TrueSkill Through Time: reliable initial skill estimates and historical comparability with **Julia**, **Python**, and **R**

Gustavo Landfried
Universidad de Buenos Aires

Esteban Mocskos
Universidad de Buenos Aires

Abstract

Knowing how individual abilities change is essential in a wide range of activities. The most widely used skill estimators in industry and academia (such as Elo and TrueSkill) propagate information in only one direction, from the past to the future, preventing them from obtaining reliable initial estimates and ensuring comparability between estimates distant in time and space. In contrast, the model TrueSkill Through Time (TTT) propagates all historical information throughout a single causal network, providing estimates with low uncertainty at any given time, enabling reliable initial skill estimates, and ensuring historical comparability. Although the TTT model was published more than a decade ago, it was not available until now in the programming languages with the largest communities. Here we offer the first software for Julia, Python, and R, accompanied by a detailed overview for the general public and an in-depth scientific explanation. After illustrating its basic mode of use, we show how to estimate the learning curves of historical players of the Association of Tennis Professionals. Analytical approximation methods and message-passing algorithms allow inference to be solved efficiently using any low-end computer, even in causal networks with millions of nodes and irregular structures.

Keywords: Learning, skill, Bayesian inference, gaming, education, sports, Julia, Python, R.

1. Introduction

Knowing how individual skills change over time is essential in the educational system and the labor market. Since skills are hidden variables, the best we can do is estimate them based on their direct observable consequences: the outcome of problem-solving and competitions. However, estimating learning curves is a sensitive issue, especially when they are used to make decisions that may impact individuals. Considering only the frequency of positive results as an indicator of the individual's ability could lead to wrong approximations, mainly because the outcome also depends on the difficulty of the challenge. For this reason, all widely used skill estimators use pairwise comparisons. With the first generative models proposed almost a century ago by Thurstone (1927) and Zermelo (2013), it is assumed that the observed result probability r depends on the performance p of an agent i and their opponent j, expressed as $P(\, r \,|\, p_i,\, p_j \,)$. The field continued to progress with the work of Bradley and Terry (1952) and Mosteller (1951a,b,c), leading to a breakthrough that took place when Elo (2008) developed a methodology for the US Chess Federation (USCF), used by the International Chess Federation (FIDE) nowadays.

More recently, these models were extended and solved through the Bayesian approach to probability Glickman (2001); Herbrich *et al.* (2006). Using Bayes's theorem we can quantify the uncertainty of the skill hypotheses using the information provided by the observed result and the specific model,

$$\underbrace{p(\overbrace{\text{Skill}_i}^{\text{Hidden}} \mid \overbrace{\text{Result}}^{\text{Observed}}, \text{Model})}_{\text{Posterior}} = \frac{\overbrace{P(\text{Result} \mid \text{Skill}_i, \text{Model})}^{\text{Likelihood}}\overbrace{p(\text{Skill}_i)}^{\text{Prior}}}{\underbrace{P(\text{Result} \mid \text{Model})}_{\text{Evidence or prior prediction}}} \tag{1}$$

where the only free variable is the skill hypothesis of agent i. The prior quantifies the uncertainty about the skill before seeing the result, and the posterior quantifies the uncertainty after seeing the result. The likelihood and the evidence can be interpreted as predictions of the observed results. Because the evidence is the same for all hypotheses, the only factor that updates our beliefs is the likelihood.

1.1. TrueSkill

In this paper, we focus on a basic causal model in which skills generate the observable result of a model (Figure 1). The agents exhibit different performances at each event, varying around their actual skill, following a distribution $\mathcal{N}(p \mid s, \beta^2)$. The model assumes that the agent with the highest performance wins, $r = (p_i > p_j)$. The parameter β^2, being the same for all

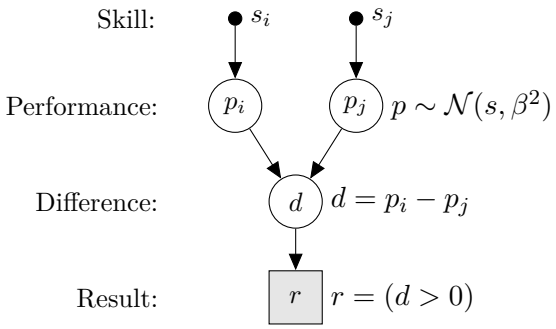


Figure 1: Generative model in which skills cause the observable results mediated by the difference of hidden performances of both random variables around their unknown true skill ($d = p_i - p_j$). The one with the highest performance wins, $r = (d > 0)$. Observable variables are painted gray, hidden in white, and constants are shown as black dots.

agents, acts as the scale of the estimates: skills at a distance of one β mean a 76 % probability of winning, independent of the absolute value of the estimates.

For example, we consider a winning case ($p_i > p_j$) using a Gaussian prior (i.e., $\mathcal{N}(s \mid \mu, \sigma^2)$) for each skill. Our prior belief about the difference in performances, $d = p_i - p_j$, is expressed as a Gaussian distribution centered on the difference in the prior estimates ($\mu_i - \mu_j$), with a variance that incorporates the uncertainty of both estimates (σ_i and σ_j) and the variance of both performances (β), $\mathcal{N}(d \mid \mu_i - \mu_j,\ 2\beta^2 + \sigma_i^2 + \sigma_j^2)$. As we observed that the agent i won, we know from the causal model that the hidden difference in performance was positive. Therefore, the prior prediction of the observed result, or evidence, is the cumulative density (Φ) of all positive values of performance difference as expressed in Equation 2. During the

rest of this work, the role of the model will be left implicit.

$$\overbrace{P(r)}^{\text{Evidence}} = 1 - \Phi(0 \,|\, \overbrace{\mu_i - \mu_j}^{\text{Expected difference}}, \overbrace{2\beta^2 + \sigma_i^2 + \sigma_j^2}^{\text{Total uncertainty}}) \tag{2}$$

The evidence is a prediction made with all the prior hypotheses. Since it is a constant, the posterior uncertainty of each hypothesis is proportional to the product of their prior uncertainty and likelihood, as shown in Equation 3. Section 3.4 shows how these expressions are obtained by applying the sum and product rules.

$$\underbrace{p(\, s_i \,|\, r\,)}_{\text{Posterior}} \propto \underbrace{1 - \Phi(0 \,|\, s_i - \mu_j,\, 2\beta^2 + \sigma_j^2)}_{\text{Likelihood } P(r|s_i)} \underbrace{\mathcal{N}(s_i \,|\, \mu_i,\, \sigma_i^2)}_{\text{Prior } p(s_i)} \tag{3}$$

Dividing the right hand with the evidence $P(r)$ leads to the normalized posterior. It is interesting to note the similarities and differences between likelihood and evidence. The likelihood quantifies the same cumulative density as the evidence but is centered on the difference between the hypothesis we are evaluating (s_i) and the opponent's mean estimate (μ_j), with a variance that includes all uncertainties except the one of s_i.

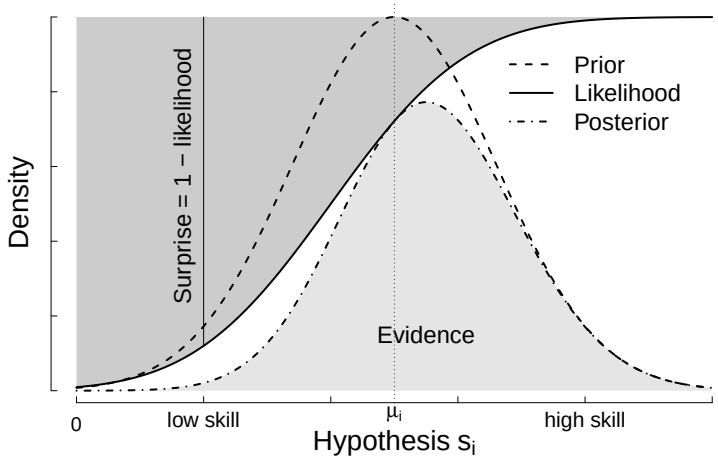


Figure 2: Update of the belief for the winning case. The proportional posterior is the product of the prior (Gaussian) and the likelihood (cumulative Gaussian). The evidence is the integral of the proportional posterior. The distributions are not necessarily on the same scale: the prior integrates to 1, while the likelihood goes from 0 to 1.

The posterior is just the prior's density that is not filtered by the likelihood. The surprise, defined as the likelihood's complement, works as a filter for the prior. In the region of very high-skill hypotheses, where the winning result would have generated almost no surprise ($\lim_{s_i \to \infty} P(r|s_i) = 1$), the posterior receives all the prior's density. In the region of low-skill hypotheses, a win would generate a great surprise ($\lim_{s_i \to -\infty} P(r|s_i) = 0$), and the posterior receives no density from the prior.

It is important to stress that the posterior, although similar, is not a Gaussian distribution, preventing us from using Equation 3 iteratively. But due to the shape of the exact posterior, a Gaussian distribution could be used as a good approximation, allowing us to avoid

the computational cost of the sampling methodologies. The success of the TrueSkill solution (Herbrich *et al.* 2006) is based on the usage of an efficient method for computing the Gaussian distribution that best approximates the exact posterior (see Section 3.6),

$$\widehat{p}(s_i|r,s_j) = \underset{\mu,\sigma}{\arg\min} \quad \mathrm{KL}(\, p(s_i|r,s_j) \,||\, \mathcal{N}(s_i|\mu,\sigma^2)\,) \tag{4}$$

in terms of Kullback-Leibler divergence minimization between the true and the approximate distribution. This method allows us to efficiently apply Equation 3 iteratively over a sequence of observations, which would otherwise be infeasible. The approach adopted by TrueSkill to treat the dynamical process, known as *filtering*, uses the last approximate posterior as the prior for the next event. The approximate posterior at any given time is defined as:

$$\widehat{\text{Posterior}}_t \propto \widehat{\text{Likelihood}}_t \overbrace{\widehat{\text{Likelihood}}_{t-1} \ldots \underbrace{\widehat{\text{Likelihood}}_1 \text{Prior}_1}_{\widehat{\text{Posterior}}_1 \text{ as Prior}_2}}^{\widehat{\text{Posterior}}_{t-1} \text{ as Prior}_t} \tag{5}$$

where $\widehat{\text{Posterior}}_i$ and $\widehat{\text{Likelihood}}_i$ represent the approximations induced by the Equation 4 at the i-th event. If we consider the likelihood as a filter of the prior, each posterior is the accumulation of all previous filters. In this way, information propagates from past to future estimates. Since skills change over time, it is necessary to incorporate some uncertainty γ after each step.

$$\widehat{p}(s_{i_t}) = \mathcal{N}(s_{i_t}|\mu_{i_{t-1}}, \sigma^2_{i_{t-1}} + \gamma^2) \tag{6}$$

Because the filtering approach is an ad-hoc procedure that does not arise from any probabilistic model, its estimates have some problems. The most obvious is that the beginning of any sequence of estimates always has high uncertainty. But temporal and spatial decouplings may also occur, preventing the comparison between distant estimates. Although the relative differences between current estimates within well-connected communities are correct, estimates separated in time and between poorly connected communities may be incorrect. All of these issues are related to the fact that information propagates in only one direction through the system and can be solved by inferring with the available information from events occurring in parallel jointly with future events.

1.2. TrueSkill Through Time model

To solve the limitations of the TrueSkill algorithm, it is necessary to perform the inference within a Bayesian network that includes all historical activities, enabling the information to propagate throughout the system. This ensures both good initial estimates and comparability of estimates distant in time and space. The connectivity between events is generated by the basic assumption that a player's skill at time t depends on his skill at an earlier time $t-1$, generating a network that acquires its structure depending on who participates in each event. Coulom (2008) and Maystre *et al.* (2019) implemented similar algorithms based on Laplacian approximations and Gaussian processes. Excluding the dynamic component, $\gamma = 0$, the prior of the agent i at the t-eth event is just the product of all their likelihoods, except the one of the t-eth event.

$$\text{Prior}_{i_t} = \text{Prior}_{i_0} \underbrace{\prod_{k=1}^{t-1} \text{Likelihood}_{i_k}}_{\text{Past information}} \underbrace{\prod_{k=t+1}^{T_i} \text{Likelihood}_{i_k}}_{\text{Future information}} \tag{7}$$

where T_i is the total number of events with agent i participation, and Prior_{i_0} is the initial prior. It produces a mutual dependence between estimates that forces us to iteratively use the last available likelihoods until convergence is reached (details in Section 3.8). Figure 3

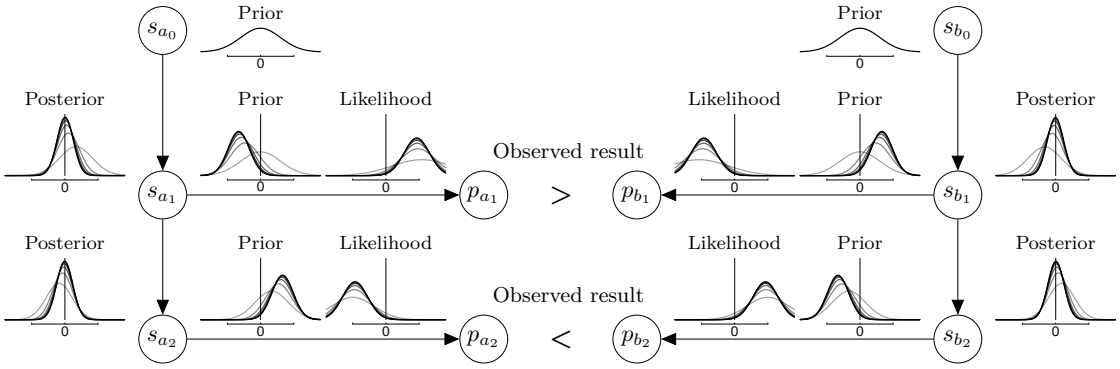


Figure 3: Convergence of a Bayesian network consisting of two events and two agents: the first game is won by player a, and the second one is won by player b. The brightness of the curves indicates the order: the first one (the clearest) corresponds to the TrueSkill estimates, and the last one (the darkest) corresponds to the TrueSkill Through Time estimates.

shows how the estimates converge in a Bayesian network with two agents and two events. According to what the data suggests (one win each), TrueSkill Through Time recovers the differences between skills, indicating that both players have the same skill (both posterior centered on zero). On the opposite, TrueSkill offers biased estimates.

The advantage of TrueSkill Through Time lies in its temporal causal model, which allows information to propagate throughout the system. Unlike neural networks that have regular structures, these Bayesian networks acquire a complex structure typically growing up to millions of parameters (e.g., video games). Notwithstanding, this procedure converges within a few linear iterations over the data. The correction of biases is a fundamental step in constructing reliable estimators that serve both for decision-making in sensitive areas and for evaluation of scientific theories that use the skill as observable data. In this work, we make available the first TrueSkill Through Time packages for Julia, Python, and R jointly with their complete scientific documentation (Landfried 2021).

2. Illustrations

This section shows how to use Julia, Python, and R packages. We present four examples: a single event, a three-event sequence, the skill evolution of a player, and the analysis of the Association of Tennis Professionals (ATP) historical data. We use both TrueSkill and TrueSkill Through Time models and show the steps to obtain the posteriors, the learning curves, and the prior prediction of the observed data (i.e., evidence). We identify the different programming languages using the following color scheme:

```
Syntax common to Julia, Python and R
Julia syntax                Python syntax                R syntax
```

where the full line is used when the syntax of the three languages coincides, and when the

languages have different syntax, we use different colors: Julia on the left, Python in the middle, and R on the right.

2.1. Single event

We define the class `Game` to model events and perform inference given the teams' composition, the result, and the typical draw probability for those events (`p_draw`). The features of the agents are defined within the class `Player`: the prior Gaussian distribution characterized by the mean (`mu`) and the standard deviation (`sigma`), the standard deviation of the performance (`beta`), and the dynamic factor of the skill (`gamma`). In the following code, we define the variables we will use later, assigning the default values.

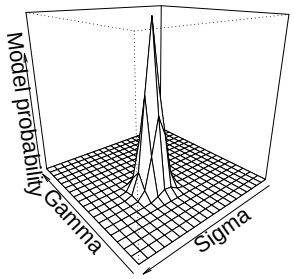
```
mu = 0.0; sigma = 6.0; beta = 1.0; gamma = 0.03; p_draw = 0.0
```

Code 1: Package parameters and their default values.

