WILFRID LAURIER UNIVERSITY

MA 680 - SEMINAR IN MATHEMATICAL MODELLING

DEPARTMENT OF MATHEMATICS

Examining Chess Ratings Utilizing Numerical Simulations

Author
Douglas BOWEN

Instructor
Dr. Connell MCCLUSKEY

Wednesday December 17, 2021

Abstract

The official description of this project is given in the instructions and is re-stated here as:

Rating systems for online chess (or Go) players. Consider a population of chess players of different skill levels. Each player is assigned a score to represent their skill level. Having a higher score than your opponent means you are more likely to win. How should scores be assigned so that a difference in scores of 100, for example, has a consistent meaning whether the two players have scores of 800 and 900, or of 2000 and 2100? Should the scores be modified after a game is played, and if so, how? This project could be done by studying the theory, or by coding a simulation of a hypothetical population.

This report aims to provide the reader with a sufficient background both of the competitive chess world and the statistical calculations that support the ratings system. The first chapter will cover general information on official FIDE chess ratings and tournaments, while the second discusses generalizations about player performances. The third and fourth chapters will cover the ratings systems derived to assign a numerical skill value to players and the analysis of scenarios utilizing this information, respectively. The fifth chapter will offer a summary of results, while the sixth contains the key function code utilized for these simulations (i.e. appendices). A bibliography can be found between chapters 5 and 6.

This LaTeX ***template*** *(up till here) is courtesy of Ryan Gauthier.*

Contents

Chapter 1

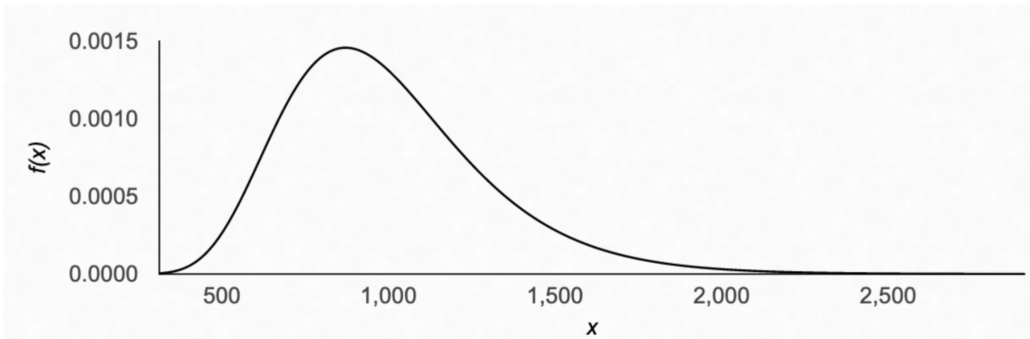
An Introduction to the World of Chess

1.1 Official FIDE Skill Ratings

The International Chess Federation (also known as Le Fédération Internationale des Échecs or "FIDE") was founded in Paris, France, on July 20th in 1924. It acts as the governing body for all official chess tournaments internationally.

As the main governing body, FIDE is responsible for calculating player rankings - a rough estimate of a players "skill". This is done using an Elo Rating System, which will be discussed in detail later on in Chapter 3.

Currently, player rankings range from 100 to 2856, though this top value fluctuates slightly month to month. The rankings of players tends to follow a roughly log-normal distribution which can be seen below.



Where X $\sim$ LogN(μ,σ) and $\mu =\sim 6.86$, $\sigma =\sim 0.3$
This also gives: E(X) = $\sim$ 1000 and Var(X) = $\sim$ 95,000

This was modelled via Chess.com data as opposed to official FIDE data as the official data is only updated month to month and is not easily cleaned; for the purposes of this project, a sufficiently populated data pool is all that is necessary as overall distributions should be quite similar across sites and systems.

It should be noted that in this case, rankings below 100 are all but a statistical impossibility. There is some inaccuracy in that ratings above 3000 are possible with this distribution, however it requires player counts well into the tens of thousands for even a few instances to occur and thus will have a negligible impact on simulations.

1.1.1 Class Brackets

Also of note are class brackets. Under FIDE regulations, players are divided into 16 "classes" that group those of similar ranking so that players only face others of similar skill levels. This will be of use later on in simulations when grouping players.

These classes are:

Lower	Upper
100	200
200	400
400	600
600	800
800	1000
1000	1200
1200	1400
1400	1600
1600	1800
1800	2000
2000	2200
2200	2300
2300	2400
2400	2500
2500	2700
2700	∞

1.2 Wins, Draws, and Losses

One important aspect of chess and its subsequent rating systems is that a players actual performance (score) will lie between the values of 0 and 1. For a loss, they will gain 0 points; a draw, 0.5 points; and lastly, a win will net them 1 point. Thus, in a single match their actual score (denoted S_i) will have the following property: $S_i \in \{0, 0.5, 1\}$

The relevance of this will be expanded on when examining the actual ratings system, so it is important to keep in mind.

1.3 Sensitivity

Similarly to the above mention of actual scores S_i, another important value determined by FIDE is K, the "Sensitivity" on a players ranking R_i. Essentially, this is the maximum amount a player can move up or down from any given single match. For later reference, the generalized K values are:

Player Rating R_i	K-Value
$100 < R_i \leq 2300$	40
$2300 < R_i \leq 2400$	20
$2400 < R_i$	10

While these are not perfectly accurate as there is some stipulation on games played [3], for our model they will suffice.

Chapter 2

Players and Their Habits

Before we analyze the ratings system(s) and move onto simulations, there are a few key assumptions that must be made about players and their performances.

2.1 Key Assumptions

The most substantial assumption made (and that allows for ratings systems to be possible) is that a players performance behaves like that of a random variable - that is to say, their performance can vary randomly game to game. This assumption is reasonable to expect as individuals can clearly have good days and bad days in many events (sports, games, etc.).

Expanding on this, we further assume that this random variation in performance follows a bell-curve shaped probability distribution. This means that an individuals performance has equal odds both when playing above their "expected" skill level (a good day) and when playing below (a bad day). We also assume that a players mean performance can change over time.

Historically, player performance was modelled utilizing a normal distribution; however, this was later changed to a logistic distribution following a rigorous statistical analysis of historical data [1]. The logistic distribution is one that more heavily favours the extremes or in other words, has "heavy tails". Thus, performances on the outer bounds (far from the mean) are slightly more likely to occur than they would be in a normal distribution.

2.2 The Logistic Distribution

2.2.1 Numerical

With our above assumptions in mind, we can now generate a cumulative distribution function (CDF) to help model the probability of a player winning.

For the logistic distribution, our CDF is as follows:

$$F(x;\mu,s) = \frac{1}{1+e^{\frac{-(x-\mu)}{s}}} \tag{2.1}$$

where x is our random variable, μ is the mean of our distribution, and s is the scale. This CDF tells us the probability that some random value of x is less than the mean μ.

That is,

$$P(x < \mu), \text{ for some scale } s$$

Now, this can be similarly applied to chess ratings. Simply put, our x becomes one players rating, while our μ becomes another players. Let us denote player ratings with R_i where $i \in \{A, B\}$ for players A and B respectively.

Let us also replace our x and μ values, where $R_A = \mu$ and $R_B = x$. Thus our CDF now tells us:

$$P(R_B < R_A), \text{ for some scale } s$$

We can also read this as "the probability that player A performs better than player B", or:

$$P(\text{Player A Wins})$$

and as this is a two player system, we can utilize a fundamental theorem of probability that the sum of all outcomes equals to one.

Thus,

$$P(\text{Player B Wins}) = 1 - P(\text{Player A Wins})$$

Alternatively, we could switch which players rating is x and which is μ, but this result is much simpler to work with.

2.2.2 Visualized

If one would prefer to visually examine the above, the following interactive graphical plots are provided:

1. Logistic Distributions
2. Logistic Function

In examining these distributions, one can clearly see the scales impact on chances of winning with regards to difference in ratings.

Chapter 3

Ratings Systems

3.1 [Simple] Elo's Rating System

The Elo rating system was developed by Arpad Elo, and was adopted by FIDE in 1970. It was made with simplicity in mind to allow for calculation by any individual with a pen and paper. Back then, computing power was not easily accessible and thus this simplicity was an important factor [2].

With the aforementioned information on player performance, we can now examine Elo's Rating System with sufficient confidence in our ability to understand the choices made by Elo.

Though the CDF of the logistic distribution can be found utilizing the function given in equation 2.1 and similarly tweaked with player ratings, Elo devised a slightly different function:

$$E_A = \frac{1}{1 + 10^{\frac{(R_B - R_A)}{400}}} \tag{3.1}$$

In essence, Elo has constructed a logistic function with scale 400 and base 10. Equivalently, Elo could have utilized e with a scale of 225 for similar results (and which would follow the logistic CDF defined in 2.1).

