DATA 604 - Final Project

Justin Hink

Monday, December 5, 2016

Abstract

The project explores basic simulation of the processes involved in a game of professional ice hockey, specifically the National Hockey League(NHL). The core idea for the simulation is that ice hockey can be modelled as two competing Poisson processes - One process being a function of the home team's offensive skill and the away team's defensive skill, the other being a function of the home team's defensive skill and the away team's offensive skill.

While the core of the simulation logic is contained within a single game simulator, additional layers of simulation were built upon this kernel in hopes of answering questions that are relevant to analytically minded fans and front offices alike. These additional layers are a playoff series simulator as well as playoff bracket simulator.

Key Words

Poisson Process In probability theory (and related fields), a Poisson Process is a type of random mathematical object that that consists of points randomly located on a mathematical space. It is a popular choice in simulation to model arrival times of events and entities. In this project, a Poisson process is used to model arrival times of in game events.

$$P(k \quad events) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Normal Distribution A very common (perhaps the most common) continuos probability distribution. The finite first and second statisitcal moments as well as it's close relationship with the Central Limit Theorem make it a friendly distribution to work with in theoretical excercises such as the one undertaken in this project. For the simulation constructed in this project, a team's instantaneous offensive and defensive talents are modelled as random samples from a normal distribution.

$$f(x|u, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-u)^2}{2\sigma^2}}$$

Monte Carlo Methods A broad class of computational methods that rely on repeated sampling to determine numerical results for problems that are difficult to find simple, closed-form analytic solutions. The simulation built in this project can be considered a Monte Carlo simulation.

Playoffs A term relatively specific to North American professional sports leagues. The playoffs, play-offs, postseason and/or finals of a sports league are a competition played after the regular season by the top competitors to determine the league champion or a similar accolade. In the NHL, playoffs are used to determine the Stanley Cup Champion.

True Talent vs. Luck Each team in the NHL has an underlying, ever changing talent level. This talent level is quite often different than statisically measured results (ex. wins, losses, goals for, goals against etc). A team's measured results and true talent are governed by the following high-level and simplified equation:

$$Outcomes_{Team} = Talent_{Team} + Luck$$

In this simulation, all inputs to the simulator are to be true talent estimates of a given team's abilities.

Literature Review

The idea of modelling hockey as two competing Poisson processes is not an original concept (Thomas, 2013). Retroactive looks at past performance have shown that on ice action can be captured fairly accurately with this concept. However, no published paper has documented the extension of this basic concept to predicting future outcomes.

Key to the simulator in this project are a number of historical team level effects. Though not published in established statistical journals, there have been a number of looks at these effects by the hockey analytics community. These include informal studies done on home ice advantage (Yue, 2013), back to back game effects (Geek, 2009) and the effects of varying amounts of rest between games (Yost, 2014). Home ice and back to back effects have been implemented into the project's simulator while the code has been constructed in a way such that other effects (such as rest) can be added with ease in the future.

There is at least one hockey simulation engine online (-, 2016). It, however, does not seem to be a serious academic effort as it's methodology has not been published. In addition to this, the results generated from the simulator do not pass the sniff test. Unreasonable outcomes such as 60 shots in a game (nearly unprcedented) can be generated with just a small number of runs and tweaks of input teams. It must however be said that the simulation time is very impressive and the interface and results generated are very user friendly. In other words, it's a pleasant to use "toy" simulator though I would certainly not use it as part of any serious decision making or use it as the basis of any wagers.

Methodology

Technology Used The code for the project was developed entirely in Python. The well established libraries for data wrangling (Pandas, numpy) and sampling from statisical distributions made this a logical choice for coding up this sort of prototype.

There was no need for persistent storage other than saving simulation results. Simple CSV files were used for this purpose.

Game Simulator Core to the entire project is the individual NHL game simulator. As inputs it takes true talent estimates for offensive and defensive skill of the home and away teams, which (if any) of the teams are playing on back to back nights and whether the game is a playoff game. These inputs are combined with pre-established home ice advantage effects (+4% to the home team) (Yue, 2013) and back to back effects (Geek, 2009) to determine a winner.

Other than these contextual effects, the simulation is modelled on the previously explored idea (Thomas, 2013) that a game of ice hockey can be modelled as two competing Poisson processes - One process being a function of the home team's offensive skill and the away team's defensive skill, the other being a function of the home team's defensive skill and the away team's offensive skill.