The initial value of `mu`, shared by all players, can be freely chosen because the difference in skills is what matters and not its absolute value. The prior's standard deviation `sigma` must be sufficiently large to include all possible skill hypotheses. `beta` (β) is, perhaps, one of the most important parameters because it works as the estimate's scale. A difference of one β between two skills ($s_i - s_j = \beta$) represents a 76 % probability of winning. Since it is the unit of measurement, we choose `beta=1.0`. The dynamic factor `gamma` is generally a fraction of `beta`. The draw probability (`p_draw`) is usually initialized with the observed draws frequency.

Parameter	Default value
`mu`	0.0
`sigma`	6.0
`beta`	1.0
`gamma`	0.03
`p_draw`	0.0

(a) Parameters.



(b) Optimization.

Figure 4: (a) presents the model's parameters and their default values. (b) shows the combination space for `sigma` and `gamma`, including the zone which maximizes the model probability given the data.

Figure 4 summarizes these default values and shows the possibility of optimizing two of them, whose values depend on the data set. We create four identical players using these values.

```
a1 = Player(Gaussian(mu, sigma), beta, gamma); a2 = Player(); a3 = Player(); a4 = Player()
```

Code 2: Players initialization.

The first player is created by explicitly writing the parameters. For the rest of them, we use the default values. The `Gaussian` class models the standard operations of Gaussian distributions, including multiplication, summation, division, and subtraction (details in Section 3.3). In the next step, we create a game with two teams of two players. When dealing with teams, the observed result depends on the sum of the performances of each member (see details in Section 3.4).

```
team_a = [ a1, a2 ]
team_b = [ a3, a4 ]
teams = [team_a, team_b]
g = Game(teams)
```

```
team_a = [ a1, a2 ]
team_b = [ a3, a4 ]
teams = [team_a, team_b]
```

```
team_a = c(a1, a2)
team_b = c(a3, a4)
teams = list(team_a, team_b)
```

Code 3: Teams and game initialization.

where the teams' order in the list implicitly defines the game's result: the teams appearing first in the list (lower index) beat those appearing later (higher index). This is one of the simplest usage examples. Later on, we will learn how to specify the result explicitly. During the initialization, the class `Game` computes the prior prediction of the observed result (`evidence`) and the approximate likelihood of each player (`likelihoods`).

```
lhs = g.likelihoods[1][1]
ev = g.evidence
ev = round(ev, digits = 3)
print(ev)
> 0.5
```

```
lhs = g.likelihoods[0][0]
ev = g.evidence
ev = round(ev, 3)
```

```
lhs = g@likelihoods
ev = g@evidence
ev = round(ev, 3)
```

Code 4: Evidence and likelihoods queries.

In this case, the evidence is 0.5 because both teams had the same prior skill estimates. Posteriors can be found by manually multiplying the likelihoods and priors, or we can call the method `posteriors()` of class `Game` to compute them. The likelihoods and posteriors are stored keeping the original order in which players and teams are specified during the initialization of the class `Game`.

```
pos = posteriors(g)
print(pos[1][1])
> Gaussian(mu = 2.361, sigma = 5.516)
print(lhs[1][1] * a1.prior)
> Gaussian(mu = 2.361, sigma = 5.516)
```

```
pos = g.posteriors()
print(pos[0][0])
print(lhs[0][0] * a1.prior)
```

```
pos = posteriors(g)
print(pos[[1]][[1]])
print(lhs[[1]][[1]]*a1@prior)
```

Code 5: Posteriors query and their manual computation.

where the printed posterior corresponds to the first player of the first team. Due to the winning result, the player's estimate now has a larger mean and a smaller uncertainty. The product of Gaussians (i.e., the likelihood times the prior) generates the same normalized posterior.

We now analyze a more complex example in which the same four players participate in a multi-team game. The players are organized into three teams of different sizes: two teams with only one player and the other with two players. The result has a single winning team and a tie between the other two losing teams. Unlike the previous example, we need to use a draw probability greater than zero.

```
ta = [a1]
tb = [a2, a3]
tc = [a4]
teams_3 = [ta, tb, tc]
result = [1., 0., 0.]
g = Game(teams_3, result, p_draw = 0.25)
```

```
ta = [a1]
tb = [a2, a3]
tc = [a4]
teams_3 = [ta, tb, tc]
result = [1, 0, 0]
```

```
ta = c(a1)
tb = c(a2, a3)
tc = c(a4)
teams_3 = list(ta, tb, tc)
result = c(1, 0, 0)
```

Code 6: Game with multiple teams of different sizes and the possibility of tie.

where the variable `teams` contains the players distributed in different teams while the variable `result` contains the score obtained by each team. The team with the highest score is the

winner, and the teams with the same score are tied. In this way, we can specify any outcome, including global draws. The evidence and posteriors can be queried in the same way as before.

2.2. Sequence of events

We can use the class `History` to compute the posteriors and evidence of a sequence of events. In the first example, we instantiate the class `History` with three players (`"a"`, `"b"`, and `"c"`) and three games. In the first game, `"a"` beats `"b"`. In the second game, `"b"` beats `"c"`, and in the third game, `"c"` beats `"a"`. In brief, all agents win one game and lose the other.

```
c1 = [["a"],["b"]]
c2 = [["b"],["c"]]
c3 = [["c"],["a"]]
composition = [c1, c2, c3]
```

```
c1 = [["a"],["b"]]
c2 = [["b"],["c"]]
c3 = [["c"],["a"]]
composition = [c1, c2, c3]
```

```
c1 = list(c("a"),c("b"))
c2 = list(c("b"),c("c"))
c3 = list(c("c"),c("a"))
composition = list(c1,c2,c3)
```

```
h = History(composition, gamma = 0.0)
```

Code 7: Initialization of a `History`'s instance with a three events sequence.

where the variables `c1`, `c2`, and `c3` model the composition of each game using the names of the agents (i.e., their identifiers), the variable `composition` is a list containing the three events, and the zero value of the parameter `gamma` specifies that skills do not change over time. The results are defined implicitly by the order in which the game compositions are initialized: the first teams in the list defeat those appearing later. The rest of the parameters are initialized using the default values, as shown in Code 1.

In this example, all agents beat each other, and their skills do not change over time. The data suggest that all agents have the same skill. After initialization, the class `History` immediately instantiates a new player for each name and activates the computation of the TrueSkill estimates, using the posteriors of each event as a prior for the next one. To access them, we can call the method `learning_curves()` of the class `History`, which returns a dictionary indexed by the names of the agents. Individual learning curves are lists of tuples: each has the time of the estimate as the first component and the estimate itself as the second.

```
lc = learning_curves(h)
print(lc["a"])
```

```
lc = h.learning_curves()
print(lc["a"])
```

```
lc = h$learning_curves()
lc_print(lc[["a"]])
```

```
> [(1, Gaussian(mu = 3.339, sigma = 4.985)), (3, Gaussian(mu = -2.688, sigma = 3.779))]
```

```
print(lc["b"])
```

```
print(lc["b"])
```

```
lc_print(lc[["b"]])
```

```
> [(1, Gaussian(mu = -3.339, sigma = 4.985)), (2, Gaussian(mu = 0.059, sigma = 4.218))]
```

Code 8: Learning curves of players participating in a sequence of events.

The learning curves of players `"a"` and `"b"` contain one tuple per game (not including the initial prior). Despite no player is more skilled than the others, the estimates obtained by TrueSkill present strong variations between players. The estimates obtained after the first game (`"a"` beats `"b"`) have the same uncertainty and mean absolute value, being positive for the winner and negative for the other. Estimates computed after the events with the participation of `"c"` have lower uncertainty and mean values closer to zero.

TrueSkill Through Time solves TrueSkill's inability to obtain correct estimates by allowing the information to propagate throughout the system. To compute them, we call the method `convergence()` of the class `History`.

```
convergence(h)
lc = learning_curves(h)
```

```
h.convergence()
lc = h.learning_curves()
```

```
h$convergence()
lc = h.learning_curves()
```

```
print(lc["a"])                print(lc["a"])                lc_print(lc[["a"]])
> [(1, Gaussian(mu = 0.0, sigma = 2.395)), (3, Gaussian(mu = -0.0, sigma = 2.395))]
print(lc["b"])                print(lc["a"])                lc_print(lc[["a"]])
> [(1, Gaussian(mu = -0.0, sigma = 2.395)), (3, Gaussian(mu = 0.0, sigma = 2.395))]
```

Code 9: Computing TrueSkill Through Time learning curves.

TrueSkill Through Time returns correct estimates (i.e., same value for all players) and has less uncertainty.

2.3. Skill evolution

We now analyze a scenario in which a new player joins a large community of already-known players. In this example, we focus on the estimation of an evolving skill. For this purpose, we establish the skill of the target player to change over time following a logistic function. We generate the community to ensure that the opponents have a skill similar to our target player throughout the evolution. We generate the target player's learning curve and 1000 random opponents in the following code.

```
import math; from numpy.random import normal, seed; seed(99); N = 1000
def skill(experience, middle, maximum, slope):
    return maximum/(1+math.exp(slope*(-experience+middle)))
target = [skill(i, 500, 2, 0.0075) for i in range(N)]
opponents = normal(target, scale = 0.5)
```

Code 10: Initialization of the target's learning curve and the community of opponents.

Here we only include the Python version (Appendix 5.1 includes Julia and R versions). The list `target` has the agent's skills at each moment: the values start at zero and grow smoothly until the target player's skill reaches two. The list `opponents` includes the randomly generated opponents' skills following a Gaussian distribution centered on each target's skills and a standard deviation of 0.5.

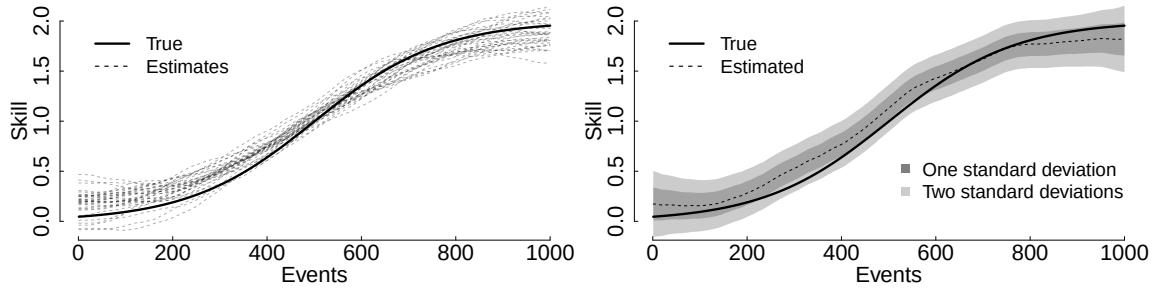
```
composition = [[["a"], [str(i)]] for i in range(N)]
results = [[1,0] if normal(target[i]) > normal(opponents[i]) else [0,1] for i in range(N)]
times = [i for i in range(N)]
priors = dict([(str(i), Player(Gaussian(opponents[i], 0.2))) for i in range(N)])
h = History(composition, results, times, priors, gamma = 0.015)
h.convergence()
mu = [tp[1].mu for tp in h.learning_curves()["a"]]
```

Code 11: Estimating the simulated learning curve from random results.

In this code, we define four variables to instantiate the class `History` to compute the target's learning curve. The variable `composition` contains 1000 games between the target player and different opponents. The list `results` is generated randomly by sampling the agents' performance following Gaussian distributions centered on their skills. The winner is the player with the highest performance. The variable `time` is a list of integer values ranging from 0 to 999 representing the time batch in which each game is located: the class `History` uses the temporal distance between events to determine the amount of dynamic uncertainty (γ^2) to be added between games. The variable `priors` is a dictionary used to customize player attributes: we assign low uncertainty to the opponents' priors as we know their skills beforehand.

The class `History` receives these four parameters and initializes the target player using the

default values and a dynamic uncertainty `gamma=0.015`. Using the method `convergence()`, we obtain the TrueSkill Through Time estimates and the target's learning curve. Because the estimates depend on random results, we repeatedly execute Code 11 to consider their variability. Figure 5 shows the evolution of the actual (solid line) and estimated (dotted line) target player's learning curves. The estimated learning curves remain close to the actual skill



(a) Mean estimates.

(b) Uncertainty of an estimate.

Figure 5: True and estimated learning curves of a new player joining a large community of already-known players. The solid line represents the target player's skill, while the dashed lines show the mean estimates. The dark and light gray areas show one and two times the uncertainty of one of the estimates.

during the whole evolution as can be seen in Fig. 5(a). For Fig. 5(b), we select one of the estimated learning curves and present its uncertainty, showing that the uncertainty interval contains the actual learning curve. This example exhibits that TrueSkill Through Time can follow the evolution of a new player's skill.

2.4. The history of the Association of Tennis Professionals (ATP)

This last example analyzes the complete history of the Association of Tennis Professionals (ATP) registered matches. The publicly-available database has 447 000 games from 1915 to 2020 with more than 19 000 participating players. The information stored in a single `CSV` file[1] includes single and double matches: if the column `double` has the letter `t`, the game is a double match. Each game has an identifier (i.e., `match_id`) and its tournament's round number (i.e., `round_number`), where 0 represents the final game, 1 the semi-final increasing until the tournament end. The file also contains players' identifiers and names. For example, column `w2_id` is the second player's identifier of the winning team, and `l1_name` is the first player's name of the losing team. Finally, we have the tournament's name (`tour_name`), its identifier (`tour_id`), the tournament's starting date (`time_start`), and the type of surface (`ground`). Here we only show the Julia code because it is far more efficient than Python and R versions (Appendix 5.2 shows the Python and R codes, and Section 4 compares the performance between them).

```
using CSV; using Dates
data = CSV.read("atp.csv")

dates = Dates.value.(data[:,"time_start"] .- Date("1900-1-1"))
```

[1] Available at `https://github.com/glandfried/tennis_atp/releases/download/atp/history.csv.zip`. Additional data at Jeff Sackmann's site: `https://github.com/JeffSackmann/tennis_atp`.

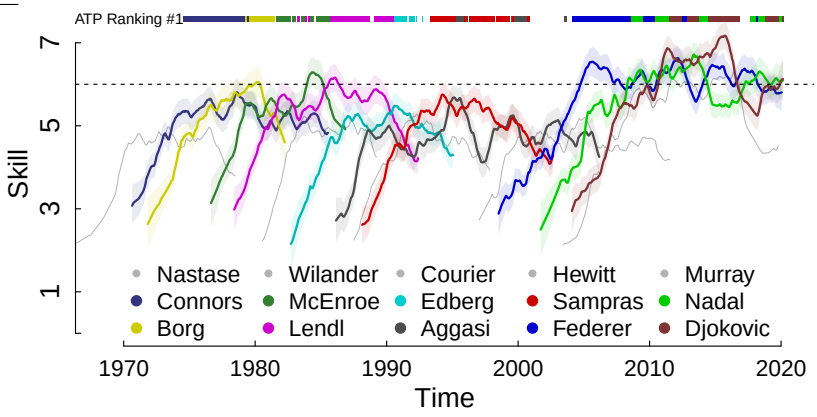
```
matches = [ r.double == "t" ? [[r.w1_id,r.w2_id],[r.l1_id,r.l2_id]] : [[r.w1_id],[r.l1_id]]
    for r in eachrow(data) ]

h = History(composition = matches, times = dates, sigma = 1.6, gamma = 0.036)
convergence(h, epsilon = 0.01, iterations = 10)
```

Code 12: The history of the Association of Tennis Professionals.

In this code, we open the file `atp.csv`, create the variables `times` and `composition`, and instantiate the class `History`. We define the event times as the days elapsed from a reference date to the tournament start date, assuming that the skill is the same within each tournament. When generating the list `composition`, we discriminate whether the games are doubles or singles. The composition's order establishes the results, placing the winning team first. When initializing the class `History`, we set the values of `sigma` and `gamma` based on an optimization procedure previously performed (recall Figure 4(b)). Finally, we use the `convergence()` method to obtain TrueSkill Through Time estimates explicitly selecting the convergence criterion: when the change between iterations is less than 0.01 or when reaching ten iterations. Table 6(a) shows the historical ranking of players in the top position of the

Pos.	Player	Weeks
1	Novak Djokovic	320
2	Roger Federer	310
3	Pete Sampras	286
4	Ivan Lendl	270
5	Jimmy Connors	268
6	Rafael Nadal	209
7	John McEnroe	170
8	Björn Borg	109
9	Andre Agassi	101
10	Lleyton Hewitt	80
11	Stefan Edberg	72
12	Jim Courier	58
13	Gustavo Kuerten	43
14	Andy Murray	41
15	Ilie Năstase	40
16	Mats Wilander	20

(a) Weeks at first position.



