Analyzing Elo Ratings For NCAA Men's Division 1 Hockey

Alex Tidd Advisors: Matt Higham and Robin Lock

2025-05-07

Table of contents

0.1 Abstract

There are 64 NCAA Division 1 Men's Hockey teams. In NCAA Division 1 Men's Hockey, the U.S. College Hockey Online (USCHO) provides the official rankings on the 64 total teams using a system that relies on expert votes. However, this method is not perfect as there is no formal quantitative analytics involved in the voting. To improve on this, a chess ratings system, called Elo, is commonly used in many sports to quantitatively rate players, weighing strength of opponent and recency of match. In this project, we modify the Elo system for NCAA Division 1 Men's Hockey by adding in weights for game goal differential (so that games in which the score differential was large potentially result in a larger bump in Elo for the winning team) and home-ice advantage (so that the team playing on home-ice has an adjusted probability of winning the game).

0.2 Introduction

NCAA Men's Division 1 Ice Hockey has 64 teams. This inherently makes ranking teams extremely hard. It is impossible to rank teams in the same manner as pro leagues like the NHL and junior leagues like the 3 CHL leagues, where teams are ranked based off of a record-point system. This is due to the fact that many teams end up with similar records, but with a league so large and full of different conferences, the strength of a victory/loss is not the same for every game. Some conferences are weaker than others, so if a weak-conference playing team has a great record, it would not be fair to rank them higher than a team in a strong conference with a mediocre record.

To rank teams, the current method is an "expert vote" system done by US College Hockey Online (USCHO). This method takes votes by "experts" and ranks teams based on how many votes each team receives. In general, these rankings are pretty accurate. However, it lacks any true quantitative analysis, instead relying on the opinions of voters.

This project aims to solve the issue presented by using a ratings system used in chess to accurately rate and rank players, called Elo, for use in collegiate hockey. In chess, Elo ratings calculate an "expected outcome" using 2 players' ratings. This expected outcome is essentially a "win probability" for each player. After the expected outcome is calculated, it is compared to the actual outcome. Depending on the outcome, a player's new rating will either be increased (if the player won) or decreased (if the player lost). The benefit to using Elo, is that the rating system takes into consideration your strength, the opponent's strength, and the recency of the match. Big wins/losses garner big adjustments in rating, whereas an expected win/loss won't inflate/tarnish a rating very much. The equation for expected outcome is:

$$E_a = \frac{1}{1 + 10^{\frac{R_b - R_a}{400}}}$$

Where:

- E_a is the player's expected outcome (Palyer A's win probability).
- R_a is the player's pregame rating.
- R_b is the opponent's pregame rating.

To update the rating after the match is played, the update equation is:

$$R'_a = R_a + k(\text{outcome} - E_a)$$

Where:

- R'_a is the player's postgame rating.
- R_a is the player's pregame rating
- k is an update factor that scales the amount of points a player changes by.

- outcome is the outcome of the match, 1 for a win, 0 for a loss.
- E_a is the player's expected outcome. In other words, the probability that Player A wins.

Here is an example of how the Elo system works:

Player A has a pregame rating of 2000 and beats Player B with a rating of 2200. To find Player A's new rating (k is set to 100 for this example):

$$E_a = \frac{1}{1 + 10^{\frac{2200-2000}{400}}}$$

$$E_a = 0.2402531$$

$$R'_a = 2000 + 100(1 - 0.2402531)$$

$$R'_a = 2075.975$$

We see that Player A's new rating is 2075.975. It grew by almost 76 points due to them beating a player they were not expecting to beat. Player A was given a win probability of 0.2402531 compared to Player B's 0.7597469. Had Player A had a pregame rating off 2200 and Player B had a pregame rating of 2000, with Player A still winning, Player A's new rating would be 2224.025, only a 24.025 point increase, as Player A was expected to win.

The goal for this project is to use this base Elo ratings system, incorporate factors such as home ice advantage and goal differential, optimize k, and make a ratings model that can accurately predict outcome and rank teams accordingly. This paper shows the steps of finding data, creating a ratings function, and optimizing our constants.

0.3 Data

There were two main data sets used for this project. The full 2023-2024 NCAA Men's Division 1 Ice Hockey schedule, which after wrangling, included 1166 games with variables: date, game_type, away_team, away_score, home_team, home_score, overtime, neutral_site,

score_diff, and outcome. The score_diff and outcome variables are in reference to the home team, with outcome being either 1: win, 0.5: tie, or 0: loss. The next data frame was the entire 2024-2025 NCAA Men's Division 1 Ice Hockey schedule. The schedule contains 1153 games which had the same variables and setup as the 2023-2024 season. Both data frames were scraped from the website: College Hockey News.

2023-2024 Season:

```
schedule2324 |> slice(1:4)
```

```
# A tibble: 4 x 10
  date       game_type      away_team   away_score home_team home_score overtime
  <date>     <chr>          <chr>            <dbl> <chr>          <dbl> <lgl>
1 2023-10-07 Non-Conference Lake Super~          2 Michigan~          5 FALSE
2 2023-10-07 Non-Conference Mass.-Lowe~          2 Alaska-A~          3 TRUE
3 2023-10-07 Non-Conference Denver               7 Alaska             3 FALSE
4 2023-10-07 Non-Conference Lindenwood           1 Air Force          4 FALSE
# i 3 more variables: neutral_site <lgl>, score_diff <dbl>, outcome <dbl>
```

2024-2025 Season:

```
schedule |> slice(1:4)
```

```
# A tibble: 4 x 10
  date       game_type      away_team   away_score home_team home_score overtime
  <date>     <chr>          <chr>            <dbl> <chr>          <dbl> <lgl>
1 2024-10-04 Non-Conference Michigan S~          2 Lake Sup~          1 TRUE
2 2024-10-04 Non-Conference Minnesota ~          5 Michigan           2 FALSE
3 2024-10-04 Non-Conference Arizona St~          8 Air Force          1 FALSE
4 2024-10-04 Non-Conference Bowling Gr~          2 Mercyhur~          1 FALSE
# i 3 more variables: neutral_site <lgl>, score_diff <dbl>, outcome <dbl>
```

A third data frame for initial rankings was created from scratch using knowledge from the final 2022-2023 season. Initial rankings were created by ranking teams from 2000 to 1300. The top 8 teams from the 2022-2023 season were assigned the rating of 2000. The ratings were then decreased by 100 every 8 teams. This initial ratings file was used as the initial ratings file to run with the 2023-2024 season to get starting ratings for the 2024-2025 season.

To get these "accurate" ratings for the 2024-2025 schedule, the 2023-2024 schedule was run through the Elo function with k = 100, a value used based off a general exploration which gave relatively accurate rankings compared to the final 2023-2024 season. After, the ratings were scaled using the equation below:

$$ratings_{pre} = (ratings_{final} \cdot 0.7) + 450$$

Where:

- $ratings_{pre}$ is the preseason ratings.
- $ratings_{final}$ is the final ratings from the previous season.

This adjustment of rankings is used to adjust for player turnover, new coaching hires, and off season improvement. The formula used is based off of the formula used by Fivethirtyeight in their NHL Elo ratings. These final adjusted ratings are what was used for the initial rating in the 2024-2025 season.

