loop over teams #################

model.pythag=lm(teamsWL~teams.pythag)
model.pythag.WAR=lm(teamsWL~teams.pythag.WAR)
model.pythag.WAR.park = lm(teamsWL~teams.pythag.WAR.park)


reg <- summary(model.pythag)$r.squared
plus <- summary(model.pythag.WAR)$r.squared
plus.park <- summary(model.pythag.WAR.park)$r.squared

#summary(fit)$r.squared
```

```
pythag.table <- rbind(pythag.table, c(alpha,reg,plus,plus.park))

print(pythag.table)
}
####### end of loop over pythag powers ##################
colnames(pythag.table)<- c("Pythag.Exp", "Regular", "WAR", "WAR.Ballpark")

sorted.table<-pythag.table[order(pythag.table$Regular),]

#write.csv(sorted.table,"sorted.WAR.ballpark2007")
```

Appendix D. 2007 Moments Code for R

```
#install.packages("moments")

require("moments")
#now bring in data
#depends on the year
data<-read.csv("/Users/victorluo/Documents/Williams College/Senior Year 13-14/thesis/Complete Season Data/
 2007completedata.csv")

teams.retro=c("ANA", "ARI", "ATL", "BAL", "BOS", "CHA", "CHN", "CIN", "CLE", "COL", "DET", "FLO", "HOU", "KCA", "LAN",
 "MIL", "MIN", "NYA", "NYN", "OAK", "PHI", "PIT", "SDN", "SEA", "SFN", "SLN", "TBA", "TEX", "TOR", "WAS")

moment.dataframe<-data.frame()

for(team in teams.retro){
    RS<-c()
    RA<-c()

    #get runs scored for pythag WL
    #also get runs scored for WHIP pythag WL
    for (i in 1:nrow(data)){
        #if the team is the home team
        if(data$Home_Team[i]==team){
            RS=c(RS, data$Home_Score[i])
        }

        #if the team is the away team
        if(data$Away_Team[i]==team){
            RS=c(RS, data$Away_Score[i])
        }
    }

    #get runs allowed for pythag WL
    for (i in 1:nrow(data)){
        #if team is the home team
        if(data$Home_Team[i]==team){
            RA=c(RA, data$Away_Score[i])
        }
        #if team is the away team
        if(data$Away_Team[i]==team){
            RA=c(RA, data$Home_Score[i])
        }
    }

    #calculate moments
    team.moments<-all.moments(RS,order.max=4)

    moment.dataframe<-rbind(moment.dataframe, team.moments)

}
moment.dataframe<-moment.dataframe[,-1]
moment.dataframe<-cbind(teams.retro,moment.dataframe)

colnames(moment.dataframe)<-c("Team","First","Second","Third","Fourth")
```

Pythagorean Formula Analysis –2011 (2 betas)
Victor Luo
Thesis 2013 – 2014

```
(*Multivariate statistics package needed*)
Needs["MultivariateStatistics`"]


(* combined weibulls... *)
W[x_, a1_, a2_, b1_, b2_, c_, d_] :=
  If[x ≥ d*b1 + (1 - d) *b2, d* (c/a1) ((x - b1) /a1) ^ (c - 1) * Exp[ - ((x - b1) /a1) ^c] +
    (1 - d) * (c/a2) ((x - b2) /a2) ^ (c - 1) * Exp[ - ((x - b2) /a2) ^c], 0];
(*defines weibull as function of 6 parameters, area under combined weibull in interval [sp,ep]*)
AW[sp_, ep_, a1_, a2_, b1_, b2_, c_, d_] :=
  If[sp ≥ d*b1 + (1 - d) *b2, (1 - d) * (N[-Exp[-((ep - b2) /a2) ^c] + Exp[-((sp - b2) /a2) ^c]]) +
    d* (N[-Exp[-((ep - b1) /a1) ^c] + Exp[-((sp - b1) /a1) ^c]]), If[ep > d*b1 + (1 - d) *b2,
    d* (N[1 - Exp[-((ep - b1) /a1) ^c]]) + (1 - d) * (N[1 - Exp[-((ep - b2) /a2) ^c]]), 0]];



(*Team scores*)
(* Team guide follows *)
(* AL EAST *)
(* Team 1 = Red Sox*)
(* Team 2 = Yankees*)
(* Team 3 = Orioles*)
(* Team 4 = Devil Rays*)
(* Team 5 = Blue Jays*)

TeamNames = {"Boston Red Sox", "New York Yankees", "Baltimore Orioles", "Tampa Bay Devil Rays",
   "Toronto Blue Jays", "Minnesota Twins", "Chicago White Sox", "Cleveland Indians", "Detroit Tigers",
   "Kansas City Royals", "Los Angeles Angels", "Oakland Athletics", "Texas Rangers",
   "Seattle Mariners", "Atlanta Braves", "Philadelphia Phillies", "Florida Marlins",
   "New York Mets", "Washington Nationals", "St. Louis Cardinals", "Houston Astros", "Chicago Cubs",
   "Cincinnati Reds", "Pittsburgh Pirates", "Milwaukee Brewers", "Los Angeles Dodgers",
   "San Francisco Giants", "San Diego Padres", "Colorado Rockies", "Arizona Diamondbacks" };
NumGames = 162;
NumTeams = 30; (* number of teams*)
DoubleNumTeams = 2 * NumTeams; (* twice the number of teams, each team has RS and RA data *)
TeamScore[1] = {5, 5, 1, 1, 4, 0, 9, 4, 4, 5, 2, 6, 4, 8, 9, 0, 5, 4, 4, 5, 7, 1, 4, 6, 4, 0, 3, 9, 7, 3, 0, 2, 4,
  9, 2, 6, 3, 5, 6, 7, 8, 1, 4, 15, 3, 5, 2, 4, 14, 14, 6, 4, 0, 3, 7, 4, 8, 9, 6, 6, 11, 8, 5, 16, 14, 0,
  3, 4, 10, 2, 12, 14, 4, 1, 1, 4, 4, 0, 1, 5, 7, 10, 2, 7, 3, 6, 10, 10, 4, 8, 6, 9, 1, 15, 2, 4, 7, 3,
  12, 1, 13, 12, 3, 1, 10, 5, 6, 3, 4, 3, 2, 10, 3, 8, 4, 2, 6, 4, 3, 3, 2, 0, 4, 7, 4, 6, 0, 11, 13, 6,
  5, 9, 4, 2, 9, 2, 0, 12, 4, 0, 14, 10, 4, 2, 5, 1, 18, 4, 2, 4, 3, 5, 5, 18, 5, 4, 1, 2, 7, 3, 8, 3};
TeamScore[1 + NumTeams] = {9, 12, 5, 3, 8, 1, 6, 9, 0, 16, 3, 7, 1, 1, 1, 5, 3, 2, 3, 0, 0, 4, 5, 2, 5, 2,
   2, 5, 3, 5, 11, 9, 0, 5, 1, 7, 9, 4, 0, 5, 7, 0, 3, 5, 9, 1, 3, 2, 2, 1, 3, 3, 3, 7, 10, 7, 6, 8, 3, 4,
   6, 3, 1, 4, 1, 4, 0, 2, 4, 4, 3, 5, 5, 5, 3, 6, 2, 5, 2, 2, 5, 4, 1, 9, 2, 4, 4, 3, 0, 6, 9, 5, 0, 10,
   6, 0, 4, 1, 8, 3, 9, 5, 4, 3, 2, 3, 9, 2, 3, 7, 3, 4, 2, 6, 3, 5, 4, 5, 5, 1, 6, 4, 3, 1, 9, 1, 4, 5, 2,
   0, 15, 3, 0, 5, 5, 4, 10, 7, 11, 1, 0, 11, 7, 7, 6, 9, 6, 5, 9, 3, 4, 8, 6, 9, 7, 6, 9, 6, 4, 6, 7, 4};
TeamScore[2] = {6, 10, 7, 4, 4, 4, 6, 9, 0, 7, 6, 3, 5, 6, 5, 6, 15, 6, 0, 2, 3, 12, 3, 5, 5, 5, 2, 0, 3,
   4, 5, 12, 3, 3, 5, 4, 0, 5, 5, 6, 4, 13, 1, 7, 9, 3, 5, 7, 3, 4, 7, 5, 10, 4, 2, 3, 5, 4, 6, 3, 11, 4,
   9, 0, 12, 12, 3, 1, 4, 10, 5, 4, 2, 2, 8, 6, 12, 5, 5, 5, 5, 2, 3, 9, 3, 1, 5, 1, 7, 1, 4, 7, 5, 2, 4,
   1, 17, 3, 7, 10, 4, 2, 2, 8, 17, 4, 3, 6, 18, 7, 3, 4, 2, 4, 9, 6, 1, 9, 7, 9, 4, 8, 8, 4, 3, 5, 4, 22,
   5, 0, 8, 3, 5, 5, 4, 3, 6, 9, 11, 5, 4, 4, 1, 0, 6, 9, 3, 1, 4, 7, 0, 6, 5, 4, 4, 8, 9, 6, 4, 2, 3, 7};
TeamScore[2 + NumTeams] = {3, 6, 10, 3, 5, 3, 9, 4, 4, 4, 5, 5, 2, 5, 6, 2, 3, 3, 2, 3, 1, 3, 5, 4, 2, 3,
   4, 4, 6, 1, 7, 5, 1, 4, 11, 5, 6, 7, 6, 2, 1, 2, 2, 3, 3, 7, 4, 3, 4, 5, 1, 0, 3, 2, 3, 2, 3, 6, 11, 8, 7,
   0, 1, 1, 4, 4, 2, 3, 3, 4, 3, 2, 10, 4, 3, 4, 2, 2, 0, 1, 2, 3, 6, 2, 5, 5, 4, 0, 16, 7, 1, 2, 4, 3, 0,
   2, 7, 4, 5, 3, 1, 9, 4, 3, 3, 2, 2, 0, 7, 2, 2, 10, 3, 6, 3, 5, 5, 2, 4, 7, 5, 4, 1, 9, 0, 6, 6, 9, 12,
   2, 3, 2, 2, 9, 2, 2, 4, 3, 10, 3, 5, 5, 2, 6, 5, 3, 2, 2, 5, 6, 3, 4, 0, 2, 2, 15, 1, 2, 7, 5, 5, 8};
TeamScore[3] = {4, 3, 5, 5, 3, 9, 5, 1, 0, 4, 5, 2, 3, 2, 3, 11, 5, 1, 3, 3, 4, 5, 2, 10, 6, 6, 2, 5, 3,
   1, 2, 2, 3, 7, 4, 2, 0, 6, 9, 7, 1, 2, 5, 8, 2, 5, 9, 6, 2, 2, 4, 3, 2, 2, 4, 5, 4, 4, 4, 3, 7, 5, 6,
   5, 1, 4, 4, 2, 7, 8, 3, 4, 5, 5, 7, 2, 1, 6, 0, 4, 5, 4, 2, 5, 4, 3, 0, 6, 4, 5, 6, 8, 10, 6, 0, 1,
   3, 3, 12, 0, 5, 4, 3, 3, 2, 8, 2, 4, 4, 6, 2, 6, 3, 6, 3, 4, 5, 8, 6, 4, 5, 3, 8, 1, 4, 8, 6, 6, 12,
   2, 3, 2, 6, 0, 6, 3, 3, 1, 10, 3, 5, 5, 2, 4, 5, 2, 4, 6, 8, 6, 2, 6, 9, 7, 6, 6, 3, 6, 6, 6, 7, 4};
TeamScore[3 + NumTeams] = {1, 1, 1, 1, 7, 5, 0, 13, 3, 7, 6, 8, 8, 4, 5, 0, 4, 3, 15, 6, 1, 4, 6, 4, 2, 4,
   6, 6, 2, 9, 6, 8, 5, 6, 2, 1, 3, 0, 3, 8, 4, 13, 17, 3, 1, 3, 2, 5, 6, 4, 6, 4, 3, 1, 8, 3, 7, 2, 0, 2, 0,
   7, 9, 6, 4, 3, 8, 4, 4, 3, 9, 5, 4, 10, 5, 6, 5, 9, 4, 5, 4, 13, 4, 13, 10, 10, 4, 8, 8, 6, 5, 3, 15, 2,
   4, 6, 2, 9, 4, 3, 8, 2, 8, 17, 4, 2, 6, 9, 5, 2, 7, 7, 4, 4, 6, 5, 6, 5, 2, 8, 6, 8, 9, 7, 1, 1, 1, 1, 5,
   0, 8, 3, 5, 13, 8, 2, 6, 8, 11, 5, 4, 4, 0, 5, 6, 5, 2, 2, 3, 2, 11, 5, 18, 5, 4, 5, 4, 5, 10, 3, 8, 3};
TeamScore[4] = {1, 1, 1, 3, 1, 1, 9, 2, 1, 16, 3, 4, 5, 4, 2, 5, 2, 4, 2, 4, 6, 2, 8, 15, 6, 5, 2, 5, 3,
   2, 3, 6, 8, 5, 4, 8, 7, 3, 0, 3, 6, 2, 6, 2, 3, 3, 4, 3, 6, 5, 3, 7, 5, 5, 0, 2, 0, 3, 6, 5, 4, 4, 0,
   7, 9, 1, 4, 0, 2, 5, 7, 2, 8, 1, 6, 5, 7, 14, 0, 4, 3, 3, 5, 8, 0, 2, 12, 5, 4, 0, 9, 5, 0, 4, 3, 0,
   2, 4, 4, 5, 5, 1, 4, 10, 8, 2, 8, 1, 9, 7, 8, 0, 4, 2, 4, 8, 4, 5, 2, 1, 6, 4, 3, 8, 8, 2, 1, 3, 0, 6,
   6, 12, 3, 0, 4, 2, 2, 6, 8, 5, 0, 5, 7, 6, 9, 5, 2, 2, 9, 3, 4, 8, 0, 2, 2, 15, 1, 6, 5, 5, 5, 8};
TeamScore[4 + NumTeams] = {4, 3, 5, 5, 5, 5, 7, 4, 6, 5, 2, 3, 2, 3, 4, 0, 1, 1, 9, 6, 4, 0, 2, 3, 1, 8,
   1, 6, 2, 3, 1, 2, 2, 3, 5, 2, 4, 0, 6, 9, 5, 6, 5, 3, 5, 5, 0, 6, 7, 0, 7, 0, 11, 4, 3, 8, 7, 2, 9, 1,
   1, 3, 7, 5, 6, 2, 0, 3, 4, 1, 4, 1, 4, 5, 3, 1, 2, 10, 5, 3, 4, 5, 1, 3, 7, 3, 5, 1, 5, 1, 6, 9, 1, 5,
   2, 4, 1, 10, 5, 0, 7, 6, 13, 8, 0, 3, 1, 3, 1, 6, 4, 8, 5, 1, 0, 7, 1, 1, 9, 3, 2, 0, 2, 0, 7, 5, 2, 2,
   2, 1, 5, 0, 7, 2, 1, 7, 3, 3, 1, 1, 8, 4, 2, 5, 1, 2, 4, 6, 2, 4, 3, 5, 5, 4, 4, 8, 5, 2, 2, 2, 3, 7};
TeamScore[5] = {13, 6, 3, 7, 5, 1, 3, 5, 1, 7, 2, 8, 7, 1, 1, 1, 6, 2, 6, 4, 0, 6, 10, 6, 5, 5, 4, 2, 2,
   3, 1, 7, 0, 2, 5, 7, 9, 2, 9, 11, 4, 5, 3, 2, 7, 2, 7, 4, 3, 1, 4, 9, 13, 11, 3, 9, 8, 3, 7, 2, 8, 9,
   2, 1, 4, 1, 6, 4, 3, 3, 4, 1, 0, 1, 1, 5, 6, 5, 2, 6, 2, 2, 6, 3, 7, 9, 2, 4, 4, 11, 5, 7, 16, 7, 1, 2,
   6, 11, 7, 2, 4, 3, 4, 3, 8, 3, 0, 7, 3, 1, 6, 5, 2, 7, 1, 8, 3, 1, 11, 5, 5, 13, 5, 7, 0, 1, 1, 4, 4,
   6, 1, 5, 0, 7, 5, 13, 8, 2, 4, 3, 1, 0, 11, 7, 0, 5, 6, 6, 5, 5, 6, 3, 3, 6, 2, 4, 5, 2, 2, 3, 1, 3};
TeamScore[5 + NumTeams] = {3, 1, 4, 6, 3, 2, 2, 6, 3, 8, 3, 3, 6, 4, 8, 9, 5, 6, 4, 6, 2, 4, 3, 7, 2, 3, 5,
   5, 3, 2, 3, 4, 9, 5, 10, 6, 3, 0, 3, 3, 2, 6, 2, 5, 5, 3, 3, 5, 7, 3, 2, 8, 4, 1, 6, 13, 4, 5, 4, 3, 5,
   8, 3, 5, 16, 14, 5, 1, 4, 2, 0, 2, 2, 5, 5, 4, 3, 0, 4, 7, 1, 6, 7, 5, 4, 7, 3, 6, 5, 7, 4, 1, 7, 1, 4,
   7, 5, 6, 5, 12, 5, 0, 12, 0, 5, 2, 3, 3, 1, 9, 7, 4, 6, 2, 4, 4, 10, 5, 2, 4, 6, 7, 1, 0, 2, 5, 0, 6, 3,
   9, 6, 6, 12, 3, 6, 0, 6, 3, 6, 9, 0, 14, 10, 4, 2, 4, 5, 18, 4, 4, 7, 0, 2, 10, 7, 3, 1, 6, 5, 4, 2, 2};