(b) Estimated learning curves for some famous male players.

Figure 6: (a) Historical ranking according to the number of weeks that players reached the first position in the ATP's ranking until March 10, 2020. (b) Estimated skill of some of the historical ATP's leaders. The shaded area represents a standard deviation of uncertainty. The top bar indicates which player was at the top of the ATP's ranking. The dotted line is located at six skill points and helps to compare the curves.

ATP's ranking according to the number of weeks occupying the first position. The table is updated to the date of writing this manuscript and, as the ATP's tour was suspended due to COVID-19, 22 weeks are excluded from consideration. Figure 6(b) presents the estimated learning curves of some famous male players in ATP's history, which we identified using different colors. The top bar indicates which player was at the top of the ATP's ranking (the bar has no color when player number 1 is not considered in our analysis).

ATP ranking points are updated every Monday according to the tournament's prestige and the stage reached. Only during brief periods, there is no coincidence between the estimated skill and the top player of the ATP's ranking, showing a good agreement between both methodologies. Notwithstanding, there are some notorious differences between estimated

players' skills and the ATP's ranking, especially concerning the historical ranking shown in Table 6(a). On the one hand, Lleyton Hewitt's position in the historical ranking is the product of a window of opportunity opened around the year 2000 since his ability is relatively low in historical terms. On the other hand, Andy Murray is the fourth most skilled player according to our estimations, but he only reaches 14th place in the historical ranking, just one place above Ilie Năstase.

TrueSkill Through Time allows comparing the relative ability of players over time, unlike historical ATP's ranking and estimators based on the filtering approach (such as TrueSkill). The learning curves share a similar pattern: they begin with rapid growth, reach an unstable plateau, and end with a slow decline (for visualization purposes, we hide the last portion of the players having long final stages). Individual learning curves enable recognition of special periods of crisis and prolonged stability of the professional players, and even the effects of emotional slumps such as those suffered by Aggasi and Djokovic. It is worthwhile to note that the skill of tennis players did not increase abruptly over the years: contrary to what is expected, the players of the 1980s were more skilled than those of the 1990s and reached a skill similar to what Federer, Nadal, and Djokovic had in 2020, even though the latter reached higher values for a longer time.

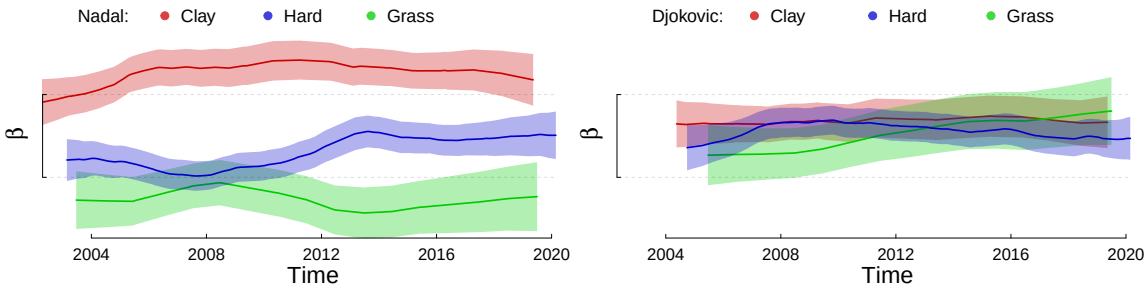
The previous example summarizes the players' skills using a single dimension. TrueSkill Through Time allows estimating multi-dimensional skills. It is widely recognized that the ability of certain tennis players varies significantly depending on the surface. To quantify this phenomenon, we propose modeling each player as a team composed of a generic player, who is included in all the games, and another player representing the player's ability on a particular surface. For example, Nadal is represented by a two-player team: *Nadal_generic* and *Nadal_clay* when playing on this kind of surface, and *Nadal_generic* and *Nadal_grass* when participating in the Wimbledon tournament.

```
players = Set(vcat((composition...)...))
priors = Dict([(p, Player(Gaussian(0., 1.6), 1.0, 0.036) ) for p in players])

composition_ground = [ r.double == "t" ? [[r.w1_id, r.w1_id*r.ground, r.w2_id,
    r.w2_id*r.ground],[r.l1_id, r.l1_id*r.ground, r.l2_id, r.l2_id*r.ground]] : [[r.w1_id,
    r.w1_id*r.ground],[r.l1_id, r.l1_id*r.ground]] for r in eachrow(data) ]

h_ground = History(composition = composition_ground, times = dates, sigma = 1.0, gamma =
    0.01, beta = 0.0, priors = priors)
convergence(h_ground, epsilon = 0.01, iterations = 10)
```

Code 13: Modeling multi-dimensional skills in ATP history.

In this example, we keep the same prior as in Code 12 for all the generic players, but in this code, we define them using the variable `priors`. We create the teams depending on whether the game is double or single, similarly to Code 12 but now adding the specific surface skills of each player as their teammate (we use the operator `*` to concatenate strings). As the specific surface skills are not defined in the variable `prior`, we use the default values defined in the class `History` for initialization. We also define `beta` as null for specific surface skills to avoid adding additional noise to the players' performance, keeping the scale of the estimates stable. We select a `sigma` that we consider sufficiently large and a dynamic factor `gamma` representing 1 % of the prior uncertainty.

In Figure 7, we show the skill difference that Nadal and Djokovic have in each of the three types of ground. Nadal has a notorious skill difference when playing on different surfaces. On



(a) Nadal. (b) Djokovic.

Figure 7: Skill difference on the three main types of surface. Each point on the y-axis represents a distance of one β, i.e., 76 % probability of winning.

the contrary, Djokovic has very similar skills in the three types. The Nadal's skill difference between clay and grass grounds is greater than one β, which means at least a 76 % difference in the probability of winning compared to itself. In the case of Nadal, it seems important to consider the skill's multi-dimensionality, while in Djokovic's case, it seems reasonable to summarize it in a single dimension. To assess whether the complexity added by modeling multi-dimensionality is appropriate in general terms, we can compare the joint prior prediction of the models, calling the method `log_evidence()` of the class `History`.

In tennis, it is sufficient to summarize the skills in a single dimension since the prior prediction is maximized when the parameters of the surface's factors (i.e., σ and γ) vanish. In other examples, where the multi-dimensionality of skills could be more relevant, it should be necessary to model the skills of all agents using different components. If we consider only the games in which Nadal participates, optimality is achieved when the parameters take the values $\sigma = 0.35$ and $\gamma = 0$, meaning that it is necessary to model multidimensional skills ($\sigma > 0$) but considering that their effect does not change over time ($\gamma = 0$). In this scenario, Nadal's ability on Clay is 0.87β higher than on Hard and 1.05β higher than on Grass.

3. Models and software

This section provides the complete mathematical documentation of the TrueSkill Through Time model. The advantage of this model lies in the strict application of probability theory: all the assumptions are made explicit through a generative model, and the inference is solved only with the rules of probability, nothing more than the sum and the product rules. Section 3.1 introduces the *sum-product algorithm*, which allows us to apply these rules to compute the marginal distributions, e.g., the posterior and the prior prediction. Section 3.2 lists the properties needed to derive the marginal distributions of interest. In Section 3.3, we introduce the operations of the class `Gaussian`, which does most of the computation. In sections 3.4, 3.5, and 3.6, we show how to solve the prior prediction and the exact posterior of an event, then we introduce the model for a draw and explain how to approximate the exact posterior in events with two teams. Section 3.7 explains the general multi-team solution requiring an iterative algorithm. Section 3.8 justifies the mathematical steps required to solve the full TrueSkill Through Time model.

3.1. Sum-product algorithm

The *sum-product algorithm* (Kschischang *et al.* 2001) takes advantage of the structure of the joint probability distribution imposed by the causal model to apply the probability rules efficiently. Any model can be factorized into the product of conditional probabilities. Based on the independence between variables, our model (Figure 1) can be factorized as,

$$p(\boldsymbol{s}, \boldsymbol{p}, d, r) = p(s_1)p(s_2)p(p_1|s_1)p(p_2|s_2)p(d|\boldsymbol{p})P(r|d) \tag{8}$$

Figure 8 shows this factorization graphically. These representations, known as *factor graphs*, have two kinds of nodes: variable-type nodes (white circles) and function-type nodes (black squares). The edge between node variables and node functions represents the mathematical relationship "the variable v is an argument of the function f". In our case, we want to compute

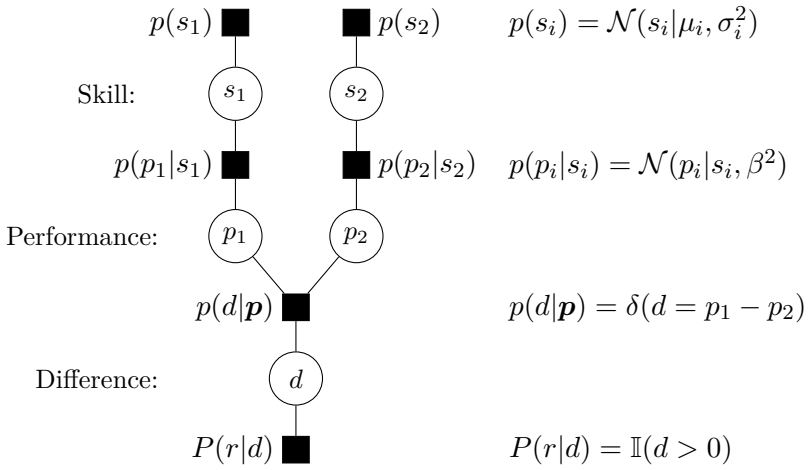


Figure 8: Graphical way of representing the joint distribution factorization induced by the basic causal model, represented by Equation 8. Black squares represent the functions, white circles represent the variables, and the edges between them represent the mathematical relationship "the variable is the argument of the function".

two marginals, the proportional posterior of the skills $p(s_i, r)$ and the prior probability of the result $p(r)$. The *sum-product algorithm* is a general way of breaking down the rules of probability as messages sent between the nodes of the *factor graph*. There are two types of messages: those sent by variable-type nodes to their function-type neighbors ($m_{v\to f}(v)$) and the ones that function-type nodes send to their variable-type neighbors ($m_{f\to v}(v)$). The former partially performs the product rule.

$$m_{v\to f}(v) = \prod_{h\in n(v)\setminus\{f\}} m_{h\to v}(v) \qquad \text{(product step)}$$

where $n(v)$ represents the set of neighbor nodes to v. In brief, the messages sent by the variable-type node v are the product of the messages that v receives from the rest of their neighbors $h \in n(v)$ except f. The messages sent by the function-type nodes encode a portion of the sum rule.

$$m_{f\to v}(v) = \int \cdots \int \left(f(\boldsymbol{h}, v) \prod_{h\in n(f)\setminus\{v\}} m_{h\to f}(h) \right) d\boldsymbol{h} \qquad \text{(sum step)}$$

where $\boldsymbol{h} = n(f) \setminus \{v\}$ is the set of all neighbors to f except v, and $f(\boldsymbol{h}, v)$ represents the function f, evaluated in all its arguments. In brief, the messages sent by a function f are also the product of the messages it receives from the rest of its neighbors $h \in n(f)$ except v, but this time it is also multiplied by itself $f(\cdot)$ and integrated (or summed) over $\boldsymbol{h}$. Finally, the marginal probability distribution of a variable v is simply the product of the messages that v receives from all its neighbors.

$$p(v) = \prod_{h \in n(v)} m_{h \to v} \qquad \text{(marginal probability)}$$

This algorithm encodes the minimum number of steps required to calculate any marginal probability distribution.

3.2. Mathematical properties and notation

The efficiency of TrueSkill Through Time is based on the analytical computation of marginal probabilities. This section lists the needed properties to derive the exact and approximate messages generated by the sum-product algorithm. The first property states that the product of two Gaussian distributions evaluated at the same point x can be expressed as the product of two other Gaussian distributions with only one evaluated at x.

$$\mathcal{N}(x|\mu_1, \sigma_1^2)\mathcal{N}(x|\mu_2, \sigma_2^2) \overset{5.3}{=} \mathcal{N}(\mu_1|\mu_2, \sigma_1^2 + \sigma_2^2)\mathcal{N}(x|\mu_*, \sigma_*^2) \qquad \text{(Gaussian product)}$$

where $\mu_* = \frac{\mu_1}{\sigma_1^2} + \frac{\mu_2}{\sigma_2^2}$ y $\sigma_*^2 = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)^{-1}$. Something similar occurs with the division of two Gaussian distributions, both evaluated at the same point x.

$$\mathcal{N}(x|\mu_1, \sigma_1^2)/\mathcal{N}(x|\mu_2, \sigma_2^2) \overset{5.6}{\propto} \mathcal{N}(x|\mu_\div, \sigma_\div^2)/\mathcal{N}(\mu_1|\mu_2, \sigma_1^2 + \sigma_2^2) \qquad \text{(Gaussian division)}$$

where $\mu_\div = \frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2}$ y $\sigma_\div^2 = \left(\frac{1}{\sigma_1^2} - \frac{1}{\sigma_2^2}\right)^{-1}$.

The indicator function $\mathbb{I}(\cdot = \cdot)$ is 1 when equality is true and 0 otherwise. It represents probabilities distributions of non-random discrete variables, such as the result of the games given the difference of performances $p(r|d)$. Similarly, the Dirac delta function $\delta(\cdot = \cdot)$ represents probabilities distributions of non-random continuous variables, such as the difference of performances given the agents' performances $p(d|\boldsymbol{p})$. If we can use it to replace a variable within an integral,

$$\iint \delta(x = h(y,z)) f(x) g(y)\, dx\, dy = \int f(h(y,z)) g(y) dy \qquad \text{(Dirac delta function)}$$

the dimensionality of the problem is reduced. We also use the properties derived from the symmetry of Gaussian distributions.

$$\mathcal{N}(x|\mu, \sigma^2) = \mathcal{N}(\mu|x, \sigma^2) = \mathcal{N}(-\mu| - x, \sigma^2) = \mathcal{N}(-x| - \mu, \sigma^2) \qquad \text{(Gaussian symmetry)}$$

The Gaussian standardization,

$$\mathcal{N}(x|\mu, \sigma^2) = \mathcal{N}((x - \mu)/\sigma|0, 1) \qquad \text{(Gaussian standardization)}$$

Equality between the Gaussian distribution and the derivative of their cumulative distribution,

$$\frac{\partial}{\partial x}\Phi(x|\mu, \sigma^2) = \mathcal{N}(x|\mu, \sigma^2) \qquad \text{(Derivative of the cumulative Gaussian)}$$

which is valid by definition. The symmetry of the cumulative Gaussian distribution.

$$\Phi(0|\mu,\sigma^2) = 1 - \Phi(0|-\mu,\sigma^2) \qquad \text{(Symmetry of the cumulative Gaussian)}$$

3.3. The Gaussian class

The `Gaussian` class does most of the computation of the packages. It is characterized by the mean (`mu`) and the standard deviation (`sigma`).

```
N1 = Gaussian(mu = 1.0, sigma = 1.0); N2 = Gaussian(1.0, 2.0)
```

Code 14: Initialization of Gaussian distributions.

