Analyzing Elo Ratings For NCAA Men's Division 1 Hockey

Alex Tidd

Advisors: Matt Higham and Robin Lock

2025-05-02

Table of contents

1 Things to note

- APA Citation
- Render as docx then pdf
- 1.5 spacing or double space
- add line numbers. On rendered file.

2 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).

3 Introduction

- Goal
- Outline of write up
- Background of Elo

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.

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 "experts".

This project aims to solve the issue presented by using a ratings system used in chess to accurately rate and rank players, called Elo, and use in for 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 players new rating will either be increased or decreased. The benefit to using Elo, is that the rating system takes into consideration your strength, the opponents 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. The equations are shown below:

Expected outcome:

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

Rating update:

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

In this case, k is an "update factor" that scales how many points are added or subtracted from a team's rating after the event of a win or loss.

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

4 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.

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, every 8 teams, the assigned rating would decrease by 100. This initial ratings file was used as the initial ratings file to run with the 2023-2024 season to get accurate ratings for the 2024-2025 season.

To get these "accurate" rating 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 where scaled using the equation below:

$$rating^` = (rating\cdot0.7) + 450$$

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 intial rating in the 2024-2025 season.

5 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:

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

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

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

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

Using this model a general exploration to see what possible values of k, d, and home_ice could render a lower mean absolute residual. Based off the exploration, k is found at roughly 100, d at roughly 50, and home_ice at roughly 50. It should be noted that k was explored first with d = 0, and home_ice = 0. Next was home_ice, which was found at varying weights of k and d = 0. d was then explored with k = 100, and home_ice = 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 the initial rating + 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, home_ice = 53.33333, and d = 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 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. 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. Which whilst technically within our bounds, is highly unrealistic. A good model that predicts win probability 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 was incorporated as shown below:

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

The new update function now looks as follows:

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

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

The addition of 0.8048 is put in as the factor for score_diff = 1 as the natural logarithm of 1 is 0, which would then render the entire update portion of the function useless, running into the same issue if k = 0.

Now that goal factor is optimized, all that was left was to rerun the grid search to optimize k and home_ice. Using the grid search method, k is optimized at 88 and home_ice 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, it maintains that all teams keep ratings above zero and keep predicted outcomes in a more realistic zone.