(* Team 6 = Twins*)
(* Team 7 = White Sox*)
(* Team 8 = Indians*)
(* Team 9 = Tigers*)
(* Team 10 = Royals*)

TeamScore[6] = {3, 1, 4, 3, 5, 3, 2, 0, 3, 4, 5, 3, 2, 3, 4, 5, 0, 4, 3, 10, 4, 2, 3, 1, 3, 2, 3, 1, 3, 9,
   0, 5, 1, 2, 7, 0, 3, 3, 2, 2, 4, 11, 7, 6, 2, 7, 4, 0, 5, 1, 5, 5, 7, 2, 8, 5, 7, 6, 6, 0, 3, 5, 3,
   8, 6, 4, 1, 6, 1, 5, 9, 1, 1, 3, 1, 2, 0, 6, 1, 6, 7, 9, 7, 3, 5, 6, 8, 3, 6, 8, 1, 4, 4, 2, 3, 2, 7,
   2, 2, 4, 2, 6, 9, 7, 1, 9, 3, 3, 1, 11, 1, 3, 1, 0, 6, 3, 5, 2, 1, 9, 1, 6, 4, 1, 9, 0, 1, 1, 1, 1,
   1, 4, 11, 0, 6, 7, 13, 6, 1, 1, 0, 0, 5, 4, 2, 1, 0, 3, 6, 4, 5, 4, 4, 4, 3, 5, 2, 6, 6, 3, 7, 1};
TeamScore[6 + NumTeams] = {13, 6, 3, 4, 4, 4, 1, 1, 5, 3, 10, 4, 5, 4, 2, 3, 11, 5, 1, 3, 3, 8, 15, 6, 4,
   11, 10, 0, 2, 2, 4, 9, 2, 10, 9, 2, 9, 11, 5, 1, 3, 1, 8, 9, 3, 8, 2, 3, 6, 0, 6, 6, 8, 4, 2, 2, 2, 0, 4,
   1, 2, 4, 9, 1, 1, 1, 0, 5, 0, 4, 2, 5, 2, 4, 11, 6, 15, 4, 0, 2, 8, 7, 0, 2, 12, 2, 5, 4, 3, 4, 2, 3, 3,
   5, 6, 1, 5, 6, 8, 1, 5, 20, 8, 2, 4, 5, 8, 7, 5, 4, 7, 5, 6, 7, 8, 4, 2, 3, 3, 6, 7, 5, 8, 8, 4, 3, 4, 8,
   6, 6, 8, 6, 4, 3, 8, 6, 5, 10, 4, 2, 4, 3, 4, 8, 3, 2, 4, 7, 7, 10, 6, 6, 5, 5, 2, 6, 8, 7, 4, 7, 4, 0};
TeamScore[7] = {15, 8, 1, 6, 10, 5, 7, 4, 6, 1, 6, 4, 3, 2, 2, 0, 1, 1, 9, 3, 0, 0, 2, 3, 1, 3, 4, 2, 4,
   6, 0, 2, 2, 6, 5, 8, 2, 6, 4, 2, 4, 0, 4, 1, 8, 4, 9, 8, 0, 8, 1, 3, 2, 8, 4, 7, 10, 7, 6, 2, 3, 3,
   5, 4, 9, 5, 3, 5, 1, 0, 1, 6, 8, 3, 3, 4, 5, 3, 1, 2, 3, 6, 6, 1, 1, 5, 3, 1, 2, 5, 4, 3, 8, 5, 3, 5,
   2, 1, 3, 4, 6, 4, 2, 3, 2, 3, 2, 0, 7, 2, 5, 6, 7, 7, 4, 4, 6, 1, 5, 6, 8, 1, 2, 4, 3, 10, 4, 0, 4,
   3, 9, 3, 8, 6, 1, 8, 2, 2, 4, 3, 4, 8, 4, 7, 3, 4, 0, 5, 2, 6, 3, 10, 3, 5, 8, 2, 1, 6, 1, 4, 2, 2};
TeamScore[7 + NumTeams] = {10, 3, 7, 7, 7, 1, 9, 2, 1, 2, 5, 7, 4, 7, 4, 5, 2, 4, 2, 9, 9, 3, 0, 2, 3, 12,
   10, 6, 6, 2, 1, 3, 3, 0, 2, 0, 6, 4, 3, 6, 3, 4, 3, 0, 2, 6, 2, 3, 4, 6, 2, 1, 4, 9, 13, 3, 7, 4, 4, 4,
   7, 1, 1, 7, 4, 7, 2, 4, 4, 1, 4, 2, 2, 6, 2, 3, 9, 0, 2, 3, 2, 4, 4, 0, 3, 4, 5, 4, 6, 8, 3, 6, 2, 0, 4,
   2, 4, 2, 0, 2, 3, 5, 1, 1, 10, 5, 3, 6, 18, 7, 3, 1, 0, 6, 3, 6, 3, 5, 4, 2, 7, 4, 4, 7, 2, 0, 5, 8, 2,
   0, 3, 0, 6, 7, 8, 9, 18, 1, 0, 0, 5, 1, 8, 3, 7, 14, 5, 6, 7, 7, 10, 5, 4, 4, 4, 11, 11, 3, 2, 3, 1, 3};
TeamScore[8] = {10, 3, 7, 3, 8, 1, 12, 2, 6, 4, 0, 3, 8, 8, 4, 7, 4, 7, 2, 3, 3, 9, 7, 8, 9, 3, 5, 4, 1,
   4, 1, 4, 5, 5, 2, 4, 5, 19, 7, 0, 2, 5, 2, 12, 3, 2, 2, 0, 7, 0, 1, 6, 13, 4, 2, 0, 0, 4, 1, 2, 7, 0,
   1, 1, 0, 6, 2, 5, 5, 5, 7, 3, 4, 3, 0, 1, 5, 4, 6, 8, 3, 5, 6, 2, 5, 5, 7, 4, 1, 8, 6, 5, 3, 5, 6, 1,
   5, 0, 2, 3, 1, 1, 0, 5, 3, 9, 2, 3, 7, 7, 7, 3, 3, 10, 3, 3, 3, 7, 4, 4, 1, 1, 7, 2, 7, 7, 2, 2, 8, 1,
   2, 6, 4, 0, 5, 1, 9, 2, 1, 6, 1, 8, 3, 7, 4, 1, 4, 7, 10, 6, 6, 4, 4, 4, 11, 6, 8, 7, 4, 0, 6, 4};
TeamScore[8 + NumTeams] = {15, 8, 1, 1, 4, 0, 3, 1, 4, 0, 2, 4, 2, 3, 2, 3, 5, 5, 3, 10, 4, 4, 2, 2, 5, 2,
   4, 1, 3, 3, 2, 3, 6, 4, 8, 7, 4, 1, 3, 1, 8, 4, 1, 4, 2, 4, 14, 5, 3, 7, 11, 3, 9, 7, 11, 4, 2, 6, 0, 3,
   11, 4, 9, 0, 4, 4, 6, 1, 1, 2, 8, 4, 3, 4, 1, 3, 4, 6, 2, 2, 1, 7, 3, 9, 3, 4, 11, 5, 7, 4, 5, 6, 8, 2, 3,
   2, 7, 3, 4, 2, 2, 3, 12, 2, 5, 6, 3, 4, 3, 8, 5, 5, 2, 3, 4, 2, 1, 8, 1, 2, 4, 10, 8, 3, 5, 12, 9, 1, 7,
   2, 1, 2, 3, 7, 4, 5, 6, 4, 10, 8, 8, 4, 7, 3, 10, 9, 7, 6, 4, 5, 12, 3, 5, 8, 2, 5, 2, 6, 6, 14, 9, 5};
TeamScore[9] = {3, 6, 10, 1, 7, 5, 5, 1, 5, 0, 5, 3, 3, 8, 2, 1, 8, 3, 3, 9, 9, 3, 3, 1, 2, 5, 2, 4, 3,
   4, 4, 6, 4, 9, 5, 10, 10, 9, 3, 3, 2, 0, 3, 1, 2, 2, 6, 7, 1, 3, 3, 3, 6, 8, 4, 4, 4, 7, 13, 8, 3, 4,
   2, 8, 3, 2, 4, 4, 6, 6, 4, 9, 0, 1, 7, 6, 6, 8, 4, 3, 9, 5, 3, 3, 6, 1, 0, 5, 3, 6, 6, 2, 2, 0, 4, 8,
   5, 6, 8, 1, 5, 3, 5, 1, 7, 12, 1, 3, 6, 5, 2, 4, 4, 3, 2, 3, 4, 5, 6, 5, 6, 7, 5, 4, 10, 8, 5, 2, 2, 2,
   8, 6, 4, 5, 2, 5, 8, 8, 9, 18, 4, 10, 8, 8, 3, 2, 14, 5, 6, 1, 3, 3, 3, 2, 6, 5, 4, 5, 10, 14, 9, 5};
TeamScore[9 + NumTeams] = {6, 10, 7, 5, 3, 9, 2, 3, 9, 2, 4, 2, 0, 4, 6, 5, 3, 13, 2, 3, 0, 0, 7, 10, 7, 9,
   3, 5, 5, 2, 0, 3, 7, 0, 2, 5, 2, 7, 1, 0, 4, 1, 4, 10, 6, 0, 3, 6, 14, 6, 4, 0, 5, 7, 2, 6, 2, 3, 7, 1, 7,
   1, 3, 1, 7, 1, 0, 6, 2, 13, 5, 1, 4, 6, 5, 7, 0, 3, 2, 14, 16, 2, 4, 15, 3, 5, 1, 4, 1, 4, 13, 1, 8, 5,
   3, 3, 7, 2, 2, 4, 2, 6, 4, 2, 12, 2, 5, 2, 5, 4, 5, 3, 3, 4, 3, 10, 3, 4, 5, 8, 9, 1, 6, 1, 1, 7, 2, 1, 3,
   0, 1, 4, 11, 9, 1, 4, 11, 1, 8, 2, 2, 1, 6, 4, 2, 1, 4, 0, 5, 6, 1, 5, 0, 10, 3, 6, 3, 6, 6, 0, 6, 4};
TeamScore[10] = {2, 2, 5, 12, 7, 7, 2, 3, 9, 3, 10, 5, 6, 7, 2, 3, 5, 5, 3, 6, 1, 7, 4, 2, 2, 4, 11, 10,
   6, 2, 9, 2, 4, 2, 1, 4, 11, 1, 0, 1, 3, 4, 2, 3, 0, 8, 3, 2, 5, 12, 1, 6, 8, 7, 2, 2, 2, 2, 0, 3, 5, 8,
   3, 4, 5, 9, 7, 1, 4, 5, 4, 4, 2, 2, 3, 4, 3, 6, 3, 2, 1, 0, 6, 16, 4, 5, 4, 1, 4, 13, 1, 4, 2, 3, 3, 2,
   4, 2, 10, 5, 0, 3, 9, 5, 4, 12, 2, 5, 2, 6, 9, 3, 3, 4, 1, 0, 7, 1, 5, 4, 2, 4, 7, 5, 3, 1, 9, 1, 6, 3,
   9, 1, 7, 2, 9, 1, 4, 11, 4, 5, 6, 11, 7, 0, 1, 3, 4, 2, 4, 7, 7, 7, 10, 5, 10, 3, 11, 3, 2, 7, 4, 0};
TeamScore[10 + NumTeams] = {4, 1, 4, 9, 6, 10, 5, 1, 5, 4, 5, 1, 5, 0, 3, 7, 4, 7, 2, 11, 3, 8, 9, 7, 8,
   3, 2, 3, 5, 3, 1, 3, 3, 5, 3, 3, 5, 3, 3, 19, 7, 5, 1, 0, 3, 9, 5, 9, 6, 7, 10, 7, 10, 3, 0, 8, 5, 7, 6,
   2, 8, 9, 2, 2, 7, 0, 4, 2, 8, 4, 5, 5, 7, 3, 5, 6, 2, 3, 4, 4, 4, 9, 9, 8, 5, 3, 1, 3, 6, 6, 2, 8, 1, 4,
   4, 5, 2, 1, 4, 4, 5, 1, 13, 12, 3, 0, 5, 3, 8, 2, 4, 4, 4, 3, 2, 4, 8, 4, 1, 5, 6, 7, 9, 4, 4, 7, 4, 6,
   4, 4, 6, 2, 8, 1, 5, 2, 5, 8, 5, 1, 9, 6, 4, 7, 4, 7, 2, 1, 0, 3, 2, 6, 3, 10, 2, 6, 1, 6, 1, 3, 7, 1};