These numbers (10, 400) were selected for the chess community specifically, where it was deemed appropriate for a player with rating "D" higher than their opponent to be more likely to win. The below table summarizes some important difference amounts.

D	P(Win) Better Player	P(Win) Worse Player	Multiple
0	50%	50%	0.00x
100	$\sim$64%	$\sim$36%	1.75x
200	$\sim$75%	$\sim$25%	3x
400	$\sim$91%	$\sim$9%	10x
800	$\sim$99%	$\sim$1%	100x

In this formula, E_A represents the "Expected Score" of Player A. Recall from Section 1.2 that we denoted S_A as the "Actual Score" of Player A where values can take on a win (1 score), a draw (0.5 score), or a loss (0 score).

In the case of a single match, player i's actual score S_i will range from 0 to 1. Similarly, their expected score E_i will also range from 0 to 1.

In multiple matches, the averages of S_i and E_i will follow the same properties. Furthermore, the sums of S_i and E_i will fall between 0 and m where m is the total number of matches played.

For a player with rating R_i, Elo proposed that their new rating, R_i', would be updated utilizing the following formula (where A could be substituted for B):

$$R_A' = R_A + K(S_A - E_A) \tag{3.2}$$

In the above equation, we now see mention of our "Sensitivity Value" previously discussed. As our values for S_i and E_i for a single match result in a difference between 0 and 1, we can see that the minimum movement is $K * (0) = 0$ while the maximum movement is $K * (1) = K$ and thus the most an individual can move from a single match is in itself K.

3.1.1 Elo System Specific Assumptions

As a quick aside, it can clearly be seen that this rating system assumes that:

(a) a players rating is always reliable and (b) a players rating does not change unless games are played. In the Glicko Rating System discussed below, we see how these shortcomings have been adjusted for.

3.2 [Advanced] Glicko's Rating System

Though the focus of this project is on Elo's Rating System (mainly due to computing power required), what would be considered the more "advanced" rating system model will be quickly examined here.

The Glicko Rating System was created by Mark Glickman in 1995. It differs from the Elo Rating System in a few ways, but, in essence, the difference between the two systems boils down to the fact that while the Elo system assumes ratings are perfectly accurate and do not change between games, the Glicko system attempts to account for this false assumption[4].

While the formula will be discussed later on, let us first denote a few important terms in the Glicko system.

For starters, instead of a sensitivity for movement, the Glicko system utilizes a variable called "Ratings Deviation" or "RD". This value can be thought of as a way to construct a confidence interval where the lowest value in the interval is a player's rating minus 2x the RD (or the margin of error). The opposite is also true in that the maximum value is a player's rating plus 2x the RD. This way, when a player has a rating R_i, we can account for uncertainty in said rating.

As an example, a generalized CI for this might be:

$$R_i \pm Z(RD) \tag{3.3}$$

where Z is our critical value (for example a 95% interval would have a Z-score of 1.96). In plain English, we can also write this as "The player has a mean rating of R_i, give or take a potential error in this calculation of $Z(RD)$".

From this we clearly conclude that, when RD is low, a player has a rating with a smaller interval. When RD is high, a player has a larger rating interval.

With this in mind, how exactly is RD calculated? We can see the initialization formula here:

$$RD = min(\sqrt{RD^2 + c^2 t}, 350) \tag{3.4}$$

where "c" is a constant that impacts the increase in RD (ratings uncertainty) based on the time periods "t" that have elapsed between games. Similarly, 350 is a suggested starting RD value for new players [4].

Along with the ratings deviations of both players, their ratings are also necessary to update both individuals to new ratings values.

For example, continuing with our player's rating being defined as R (or R_i), we can find the update formula(s) as:

$$R' = R + \frac{q}{1/RD^2 + 1/d^2} \sum_{j=1}^{m} g(RD_j)(s_j - E[s|r, r_j, RD_j]) \tag{3.5}$$

and

$$RD' = \sqrt{(\frac{1}{RD^2} + \frac{1}{d^2})^{-1}} \tag{3.6}$$

where

$$q = \frac{ln(10)}{400} \tag{3.7}$$

$$g(RD) = \frac{1)}{\sqrt{1 + 3q^2(RD^2)/\pi^2}} \tag{3.8}$$

$$E[s|r, r_j, RD_j] = \frac{1}{1 + 10^{-g(RD_j)(r - r_j)/400}} \tag{3.9}$$

$$d^2 = [q^2 \sum_{j=1}^{m} (g(RD_j))^2 E[s|r, r_j, RD_j](1 - E[s|r, r_j, RD_j])]^{-1} \tag{3.10}$$

As one can see, this model is much more involved and certainly requires significant computing power to help calculate ratings.

An example will not be examined in this report, however, one can be found here on page 4 if interested.

3.2.1 Glicko-2 Variant

A newer variant of the Glicko system, aptly named Glicko-2 (again created by Mark Glickman), further expands upon his initial system.

This expansion makes one key change; it adds a volatility factor σ. This volatility factor further impacts expected fluctuations in a player's rating by accounting for hot, stable, and cold streaks to allow ratings to move more dynamically [5].

This change slightly complicates the previously derived Glicko system formulas and thus will not be further examined.

If interested, one can view the new derivations here .

3.3 Key Differences

As a quick recap, the following basic differences can be found between the two systems:

Factor	**Elo**	**Glicko**
Rating Reliability	Always Reliable	Unreliable
Update Requirements	Sensitivity "K" Player's Ratings Player's Scores	Time "t" Constant "c" Player's Ratings Player's Score's Player's RD's Player's Volatility*

While Glicko's system is much more involved, the updating system in Elo's method is still very good and provides an accurate picture of skill - hence why FIDE opts to utilize the Elo system rather than switch to Glicko.

Chapter 4

Scenarios Examined

Utilizing Elo's rating system, we can now begin examining potential simulations - the core of this report. We will examine two specific scenarios with a variety of changing variables.

A common term that will appear below in these scenarios is "Error Term" or "Convergence of Error Term". The error term is the absolute value of the difference in public and true rating of an individual as a percentage of their true rating. In the case of the individual scenario, this is that individuals error. In the case of a new population, this is the average error of all players involved. The convergence term is the error value at which it no longer fluctuates.

All simulations utilize the same seeds (random number generation order) and player distributions and thus are comparable across runs/varied scenarios.

4.1 Simulation Factors

Before examining the two scenarios, it is important to understand the factors that impact them.

There are 6 primary factors that impact both scenarios. These are:

- # of Simulations, n
  - The number of simulations helps us generate an accurate picture by removing any fluctuations caused by the inherent nature of random variables. One simulation might vary from the next by quite a lot, but over time will approach an average value.
- # of Matches, m
  - The number of matches is simply a count of how many "Player A vs. Player B" scenarios occur in the simulation.
- # of Players, p
  - The number of players is the total population size.
- Ratings
  - There are two ratings, a public rating and a true rating.
    - Public Ratings are assigned in a variety of scenario-specific ways (to be examined later).
    - True Ratings are assigned randomly following the lognormal distribution of players discussed in Chapter 1.
- Groupings
  - Groupings is a binary factor; either on ("Y") or off ("N"). When on, players are only able to match with other players who have a public rating similar to their own (following FIDE Class Brackets mentioned in Chapter 1).
- Anchors
  - Anchors is another binary factor; either on ("Y") or off ("N"). When on, a player whose rating is centered in each class (and that does not update after games) is added to the population.

4.2 Individual vs Population

In this scenario, an individual of unknown skill will enter a predefined population of players (already at their true skill levels) starting at a beginner level public rating of 600.

The results of this scenario with multiple adjustments made are summarized here:.

