1RT705: Group 2230

1 Modeling

The Trueskill Bayesian model for one match between two players. The model consists of four random variables, two Gaussian random variables s_1 and s_2 for the skills of the players, one Gaussian random variable t, with mean $s_1 - s_1$, for the outcome of the game, and one discrete random variable $y = sign(t)$ for the result of the game.

$$s_1 \sim \mathcal{N}(\mu_1, \sigma_1^2) \tag{1}$$

$$s_2 \sim \mathcal{N}(\mu_2, \sigma_2^2) \tag{2}$$

$$p(t|s_1, s_2) \sim \mathcal{N}(s_1 - s_2, \sigma_3^2) \tag{3}$$

$$p(y|t) \sim \delta(y - sign(t)) \tag{4}$$

2 Computing with the model

The given task is to compute $p(s_1, s_2|t, y)$: the full conditional distribution of the skills. Since s_1 and s_2 are independent from y if t is observed (Question 3, "Baysian Network"), then the full conditional distribution of the skills can be written as

$$p(s_1, s_2|t, y) = p(s_1, s_2|t) \tag{5}$$

Using equation 1, 2 and 3 we can the use Theorem 1 (Marginalization) and Corollary 1 (Affine transformation – conditional) to compute equation 6.[2]

$$\mathcal{N}(\mu_{s_1,s_2|t}, \Sigma_{s_1,s_2|t}) \tag{6}$$

where

$$\mu_{s_1,s_2|t} = \Sigma_{s_1,s_2|t}(\Sigma^{-1}_{s_1,s_2}\mu_{s_1,s_2} + M^T\Sigma^{-1}_{t|s_1,s_2}t)$$

$$\Sigma_{s_1,s_2|t} = (\Sigma^{-1}_{s_1,s_2} + M^T\Sigma^{-1}_{t|s_1,s_2}M)^{-1}$$

and

$$\mu_{s_1,s_2} = \begin{pmatrix}\mu_{s_1}\\ \mu_{s_2}\end{pmatrix}$$

$$\Sigma_{s_1,s_2} = \begin{pmatrix}\sigma_1^2 & 0\\ 0 & \sigma_2^2\end{pmatrix}$$

$$M = (1, -1)$$

The second task was to compute $p(t|s_1, s_2, y)$: the full conditional distribution of the outcome, which can be written as equation 7.

$$p(t|s_1, s_2, y) \propto p(t|s_1, s_2)\delta(y - sign(t)) \tag{7}$$

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

This is a Truncated normal distribution with parameters: $\mu = s_1 - s_2, \sigma = \sigma_3, a$ and b.

where

$$y = 1 : a = 0, b = \infty$$
$$y = -1 : a = -\infty, b = 0$$

The third task was to compute $p(y = 1)$: the marginal probability that Player 1 wins the game. $p(y = 1)$ is the same as computing $p(t > 1)$. The marginal distribution of t can be calculated using Corollary 2 (Affine transformation – Marginalization).[2]

$$p(t) = \mathcal{N}(t, \mu_t, \Sigma_t) \tag{8}$$

where

$$\mu_t = (\mu_1 - \mu_2)$$
$$\Sigma_t = \sigma_1^2 + \sigma_2^2 + \sigma_3^2$$

$p(t > 1)$ is calculated with the Cumulative distribution function for a Normal distribution.

3 Bayesian Network

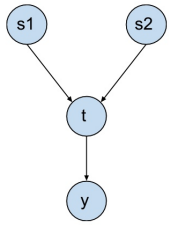


Figure 1: The Bayesian Network

If t is unobserved, then s_1 and s_2 are independent. If t is observed, then s_1 and y, respective s_2 and y are independent.[1]

4 A first Gibbs sampler

Figure 2 shows the samples of the posterior distributions generated by the Gibbs sampler given $y = 1$. The hyperparameters used for this project are chosen to be: $\mu_1, \mu_2 = 25$, $\sigma_1, \sigma_2 = \frac{25}{3}$, $\sigma_3 = \frac{25}{6}$ The initial condition for both s_1 and s_2 are =1. The burn in time in this case is negligible, since the distributions just after one sampling adopts its stationary state.