The first step to simulating this, is to generate arrival times of on ice events that can potentially lead to goals scored by either team. The following code shows how the project's simulation accomplishes this.

```
def get_game_event_times(self, lam):
    events = np.random.poisson(lam, 20)
    df = pd.DataFrame({'iat':events})
    df['at'] = df['iat'].cumsum()
    filtered  = [ num for num in list(df['at']) if num < 60 ]
    return filtered

lam = (home_gf_neut + home_ga_neut + away_gf_neut + away_ga_neut) / 4.0 * LAMBDA_GOAL_K
events = self.get_game_event_times(lam)
```

The number of events generated are proportional to the goal generating and goal suppression talents of each of the teams involved. This should be intuitive as we expect that if two teams who are known to be high scoring (high offensive talent) play against each other we would have a higher number of goals than a matchup of two more defensive minded teams.

With a collection of potential goal events created, the next step is to determine what team (if any) scored a goal for each of the events. This is where we model the two conflicts between each of the teams. To do this, 4 samples are drawn from 4 different normal distributions: 1 centered around home team offensive true talent, 1 centered around home team defensive true talent, 1 centered around away team offensive true talent and 1 centered around away team defensive true talent.

These 4 samples are then scaled by our contextual effects (home ice advantage and back to back effects) and evaluated together to determine which team scored a goal for that event. The probability that one team is chosen as the goal scorer instead of the other is directly related to their talent levels and the contextual effects.

The following code listing shows this process of generating the 4 relevant samples and the conflict evaluation. Again, there are 3 potential outcomes for each event: Home team goal, Away team goal or No goal.

```
home_goals = 0
away_goals = 0

home_running_goals = []
away_running_goals = []
for event in events:
    home_gf = np.random.normal(home_gf_neut, self.GOAL_SD *2.25, 1) *
                (1 + home_sched_boost / 2.0 + self.HOME_ICE_AD / 2.0 )
    home_ga = np.random.normal(home_ga_neut, self.GOAL_SD *2.25, 1) /
                (1 + home_sched_boost / 2.0 + self.HOME_ICE_AD / 2.0 )

    away_gf = np.random.normal(away_gf_neut, self.GOAL_SD *2.25, 1) /
                (1 + home_sched_boost / 2.0 + self.HOME_ICE_AD / 2.0 )
    away_ga = np.random.normal(away_ga_neut, self.GOAL_SD *2.25, 1) *
                (1 + home_sched_boost / 2.0 + self.HOME_ICE_AD / 2.0 )

    # simulate conflicts
    if home_gf > away_gf and home_ga < away_gf:
        home_goals = home_goals + 1
    elif home_gf < away_gf and home_ga > away_gf:
        away_goals = away_goals +1

    home_running_goals.append(home_goals)
    away_running_goals.append(away_goals)
```

With all game events simulated, the next step is to determine which team won the game. This is trivial (whoever has more goals wins). In the event of a tie, the way things are settled differs whether it is a playoff game or a regular season game. In the NHL, regular season games that are not decided in regulation time go to a 5 minute period of overtime with 3 skaters and a goalie from each team on the ice simultaneously (as opposed to the 5 skaters plus goalie in regulation play). If the game is still not decided after these 5 additional mintues, the game is decided by a shootout.

For NHL playoff games, the "3 on 3 + shootout" mechanism is not used. Instead, regular play continues until one team scores. Each of these processes are simulated in the code below. Note that 3 on 3 hockey is a new phenomenon (as of 2015) and as such, the analyst community does not really understand it at this point. As such, the regular season overtime model below is very much subject to change.