True Rating	Players	Error Term ∣ Matches No Grouping	Error Term ∣ Matches Grouping	Link
2800	10	23.3% ∣ >1000	54.6% ∣ 50	YT$_1$ ∣ YT$_2$
2500		14.8% ∣ >1000	50.3% ∣ 50	YT$_1$ ∣ YT$_2$
2200		6.30% ∣ >1000	41.8% ∣ 50	
1900		1.14% ∣ 800	35.7% ∣ 50	
1600		0.04% ∣ 400	18.4% ∣ 100	
1300		0.00% ∣ 200	9.19% ∣ 100	
1000		0.00% ∣ 100	4.79% ∣ 100	YT$_1$ ∣ YT$_2$
700		0.00% ∣ 100	1.29% ∣ 90	
400		0.01% ∣ 400	25.62% ∣ 100	
2800	1,000	19.0% ∣ >1000	17.3% ∣ 100	
2500		10.6% ∣ >1000	7.09% ∣ 125	
2200		3.18% ∣ 1000	0.09% ∣ 125	
1900		0.16% ∣ 600	0.00% ∣ 120	
1600		0.00% ∣ 300	0.00% ∣ 115	
1300		0.00% ∣ 175	0.00% ∣ 100	
1000		0.00% ∣ 100	0.00% ∣ 100	
700		0.00% ∣ 100	0.00% ∣ 90	
400		0.00% ∣ 300	0.00% ∣ 100	
2800	10,000	19.1% ∣ >1000	0.70% ∣ 400	
2500		10.7% ∣ >1000	0.04% ∣ 250	
2200		3.28% ∣ 1000	0.00% ∣ 170	
1900		0.17% ∣ 600	0.00% ∣ 100	
1600		0.00% ∣ 300	0.00% ∣ 90	
1300		0.00% ∣ 150	0.00% ∣ 70	
1000		0.00% ∣ 100	0.00% ∣ 60	
700		0.00% ∣ 100	0.00% ∣ 50	
400		0.00% ∣ 300	0.00% ∣ 80	

4.2.1 Scenario Insights

With this table, we can examine how the error term converges in different scenarios, as well as how many matches it took the player to actually reach said convergence point (roughly where the error stops fluctuating much). From these two data points, we can quickly see (a) whether or not the individual reaches their skill level, (b) at what point adding more matches is of no help, and (c) how grouping vs. not grouping impacts the aforementioned.

Based on the above table, a few valuable insights can be pulled. First and foremost, the further the individuals true skill is from the mean rating of 1,000:

- The more grouping hurts at low player counts.
- The more not grouping hurts at high player counts.

We can also see that not grouping has little impact on error rate between low and high player counts in that there is only a small difference in convergence point in either case.

An important insight that can be seen here is that there exists an equilibrium number of players where grouping and not grouping produce similar error convergence points. For example, when the player count is $\sim$1,000, the convergence point is quite similar across grouping and not grouping.

Similarly to examining the error rates, we can also examine the number of matches taken to converge to said rate. We can clearly see that at high player counts, grouping converges much more quickly with regards to the number of matches required. At low player counts, it "converges" more quickly as well; however, the trade-off is that the error term is much higher (as no more matches can be played due to bin mechanics).

4.3 New Population

In this scenario, instead of taking an individual and seeing how long it takes for them to approach their true skill rating, we examine the same but for an entire population of newly-generated players. In this instance, we look at the average error across all players in the population instead of just one individual. This allows us to draw insight on as to how many games are needed for a certain number of players to all receive accurate ratings. It should be noted our simulation count "n" is low and thus inherent variation is present.

These players have their true rating assigned in normal fashion following the lognormal distribution, however, their public rating is assigned in a variety of different ways such as:

- Completely Random
  - All individuals get a public rating that follows an $\sim$U(0,2900) distribution. That is to say, any rating between 0 and 2900 is equally likely to occur.
- Hybrid
  - All individuals have a 1-in-3 chance of receiving one of the following numbers: 600, 1000, 1200.
- Range
  - All individuals are able to select their own public rating and are accurate to within a maximum the specified range; that is to say, if the range is set to 30%, individuals are able to select their public rating to within $\pm$30% of their true skill.
- Fixed
  - All individuals are set to the same public rating as a starting point.

4.3.1 Methods

As discussed above, there a few different methods of assigning ones public rating. Here we examine a select few choices and see how their error terms start/converge. Anchors are not on.

Method	**Players**	**Error Term ∣ Matches** No Grouping	**Error Term ∣ Matches** Grouping	**Link**
Random		90% ∣ 200	90% ∣ 0	YT
Hybrid		11% ∣ 250	30% ∣ 120	YT
Range (30%)	10	7% ∣ 300	8.5% ∣ 80	YT
Fixed (1000)		8% ∣ 600	12% ∣ 300	YT
Random		90% ∣ 0	90% ∣ 0	
Hybrid		32% ∣ 0	32% ∣ 0	
Range (30%)	1,000	14% ∣ 0	10% ∣ 0	
Fixed (1000)		25% ∣ 0	25% ∣ 0	
Random		90% ∣ 0	90% ∣ 0	
Hybrid		32% ∣ 0	32% ∣ 0	
Range (30%)	10,000	15% ∣ 0	14% ∣ 0	
Fixed (1000)		26% ∣ 0	26% ∣ 0	

4.3.2 Scenario Insights

Similar to when examining an individual facing a pre-established population of players, we can utilize this table to draw some useful insights about methods in assigning public ratings, as well as see some unique trends.

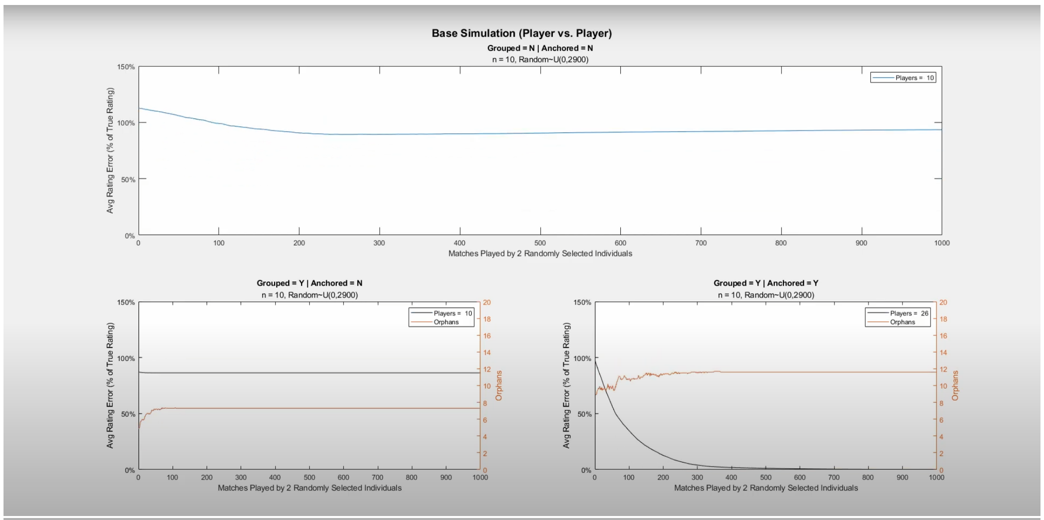
First, with respect to the error terms, we can clearly see that as the number of players increases, the error term "converges" to the same point in the same number of games, regardless of if players are grouped or not. This may look strange, but becomes clear when we remember that this error term is the **average** of the players in the population. Because of this, the term "converge" may be a bit misleading. As the number of players increases, the number of matches required to reduce said error term increases greatly (to a value much higher than my current system specs can handle). Thus this is actually only the "convergence point" in the case of 1,000 matches. If more matches were examined, this value would begin to decrease again.

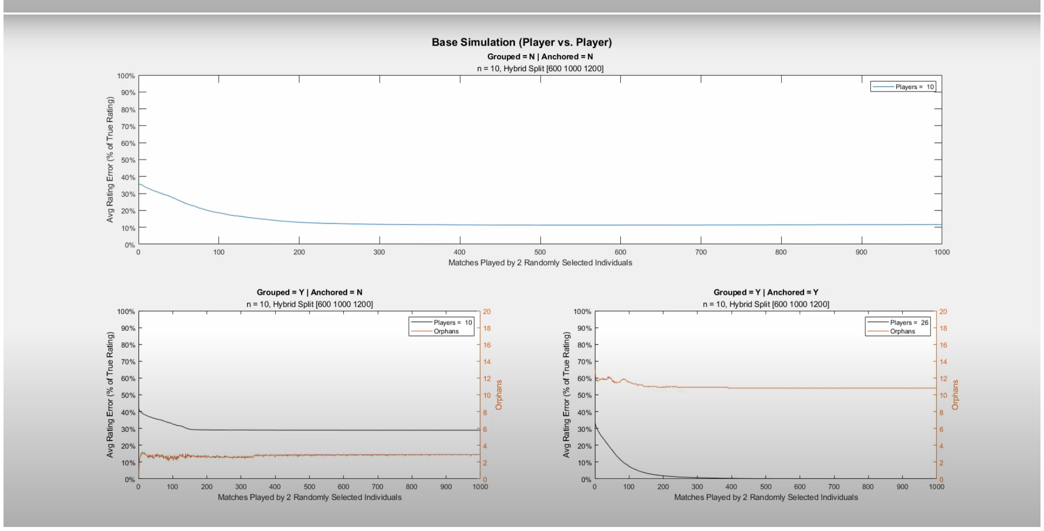
As an example, say we take 10 players and that their average error is 10%. If these 10 players play 1000 matches, it is likely that their average error term will converge properly to some value lower than 10%. However, if we take 10,000 players and have them play 1,000 matches, there will still be ~8,000 players who have not even played a match and whose error still remains at 10%. Thus, the average error will remain flat at 10%.