0.4 Elo Model

To add goal differential into the Elo system, it was determined that it should be placed in the "rating update" portion of the Elo system, due to goal differential being a "post-game" statistic. On the other hand, home ice advantage affects the predicted probability of a home team winning. Therefore, it was added to the "expected outcome calculation" portion of the Elo system. Home ice advantage was also incorporated as a simple point "boost" for the home team instead of a multiplier, based off of FiveThirtyEight's NHL Elo model.

In order to obtain the "best" values for each parameter, a grid search method was used to optimize each of the three parameters to lower the mean absolute residual of the season. In this case, the "absolute residual" is calculated by the absolute value of the difference of the game outcome and the expected outcome of the home team. Essentially we want the smallest

difference between actual outcome and expected outcome. The smaller the difference, the more accurate our function is predicting actual outcomes of games.

The un-optimized Elo function is as follows:

$$E_{home} = \frac{1}{1 + 10^{\frac{R_{away} - (R_{home} + \text{homeIce})}{400}}}$$

$$E_{away} = \frac{1}{1 + 10^{\frac{(R_{home} + \text{homeIce}) - R_{away}}{400}}}$$

Where:

- E_{home} is the expected outcome of the home team (win probability).
- E_{away} is the expected outcome of the away team (win probability).
- R_{home} is the pregame rating of the home team.
- R_{away} is the pregame rating of the away team.
- homeIce is the home ice advantage factor.

Using the calculated E_{home} and E_{away}, the new ratings for the home and away teams are calculated with the equations below:

$$R'_{home} = R_{home} + k(d(\text{scoreDiff}) + (\text{outcome} - E_{home}))$$

$$R'_{away} = R_{away} + k(d(\text{scoreDiff}) + (\text{outcome} - E_{away}))$$

Where:

- R'_{home} is the post game rating of the home team.

- R'_{away} is the post game rating of the away team.
- R_{home} is the pregame rating of the home team.
- R_{away} is the pregame rating of the away team.
- k is the update factor, scaling the amount of points a team gains from a win or loss.
- d is the goal differential factor.
- scoreDiff is the score differential in relation to the home team.
- outcome is the outcome of a game, 1 for a win, 0.5 for a tie, 0, for a loss.
- E_{home} is the expected outcome of the home team.
- E_{away} is the expected outcome of the away team.

Using the above model a general exploration into possible values of k, d, and homeIce could render a lower mean absolute residual. Based off the exploration, k is found at roughly 100, d at roughly 50, and homeIce at roughly 50. It should be noted that k was explored first with $d = 0$, and homeIce $= 0$. Next was homeIce, which was found at varying weights of k and $d = 0$. d was then explored with $k = 100$, and homeIce $= 0$. The only reason k is held in all three cases, is that if $k = 0$, the function would not be able to update rating as the update portion would always be $R_{home} + 0$ or $R_{away} + 0$.

These explorations provided essential knowledge for the grid search as initial optimization could be found using smaller ranges of values, significantly decreasing the run time of each optimization. This background knowledge cut run time of one optimization dawn from 16.3 days to roughly 30 minutes as each optimization grid search used 1000 combinations of the three variables instead of 4.5 million.

Using this method, k was found to be optimized at 37.22222, homeIce optimized at 53.33333, and d optimized at 40.55556. This provided a mean absolute residual of 0.3515844. This looked very promising at first glance. However, upon deeper inspection, two major concerns

were found. First, the bottom four teams had end of season ratings in the negatives. Second, there were many residuals of 1, -1, and 0. This raised the most concern as expected outcome is bounded by $0 <$ expected outcome < 1, meaning we should never see residuals of exactly 1 or 0, unless the predicted outcome is exactly 1 or 0. Upon deeper inspection, it was found that there were many games were expected outcome was to the effect of 0.9 repeating or 1e-20. While technically within our bounds, are highly unrealistic due to the incredibly high and low probabilities. A good model that predicts win probability for college hockey should never give a pregame win probability of nearly 100% or 0%. Something needed to be changed with the model. Goal differential was looked at first.

Upon inspecting and researching different ways to incorporate goal differential in an Elo function, FiveThirtyEight was looked at again. In their NHL model, goal differential, d, was incorporated as shown below:

$$(0.6686 \cdot ln(\text{scoreDiff})) + 0.8048)$$

This approach is much more stable for a few reasons. First, a logarithmic function levels out at the upper a lower bounds. This mimics real life as there is much of a difference in blowout games. A 7-0 game and a 12-0 game are both considered relatively equal games as they're both blowouts. This stops team from gaining huge amounts of points by really blowing out a team well after it is already "blown out". The function in addition to downplaying the effects of extremes, also contains all the ratings in the positive ratings. This stabilizes the ratings so we don't see extreme jumps in rating update from an upset win, and won't tank a team for being upset.

The new update function now looks as follows:

$$R'_{home} = R_{home} + k(((0.6686 \cdot ln(\text{scoreDiff})) + 0.8048)(\text{outcome} - E_{home}))$$

$$R'_{away} = R_{away} + k(((0.6686 \cdot ln(\text{scoreDiff})) + 0.8048)(\text{outcome} - E_{away}))$$

0.8048 is added into the d factor to account for games where scoreDiff $= 1$ since $ln(1) = 0$. This means that any game where scoreDiff $= |1|$ d is equal to 0.8048.

Now that goal factor is optimized, all that was left was to rerun the grid search to optimize k and homeIce. Using the grid search method, k is optimized at 88 and homeIce optimized at 40, which gives a mean absolute residual of 0.3876531. Whilst this may not have a mean absolute residual lower than the first model, all teams maintain ratings above zero and keep realistic predicted probabilities.

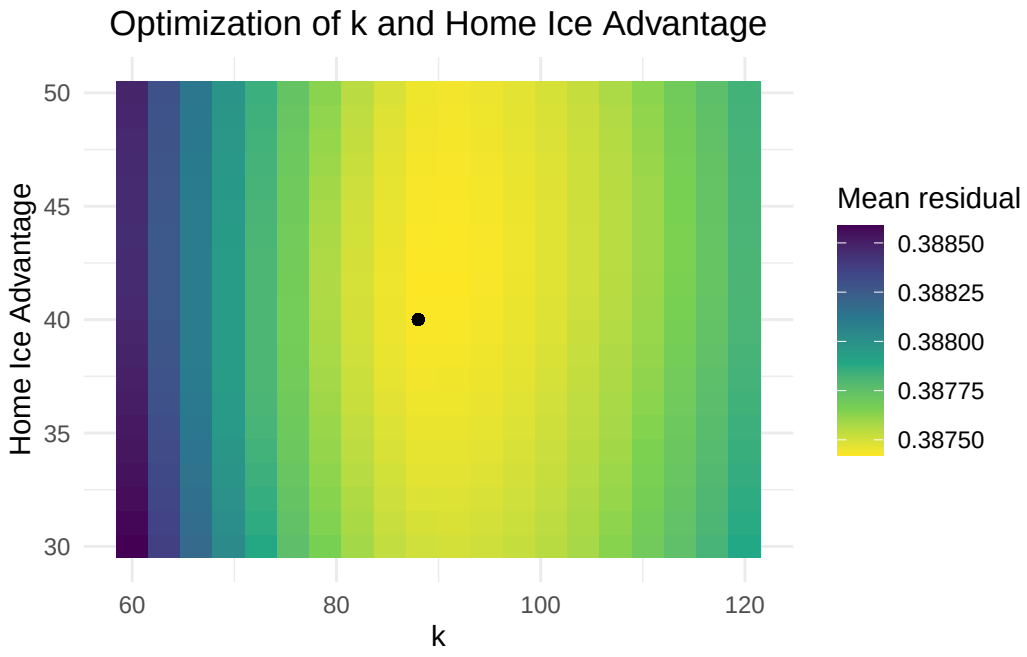
0.5 Results

Results from this optimization show that $k = 88$, homeIce $= 40$, and the calculation to utilize $d = (0.6686 \cdot ln(\text{scoreDiff})) + 0.8048)$.