Also worth noting that the shootout is modelled as a simple coin flip (50-50 proposition). While probably close to accurate, this could be improved as well but it would necessitate the incorporation of individual goaltender skill. This is not something that has been incorporated at this time.

```
home_wins = 0
away_wins = 0
home_OTLs = 0
away_OTLs = 0
home_SOLs = 0
away_SOLs = 0

if home_goals != away_goals:
    if home_goals > away_goals:
        home_goals = home_goals + 1
        home_wins = 1
    else:
        away_goals = away_goals + 1
        away_wins = 1
else:
    if not is_playoffs:
        # game goes to OT
        home_gf_ot = np.random.normal(home_gf_neut, self.GOAL_SD, 1)
        * (1 + home_sched_boost + self.HOME_ICE_AD)
        home_ga_ot = np.random.normal(home_ga_neut, self.GOAL_SD, 1)
        / (1 + home_sched_boost + self.HOME_ICE_AD)

        away_gf_ot = np.random.normal(away_gf_neut, self.GOAL_SD, 1)
        / (1 + home_sched_boost + self.HOME_ICE_AD)
        away_ga_ot = np.random.normal(away_ga_neut, self.GOAL_SD, 1)
        * (1 + home_sched_boost + self.HOME_ICE_AD)

        home_goals_ot = (home_gf_ot + away_ga_ot) / 2.0
        away_goals_ot = (home_ga_ot + away_gf_ot) / 2.0

        ot_diff = home_goals_ot - away_goals_ot

        if ot_diff > 1:
            home_wins = 1
            away_OTLs = 1
        elif ot_diff < -1.0:
            away_wins = 1
            home_OTLs = 1
        else:
            # game goes to a shootout
            flip = np.random.uniform(0,1,1)

            if flip[0] > 0.5:
                home_wins = 1
                away_SOLs = 1
            else:
                away_wins = 1
                home_SOLs = 1
    else:
        # playoffs have no ties or shootouts
```

```
            home_gf_ot = np.random.normal(home_gf_neut, self.GOAL_SD, 1)
            * (1 + home_sched_boost + self.HOME_ICE_AD)
            home_ga_ot = np.random.normal(home_ga_neut, self.GOAL_SD, 1)
            / (1 + home_sched_boost + self.HOME_ICE_AD)

            away_gf_ot = np.random.normal(away_gf_neut, self.GOAL_SD, 1)
            / (1 + home_sched_boost + self.HOME_ICE_AD)
            away_ga_ot = np.random.normal(away_ga_neut, self.GOAL_SD, 1)
            * (1 + home_sched_boost + self.HOME_ICE_AD)

            home_goals_ot = (home_gf_ot + away_ga_ot) / 2.0
            away_goals_ot = (home_ga_ot + away_gf_ot) / 2.0

            if(home_goals_ot > away_goals_ot):
                home_goals = home_goals + 1
                home_wins = 1
            else:
                away_goals = away_goals + 1
                away_wins = 1
```

Playoff Series Simulator On top of the core game simulator, 2 additional layers of simulation logic were built. The first of these is a playoff series simulator. In the NHL, a playoff series is a set of games between two teams. The first team to win 4 games in the series is declared the series winner (and advances in the overall playoffs). The higher seeded team (the team with the better record in the regular season) has, on average, more home games in a given series. The schedule of who plays at home is given by the following schedule:

Game	Home Team
1	Higher Seed
2	Higher Seed
3	Lower Seed
4	Lower Seed
5	Higher Seed
6	Lower Seed
7	Higher Seed

Table 1: Series Home Ice Advantage Schedule

Note that games 5,6 and 7 are only played if a winner of the series has not already been determined.