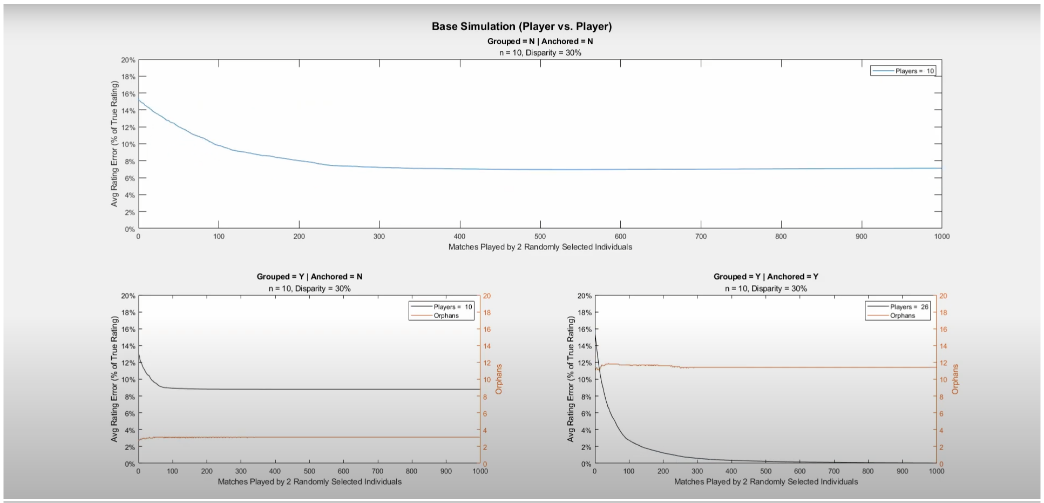
With this in mind, it becomes much more helpful to simply examine the curve of the error term over the 1,000 matches at a lower player count (in our case, 10). It also becomes helpful to examine the case of groupings with anchors, as it will allow us to see how the error term converges when there are always players to play against. Examining this allows us to see how error rates change with a sufficient ratio of players:matches.

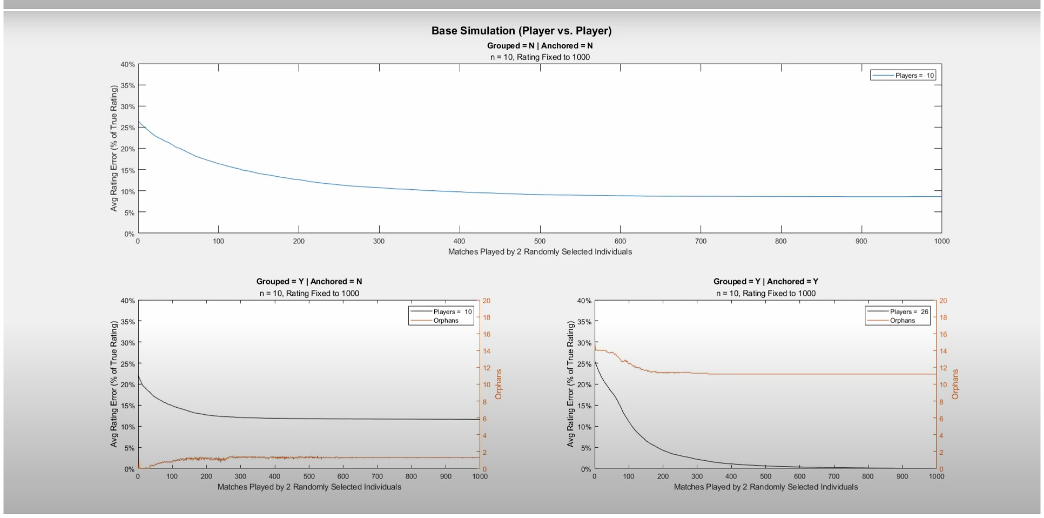
Keeping the above in mind, we can see the four above cases graphically below at 10 players and 1,000 matches (of note: Disparity = Range Scenario, and Orphans are simply the # of individuals in class brackets without an opponent to face).

You may have to zoom in to see values, but they should still retain legibility. The order of images (1 to 4) displays the following methods Random, Hybrid, Range (30%), Fixed (1000):






Base Simulation (Player vs. Player)
Grouped = N | Anchored = N
n = 10, Disparity = 30%
Avg Rating Error (% of True Rating)
Matches Played by 2 Randomly Selected Individuals
Players = 10
Grouped = Y | Anchored = N
n = 10, Disparity = 30%
Players = 10
Orphans
Grouped = Y | Anchored = Y
n = 10, Disparity = 30%
Players = 26
Orphans


Base Simulation (Player vs. Player)
Grouped = N | Anchored = N
n = 10, Rating Fixed to 1000
Avg Rating Error (% of True Rating)
Matches Played by 2 Randomly Selected Individuals
Players = 10
Grouped = Y | Anchored = N
n = 10, Rating Fixed to 1000
Players = 10
Orphans
Grouped = Y | Anchored = Y
n = 10, Rating Fixed to 1000
Players = 26
Orphans

Though the scales on the y-axis are different (so as to show changes clearly at high match counts), we can see that in the case of grouped and anchored scenarios, the closer the starting average error is to 0%, the faster the term converges to 0. In the case of no anchors existing, we can similarly see that the error reaches its convergence point more quickly the closer the starting average error is to 0% (this is evidenced by the steeper curvature of the function).

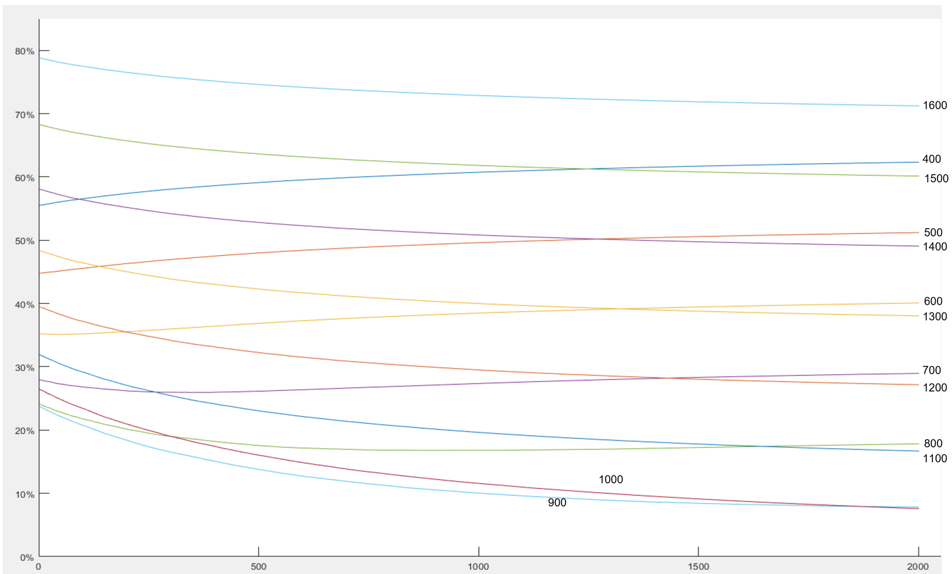
With this in mind, an overall generalization can be made that whichever system manages to start with the lowest error will converge the quickest with a sufficient number of players and matches. This is somewhat trivial and to be expected, but is important to confirm nonetheless. Similarly trivial, we can see that grouping allows for convergence to occur much more quickly, even when anchors do not exist (though similarly to the individual scenario, a sufficient player pool is necessary for matches to be played in each bin).

We can summarize the base average error term of the population via different scenarios as follows:

Method	**Error Term**
Random	~90%
Hybrid	~35%
Range (30%)	~15%
Fixed (1000)	~27%

The optimal choice here is clearly the scenario where a player can accurately select their public rating to within 30% of their true rating - however, this is unrealistic to expect. More realistically, it would make sense for a site to either (a) set all players to 1,000, the mean of the true rating distribution, or (b) allow players to choose between a 600, 1000, or 1200 rating. In terms of efficiency, fixing ratings to 1,000 would be the better choice; but allowing players some choice would be beneficial to them at only a slight loss in efficiency.