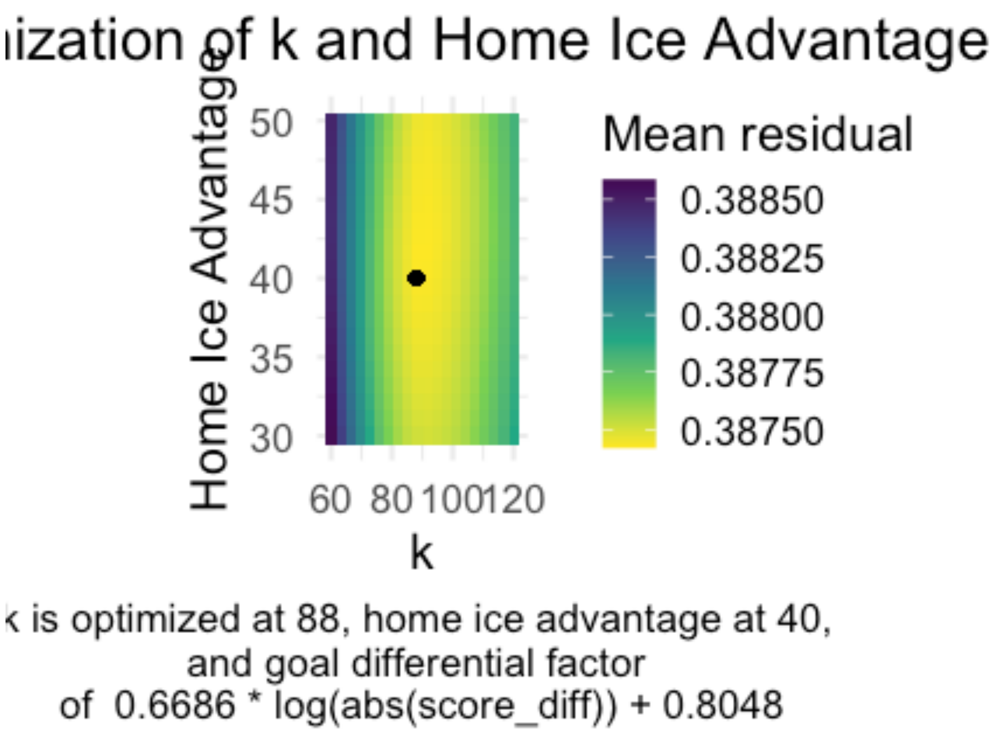
6 Results

Results from this optimization show that k = 88, home_ice = 40, and the calculation to utilize score differential = $\big(0.6686 \cdot ln(\text{scoreDiff})\big) + 0.8048)$. The plots below were used to further imply this.

Grid search for k and home_ice:

```
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",
                          ##limits = c(0.36, 0.41),
                          oob = scales::squish,
                          name = "Mean residual",
```

```
                        direction = -1) +
  labs(x = "k",
       y = "Home Ice Advantage",
       title = "Optimization of k and Home Ice Advantage",
       caption = "k is optimized at 88, home ice advantage at 40, \nand goal
differential factor \nof  0.6686 * log(abs(score_diff)) + 0.8048") +
  theme_minimal(base_size = 16) +
  theme(legend.position = "right",
        plot.title = element_text(hjust = 0.5),
        plot.caption = element_text(hjust = 0.5),
        plot.margin = margin(t = 40, b = 20, l = 60, r = 60))
```