(* AL WEST *)
(* Team 11 = Angels*)
(* Team 12 = Athletics*)
(* Team 13 = Rangers*)
(* Team 14 = Mariners*)

TeamScore[11] = {4, 1, 4, 9, 5, 5, 2, 6, 3, 0, 2, 4, 4, 7, 4, 1, 15, 4, 2, 3, 0, 0, 5, 8, 1, 8, 1, 6, 5, 3, 5,
   11, 2, 3, 6, 0, 6, 4, 1, 3, 4, 4, 0, 0, 1, 9, 4, 4, 4, 1, 4, 3, 6, 0, 6, 10, 3, 0, 3, 2, 3, 1, 1, 3,
   2, 7, 0, 6, 4, 1, 4, 1, 7, 2, 2, 6, 8, 6, 2, 4, 11, 1, 0, 7, 3, 5, 1, 4, 5, 4, 9, 4, 3, 4, 3, 1, 0,
   9, 1, 6, 2, 9, 2, 2, 3, 12, 2, 5, 2, 5, 4, 7, 1, 1, 2, 6, 3, 5, 5, 2, 4, 4, 3, 3, 2, 8, 9, 7, 5, 8,
   7, 8, 5, 3, 13, 1, 4, 5, 10, 4, 7, 1, 3, 2, 6, 5, 3, 6, 4, 3, 2, 11, 2, 10, 7, 3, 1, 4, 5, 3, 3, 1};
TeamScore[11 + NumTeams] = {2, 2, 5, 12, 3, 1, 3, 5, 1, 4, 0, 3, 3, 2, 2, 7, 4, 1, 4, 4, 5, 7, 0, 3, 2, 5,
   2, 5, 9, 7, 3, 0, 1, 4, 5, 8, 2, 6, 4, 2, 5, 5, 14, 3, 2, 0, 5, 1, 1, 6, 1, 4, 5, 1, 5, 8, 7, 2, 2, 3,
   5, 5, 4, 4, 4, 5, 9, 3, 0, 3, 3, 6, 3, 1, 5, 5, 3, 1, 3, 3, 5, 0, 5, 1, 1, 1, 0, 5, 1, 3, 3, 2, 5, 2, 4,
   9, 7, 8, 0, 1, 3, 3, 3, 1, 1, 7, 12, 1, 3, 1, 11, 1, 0, 5, 1, 4, 9, 6, 1, 11, 5, 8, 7, 4, 1, 3, 8, 1, 4,
   0, 11, 4, 9, 5, 6, 2, 3, 13, 6, 1, 3, 2, 1, 1, 0, 6, 6, 3, 1, 8, 6, 2, 3, 6, 2, 4, 3, 2, 6, 4, 10, 3};
TeamScore[12] = {2, 2, 7, 6, 3, 2, 1, 1, 5, 2, 5, 7, 0, 4, 6, 5, 5, 3, 0, 0, 9, 5, 0, 3, 2, 3, 2, 7, 5,
   1, 3, 3, 3, 3, 5, 7, 2, 3, 6, 3, 5, 14, 3, 1, 1, 0, 4, 1, 6, 1, 4, 6, 4, 6, 0, 3, 2, 6, 8, 3, 2, 0,
   2, 4, 7, 2, 4, 4, 2, 8, 5, 4, 2, 7, 2, 1, 0, 4, 1, 1, 0, 4, 5, 2, 7, 1, 2, 2, 0, 5, 6, 0, 5, 2, 4, 9,
   3, 7, 7, 4, 5, 7, 6, 13, 8, 5, 8, 7, 4, 2, 4, 4, 8, 5, 4, 4, 10, 1, 1, 6, 2, 8, 6, 0, 2, 5, 0, 6, 6,
   9, 15, 3, 0, 1, 2, 3, 7, 9, 3, 8, 6, 4, 7, 4, 8, 1, 6, 3, 1, 6, 1, 5, 0, 2, 2, 4, 3, 2, 6, 2, 7, 2};
TeamScore[12 + NumTeams] = {6, 5, 1, 7, 5, 1, 2, 0, 3, 1, 6, 4, 3, 8, 2, 1, 0, 5, 1, 4, 1, 2, 5, 8, 1, 1,
   11, 2, 4, 4, 1, 4, 2, 4, 2, 2, 7, 4, 2, 4, 4, 0, 4, 11, 2, 3, 5, 4, 1, 4, 3, 2, 2, 4, 5, 10, 4, 8, 9, 6,
   4, 4, 3, 9, 5, 3, 5, 7, 1, 4, 2, 2, 1, 3, 3, 4, 1, 1, 3, 0, 3, 5, 4, 4, 2, 2, 4, 0, 6, 8, 7, 2, 3, 4, 3,
   1, 8, 5, 17, 3, 7, 5, 1, 4, 10, 9, 3, 3, 8, 4, 7, 8, 0, 4, 1, 8, 3, 9, 7, 7, 6, 4, 5, 7, 0, 1, 1, 5, 4,
   22, 5, 9, 4, 2, 6, 4, 0, 2, 0, 5, 11, 7, 0, 13, 7, 8, 3, 6, 4, 1, 3, 3, 3, 7, 3, 3, 1, 4, 5, 4, 0, 0};
TeamScore[13] = {9, 12, 5, 6, 3, 7, 0, 13, 3, 2, 4, 2, 5, 2, 5, 7, 4, 1, 11, 3, 8, 4, 3, 7, 2, 1, 11, 2,
   4, 3, 5, 1, 1, 7, 5, 2, 7, 4, 2, 5, 4, 3, 5, 1, 2, 0, 2, 4, 6, 2, 7, 10, 7, 11, 4, 3, 7, 11, 4, 2, 7, 1,
   7, 4, 9, 1, 1, 4, 4, 2, 6, 5, 2, 8, 5, 3, 8, 5, 5, 7, 3, 0, 15, 5, 4, 13, 4, 13, 6, 8, 7, 2, 5, 4, 5, 3,
   7, 8, 0, 12, 5, 0, 20, 8, 2, 4, 2, 3, 3, 5, 4, 5, 8, 5, 5, 9, 7, 3, 9, 7, 7, 8, 7, 4, 1, 7, 2, 0, 4, 5,
   2, 0, 11, 4, 9, 2, 1, 7, 10, 7, 11, 1, 8, 4, 13, 7, 8, 10, 9, 7, 0, 7, 3, 7, 3, 3, 5, 7, 12, 4, 10, 3};
TeamScore[13 + NumTeams] = {5, 5, 1, 4, 2, 3, 5, 1, 0, 0, 5, 3, 3, 5, 6, 1, 15, 4, 6, 1, 7, 6, 10, 6, 5, 3,
   2, 7, 5, 4, 2, 3, 4, 5, 12, 7, 2, 1, 3, 4, 0, 4, 4, 2, 3, 2, 0, 0, 8, 1, 12, 1, 6, 5, 5, 0, 4, 2, 0, 0,
   13, 8, 3, 5, 3, 8, 6, 12, 12, 3, 2, 4, 4, 3, 4, 5, 1, 14, 8, 3, 2, 7, 5, 9, 6, 4, 2, 5, 0, 5, 6, 0, 0, 0,
   1, 1, 0, 9, 1, 2, 4, 3, 6, 9, 7, 1, 3, 0, 7, 6, 5, 2, 7, 7, 3, 2, 6, 4, 1, 1, 6, 4, 3, 3, 2, 4, 3, 10, 0,
   11, 13, 6, 7, 8, 5, 0, 4, 2, 0, 12, 4, 5, 0, 5, 4, 8, 1, 4, 1, 4, 4, 6, 0, 2, 2, 4, 3, 3, 5, 3, 3, 1};
TeamScore[14] = {6, 5, 1, 4, 2, 3, 3, 1, 4, 8, 3, 3, 1, 5, 0, 3, 3, 13, 2, 1, 4, 1, 2, 7, 10, 7, 5, 2,
   2, 4, 2, 3, 3, 0, 2, 6, 2, 1, 4, 5, 1, 3, 2, 4, 4, 6, 8, 2, 3, 4, 5, 1, 4, 3, 1, 8, 7, 2, 9, 1, 1, 7,
   1, 3, 1, 7, 3, 0, 3, 4, 1, 2, 5, 1, 0, 5, 2, 2, 1, 4, 3, 6, 0, 3, 2, 4, 0, 1, 3, 3, 2, 0, 0, 1, 1, 5,
   6, 5, 4, 1, 8, 3, 1, 9, 0, 3, 1, 8, 4, 7, 0, 5, 1, 2, 6, 4, 4, 5, 5, 6, 7, 1, 2, 0, 7, 3, 5, 12, 9,
   2, 0, 3, 5, 6, 2, 3, 2, 0, 5, 3, 2, 1, 4, 7, 2, 1, 3, 2, 2, 4, 6, 0, 12, 5, 5, 2, 3, 3, 5, 4, 0, 0};
TeamScore[14 + NumTeams] = {2, 2, 7, 6, 3, 7, 12, 2, 6, 7, 2, 8, 5, 6, 7, 2, 8, 3, 3, 0, 0, 9, 5, 3, 1, 2,
   4, 0, 3, 3, 5, 1, 2, 6, 5, 7, 4, 2, 5, 2, 2, 0, 1, 1, 0, 1, 7, 4, 0, 3, 4, 7, 3, 2, 2, 2, 0, 3, 6, 3, 5,
   4, 4, 2, 8, 3, 6, 4, 1, 2, 5, 0, 6, 2, 1, 1, 4, 1, 3, 5, 5, 0, 1, 1, 1, 2, 2, 5, 4, 9, 4, 5, 4, 5, 3, 6,
   11, 7, 7, 3, 12, 10, 4, 2, 8, 2, 8, 4, 2, 4, 1, 1, 2, 9, 7, 3, 6, 4, 3, 5, 13, 5, 3, 8, 8, 2, 7, 7, 2,
   4, 3, 9, 3, 13, 1, 4, 9, 3, 8, 7, 1, 3, 1, 3, 4, 2, 9, 3, 1, 0, 7, 3, 6, 4, 4, 3, 5, 7, 12, 2, 7, 2};