Below is the grid search for k and homeIce. Multiple grid searches were conducted based on the explorations until the sequence was narrowed to a range of 10. The plot below shows a large range for both k and d to aid in mapping visual.

```
library(furrr)
library(progressr)
library(scico)
plan(multisession)
handlers("progress")
options(progressr.enable = TRUE)

grid_100 = expand.grid(k = seq(60, 120, length.out = 20),
                       home_ice = seq(30, 50, length.out = 20),
                       d = seq(0, 100, length.out = 1))

mean_residuals_100 =
  with_progress(
    {future_pmap_dbl(grid_100, \
                     (k, home_ice, d) update_rankings_residuals(season = schedule_reg,
```

```
                                                                    end_date = "2025-03-25",
                                                                    ratings = rankings2324,
                                                                    k = k,
                                                                    home_ice = home_ice,
                                                                    d = d), .progress = TRUE)})

residual_100_df <- grid_100 |> mutate(mean_residual = mean_residuals_100)

ggplot(data = residual_100_df, aes(x = k,
                                   y = home_ice)) +
   geom_tile(aes(fill = mean_residual)) +
  geom_point(aes(x = 88, y = 40), color = "black", fill = "black") +
  scale_fill_viridis_c(option = "D",
                       oob = scales::squish,
                       name = "Mean residual",
                       direction = -1) +
  labs(x = "k",
       y = "Home Ice Advantage",
       title = "Optimization of k and Home Ice Advantage") +
  theme_minimal() +
  theme(legend.position = "right",
        plot.title = element_text(hjust = 0.5))
```



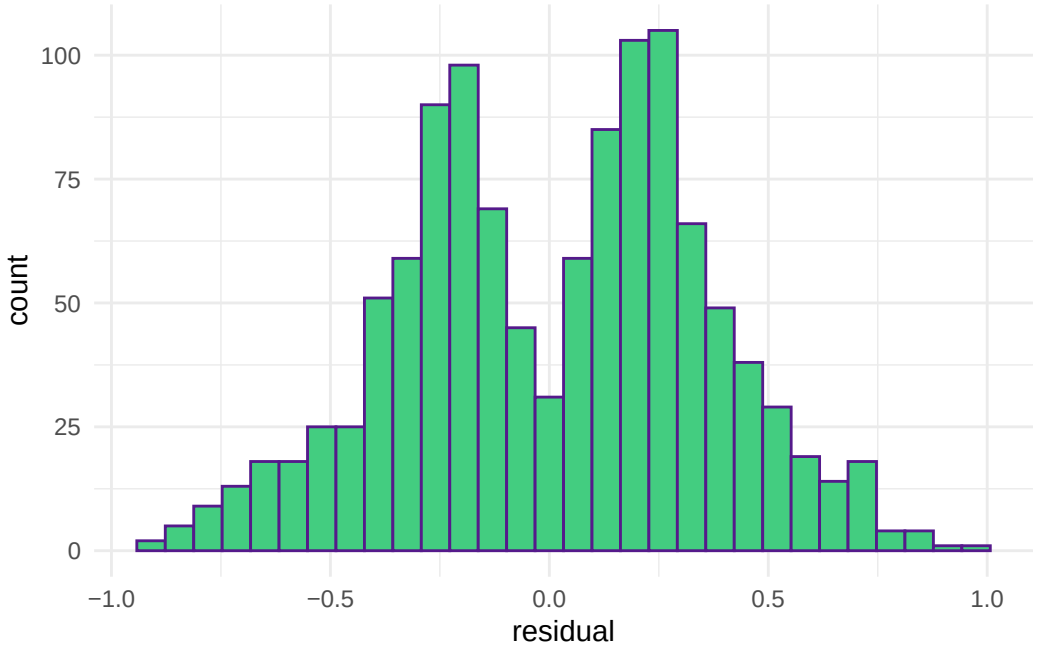
The heat map showing the grid search to d, k, and homeIce. The factors are all optimized to

lower the mean absolute value of the difference between expected outcome and actual outcome. In this case, the optimized values were able to lower the mean absolute value of the residuals to 0.3874257. d was held at $(0.6686 \cdot ln(\text{scoreDiff})) + 0.8048$. $k = 88$, homeIce $= 40$.

After finding the optimized values for k and homeIce using $d = (0.6686 \cdot ln(\text{scoreDiff})) + 0.8048$, the entire season was run using the update_rankings_residuals() function. This function allowed us to look at residuals for every game in the 2024-2025 season. To see how the function was behaving with respects to home teams and away teams, a plot of the residuals was looked at instead of mean residuals, which take away the ability to investigate how the function deals with home and away teams. A plot of the residuals of the 2024-2025 season is shown below.

```
apr15 = apr_15_ranking |> bind_rows()

schedule_elo = schedule |>
  mutate(home_elo = NA) |>
  mutate(away_elo = NA)

schedule_apr15 = left_join(schedule_elo, apr15,
                           by = join_by(date == date, home_team == Team)) |>
  mutate(home_elo = rating) |>
  select(-rating)

merged_sched_apr15 = left_join(schedule_apr15, apr15,
                           by = join_by(date == date, away_team == Team)) |>
  mutate(away_elo = rating) |>
  select(-rating)

schedule_full_apr15 = merged_sched_apr15 |>
  mutate(outcome_away = abs(outcome - 1)) |>
  ## Calculating expected outcome variable for home and away team
  mutate(exp_home = 1/(1 + 10^((away_elo - home_elo)/400))) |>
  mutate(exp_away = 1/(1 + 10^((home_elo - away_elo)/400))) |>
  ## Using expected outcome variable to generate new Elo ratings
  ## based on actual outcome and expected outcome
  mutate(elo_new_home = home_elo + 100 * (outcome - exp_home)) |>
  mutate(elo_new_away = away_elo + 100 * (outcome_away - exp_away))

## Making a residual column
schedule_full_apr15 <- schedule_full_apr15 |>
  mutate(residual = outcome - exp_home) |>
```

```
  mutate(abs_residual = abs(residual))

ggplot(data = schedule_full_apr15, aes(x = residual)) +
  geom_histogram(color = "purple4", fill = "seagreen3") +
  theme_minimal()
```



The plot of residuals shows all the differences between expected outcome and actual outcome. The positive residuals past 0.5 indicates the the model is predicting a home loss when actual result is a home win. Negative residuals beyond -0.5 indicates that the model predicted a home win with an observed home loss. We see that the model has large amounts of residuals around the |0.25| which means that the expected outcome is still within the range of the actual outcome. For example, actual outcome is 0, and expected is 0.25, the model still predicted a loss. The only grey area is around ties, since ties are 0.5, a |0.25| difference could be construed as tie or a win/loss.