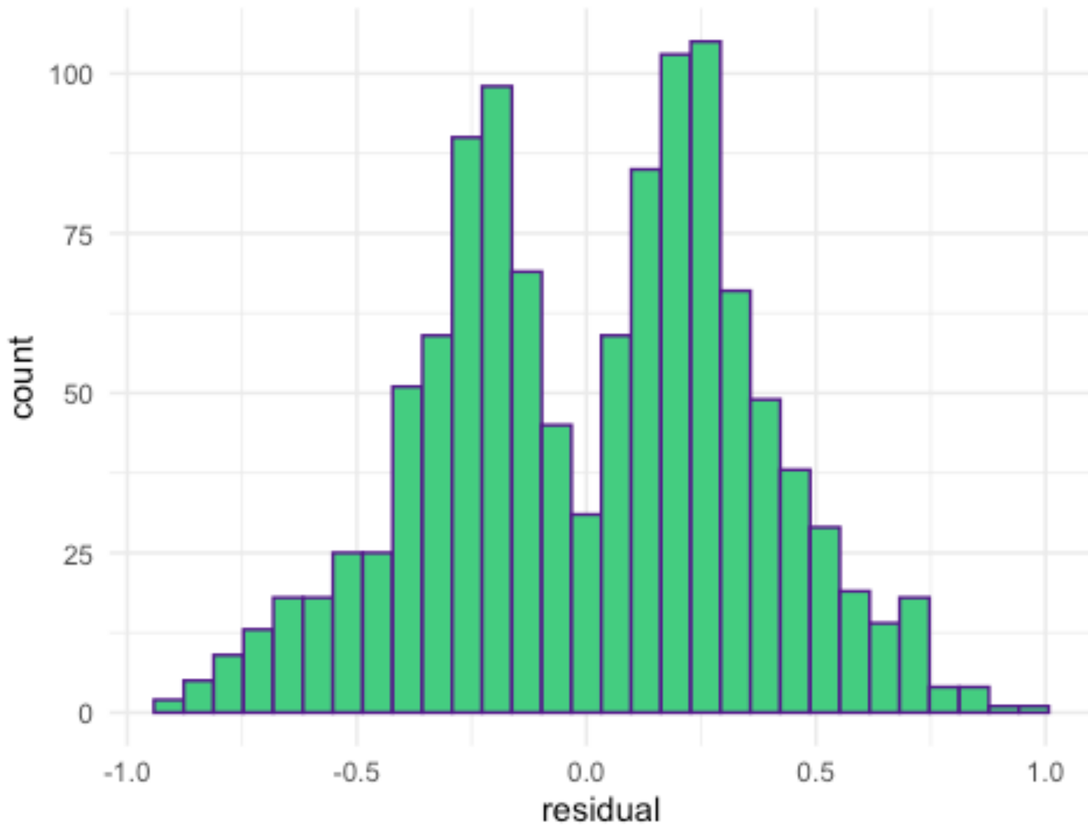


Heat map showing the grid search to optimize update factor, k, and home ice advantage, home_ice. k and home_ice are optimized to lower the mean absolute value of the difference between expected outcome and actual outcome. Goal differential factor was held at $\left(0.6686 \cdot ln(\text{scoreDiff})\right) + 0.8048$. k = 88, home_ice = 40.

Plot of residuals:

```
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") +
  labs(caption = "Positive residual past 0.5 indicates model predicting a
home loss when actual result is a home win. Negative residual beyond -0.5
indicate a predicted home wins with an observed home loss") +
  theme_minimal()
```



Negative residual beyond -0.5 indicate a predicted home wins with an observed home loss

Plot of residuals shows all the differences between expected outcome and actual outcome. Positive residual past 0.5 indicates model predicting a home loss when actual result is a home win. Negative residual beyond -0.5 indicate a predicted home wins 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.

Binned expected value graph:

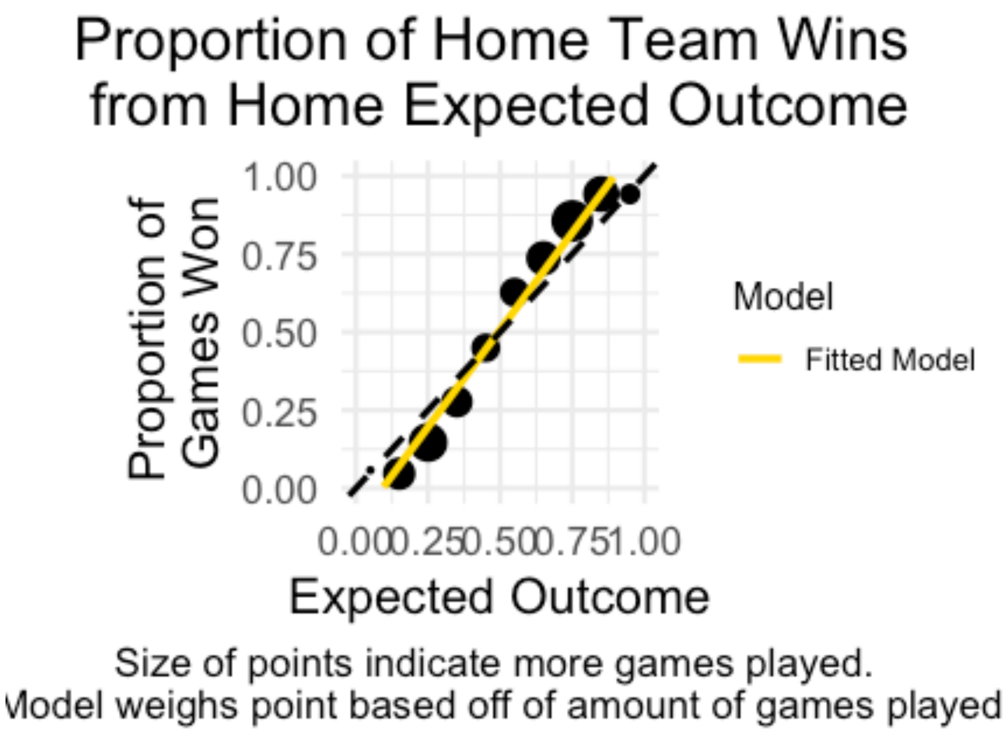
```
schedule_full_apr15 |> summarise(mean_resid = mean(abs_residual, na.rm =
TRUE))

# A tibble: 1 × 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
```

```
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(base_size = 16) +
  theme(legend.position = "right",
        legend.background = element_rect(fill = "white", color = NA),
        legend.title = element_text(size = 12),
        legend.text = element_text(size = 10),
```

```
        plot.title = element_text(hjust = 0.5),
        plot.caption = element_text(hjust = 0.5),
        plot.margin = margin(t = 40, b = 20, l = 40, r = 40)) +
  xlim(c(0, 1)) +
  ylim(c(0, 1))
```