The class overwrites the addition (`+`), subtraction (`-`), product (`*`), and division (`/`) to compute the marginal distributions used in the TrueSkill Through Time model.

$$\mathcal{N}(x|\mu_1,\sigma_1^2)\mathcal{N}(x|\mu_2,\sigma_2^2) \overset{5.3}{\propto} \mathcal{N}(x|\mu_*,\sigma_*^2) \qquad \texttt{(N1 * N2)}$$

$$\mathcal{N}(x|\mu_1,\sigma_1^2)/\mathcal{N}(x|\mu_2,\sigma_2^2) \overset{5.6}{\propto} \mathcal{N}(x|\mu_\div,\sigma_\div^2) \qquad \texttt{(N1 / N2)}$$

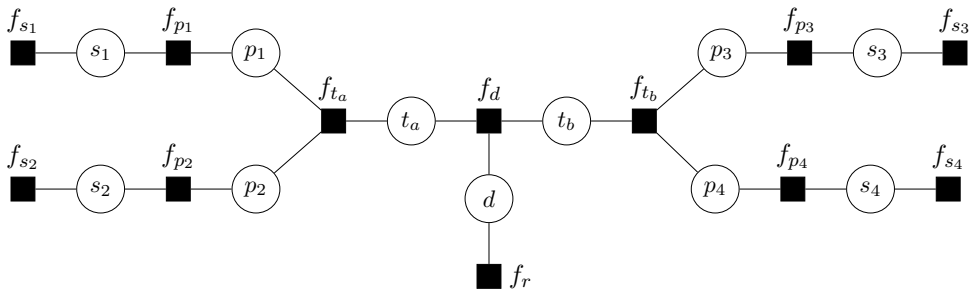
$$\iint \delta(t=x+y)\mathcal{N}(x|\mu_1,\sigma_1^2)\mathcal{N}(y|\mu_2,\sigma_2^2)dxdy \overset{5.4}{=} \mathcal{N}(t|\mu_1+\mu_2,\sigma_1^2+\sigma_2^2) \qquad \texttt{(N1 + N2)}$$

$$\iint \delta(t=x-y)\mathcal{N}(x|\mu_1,\sigma_1^2)\mathcal{N}(y|\mu_2,\sigma_2^2)dxdy \overset{5.4}{=} \mathcal{N}(t|\mu_1-\mu_2,\sigma_1^2+\sigma_2^2) \qquad \texttt{(N1 - N2)}$$

Although these properties are widely known, we attach their complete demonstrations in the supplemental material.

3.4. The exact solution for events with two teams

In the presence of teams, the Elo model assumes that a team's performance is the sum of the performance of its members. The team with the highest performance wins, $r = (t_i > t_j)$. Figure 9 shows the graphical factorization of the Elo team model. In this example, we have



$$f_{s_i} = \mathcal{N}(s_i|\mu_i,\sigma^2) \quad f_{p_i} = \mathcal{N}(p_i|s_i,\beta^2) \quad f_{t_e} = \delta(t_e = \textstyle\sum_{i\in A_e} p_i) \quad f_d = \delta(d = t_a - t_b) \quad f_r = \mathbb{I}(d > 0)$$

Figure 9: Factor graph of a game with two teams with two players each. The Elo's team model incorporate a new variable, t, that models the team performance.

two teams with two players each. Every two-teams game has an analytical solution.

This section shows the steps to compute the exact evidence and likelihoods for a game with two teams. We only need the *sum-product algorithm* and the properties mentioned above. We will start first with the "descending" messages from the priors to the result until computing the evidence. Then, we continue with the "ascending" messages from the observed result and to the priors until computing each agent's posterior.

Descending messages. Following the sum step of the sum-product algorithm and the factorization of the model displayed in Fig. 9, messages sent by the skill factors f_{s_i} to the variable s_i are just the priors.

$$m_{f_{s_i}\to s_i}(s_i) = \mathcal{N}(s_i|\mu_i, \sigma_i^2) \qquad \texttt{(prior)}$$

We have access to this message by calling the `prior` attribute of the `Player` class. Following the product step of the sum-product algorithm and the factorization of the model, the message sent by the variable s_i to the performance factor f_{p_i} is the prior. Since it is trivial to calculate the messages sent by the variables (they are the product of the messages received from behind), we do not include them. Then, the message sent by the performance factors f_{p_i} to the variable p_i is:

$$m_{f_{p_i}\to p_i}(p_i) = \int \mathcal{N}(p_i|s_i, \beta^2)\mathcal{N}(s_i|\mu_i, \sigma_i^2)ds_i = \mathcal{N}(p_i|\mu_i, \beta^2 + \sigma_i^2) \qquad \texttt{(performance())}$$

We have access to these messages through the `performance()` method of the `Player` class.

```
p1 = performance(a1)
p2 = performance(a2)
p3 = performance(a3)
p4 = performance(a4)
```

```
p1 = a1.performance()
p2 = a2.performance()
p3 = a3.performance()
p4 = a4.performance()
```

```
p1 = performance(a1)
p2 = performance(a2)
p3 = performance(a3)
p4 = performance(a4)
```

Code 15: Computing the individual prior performance.

where the agents `a1`, `a2`, `a3`, and `a4` were initialized in Code 2. The message sent by the team factors f_{t_e} to the team variable t_e is an integral over all the individual performance variables,

$$\begin{aligned} m_{f_{t_e}\to t_e}(t_e) &= \iint \delta(t_e = p_i + p_j)\mathcal{N}(p_i|\mu_i, \beta^2 + \sigma_i^2)\mathcal{N}(p_j|\mu_j, \beta^2 + \sigma_j^2)dp_i dp_j \\ &= \mathcal{N}(t_e|\underbrace{\mu_i + \mu_j}_{\mu_e}, \underbrace{2\beta^2 + \sigma_i^2 + \sigma_j^2}_{\sigma_e^2}) \end{aligned} \qquad \texttt{(ta = p1 + p2)}$$

where the Dirac delta function imposes the constraint that the sum of the individual performances equals a constant team performance value t_e. Applying the already presented properties, we can solve this integral analytically, obtaining the prior performance of the teams, which is a Gaussian distribution centered on the sum of the mean estimates $\mu_e = \mu_i + \mu_j$ with a variance σ_e^2 including both the uncertainties of the estimates (i.e., $\sigma_i^2 + \sigma_j^2$) and the variance of the individual performances (i.e., $2\beta^2$). We have access to this message using the operator `+` of the class `Gaussian` to sum the agents' performances,

```
ta = p1 + p2; tb = p3 + p4
```

Code 16: Computing the team prior performance.

The next message, sent by the difference factor f_{d_1} to the difference variable d_1, is:

$$\begin{aligned} m_{f_d\to d}(d) &= \iint \delta(d = t_a - t_b)\mathcal{N}(t_a|\mu_a, \sigma_a^2)\mathcal{N}(t_b|\mu_b, \sigma_b^2)dt_a dt_b \\ &= \mathcal{N}(d|\underbrace{\mu_a - \mu_b}_{\substack{\text{Expected difference}\\ :\ \psi}}, \underbrace{\sigma_a^2 + \sigma_b^2}_{\substack{\text{Total}\\ \text{uncertainty}\ :\ \vartheta^2}}) = \mathcal{N}(d|\psi, \vartheta^2) \end{aligned} \qquad \texttt{(d = ta - tb)}$$

The prior difference of performance is a Gaussian distribution centered on the prior expected difference between teams $\psi = \mu_a - \mu_b$ with variance $\vartheta^2 = \sigma_a^2 + \sigma_b^2$ that includes the uncertainty

of both teams. We have access to this message using the operator `-` of the class `Gaussian` to get the difference of teams' performances,

```
d = ta - tb
```

Code 17: Computing the prior perfomance' difference

The last descending message, sent by the factor f_r to the variable r, allows to compute the evidence, i.e., the prior prediction of the observed result.

$$m_{f_r \to r}(r) = \int \mathbb{I}(d > 0)\mathcal{N}(d|\psi, \vartheta^2)dd = 1 - \Phi(0|\psi, \vartheta^2) \qquad \texttt{(evidence)}$$

We have access to this message by computing the cumulative value from 0 to ∞ of the team difference of performances distribution.

```
e = 1.0 - cdf(d, 0.0)
```
```
e = 1 - cdf(0,d.mu,d.sigma)
```
```
e = 1 - cdf(0,d@mu,d@sigma)
```

Code 18: Computing the prior prediction of the observed result (or evidence).

where `e` contains the value of equation `evidence`.

Ascending messages. Now, we examine the ascending messages. The result factor f_r sends to the difference variable d the first ascending message.

$$m_{f_r \to d}(d) = \mathbb{I}(d > 0) \tag{9}$$

This message contains the indicator function of the factor f_r that transmits the information of the observed result. The message sent by difference factor f_d to the winning team performance variables t_e is:

$$\begin{aligned} m_{f_d \to t_a}(t_a) &= \iint \delta(d = t_a - t_b)\mathbb{I}(d > 0)\mathcal{N}(t_b|\mu_b, \sigma_b^2)\, dd\, dt_b \\ &= \int \mathbb{I}(t_a > t_b)\mathcal{N}(t_b|\mu_b, \sigma_b^2)\, dt_b = 1 - \Phi(0|t_a - \mu_b, \sigma_b^2) = \Phi(t_a|\mu_b, \sigma_b^2) \end{aligned} \tag{10}$$

In this case, the previous upstream message is integrated with the downstream message from the other team. This message, parametrized at t_a, is the cumulative of the Gaussian distribution of the opposing team's performances from t_a to ∞, and encodes the likelihood of the winning team performance hypotheses. The message sent by team performance factor f_{t_a} to the variable of the individual performance p_1 is:

$$\begin{aligned} m_{f_{t_a} \to p_1}(p_1) &= \iint \delta(t_a = p_1 + p_2)\, N(p_2|\mu_2, \beta^2 + \sigma_2^2)\, \Phi(t_a|\mu_b, \sigma_b^2)\, dt_a dp_2 \\ &= \int \mathcal{N}(p_2|\mu_2, \beta^2 + \sigma_2^2)\, \Phi(p_1 + p_2|\mu_b, \sigma_b^2)\, dp_2 \\ &= 1 - \Phi(0|p_1 + \underbrace{\mu_2 - \mu_b}_{\mu_1 - \psi}, \underbrace{\beta^2 + \sigma_2^2 + \sigma_b^2}_{\vartheta^2 - (\sigma_1^2 + \beta^2)}) \end{aligned} \tag{11}$$

Again, the previous upstream message is integrated with a downstream message, the prior performance of their teammate. The message, parameterized at $p1$, encodes the likelihood of the individual performance hypotheses of the winning player. The last message, sent by

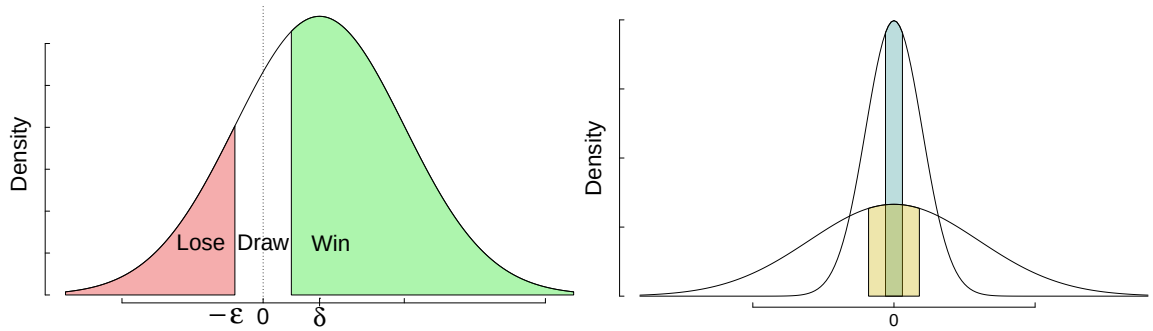
individual performance factor f_{p_1} to the skill variable s_1 is:

$$\begin{aligned} m_{f_{p_1} \to s_1}(s_1) &= \int N(p_1|s_1, \beta^2)\, \Phi(p_1|\mu_1 - \psi, \vartheta^2 - (\sigma_1^2 + \beta^2))\, dp_1 \\ &= 1 - \Phi(0|\underbrace{(s_1 + \mu_2) - (\mu_3 + \mu_4)}_{\substack{\text{Expected difference} \\ \text{parameterized in } s_1}}, \underbrace{\vartheta^2 - \sigma_1^2}_{\substack{\text{Total uncertainty} \\ \text{except the one of } s_1}}) \end{aligned} \tag{12}$$

This corresponds to the exact likelihood presented in Equation 3, which computes the prior probability of winning result if the player's actual skill was s_1.

3.5. A basic model for a draw

The model for a draw assumes that a tie occurs when the difference in performance does not exceed a threshold, $|t_a - t_b| \leq \varepsilon$. In Figure 10(a), we graphically display the probabilities of the three possible outcomes.



(a) The three possible results. (b) Constant draw probability.

Figure 10: Distribution of performance difference under the draw model. (a): example of the areas corresponding to the probability of losing, drawing and winning. (b) shows how the tie margin should be adapted to keep the tie probability constant when the uncertainty of the distribution changes.

This model requires establishing the threshold for considering a draw. Herbrich et al. (Herbrich *et al.* 2006) propose using the empirical frequency of ties to define it. However, this value depends on the actual skill difference, which is unknown. Assuming that we can define the "probability of a draw between teams with the same skill" then the threshold also depends on the number of players. Figure 10(b) shows that to keep the tie area constant, it is necessary to adapt the threshold according to the uncertainty. Since the observed results are independent of our beliefs, the only source of uncertainty is the variance of individual performance β. Then, we can define an expression that links the threshold ε with the probability of a tie:

$$\text{Draw probability} = \Phi\left(\frac{\varepsilon}{\sqrt{n_1 + n_2}\beta}\right) - \Phi\left(\frac{-\varepsilon}{\sqrt{n_1 + n_2}\beta}\right) \tag{13}$$

In the Code 19, we use the function `compute_margin()` to get the size of the margin.

```
na = length(team_a)
nb = length(team_b)
sd = sqrt(na + nb) * beta
```

```
na = len(team_a)
nb = len(team_b)
sd = math.sqrt(na + nb)* beta
```

```
na = length([a1, a2])
nb = length([a3, a4])
sd = sqrt(na + nb) * beta
```

```
p_draw = 0.25
margin = compute_margin(p_draw, sd)
```

Code 19: Computing the draw margin.

where the individual agents (`a1` to `a4`) were initialized in Code 2 and `beta` in Code 1.

3.6. Optimal approximation of the exact posterior

In Section 3.4, we presented the procedure to obtain the exact posterior. In this section, we show how to obtain the Gaussian distribution that best approximates the exact posterior, considering the possibility of ties. The packages solve it with the two lines of Code 20.

```
g = Game(teams, p_draw = 0.25)
post = posteriors(g)          post = g.posteriors()          post = posteriors(g)
```

Code 20: Computing the approximate posterior.

where the variable `teams` was initialized in Code 3. The need to approximate the posterior occurs because the distribution of the difference of performances is a truncated Gaussian.

$$p(d) = \begin{cases} \mathcal{N}(d|\psi, \vartheta^2)\mathbb{I}(-\varepsilon < d < \varepsilon) & \text{tie} \\ \mathcal{N}(d|\psi, \vartheta^2)\mathbb{I}(d > \varepsilon) & \text{not tie} \end{cases} \tag{14}$$

It is known that the exponential family, to which the Gaussian distribution belongs, minimizes the Kullback-Leibler divergence with respect to the true distribution p, $KL(p||q)$ when both have the same moments (Minka 2005). The expectation and variance of a truncated Gaussian $\mathcal{N}(x|\mu, \sigma^2)$ in the range from a to b are:

$$E(X|a < X < b) = \mu + \sigma \frac{\mathcal{N}(\alpha) - \mathcal{N}(\beta)}{\Phi(\beta) - \Phi(\alpha)} \tag{15}$$

$$V(X|a < X < b) = \sigma^2 \left(1 + \left(\frac{\alpha N(\alpha) - \beta N(\beta)}{\Phi(\beta) - \Phi(\alpha)}\right) - \left(\frac{N(\alpha) - N(\beta)}{\Phi(\beta) - \Phi(\alpha)}\right)^2\right) \tag{16}$$

where $\beta = \frac{b-\mu}{\sigma}$ and $\alpha = \frac{a-\mu}{\sigma}$. With a single-sided truncation, they can be simplified as:

$$E(X|a < X) = \mu + \sigma \frac{\mathcal{N}(\alpha)}{1 - \Phi(\alpha)} \quad , \quad V(X|a < X) = \sigma^2 \left(1 + \left(\frac{\alpha \mathcal{N}(\alpha)}{1 - \Phi(\alpha)}\right) - \left(\frac{\mathcal{N}(\alpha)}{1 - \Phi(\alpha)}\right)^2\right)$$

Then, the Gaussian that best approximates $p(d)$ is:

$$\widehat{p}(d) = \mathcal{N}(d|\widehat{\psi}, \widehat{\vartheta}^2) = \begin{cases} \mathcal{N}\Big(d \,|\, E(d| - \varepsilon < d < \varepsilon),\, V(d| - \varepsilon < d < \varepsilon)\Big) & \text{tie} \\ \mathcal{N}\Big(d \,|\, E(d|d > -\varepsilon),\, V(d|d > -\varepsilon)\Big) & \text{not tie} \end{cases} \quad (\texttt{approx()})$$

```
tie = true                    tie = True                    tie = T
d_approx = approx(d, margin, !tie)
```

Code 21: Computing the approximation of the performance difference.

where the difference distribution `d` was initialized in Code 17 and the variable `margin` in Code 19. Given $\widehat{p}(d)$, we can compute the approximate ascending message using the methods of the `Gaussian` class. To derive the first ascending message, recall that any marginal distribution can be calculated as the product of the messages received from all its neighboring factors.

$$\begin{aligned} m_{d\to f_d}(d) = \frac{p(d)}{m_{f_d\to d}(d)} &\approx \frac{\widehat{p}(d)}{m_{f_d\to d}(d)} \\ &= \frac{\mathcal{N}(d\,|\,\widehat{\psi},\,\widehat{\vartheta}^{\,2})}{\mathcal{N}(d|\psi,\vartheta^2)} \propto N(d,\psi_{\div},\vartheta^2_{\div}) \end{aligned} \qquad \texttt{(approx\_lh\_d)}$$

where $m_{f_r\to d}(d) = m_{d\to f_d}(d)$ holds due to the factorization of the model. We have access to this message when we use the `/` operator of the `Gaussian` class to divide the distributions:

```
approx_lh_d = d_approx / d
```

Code 22: Computing the first approximate message.