After looking into the residuals, we wanted to test just how accurate the predicted win probabilities were. To do this, we took the entire season and binned each game by expected value in increments of 0.1. We then plotted the expected values against the proportion of

games within the binned expectation that resulted in a home win. We expect a linear relation: $\hat{p}_{win} = \hat{expected}_{binned}$. The expected relation is showing that of all teams with a certain predicted win proportion, the proportion of times those teams win should be equal to that predicted win proportion. Lastly, each binned expected value is weighted by the amount of games in each bin so the linear model isn't skewed by bins with minimal games. The plot is shown below.

```
schedule_full_apr15 |> summarise(mean_resid = mean(abs_residual, na.rm = TRUE))
```

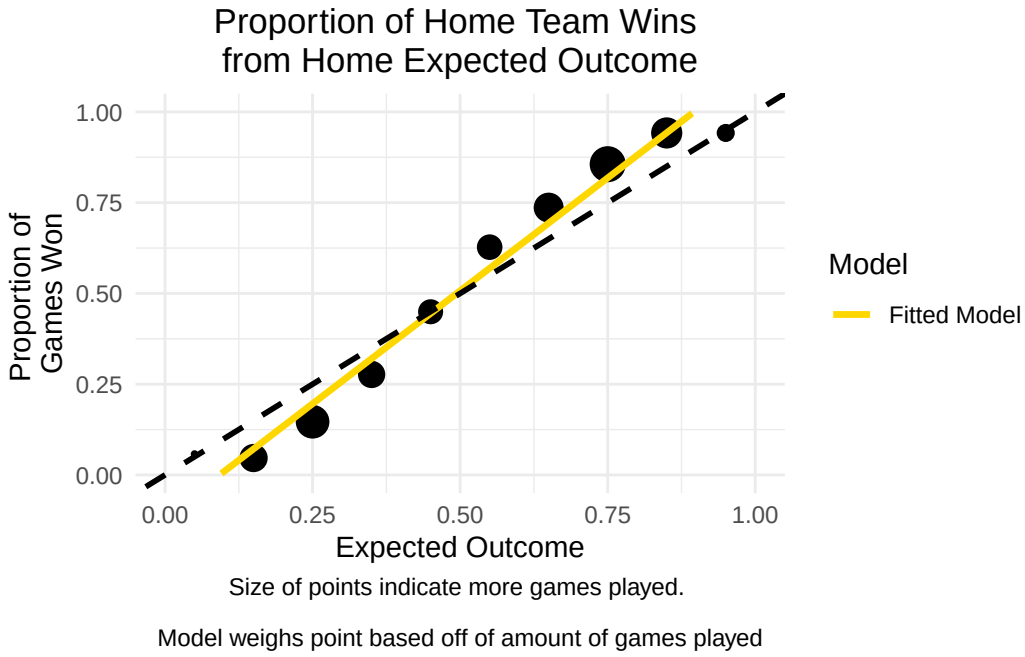
```
# A tibble: 1 x 1
  mean_resid
       <dbl>
1      0.294
```

```
prop_wins15 <- schedule_full_apr15 |>
  mutate(binned_exp = floor(exp_home / 0.1) * 0.1 + 0.05) |>
  group_by(binned_exp) |>
  summarise(win_prop = mean(outcome, na.rm = TRUE),
            totalgames = n()) |>
  filter(!is.na(binned_exp))

modgdha = lm(win_prop ~ binned_exp, data = prop_wins15, weights = totalgames)
summary(modgdha)
```

```
Call:
lm(formula = win_prop ~ binned_exp, data = prop_wins15, weights = totalgames)

Weighted Residuals:
     Min       1Q   Median       3Q      Max
-1.03772 -0.41495  0.01982  0.49843  0.79635

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.11495    0.04086  -2.813   0.0227 *
binned_exp   1.24411    0.06996  17.782 1.02e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6344 on 8 degrees of freedom
```

```
Multiple R-squared:  0.9753,    Adjusted R-squared:  0.9722
F-statistic: 316.2 on 1 and 8 DF,  p-value: 1.024e-07
```

```r
ggplot(data = prop_wins15, aes(x = binned_exp,
                               y = win_prop,
                               size = totalgames)) +
  geom_point(color = "black", shape = 16) +
  geom_smooth(aes(color = "Fitted Model",
                  weight = totalgames), method = "lm", se = FALSE, size = 1.2) +
  geom_abline(data = data.frame(1),
              aes(color = "Expected Linear Model",
                  linetype = "Expected Linear Model"),
              slope = 1, intercept = 0, linetype = 2, size = 1) +
  scale_color_manual(name = "Model",
                     values = c("Fitted Model" = "gold",
                                "Expected Linear Model" = "black")) +
  labs(title = "Proportion of Home Team Wins \nfrom Home Expected Outcome",
       x = "Expected Outcome",
       y = "Proportion of \nGames Won",
       caption = "Size of points indicate more games played.
       \nModel weighs point based off of amount of games played") +
  guides(size = "none") +
  theme_minimal() +
  theme(legend.position = "right",
        plot.title = element_text(hjust = 0.5),
        plot.caption = element_text(hjust = 0.5)) +
  xlim(c(0, 1)) +
  ylim(c(0, 1))
```

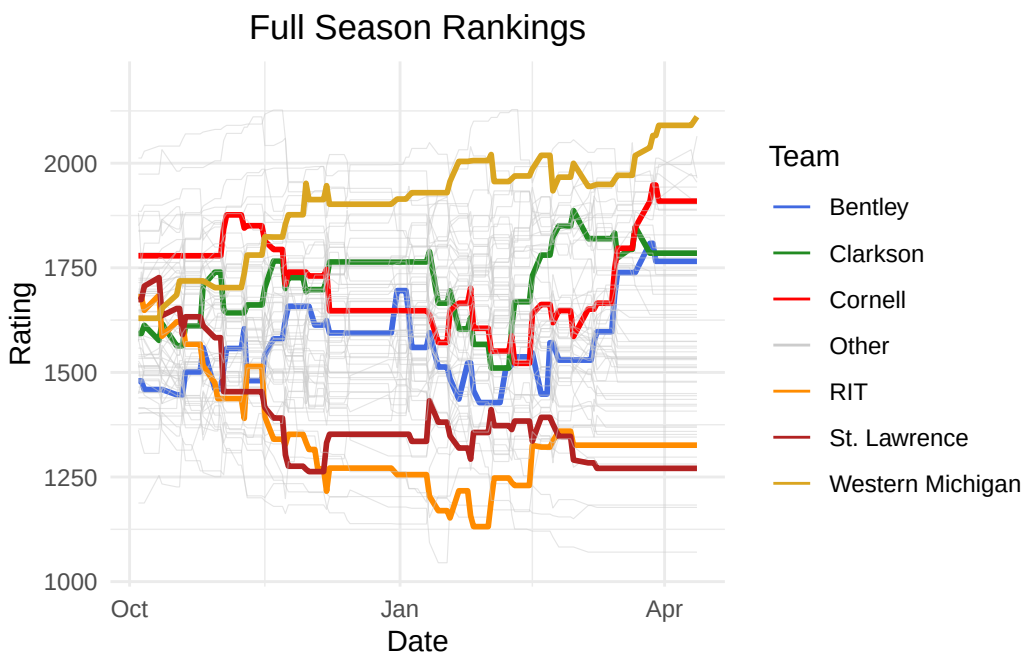


This plot bins every game played in the 2024-2025 season by every 0.1. Then uses the amount of games played in each bin to weigh each point. The y-axis is the proportion of wins for each binned proportion. We expect a slope of 1. This means that for example, at expected outcome of 0.9 we expect 90% of the games are won. In this case the fitted model is $\hat{p}_{win} = 1.24411\hat{expected}_{binned} - 0.11495$. This further shows the accuracy of the model.