One unique thing to note that is not observed in these 4 cases is scenarios of other fixed ratings. Though not examined in detail, it is important to mention that assigning players with a fixed rating too far below the mean can result in the error term increasing (due to it being a percentage of true rating), while assigning a fixed rating very slightly under the mean at 800 or 900 can result in a lower starting error but higher error than the 1,000 case after more matches are played. This can quickly be seen here, though not examined in detail:



As a quick example to illustrate why this occurs, we take two players both publicly rated at 400 and true ratings of 800, 1200. We will call this Player A and Player B respectively. Player A has an "Actual Score" of 0.0909090909 while Player B has one of 0.909090909, though their expected scores are both 0.5. Let us assume K = 40 in line with FIDE classes.

In this case, the average error is $\frac{(\frac{400}{800}+\frac{800}{1200})}{2} = 58.33\%$. Player B will have their rating increase by $40 * (0.90909 - 0.5) =\sim 16$ while Player A will have their rating decrease by this amount. The new average error is $\frac{(\frac{416}{800}+\frac{784}{1200})}{2} = 58.67\%$, which has seen it slightly increase.

4.4 Shortcomings

For both scenarios, there were a few shortcomings & important things to note with regards to Elo's rating system and its application here. For starters, ratings below 100 were allowed as fringe cases occurred where one might drop slightly below the 100 bin (instead of capping them at 100, this was allowed as it was incredibly rare). Secondly, two of our key assumptions are not actually true in nature - that is, skill does change between games, and an inherent advantage does exist for players on the white side. However, the Glicko system that was mentioned accounts for skill changes, while the inherent advantage by side is averages out as over many matches an individual will approach a 50-50 game split on white vs. black. Lastly, it would be wise to also note that the Elo rating of one population, time period, and/or system cannot be directly compared to another.

Something not mentioned before is in regards to how matches are played in FIDE settings. Generally, players in FIDE tournaments play matches in round-robin format to some extent where they face multiple opponents before ratings are updated. This was too computationally intensive, however, the results would be quite similar (albeit with some lower accuracy in E_i calculations). Ratings for one match utilize actual and expected scores between 0 and 1, while ratings for multiple (round-robin) matches use scores between 0 and "m" matches. Thus in say, 8 matches, a players rating can move at most K*(8) where their expected and actual scores are summed across matches played.

Chapter 5

Conclusion

To wrap everything up, a quick summary will be given of the information covered thus far.

5.1 Background

FIDE, the international chess federation, assigns players a rating to measure their skill. This rating changes over time via a rating system (Elo's) and can be between 100 and $\sim$2900. The current distribution of ratings follows a lognormal one as mentioned earlier.

5.2 Setup

A player's skill is assumed to follow a bell-curve shaped distribution, notably a logistic one, with a scale that ensures a ratings difference of 400 indicates one is 10x more likely to win. A player's skill is also assumed to stay constant between games in that if an individual does not play for a month, their skill will remain the exact same.

Elo's rating system utilizes this logistic distribution to find the probability that one player wins or loses based on ratings, outputting an expected score between 0 and 1, while an actual score is determined off of a win (1), draw (0.5), or loss (0) and also between 0 and 1 for a single game.

Two simulation scenarios were examined; one where an individual enters a well-defined population, and another where a population is "initialized" with all-new players.

5.3 Findings

5.3.1 Individual vs. Population

In the scenario where an individual enters a pre-defined population, we were able to conclude that:

1. At high player counts, not grouping hurts efficiency, while grouping helps it. At low player counts, grouping hurts efficiency while not grouping helps it.

2. Grouping has little impact on error rate between low and high player counts.

3. An equilibrium player count exists where grouping and not grouping are similar in impact.

4. The number of matches required to converge is much lower when grouping, though the trade-off is that in cases of low player counts the converged error rate is much higher than when not grouping. In cases where there are enough players to reach a sub 1% error rate, grouping is drastically lower.

5.3.2 New Population

In the scenario where a new population is generated, we were able to conclude that:

1. The closer the average starting error is to 0%, the faster the population converges as a whole.

2. Grouping players allows for convergence to occur more quickly even at low player counts.

Bibliography

[1] Regan, K., Macieja, B., Haworth, G. - *Understanding Distributions of Chess Performances*
https://cse.buffalo.edu/ regan/papers/pdf/RMH11b.pdf

[2] Arpad Elo (1978), *The Rating of Chessplayers, Past and Present*,

[3] International Chess Federation
https://ratings.fide.com/calculator$_r$td.phtml

[4] Glickman, M., *The Glicko System*
http://www.glicko.net/glicko/glicko.pdf

[5] Glickman, M., *Example of the Glicko-2 system*
http://www.glicko.net/glicko/glicko2.pdf

[6] Glickman, M., *Rating the Chess Rating System*
http://www.glicko.net/research/chance.pdf

[7] Kaloumenos, M., *A brief comparison guide between the ELO and the Glicko rating systems*
https://www.englishchess.org.uk/wp-content/uploads/2012/04/Elo_vs_Glicko.pdf

Chapter 6

Appendix

The primary functions that simulations run on has been included here for convenience, though it is somewhat messy. Functions to plot have not been included, but can be sent on request.

6.1 General

```
function [GM,Ratings] = Initial_Ratings(p,Type,GM_Elo,Disparity)
    format long g

    %GENERAL CONTENT
    %Initialize Matrices
    Ratings = zeros(2,p);
    GM = zeros(2,1);

    %Section for True Ratings
    m = 1000 ; % mean
    v = 95000; %variance
    mu = log((m^2)/sqrt(v+m^2));
    sigma = sqrt(log(v/(m^2)+1));

    %Ensure Vector is Same in All Instances (so RNG seed is secure)
    True_Rating = lognrnd(mu,sigma,1,p);


    %TYPE SPECIFIC CONTENT
    %Select Version
```

```
    if isstring(Type) && Type == "GM"
        %Applying Ratings
        Ratings(1,:) = True_Rating; %Public Rating
        Ratings(2,:) = True_Rating; %True Rating

        %Setting GM Ratings
        GM(1) = 600; %Start GM's at 600
        GM(2) = GM_Elo; %GM

    elseif class(Type) == 'double'
        %Make for Any Number
        %Applying Ratings
        Ratings(1,:) = ones(1,p)*Type; %Public Rating = 600
        Ratings(2,:) = True_Rating; %True Rating

        GM = [];

    elseif isstring(Type) && Type == "Random"
        %Applying Ratings
        Ratings(1,:) = ceil(2900.*rand(1,p)); %Public Rating = U(0,2900)
        Ratings(2,:) = True_Rating; %True Rating

        GM = [];

    elseif isstring(Type) && Type == "Range"
        %Create Variation in Public Rating from True Rating
        Up_Down = rand(1,p);
        Up_Down(Up_Down <= 0.5) = -1;
        Up_Down(Up_Down > 0.5) = 1;

        Disparity = Up_Down*Disparity;
        Adjustment = 1+rand(1,p).*Disparity;

        %Applying Ratings
        Ratings(1,:) = Adjustment.*True_Rating; %Public Rating +/- RNG Disparity
        Ratings(2,:) = True_Rating; %True Rating

        GM = [];

    elseif isstring(Type) && Type == "Hybrid"
        Hybrid_Rand = rand(1,p);
        Hybrid_Rand(Hybrid_Rand <= 1/3) = 600;
        Hybrid_Rand(Hybrid_Rand <= 2/3) = 1000;
        Hybrid_Rand(Hybrid_Rand <= 3) = 1200;
```

```
        Ratings(1,:) = Hybrid_Rand; %Public Rating equally split 600 1000
    1200
        Ratings(2,:) = True_Rating; %True Rating

        GM = [];

    elseif isstring(Type) && Type == "HybridDist"
        Hybrid_Rand = rand(1,p);
        Hybrid_Rand(Hybrid_Rand <= 0.25) = 600;
        Hybrid_Rand(Hybrid_Rand <= (0.50+0.25)) = 1200;
        Hybrid_Rand(Hybrid_Rand <= (0.50+0.25+0.25)) = 1000;

        Ratings(1,:) = Hybrid_Rand; %Public Rating equally split 600 1000
    1200
        Ratings(2,:) = True_Rating; %True Rating

        GM = [];
    end

end
```