All these approximate messages can be interpreted as likelihoods because they transmit the information of the observed result. The approximate message sent by difference factor f_d to the winning team performance variables t_a is:

$$\begin{aligned} \widehat{m}_{f_d\to t_a}(t_a) &= \iint \delta(d = t_a - t_b)\mathcal{N}(d_1|\psi_{\div},\vartheta^2_{\div})\mathcal{N}(t_b|\mu_b,\sigma_b^2)\,dd\,dt_b \\ &= \int \mathcal{N}(t_a - t_b|\psi_{\div},\vartheta^2_{\div})\mathcal{N}(t_b|\mu_b,\sigma_b^2)\,d_{t_b} = \mathcal{N}(t_a\,|\,\mu_b+\psi_{\div}\,,\,\vartheta^2_{\div}+\sigma_b^2) \end{aligned} \tag{17}$$

The approximate message sent by team performance factor f_{t_a} to the winning individual performance variables p_1 is:

$$\begin{aligned} \widehat{m}_{f_{t_a}\to p_1}(p_1) &= \iint \delta(t_a = p_1 + p_2)\mathcal{N}(t_a\,|\,\mu_b+\psi_{\div}\,,\,\vartheta^2_{\div}+\sigma_b^2)\mathcal{N}(p_2|\mu_2,\sigma_2^2+\beta^2)\,dt_a d_{p_2} \\ &= \int \mathcal{N}(p_1+p_2\,|\,\mu_b+\psi_{\div}\,,\,\vartheta^2_{\div}+\sigma_b^2)\mathcal{N}(p_2|\mu_2,\sigma_2^2+\beta^2)\,d_{p_2} \\ &= \mathcal{N}(p_1\mid \underbrace{\mu_b-\mu_2}_{\mu_1-\psi}+\psi_{\div}\,,\,\vartheta^2_{\div}+\underbrace{\sigma_b^2+\sigma_2^2+\beta^2}_{\vartheta^2-(\sigma_1^2+\beta^2)}) \end{aligned} \tag{18}$$

The approximate message sent by the individual performance factor f_{p_1} to the winning skill variables s_1 is,

$$\begin{aligned} \widehat{m}_{f_{p_1}\to s_1}(s_1) &= \int \mathcal{N}(p_1|s_1,\beta^2)\mathcal{N}(p_1|\mu_1-\psi+\psi_{\div},\vartheta^2_{\div}+\vartheta^2-\sigma_1^2-\beta^2)dp_1 \\ &= \mathcal{N}(s_1|\mu_1-\psi+\psi_{\div},\vartheta^2_{\div}+\vartheta^2-\sigma_1^2) \end{aligned} \tag{19}$$

Finally, the approximate proportional posterior of the variable s_1 is obtained by multiplying the messages received from its neighboring factors.

$$\widehat{p}(s_1,r) = \mathcal{N}(s_1|\mu_1,\sigma_1^2)\mathcal{N}(s_1|\mu_1-\psi+\psi_{\div},\vartheta^2_{\div}+\vartheta^2-\sigma_1^2) \qquad \texttt{(posterior)}$$

We have access to the normalized posterior using the operator `*` of the `Gaussian` class, or getting the first element of the list `post` computed in Code 20.

```
mu = a1.prior.mu
sigma2 = a1.prior.sigma^2
phi = d.mu
v2 = d.sigma^2
phi_div = approx_lh_d.mu
v2_div = approx_lh_d.sigma^2
prior = a1.prior
posterior = post[1][1]
```

```
mu = a1.prior.mu
sigma2 = a1.prior.sigma**2
phi = d.mu
v = d.sigma**2
phi_div = approx_lh_d.mu
v2_div = approx_lh_d.sigma**2
prior = a1.prior
posterior = post[0][0]
```

```
mu = a1@prior@mu
sigma2 = a1@prior@sigma^2
phi = d@mu
v2 = d@sigma^2
phi_div = approx_lh_d@mu
v2_div = approx_lh_d@sigma^2
prior = a1@prior
posterior = post[[1]][[1]]
```

```
print( prior * Gaussian(mu - phi + phi_div, sqrt(v2 + v2_div - sigma2)) )
> Gaussian(mu = 2.461, sigma = 5.507)
print(posterior)
> Gaussian(mu = 2.461, sigma = 5.507)
```

Code 23: Accessing the approximate posterior.

with `a1`, `d`, and `approx_lh_d` computed in codes 2, 17, and 22. The obtained posterior is the same as the one returned by the `Gaussian` class.

3.7. Multiple teams

The interface of the packages does not show any difference between two-team and multiple-team games, as can be seen in codes 3 and 6. However, in cases where there are more than two teams, we need to implement an iterative algorithm due to a mutual dependency between results. Thanks to the transitivity of results, if k teams participate in an event, it is sufficient to evaluate $k-1$ differences of performance d_i between teams in consecutive positions. For this purpose, we define an ordered list o containing the teams according to the observed result. The winning team is the first element (i.e., o_1), and o_i represents the team in position i. Figure 11 shows the factorization of the general TrueSkill model. The general idea is to

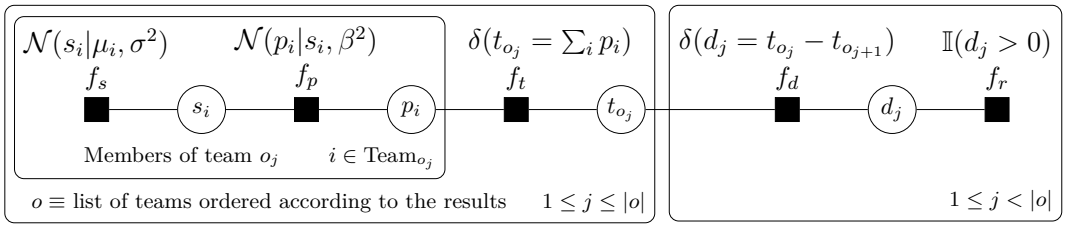


Figure 11: General factor graph of the team model. The subscripts appearing at the bottom right of the plates indicate replication. The j subscript of the left plate opens the k team performances, and i subscript of the inner plate displays their players. The subscript j of the right plate opens the $k-1$ comparisons between consecutive teams.

repeatedly update forward and backward all messages in the shortest path between any two marginals $p(d_j)$ until convergence.

Let's analyze the algorithm involved in solving a game with three teams. Instead of using the message notation proposed by the sum-product algorithm, we name the messages following the criteria shown in Figure 12.

Before we start, we set the landscape: we compute the teams' prior performance using the function `performance()`. Then, we initialize the undefined messages using a neutral form, such as a Gaussian distribution with infinite variance. Lastly, we compute the margins for each comparison d_j.

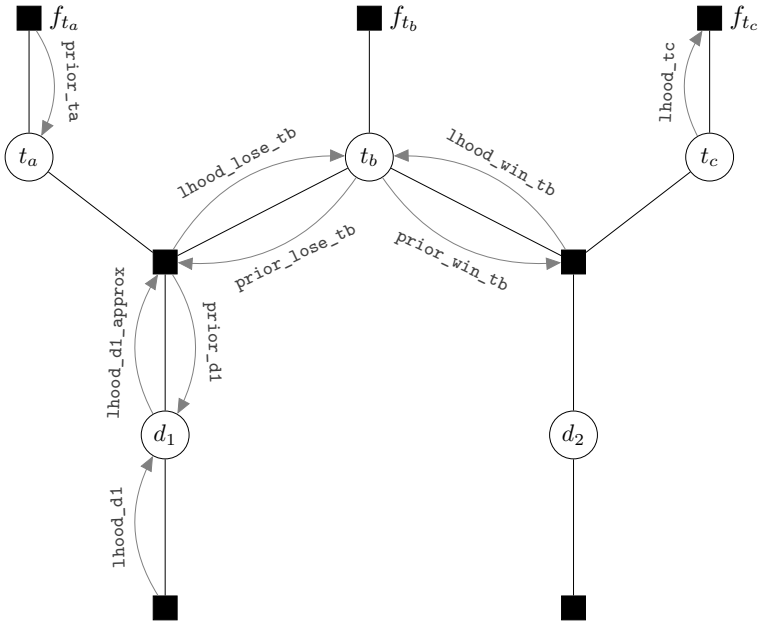


Figure 12: Factorization of a game with three teams. We only show factors from the teams to the results. The names describe the iterative procedure known as *loopy belief propagation*.

```
team_a = [a1]                    team_a = [a1]                    team_a = c(a1)
team_b = [a2, a3]                team_b = [a2, a3]                team_b = c(a2, a3)
team_c = [a4]                    team_c = [a4]                    team_c = c(a4)
prior_ta= performance(team_a); prior_tb= performance(team_b); prior_tc= performance(team_c)
N_inf = Gaussian(0., Inf)        N_inf = Gaussian(0, inf)         N_inf = Gaussian(0, Inf)
lhood_win_ta = N_inf; lhood_lose_tb = N_inf; lhood_win_tb = N_inf; lhood_lose_tc = N_inf
margin = compute_margin(p_draw, sqrt(3)*beta)
```

Code 24: Setting the landscape.

where the teams are defined in Code 3. Since there are three players in both comparisons, we adjust both margins with the same size. We start the iterative process by approximating the distribution d_1. Note that any marginal distribution is the product of the received messages from the neighbors.

```
prior_lose_tb = prior_tb * lhood_win_tb
prior_d1 = prior_ta - prior_lose_tb
lhood_d1_approx = approx(prior_d1, margin, !tie) / prior_d1
```

Code 25: Approximating the distribution d_1 with the last message sent by t_b.

In the first line, we initialize the message that the variable t_b sends to the factor node f_{d_1}: the product of the messages received from behind. Note that in the first loop, it is equivalent to `prior_tb` because the variable `lhood_win_tb` has a neutral value. In the second line, we compute the message sent by the factor f_{d_1} to the variable d_1. In the last line, we compute the approximate message sent by the variable d_1 to the factor f_{d_1}. This allows us to update the message received by the variable t_b from the factor f_{d_1}.

```
lhood_lose_tb = prior_ta - lhood_d1_approx
```

Code 26: Updating the messages of t_b with the last approximation of d_1.

Here we compute the message sent by the factor f_{d_1} to the variable t_b. Then we approximate the distribution d_2 using the updated messages.

```
prior_win_tb = prior_tb * lhood_lose_tb
prior_d2 = prior_win_tb - prior_tc
lhood_d2_approx = approx(prior_d2, margin, tie) / prior_d2
```

Code 27: Approximating de distribution d_2 with the last messages sent by t_b.

In the first line, we initialize the message sent by the variable t_b to the factor node f_{d_2}. In the second line, we compute the message sent by the factor f_{d_2} to the variable d_2. In the last line, we compute the approximate message sent by the variable d_2 to the factor f_{d_2}. This allows us to update the message received by the variable t_b from the factor f_{d_2}.

```
lhood_win_tb = prior_lose_tc + lhood_d2_approx
```

Code 28: Updating the t_b distribution with the last approximation of d_2.

Here we compute the message sent by the factor f_{d_2} to the variable t_b. The codes 25 to 28 must be included in a cycle until reaching convergence. Once finished, we send the upstream messages to the teams at both ends.

```
lhood_win_ta = posterior_lose_tb + lhood_d1_approx
lhood_lose_tc = posterior_win_tb - lhood_d2_approx
```

Code 29: Computing the messages received by f_t factors of the winning and losing teams.

These are the messages sent by the factors f_{d_1} and f_{d_2} to the variables t_a and t_c respectively. Finally, we compute the likelihood of each team.

```
lhood_ta = lhood_win_ta
lhood_tb = lhood_lose_tb * lhood_win_tb
lhood_tc = lhood_lose_tc
```

Code 30: Ascending messages sent by factors f_t to the variables t.

The following ascending messages are obtained as described in Section 3.6.

3.8. Information propagation in the history class

This section explains how the information provided by the data propagates throughout the Bayesian network containing the history of events. TrueSkill Through Time creates a unique causal model in which all historical activities are linked. The connectivity between events is generated by the assumption that a player's skill at time t depends on his skill at an earlier time $t-1$. The model states that within each time step t (e.g., day, week, month, year) each agent i participates in all events. Figure 13 shows a part of the graphical factorization of the event history: the neighboring nodes to a skill variable and the messages between these nodes. By the sum-product algorithm, we know that the marginal distribution of any variable is the product of the messages it receives from its neighbors. Using the names presented in Figure 13, we know that the posterior distribution of an agent's skill i at time t is:

$$\text{posterior}(s_t) = \textit{forwardPrior}(s_t) \cdot \textit{backwardPrior}(s_t) \cdot \prod_{k=1}^{K_t} \textit{likelihood}_k(s_t) \tag{20}$$

where K_t is the number of events in which the agent participates in time step t. The *likelihood* messages are the likelihoods of the events described in sections 3.6 and 3.7, and the

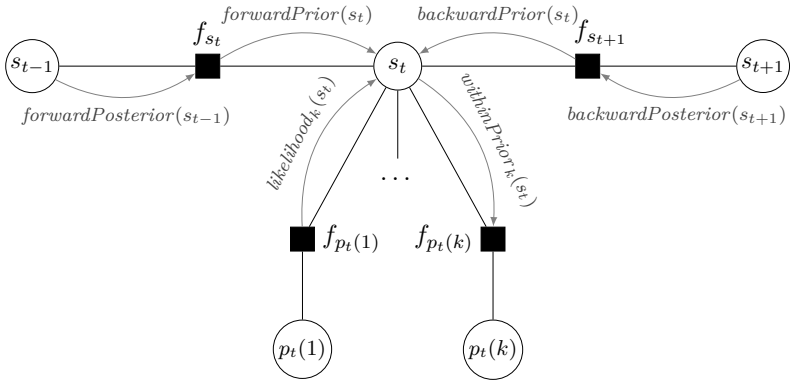


Figure 13: Nodes of the factor graph of a history of events that are neighbors to an agent's skill variable at time step t. The variables $p_t(j)$ represents the performance p that the agent had in their j-th game within the time step t. The names of the arrows represent the messages computed by the sum-product algorithm.

forwardPrior and *backwardPrior* messages are the neighboring skill estimates, to which some uncertainty γ is added in each time step.

$$forwardPrior(s_t) = \mathcal{N}(s_t \,|\, \mu_f, \sigma_f^2 + \gamma^2) \quad , \quad backwardPrior(s_t) = \mathcal{N}(s_t \,|\, \mu_b, \sigma_b^2 + \gamma^2) \tag{21}$$

where $forwardPosterior(s_{t-1}) = \mathcal{N}(s_{t-1} \,|\, \mu_f, \sigma_f^2)$ and $backwardPosterior(s_{t+1}) = \mathcal{N}(s_{t-1} \,|\, \mu_b, \sigma_b^2)$. The message $forwardPrior(s_t)$ is the $forwardPosterior(s_{t-1})$ after the dynamic uncertainty of time t is added, f_{s_t}. These messages match the prior and posterior definitions of the basic TrueSkill model. The coincidence arises from applying the sum-product algorithm. But from its application also emerges the $backwardPrior(s_t)$ message, which is the $backwardPosterior(s_{t+1})$ after the dynamic uncertainty of time $t+1$ is added, $f_{s_{t+1}}$.