Finally, we looked at the final rankings using the update_rankings_iter_gd_ha() function. This function runs the entire season and gathers ratings for each day and stores the rankings in a list. To see what the final rankings were, we pulled from the final day of the 2024-2025 post-season. To investigate the accuracy of the model, the rankings were compared to conference standings and the results of the NCAA Men's Division 1 Hockey National Championship Tournament. Final rankings are shown below with plot showing the progress of each team throughout the entire season.

```
apr15 = apr_15_ranking |> bind_rows()

apr15_lagged = apr15 |> group_by(date) |>
```

```
  summarise(last_date = last(date)) |>
  mutate(lag_date = lag(last_date)) |>
  select(-last_date)

apr15_full = left_join(apr15, apr15_lagged, join_by(date == lag_date)) |>
  select(-date) |>
  rename(date = date.y)

highlight = c("St. Lawrence", "Western Michigan",
              "Clarkson", "Bentley", "RIT", "Cornell")

highlighted_color = c(
  "St. Lawrence" = "firebrick",
  "Western Michigan" = "goldenrod",
  "Clarkson" = "forestgreen",
  "Bentley" = "royalblue",
  "RIT" = "darkorange",
  "Cornell" = "red")

apr15_color = apr15_full |> mutate(highlight =
                                     if_else(Team %in% highlight,
                                             Team, "Other"))

ggplot(data = apr15_color, aes(x = date,
                               y = rating,
                               group = Team)) +
  geom_line(aes(color = highlight,
                alpha = highlight,
                linewidth = highlight)) +
  scale_color_manual(values = c("Other" = "grey80",
                                highlighted_color),
                     name = "Team") +
  scale_alpha_manual(values = c("Other" = 0.5,
                                setNames(rep(1, length(highlighted_color)),
                                         names(highlighted_color))),
                     guide = "none") +
  scale_linewidth_manual(values = c("Other" = 0.2,
                                    setNames(rep(1, length(highlighted_color)),
                                             names(highlighted_color))),
                         guide = "none") +
  theme_minimal() +
  labs(color = "Team",
       title = "Full Season Rankings",
       x = "Date",
       y = "Rating") +
  theme(legend.position = "right",
        plot.title = element_text(hjust = 0.5))
```




```
library(knitr)
kable(apr_15_ranking[[110]] |> arrange(desc(rating))
      |> select(-date))
```

Team	rating
Western Michigan	2186.155
Boston University	1989.620
Connecticut	1970.259
Penn State	1964.211
Denver	1954.499
Michigan State	1942.901
Boston College	1915.494
Cornell	1909.227
Minnesota State	1902.014

Team	rating
Maine	1890.040
Massachusetts	1887.625
North Dakota	1828.803
Minnesota	1797.792
Quinnipiac	1796.489
Arizona State	1785.924
Clarkson	1784.968
Omaha	1773.625
Bentley	1765.578
St. Thomas	1744.077
Providence	1734.009
Michigan	1732.625
Northeastern	1730.305
Dartmouth	1706.138
Notre Dame	1697.344
Ohio State	1686.652
Long Island	1661.948
New Hampshire	1648.151
Holy Cross	1638.222
Harvard	1631.678
Bowling Green	1592.596
Colorado College	1574.595
Bemidji State	1573.813
Colgate	1562.663
Minnesota-Duluth	1558.239
Wisconsin	1549.233
Alaska	1537.669

Team	rating
Vermont	1535.678
Augustana	1533.120
St. Cloud State	1532.893
Brown	1530.410
Sacred Heart	1517.923
Merrimack	1513.541
Mass.-Lowell	1511.434
Union	1500.476
Princeton	1478.800
Army	1476.647
Lake Superior	1449.483
Ferris State	1444.334
Air Force	1435.492
Michigan Tech	1423.766
Lindenwood	1419.866
American Int'l	1407.480
Stonehill	1372.082
Niagara	1368.785
Rensselaer	1359.029
Canisius	1351.751
Alaska-Anchorage	1345.439
RIT	1326.104
Northern Michigan	1297.569
Yale	1281.107
St. Lawrence	1270.716
Robert Morris	1183.058
Miami	1177.356

Team	rating
Mercyhurst	1070.477

In these final season rankings, our Elo system ranked Western Michigan as the top team, with Boston University second. In fact, of the top 4 teams in our model, 3 of them made the Frozen Four, with Denver finishing in 5th. Western Michigan ultimately ended up winning the National Championship.

The progress chart shows some interesting patterns with some teams, especially Western Michigan. They started the season middle of the pack, but with continuous strong wins, they were able to reach 2nd place before the last quarter of the season. Other teams are highlighted based on relation to St. Lawrence, such as Cornell (ECAC champions) and Clarkson University (St. Lawrence's biggest rival). Other schools were chosen for personal reasons, Bentley was chosen as they won their conference, Atlantic Hockey America for the first time in the school's history and is also the school of my brother. RIT was also chosen due to it being my hometown. One thing to note is how almost all of these schools started in the middle of the rankings, but all finished in very different spots, showing just how much a team's strength can change season to season. This further shows the usefulness of adjusting team ratings prior to the season starting.

Finally, with the model optimized and set let's run an example, using the last game of the year, the NCAA Championship game between Western Michigan and Boston University.

```
library(knitr)
kable(schedule_full_apr15 |> slice(1153) |>
  select(c(1, 3, 4, 5, 6, 9, 10, 11, 12)))
```

date	away_team	away_score	home_team	home_score	score_diff	outcome	home_elo	away_elo
2025-04-12	Western Michigan	6	Boston University	2	-4	0	1989.62	2186.155

Western Michigan:

$$E_{WM} = \frac{1}{1 + 10^{\frac{(1990+40)-2186}{400}}}$$

$$E_{WM} = 0.71054$$

$$R'_{WM} = 2186 + 88(((0.6686 \cdot ln(4)) + 0.8048)(1 - 0.71054))$$

$$R'_{WM} = 2230.11$$

From this calculation, Western Michigan was a strong favorite to win this game, which they won handily, because of this their rating increased by 44 points to 2230.11. Now, let's see how Boston University was affected.

Boston University:

$$E_{BU} = \frac{1}{1 + 10^{\frac{2186-(1990+40)}{400}}}$$

$$E_{BU} = 0.28946$$

$$R'_{BU} = 1990 + 88(((0.6686 \cdot ln(-4)) + 0.8048)(1 - 0.28946)$$

$$R'_{BU} = 1945.89$$

We see that again the change in rating is 44 with BU decreasing by 44 to 1945.89. This again, is a small decrease since BU were underdogs, and proceeded to lose by a large amount.

0.6 Conclusion

In order to make a better ratings system in NCAA Men's Division 1 Ice Hockey, an Elo style system was created. Using goal differential, an update factor, and home ice advantage, an

Elo model was optimized using a grid search method to get the mean absolute value of game residuals down to 0.388. The model successfully ranked the four teams to make the Frozen Four in the top five and successfully ranked the national champions, Western Michigan, in the top spot for the end of 2024-2025 season rankings.