(* NL EAST *)
(* Team 15 = Braves*)
(* Team 16 = Phillies*)
(* Team 17 = Marlins*)
(* Team 18 = Mets*)
(* Team 19 = Expos*)
TeamScore[15] = {2, 3, 11, 2, 0, 4, 2, 6, 2, 0, 5, 1, 5, 4, 4, 2, 2, 10, 1, 3, 4, 5, 9, 3, 8, 7, 3, 2, 6,
   6, 8, 8, 2, 5, 0, 5, 6, 3, 6, 4, 5, 3, 3, 3, 4, 1, 0, 5, 1, 2, 4, 1, 7, 2, 2, 4, 4, 6, 0, 4, 1, 3, 3,
   11, 6, 4, 3, 3, 0, 9, 2, 4, 4, 2, 5, 5, 2, 10, 1, 3, 5, 5, 4, 5, 4, 4, 5, 9, 6, 2, 4, 1, 11, 2, 9, 7,
   3, 2, 9, 6, 2, 3, 1, 4, 2, 2, 5, 5, 1, 3, 3, 6, 4, 7, 6, 8, 4, 6, 10, 4, 5, 5, 2, 5, 1, 4, 8, 1, 3,
   5, 2, 8, 0, 2, 3, 5, 6, 1, 4, 0, 3, 2, 6, 5, 3, 3, 3, 4, 7, 4, 2, 1, 5, 5, 4, 0, 7, 1, 0, 2, 1, 3};
TeamScore[15 + NumTeams] = {0, 6, 2, 1, 1, 5, 4, 3, 10, 3, 0, 5, 6, 2, 0, 3, 4, 1, 6, 5, 1, 2, 6, 5, 2, 0,
   5, 3, 5, 2, 3, 0, 1, 0, 3, 2, 7, 7, 5, 5, 3, 2, 2, 1, 5, 2, 9, 4, 4, 0, 2, 5, 6, 1, 3, 5, 3, 3, 5, 6, 0,
   2, 2, 4, 3, 1, 8, 4, 4, 8, 6, 5, 2, 0, 1, 1, 11, 1, 4, 1, 4, 3, 0, 4, 5, 1, 3, 1, 3, 3, 1, 14, 1, 5, 8,
   4, 12, 3, 6, 4, 11, 4, 3, 3, 1, 5, 0, 1, 3, 5, 9, 4, 1, 11, 5, 5, 3, 2, 4, 8, 6, 4, 1, 7, 0, 2, 1, 0,
   0, 4, 3, 3, 6, 9, 1, 2, 8, 2, 3, 9, 6, 3, 5, 1, 4, 4, 6, 5, 1, 1, 12, 0, 7, 6, 0, 4, 4, 4, 3, 4, 7, 4};
TeamScore[16] = {5, 9, 7, 1, 10, 11, 3, 10, 3, 4, 3, 4, 3, 3, 3, 0, 4, 3, 2, 4, 3, 0, 5, 8, 10, 2, 1, 4,
   7, 7, 0, 3, 2, 6, 1, 5, 5, 3, 2, 1, 1, 2, 1, 3, 2, 0, 10, 3, 5, 10, 6, 5, 5, 5, 2, 1, 1, 3, 7, 3, 2, 2,
   3, 7, 7, 4, 9, 8, 5, 3, 2, 5, 0, 10, 4, 2, 1, 1, 3, 5, 2, 2, 7, 5, 4, 1, 14, 6, 3, 1, 14, 7, 2, 8, 1,
   4, 9, 3, 8, 5, 4, 7, 1, 1, 10, 7, 6, 4, 5, 8, 3, 9, 2, 1, 5, 2, 9, 2, 11, 2, 9, 4, 4, 5, 4, 10, 9, 4,
   5, 3, 9, 3, 6, 5, 4, 4, 9, 6, 3, 7, 5, 3, 2, 1, 2, 1, 3, 2, 2, 9, 0, 3, 3, 0, 5, 1, 1, 3, 9, 4, 7, 4};
TeamScore[16 + NumTeams] = {4, 4, 3, 7, 7, 0, 6, 2, 0, 7, 2, 0, 4, 2, 6, 9, 3, 0, 0, 2, 1, 4, 7, 4, 3, 1,
   2, 1, 4, 3, 5, 0, 5, 4, 2, 3, 4, 5, 3, 3, 2, 1, 7, 2, 0, 2, 3, 6, 4, 4, 4, 2, 9, 4, 10, 2, 2, 6, 3, 1,
   6, 0, 4, 5, 1, 3, 1, 1, 4, 0, 4, 1, 2, 2, 0, 12, 0, 4, 1, 0, 1, 5, 6, 3, 7, 0, 2, 7, 2, 4, 1, 2, 11, 5,
   6, 2, 1, 1, 6, 3, 5, 2, 2, 4, 3, 4, 5, 3, 0, 6, 0, 2, 1, 3, 3, 1, 8, 4, 3, 3, 2, 1, 8, 0, 5, 0, 4, 7,
   6, 2, 0, 0, 4, 3, 8, 5, 0, 3, 2, 2, 3, 2, 3, 5, 5, 0, 1, 1, 4, 2, 5, 4, 4, 3, 7, 6, 2, 6, 4, 2, 1, 3};
TeamScore[17] = {6, 4, 2, 3, 7, 3, 4, 7, 1, 0, 5, 6, 4, 2, 6, 6, 9, 4, 1, 6, 5, 4, 4, 7, 3, 9, 6, 5, 8,
   3, 2, 2, 8, 4, 2, 3, 6, 1, 4, 2, 5, 1, 5, 5, 0, 5, 7, 1, 2, 6, 0, 4, 5, 5, 5, 2, 5, 2, 0, 2, 2, 6, 5,
   1, 9, 1, 1, 4, 0, 1, 4, 1, 1, 5, 5, 1, 4, 1, 0, 3, 5, 5, 9, 6, 0, 2, 7, 5, 6, 6, 5, 6, 1, 13, 7, 4,
   0, 3, 3, 6, 8, 5, 11, 7, 5, 0, 1, 3, 7, 4, 4, 2, 1, 4, 5, 3, 2, 2, 0, 2, 4, 6, 5, 1, 3, 1, 3, 6, 6,
   2, 6, 1, 1, 6, 2, 5, 3, 8, 5, 9, 4, 0, 13, 3, 4, 5, 1, 1, 1, 1, 3, 4, 3, 6, 0, 4, 1, 4, 5, 4, 3, 1};
TeamScore[17 + NumTeams] = {2, 6, 9, 2, 4, 5, 3, 5, 7, 5, 1, 5, 3, 3, 0, 0, 5, 1, 3, 3, 4, 2, 5, 6, 4, 5,
   5, 7, 7, 6, 3, 5, 0, 6, 1, 5, 5, 0, 8, 1, 7, 5, 3, 3, 4, 1, 6, 0, 3, 1, 8, 15, 2, 6, 6, 3, 6, 7, 1, 3,
   3, 4, 9, 5, 12, 9, 8, 5, 3, 5, 7, 2, 2, 2, 6, 5, 2, 2, 1, 0, 4, 15, 5, 4, 1, 14, 6, 0, 3, 1, 4, 3, 2, 3,
   5, 1, 4, 14, 5, 7, 5, 4, 2, 5, 2, 5, 5, 1, 3, 3, 7, 3, 2, 8, 8, 4, 6, 1, 3, 5, 7, 5, 12, 3, 4, 14, 4, 8,
   5, 3, 5, 2, 5, 0, 3, 7, 5, 4, 4, 3, 7, 1, 4, 0, 1, 4, 7, 4, 3, 2, 0, 1, 4, 5, 4, 0, 4, 6, 9, 6, 2, 3};
TeamScore[18] = {2, 6, 9, 7, 7, 0, 2, 8, 3, 6, 4, 5, 4, 2, 0, 3, 1, 3, 9, 4, 6, 8, 6, 6, 3, 3, 1, 2, 6,
   0, 5, 6, 4, 2, 1, 4, 9, 6, 3, 7, 1, 3, 1, 2, 3, 3, 1, 7, 3, 4, 2, 9, 7, 1, 3, 9, 3, 5, 6, 2, 6, 4, 8,
   2, 7, 1, 4, 4, 8, 3, 6, 3, 3, 3, 4, 1, 14, 8, 14, 16, 2, 1, 2, 3, 5, 6, 5, 0, 5, 1, 2, 2, 11, 5, 1, 4,
   6, 2, 7, 5, 4, 4, 8, 8, 10, 8, 0, 2, 3, 3, 1, 11, 5, 9, 5, 5, 2, 3, 4, 3, 5, 1, 7, 1, 9, 2, 0, 4, 7,
   6, 2, 5, 0, 3, 7, 7, 7, 6, 3, 7, 1, 5, 1, 5, 4, 6, 2, 2, 0, 1, 12, 0, 7, 6, 5, 8, 2, 6, 4, 5, 4, 3};
TeamScore[18 + NumTeams] = {6, 4, 2, 1, 10, 11, 6, 4, 7, 7, 5, 6, 9, 4, 4, 2, 6, 4, 1, 1, 4, 4, 4, 3, 4,
   10, 2, 1, 7, 2, 2, 3, 2, 4, 2, 3, 5, 4, 7, 4, 2, 0, 0, 1, 7, 9, 11, 4, 9, 6, 5, 5, 3, 5, 9, 8, 6, 0, 4, 1,
   7, 1, 1, 3, 0, 3, 3, 0, 9, 4, 1, 7, 7, 2, 1, 8, 5, 5, 3, 9, 5, 5, 5, 2, 2, 0, 3, 6, 2, 3, 4, 7, 2, 8, 4,
   2, 5, 6, 6, 8, 5, 2, 6, 2, 9, 5, 3, 3, 7, 4, 4, 7, 6, 8, 4, 9, 3, 4, 6, 5, 4, 6, 3, 6, 11, 6, 10, 9, 4,
   0, 1, 1, 6, 2, 5, 3, 8, 3, 9, 4, 0, 6, 5, 4, 5, 10, 3, 3, 2, 10, 2, 1, 5, 11, 6, 6, 1, 3, 9, 6, 5, 0};
TeamScore[19] = {0, 6, 2, 2, 4, 5, 6, 4, 7, 7, 2, 0, 4, 8, 5, 8, 3, 0, 2, 6, 2, 4, 3, 4, 3, 1, 5, 2, 1,
   4, 3, 3, 5, 0, 7, 7, 5, 5, 0, 8, 4, 0, 0, 17, 3, 1, 3, 6, 4, 2, 1, 4, 4, 10, 2, 6, 0, 0, 9, 4, 2, 1,
   3, 2, 2, 2, 8, 10, 7, 8, 4, 4, 6, 2, 1, 9, 0, 2, 3, 5, 0, 2, 3, 4, 2, 5, 3, 5, 9, 2, 1, 2, 1, 5, 8,
   5, 6, 2, 7, 6, 1, 2, 5, 2, 5, 3, 3, 5, 9, 4, 3, 5, 7, 3, 3, 2, 3, 4, 3, 6, 1, 3, 8, 0, 5, 4, 0, 2, 1,
   3, 3, 4, 9, 1, 2, 3, 8, 3, 7, 3, 4, 4, 3, 8, 3, 3, 2, 10, 0, 1, 4, 4, 3, 7, 6, 4, 4, 3, 6, 2, 3};
TeamScore[19 + NumTeams] = {2, 3, 11, 3, 7, 3, 2, 8, 3, 4, 3, 4, 3, 4, 1, 6, 5, 5, 7, 3, 4, 6, 6, 3, 0, 2,
   2, 0, 4, 7, 7, 2, 2, 8, 6, 3, 6, 6, 1, 4, 2, 3, 1, 5, 8, 2, 11, 7, 6, 1, 2, 5, 5, 2, 1, 1, 4, 2, 4, 5,
   1, 3, 7, 1, 1, 0, 6, 0, 4, 4, 2, 7, 5, 1, 0, 5, 3, 1, 4, 11, 1, 1, 5, 3, 10, 4, 2, 4, 10, 3, 2, 0, 11,
   2, 9, 2, 7, 3, 2, 7, 3, 11, 7, 5, 8, 0, 2, 3, 3, 6, 6, 3, 15, 2, 1, 4, 4, 2, 11, 4, 2, 1, 4, 5, 4, 1, 2,
   4, 8, 4, 6, 5, 2, 3, 5, 7, 7, 6, 2, 7, 7, 3, 9, 2, 2, 2, 0, 1, 3, 4, 3, 3, 0, 5, 1, 7, 1, 0, 4, 3, 1};

(* NL CENTRAL *)
(* Team 20 = Cardinals*)
(* Team 21 = Astros*)
(* Team 22 = Cubs*)
(* Team 23 = Reds*)
(* Team 24 = Pirates*)
(* Team 25 = Brewers*)

TeamScore[20] = {3, 3, 2, 3, 3, 1, 4, 2, 6, 8, 8, 15, 9, 11, 9, 1, 6, 5, 5, 4, 3, 3, 5, 6, 11, 5, 3, 5, 5, 7, 7,
   6, 6, 0, 3, 6, 4, 9, 5, 3, 7, 3, 2, 5, 4, 0, 3, 9, 3, 3, 1, 10, 4, 4, 3, 4, 5, 7, 6, 5, 3, 7, 1, 9,
   0, 3, 3, 6, 0, 4, 4, 5, 5, 2, 0, 12, 4, 3, 0, 6, 5, 9, 5, 1, 3, 1, 8, 8, 1, 6, 7, 4, 5, 4, 1, 2, 5,
   6, 6, 9, 3, 10, 3, 2, 3, 9, 13, 3, 2, 8, 5, 7, 3, 2, 8, 3, 1, 5, 6, 1, 6, 2, 4, 7, 4, 0, 6, 1, 2, 4,
   8, 5, 0, 7, 2, 8, 8, 8, 6, 2, 1, 4, 2, 4, 4, 6, 5, 6, 3, 4, 2, 5, 4, 11, 6, 6, 1, 2, 3, 4, 13, 8};
```