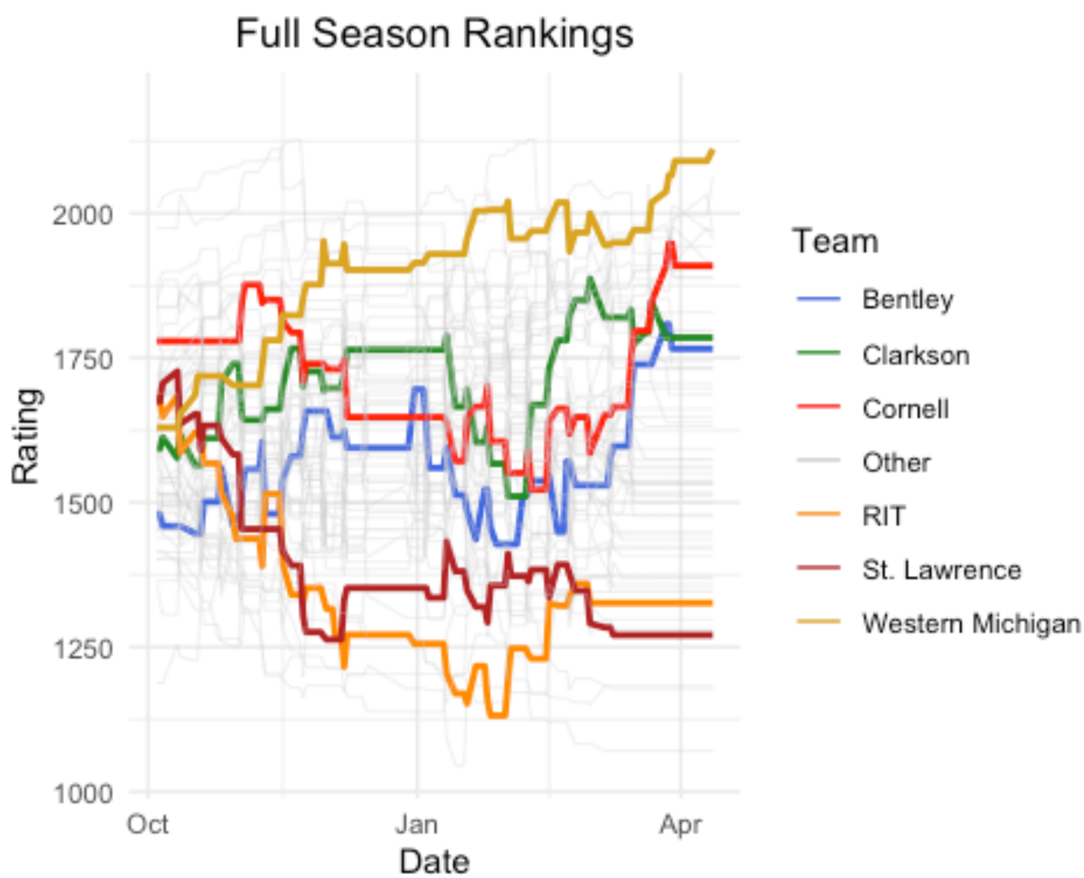


This plot bins every game played in the 2024-2025 season by every 0.1. Then uses the amount of games played in each bin to weigh each point. The y-axis is the proportion of wins for each binned proportion. We expect a slope of 1. This means that for example, at expected outcome of 0.9 we expect 90% of the games are won. In this case the slope of our fitted model is 1.24 with an intercept of -0.11. This further shows the accuracy of the optimized Elo function.