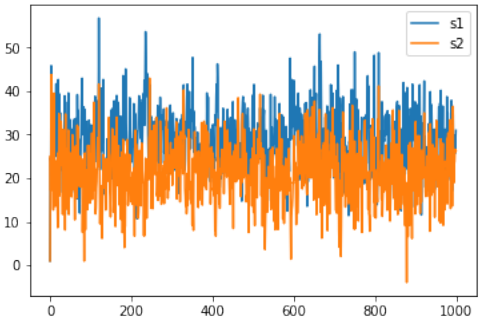


Figure 2: The samples of the posterior distribution when y=1

Figure 3 shows a plot of a function that uses the mean and covariance of the samples drawn by the Gibbs sampler to find a Gaussian approximation of the posterior distribution of the skills. We can see that the distribution of s_1 has a higher μ which is reasonable since y=1 (s_1 won the game) was given.

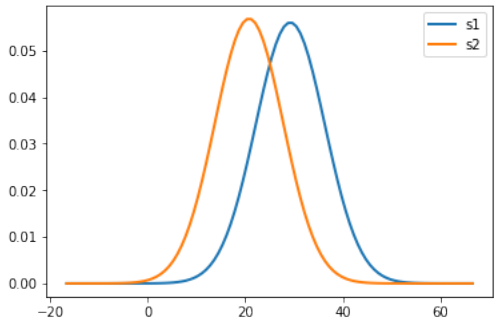


Figure 3: A Gaussian approximation of the posterior distribution of the skills

A reasonable number of samples is 1000, since the computational time is short (just under 1 second), and with a high accuracy of the estimate. This is evaluated after comparing the figures 4 and 5.

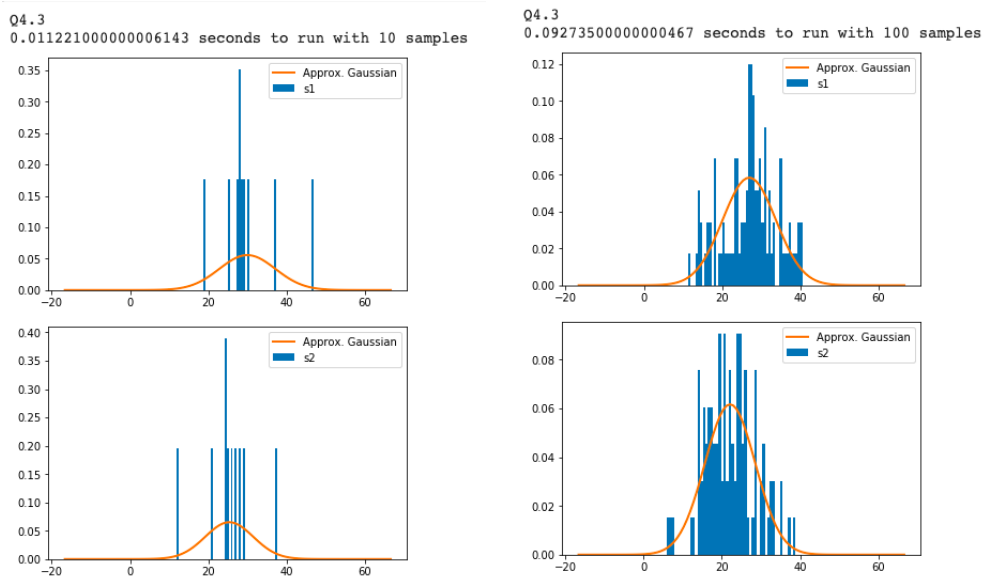


Figure 4: To the left is the histogram of the samples generated (after burn-in) together with the fitted Gaussian posterior with 10 samples. To the right is the histogram of the samples generated (after burn-in) together with the fitted Gaussian posterior with 100 samples