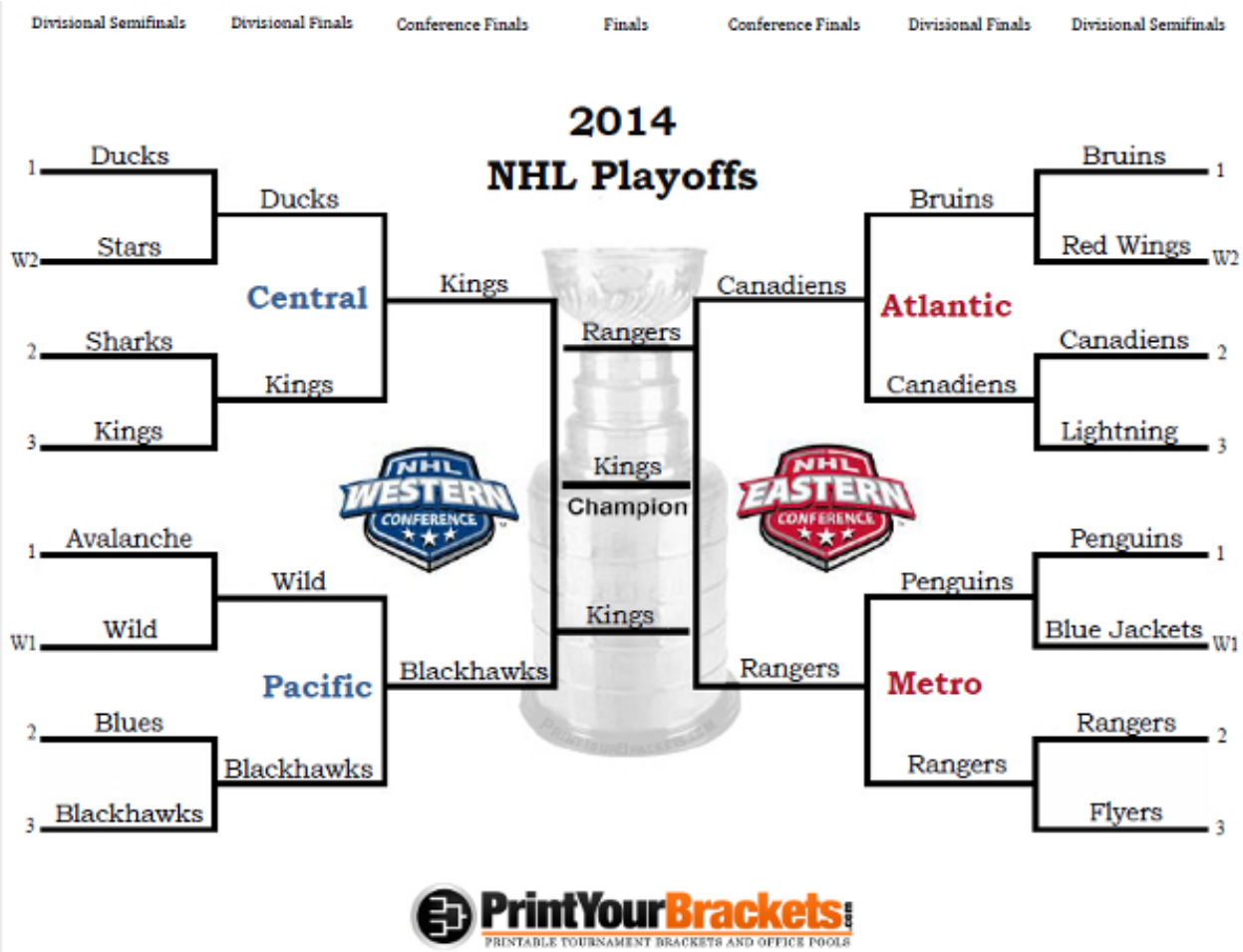
The following code listing is for the simulator's SeriesSimulator class. It essentially schedules games as per the above table and delegates individual game simulation to the code described previously in the Game Simulator section of the report.

```
class SeriesSimulator(object):
    """An NHL Playoff Series Simulator"""

    def __init__(self, game_simulator):
        self.game_simulator = game_simulator
```

```
    def simulate_series(self, high_seed, low_seed):
        high_seed_wins = 0
        low_seed_wins = 0

        # games 1 & 2 (high seed at home)
        for i in range(0,2):
            home_wins, away_wins = self.simulate_series_game(high_seed, low_seed)
            high_seed_wins += home_wins
            low_seed_wins += away_wins

        # game 3 and 4 (low seed at home)
        for i in range(0,2):
            home_wins, away_wins = self.simulate_series_game(low_seed, high_seed)
            high_seed_wins += away_wins
            low_seed_wins += home_wins

        # game 5 (high seed at home)
        if high_seed_wins < 4 and low_seed_wins < 4:
            home_wins, away_wins = self.simulate_series_game(high_seed, low_seed)
            high_seed_wins += home_wins
            low_seed_wins += away_wins

        # game 6 (low seed at home)
        if high_seed_wins < 4 and low_seed_wins < 4:
            home_wins, away_wins = self.simulate_series_game(low_seed, high_seed)
            high_seed_wins += away_wins
            low_seed_wins += home_wins

        # game 7 (high seed at home)
        if high_seed_wins < 4 and low_seed_wins < 4:
            home_wins, away_wins = self.simulate_series_game(high_seed, low_seed)
            high_seed_wins += home_wins
            low_seed_wins += away_wins

        if high_seed_wins > low_seed_wins:
            return high_seed
        else:
            return low_seed

    def simulate_series_game(self, home_team, away_team):
        home_goals,
        away_goals,
        home_wins,
        away_wins,
        home_OTLs,
        away_OTLs,
        home_SOLs,
        away_SOLs,
        event_chart = self.game_simulator
                                .simulate_game_poisson(home_team.gf,
                                home_team.ga,
                                away_team.gf,
                                away_team.ga,
```

```
                                False,
                                False,
                                True)
        return home_wins, away_wins
```

Playoff Simulator The final layer of simulation created for this project is an overall playoff simulation engine/class. This class simulates a playoff bracket based on a input set of playoff seeds. A playoff bracket is simply a set of playoff series simulations occuring at a set schedule. The following is an example of what the 2014 NHL season's playoff bracket looked like (www.printyourbrackets.com, 2014).