6.2 Individual vs. Population

```
function [Error, GM_Score] = GM_Simulation(n,m,p,GM_Elo,Group,Anchor,
    Disparity)
    %tic
    rng(10)

    %Initialize Error Matrix
    Error = zeros(n,m+1);
    GM_Score = zeros(n,m+1); %For Unique GM Plot
    %X_Line = zeros(n,1); %For Unique GM Plot

    for s=1:n %START OF SIMULATIONS (N)

        [GM, Ratings] = Initial_Ratings(p,"GM",GM_Elo,Disparity);

        %For error before any matches occur or "m=0"
        Error(s,1) = abs(GM(1) - GM(2))/GM(2); %As Percent of TR
        GM_Score(s,1) = GM(1);
        %X_Line(s,1) = 0;

        %Add Anchors for Groups
```

```
    if Anchor == "Y"
        Anchors = [150 300 500 700 900 1100 1300 1500 1700 1900 2100 2250 2350 2450 2600 2882; 150 300 500 700 900 1100 1300 1500 1700 1900 2100 2250 2350 2450 2600 2882];
        Ratings = [Ratings Anchors];
    end

    for i=1:m %START OF MATCHES (M)
        if Group == "Y"

            %Update Groupings Based on Anchor Status
            if Anchor == "Y"
                [ClassSSSp,ClassSSS,ClassSSp,ClassSS,ClassSp,ClassS,ClassA,ClassB,ClassC,ClassD,ClassE,ClassF,ClassG,ClassH,ClassI,ClassJ] = Groupings(p+width(Anchors),Ratings);
            else
                [ClassSSSp,ClassSSS,ClassSSp,ClassSS,ClassSp,ClassS,ClassA,ClassB,ClassC,ClassD,ClassE,ClassF,ClassG,ClassH,ClassI,ClassJ] = Groupings(p,Ratings);
            end

            %Now Set Ranges
            switch true
                case GM(1) >= 100 && GM(1) < 200
                    Pool = ClassJ;
                case GM(1) >= 200 && GM(1) < 400
                    Pool = ClassI;
                case GM(1) >= 400 && GM(1) < 600
                    Pool = ClassH;
                case GM(1) >= 600 && GM(1) < 800
                    Pool = ClassG;
                case GM(1) >= 800 && GM(1) < 1000
                    Pool = ClassF;
                case GM(1) >= 1000 && GM(1) < 1200
                    Pool = ClassE;
                case GM(1) >= 1200 && GM(1) < 1400
                    Pool = ClassD;
                case GM(1) >= 1400 && GM(1) < 1600
                    Pool = ClassC;
                case GM(1) >= 1600 && GM(1) < 1800
                    Pool = ClassB;
                case GM(1) >= 1800 && GM(1) < 2000
                    Pool = ClassA;
                case GM(1) >= 2000 && GM(1) < 2200
                    Pool = ClassS;
```

```
                case GM(1) >= 2200 && GM(1) < 2300
                    Pool = ClassSp;
                case GM(1) >= 2300 && GM(1) < 2400
                    Pool = ClassSS;
                case GM(1) >= 2400 && GM(1) < 2500
                    Pool = ClassSSp;
                case GM(1) >= 2500 && GM(1) < 2700
                    Pool = ClassSSS;
                case GM(1) >= 2700
                    Pool = ClassSSSp;
                otherwise
                    fprintf('A VALUE OF <100 OCCURED, ERROR ERROR ERROR, %f\n',i)
            end %END CASES

            %Exit Simulation if Nobody Left to Play
            if isempty(Pool)
                %fprintf("There is an empty pool/bin after %.0f matches!\n",i)
                %fprintf("We are in %s\n",Tag)
                Error(s,(i+1):m) = abs(GM(1) - GM(2))/GM(2);
                GM_Score(s,(i+1):m) = GM(1); %Unique GM Plot
                %X_Line(s) = i+1; %Unique GM Plot
                %i=m;
                break
            end

            %Select a Random Player
            Player = ceil(rand*width(Pool));

            %Calculate Expected/Actual Scores
            Player_ES = 1./(1+10.^((GM(1) - Pool(1,Player))/400));
            GM_ES = 1 - Player_ES;

            Player_AS = 1./(1+10.^((GM(2) - Pool(2,Player))/400));
            GM_AS = 1 - Player_AS;

            %Set Sensitivity by FIDE Regulations
            if GM(1) < 2300
                K = 40;
            elseif GM(1) < 2400
                K = 20;
            else
                K = 10;
            end
```

```
            %Update Rating(s)
            Old_GM = GM(1); %Used to update group score
            GM(1) = GM(1) + K*(GM_AS - GM_ES); %New GM Rating

            %Update GM Rating in Group
            switch true
                case Old_GM >= 100 && Old_GM < 200
                    ClassJ(1,Player) = Pool(1,Player);
                case Old_GM >= 200 && Old_GM < 400
                    ClassI(1,Player) = Pool(1,Player);
                case Old_GM >= 400 && Old_GM < 600
                    ClassH(1,Player) = Pool(1,Player);
                case Old_GM >= 600 && Old_GM < 800
                    ClassG(1,Player) = Pool(1,Player);
                case Old_GM >= 800 && Old_GM < 1000
                    ClassF(1,Player) = Pool(1,Player);
                case Old_GM >= 1000 && Old_GM < 1200
                    ClassE(1,Player) = Pool(1,Player);
                case Old_GM >= 1200 && Old_GM < 1400
                    ClassD(1,Player) = Pool(1,Player);
                case Old_GM >= 1400 && Old_GM < 1600
                    ClassC(1,Player) = Pool(1,Player);
                case Old_GM >= 1600 && Old_GM < 1800
                    ClassB(1,Player) = Pool(1,Player);
                case Old_GM >= 1800 && Old_GM < 2000
                    ClassA(1,Player) = Pool(1,Player);
                case Old_GM >= 2000 && Old_GM < 2200
                    ClassS(1,Player) = Pool(1,Player);
                case Old_GM >= 2200 && Old_GM < 2300
                    ClassSp(1,Player) = Pool(1,Player);
                case Old_GM >= 2300 && Old_GM < 2400
                    ClassSS(1,Player) = Pool(1,Player);
                case Old_GM >= 2400 && Old_GM < 2500
                    ClassSSp(1,Player) = Pool(1,Player);
                case Old_GM >= 2500 && Old_GM < 2700
                    ClassSSS(1,Player) = Pool(1,Player);
                case Old_GM >= 2700
                    ClassSSSp(1,Player) = Pool(1,Player);
                otherwise
                    fprintf('A VALUE OF <100 OCCURED, ERROR ERROR ERROR, %f\n',i)
            end

            %UPDATE RATINGS FOR NEXT LOOP
```

```
                Ratings = [ClassJ ClassI ClassH ClassG ClassF ClassE
ClassD ClassC ClassB ClassA ClassS ClassSp ClassSS ClassSSp ClassSSS
ClassSSSp];

                %Inputting Error after Match i
                %Error(s,i) = abs(GM(1) - GM(2));







           %SECTION FOR NON-GROUP
           else
                %Select a Random Player
                Player = ceil(rand*p);

                %Calculate Expected/Actual Scores
                Player_ES = 1./(1+10.^((GM(1) - Ratings(1,Player))/400));
                GM_ES = 1 - Player_ES;

                Player_AS = 1./(1+10.^((GM(2) - Ratings(2,Player))/400));
                GM_AS = 1 - Player_AS;

                %Set Sensitivity by FIDE Regulations
                if GM(1) < 2300
                    K = 40;
                elseif GM(1) < 2400
                    K = 20;
                else
                    K = 10;
                end

                %Update Rating(s) - Player is not Updated (@ True Skill
Alrdy)
                GM(1) = GM(1) + K*(GM_AS - GM_ES); %GM

           end %END OF GROUP IF/ELSE

           Error(s,i+1) = abs(GM(1) - GM(2))/GM(2); %As Percent of TR
           GM_Score(s,i+1) = GM(1); %Unique GM Plot
           %X_Line(s,i+1) = i; %Unique GM Plot

       end %END OF MATCHES (M)

```

```
    end %END OF SIMULATIONS (N)

    %TEMPORARY (EVENTUALL WILL BE IN PLOTTING FUNCTION)
    plot(mean(Error,1)*100)
    final_err = (mean(Error,1)*100);
    final_err = round(final_err(m-1),2);
    fprintf('Err: %f\n',final_err)
    ytickformat('percentage')
    %ylim([0 15])
    %REMINDER - WITH ANCHOR OFF THE GRAPH WILL LOOK WEIRD AS IT IS ENDING
    %EARLY

    %plot(mean(GM_Score,1))
    %yline(GM_Elo);
    %xline(mean(X_Line));

%     %FIGURE OUT TO PUT INTO PLOTTING FUNCTION
%     %CALL WITH "GM_Simulation(1,100,100,2500,"Y","N",0);"
%     if Anchor == "N"
%         %Removes Matches Missed
%         Error(:,all(Error == 0)) = [];
%
%         figure
%         hold on
%         %legend("",'Location','southwest')
%         for zz = 1:size(Error,1)
%             plot(Error(zz,:));
%         end
%     end
%     %FIGURE OUT TO PUT INTO PLOTTING FUNCTION

    %Time2Run = toc;

end %END FUNCTION
```