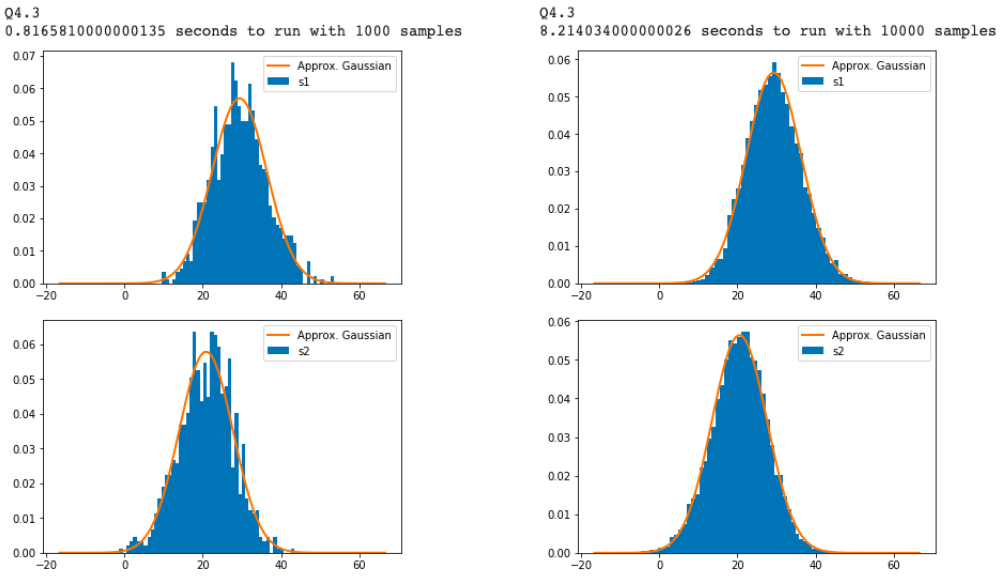


Figure 5: To the left is the histogram of the samples generated (after burn-in) together with the fitted Gaussian posterior with 1000 samples. To the right is the histogram of the samples generated (after burn-in) together with the fitted Gaussian posterior with 10000 samples

Figure 6 shows a Gaussian approximation of the posterior distribution of the skills, together with the prior for s_1 respective s_2. The plot shows that the variance has decreased for the posterior distribution, and the distribution of s_1 ha a higher mean than the prior while the distribution of s_2 has a lower mean than the prior. This is reasonable since y=1 (s_1 won the game) was given.

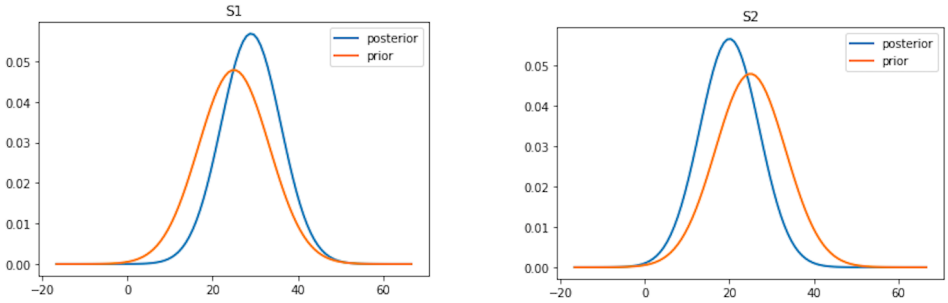


Figure 6: A Gaussian approximation of the posterior distribution of the skills, plotted together with the prior distribution for s_1 respective s_2

5 Assumed density filtering

Since the final ranking is not deterministic the result will differ slightly each execution round, but is more or less represented by the table in Figure 7. The variance describes how spread the values are from the mean, i.e. the uncertainty of a team's skill. The bigger the variance of the two teams playing, the more likely it is that the less skilled team wins.