The amount of dynamic uncertainty added to the *forwardPrior* and *backwardPrior* messages in Equation 21 is the same. However, sometimes we would like the dynamic uncertainty to depend on the temporal distance between adjacent skill variables. Our packages support both options. When we initialize the class `History` without specifying the time of the events, as we did in Codes 7 and 11, the dynamic factors always incorporate a single γ^2 between variables that are adjacent according to the composition of the events. When we specify the time of the events, as performed in Code 12, the adjacency depends on those times, and the dynamic uncertainty depends on the elapsed time between two adjacent variables s_t and s_{t-1}.

$$f_{s_t} = \mathcal{N}(s_t | s_{t-1}, \text{elapsedTime}(s_t, s_{t-1}) \cdot \gamma^2) \tag{22}$$

The priors, used to compute the likelihoods of the event, are the descending messages *withinPrior*. Following the sum-product algorithm, we know that the messages sent by the variable are the product of the messages they received from behind, so the message *withinPrior* is:

$$\begin{aligned} withinPrior_q(s_t) &= forwardPrior(s_t) \cdot backwardPrior(s_t) \cdot \prod_{\substack{k=1 \\ q \neq k}}^{K_t} likelihood_k(s_t) \\ &= \frac{\text{posterior}(s_t)}{likelihood_q(s_t)} \end{aligned} \tag{23}$$

The *withinPrior* contains the information of the posterior(s_t), except for the event's likelihood$_q$ for which it is prior. To solve the mutual dependency between likelihoods, we repeatedly update the forward and backward messages until reaching convergence. At each forward pass, we keep track of the last forward message of each agent *forwardPosterior*, and at each backward pass, we keep track of the last backward message of each agent, *backwardPosterior*.

$$\begin{aligned} \textit{forwardPosterior}(s_t) &= \frac{\text{posterior}(s_t)}{\textit{backwardPrior}(s_t)} = \textit{forwardPrior}(s_t) \cdot \prod_{k=1}^{K_t} \textit{likelihood}_k(s_t) \\ \textit{backwardPosterior}(s_t) &= \frac{\text{posterior}(s_t)}{\textit{forwardPrior}(s_t)} = \textit{backwardPrior}(s_t) \cdot \prod_{k=1}^{K_t} \textit{likelihood}_k(s_t) \end{aligned} \tag{24}$$

The messages that are not yet defined, for example, the $\textit{backwardPrior}(s_t)$ in the first forward pass, should be replaced by a neutral form such as the Gaussian distribution with infinite variance. This algorithm requires only a few linear iterations on the data to converge and allows scaling to millions of observations in a few seconds.

3.9. Predictive performance

This section evaluates the predictive performance and efficiency of our TTT model implementation. We compare its predictive performance to other models. In contrast to commonly used skill estimators that propagate information in only one direction, using the last posterior as the prior probability to the next event (*filtering* approach), the TrueSkill Through Time model (Dangauthier *et al.* 2007) propagates all historical information throughout the entire network of events, providing estimates with low uncertainty at any given time, offering reliable initial skill estimates, and guaranteeing comparability between estimates which are distant in time and space (*smoothing* approach). Other smoother approaches were proposed by Glickman (1999) (Smooth Glicko), Coulom (2008) (Whole History Rating), and Maystre *et al.* (2019) (KickScore).

To verify the predictive performance, Table 1 compares our implementation of the TTT algorithm against KickScore, TrueSkill, Elo, and a "Constant" model in which skills do not change over time. By definition, the probability of a model given the data is

$$P(\text{Model}|\text{Data}) = \frac{P(\text{Data}|\text{Model})P(\text{Model})}{P(\text{Data})} \tag{25}$$

When we do not have access to all models, we cannot compute the probability $P(\text{Model}|\text{Data})$, but we can compare models.

$$\frac{P(\text{Model}_i|\text{Data})}{P(\text{Model}_j|\text{Data})} = \frac{P(\text{Data}|\text{Model}_i)P(\text{Model}_i)}{P(\text{Data}|\text{Model}_j)P(\text{Model}_j)} \overset{\text{if } *}{=} \frac{P(\text{Data}|\text{Model}_i)}{P(\text{Data}|\text{Model}_j)} \tag{26}$$

This expression is known as the *Bayes factor*, usually expressed using a logarithmic scale to report the difference in orders of magnitude ($\log_2$BF). When we have no prior preference over any model (if $*$), we only need to compare the prior predictions of the models.

$$P(\text{Data}|\text{Model}) = P(d_1|\text{Model})P(d_2|d_1, \text{Model}) \dots P(d_n|d_1, \dots, d_{n-1}, \text{Model}) \tag{27}$$

where each element of the product is a prediction of the following data point based on the previous data set and the model. In this work we report the geometric mean (GM) because

it can be interpreted as the growth rate of the joint prediction,

$$\text{Geometric Mean}(P(\text{Data}|\text{Model})) = P(\text{Data}|\text{Model})^{1/n} \tag{28}$$

where n is the total amount of data.

We follow the methodology employed by Maystre *et al.* (2019), analyzing the four databases on which they reported their prior predictions: ATP (tennis), NBA (basketball), FIFA (football), and FIDE (chess). This methodology splits the data into two subsets: the initial 70 % used to train the hyperparameters and the last 30 % to evaluate the models. To predict the outcome of one observation at time t, we use all the data up to the day preceding t (in both training and test sets). Table 1 summarizes the model comparisons in each of the four databases. The

Dataset Test size	Constant†		Elo†		TrueSkill		KickScore†		TTT	
	GM	$\log_2$BF	GM	$\log_2$BF	GM	$\log_2$BF	GM	$\log_2$BF	GM	LOOCV
Tennis 186 361	0.5593	7910	0.5695	3051	0.5722	1780	0.5758	93	**0.5760**	0.5908
Basketball 20 300	0.5006	1771	0.5305	72	0.5316	11	**0.5328**	-55	0.5318	0.5382
Chess 92 004	0.3570	520	0.3552	1190	0.3580	148	**0.3584**	0	**0.3584**	0.3641
Football 5759	0.3949	30	0.3867	204	0.3921	89	**0.3961**	4	**0.3963**	0.3974

Table 1: Model comparison in four databases. The GM columns report the geometric mean of the prior predictions of the models, Equation 28. The $\log_2$BF columns report the Bayes Factor (Equation 26) between TTT and the other models in the logarithmic scale. For reference, the LOOCV column reports the geometric mean of the predictions using all historical data, $p(d_i|d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_n, M)$. The geometric mean of the models marked with the symbol † was presented in Maystre *et al.* (2019).

TTT model outperforms KickScore in the tennis database by 93 orders of magnitude, while KickScore outperforms TTT in the chess database by 55 orders of magnitude. We consider that the models are tied in the chess and football databases as long as the difference between them is less than ten orders of magnitude. Unifying the four databases into one, the TTT model outperforms KickScore by more than 40 orders of magnitude.

To compute the predictions, Maystre *et al.* (2019) use an HPC cluster to create, in parallel, a different model per day, each requiring about 100 iterations to reach convergence. Instead, it was sufficient for us to use a desktop computer to create a single model adding data one day at a time and performing a single iteration at each step. The model selection was made by optimizing the hyperparameters, but a fully Bayesian model selection should compute predictions by integrating the entire hyperparameter space. This procedure usually penalizes complex models when the search in the hyperparameter space is unnecessary. With no more than three intuitive hyperparameters (prior uncertainty, dynamic uncertainty, and draw probability), TTT achieves similar or even better results than KickScore. Moreover, TTT can compute these results more efficiently because KickScore must optimize on a much larger hyperparameter space, which includes the definition of a kernel and then the hyperparameters specific to that kernel.

When the goal is to estimate the learning curves over time as accurately as possible, instead of forcing the model to make the inference using only past information (as we did to compare the performance of the models), it is convenient to make the inference using all the historical

data. This is useful because probabilistic models can use information from the future to infer past events, something we intuitively do when we suspect that outstanding athletes were also skilled some time before they became famous. The LOOCV column of Table 1 reports the geometric mean of predictions that use all historical information except the event under consideration, $p(d_i|d_1,\ldots,d_{i-1},d_{i+1},\ldots,d_n,M)$. Although we do not use these offline predictions to compare models (Bayes Factor), it is interesting to note that incorporating all historical information into the TTT model improves its estimates by hundreds of orders of magnitude.

Because skill estimation is a sensitive topic for individuals, we present an explanation that is as intuitive and complete as possible in the following sections. The communities under evaluation in the video game industry and educational systems need to know how the algorithm that measures their ability works. This paper offers a complete and accessible scientific report to the general public jointly with the software tool.

4. Summary and discussion

The main contribution of this work is the implementation of one of the most important models in the video game industry, which was not yet available in the programming languages with larger communities, such as Julia, Python, and R. The fact that such an important algorithm has been absent in these programming languages possibly has been the consequence of the lack of documentation that would allow software developers to understand all the theoretical aspects required for its implementation. This information is not present either in the original articles (Herbrich *et al.* 2006; Dangauthier *et al.* 2007) or in available informal technical reports such as the one developed by Mosser (2011). For this reason, we have devoted the methodology section to documenting all the theoretical arguments step by step to make it comprehensible to anyone with basic knowledge of probability. But in addition, as skill estimation is a sensitive topic, we explain in the introductory section the details of this skill estimation model, contextualizing the importance and benefits of the skill model we propose.

Most online game servers and scientific articles still use some skill estimates based on a filtering approach. These models share a major shortcoming: they propagate historical information in only one direction through the system, from the past to the future. This strategy produces poor initial skill estimates and also prevents the comparison of estimates distant in time and space. The strict application of the rules of probability forces us to perform inference using all available information, including that of the future, something we intuitively do when we conclude that famous athletes were also skilled before becoming famous. For this reason, the TTT model, by propagating all historical information through the entire causal network, provides estimates with low uncertainty throughout the time series, allowing reliable initial skill estimates and ensuring historical comparability. In this way, the prior predictions are several orders of magnitude more accurate than those obtained with other models (as shown in Table 1). With no more than three intuitive hyperparameters (a priori uncertainty, dynamic uncertainty and tie probability), the TTT model achieves similar or even better results than more complex models such as KickScore.

Propagating the historical information correctly throughout the causal network of events produces a mutual dependency between the estimates that forces the implementation of an iterative algorithm. Based on the analytical approximation methods and message-passing

algorithms, the TTT model can be solved efficiently using any low-end computer, even in causal networks with millions of nodes and irregular structures. We have shown that this procedure requires a few iterations to converge even on large databases such as the Association of Professional Tennis. In the next section, computational details, we study the execution times of the three packages presented in this paper. Our Python package solves individual events ten times faster than the original **trueskill** `0.4.5` package (Lee 2012). In turn, the Julia package solves event sequences ten times faster than our Python package. In contrast, our R package is slower than the others, including the original **trueskill** 0.4.5 package.

We hope that this first open implementation in Julia, Python, and R, of one of the most important skill estimation models, accompanied by an in-depth scientific explanation, will provide the conditions to generate a continuous collective improvement process. For example, our implementation does not yet have the extension made by Guo *et al.* (2012), in which the observable data are no longer the ordinal values of the results (lose/tie/win) but the number of points in favor and against each team. One of the current model challenges is distinguishing the individuals' ability in the same team using the team's outcome as the only observable. This could be solved by incorporating other observables, specific to individuals. This is the strategy used by the TrueSkill 2 by Minka, Cleven, and Zaykov (2018), which incorporates additional information readily available in online shooters, such as the number of individual achievements, their tendency to quit, and their skill in other gaming modes. TrueSkill 2 has been developed as an extension of the TrueSkill Through Time model, significantly improving its predictive performance. Now that we offer the first implementation of TrueSkill Through Time in Julia, Python, and R, the challenge is to incorporate all the extensions available in the literature and develop an application with as much flexibility as possible, allowing us to address the broadest possible variety of situations.

Computational details

This section reports the execution times of the examples presented in Section 2. We use a low-end workstation with 4 GB of RAM and an AMD processor (A4-6210, 1.00 GHz, cache 2048 kB). We analyze the execution times using the command `@timed` in Julia, the command `timeit` in Python, and the package `microbenchmark` (Mersmann, Beleites, Hurling, Friedman, and Ulrich 2011) in R.

Single event: Section 2.1 presents two examples: a game with two teams in Code 3 and a three-teams game in Code 6. Here we evaluate the efficiency during the initialization of the `Game` class and the application of method `posteriors()`. The original **trueskill** `0.4.5` package solves these two steps using the `rate()` function. Table 2(a) shows the runtime values. Our Python and Julia packages are 10 and 20 times faster than the original **trueskill** package, while the R package is three times slower. Table 2(b) presents the execution times for the three-team game example. Whenever there are more than two teams in the game, it is necessary to run an iterative algorithm that increases the time required to compute the posteriors. Again, our Python and Julia packages are faster than the original **trueskill** package, while the R package is slower.

Version	Runtime	Speedup
trueskill 0.4.5	1.45 ms	1.0X
R 3.4.4	4.35 ms	0.33X
Python 3.6.9	0.14 ms	10.4X
Julia 1.5.0	0.064 ms	22.7X

(a) : Two-teams game

Version	Runtime	Speedup
trueskill 0.4.5	2.45 ms	1.0X
R 3.4.4	31.3 ms	0.078X
Python 3.6.9	0.93 ms	2.63X
Julia 1.5.0	0.096 ms	25.5X

(b) : Three-teams game

Table 2: Execution time of the `Game` class initialization and call to the `posteriors()` method. The reference is the original **trueskill** 0.4.5 package running with Python 3.6.9.

A sequence of events: Section 2.2 presents the initialization of the class `History` and the call to the method `convergence()` in codes 7 and 9, respectively. Although the initialization of the `History` class computes the original **trueskill** 0.4.5 package estimates, we do not compare them because this package does not have a function to handle sequences of events automatically. Table 3 presents the execution times for the initialization of the `History` class and call of a single `convergence()` iteration. The initialization of the `History` class

Version	`History`	`convergence()`
R 3.4.4	74.4 ms	46.8 ms
Python 3.6.9	1.09 ms	1.88 ms
Julia 1.5.0	0.31 ms	0.58 ms

Table 3: Three events of two teams: execution time for the initialization of the `History` class and a single iteration of the method `convergence()`.

includes the creation of the Bayesian network and one sweep through all the events, while the convergence performs two passes through the sequence of events, one backward and one forward. The initialization in the Python package is even faster than the computation of a single isolated event using the original **trueskill** package. The Julia package increases the time difference from our Python package. And the R package, while maintaining the time difference from Python during convergence, has an additional delay during initialization due to the creation of classes by reference.

Skill evolution: Section 2.3 analyzes the skill evolution of a player playing against 1000 different opponents. Table 4 presents the execution times to initialize the class `History` and perform a single convergence iteration. In this case, the Julia package is ten times faster than

Language	`History`	`convergence()`
R 3.4.4	31 000 ms	26 000 ms
Python 3.6.9	380.2 ms	876.3 ms
Julia 1.5.0	38.1 ms	75.7 ms

Table 4: Initialization of the `History` class and a single iteration of the `convergence()` method in a sequence of 1000 games of a player against different opponents.

the Python package in the initialization and singe iteration. This suggests that having more workload improves the relative performance of the Julia package. In contrast to the behavior

of the Julia package and Python packages, the R version does not increase its execution time during convergence, highlighting the relative impact of class initialization in this language.