The project's playoff simulator simulates this bracket/tournament process by leveraging the series simulator described previously and delegates calls to it for individual series simulation. The listing below shows this relatively straight forward piece of code.

```
class PlayoffSimulator(object):
    game_simulator = GameSimulator()
    series_simulator = SeriesSimulator(game_simulator)

    # simulate conference
    def sim_conference(self, seed1, seed2, seed3, seed4, seed5, seed6, seed7, seed8):
        # round 1
        r1_w1 = self.series_simulator.simulate_series(seed1, seed8)
```

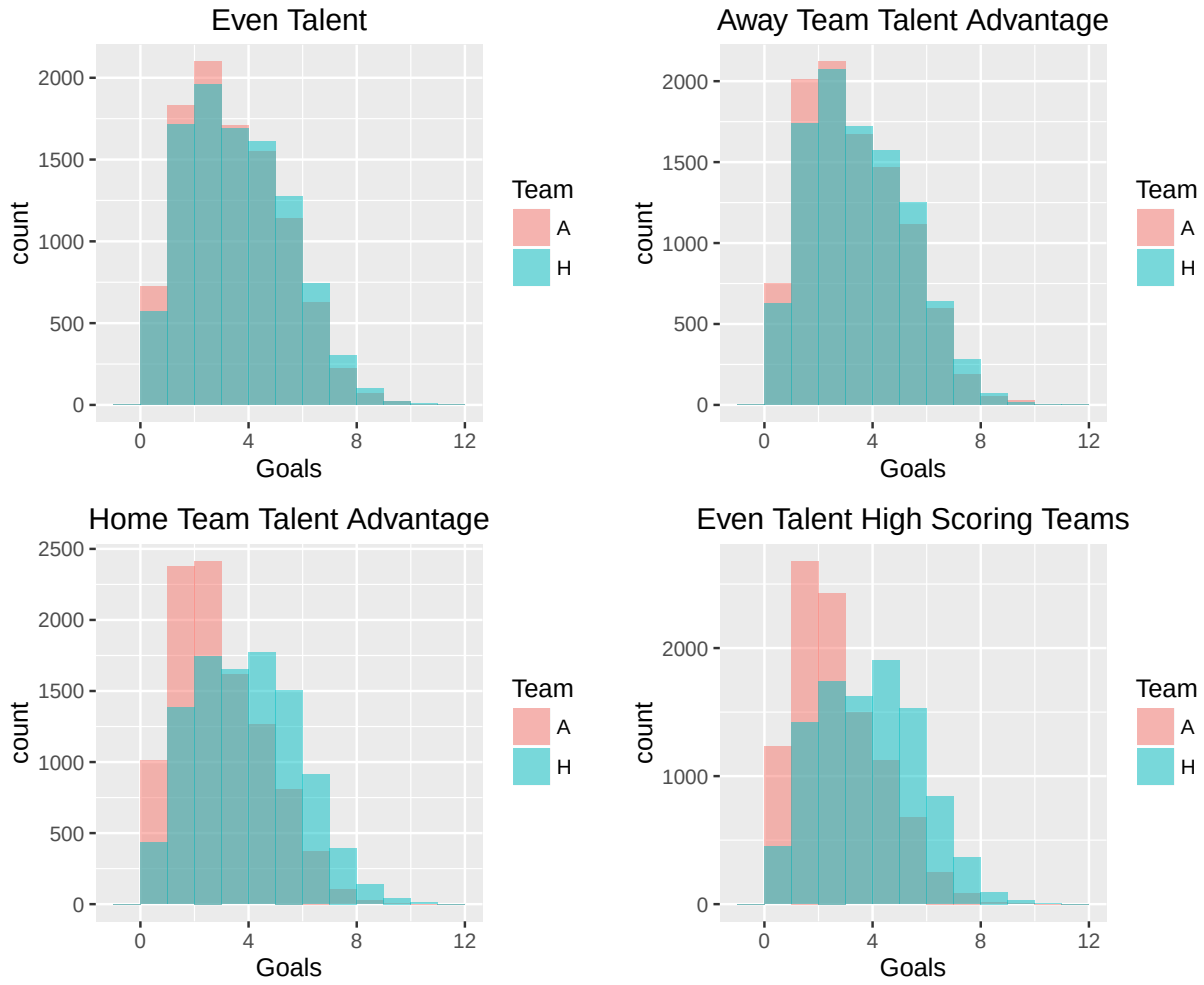
```
        r1_w2 = self.series_simulator.simulate_series(seed2, seed7)
        r1_w3 = self.series_simulator.simulate_series(seed3, seed6)
        r1_w4 = self.series_simulator.simulate_series(seed4, seed5)

        # round 2
        r2_w1 = self.series_simulator.simulate_series(r1_w1, r1_w4)
        r2_w2 = self.series_simulator.simulate_series(r1_w2, r1_w3)

        # round 3 (conference finals)
        conf_champ = self.series_simulator.simulate_series(r2_w1, r2_w2)

        return conf_champ

    def sim_playoffs(self, east_teams, west_teams):
        # simulate conferences
        east_champ = self.sim_conference(east_teams[0],
                                         east_teams[1],
                                         east_teams[2],
                                         east_teams[3],
                                         east_teams[4],
                                         east_teams[5],
                                         east_teams[6],
                                         east_teams[7])
        west_champ = self.sim_conference(west_teams[0],
                                         west_teams[1],
                                         west_teams[2],
                                         west_teams[3],
                                         west_teams[4],
                                         west_teams[5],
                                         west_teams[6],
                                         west_teams[7])

        # simulate stanley cup final

        # for simulation purposes, determine home ice with a coin flip
        flip = np.random.uniform(0,1,1)

        if(flip[0] < 0.5):
            champ = self.series_simulator.simulate_series(east_champ, west_champ)
        else:
            champ = self.series_simulator.simulate_series(west_champ, east_champ)

        return champ
```