This model allows teams to be ranked quantitatively and use factors such as strength of schedule and quality of win/loss to accurately scale the effect of each win or loss. In the future, I would like to try to add more parameters in to the model to see how big of an effect things like, OT and neutral site have on expected outcome. Mostly I would like to break down score differential into goals for and goals against to see if having good defense is more important than good offense and vice versa. My biggest regret is not being able to optimize the score differential parameter myself, and if given more time, I would do this.

0.7 Code Appendix

Libraries:

```
library(elo)
library(dplyr)
library(tidyverse)
library(cowplot)
library(ggrepel)
library(lubridate)
library(rvest)
library(here)
library(forcats)
library(progressr)
library(furrr)
library(vctrs)
library(purrr)
library(knitr)
```

Scraping function:

```r
##Function to load in schedule
scrape_men <- function(season = "20232024"){
  ## URL for schedule data frame
  url_hockey <-paste("https://www.collegehockeynews.com/schedules/?season=",
                     season, sep = "")
  ## Selecting which schedule table to grab
  tab_hockey <- read_html(url_hockey) |>
    html_nodes("table")

  ## The website likes to switch which table it uses.
  ## If function doesn't work try changing which table number you select
  stats_dirty <- tab_hockey[[1]] |> html_table()

  ## Creating regex for date, and conference to make date and
  ##conference columns in dataframe
  regex_date <- "October|November|December|January|February|March|April"
  regex_conference <-
    "Atlantic Hockey|Big Ten|CCHA|ECAC|Hockey East|NCHC|Ind|Exhibition|Non-Conference"
  ## Combining regexs with original table so that the original
  ## scraped dataframe has date and conference as variables
  stats_regex <- stats_dirty |>
    mutate(date = if_else(str_detect(X1, regex_date),
                          true = X1, false = NA_character_),
           conference = if_else(str_detect(X1, regex_conference),
                                true = X1, false = NA_character_))

  ## Filling in respective dates and conferences
  stats_filled <-stats_regex |> fill(date, .direction = "down") |>
    ## Selecting date and congference so they show up as X1 and X2
    ## in the dataframe
    fill(conference, .direction = "down") |>
    select(date, conference, everything())
  ##filtering out anywhere that a conference value is undetected
  ## (Game category, not an actual game played)
  stats_filled_cleaner <- stats_filled |>
    filter(!str_detect(X1, regex_date) &
             !str_detect(X1, regex_conference))

  print(head(stats_filled_cleaner))

  ## Dataframe is now in a format that is able to be worked on.
  ## Now creating specific variables that we want to look at
  ## Selecting first 8 columns
  schedule_new <- stats_filled_cleaner |>
    select(date, conference, X1, X2, X3, X4, X5, X6) |>
    ## Taking out first two rows (no data in them).
    ##Renaming columns to match what their variable is.
```

```
    slice(-1 , -2) |>
    rename(game_type = conference,
           away_team = X1,
           away_score = X2,
           location_marker = X3,
           home_team = X4,
           home_score = X5,
           overtime = X6) |>
    ## Take out the day of the week in our date columns as we don't
    ## need to know if a game was played on a Monday per-se.
    separate(col = date, into = c("weekday", "dm", "y"),
             sep = ", ") |>
    unite("new_date", c(dm, y),
          sep = " ") |>
    select(-weekday) |>
    ##making date column into a <date> variable
    mutate(date = mdy(new_date)) |>
    ## Taking out the <chr> date variable
    select(-new_date) |>
    select(date, everything()) |>
    ## Filtering out where there is no away team since
    ## that means no game was played
    filter(away_team != "") |>
    ## Filtering out exhibition games since
    ## we aren't looking at exhibition games
    filter(game_type != "Exhibition") |>
    ## Turning scores from <chr> to <dbl> variables
    mutate(away_score = as.double(away_score)) |>
    mutate(home_score = as.double(home_score)) |>
    ## creating a variable to indicate if a game was played at a neutral site
    mutate(neutral_site = case_when(location_marker == "vs." ~ 1,
                                    location_marker == "at" ~ 0)) |>
    ## Making the neutral_site variable as <lgl>
    mutate(neutral_site = as.logical(neutral_site)) |>
    ## taking out location_marker
    select(-location_marker) |>
    ## Making a logical overtime variable. Note we are not
    ## differentiating between OT and 2OT
    mutate(overtime = case_when(overtime == "" ~ 0,
                                overtime == "ot" ~ 1,
                                overtime == "2ot" ~ 1)) |>
    mutate(overtime = as.logical(overtime)) |>
    ##Filtering out NA "overtime" values as this indicates
    ## no game played, since overtime will either be TRUE or FALSE
    filter(!is.na(overtime))|>
    ## Creating a score differential variable to indicate a win,
    ## loss, or tie for the home team. If we know the outcome
```

```
    ## for the home team, we know the outcome for the away team.
    mutate(score_diff = home_score - away_score) |>
    ## making an outcome variable for home team so ties get
    ##input as 0.5, wins get input as 1, and loss get input as 0.
    mutate(outcome =
             case_when(score_diff == 0 ~ "0.5",
                       score_diff > 0 ~ "1",
                       score_diff < 0 ~ "0")) |>
    ## turning score_diff from <chr> to <dbl>
    mutate(outcome = as.double(outcome)) |>
    ## Filtering out games where D1 team played against
    ##D3 teams as these are exhibition as well
    filter(game_type != "Non-Conference v. D3")

  ## Tidy schedule is returned
  return(schedule_new)
}

##Load in Schedule
schedule <- scrape_men("20242025")

## Load in my arbitrary initial elo ranking
X22Rankings <- read_csv(here("datasets_dataframes/22Rankings.csv"))
```

Function to do single day ratings:

```
##Function to update rankings
##rating is the variable, ratings is the df.
update_rankings <- function(season, game_date, ratings, k = 20){
  ## Filters schedule to a specific date
  elo_ratings_update <- season |> filter(date == game_date) |>
    ## Joins the Elo ratings from our rating file to the schedule file.
    ## Puts updated ratings in the schedule
    left_join(ratings, by = join_by(away_team == Team)) |>
    rename(away_elo = rating) |>
    ## Updates ratings for home team in the schedule file
    left_join(ratings, by = join_by(home_team == Team)) |>
    rename(home_elo = rating) |>
    ## Creating an away team outcome variable.
    ##Opposite of home team or same if tie.
    mutate(outcome_away = abs(outcome - 1)) |>
    ## Calculating expected outcome variable for home and away team
    mutate(exp_home = 1/(1 + 10^((away_elo - home_elo)/400))) |>
    mutate(exp_away = 1/(1 + 10^((home_elo - away_elo)/400))) |>
    ## Using expected outcome variable to generate new
```

```
    ## Elo ratings based on actual outcome and expected outcome
    mutate(elo_new_home = home_elo + k*(outcome - exp_home)) |>
    mutate(elo_new_away = away_elo + k*(outcome_away - exp_away)) |>
    rename(date_update_rankings = date)

  ranked_home <- left_join(ratings, elo_ratings_update,
                           by = join_by(Team == home_team)) |>
    relocate(elo_new_home) |>
    mutate(rating = if_else(!is.na(elo_new_home),
                            true = elo_new_home,
                            false = rating)) |>
    select(Team, rating, date_update_rankings)

  ratings_new <- left_join(ranked_home, elo_ratings_update,
                           by = join_by(Team == away_team)) |>
    relocate(elo_new_away) |>
    mutate(rating = if_else(!is.na(elo_new_away),
                            true = elo_new_away,
                            false = rating)) |>
    select(Team, rating, date_update_rankings.x) |>
    mutate(date_update_rankings.x = game_date) |>
    rename(date = date_update_rankings.x)

  return(ratings_new)
}
```