The history of the ATP: Section 2.4 analyzes a historical database with 447 000 events. In this real-life scenario, we perform the analysis using an additional workstation to show the impact of different hardware on the execution time. We include the same workstation as before (called **A** in this section) and a new workstation (called **B**) with 16 GB of RAM and an Intel processor (i5-3330, 3.00 GHz, total cache 6144 kB). Table 5 presents the total runtime for the `History` class initialization and the call to the `convergence()` method. The initialization

Workstation	Version	Runtime
A	Python 3.6.9	4498.8 s
B	Python 3.6.8	1368.6 s
A	Julia 1.5.0	387.5 s
B	Julia 1.5.3	138.5 s

Table 5: Initialization of the class `History` and ten iterations of the method `convergence()` of the ATP database.

and the ten convergence iterations evaluate approximately 447000×21 events: one sweep over all events corresponding to computing TrueSkill estimates during initialization, and two passes over all items per iteration (backward and forward). Given the execution times of a single event included in Table 2(a), the minimum execution time it would take to compute the TrueSkill Through Time estimates using the original **trueskill** 0.4.5 package could not be less than 13 611 s on computer A, three times longer than it takes our Python package to perform the entire computation. Our Julia package takes less time than expected if we compare it to the values reported in Table 2(a). This improvement is based on the optimizations performed automatically by the Julia runtime support.

Supplemental material

The source codes for the TrueSkill Through Time packages can be found at:

- `github.com/glandfried/TrueSkillThroughTime.jl` (Julia)
- `github.com/glandfried/TrueSkillThroughTime.py` (Python)
- `github.com/glandfried/TrueSkillThroughTime.R` (R)

Acknowledgments

Special thanks to the authors of the original paper, Ralf Herbrich, Tom Minka, Thore Graepel, and Pierre Dangauthier (Herbrich *et al.* 2006; Dangauthier *et al.* 2007), to Heungsub Lee for publishing the TrueSkill model in Python (Lee 2012), and Lucas Maystre for personally assisting us with the model comparison (Maystre *et al.* 2019). We also thank Matias Mazzanti for constructive discussions during the implementation of our package. This work is partially

supported by grants from Universidad de Buenos Aires (UBA) UBACYT 13320150100020CO, and Agencia Nacional de Promoción Científica y Tecnológica (ANPCYT) PICT-2015-2761.

References

Bradley RA, Terry ME (1952). "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika*, **39**(3/4), 324–345. ISSN 0006-3444. doi:10.2307/2334029.

Coulom R (2008). "Whole-History Rating: A Bayesian Rating System for Players of Time-Varying Strength." In HJ van den Herik, X Xu, Z Ma, MHM Winands (eds.), *Proc of the 6th Int Conf on Computers and Games*, pp. 113–124. Springer, Berlin, Heidelberg. ISBN 978-3-540-87607-6. doi:10.1007/978-3-540-87608-3_11.

Dangauthier P, Herbrich R, Minka T, Graepel T (2007). "TrueSkill Through Time: Revisiting the History of Chess." In JC Platt, D Koller, Y Singer, ST Roweis (eds.), *Proc of the Twenty-First Annual Conf on Neural Information Processing Systems*, pp. 337–344. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2007/file/9f53d83ec0691550f7d2507d57f4f5a2-Paper.pdf.

Elo AE (2008). *The Rating of Chess Players, Past and Present*. Ishi Press. ISBN 978-0-923891-27-7. Originally published in 1978.

Glickman ME (1999). "Parameter Estimation in Large Dynamic Paired Comparison Experiments." *J Roy Stat Soc C-App*, **48**(3), 377–394. ISSN 1467-9876. doi:10.1111/1467-9876.00159.

Glickman ME (2001). "Dynamic Paired Comparison Models with Stochastic Variances." *J Appl Stat*, **28**(6), 673–689. doi:10.1080/02664760120059219.

Guo S, Sanner S, Graepel T, Buntine W (2012). "Score-Based Bayesian Skill Learning." In PA Flach, T De Bie, N Cristianini (eds.), *Proc of the Joint European Conf on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, volume 7523 of *Lecture Notes in Computer Science*, pp. 106–121. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-642-33460-3. doi:10.1007/978-3-642-33460-3_12.

Herbrich R, Minka R, Graepel T (2006). "TrueSkill™: A Bayesian Skill Rating System." In B Schölkopf, J Platt, T Hoffman (eds.), *Proc of the Conf in Advances in Neural Information Processing Systems*, pp. 569–576. MIT Press, Cambridge, MA. ISBN 978-0262256919. URL https://proceedings.neurips.cc/paper/2006/file/f44ee263952e65b3610b8ba51229d1f9-Paper.pdf.

Kschischang FR, Frey BJ, Loeliger HA (2001). "Factor Graphs and the Sum-Product Algorithm." *IEEE T Inform Theory*, **47**(2), 498–519. ISSN 0018-9448. doi:10.1109/18.910572.

Landfried G (2021). "TrueSkill Through Time: the Julia, Python and R packages." https://github.com/glandfried/TrueSkillThroughTime.

Lee H (2012). "TrueSkill. The Video Game Rating System." Last checked on July 13, 2023, URL https://github.com/sublee/trueskill/.

Maystre L, Kristof V, Grossglauser M (2019). "Pairwise Comparisons with Flexible Time-Dynamics." In *Proc of the 25th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining*, pp. 1236–1246. ACM, New York, NY, USA. ISBN 9781450362016. doi:10.1145/3292500.3330831.

Mersmann O, Beleites C, Hurling R, Friedman A, Ulrich JM (2011). ***microbenchmark**: Accurate Timing Functions*. R package version 1.4.9, URL https://cran.r-project.org/package=microbenchmark.

Minka T (2005). "Divergence Measures and Message Passing." *Technical Report MSR-TR-2005-173*, Microsoft Research. URL https://www.microsoft.com/en-us/research/publication/divergence-measures-and-message-passing/.

Minka T, Cleven R, Zaykov Y (2018). "TrueSkill 2: An Improved Bayesian Skill Rating System." *Technical Report MSR-TR-2018-8*, Microsoft. URL https://www.microsoft.com/en-us/research/publication/trueskill-2-improved-bayesian-skill-rating-system/.

Mosser J (2011). "The Math Behind TrueSkill." *Technical report*. Last checked on July 13, 2023, URL http://www.moserware.com/assets/computing-your-skill/The%20Math%20Behind%20TrueSkill.pdf.

Mosteller F (1951a). "Remarks on the Method of Paired Comparisons: I. The Least Squares Solution assuming Equal Standard Deviations and Equal Correlations." *Psychometrika*, **16**(1), 3–9. ISSN 1860-0980. doi:10.1007/BF02313422.

Mosteller F (1951b). "Remarks on the method of paired comparisons: II. The Effect of an Aberrant Standard Deviation when Equal Standard Deviations and Equal Correlations are assumed." *Psychometrika*, **16**(2), 203–206. ISSN 1860-0980. doi:10.1007/BF02289115.

Mosteller F (1951c). "Remarks on the Method of Paired Comparisons: III. A Test of Significance for Paired Comparisons when Equal Standard Deviations and Equal Correlations are assumed." *Psychometrika*, **16**(2), 207–218. ISSN 1860-0980. doi:10.1007/BF02289116.

Thurstone LL (1927). "Psychophysical Analysis." *Am J Psychol*, **38**(3), 368–389. ISSN 0002-9556. doi:10.2307/1415006.

Zermelo E (2013). *Ernst Zermelo - Collected Works Volume II*, volume 23 of *Schriften der Mathematisch-naturwissenschaftlichen Klasse*, chapter The Calculation of the Results of a Tournament as a Maximum Problem in the Calculus of Probability, pp. 616–671. Springer-Verlag, Berlin Heidelberg. ISBN 978-3-540-70856-8. doi:10.1007/978-3-540-70856-8. Translation of 1929 original paper.

5. Appendix

5.1. Skill evolution

We attach the Julia and R codes that solve the example presented in the Section 2.3 about estimation the skill evolution of a new player.

```
using Random; Random.seed!(999); N = 1000
function skill(experience, middle, maximum, slope)
   return maximum/(1+exp(slope*(-experience+middle)))
end
target = skill.(1:N, 500, 2, 0.0075)
opponents = Random.randn.(1000)*0.5 .+ target

composition = [[["a"], [string(i)]] for i in 1:N]
results = [r? [1.,0.]:[0.,1.] for r in (Random.randn(N).+target.>Random.randn(N).+opponents)]
times = [i for i in 1:N]
priors = Dict{String,Player}()
for i in 1:N priors[string(i)] = Player(Gaussian(opponents[i], 0.2)) end

h = History(composition, results, times, priors, gamma=0.015)
convergence(h)
mu = [tp[2].mu for tp in learning_curves(h)["a"]]
```

Code 31: Skill evolution example using Julia.

```
N = 1000
skill <- function(experience, middle, maximum, slope){
   return(maximum/(1+exp(slope*(-experience+middle)))) }
target = skill(seq(N), 500, 2, 0.0075)
opponents = rnorm(N,target,0.5)

composition = list(); results = list(); times = c(); priors = hash()
for(i in seq(N)){composition[[i]] = list(c("a"), c(toString(i)))}
for(i in
    seq(N)){results[[i]]=if(rnorm(1,target[i])>rnorm(1,opponents[i])){c(1,0)}else{c(0,1)}}
for(i in seq(N)){times = c(times,i)}
for(i in seq(N)){priors[[toString(i)]] = Player(Gaussian(opponents[i],0.2))}

h = History(composition, results, times, priors, gamma=0.015)
h$convergence(); lc_a = h$learning_curves()$a; mu = c()
for(tp in lc_a){mu = c(mu,tp[[2]]@mu)}
```

Code 32: Skill evolution example using R.

5.2. The history of the Association of Tennis Professionals (ATP)

```
import pandas as pd; from datetime import datetime
df = pd.read_csv('input/history.csv')

columns = zip(df.w1_id, df.w2_id, df.l1_id, df.l2_id, df.double)
composition = [[[w1,w2],[l1,l2]] if d=='t' else [[w1],[l1]] for w1, w2, l1, l2, d in columns]
days = [ datetime.strptime(t, "%Y-%m-%d").timestamp()/(60*60*24) for t in df.time_start]

h = History(composition = composition, times = days, sigma = 1.6, gamma = 0.036)
```

```
h.convergence(epsilon=0.01, iterations=10)
```

Code 33: The history of the Association of Tennis Professionals using Python.

```
data = read.csv("input/history.csv", header=T)

get_composition = function(x){
    res = list()
    if (x["double"]=="t"){
        res[[1]] = c(x["w1_name"],x["w2_name"])
        res[[2]] = c(x["l1_name"],x["l2_name"])
    }else{
        res[[1]] = c(x["w1_name"])
        res[[2]] = c(x["l1_name"])
    }
    return(res)
}
composition = apply(data, 1, get_composition )
days = as.numeric(as.Date(data[,"time_start"], format = "%Y-%m-%d"))

h = History(composition = composition, times = days, sigma = 1.6, gamma = 0.036)
h$convergence(epsilon=0.01, iterations=10)
```

Code 34: The history of the Association of Tennis Professionals using R.

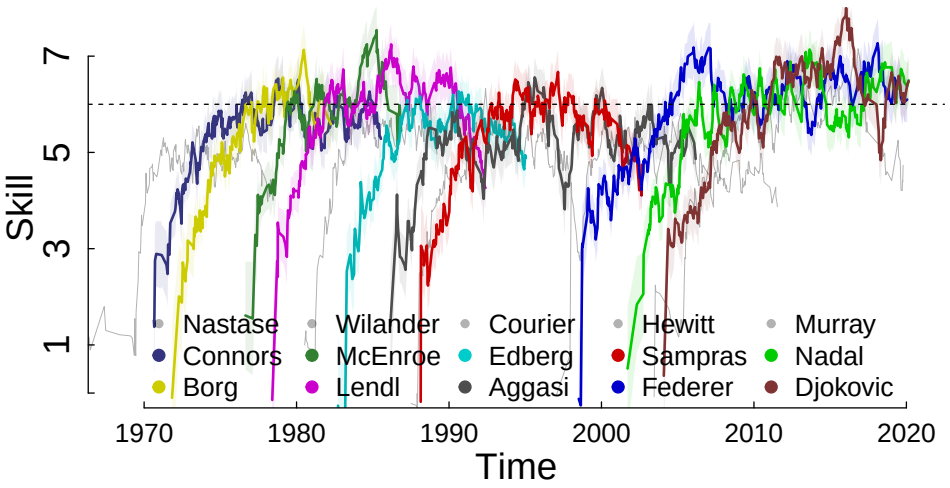


Figure 14: ATP skill estimates using the TrueSkill model.

5.3. Gaussian product

The problem we must solve is:

$$\int \mathcal{N}(x|\mu_1, \sigma_1^2)\mathcal{N}(x|\mu_2, \sigma_2^2)dx \tag{29}$$

By definition,

$$\begin{aligned}\mathcal{N}(x|y,\beta^2)\mathcal{N}(x|\mu,\sigma^2) &= \frac{1}{\sqrt{2\pi}\sigma_1}e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}\frac{1}{\sqrt{2\pi}\sigma_2}e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}} \\ &= \frac{1}{2\pi\sigma_1\sigma_2}\exp\Bigg(-\underbrace{\left(\frac{(x-\mu_1)^2}{2\sigma_1^2}+\frac{(x-\mu_2)^2}{2\sigma_2^2}\right)}_{\theta}\Bigg)\end{aligned} \tag{30}$$

Then,

$$\theta = \frac{\sigma_2^2(x^2+\mu_1^2-2x\mu_1)+\sigma_1^2(x^2+\mu_2^2-2x\mu_2)}{2\sigma_1^2\sigma_2^2} \tag{31}$$

We expand and reorder the factors by powers of x

$$\frac{(\sigma_1^2+\sigma_2^2)x^2-(2\mu_1\sigma_2^2+2\mu_2\sigma_1^2)x+(\mu_1^2\sigma_2^2+\mu_2^2\sigma_1^2)}{2\sigma_1^2\sigma_2^2} \tag{32}$$

We divide the numerator and denominator by the factor of x^2

$$\frac{x^2-2\frac{(\mu_1\sigma_2^2+\mu_2\sigma_1^2)}{(\sigma_1^2+\sigma_2^2)}x+\frac{(\mu_1^2\sigma_2^2+\mu_2^2\sigma_1^2)}{(\sigma_1^2+\sigma_2^2)}}{2\frac{\sigma_1^2\sigma_2^2}{(\sigma_1^2+\sigma_2^2)}} \tag{33}$$

This equation is quadratic in x, and is therefore proportional to a Gaussian density function with standard deviation

$$\sigma_\times = \sqrt{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}} \tag{34}$$

and mean

$$\mu_\times = \frac{(\mu_1\sigma_2^2+\mu_2\sigma_1^2)}{(\sigma_1^2+\sigma_2^2)} \tag{35}$$

Since a term $\varepsilon = 0$ can be added to complete the square in θ, this proof is sufficient when no normalization is needed.

$$\varepsilon = \frac{\mu_\times^2-\mu_\times^2}{2\sigma_\times^2} = 0 \tag{36}$$

By adding this term to θ we obtain

$$\theta = \frac{x^2-2\mu_\times x+\mu_\times^2}{2\sigma_\times^2}+\underbrace{\frac{\frac{(\mu_1^2\sigma_2^2+\mu_2^2\sigma_1^2)}{(\sigma_1^2+\sigma_2^2)}-\mu_\times^2}{2\sigma_\times^2}}_{\varphi} \tag{37}$$

Reorganizing φ

$$\begin{aligned}
\varphi &= \frac{\frac{(\mu_1^2\sigma_2^2+\mu_2^2\sigma_1^2)}{(\sigma_1^2+\sigma_2^2)} - \left(\frac{(\mu_1\sigma_2^2+\mu_2\sigma_1^2)}{(\sigma_1^2+\sigma_2^2)}\right)^2}{2\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}} \\
&= \frac{(\sigma_1^2+\sigma_2^2)(\mu_1^2\sigma_2^2+\mu_2^2\sigma_1^2) - (\mu_1\sigma_2^2+\mu_2\sigma_1^2)^2}{\sigma_1^2+\sigma_2^2}\frac{1}{2\sigma_1^2\sigma_2^2} \\
&= \frac{(\mu_1^2\sigma_1^2\sigma_2^2 + \cancel{\mu_2^2\sigma_1^4} + \cancel{\mu_1^2\sigma_2^4} + \mu_2^2\sigma_1^2\sigma_2^2) - (\cancel{\mu_1^2\sigma_2^4} + 2\mu_1\mu_2\sigma_1^2\sigma_2^2 + \cancel{\mu_2^2\sigma_1^4})}{\sigma_1^2+\sigma_2^2}\frac{1}{2\sigma_1^2\sigma_2^2} \\
&= \frac{(\sigma_1^2\sigma_2^2)(\mu_1^2+\mu_2^2-2\mu_1\mu_2)}{\sigma_1^2+\sigma_2^2}\frac{1}{2\sigma_1^2\sigma_2^2} = \frac{\mu_1^2+\mu_2^2-2\mu_1\mu_2}{2(\sigma_1^2+\sigma_2^2)} = \frac{(\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}
\end{aligned} \tag{38}$$