Results

Game Simulator

The following plots represent a typical 10000 game run for the core game simulator in a number of contexts (different sets of inputs).

Goals



All 8 of these goal distributions align with intuition. The top left represents a simulation of identically talented teams with only home ice advantage in play. As we would expect, this home ice advantage leads to the home team scoring, on average, more goals than their opposition.

The top right shows a scenario in which we bump the talent level of the away team. We can see that the effects of home ice advantage are partially mitigated and that the number of expected goals is closer than in the first run.

The bottom left represents a related scenario in which we setup the simulation such that the home team has greater true talent than their opponents. The gap in expected goals scored grows, ie-the home team is expected to score more, conceed less and ultimately win more.

The bottom right plot is again an even talent scenario. However this time, both teams are assigned to be higher scoring in nature. The effect here is interesting, non-intuitive and, possibly not correct. However, what we are seeing makes sense given the way the simulation is constructed. In this plot, home ice advantage seems to be amplified. Why might that be the case if the teams are of even talent? The reason for this effect is that the increase in collective scoring has amplified the number of potential goal events in an average game as our arrival times are proportional to offensive skill. With more events, there will be more confrontations in our model. With the number of confrontations increased, there is more opportunity for home ice advantage to be applied and have a larger overall effect.

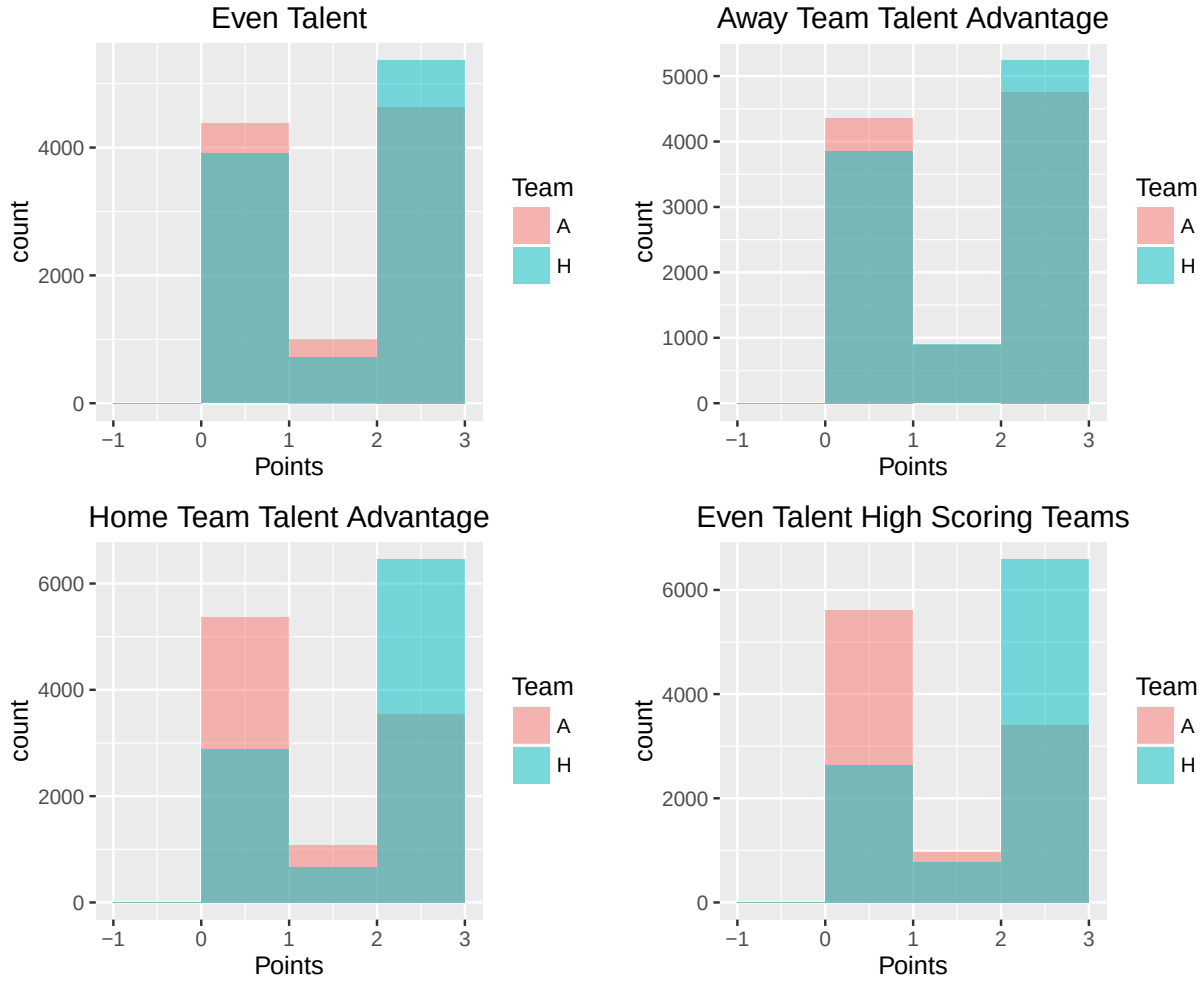
This is not a totally foreign or crazy concept. In other games in which confrontation or conflict is a core

component, it is a well known strategy that you should seek to increase the number of confrontations if you have an advantage or edge. For example, in poker, good players will try to play more hands and seek more confrontations with someone they feel has an inferior level of skill. There coulld be a valid analogue here but it will take more research and deeper looks at historical data to see if this effect that has come out of our simulation is a new insight or simply an artifact of flawed design.