Creating a vector of dates to be iterated to automate ratings:

```
dates_vec <- unique(schedule$date)
## defining what new_rankings is going to be
new_rankings = rankings
## Creating a for loop with the update_rankings
## function to update rankings up to a specified date
for (i in dates_vec) {
  new_rankings <- update_rankings(season = schedule,
                                  game_date = i,
                                  ratings = new_rankings,
                                  k = 100)

}
```

Function to iterate through dates to automate ratings

```
update_rankings_iter <- function(season, end_date, ratings, k){

  new_rankings <- list()

  season_cut <- season |>
    ## Filter by a specified end date to deal with NA values
    ##(games that have yet to be played)
    filter(date <= ymd(end_date))
  ## Creating a vector for unique dates in a season
  dates_vector <- unique(season_cut$date)
  ## Defining our rankings within the function
  new_rankings[[1]] <- ratings
  ## Creating a for loop for the function to generate
  ##new ratings with the update_rankings function
  for (i in 1:length(dates_vector)) {
    new_rankings[[i + 1]] <- update_rankings(season = season_cut,
                                             game_date = dates_vector[i],
                                             ratings = new_rankings[[i]],
                                             k = k)
  }
  return(new_rankings)
}
```

Function to update single day ratings with all three parameters, k, score differential, home ice advantage:

```
update_rankings_gd_ha <- function(season,
                                  game_date,
                                  ratings,
                                  k = 100,
                                  home_ice = 50,
                                  d = 0.5){
  ## Filters schedule to a specific date
  elo_ratings_update <- season |> filter(date == game_date) |>
    ## Joins the Elo ratings from our rating file to the schedule file.
    ## Puts updated ratings in the schedule
    left_join(ratings, by = join_by(away_team == Team)) |>
    rename(away_elo = rating) |>
    ## Updates ratings for home team in the schedule file
    left_join(ratings, by = join_by(home_team == Team)) |>
    rename(home_elo = rating) |>
    ## Creating an away team outcome variable. Opposite of
    ## home team or same if tie.
    mutate(outcome_away = abs(outcome - 1)) |>
```

```
    ## Calculating expected outcome variable for home and away team
    mutate(exp_home = 1/(1 + 10^((away_elo - (home_elo + home_ice))/400))) |>
    mutate(exp_away = 1/(1 + 10^(((home_elo + home_ice) - away_elo)/400))) |>
    ## Using expected outcome variable to generate new Elo ratings based
    ## on actual outcome and expected outcome
    ## fivethirtyeights parameters
    ## 0.6686 * log(abs(score_diff)) + 0.8048
    mutate(score_mult = if_else(score_diff == 0,
                                true = 0.8048,
                                false =
                                  0.6686 * log(abs(score_diff)) + 0.8048)) |>
    mutate(elo_new_home =
             home_elo + k * score_mult * (outcome - exp_home)) |>
    mutate(elo_new_away =
             away_elo + k * score_mult * (outcome_away - exp_away)) |>
    rename(date_update_rankings = date)

  ranked_home_gd_ha <- left_join(ratings,
                                 elo_ratings_update,
                                 by = join_by(Team == home_team)) |>
    relocate(elo_new_home) |>
    mutate(rating = if_else(!is.na(elo_new_home),
                            true = elo_new_home,
                            false = rating)) |>
    select(Team, rating, date_update_rankings)

  ratings_new_gd_ha <- left_join(ranked_home_gd_ha,
                                 elo_ratings_update,
                                 by = join_by(Team == away_team)) |>
    relocate(elo_new_away) |>
    mutate(rating = if_else(!is.na(elo_new_away),
                            true = elo_new_away,
                            false = rating)) |>
    select(Team, rating, date_update_rankings.x) |>
    mutate(date_update_rankings.x = game_date) |>
    rename(date = date_update_rankings.x)

  return(ratings_new_gd_ha)
}
```

Function to iterate through entire season automatically:

```
rankings = X22Rankings

##Function to update rankings
##rating is the variable, ratings is the df.
update_rankings_gd_ha <- function(season,
                                  game_date,
                                  ratings,
                                  k = 100,
                                  home_ice = 50,
                                  d = 0.5){
  ## Filters schedule to a specific date
  elo_ratings_update <- season |> filter(date == game_date) |>
    ## Joins the Elo ratings from our rating file to the schedule file.
    ## Puts updated ratings in the schedule
    left_join(ratings, by = join_by(away_team == Team)) |>
    rename(away_elo = rating) |>
    ## Updates ratings for home team in the schedule file
    left_join(ratings, by = join_by(home_team == Team)) |>
    rename(home_elo = rating) |>
    ## Creating an away team outcome variable.
    ## Opposite of home team or same if tie.
    mutate(outcome_away = abs(outcome - 1)) |>
    ## Calculating expected outcome variable for home and away team
    mutate(exp_home = 1/(1 + 10^((away_elo - (home_elo + home_ice))/400))) |>
    mutate(exp_away = 1/(1 + 10^(((home_elo + home_ice) - away_elo)/400))) |>
    ## Using expected outcome variable to generate new Elo ratings
    ## based on actual outcome and expected outcome
    ## 0.6686 * log(abs(score_diff)) + 0.8048
    mutate(score_mult = if_else(score_diff == 0,
                                true = 0.8048,
                                false =
                                  0.6686 * log(abs(score_diff)) + 0.8048)) |>
    mutate(elo_new_home =
             home_elo + k * score_mult * (outcome - exp_home)) |>
    mutate(elo_new_away =
             away_elo + k * score_mult * (outcome_away - exp_away)) |>
    mutate(elo_new_home = if_else(elo_new_home < 100,
                                  true = 100,
                                  false = elo_new_home)) |>
    mutate(elo_new_away = if_else(elo_new_away < 100,
                                  true = 100,
                                  false = elo_new_away)) |>
    rename(date_update_rankings = date)

  ## Find out which is the right date, select() and relocate(),
  ## **DO THIS FIRST**: try renaming date in one of the df to make
  ## less confusing (at start).
```

```
  ranked_home_gd_ha <- left_join(ratings,
                                 elo_ratings_update,
                                 by = join_by(Team == home_team)) |>
    relocate(elo_new_home) |>
    mutate(rating = if_else(!is.na(elo_new_home),
                            true = elo_new_home,
                            false = rating)) |>
    select(Team, rating, date_update_rankings)

  ratings_new_gd_ha <- left_join(ranked_home_gd_ha, elo_ratings_update,
                                 by = join_by(Team == away_team)) |>
    relocate(elo_new_away) |>
    mutate(rating = if_else(!is.na(elo_new_away),
                            true = elo_new_away,
                            false = rating)) |>
    select(Team, rating, date_update_rankings.x) |>
    mutate(date_update_rankings.x = game_date) |>
    rename(date = date_update_rankings.x)

  return(ratings_new_gd_ha)
}
```