```
TeamScore[20 + NumTeams] = {5, 11, 0, 4, 2, 3, 5, 3, 1, 2, 13, 5, 5, 2, 2, 2, 8, 3, 0, 2, 5, 0, 6, 5, 7, 3,
    2, 6, 6, 5, 8, 3, 0, 4, 1, 4, 11, 1, 6, 7, 9, 1, 1, 1, 2, 3, 0, 8, 1, 2, 3, 3, 15, 3, 7, 3, 7, 12, 1, 4,
    2, 4, 4, 2, 8, 5, 4, 8, 10, 7, 5, 4, 4, 10, 4, 2, 5, 6, 5, 2, 1, 6, 3, 5, 8, 0, 1, 9, 4, 7, 6, 2, 6, 1,
    3, 4, 6, 2, 4, 1, 4, 5, 1, 4, 5, 2, 5, 6, 6, 7, 10, 4, 2, 1, 4, 5, 5, 2, 1, 6, 2, 6, 5, 2, 5, 3, 2, 2,
    13, 9, 4, 4, 7, 4, 1, 3, 4, 11, 4, 3, 4, 2, 0, 3, 3, 3, 6, 4, 2, 2, 9, 0, 3, 6, 5, 8, 5, 1, 2, 5, 6, 0};
TeamScore[21] = {4, 4, 3, 2, 4, 3, 3, 5, 7, 4, 11, 5, 1, 2, 5, 6, 6, 4, 1, 7, 9, 1, 6, 5, 7, 0, 2, 5,
    10, 2, 4, 3, 1, 4, 1, 3, 4, 4, 7, 4, 2, 1, 1, 2, 5, 5, 3, 4, 4, 2, 6, 3, 2, 12, 7, 3, 7, 1, 3, 2, 4,
    4, 2, 4, 3, 1, 8, 0, 3, 4, 7, 7, 0, 3, 4, 5, 1, 2, 10, 3, 2, 7, 5, 4, 1, 3, 1, 8, 0, 3, 1, 4, 0, 6, 5,
    2, 7, 3, 2, 1, 4, 5, 1, 4, 5, 0, 2, 4, 4, 1, 5, 1, 5, 3, 9, 9, 3, 5, 0, 1, 0, 3, 6, 4, 6, 7, 4, 5, 6,
    6, 3, 1, 1, 4, 7, 8, 2, 2, 2, 0, 1, 4, 4, 3, 9, 2, 5, 5, 0, 3, 1, 3, 3, 4, 0, 9, 11, 2, 3, 5, 6, 0};
TeamScore[21 + NumTeams] = {5, 9, 7, 8, 12, 2, 4, 7, 1, 5, 2, 9, 0, 4, 3, 8, 1, 3, 9, 14, 6, 4, 5, 6, 11,
    5, 1, 0, 4, 3, 10, 2, 6, 5, 6, 7, 3, 6, 3, 7, 3, 3, 5, 4, 2, 7, 2, 3, 5, 1, 7, 11, 4, 7, 3, 1, 4, 3, 6, 7,
    7, 1, 9, 11, 6, 4, 3, 1, 7, 5, 3, 0, 1, 8, 5, 3, 5, 7, 14, 7, 3, 0, 7, 10, 2, 5, 5, 2, 5, 6, 6, 5, 4, 4,
    7, 5, 6, 2, 4, 5, 5, 10, 3, 2, 3, 4, 6, 5, 3, 5, 4, 8, 7, 7, 1, 11, 6, 8, 1, 6, 7, 4, 5, 3, 0, 5, 6, 9,
    8, 7, 1, 2, 2, 3, 4, 2, 0, 8, 8, 4, 3, 1, 5, 4, 3, 8, 1, 2, 1, 4, 2, 2, 2, 6, 2, 6, 2, 4, 19, 4, 13, 8};
TeamScore[22] = {3, 5, 4, 4, 6, 4, 7, 0, 5, 5, 2, 9, 0, 8, 5, 1, 2, 4, 2, 10, 3, 3, 3, 2, 4, 5, 3, 2, 4,
    5, 4, 3, 0, 4, 11, 1, 11, 0, 4, 5, 7, 5, 5, 9, 1, 11, 4, 9, 2, 0, 3, 7, 3, 1, 1, 4, 2, 2, 2, 4, 4, 5,
    1, 3, 1, 5, 5, 12, 3, 3, 4, 6, 2, 3, 6, 2, 3, 7, 7, 3, 2, 5, 4, 0, 3, 4, 2, 4, 10, 4, 6, 1, 3, 2, 3, 5,
    6, 2, 1, 4, 5, 5, 2, 0, 2, 2, 5, 6, 5, 11, 1, 7, 4, 11, 7, 1, 4, 4, 4, 8, 6, 4, 5, 3, 5, 3, 2, 0, 4,
    3, 3, 2, 4, 2, 7, 5, 0, 1, 5, 6, 4, 2, 6, 4, 5, 10, 12, 1, 2, 6, 4, 2, 2, 5, 1, 7, 5, 1, 2, 0, 6, 2};
TeamScore[22 + NumTeams] = {6, 3, 5, 1, 5, 6, 4, 6, 6, 4, 11, 5, 5, 3, 9, 0, 1, 5, 12, 8, 7, 5, 4, 11, 2,
    3, 4, 5, 1, 1, 5, 2, 2, 6, 4, 9, 4, 3, 7, 7, 5, 1, 15, 3, 5, 1, 7, 3, 4, 10, 2, 12, 7, 3, 6, 5, 3, 8, 8,
    1, 3, 7, 7, 4, 0, 4, 9, 7, 1, 4, 10, 3, 3, 4, 4, 3, 6, 3, 13, 6, 1, 2, 6, 1, 1, 5, 3, 5, 9, 7, 3, 9, 6, 1,
    13, 7, 1, 4, 9, 2, 1, 4, 3, 2, 4, 9, 13, 3, 3, 6, 0, 6, 3, 4, 8, 3, 2, 3, 10, 4, 5, 3, 6, 4, 4, 0, 6, 3,
    5, 2, 8, 5, 6, 3, 0, 2, 4, 3, 7, 3, 3, 4, 3, 5, 4, 6, 8, 2, 7, 8, 3, 1, 3, 2, 5, 1, 1, 2, 3, 2, 2, 9};
TeamScore[23] = {7, 4, 12, 8, 12, 2, 2, 6, 8, 3, 8, 2, 1, 11, 6, 3, 4, 1, 7, 2, 5, 0, 9, 2, 7, 6, 4, 5, 4,
    3, 10, 5, 2, 2, 6, 7, 3, 6, 7, 9, 7, 7, 0, 3, 4, 1, 4, 3, 6, 4, 4, 5, 6, 1, 7, 2, 4, 2, 8, 6, 8, 8, 1,
    3, 2, 10, 2, 6, 3, 7, 2, 0, 2, 3, 2, 10, 4, 10, 5, 5, 3, 4, 2, 1, 7, 0, 1, 9, 4, 7, 8, 3, 6, 1, 3, 0,
    0, 3, 4, 11, 4, 2, 6, 2, 9, 4, 7, 9, 3, 5, 4, 3, 4, 8, 7, 2, 3, 2, 5, 13, 3, 4, 2, 1, 11, 3, 5, 8, 5,
    3, 4, 6, 5, 2, 0, 0, 4, 11, 4, 3, 3, 4, 3, 4, 7, 1, 8, 2, 7, 8, 3, 1, 1, 2, 6, 2, 3, 3, 5, 6, 5, 0};
TeamScore[23 + NumTeams] = {6, 2, 3, 2, 4, 3, 13, 1, 10, 2, 2, 3, 6, 2, 7, 9, 5, 3, 4, 4, 3, 3, 5, 3, 6, 7,
    3, 9, 10, 2, 4, 4, 3, 0, 1, 3, 4, 5, 3, 7, 4, 5, 5, 5, 5, 2, 12, 10, 3, 5, 10, 1, 7, 2, 3, 7, 3, 1, 11, 9,
    2, 2, 4, 0, 3, 2, 4, 4, 2, 2, 3, 4, 1, 5, 4, 2, 5, 5, 7, 0, 4, 3, 8, 3, 5, 1, 8, 8, 5, 8, 4, 4, 5, 4, 1,
    2, 1, 1, 6, 2, 3, 4, 8, 8, 10, 3, 2, 0, 4, 1, 5, 4, 11, 7, 10, 3, 2, 1, 3, 1, 7, 6, 1, 3, 8, 5, 4, 6, 6,
    2, 3, 3, 4, 3, 9, 3, 6, 8, 6, 2, 4, 2, 6, 1, 12, 4, 12, 1, 2, 6, 6, 10, 8, 3, 4, 0, 4, 4, 4, 5, 4, 3};
TeamScore[24] = {6, 3, 5, 4, 2, 3, 1, 4, 4, 5, 0, 1, 6, 2, 7, 9, 0, 0, 5, 7, 3, 4, 2, 2, 2, 3, 1, 8, 4,
    5, 7, 2, 6, 5, 4, 3, 0, 2, 2, 6, 2, 5, 5, 10, 6, 0, 0, 2, 4, 10, 2, 3, 5, 9, 8, 2, 6, 3, 8, 3, 0, 1,
    3, 0, 3, 1, 7, 5, 1, 1, 2, 3, 9, 5, 3, 6, 2, 7, 1, 6, 1, 5, 3, 10, 5, 5, 2, 7, 3, 9, 4, 4, 7, 2, 1,
    1, 4, 1, 4, 3, 3, 1, 5, 3, 4, 5, 3, 6, 0, 6, 5, 2, 3, 5, 0, 9, 2, 0, 1, 6, 5, 2, 8, 5, 4, 1, 9, 4, 2,
    4, 4, 7, 4, 4, 2, 0, 4, 3, 7, 3, 3, 1, 5, 4, 0, 1, 6, 4, 2, 6, 2, 1, 1, 0, 5, 5, 4, 4, 4, 9, 4, 3};
TeamScore[24 + NumTeams] = {3, 5, 4, 3, 3, 1, 7, 3, 6, 6, 6, 4, 1, 11, 6, 3, 6, 6, 9, 2, 6, 2, 3, 0, 5, 0,
    4, 4, 3, 6, 4, 3, 1, 4, 1, 10, 2, 5, 8, 9, 4, 0, 3, 1, 2, 2, 2, 4, 2, 0, 3, 7, 1, 3, 9, 1, 3, 7, 5, 2, 2,
    8, 2, 7, 1, 0, 3, 4, 5, 5, 5, 8, 3, 4, 1, 4, 4, 6, 2, 2, 2, 3, 4, 2, 3, 1, 8, 4, 6, 1, 0, 6, 5, 0, 0, 3,
    6, 9, 3, 1, 4, 2, 2, 10, 7, 6, 5, 11, 1, 7, 15, 13, 7, 0, 6, 2, 7, 1, 2, 2, 4, 7, 11, 3, 5, 8, 2, 11, 0,
    8, 5, 0, 7, 7, 8, 2, 6, 1, 5, 6, 1, 4, 4, 13, 3, 4, 5, 6, 3, 2, 7, 6, 15, 1, 3, 8, 3, 3, 5, 8, 6, 7};
TeamScore[25] = {6, 2, 3, 1, 1, 5, 4, 4, 6, 6, 6, 4, 3, 4, 1, 6, 9, 3, 14, 6, 4, 5, 3, 6, 5, 1, 0, 2, 3,
    0, 1, 0, 4, 1, 4, 8, 6, 5, 8, 9, 2, 0, 5, 0, 7, 3, 3, 11, 7, 6, 4, 3, 6, 3, 7, 3, 6, 3, 6, 7, 1, 7, 1,
    8, 5, 4, 0, 4, 9, 7, 4, 4, 3, 4, 5, 3, 4, 11, 6, 2, 2, 0, 2, 8, 7, 6, 3, 3, 5, 8, 4, 4, 3, 0, 8, 4, 0,
    11, 5, 0, 4, 2, 1, 3, 2, 4, 4, 6, 5, 6, 7, 10, 8, 7, 7, 5, 5, 2, 7, 1, 2, 3, 2, 3, 1, 6, 11, 6, 8, 2,
    11, 0, 5, 6, 3, 1, 3, 4, 8, 8, 4, 4, 2, 0, 2, 3, 2, 3, 2, 2, 6, 10, 8, 2, 5, 1, 4, 6, 9, 8, 6, 7};
TeamScore[25 + NumTeams] = {7, 4, 12, 2, 0, 4, 2, 7, 0, 5, 0, 1, 4, 8, 5, 3, 0, 4, 7, 9, 1, 9, 2, 7, 0, 2,
    5, 6, 8, 8, 2, 6, 0, 3, 3, 6, 13, 2, 2, 6, 1, 3, 2, 1, 6, 2, 1, 3, 6, 4, 5, 2, 0, 7, 2, 4, 5, 2, 5, 2,
    2, 6, 4, 0, 3, 3, 1, 5, 5, 12, 10, 2, 12, 8, 1, 6, 3, 1, 2, 12, 5, 5, 6, 7, 9, 8, 7, 1, 4, 7, 8, 3, 12,
    4, 7, 3, 3, 3, 2, 4, 2, 4, 2, 2, 0, 2, 0, 2, 4, 2, 8, 5, 1, 5, 3, 3, 1, 5, 2, 0, 1, 0, 1, 1, 5, 1, 9, 2,
    1, 9, 4, 2, 2, 4, 2, 2, 8, 8, 2, 2, 0, 1, 4, 2, 7, 5, 3, 2, 1, 6, 3, 1, 1, 5, 1, 7, 1, 4, 5, 9, 4, 3};

(* NL WEST *)
(* Team 26 = Dodgers*)
(* Team 27 = Giants*)
(* Team 28 = Padres*)
(* Team 29 = Rockies*)
(* Team 30 = Diamondbacks*)