Points

In the NHL's regular season, 2 points are awarded to a team for winning a game in regulation, 2 for winning a game in overtime or a shootout, 1 for an overtime loss, 1 for a shootout loss and 0 for a regulation loss.

The following graphs show the points awarded in the same runs of the simulator that we examined in the previous section.



There are no new surprising effects discovered with these plots. Home ice advantage and talent levels adjust the expected point distributions in a fashion that is to be expected.

Playoff Simulator

As discussed, additional simulation layers were built upon the core game simulator. The playoff simulator's primary intention is to take an input set of teams in a particular tournament seeding and determine the probabilities of each team winning the Stanley Cup.

The following table represents what could be considered a relatively "normal" set of playoff seeds. The top seeds in each conference (East and West) are the best in terms of offensive and defensive talent. Talent levels degrade as the seeding order increases.

Team	Off True Talent	Def True Talent
East1	3.5	2.7
East2	3.4	2.8
East3	3.4	3.0
East4	3.3	3.0
East5	3.3	3.1
East6	3.2	3.0
East7	3.2	3.1
East8	3.1	3.1
West1	3.5	2.7
West2	3.4	2.8
West3	3.4	3.0
West4	3.3	3.0
West5	3.3	3.1
West6	3.2	3.0
West7	3.2	3.1
West8	3.1	3.1

Table 2: Playoff Simulation Seeding and Talent (Goals Per Game)

With this set of seeds (think of this as a playoff bracket with the first round filled in), the playoff simulator was run for 100,000 iterations. The following table shows the resulting championship probabilities for each of our playoff participants.

team	champ_prob
East1	0.201
East2	0.106
East3	0.053
East4	0.048
East5	0.030
East6	0.025
East7	0.020
East8	0.017
West1	0.202
West2	0.106
West3	0.052

team	champ_prob
West4	0.047
West5	0.030
West6	0.025
West7	0.020
West8	0.017

Table 3: Simulated Championship Probabilities

The results here look completely reasonable. Better teams win more, low seeds have a non-zero chance of winning and the conference's are symmetric (necesarry because of our symmetric talent inputs).