Creating a vector of dates to use again to automate:

```
dates_vec <- unique(schedule$date)
## defining what new_rankings is going to be
new_rankings = rankings
## Creating a for loop with the update_rankings
## function to update rankings up to a specified date
for (i in dates_vec) {
  new_rankings <- update_rankings_gd_ha(season = schedule,
                                        game_date = i,
                                        ratings = new_rankings,
                                        k = 100,
                                        home_ice = 50,
                                        d = 0.5)

}
```

Function to return a completed set of rankings:

```r
update_rankings_iter_gd_ha <- function(season,
                                       end_date,
                                       ratings,
                                       k,
                                       home_ice,
                                       d){

  new_rankings <- list()

  season_cut <- season |>
    ## Filter by a specified end date to deal
    ##with NA values (gmaes that have yet to be played)
    filter(date <= ymd(end_date))
  ## Creating a vector for unique dates in a season
  dates_vector <- unique(season_cut$date)
  ## Defining our rankings within the function
  new_rankings[[1]] <- ratings
  ## Creating a for loop for the function to
  ## generate new ratings with the update_rankings function
  for (i in 1:length(dates_vector)) {
    new_rankings[[i + 1]] <- update_rankings_gd_ha(season = season_cut,
                                                   game_date = dates_vector[i],
                                                   ratings = new_rankings[[i]],
                                                   k = k,
                                                   home_ice = home_ice,
                                                   d = d)
  }
  return(new_rankings)
}
```

Function to run ratings and return the mean residual

```r
update_rankings_residuals = function(season,
                                     end_date,
                                     ratings,
                                     k,
                                     home_ice,
                                     d){

  new_rankings = update_rankings_iter_gd_ha(season = season,
                                            end_date = end_date,
                                            ratings = ratings,
                                            k = k,
                                            home_ice = home_ice,
                                            d = d)
```

```
full_rankings = new_rankings |> bind_rows()

lagged_dates = full_rankings |> group_by(date) |>
  summarise(last_date = last(date)) |>
  mutate(lag_date = lag(last_date)) |>
  select(-last_date)

lagged_rankings <- left_join(full_rankings, lagged_dates,
                             join_by(date == lag_date)) |>
  select(-date) |>
  rename(date = date.y)

season = season |>
  mutate(home_elo = NA) |>
  mutate(away_elo = NA)

merged_season_home = left_join(season, lagged_rankings,
                               by = join_by(date == date, home_team == Team)) |>
  mutate(home_elo = rating) |>
  select(-rating)

merged_season =  left_join(merged_season_home, lagged_rankings,
                           by = join_by(date == date, away_team == Team)) |>
  mutate(away_elo = rating) |>
  select(-rating)

full_season = merged_season |>
  mutate(outcome_away = abs(outcome - 1)) |>
  ## Calculating expected outcome variable for home and away team
  mutate(exp_home = 1/(1 + 10^((away_elo - (home_elo + home_ice))/400))) |>
  mutate(exp_away = 1/(1 + 10^(((home_elo + home_ice) - away_elo)/400))) |>
  ## Using expected outcome variable to generate new Elo ratings
  ## based on actual outcome and expected outcome
  mutate(score_mult = if_else(score_diff == 0,
                              true = 0.8048,
                              false =
                                0.6686 * log(abs(score_diff)) + 0.8048)) |>
  mutate(elo_new_home =
           home_elo + k * score_mult * (outcome - exp_home)) |>
  mutate(elo_new_away =
           away_elo + k * score_mult * (outcome_away - exp_away)) |>
  mutate(elo_new_home = if_else(elo_new_home < 100,
                                true = 100,
                                false = elo_new_home)) |>
  mutate(elo_new_away = if_else(elo_new_away < 100,
                                true = 100,
```

```
                                    false = elo_new_away)) |>
    mutate(residual = outcome - exp_home) |>
    mutate(abs_residual = abs(residual))

  mean_residual = full_season |> summarise(avg = mean(abs_residual,
                                                      na.rm = TRUE))

  return(pull(mean_residual))
}
```

Additional data frames:

```
## Getting rankings from 2324:
## loading in the 2324 season
schedule2324 = scrape_men("20232024")

## running the 2023-24 season through the update_rankings_iter() function
rankings2324 = update_rankings_iter(schedule2324,
                                    "2024-04-13",
                                    X22Rankings,
                                    100)[[105]]

## adjusting the rankings for preseason rankings using
##the same method as fivethirtyeight.com

rankings2324 = rankings2324 |>
  mutate(rating = rating * 0.7 + (0.3 * 1500)) |>
  select(-date)
rankings2324 |> arrange(desc(rating))
```

Optimization process:

```
## Optimizing with regular season gamees only:
schedule_reg = schedule |> filter(game_type != "Big Ten Tournament") |>
  filter(game_type != "ECAC Tournament") |>
  filter(game_type != "CCHA Tournament") |>
  filter(game_type != "CCHA Tournament") |>
  filter(date < "2025-03-08")


## MAPPING ##

## setting up grid of potential k, d, and home ice values
```

```
plan(multisession)
handlers("progress")
options(progressr.enable = TRUE)

grid = expand.grid(k = seq(75, 105, length.out = 10),
                   home_ice = seq(35, 45, length.out = 3),
                   d = seq(40, 50, length.out = 1))

mean_residuals = with_progress(
  {future_pmap_dbl(grid,
                   \
                   (k,home_ice,d)update_rankings_residuals(
                     season = schedule_reg,
                     end_date = "2025-03-25",
                     ratings = rankings2324,
                     k = k,
                     home_ice = home_ice,
                     d = d),
                   .progress = TRUE)})

residual_df <- grid |> mutate(mean_residual = mean_residuals)
residual_df |> arrange(mean_residual)

optimal = residual_df |> filter(mean_residual == min(mean_residual))
```

Optimized ratings:

```
apr_15_ranking = update_rankings_iter_gd_ha(schedule, "2025-04-15",
                                            rankings2324,
                                            optimal$k,
                                            optimal$home_ice,
                                            0)
```

Data frame for mapping plot:

```
plan(multisession)
handlers("progress")
options(progressr.enable = TRUE)

grid_100 = expand.grid(k = seq(60, 120, length.out = 20),
                       home_ice = seq(30, 50, length.out = 20),
                       d = seq(0, 100, length.out = 1))
```

```
mean_residuals_100 = with_progress(
  {future_pmap_dbl(grid_100, \
                   (k,home_ice,d)update_rankings_residuals(season = schedule_reg,
                                                           end_date = "2025-03-25",
                                                           ratings = rankings2324,
                                                           k = k,
                                                           home_ice = home_ice,
                                                           d = d), .progress = TRUE)})

residual_100_df <- grid_100 |> mutate(mean_residual = mean_residuals_100)
```

0.8 Works Cited

College Hockey News. (2024). *Men's Division I Hockey Schedule* – 2023–2024 Season. https://www.collegehockeynews.com/schedules/

College Hockey News. (2025). *Men's Division I Hockey Schedule* – 2024–2025 Season. https://www.collegehockeynews.com/schedules/

Silver, N. (n.d.). *How our NHL predictions work*. FiveThirtyEight. https://fivethirtyeight.com/methodology/how-our-nhl-predictions-work/