	Skill	Standard Deviation	Rank
Atalanta	28.951417	2.441090	1
Milan	27.121873	2.659279	2
Roma	26.942464	2.301056	3
Inter	26.801433	2.374710	4
Napoli	26.724015	2.645592	5
Torino	26.555280	2.431742	6
Bologna	25.469011	2.407921	7
Juventus	24.687043	2.539508	8
Udinese	24.173970	2.171861	9
Empoli	23.710267	2.229341	10
Lazio	23.119308	2.235598	11
Spal	22.866036	2.291159	12
Sampdoria	22.762024	2.162140	13
Genoa	21.640384	2.265332	14
Cagliari	21.381627	2.122125	15
Sassuolo	20.823045	2.342561	16
Parma	20.632253	2.167308	17
Frosinone	18.764100	2.174588	18
Fiorentina	18.419086	2.156602	19
Chievo	16.767330	2.470285	20

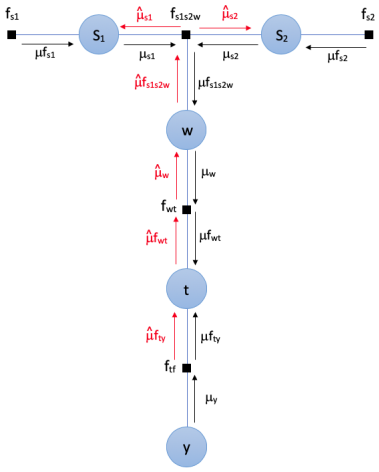
Figure 7: The final ranking

When changing the order of the matches, the skills also change more significantly then just the built in stochastic of the sampling. From looking at the order of the matches and the final ranking, it becomes clear that an early match affects a teams skill more than a later match. For example, winning the first match gives a more positive effect on a team's skill than winning the last match does.

6 Using the model for predictions

The model used for prediction was based on the ADF-model. A prediction function, taking estimates from the last iteration, calculates the expected results before the next ADF-iteration and any updates of the parameters. This prediction is than compared with the actual outcome of the match, which we then calculated our prediction rate from. The function gave 178 (out of 380) correct predictions. This equals a prediction rate of 0.468421. Assuming that guessing includes draws and roughly have a probability of 1/3, the prediction rate of the model is then better than random guessing. That the prediction does not take draws into account is however a clear downside. If disregarding games which ends in a draw, the prediction rate becomes 0.654412. This is also better than random guessing, which would have a probability of 1/2, given two possible outcomes of the game.

7 Factor graph



$$f_{s_1} = \mathcal{N}(s_1; \mu_1, \sigma_1^2) \tag{9}$$
$$f_{s_2} = \mathcal{N}(s_2; \mu_2, \sigma_2^2) \tag{10}$$
$$f_{s_1 s_2 w} = \delta(w - (s_1 - s_2)) \tag{11}$$
$$f_{wt} = \mathcal{N}(t; w, \sigma_3^2) \tag{12}$$
$$f_{ty} = \delta(y - sign(t)) \tag{13}$$

Figure 8: The Factor Graph with messages

$$\mu_y = 1_{[y=1]} \tag{14}$$

$$\mu_{fty} = \sum_y f_{ty}\mu_y \tag{15}$$

$$\mu_{fs_1} = f_{s_1} \tag{16}$$

$$\mu_{fs_2} = f_{s_2} \tag{17}$$

$$\mu_{s1} = \mu_{fs_1} \tag{18}$$

$$\mu_{s2} = \mu_{fs_2} \tag{19}$$

$$\mu_{fs_1s_2w} = \int\int f_{s_1s_2w}\mu_{fs_1}\mu_{fs_2}ds_1ds_2 \tag{20}$$

$$\mu_w = \mu_{fs_1s_2w} \tag{21}$$

$$\mu_{fwt} = \int f_{wt}\mu_w dw \tag{22}$$

8 A message-passing algorithm

Figure 9 shows both the posterior distribution of the skills after one match given the result that player 1 one the game, computed with a message passing algorithm, and the Gaussian approximation from Gibbs sampling. The posteriors are similar, which means that both these methods come to approximately the same posterior distributions.