TeamScore[26] = {2, 4, 0, 7, 0, 5, 4, 4, 2, 6, 4, 3, 5, 2, 2, 2, 4, 1, 6, 5, 12, 8, 7, 4, 2, 5, 3, 2, 0, 5, 1,
    1, 3, 2, 4, 1, 10, 2, 4, 0, 1, 1, 3, 5, 1, 6, 2, 3, 3, 5, 1, 3, 1, 8, 7, 8, 0, 1, 11, 9, 1, 6, 0, 7,
    5, 11, 10, 4, 2, 2, 3, 0, 1, 4, 6, 5, 3, 1, 3, 15, 4, 0, 5, 1, 1, 2, 0, 3, 6, 1, 1, 4, 6, 2, 1, 0,
    3, 1, 2, 7, 3, 8, 3, 1, 9, 4, 3, 6, 1, 0, 7, 5, 3, 3, 1, 8, 1, 6, 7, 0, 1, 1, 5, 8, 6, 3, 2, 13, 9,
    6, 7, 6, 4, 8, 4, 6, 8, 2, 3, 2, 7, 7, 2, 3, 1, 2, 4, 3, 2, 7, 6, 15, 2, 5, 8, 2, 0, 6, 4, 6, 7};
TeamScore[26 + NumTeams] = {1, 3, 10, 5, 3, 7, 2, 0, 7, 1, 5, 4, 9, 11, 9, 1, 2, 10, 1, 3, 2, 10, 3, 5,
    4, 4, 2, 5, 7, 2, 4, 5, 6, 4, 2, 4, 3, 0, 3, 1, 4, 2, 0, 8, 3, 4, 9, 8, 4, 4, 2, 2, 6, 0, 1, 2, 3, 2, 8,
    6, 3, 2, 2, 9, 6, 7, 8, 6, 3, 7, 7, 7, 0, 0, 1, 7, 8, 6, 2, 0, 6, 1, 0, 7, 3, 5, 6, 5, 0, 0, 0, 1, 4,
    3, 4, 5, 5, 0, 7, 6, 1, 5, 2, 3, 5, 6, 6, 2, 0, 3, 4, 3, 4, 5, 2, 9, 0, 1, 0, 3, 2, 3, 1, 2, 7, 5, 1,
    2, 4, 1, 6, 7, 1, 5, 2, 4, 6, 1, 4, 7, 3, 4, 1, 0, 8, 7, 5, 2, 6, 2, 1, 1, 1, 8, 2, 0, 3, 2, 2, 7, 5};
TeamScore[27] = {1, 3, 10, 5, 1, 8, 5, 3, 1, 1, 5, 4, 5, 5, 5, 8, 6, 2, 1, 2, 6, 3, 0, 5, 0, 2, 2, 0, 7,
    2, 2, 4, 3, 3, 1, 4, 3, 4, 3, 4, 3, 8, 3, 2, 3, 5, 1, 6, 0, 5, 2, 0, 7, 3, 7, 12, 3, 1, 2, 5, 1, 3,
    0, 3, 2, 4, 6, 5, 2, 2, 2, 1, 2, 5, 2, 4, 1, 3, 13, 6, 1, 2, 4, 15, 3, 3, 3, 6, 2, 2, 3, 4, 6, 6, 3,
    4, 5, 5, 0, 2, 4, 2, 2, 2, 4, 3, 2, 0, 2, 1, 8, 0, 2, 1, 3, 0, 6, 2, 1, 3, 5, 4, 1, 7, 0, 0, 5, 6, 5,
    2, 1, 2, 2, 3, 0, 2, 4, 6, 2, 1, 7, 6, 1, 1, 0, 8, 8, 3, 3, 8, 9, 6, 12, 1, 8, 2, 1, 2, 2, 3, 7, 3};
TeamScore[27 + NumTeams] = {2, 4, 0, 7, 3, 4, 4, 2, 6, 6, 4, 3, 2, 3, 6, 1, 3, 10, 4, 5, 9, 2, 2, 2, 3, 1,
    5, 2, 6, 0, 5, 3, 2, 0, 0, 3, 2, 11, 0, 7, 5, 5, 1, 1, 0, 4, 5, 7, 1, 4, 3, 6, 3, 4, 5, 7, 1, 2, 1, 4,
    2, 1, 3, 2, 10, 2, 5, 2, 3, 5, 4, 2, 9, 1, 1, 3, 0, 1, 7, 3, 2, 5, 3, 3, 6, 5, 5, 5, 1, 5, 1, 2, 2, 1,
    11, 3, 0, 3, 1, 4, 2, 1, 7, 1, 1, 4, 7, 9, 5, 6, 1, 3, 9, 2, 1, 5, 0, 9, 2, 0, 2, 5, 2, 5, 1, 6, 7, 4,
    7, 1, 3, 1, 1, 4, 7, 5, 0, 2, 7, 4, 2, 4, 3, 2, 3, 1, 3, 2, 1, 5, 1, 5, 5, 2, 5, 8, 3, 15, 5, 1, 0, 6};
TeamScore[28] = {5, 11, 0, 3, 4, 2, 0, 7, 2, 2, 3, 0, 4, 3, 8, 0, 1, 5, 0, 0, 2, 1, 5, 2, 0, 2, 5, 7, 3,
    6, 4, 4, 0, 4, 3, 6, 13, 7, 9, 8, 8, 1, 2, 1, 1, 0, 1, 1, 2, 3, 1, 2, 5, 3, 5, 3, 4, 3, 6, 7, 0, 2, 3,
    7, 1, 1, 0, 3, 3, 3, 5, 0, 4, 5, 5, 5, 11, 1, 4, 4, 4, 4, 0, 1, 1, 5, 5, 5, 1, 0, 0, 1, 2, 1, 11, 3,
    4, 14, 5, 1, 6, 3, 5, 1, 3, 4, 2, 6, 8, 2, 0, 3, 15, 13, 7, 8, 4, 9, 3, 3, 1, 7, 4, 6, 3, 3, 4, 14, 4,
    7, 1, 0, 1, 1, 1, 5, 2, 0, 4, 7, 2, 4, 3, 1, 2, 5, 7, 3, 2, 1, 2, 3, 1, 8, 2, 4, 0, 3, 2, 2, 2, 9};
TeamScore[28 + NumTeams] = {3, 3, 2, 1, 8, 4, 4, 2, 3, 8, 2, 1, 2, 5, 6, 1, 2, 4, 3, 2, 4, 3, 3, 8, 7, 3,
    2, 0, 4, 5, 7, 3, 6, 3, 4, 8, 6, 12, 7, 2, 4, 6, 5, 0, 4, 4, 6, 3, 3, 1, 2, 1, 4, 2, 4, 4, 7, 1, 3, 2,
    3, 0, 5, 3, 2, 2, 2, 1, 6, 6, 6, 1, 5, 14, 4, 1, 2, 10, 1, 3, 2, 1, 6, 0, 3, 3, 3, 6, 2, 1, 1, 4, 6, 6,
    3, 4, 0, 3, 3, 3, 8, 5, 4, 6, 4, 3, 3, 10, 3, 6, 1, 0, 5, 2, 3, 9, 5, 5, 2, 5, 13, 3, 5, 1, 7, 1, 3, 1,
    3, 5, 2, 5, 3, 6, 4, 8, 4, 3, 5, 2, 7, 6, 1, 4, 3, 6, 6, 8, 3, 3, 0, 1, 5, 2, 1, 0, 2, 0, 6, 0, 6, 2};
TeamScore[29] = {6, 3, 3, 7, 7, 3, 6, 6, 7, 5, 6, 9, 5, 3, 9, 1, 3, 10, 1, 3, 3, 5, 4, 0, 4, 4, 3, 6, 2,
    3, 2, 0, 2, 3, 5, 12, 7, 2, 7, 5, 1, 7, 6, 2, 1, 12, 2, 1, 3, 3, 15, 3, 1, 2, 3, 1, 2, 1, 3, 0, 5, 9,
    6, 7, 8, 1, 6, 6, 13, 5, 1, 8, 4, 3, 4, 3, 4, 3, 3, 2, 4, 9, 9, 8, 1, 3, 1, 3, 3, 2, 0, 12, 4, 7, 3, 4,
    12, 3, 6, 8, 3, 0, 5, 2, 3, 3, 10, 3, 3, 0, 6, 6, 3, 15, 2, 10, 3, 2, 1, 1, 6, 2, 7, 5, 12, 2, 7, 5, 9,
    8, 7, 1, 6, 7, 1, 4, 2, 3, 5, 2, 7, 8, 3, 1, 12, 4, 1, 6, 5, 1, 5, 5, 2, 1, 0, 6, 2, 4, 19, 1, 0, 6};
TeamScore[29 + NumTeams] = {7, 1, 0, 5, 1, 4, 4, 5, 6, 4, 5, 4, 0, 8, 5, 8, 6, 2, 4, 1, 6, 3, 3, 3, 1, 8,
    4, 4, 3, 4, 3, 3, 1, 4, 9, 7, 9, 8, 4, 3, 2, 1, 7, 3, 3, 4, 5, 2, 6, 10, 4, 4, 7, 8, 0, 3, 1, 2, 0, 2, 3,
    7, 5, 11, 10, 3, 3, 3, 6, 4, 9, 7, 3, 4, 2, 8, 6, 7, 2, 3, 6, 0, 6, 16, 4, 5, 9, 6, 2, 1, 2, 3, 0, 8, 4,
    7, 3, 2, 9, 4, 12, 7, 8, 3, 1, 2, 6, 8, 4, 5, 8, 3, 5, 7, 3, 7, 2, 3, 2, 6, 1, 6, 4, 6, 5, 8, 6, 3, 5,
    6, 6, 6, 7, 6, 5, 9, 4, 0, 4, 7, 10, 3, 5, 4, 7, 1, 2, 2, 8, 9, 6, 12, 8, 2, 4, 9, 11, 2, 3, 3, 7, 3};
TeamScore[30] = {7, 1, 1, 5, 6, 13, 1, 10, 2, 13, 5, 2, 3, 6, 5, 3, 4, 1, 4, 4, 4, 7, 4, 11, 2, 3, 4, 4,
    4, 3, 3, 6, 3, 0, 3, 2, 3, 1, 4, 4, 6, 5, 2, 8, 9, 3, 4, 5, 2, 6, 7, 11, 4, 15, 2, 6, 1, 4, 2, 4, 5, 2,
    2, 4, 9, 5, 12, 5, 2, 3, 4, 2, 2, 7, 3, 5, 7, 0, 3, 4, 6, 2, 4, 4, 2, 8, 7, 1, 4, 7, 6, 2, 4, 3, 4, 3,
    3, 2, 4, 4, 12, 7, 6, 4, 3, 5, 6, 6, 5, 6, 1, 4, 3, 4, 1, 11, 6, 8, 4, 6, 5, 3, 2, 1, 2, 1, 0, 1, 2,
    4, 8, 5, 3, 6, 5, 9, 4, 2, 7, 4, 10, 3, 5, 4, 3, 6, 6, 7, 5, 2, 0, 1, 5, 1, 3, 8, 3, 15, 5, 2, 7, 5};
TeamScore[30 + NumTeams] = {6, 3, 4, 6, 4, 2, 6, 8, 8, 8, 15, 5, 5, 5, 4, 1, 7, 4, 6, 8, 0, 5, 8, 2, 4, 5,
    3, 3, 6, 2, 4, 0, 4, 1, 4, 3, 4, 0, 1, 8, 1, 4, 1, 7, 6, 2, 12, 2, 1, 3, 6, 3, 2, 4, 5, 5, 6, 0, 0, 9,
    8, 3, 0, 6, 5, 1, 9, 6, 5, 2, 1, 6, 8, 2, 2, 3, 6, 6, 8, 5, 4, 6, 5, 2, 7, 6, 3, 3, 1, 6, 7, 4, 6, 2,
    1, 0, 11, 5, 0, 8, 3, 0, 1, 3, 4, 9, 4, 3, 2, 1, 8, 7, 5, 3, 9, 9, 3, 5, 3, 4, 3, 2, 9, 4, 4, 8, 1, 4,
    0, 2, 1, 0, 1, 1, 1, 4, 2, 6, 2, 1, 7, 8, 3, 1, 2, 5, 7, 2, 4, 3, 2, 3, 1, 0, 5, 5, 1, 2, 2, 4, 6, 7};
```

```
(* this is a very important function. it has two inputs,
the bins used to analyze the team data and find the best fit parameters,
and the bins used for the chisquare test on the indep of rs and ra. *)
(* these are stored in Bins and ChiIndepBins. the program below computes the lengths and other relevent
 quantities which are used by programs later. To make sure there is at least SOME choice of bins,*)
BinCreate[binlist_, chibinlist_] := Module[{},
   Bins = binlist;
   ChiIndepBins = chibinlist;
   NumBins = Length[Bins] - 1;
   NumBinsChiTests = Length[ChiIndepBins] - 1;
   FirstBin = Bins[[1]];
   FinalBin = Last[Bins];
   MaxBinValue = Last[Bins];
   Print["Set of Bins is = ", Bins, " and number of bins = ", NumBins];
   Print["Set of Bins for Independence tests of rs and ra is ",
    ChiIndepBins, " and the number of bins = ", NumBinsChiTests];
   Print[" "];

   (* these lines assign to each integer what bin it is in *)
   (* we store that in xtoBins *)
   For[k = 0, k ≤ 40, k++,
    {
     For[m = 1, m ≤ NumBins, m++,
       {
        If[k ≥ Bins[[m]] && k < Bins[[m + 1]], xtoBins[k] = m];
       }];
    }];

  ];   (* end of module *)

(* We now 'bin' the TeamScore data. It is important that we know there are NumGames =
 162 in a season. In fact, we put an error check on that below. The program above bins the data and
  saves to ObsTeamScore. We go from i=1 to i=NumTeams. We initialize a temporary array to be zero,
and then for each game we see which bin the score falls into. this is easily done using
 the Ceiling function. we increment the temp counter by one. after finishiing this
loop we initialize ObsTeamScore to be zero and then save the temp array here.
As some of the older code uses NumObs, we set that to be NumGames. *)
(* in the code below I changed the formula inside to 1 + (teamscore - 1stbin) / binsize;
the reason is a team could score zero and this would then be stored in the zeroth entry of our list,
which would be bad. *)
BinTeamData[binlist_, chibinlist_] := Module[{},
   BinCreate[binlist, chibinlist];
   (*Loop over double num teams for rs and ra*)
   For[i = 1, i ≤ DoubleNumTeams, i++,
    {
     Clear[TempBin];
     NumObs = Length[TeamScore[i]];
     For[j = 0, j ≤ NumBins + 2, j++, TempBin[j] = 0];
     For[j = 1, j ≤ NumObs, j++,
      {
       For[k = 1, k ≤ NumBins, k++, If[TeamScore[i][[j]] ≥ Bins[[k]] && TeamScore[i][[j]] < Bins[[k + 1]],
          TempBin[k]++]]; (*If it lies in bin k*)
      }
     ];
     ObsTeamScore[i] = {};
     For[k = 1, k ≤ NumBins + 1, k++, ObsTeamScore[i] = Append[ObsTeamScore[i], TempBin[k]]];
     If[Sum[ObsTeamScore[i][[k]], {k, 1, NumBins}] ≠ NumObs,
      Print["Error - do not have enough games! Trouble with team ", i, "; only have ",
       Sum[ObsTeamScore[i][[k]], {k, 1, NumBins}], " games."]]; (*Error message if not enough games*)
    }];
  ];
```