Final rankings are shown below:

```
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)) |>
```

```r
  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))
```




```
print(n = 64, apr_15_ranking[[110]] |> arrange(desc(rating)))
```

```
# A tibble: 64 × 3
   Team               rating date
   <chr>               <dbl> <date>
 1 Western Michigan    2186. 2025-04-12
 2 Boston University   1990. 2025-04-12
 3 Connecticut         1970. 2025-04-12
 4 Penn State          1964. 2025-04-12
 5 Denver              1954. 2025-04-12
 6 Michigan State      1943. 2025-04-12
 7 Boston College      1915. 2025-04-12
 8 Cornell             1909. 2025-04-12
 9 Minnesota State     1902. 2025-04-12
10 Maine               1890. 2025-04-12
11 Massachusetts       1888. 2025-04-12
12 North Dakota        1829. 2025-04-12
13 Minnesota           1798. 2025-04-12
14 Quinnipiac          1796. 2025-04-12
15 Arizona State       1786. 2025-04-12
16 Clarkson            1785. 2025-04-12
17 Omaha               1774. 2025-04-12
18 Bentley             1766. 2025-04-12
```

```
19 St. Thomas         1744. 2025-04-12
20 Providence         1734. 2025-04-12
21 Michigan           1733. 2025-04-12
22 Northeastern       1730. 2025-04-12
23 Dartmouth          1706. 2025-04-12
24 Notre Dame         1697. 2025-04-12
25 Ohio State         1687. 2025-04-12
26 Long Island        1662. 2025-04-12
27 New Hampshire      1648. 2025-04-12
28 Holy Cross         1638. 2025-04-12
29 Harvard            1632. 2025-04-12
30 Bowling Green      1593. 2025-04-12
31 Colorado College   1575. 2025-04-12
32 Bemidji State      1574. 2025-04-12
33 Colgate            1563. 2025-04-12
34 Minnesota-Duluth   1558. 2025-04-12
35 Wisconsin          1549. 2025-04-12
36 Alaska             1538. 2025-04-12
37 Vermont            1536. 2025-04-12
38 Augustana          1533. 2025-04-12
39 St. Cloud State    1533. 2025-04-12
40 Brown              1530. 2025-04-12
41 Sacred Heart       1518. 2025-04-12
42 Merrimack          1514. 2025-04-12
43 Mass.-Lowell       1511. 2025-04-12
44 Union              1500. 2025-04-12
45 Princeton          1479. 2025-04-12
46 Army               1477. 2025-04-12
47 Lake Superior      1449. 2025-04-12
48 Ferris State       1444. 2025-04-12
49 Air Force          1435. 2025-04-12
50 Michigan Tech      1424. 2025-04-12
51 Lindenwood         1420. 2025-04-12
52 American Int'l     1407. 2025-04-12
53 Stonehill          1372. 2025-04-12
54 Niagara            1369. 2025-04-12
55 Rensselaer         1359. 2025-04-12
56 Canisius           1352. 2025-04-12
57 Alaska-Anchorage   1345. 2025-04-12
58 RIT                1326. 2025-04-12
59 Northern Michigan  1298. 2025-04-12
60 Yale               1281. 2025-04-12
61 St. Lawrence       1271. 2025-04-12
62 Robert Morris      1183. 2025-04-12
63 Miami              1177. 2025-04-12
64 Mercyhurst         1070. 2025-04-12
```

1. In these final season ranking 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.

7 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, and 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 would do this.

8 Code Appendix

Libraries:

```r
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)
```

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")

```

```r
  ## 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 conferene 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)
```

```r
}

##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:

```r
##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 (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
updtae_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

```

```r
##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 = home_elo + k*(outcome - exp_home) + d *
score_diff) |>
    mutate(elo_new_home = if_else(elo_new_home < 100,
                                  true = 100,
                                  false = elo_new_home)) |>
    ##mutate(elo_new_away = away_elo + k*(outcome_away - exp_away) + d * -1 *
(score_diff)) |>
    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:

Function to run ratings and return the mean residual

```
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:

Optimization process:

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)
```

9 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/

9.1