Note on performance: The playoff simulator is very slow. To run 100,000 iterations took almost 10 hours on top of the line modern hardware. Clearly the code is in need of optimization as I don't see it being that calculation intensive.

Future Work

Shot Events as Basis of Simulation As discussed, in it's current state, the simulator is focused on goal events only. It is a well known maxim in the hockey analytics community that a team's true talent is more directly measured by their ability to generate (offense) and prevent (defense) shot attempts and that shooting percentage is very noisy. It follows then that the simulator could be enhanced by changing the basis of simulation to shot generating events.

The basic duelling Poisson process methodology would stay intact with only parameter tweaking in the Poisson and Gaussian distributions to accurately depict the increased number of events. In addition to creating generally more accurate and fine grained simulation, this would allow for goalie skill to be incorporated as it's own independent factor. In the simulator's current state, a team's goalie skill is simply a part of a team's goal prevention skill. With this modification, a shot event would be generated in one direction and then a new probabilistic coinflip incorporating goalie and opposing shooter skill would combine to determine whether the attempt resulted in a goal, a missed shot or a save.

Player Level Events and Statistics Related to the above enhancement, an appropriate future step would be to assign statistics to individual players involved in the game. For example, if Team A scores a goal, assign a player from Team A who scored the goal as well as the players who assisted on it. Similarly the opposing goaltender would be debited with a goal against and all players on the ice would have their plus-minus statistics adjusted accordingly.

The assignment of these events appears non trivial at first and will require significant research before this can be done in a statistically valid fashion.

Full Season Simulation It would be relatively simple to build an additional layer to the simulator that allows us to simulate an entire NHL season. This, ideally, would have the full NHL schedule for a given year loaded into it and simulate outcomes for all games. Done a large number of times, this would allow us to answer questions such as: "What is the probability the Rangers will make the playoffs this year?" or "What is the probability the Islanders finish last and are awarded the first pick in next year's entry draft?".

Player Projections With a full season simulator in hand, building aggregated statistical projections for a given player's season would be a useful and popular feature. For example, "Sidney Crosby is expected to score 37 goals in the upcoming season with a standard error of 7 goals". This would be a useful scaling of the simulation outputs for front offices and fantasy hockey players alike.

Increased Performance For whatever reason (probably a Java/C++ guy trying to write Python), the simulator in it's current state runs much slower than one should expect. To run 10,000 playoff simulations takes almost 10 minutes. Given the relative simplicity of the calculations, this seems to be orders of magnitude away from expectation.

Python is a notorious slow language but it is not likely to blame for the majority of this. Some detailed profiling, debugging and optimization would need to occur on the current code base before any future extensions can be built.

Conclusion

In conclusion, a basic team level simulation engine for the National Hockey League was created. The core mathematical idea for the simulation was that ice hockey can be modelled as two competing Poisson processes that correlate with home and away offensive and defensive skills.

The game simulator was able to provide answers to common questions such as "What are the chances Team A will beat Team B" tonight. Important game effects such as home ice advantage and back to back rest effects were incorporated into the simulation engine with a computational framework being established to incorporate further effects in the future (ex Number of cummulative travel miles, altitude).

On top of the core game simulator, additional layers of logic were built which fascilitated the answering of playoff related questions. The probability of each team winning the Stanley Cup in a syntheticall derived year was simulated and shown to be reasonable.

While simple and unfortunately smaller in scope than originally hoped for, the project has established itself as a building block for future work. With features flushed out and performance increased it seems quite possible the work can be expanded into a modest hobbyist style website.

References

-, -. Whatifsports. Dec. 2016. URL: https://www.whatifsports.com/nhl/.

Geek, J. Whatifsports. Dec. 2009. URL: http://hockeynumbers.blogspot.com/2009/02/back-to-back-game-statistics.html.

Thomas, A. Competing process hazard function models for player ratings in ice hockey. Jul. 2013. URL: https://arxiv.org/abs/1208.0799.

www.printyourbrackets.com. 2014-nhl-playoff-bracket-results.jpg. 2014. URL: https://www.printyourbrackets.com/nhl-playoff-results/2014-nhl-playoff-bracket-results.jpg.

Yost, T. Rest makes a major difference in NHL performance. Oct. 2014. URL: http://www.tsn.ca/yost-rest-makes-a-major-difference-in-nhl-performance-1.120073.

Yue, W. Home Ice Advantage. Dec. 2013. URL: http://hockeymetrics.net/home-ice-advantage/.

Appendix A: Code Repo

https://github.com/jhink7/nhl_simulator