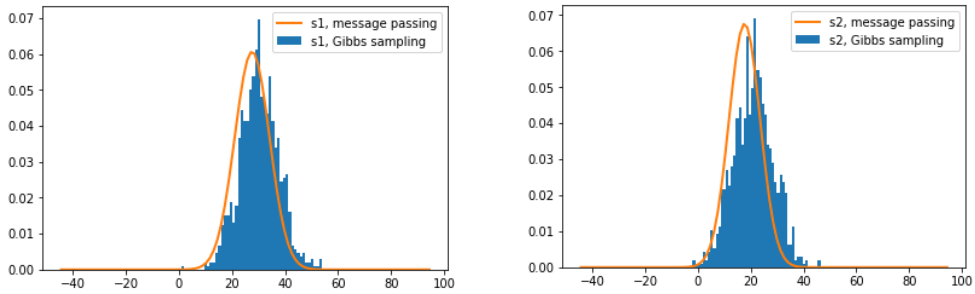


Figure 9: The posterior computed with message passing plotted together with the Gaussian approximation of the posterior, for s_1 respective s_2

9 Your own data

The developed TrueSkill methods was then tested on a dataset of all NBA games of season 2013-2014. The data was found through basketball-reference.com and downloaded using MS Power Query. After cleaning up the NBA dataset it was similar to the SerieA dataset and contains abbreviations of home and visting teams together with how many points they scored during the basketball game. No pre-processing was done except making sure the columns would be compatible with the TrueSkill functions, e.g. how many points the home team scored was named 'score1'. After running the ADF-function together with the dataset the team with the highest skill was SAS, i.e. San Antonio Spurs, which is also the winning team of that NBA season. However, the skills do not perfectly correspond to official rankings but show a lot of similarity. A full table of the final skills can be found, together with official rankings, in Appendix. Using the prediction function from question 6, a prediction rate of 0.581301 is received. There is two clear possible causes for receiving a higher prediction rate on this dataset compared to the one previously used. The first is that a lot of more points are scored during a basketball game compared to goals during a football match, making the case of a draw very rare. The second is that even though there are more teams in NBA, there are relatively seen games in the dataset which entails more observations, i.e. information about each team's skill. This possible cause is straightened by that the later predictions are more accurate than earlier ones.

10 Open-ended project extension

As illustrated in for example question 6, the model has some limitations and a higher prediction rate would be desirable. Almost a third of all matches in the given dataset "SerieA" ends in a draw, so a natural first extension to the prediction algorithm is to include the scenario of a draw. This was therefore implemented through taking the value of t into account instead of just the sign of t. A sufficiently small absolute value of t was then predicted to 0, and not 1 or -1. This threshold value was chosen to be 1, after a few experiments. However, this value is not guaranteed to be the optimal one, not for this data set and especially not for other data sets. Knowing that t will be very small during the first matches, due to initializing teams with the same mean and skill variance, draws was disregarded until the 20th match prediction. The number of correct predictions then became approximately 189 (out of 380). This gives a prediction rate of 0.497368, which is just slightly higher than the prediction rate before modelling draws. The prediction rate is lower than if disregarding draws or if the number of draws is almost negligible, like in the NBA dataset, which indicates the ability to model draws still needs to be improved.

References

[1] C. Bishop. *Pattern Recognition And Machine Learning*. Springer, 2006.

[2] N. Wahlström. *Advanced Probabilistic Machine Learning. Lecture 2 – Bayesian linear regression*. Department of Information Technology, Uppsala University, 2019.

11 Appendix

11.1 Question 9

	Skill	Standard Deviation	Rank
SAS	26.377039	2.677208	1
OKC	25.207947	2.338121	2
IND	24.947265	2.269921	3
MIA	24.762758	2.270062	4
HOU	24.269946	2.289646	5
POR	23.913343	2.189309	6
TOR	22.950961	2.183370	7
LAC	22.639585	2.321207	8
GSW	22.482919	1.994135	9
DAL	22.286159	2.070395	10
WAS	22.126563	2.030091	11
MEM	21.760838	2.158928	12
DEN	21.263193	1.957315	13
ATL	20.797562	1.996026	14
CHA	20.636158	1.943446	15
NYK	20.389839	1.840131	16
BRK	20.381852	1.918342	17
CHI	20.370694	2.098824	18
PHO	20.288569	1.989343	19
MIN	19.735442	2.027712	20
NOP	19.580789	1.825591	21
UTA	18.578883	2.012587	22
CLE	18.509251	1.919355	23
LAL	18.451016	2.085385	24
SAC	17.431820	2.062530	25
DET	17.212039	1.780286	26
ORL	16.341263	1.881120	27
PHI	16.248236	1.817708	28
BOS	15.757575	1.894714	29
MIL	14.827646	1.774168	30

Figure 10: The final ranking of the teams in season 2013-2014 of NBA.