```
(* This gives area in Bin k. note it is a function of the parameters a1,
a2,b1,b2,c,d and we need to know the two endpoints,
as we need the starting point of the bin and the ending point. *)
AWBin[k_, a1_, a2_, b1_, b2_, c_, d_] := AW[Bins[[k]], Bins[[k + 1]], a1, a2, b1, b2, c, d];
(*We must multiply area by NumGames. The integral over everything gives 1,
but we observe NumGames things. As the sum Pred equals the sum of the Obs,
we must mult each area by NumGames *)
Pred[team_, k_, a1_, a2_, b1_, b2_, c_, d_] := Length[TeamScore[team]] * AWBin[k, a1, a2, b1, b2, c, d] ;
ChiPred[team_, sp_, ep_, a1_, a2_, b1_, b2_, c_, d_] :=
  Length[TeamScore[team]] * AW[sp, ep, a1, a2, b1, b2, c, d] ;
(* This is the difference b/w Obs and Pred in bin k for team team_.  *)
ObsPred[team_, k_, a1_, a2_, b1_, b2_, c_, d_] :=
  (ObsTeamScore[team][[k]] - Pred[team, k, a1, a2, b1, b2, c, d])^2 ;
ChiObsPred[team_, sp_, ep_, a1_, a2_, b1_, b2_, c_, d_] :=
  (( Sum[If[TeamScore[team][[j]] ≥ sp && TeamScore[team][[j]] < ep, 1, 0],
        {j, 1, Length[TeamScore[team]]} ] - ChiPred[team, sp, ep, a1, a2, b1, b2, c, d])^
     2 ) / ChiPred[team, sp, ep, a1, a2, b1, b2, c, d];
(* This is the total error, again for just ONE team." *)
Error[team_, a1_, a2_, b1_, b2_, c_, d_] := Sum[ObsPred[team, k, a1, a2, b1, b2, c, d], {k, 1, NumBins}]
(* this is the predicted mean score from a weibull with params a1,a2,b,c,d *)
MeanScore[a1_, a2_, b1_, b2_, c_, d_] := d * (b1 + a1 * Gamma[1 + (1 / c)]) + (1 - d) * (b2 + a2 * Gamma[1 + (1 / c)])
(* this calculates the mean of a team's scores *)
MeanOfTeam[i_] := 1.0 * Sum[TeamScore[i][[j]], {j, 1, Length[TeamScore[i]]}] / Length[TeamScore[i]];

(* predict number of wins*)
PredWonLoss[a1rs_, a2rs_, a1ra_, a2ra_, b1rs_, b2rs_, b1ra_, b2ra_, c_, d_, f_] :=
 {(d * f) * (a1rs ^ c / (a1rs ^ c + a1ra ^ c)) + (d) * (1 - f) * (a1rs ^ c / (a1rs ^ c + a2ra ^ c)) +
   (f) * (1 - d) * (a2rs ^ c / (a2rs ^ c + a1ra ^ c)) + ((1 - d) * (1 - f)) * (a2rs ^ c / (a2rs ^ c + a2ra ^ c)),
  (MeanScore[a1rs, a2rs, b1rs, b2rs, c, d] - (d * b1rs + (1 - d) * b2rs)),
  (MeanScore[a1ra, a2ra, b1ra, b2ra, c, d] - (f * b1ra + (1 - f) * b2ra))}
(*Won Loss shift for different bins*)
PredWonLossShift[a1rs_, a2rs_, a1ra_, a2ra_, b1rs_, b2rs_, b1ra_, b2ra_, b2_, c_, d_, f_] :=
 {(d * f) * (a1rs ^ c / (a1rs ^ c + a1ra ^ c)) + (d) * (1 - f) * (a1rs ^ c / (a1rs ^ c + a2ra ^ c)) +
   (f) * (1 - d) * (a2rs ^ c / (a2rs ^ c + a1ra ^ c)) + ((1 - d) * (1 - f)) * (a2rs ^ c / (a2rs ^ c + a2ra ^ c)),
  (MeanScore[a1rs, a2rs, b1rs, b2rs, c, d] - (d * b1rs + (1 - d) * b2rs) + b2),
  (MeanScore[a1ra, a2ra, b1ra, b2ra, c, d] - (f * b1ra + (1 - f) * b2ra) + b2)}
(*Won Loss percentage*)
ObservedWonLossPercentage[i_] :=
  Sum[ If[TeamScore[i][[j]] > TeamScore[i + NumTeams][[j]], 1, 0], {j, 1, Length[TeamScore[i]]}] /
   (1.0 * Length[TeamScore[i]]);
```

```
(*we will save data about the teams from our curve fitting here.
1) a_RS1... 2) a_RS2... 3) a_RA1... 4)a_RA2... 5) b_RS1... 6)b_RS2
                                       7) b_RA1 8) b_RA2 9) c... 10) d
11) error from least squares
12) error from chisquare RS
13) error from chisquare RA
14) observed number of wins
15) observed number of losses
16) predicted number of wins
17) predicted number of losses
18) observed won-loss percentage
19) predicted won-loss percentage
20) number of games off by
21) error from independence tests *)

For[i = 1, i ≤ NumTeams, i++,
  For[k = 1, k ≤ 21, k++, TeamBestFits[i, k] = 0]];
For[i = 1, i ≤ NumTeams, i++,
  {
   (*Observed number of wins*)
   TeamBestFits[i, 15] =
    Sum[If[TeamScore[i][[j]] > TeamScore[i + NumTeams][[j]], 1, 0], {j, 1, Length[TeamScore[i]]}];
   (*Observed number of losses*)
   TeamBestFits[i, 16] =
    Sum[If[TeamScore[i][[j]] > TeamScore[i + NumTeams][[j]], 0, 1], {j, 1, Length[TeamScore[i]]}];
   (*Observed won-loss percentage*)
   TeamBestFits[i, 19] = ObservedWonLossPercentage[i];
  }];


(* THIS IS THE KEY PART OF THE CODE. IT FINDS THE BEST VALUES FOR THE PARAMETERS TO
  MINIMIZE THE SUM OF SQUARES OF ERRORS. WE ASSUME RS AND RA COME FROM TWO INDEPENDNET
  WEIBULLS WITH PARAMS (a1rs,a2rs,b1rs,b2rs,c,d) AND (a1ra,a2ra,b1ra,b2ra,c,d); *)
(* the input is which team we want to study *)
(* if printchoice = 1 it prints out results, else does not print *)
BestWeibullParameters[i_, printchoice_, list_] :=
 Module[ {a1rs, a2rs, a1ra, a2ra, b1rs, b2rs, b1ra, b2ra, c, d, f},
  (* i represents the team *)
  Clear[a1rs]; Clear[a2rs]; Clear[a1ra]; Clear[a2ra];
  Clear[b1rs]; Clear[b2rs]; Clear[b1ra]; Clear[b2ra]; Clear[c]; Clear[d]; Clear[f];
  b1rs = -.5;
  b2rs = -.5;
  b1ra = -.5;
  b2ra = -.5;
  temp3 =
   NMinimize[ {Error[i, a1rs, a2rs, b1rs, b2rs, c, d] + Error[i + NumTeams, a1ra, a2ra, b1ra, b2ra, c, f],
      a1rs > 0, a2rs > 0, a1ra > 0, a2ra > 0,  c > 0, d ≥ 0, f ≥ 0},
     {a1rs, a2rs, a1ra, a2ra, c, f}, MaxIterations → MaxNumIters];
  TeamBestFits[i, 1] = Last[temp3[[2, 1]]];
  TeamBestFits[i, 2] = Last[temp3[[2, 2]]];
  TeamBestFits[i, 3] = Last[temp3[[2, 3]]];
  TeamBestFits[i, 4] = Last[temp3[[2, 4]]];
  TeamBestFits[i, 5] = b1rs;
  TeamBestFits[i, 6] = b2rs;
  TeamBestFits[i, 7] = b1ra;
  TeamBestFits[i, 8] = b2ra;
  TeamBestFits[i, 9] = Last[temp3[[2, 5]]];
   TeamBestFits[i, 10] = Last[temp3[[2, 6]]] ;
  TeamBestFits[i, 11] = Last[temp3[[2, 7]]] ;

  (*First[temp3]*)
 ]

(*Chi Square Test of the Weibull parameters *)
ChiSquareTestWeibullParams[i_, a1_, a2_, b1_, b2_, c_, d_, chierrorlist_] :=
  Sum[ChiObsPred[i, chierrorlist[[k]], chierrorlist[[k + 1]], a1, a2, b1, b2, c, d],
   {k, 1, Length[chierrorlist] - 1}];
```

```
(*chi sq test for independence of rs and ra*)
(* we have a grid NumBins by NumBins,say an r x c table. the predicted value of entry
 E_{r,c} is (Sum of row r)*(Sum of colmn cc) / SampleSize.
   Then we look at sum_{r,cc} (Obs - Exp)^2 / Exp. *)
(* RS is the rows, RA is the columns *)
ChiSquareTestIndepRSRA[i_, chierrorlist_] := Module[{chinumgames, numloop, rowO, colO},
   (*Initialize everything as 0*)
   For[k1 = 1, k1 ≤ Length[chierrorlist] - 1, k1++,
    For[k2 = 1, k2 ≤ Length[chierrorlist] - 1, k2++,
     {
      Obs[k1, k2] = 0;
      Expect[k1, k2] = 0;
     }]];
   chinumgames = 0;
   For[j = 1, j ≤ Length[TeamScore[i]], j++,
    {
     k1 = 0;
     k2 = 0;
     For[z = 1, z ≤ Length[chierrorlist] - 1, z++,
      {
       If[TeamScore[i][[j]] ≥ chierrorlist[[z]], k1 = z]; (*Runs scored*)
       If[TeamScore[i + NumTeams][[j]] ≥ chierrorlist[[z]], k2 = z]; (*Runs allowed*)
      }
     ];
     Obs[k1, k2]++;
     chinumgames++;
    }
   ];

   (*Error messsage to make sure each row and column has observations*)
   For[k1 = 1, k1 ≤ Length[chierrorlist] - 1,
    k1++, If[Sum[ Obs[k1, cc], {cc, 1, Length[chierrorlist] - 1}] == 0,
      Print["Danger: row ", k1, " has 0 observations."]];];
   For[k2 = 1, k2 ≤ Length[chierrorlist] - 1, k2++, If[Sum[ Obs[r, k2], {r, 1, Length[chierrorlist] - 1}] ==
       0, Print["Danger: column ", k2, " has 0 observations."]];];

   (* NEW EXPECTED VALUE ASSIGNMENTS *)
   For[k1 = 1, k1 ≤ Length[chierrorlist] - 1, k1++,
    For[k2 = 1, k2 ≤ Length[chierrorlist] - 1, k2++,
      Expect[k1, k2] = 1;
     ];
   ];
   For[k1 = 1, k1 ≤ Length[chierrorlist] - 1, k1++,
     {
     Expect[k1, k1] = 0;
     rowO[k1] = Sum[ Obs[k1, k2], {k2, 1, Length[chierrorlist] - 1}];
     colO[k1] = Sum[ Obs[k2, k1], {k2, 1, Length[chierrorlist] - 1}];
    }]; (* DIAG ENTRIES 0 *)

   For[numloop = 2, numloop ≤ 100, numloop++,
    {
     For[k1 = 1, k1 ≤ Length[chierrorlist] - 1, k1++,
      For[k2 = 1, k2 ≤ Length[chierrorlist] - 1, k2++,
        {
         If[Mod[numloop, 2] == 0, TempExpect[k1, k2] = N[Expect[k1, k2] *
              rowO[k1] / Sum[ Expect[k1, cc], {cc, 1, Length[chierrorlist] - 1}]] , TempExpect[k1, k2] =
            N[Expect[k1, k2] * colO[k2] / Sum[ Expect[cc, k2], {cc, 1, Length[chierrorlist] - 1}]]; ];
        }];];
     For[k1 = 1, k1 ≤ Length[chierrorlist] - 1, k1++,
      For[k2 = 1, k2 ≤ Length[chierrorlist] - 1, k2++, Expect[k1, k2] = TempExpect[k1, k2]];];
    }]; (* END NUMLOOP *)

   For[k1 = 1, k1 ≤ Length[chierrorlist] - 1, k1++,
    For[k2 = 1, k2 ≤ Length[chierrorlist] - 1, k2++,
      {
       TempExpect[k1, k2] = 0;
      }];
   ];
   For[k1 = 1, k1 ≤ Length[chierrorlist] - 1, k1++,
    For[k2 = 1, k2 ≤ Length[chierrorlist] - 1, k2++,
        TempExpect[k1, k1] = 1;
       ];
     ];

     (*chi sq for independence of rs and ra*)
     TeamBestFits[i, 22] = N[Sum[ (Obs[k1, k2] - Expect[k1, k2])^2 / (Expect[k1, k2] + TempExpect[k1, k2]),
        {k2, 1, Length[chierrorlist] - 1}, {k1, 1, Length[chierrorlist] - 1}]];
    ];
(*end chi sq for independence of rs and ra*)

(*Finds Weibull parameters using max likelihood method*)
MaxLIikelihoodWeibullParams[i_, chierrorlist_] :=
  NMaximize[ {Sum[
      ObsTeamScore[i][[k]] Log[ AW[chierrorlist[[k]], chierrorlist[[k + 1]], ars1, ars2, -.5, -.5, c, d]] ,
       {k, 1, Length[chierrorlist] - 1}]  + Sum[  ObsTeamScore[i + NumTeams][[k]]
        Log[ AW[chierrorlist[[k]], chierrorlist[[k + 1]], ara1, ara2, -.5, -.5, c, f]],
       {k, 1, Length[chierrorlist] - 1}], ars1 > 2, ars2 > 2, ara1 > 2,  ara2 > 2, ars1 > ars2,
    ara1 > ara2,  c > 1,  c < 4, d ≥ 0, d ≤ 1, f ≥ 0,  f ≤ 1},  {ars1, ars2 , ara1, ara2, c, d, f}];
(* used to be c < 15 made c < 5 as issues compiling *)