6.2.1 Groupings

```
function [ClassSSSp,ClassSSS,ClassSSp,ClassSS,ClassSp,ClassS,ClassA,ClassB
    ,ClassC,ClassD,ClassE,ClassF,ClassG,ClassH,ClassI,ClassJ] = Groupings(
    p,Ratings)
    format long g

    %Initialize Groupings
    ClassSSSp = zeros(2,p); %2700+
    ClassSSS = zeros(2,p); %2500 to %2699
```

```
    ClassSSp = zeros(2,p); %2400 to 2499
    ClassSS = zeros(2,p); %2300 to 2399
    ClassSp = zeros(2,p); %2200 to 2299
    ClassS = zeros(2,p); %2000 to 2199
    ClassA = zeros(2,p); %1800 to 1999
    ClassB = zeros(2,p); %1600 to 1799
    ClassC = zeros(2,p); %1400 to 1599
    ClassD = zeros(2,p); %1200 to 1399
    ClassE = zeros(2,p); %1000 to 1199
    ClassF = zeros(2,p); %800 to 999
    ClassG = zeros(2,p); %600 to 799
    ClassH = zeros(2,p); %400 to 599
    ClassI = zeros(2,p); %200 to 399
    ClassJ = zeros(2,p); %100 to 199

    for i=1:p
        PR = Ratings(1,i);
        TR = Ratings(2,i);

        %Assign Players to Group Bins
        switch true
            case PR<200
                ClassJ(1,i) = PR;
                ClassJ(2,i) = TR;
            case PR>=200 && PR<400
                ClassI(1,i) = PR;
                ClassI(2,i) = TR;
            case PR>=400 && PR<600
                ClassH(1,i) = PR;
                ClassH(2,i) = TR;
            case PR>=600 && PR<800
                ClassG(1,i) = PR;
                ClassG(2,i) = TR;
            case PR>=800 && PR<1000
                ClassF(1,i) = PR;
                ClassF(2,i) = TR;
            case PR>=1000 && PR<1200
                ClassE(1,i) = PR;
                ClassE(2,i) = TR;
            case PR>=1200 && PR<1400
                ClassD(1,i) = PR;
                ClassD(2,i) = TR;
            case PR>=1400 && PR<1600
                ClassC(1,i) = PR;
                ClassC(2,i) = TR;
```

```
            case PR>=1600 && PR<1800
                ClassB(1,i) = PR;
                ClassB(2,i) = TR;
            case PR>=1800 && PR<2000
                ClassA(1,i) = PR;
                ClassA(2,i) = TR;
            case PR>=2000 && PR<2200
                ClassS(1,i) = PR;
                ClassS(2,i) = TR;
            case PR>=2200 && PR<2300
                ClassSp(1,i) = PR;
                ClassSp(2,i) = TR;
            case PR>=2300 && PR<2400
                ClassSS(1,i) = PR;
                ClassSS(2,i) = TR;
            case PR>=2400 && PR<2500
                ClassSSp(1,i) = PR;
                ClassSSp(2,i) = TR;
            case PR>=2500 && PR<2700
                ClassSSS(1,i) = PR;
                ClassSSS(2,i) = TR;
            case PR>=2700
                ClassSSSp(1,i) = PR;
                ClassSSSp(2,i) = TR;
            otherwise
                fprintf('<100 ERROR IN GROUPING FUNCTION, %f\n',i)
        end

    end

    %Remove 0 Columns
    ClassSSSp(:,all(ClassSSSp == 0)) = [];
    ClassSSS(:,all(ClassSSS == 0)) = [];
    ClassSSp(:,all(ClassSSp == 0)) = [];
    ClassSS(:,all(ClassSS == 0)) = [];
    ClassSp(:,all(ClassSp == 0)) = [];
    ClassS(:,all(ClassS == 0)) = [];
    ClassA(:,all(ClassA == 0)) = [];
    ClassB(:,all(ClassB == 0)) = [];
    ClassC(:,all(ClassC == 0)) = [];
    ClassD(:,all(ClassD == 0)) = [];
    ClassE(:,all(ClassE == 0)) = [];
    ClassF(:,all(ClassF == 0)) = [];
    ClassG(:,all(ClassG == 0)) = [];
    ClassH(:,all(ClassH == 0)) = [];
```

```
    ClassI(:,all(ClassI == 0)) = [];
    ClassJ(:,all(ClassJ == 0)) = [];

end
```