Then,

$$\theta = \frac{(x-\mu_\times)^2}{2\sigma_\times^2} + \frac{(\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)} \tag{39}$$

Putting θ in place

$$\begin{aligned}
\mathcal{N}(x|y,\beta^2)\mathcal{N}(x|\mu,\sigma^2) &= \frac{1}{2\pi\sigma_1\sigma_2}\exp\Bigg(-\underbrace{\left(\frac{(x-\mu_\times)^2}{2\sigma_\times^2} + \frac{(\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}\right)}_{\theta}\Bigg) \\
&= \frac{1}{2\pi\sigma_1\sigma_2}\exp\left(-\frac{(x-\mu_\times)^2}{2\sigma_\times^2}\right)\exp\left(-\frac{(\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}\right)
\end{aligned} \tag{40}$$

Multiplying by $\sigma_\times\sigma_\times^{-1}$

$$\frac{\overbrace{\cancel{\sigma_1\sigma_2}}^{\sigma_\times}}{\sqrt{\sigma_1^2+\sigma_2^2}}\frac{1}{\sigma_\times}\frac{1}{2\pi\cancel{\sigma_1\sigma_2}}\exp\left(-\frac{(x-\mu_\times)^2}{2\sigma_\times^2}\right)\exp\left(-\frac{(\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}\right) \tag{41}$$

Then,

$$\frac{1}{\sqrt{2\pi}\sigma_\times}\exp\left(-\frac{(x-\mu_\times)^2}{2\sigma_\times^2}\right)\frac{1}{\sqrt{2\pi(\sigma_1^2+\sigma_2^2)}}\exp\left(-\frac{(\mu_1-\mu_2)^2}{2(\sigma_1^2+\sigma_2^2)}\right) \tag{42}$$

Going back to the integral

$$\begin{aligned}
I &= \int \mathcal{N}(x|\mu_\times,\sigma_\times^2) \quad \overbrace{\mathcal{N}(\mu_1|\mu_2,\sigma_1^2+\sigma_2^2)}^{\text{Scalar magnitude independent of x}} \quad dx \\
&= \mathcal{N}(\mu_1|\mu_2,\sigma_1^2+\sigma_2^2)\underbrace{\int \mathcal{N}(x|\mu_\times,\sigma_\times^2)dx}_{\text{Integrates to 1}} \\
&= \mathcal{N}(\mu_1|\mu_2,\sigma_1^2+\sigma_2^2)
\end{aligned} \tag{43}$$

5.4. Sum of Gaussians

Proof by induction,

Base case

$$P(1) := \int \delta(t_1 = x_1)\mathcal{N}(x_1|\mu_1, \sigma_1^2)dx_1 = \mathcal{N}(t_1|\mu_1, \sigma_1^2) \tag{44}$$

The proposition $P(1)$ is true given the properties of the delta Dirac function.

$$\begin{aligned} P(2) &:= \iint \delta(t_2 = x_1 + x_2)\mathcal{N}(x_1|\mu_1, \sigma_1^2)\mathcal{N}(x_2|\mu_2, \sigma_2^2)dx_1dx_2 \\ &\overset{45.1}{=} \int \mathcal{N}(x_1|\mu_1, \sigma_1^2)\mathcal{N}(t_2 - x_1|\mu_2, \sigma_2^2)dx_1 \\ &\overset{45.2}{=} \int \mathcal{N}(x_1|\mu_1, \sigma_1^2)\mathcal{N}(x_1|t_2 - \mu_2, \sigma_2^2)dx_1 \\ &\overset{*}{=} \int \underbrace{N(t_2|\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)}_{\text{const.}}\underbrace{\mathcal{N}(x_1|\mu_*, \sigma_*^2)dx_1}_{1} \\ &= \mathcal{N}(t_2|\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2) \end{aligned} \tag{45}$$

where $\overset{45.1}{=}$ is valid for the properties of the dirac delta function, $\overset{45.2}{=}$ is valid for the symmetry of the Gaussians, and $\overset{*}{=}$ is valid by de proof at Section 5.3. Therefore, $P(2)$ is valid.

Inductive step $P(n) \Rightarrow P(n+1)$

Given,

$$P(n) := \int \cdots \int \delta(t_n = \sum_{i=1}^{n} x_i)\left(\prod_{i=1}^{n}\mathcal{N}(x_i|\mu_i, \sigma_i^2)\right)dx_1 \ldots dx_n = \mathcal{N}(t|\sum_{i=1}^{n}\mu_i, \sum_{i=1}^{n}\sigma_i^2) \tag{46}$$

We want to see that $P(n+1)$ is valid.

$$P(n+1) := \int \cdots \int \delta(t_{n+1} = x_{n+1} + \sum_{i=1}^{n} x_i)\left(\prod_{i=1}^{n}\mathcal{N}(x_i|\mu_i, \sigma_i^2)\right)\mathcal{N}(x_{n+1}|\mu_{n+1}, \sigma_{n+1}^2)dx_1 \ldots dx_n dx_{n+1} \tag{47}$$

By independence

$$\int \mathcal{N}(x_{n+1}|\mu_{n+1}, \sigma_{n+1}^2)\left(\int \cdots \int \delta(t_{n+1} = x_{n+1} + \sum_{i=1}^{n} x_i)\left(\prod_{i=1}^{n}\mathcal{N}(x_i|\mu_i, \sigma_i^2)\right)dx_1 \ldots dx_n\right)dx_{n+1} \tag{48}$$

By inductive hypothesis

$$\int \mathcal{N}(x_{n+1}|\mu_{n+1}, \sigma_{n+1}^2)\mathcal{N}(t_{n+1} - x_{n+1}|\sum_{i=1}^{n}\mu_i, \sum_{i=1}^{n}\sigma_i^2)dx_{n+1} \tag{49}$$

By de proof of Section 5.3

$$\mathcal{N}(t_{n+1}|\mu_{n+1} + \sum_{i=1}^{n}\mu_i, \sigma_{n+1}^2\sum_{i=1}^{n}\sigma_i^2)dx_{n+1} \tag{50}$$

Therefore, $P(n+1)$ is valid

5.5. A Gaussian multiplied by an cumulative Gaussian.

We want to solve the integral

$$f(x) = \int \mathcal{N}(y;\mu_1,\sigma_1^2)\Phi(y+x;\mu_2,\sigma_2^2)dy \tag{51}$$

To do so, we take the derivative of the function $\frac{\partial}{\partial x}f(x) = \theta(x)$,

$$\theta(x) = \frac{\partial}{\partial x}\int \mathcal{N}(y;\mu_1,\sigma_1^2)\Phi(y+x|\mu_2,\sigma_2^2)dy \tag{52}$$

$$\theta(x) = \int \mathcal{N}(y|\mu_1,\sigma_1^2)\frac{\partial}{\partial x}\Phi(y+x|\mu_2,\sigma_2^2)dy \tag{53}$$

The derivative of Φ is indeed a Gaussian,

$$\begin{aligned}\theta(x) &= \int \mathcal{N}(y|\mu_1,\sigma_1^2)\mathcal{N}(y+x|\mu_2,\sigma_2^2)dy \\ &= \int \mathcal{N}(y|\mu_1,\sigma_1^2)\mathcal{N}(y|\mu_2-x,\sigma_2^2)dy\end{aligned} \tag{54}$$

By the proof at Section 5.3 we know

$$\theta(x) = \mathcal{N}(\mu_1|\mu_2-x,\sigma_1^2+\sigma_2^2) \tag{55}$$

By symmetry

$$\theta(x) = \mathcal{N}(x|\mu_2-\mu_1,\sigma_1^2+\sigma_2^2) \tag{56}$$

Returning to $f(x)$

$$f(x) = \Phi(x|\mu_2-\mu_1,\sigma_1^2+\sigma_2^2) \tag{57}$$

5.6. Gaussian division

$$\kappa = \frac{\mathcal{N}(x|\mu_f,\sigma_f^2)}{\mathcal{N}(x|\mu_g,\sigma_g^2)} = \mathcal{N}(x|\mu_f,\sigma_f^2)\mathcal{N}(x|\mu_g,\sigma_g^2)^{-1} \tag{58}$$

By definition

$$\begin{aligned}\kappa &= \frac{1}{\sqrt{2\pi}\sigma_f}e^{-\left(\frac{(x-\mu_f)^2}{2\sigma_f^2}\right)}\left(\frac{1}{\sqrt{2\pi}\sigma_g}e^{-\left(\frac{(x-\mu_g)^2}{2\sigma_g^2}\right)}\right)^{-1} \\ &= \frac{1}{\cancel{\sqrt{2\pi}}\sigma_f}e^{-\left(\frac{(x-\mu_f)^2}{2\sigma_f^2}\right)}\frac{\cancel{\sqrt{2\pi}}\sigma_g}{1}e^{\left(\frac{(x-\mu_g)^2}{2\sigma_g^2}\right)} \\ &= \frac{\sigma_g}{\sigma_f}\exp\Bigg(-\underbrace{\left(\frac{(x-\mu_f)^2}{2\sigma_f^2}-\frac{(x-\mu_g)^2}{2\sigma_g^2}\right)}_{\theta}\Bigg)\end{aligned} \tag{59}$$

Reorganizing θ

$$\begin{aligned}\theta &= \frac{(x-\mu_f)^2}{2\sigma_f^2} - \frac{(x-\mu_g)^2}{2\sigma_g^2} = \frac{\sigma_g^2(x-\mu_f)^2 - \sigma_f^2(x-\mu_g)^2}{2\sigma_f^2\sigma_g^2} \\ &= \frac{\sigma_g^2(x^2+\mu_f^2-2\mu_f x) - \sigma_f^2(x^2+\mu_g^2-2\mu_g x)}{2\sigma_f^2\sigma_g^2}\end{aligned} \tag{60}$$

We expand and sort terms based on x,

$$\begin{aligned}\theta &= \left((\sigma_g^2-\sigma_f^2)x^2 - 2(\sigma_g^2\mu_f - \sigma_f^2\mu_g)x + (\sigma_g^2\mu_f^2 - \sigma_f^2\mu_g^2)\right)\frac{1}{2\sigma_f^2\sigma_g^2} \\ &= \left(x^2 - \frac{2(\sigma_g^2\mu_f - \sigma_f^2\mu_g)}{(\sigma_g^2-\sigma_f^2)}x + \frac{(\sigma_g^2\mu_f^2 - \sigma_f^2\mu_g^2)}{(\sigma_g^2-\sigma_f^2)}\right)\frac{(\sigma_g^2-\sigma_f^2)}{2\sigma_f^2\sigma_g^2}\end{aligned} \tag{61}$$

This is quadratic in x. Since a term $\varepsilon = 0$ independent of x can be added to complete the square in θ, this test is sufficient to determine the mean and variance when it is not necessary to normalize.

$$\sigma_\div = \sqrt{\frac{\sigma_f^2\sigma_g^2}{(\sigma_g^2-\sigma_f^2)}} \tag{62}$$

$$\mu_\div = \frac{(\sigma_g^2\mu_f - \sigma_f^2\mu_g)}{(\sigma_g^2-\sigma_f^2)} \tag{63}$$

adding $\varepsilon = \frac{\mu_\div^2 - \mu_\div^2}{2\sigma_\div^2} = 0$

$$\theta = \frac{x^2 - 2\mu_\div x + \mu_\div^2}{2\sigma_\div^2} + \underbrace{\frac{\frac{(\sigma_g^2\mu_f^2 - \sigma_f^2\mu_g^2)}{(\sigma_g^2-\sigma_f^2)} - \mu_\div^2}{2\sigma_\div^2}}_{\varphi} \tag{64}$$

Reorganizing φ

$$\begin{aligned}\varphi &= \left(\frac{(\sigma_g^2\mu_f^2 - \sigma_f^2\mu_g^2)}{(\sigma_g^2-\sigma_f^2)} - \left(\frac{(\sigma_g^2\mu_f - \sigma_f^2\mu_g)}{(\sigma_g^2-\sigma_f^2)}\right)^2\right)\frac{(\sigma_g^2-\sigma_f^2)}{2\sigma_f^2\sigma_g^2} \\ &= \left((\sigma_g^2\mu_f^2 - \sigma_f^2\mu_g^2)(\sigma_g^2-\sigma_f^2) - \left((\sigma_g^2\mu_f - \sigma_f^2\mu_g)\right)^2\right)\frac{1}{2\sigma_f^2\sigma_g^2(\sigma_g^2-\sigma_f^2)} \\ &= \left(\cancel{\sigma_g^4\mu_f^2} - \sigma_f^2\sigma_g^2\mu_f^2 - \sigma_f^2\sigma_g^2\mu_g^2 + \cancel{\sigma_f^4\mu_g^2} - (\cancel{\sigma_g^4\mu_f^2} + \cancel{\sigma_f^4\mu_g^2} - 2\sigma_f^2\sigma_g^2\mu_f\mu_g)\right)\frac{1}{2\sigma_f^2\sigma_g^2(\sigma_g^2-\sigma_f^2)}\end{aligned} \tag{65}$$

Canceling $\sigma_f^2\sigma_g^2$

$$\varphi = \frac{-\mu_g^2 - \mu_f^2 + 2\mu_f\mu_g}{2(\sigma_g^2-\sigma_f^2)} = \frac{-(\mu_g-\mu_f)^2}{2(\sigma_g^2-\sigma_f^2)} \tag{66}$$

Then θ

$$\theta = \frac{(x-\mu_\div)^2}{2\sigma_\div^2} - \frac{(\mu_g-\mu_f)^2)}{2(\sigma_g^2-\sigma_f^2)} \tag{67}$$

Returning to the original expression

$$\begin{aligned}\kappa &= \frac{\sigma_g}{\sigma_f}\exp\left(-\frac{(x-\mu_\div)^2}{2\sigma_\div^2} + \frac{(\mu_g-\mu_f)^2)}{2(\sigma_g^2-\sigma_f^2)}\right) \\ &= \frac{\sigma_g}{\sigma_f}\exp\left(-\frac{(x-\mu_\div)^2}{2\sigma_\div^2}\right)\exp\left(\frac{(\mu_g-\mu_f)^2)}{2(\sigma_g^2-\sigma_f^2)}\right)\end{aligned} \tag{68}$$

Multiplying by $\frac{\sqrt{2\pi}}{\sqrt{2\pi}}\frac{\sigma_\div}{\sigma_\div}\frac{\sqrt{\sigma_g^2-\sigma_f^2}}{\sqrt{\sigma_g^2-\sigma_f^2}} = 1$,

$$\begin{aligned}\kappa &= \frac{1}{\sqrt{2\pi}\sigma_\div}e^{-\frac{(x-\mu_\div)^2}{2\sigma_\div^2}}\left(\frac{1}{\sqrt{2\pi(\sigma_g^2-\sigma_f^2)}}e^{-\frac{(\mu_g-\mu_f)^2)}{2(\sigma_g^2-\sigma_f^2)}}\right)^{-1}\frac{\sigma_\div}{\sqrt{\sigma_g^2-\sigma_f^2}}\frac{\sigma_g}{\sigma_f} \\ &= \frac{\mathcal{N}\left(x|\mu_\div,\sigma_\div\right)}{\mathcal{N}\left(\mu_g|\mu_f,\sigma_g^2-\sigma_f^2\right)}\frac{\sigma_g^2}{\sigma_g^2-\sigma_f^2}\end{aligned} \tag{69}$$

Affiliation:

Gustavo Andrés Landfried
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Buenos Aires, Argentina
E-mail: glandfried@dc.uba.ar

and

Esteban Mocskos
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Buenos Aires, Argentina
and
Centro de Simulación Computacional p/Aplic Tecnológicas, CSC-CONICET
Buenos Aires, Argentina
E-mail: emocskos@dc.uba.ar