Clear[LeastSqs];
For[i = 1, i ≤ 21, i ++,
  For[j = 1, j ≤ 8, j ++,
   LeastSqs[i, j] = 0]];
chilistRSRA = {-.5, .5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5, 11.5, Infinity};
BinTeamData[chilistRSRA, {-.5, .5, 1.5, 2.5, 3.5, 4.5, Infinity}];
For[ii = 1, ii ≤  NumTeams, ii ++,
  {
   temptemp = MaxLIikelihoodWeibullParams[ii, chilistRSRA];
   (*Print[temptemp];, maybe a problem here... switch rs and ra*)
   tars1 = Last[temptemp[[2, 1]]];
   tars2 = Last[ temptemp[[2, 2]]];
   tara1 = Last[temptemp[[2, 3]]];
   tara2 = Last[temptemp[[2, 4]]];
   tc = Last[temptemp[[2, 5]]];
   td = Last[temptemp[[2, 6]]];
   tf = Last[temptemp[[2, 7]]];
   LeastSqs[ii, 1] = tars1; LeastSqs[ii, 2] = tars2; LeastSqs[ii, 3] = tara1;
   LeastSqs[ii, 4] = tara2;
   LeastSqs[ii, 5] = tc;
   LeastSqs[ii, 6] = td;
   LeastSqs[ii, 7] = tf;
   LeastSqs[ii, 8] =  ChiSquareTestWeibullParams[ii, tars1, tars2, -.5, -.5, tc, td, chilistRSRA] +
     ChiSquareTestWeibullParams[ii + NumTeams, tara1, tara2, -.5, -.5, tc, tf, chilistRSRA] ;
   Print["Team = ", TeamNames[[ii]] ];
   Print["Pred Ave RS = ",
    td * (tars1 * Gamma[1 + (1.0 / tc )] - .5) + (1 - td) * (tars2 * Gamma[1 + (1.0 / tc )] - .5), "; Pred Ave RA = ",
    tf * (tara1 * Gamma[1 + (1.0 / tc )] - .5) + (1 - tf) * (tara2 * Gamma[1 + (1.0 / tc )] - .5),
    "; Pred c = ", tc , "; Pred d = ", td, ";Pred f = ", tf, "; Pred RS_1 = " , tars1,
    "; Pred RS_2 = " , tars2, "; Pred RA_1 = " , tara1, "; Pred RA_2 = " , tara2];
   Print["Total ChiSq for RSRA = ", LeastSqs[ii, 8], " with ",
    2 * (Length[chilistRSRA] - 1) - 1 - 3, " deg freedom."];
  }];

Off[NMaximize::cvmit]
```

```
(*finding least squared won loss for team*)
LeastSqsWonLossPercentageForATeam[i_] := Module[{a1rs, a2rs, a1ra, a2ra, b1rs, b2rs, b1ra, b2ra, c, d, f},
    (* we are studing team i *)
   Print["Analysis for team = ", i, ", the ", TeamNames[[i]]];
   BestWeibullParameters[i, 1]; (* find the best fit params, print out answers *)
   a1rs = LeastSqs[i, 1];
   a2rs = LeastSqs[i, 2];
   a1ra = LeastSqs[i, 3];
   a2ra = LeastSqs[i, 4];
   b1rs = -.5;
   b2rs = -.5;
   b1ra = -.5;
   b2ra = -.5;
   c = LeastSqs[i, 5];
   d = LeastSqs[i, 6];
   f = LeastSqs[i, 7];

   TeamBestFits[i, 20] = First[PredWonLoss[a1rs, a2rs, a1ra, a2ra, b1rs, b2rs, b1ra, b2ra, c, d, f]];
   TeamBestFits[i, 21] = (ObservedWonLossPercentage[i] -
       First[PredWonLoss[a1rs, a2rs, a1ra, a2ra, b1rs, b2rs, b1ra, b2ra, c, d, f]]) * Length[TeamScore[i]];
   TeamBestFits[i, 17] = TeamBestFits[i, 20] * Length[TeamScore[i]];
   TeamBestFits[i, 18] = (1 - TeamBestFits[i, 20]) * Length[TeamScore[i]];


   Print["RS: Predicted = ", MeanScore[a1rs, a2rs, b1rs, b2rs, c, d], " Actual = ", MeanOfTeam[i]];
   Print["RA: Predicted = ",
    MeanScore[a1ra, a2ra, b1ra, b2ra, c, f], " Actual = ", MeanOfTeam[i + NumTeams]];
   Print["Wins: Observed = ", TeamBestFits[i, 15], "; Predicted = ", TeamBestFits[i, 17]];
   Print["Losses: Observed = ", TeamBestFits[i, 16], "; Predicted = ", TeamBestFits[i, 18]];
   Print["Observed Won-Loss Percentage = ", TeamBestFits[i, 19] ];
   Print["Predicted Won-Loss Percentage = ", TeamBestFits[i, 20]];
   Print["Uncorrected Pred Won-Loss Percentage = ",
     First[PredWonLossShift[a1rs, a2rs, a1ra, a2ra, b1rs, b2rs, b1ra, b2ra, .5, c, d, f]] ,
     "; this is without subtracting 1/2."]
    Print["Number of Games off by is about ", TeamBestFits[i, 21]];

   Print["Error Analysis: ChiSquare for RS + RA = ",
    LeastSqs[i, 8], "; deg freedom = ", 2 * (Length[chilistRSRA] - 1) - 1 - 3];

   (*FIXING INDETERMINATE CASES*)
   (*pirates really messed up*)
   chilistIndep = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, Infinity};
   If[i == 17, chilistIndep = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, Infinity}];
   If[i == 24, chilistIndep = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, Infinity}];
   If[i == 28, chilistIndep = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, Infinity}];
   If[i == 30, chilistIndep = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, Infinity}];
   (* chilistIndep = {0,1,2,3,4,5,6,7,8,9,10,11,12,Infinity};  *)
   ChiSquareTestIndepRSRA[i, chilistIndep];
   Print["Error Analysis: ChiSquare for Independence of RS and RA = ", TeamBestFits[i, 22],
    "; deg freedom = ", (Length[chilistIndep] - 1 - 1) ^2 - (Length[chilistIndep] - 1)];
   (* MUST SUBTRACT Length[chierrorlist]-1 AS CANNOT HAVE RS = RA IN A GAME -- STRUCTURAL ZEROS *)
   Print[" "];
  ];
```

```
(*Printing out analysis of the teams*)
For[i = 1, i ≤ NumTeams, i++, LeastSqsWonLossPercentageForATeam[i]]
cData = {};
dData = {};
gamesoffDataUnsigned = {};
gamesoffDataSigned = {};
For[i = 1, i ≤ NumTeams, i++,
  {
   cData = Append[cData, LeastSqs[i, 5]];
   dData = Append[dData, LeastSqs[i, 6]];
   fData = Append[dData, LeastSqs[i, 7]];
   gamesoffDataUnsigned = Append[gamesoffDataUnsigned, Abs[TeamBestFits[i, 21]]];
   gamesoffDataSigned = Append[gamesoffDataSigned, TeamBestFits[i, 21]];
  }];
Print["Median c = ", Median[cData], "; Mean c = ",
  Mean[cData], "; StDev(Mean of c) = ", StandardDeviation[cData]];
Print["Median d = ", Median[dData], "; Mean d = ", Mean[dData],
  "; StDev(Mean of d) = ", StandardDeviation[dData]];
Print["Median f = ", Median[fData], "; Mean f = ", Mean[fData],
  "; StDev(Mean of f) = ", StandardDeviation[fData]];
Print["UNSIGNED: Median # games off = ", Median[gamesoffDataUnsigned]];
Print["UNSIGNED: Mean # games off = ", Mean[gamesoffDataUnsigned]];
Print["UNSIGNED: StDev(Mean # games off) = ", StandardDeviation[gamesoffDataUnsigned]];
Print["SIGNED: Median # games off = ", Median[gamesoffDataSigned]];
Print["SIGNED: Mean # games off = ", Mean[gamesoffDataSigned]];
Print["SIGNED: StDev(Mean # games off) = ", StandardDeviation[gamesoffDataSigned]];

SimOutput =
  {{StyleForm[Team, "Subsection"], StyleForm[Obs Wins, "Subsection"], StyleForm[Pred Wins, "Subsection"],
    StyleForm[ObsPerc, "Subsection"], StyleForm[PredPerc, "Subsection"],
    StyleForm[GamesDiff, "Subsection"], StyleForm["     γ   ", "Subsection"]}};
For[i = 1, i ≤ NumTeams, i++,
  SimOutput = Append[SimOutput, {TeamNames[[i]], TeamBestFits[i, 15],
      SetAccuracy[TeamBestFits[i, 17], 2], SetAccuracy[TeamBestFits[i, 19], 4],
      SetAccuracy[TeamBestFits[i, 20], 4], If[Abs[TeamBestFits[i, 21]] > .1,
       SetAccuracy[TeamBestFits[i, 21], 3], SetAccuracy[0.0, 0]],    SetAccuracy[cData[[i]], 3]}];];
PaddedForm[TableForm[SimOutput, TableAlignments → {Left, Center, Center, Center, Center, Right}], 4]

SimErrOutput = {{StyleForm[Team, "Subsection"],
    StyleForm["RS+RA χ2: 16 d.f.", "Subsection"], StyleForm["Indep χ2: 109 d.f", "Subsection"]}};
For[i = 1, i ≤ NumTeams, i++,
  SimErrOutput = Append[SimErrOutput, {TeamNames[[i]], SetAccuracy[LeastSqs[i, 8], 3],
      SetAccuracy[TeamBestFits[i, 22], 3]}];];
PaddedForm[TableForm[SimErrOutput, TableAlignments → {Left, Center, Center, Right}], 8]

SimOutput =
  {{StyleForm[Team, "Subsection"], StyleForm["Obs RS", "Subsection"], StyleForm["Pred RS", "Subsection"],
    StyleForm["z-stat", "Subsection"], StyleForm["Obs RA", "Subsection"],
    StyleForm["Pred RA", "Subsection"], StyleForm["z-stat", "Subsection"]}};
(*MeanScore[a1_,a2_,b1_,b2_,c_,d_] *)
For[i = 1, i ≤ NumTeams, i++,
  SimOutput = Append[SimOutput, {TeamNames[[i]], SetAccuracy[MeanOfTeam[i], 3],
     SetAccuracy[MeanScore[LeastSqs[i, 1], LeastSqs[i, 2], -.5, -.5, LeastSqs[i, 5], LeastSqs[i, 6]], 3],
     SetAccuracy[(MeanOfTeam[i] - MeanScore[LeastSqs[i, 1], LeastSqs[i, 2], -.5, -.5, LeastSqs[i, 5],
          LeastSqs[i, 6]]) / (StandardDeviation[TeamScore[i]] / Sqrt[Length[TeamScore[i]]]), 3],
     SetAccuracy[MeanOfTeam[i + NumTeams], 3], SetAccuracy[MeanScore[LeastSqs[i, 3],
       LeastSqs[i, 4], -.5, -.5, LeastSqs[i, 5], LeastSqs[i, 7]], 3],
     SetAccuracy[(MeanOfTeam[i + NumTeams] - MeanScore[LeastSqs[i, 3],
          LeastSqs[i, 4], -.5, -.5, LeastSqs[i, 5], LeastSqs[i, 7]]) /
        (StandardDeviation[TeamScore[i + NumTeams]] / Sqrt[Length[TeamScore[i]]]), 3]}]];
PaddedForm[TableForm[SimOutput, TableAlignments → {Left, Center, Center, Center, Center, Center}], 4]

(*Gets rid of Nminimize error messages*)
(*MIGHT NEED TO GET RID OF THIS LINE LATER*)
Off[NMinimize::cvmit]
```

```
(* WARNING: when we run this, we change the bins *)
BinCreate[{-.5, .5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5, 11.5, 12.5,
  13.5, 14.5, 15.5, 16.5, 17.5, 18.5, Infinity}, {-.5, .5, 1.5, 2.5, 3.5, 4.5, Infinity}]
BinTeamData[{-.5, .5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5, 11.5, 12.5, 13.5,
  14.5, 15.5, 16.5, 17.5, 18.5, Infinity}, {-.5, .5, 1.5, 2.5, 3.5, 4.5, Infinity}]
Bins
chilistRSRA
NumBins = Length[Bins] - 1
ObsTeamScore[1]
ObsTeamScore[1][[xtoBins[Floor[14.5]]]]

ObsStepFn[i_, x_, ra_] :=
 If[x < Bins[[1]], 0, If[x > Bins[[NumBins + 1]], 0, ObsTeamScore[i + ra * NumTeams][[xtoBins[Floor[x + .5]]]]]]

Bins
For[i = 1, i ≤ NumTeams, i++,
  {
   NumObs = Length[TeamScore[i]];
   Print["Team = ", i, ": Plots of RS (predicted vs observed) for ", TeamNames[[i]]];
   Print[Plot[{ObsStepFn[i, x, 0], NumObs *
       W[x, LeastSqs[i, 1], LeastSqs[i, 2], -.5, -.5, LeastSqs[i, 5], LeastSqs[i, 6]]}, {x, -.5, 20}]];
   Print["Team = ", i, ": Plots of RA (predicted vs observed) for ", TeamNames[[i]]];
   Print[Plot[{ObsStepFn[i, x, 1], NumObs * W[x, LeastSqs[i, 3], LeastSqs[i, 4],
        -.5, -.5, LeastSqs[i, 5], LeastSqs[i, 6]]}, {x, -.5, 20}]]; Print[" "];
  }];
```

REFERENCES

[MCGLP] - Steven J. Miller, Taylor Corcoran, Jennifer Gossels, Victor Luo, and Jaclyn Porfilio. “Pythagoras at the Bat”. Social Networks and the Economics of Sports (organized by Victor Zamaraev). To be published by Springer-Verlag. 2014.

[MIL] - Steven J. Miller. “A Derivation of the Pythagorean Won-Loss Formula in Baseball”. Chance Magazine. 2007.

[MUR] - G. Muraleedharan. “Characteristic and Moment Generating Functions of Three Parameter Weibull Distribution-an Independent Approach”. Research Journal of Mathematical and Statistical Sciences, Vol. 1(8), 25-27, September 2013.