6.3 New Players

```
function [Error, Empty_Brackets, Orphans] = Base_Simulation(n,m,p,GM_Elo,
    Group,Anchor,Disparity,Type)
    %tic
    rng(10)

    %POTENTIALLY ADD - IF LAST ENTRY IN POOL FOR BIN (I.E. WHAT SHOULD BE
    %ANCHOR) THEN DO NOT UPDATE THEIR SKILL! Can just do if else statement

    %To make anchor allowed when group is N, need to update "p=player" to
    %be 16 higher

    %Initialize Error Matrix
    %Error = zeros(n,m);
    Error = zeros(n,m+1); %m+1 so that the first index is match "0" or
    initial error
    Empty_Brackets = zeros(n,m);
    Orphans = zeros(n,m);

    for s=1:n %START OF SIMULATIONS (N)

        [GM, Ratings] = Initial_Ratings(p,Type,GM_Elo,Disparity);

        %For error before any matches occur or "m=0"
        Error(s,1) = mean(abs(Ratings(1,:) - Ratings(2,:))./Ratings(2,:));
     %As Percent

        %Add Anchors
        if Anchor == "Y"
            Anchors = [150 300 500 700 900 1100 1300 1500 1700 1900 2100
    2250 2350 2450 2600 2882; 150 300 500 700 900 1100 1300 1500 1700 1900
     2100 2250 2350 2450 2600 2882];
            Ratings = [Ratings Anchors];
        end

        for i=1:m %START OF MATCHES (M)

            if Group == "N"
```

```
            %Select 2 Random Players
            A = ceil(rand*p);
            B = ceil(rand*p);

            while A == B %Prevent the "same person" playing
                A = ceil(rand*p);
            end

            %Calculate Expected/Actual Scores
            A_ES = 1./(1+10.^((Ratings(1,B) - Ratings(1,A))/400));
            B_ES = 1 - A_ES;

            A_AS = 1./(1+10.^((Ratings(2,B) - Ratings(2,A))/400));
            B_AS = 1 - A_AS;

            %Set Sensitivity by FIDE Regulations (FOR PLAYER A)
            if Ratings(1,A) < 2300
                K = 40;
            elseif Ratings(1,A) < 2400
                K = 20;
            else
                K = 10;
            end

            %Update Player A Rating
            if Anchor == "Y" && ismember(Ratings(1,A),Anchors(1,:)) && ismember(Ratings(2,A),Anchors(2,:)) %DO NOT UPDATE ANCHOR RATINGS
                Ratings(1,A) = Ratings(1,A);
            else
                Ratings(1,A) = Ratings(1,A) + K*(A_AS - A_ES);
            end

            %Set Sensitivity by FIDE Regulations (FOR PLAYER B)
            if Ratings(1,B) < 2300
                K = 40;
            elseif Ratings(1,B) < 2400
                K = 20;
            else
                K = 10;
            end

            %Update Player B Rating
            if Anchor == "Y" && ismember(Ratings(1,B),Anchors(1,:)) && ismember(Ratings(2,B),Anchors(2,:)) %DO NOT UPDATE ANCHOR RATINGS
                Ratings(1,B) = Ratings(1,B);
```

```
                else
                    Ratings(1,B) = Ratings(1,B) + K*(B_AS - B_ES);
                end
                %Ratings(1,B) = Ratings(1,B) + K*(B_AS - B_ES);






            else %IE Grouped
                RP = Ratings(1,:);
                RT = Ratings(2,:);

                %Create Class Ratings
                if Anchor == "Y"
                    Class_Matrix_PR = zeros(16,p+width(Anchors));
                    Class_Matrix_TR = zeros(16,p+width(Anchors));
                else
                    Class_Matrix_PR = zeros(16,p);
                    Class_Matrix_TR = zeros(16,p);
                end

                ClassSSSp_PR = RP(RP >= 2700);
                ClassSSSp_TR = RT(RP >= 2700);
                ClassSSS_PR = RP(RP >= 2500 & RP < 2700);
                ClassSSS_TR = RT(RP >= 2500 & RP < 2700);
                ClassSSp_PR = RP(RP >= 2400 & RP < 2500);
                ClassSSp_TR = RT(RP >= 2400 & RP < 2500);
                ClassSS_PR = RP(RP >= 2300 & RP < 2400);
                ClassSS_TR = RT(RP >= 2300 & RP < 2400);
                ClassSp_PR = RP(RP >= 2200 & RP < 2300);
                ClassSp_TR = RT(RP >= 2200 & RP < 2300);
                ClassS_PR = RP(RP >= 2000 & RP < 2200);
                ClassS_TR = RT(RP >= 2000 & RP < 2200);
                ClassA_PR = RP(RP >= 1800 & RP < 2000);
                ClassA_TR = RT(RP >= 1800 & RP < 2000);
                ClassB_PR = RP(RP >= 1600 & RP < 1800);
                ClassB_TR = RT(RP >= 1600 & RP < 1800);
                ClassC_PR = RP(RP >= 1400 & RP < 1600);
                ClassC_TR = RT(RP >= 1400 & RP < 1600);
                ClassD_PR = RP(RP >= 1200 & RP < 1400);
                ClassD_TR = RT(RP >= 1200 & RP < 1400);
                ClassE_PR = RP(RP >= 1000 & RP < 1200);
                ClassE_TR = RT(RP >= 1000 & RP < 1200);
```

```
                ClassF_PR = RP(RP >= 800 & RP < 1000);
                ClassF_TR = RT(RP >= 800 & RP < 1000);
                ClassG_PR = RP(RP >= 600 & RP < 800);
                ClassG_TR = RT(RP >= 600 & RP < 800);
                ClassH_PR = RP(RP >= 400 & RP < 600);
                ClassH_TR = RT(RP >= 400 & RP < 600);
                ClassI_PR = RP(RP >= 200 & RP < 400);
                ClassI_TR = RT(RP >= 200 & RP < 400);
                ClassJ_PR = RP(RP < 200);
                ClassJ_TR = RT(RP < 200);

                Class_Matrix_PR(1,1:width(ClassSSSp_PR)) = ClassSSSp_PR;
                Class_Matrix_TR(1,1:width(ClassSSSp_TR)) = ClassSSSp_TR;
                Class_Matrix_PR(2,1:width(ClassSSS_PR)) = ClassSSS_PR;
                Class_Matrix_TR(2,1:width(ClassSSS_TR)) = ClassSSS_TR;
                Class_Matrix_PR(3,1:width(ClassSSp_PR)) = ClassSSp_PR;
                Class_Matrix_TR(3,1:width(ClassSSp_TR)) = ClassSSp_TR;
                Class_Matrix_PR(4,1:width(ClassSS_PR)) = ClassSS_PR;
                Class_Matrix_TR(4,1:width(ClassSS_TR)) = ClassSS_TR;
                Class_Matrix_PR(5,1:width(ClassSp_PR)) = ClassSp_PR;
                Class_Matrix_TR(5,1:width(ClassSp_TR)) = ClassSp_TR;
                Class_Matrix_PR(6,1:width(ClassS_PR)) = ClassS_PR;
                Class_Matrix_TR(6,1:width(ClassS_TR)) = ClassS_TR;
                Class_Matrix_PR(7,1:width(ClassA_PR)) = ClassA_PR;
                Class_Matrix_TR(7,1:width(ClassA_TR)) = ClassA_TR;
                Class_Matrix_PR(8,1:width(ClassB_PR)) = ClassB_PR;
                Class_Matrix_TR(8,1:width(ClassB_TR)) = ClassB_TR;
                Class_Matrix_PR(9,1:width(ClassC_PR)) = ClassC_PR;
                Class_Matrix_TR(9,1:width(ClassC_TR)) = ClassC_TR;
                Class_Matrix_PR(10,1:width(ClassD_PR)) = ClassD_PR;
                Class_Matrix_TR(10,1:width(ClassD_TR)) = ClassD_TR;
                Class_Matrix_PR(11,1:width(ClassE_PR)) = ClassE_PR;
                Class_Matrix_TR(11,1:width(ClassE_TR)) = ClassE_TR;
                Class_Matrix_PR(12,1:width(ClassF_PR)) = ClassF_PR;
                Class_Matrix_TR(12,1:width(ClassF_TR)) = ClassF_TR;
                Class_Matrix_PR(13,1:width(ClassG_PR)) = ClassG_PR;
                Class_Matrix_TR(13,1:width(ClassG_TR)) = ClassG_TR;
                Class_Matrix_PR(14,1:width(ClassH_PR)) = ClassH_PR;
                Class_Matrix_TR(14,1:width(ClassH_TR)) = ClassH_TR;
                Class_Matrix_PR(15,1:width(ClassI_PR)) = ClassI_PR;
                Class_Matrix_TR(15,1:width(ClassI_TR)) = ClassI_TR;
                Class_Matrix_PR(16,1:width(ClassJ_PR)) = ClassJ_PR;
                Class_Matrix_TR(16,1:width(ClassJ_TR)) = ClassJ_TR;

                for c=1:16 %Class Bins
```

```
                %Set Pool Size to Class Size
                Pool = nnz(Class_Matrix_PR(c,:));

                if Pool == 0
                    Empty_Brackets(s,i) = Empty_Brackets(s,i) + 1;
                    continue

                elseif Pool == 1
                    Orphans(s,i) = Orphans(s,i) + 1;
                    continue
                end

                %Select 2 Players
                A = ceil(rand*Pool);
                B = ceil(rand*Pool);

                while A == B %Prevent the "same person" playing
                    A = ceil(rand*Pool);
                end

                %Calculate Expected/Actual Scores
                A_ES = 1./(1+10.^((Class_Matrix_PR(c,B) - Class_Matrix_PR(c,A))/400));
                B_ES = 1 - A_ES;

                A_AS = 1./(1+10.^((Class_Matrix_TR(c,B) - Class_Matrix_TR(c,A))/400));
                B_AS = 1 - A_AS;

                %Set Sensitivity by FIDE Regulations (FOR PLAYER A)
                if Class_Matrix_PR(c,A) < 2300
                    K = 40;
                elseif Class_Matrix_PR(c,A) < 2400
                    K = 20;
                else
                    K = 10;
                end

                %Update Player A Rating - ADJUSTED TO NOT FOR ANCHORS
                if Anchor == "Y" && ismember(Class_Matrix_PR(c,A), Anchors(1,:)) && ismember(Class_Matrix_TR(c,A),Anchors(2,:)) %DO NOT UPDATE ANCHOR RATINGS
                    Class_Matrix_PR(c,A) = Class_Matrix_PR(c,A); %i.e. does not update if anchor
                else
```

```
                        Class_Matrix_PR(c,A) = Class_Matrix_PR(c,A) + K*(A_AS - A_ES);
                    end

                    %Set Sensitivity by FIDE Regulations (FOR PLAYER B)
                    if Class_Matrix_PR(c,B) < 2300
                        K = 40;
                    elseif Class_Matrix_PR(c,B) < 2400
                        K = 20;
                    else
                        K = 10;
                    end

                    %Update Player B Rating
                    if Anchor == "Y" && ismember(Class_Matrix_PR(c,B),Anchors(1,:)) && ismember(Class_Matrix_TR(c,B),Anchors(2,:)) %DO NOT UPDATE ANCHOR RATINGS
                        Class_Matrix_PR(c,B) = Class_Matrix_PR(c,B); %i.e. does not update if anchor
                    else
                        Class_Matrix_PR(c,B) = Class_Matrix_PR(c,B) + K*(B_AS - B_ES);
                    end
                    %Class_Matrix_PR(c,B) = Class_Matrix_PR(c,B) + K*(B_AS - B_ES);

                end %End of Class Bins

                %Update Ratings Matrix
                Ratings(1,:) = nonzeros(Class_Matrix_PR)';
                Ratings(2,:) = nonzeros(Class_Matrix_TR)';


            end %END OF GROUP IF/ELSE

            %Inputting Error after Simulation s, Match i
            if Anchor == "N"
                Error(s,i+1) = mean(abs(Ratings(1,:) - Ratings(2,:))./Ratings(2,:)); %As Percent
            elseif Anchor == "Y"
                Error(s,i+1) = sum(abs(Ratings(1,:) - Ratings(2,:))./Ratings(2,:),2)/(p); %As Percent - not including anchors!
            end

        end %END OF MATCHES (M)
```

```
    end %END OF SIMULATIONS (N)

    %timer = toc

    %mean(mean(Error,1))
    %hold on
    plot(mean(Error,1)*100)
    %ylim([0 85]);
    %lgnd = sprintf('%.0f',Type);
    %legend(lgnd);
    %ytickformat('percentage')
    %plot(0:length(mean(Error,1))-1,mean(Error,1)*100);

end %END FUNCTION
```