Rk	Team	Conf	Div	W	L	W/L% ▼
1	San Antonio Spurs	W	SW	62	20	.756
2	Oklahoma City Thunder	W	NW	59	23	.720
3	Los Angeles Clippers	W	P	57	25	.695
4	Indiana Pacers	E	C	56	26	.683
5	Houston Rockets	W	SW	54	28	.659
6	Miami Heat	E	SE	54	28	.659
7	Portland Trail Blazers	W	NW	54	28	.659
8	Golden State Warriors	W	P	51	31	.622
9	Memphis Grizzlies	W	SW	50	32	.610
10	Dallas Mavericks	W	SW	49	33	.598
11	Phoenix Suns	W	P	48	34	.585
12	Toronto Raptors	E	A	48	34	.585
13	Chicago Bulls	E	C	48	34	.585
14	Washington Wizards	E	SE	44	38	.537
15	Brooklyn Nets	E	A	44	38	.537
16	Charlotte Bobcats	E	SE	43	39	.524
17	Minnesota Timberwolves	W	NW	40	42	.488
18	Atlanta Hawks	E	SE	38	44	.463
19	New York Knicks	E	A	37	45	.451
20	Denver Nuggets	W	NW	36	46	.439
21	New Orleans Pelicans	W	SW	34	48	.415
22	Cleveland Cavaliers	E	C	33	49	.402
23	Detroit Pistons	E	C	29	53	.354
24	Sacramento Kings	W	P	28	54	.341
25	Los Angeles Lakers	W	P	27	55	.329
26	Boston Celtics	E	A	25	57	.305
27	Utah Jazz	W	NW	25	57	.305
28	Orlando Magic	E	SE	23	59	.280
29	Philadelphia 76ers	E	A	19	63	.232
30	Milwaukee Bucks	E	C	15	67	.183

Figure 11: The of ranking of the teams in season 2013-2014 of NBA.
Source: https://www.basketball-reference.com/leagues/NBA_2014_ratings.html

11.2 Python Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Miniproject - APML

"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import truncnorm
from scipy.stats import norm
import time

print("Q4.1")

M = np.array([[1, -1]])
sigma1, sigma2, sigma3 = 25/3, 25/3, 25/6
mu1, mu2 = 25, 25
mu = np.array([[mu1], [mu2]])
sigmaA = np.array([[sigma1**2, 0], [0, sigma2**2]])
sigmaAB = np.linalg.inv(np.linalg.inv(sigmaA)+np.transpose(M)*(1/sigma3**2)@M)

myclip_a = 0
myclip_b = 1000
my_mean = mu1 - mu2
my_std = sigma3
a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std

# Gibbs sampling
K = 1000
s1 = np.zeros(K)
s2 = np.zeros(K)
t = np.zeros(K)
s1[0] = 1
s2[0] = 1
t[0] = 0
for k in range(K-1):
  muAB = sigmaAB@(np.linalg.inv(sigmaA)@mu+np.transpose(M)*(t[k]/sigma3**2))
  muAB2 = np.ravel(muAB)
  s1[k+1], s2[k+1] = np.random.multivariate_normal(muAB2, sigmaAB)
  my_mean = s1[k+1] - s2[k+1]
  a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std
  t[k+1] = truncnorm.rvs(a, b, my_mean, my_std)


plt.plot(s1,label="s1")
plt.plot(s2,label="s2")
plt.legend()
plt.show()

#*****************************************************************************

print("Q4.2")

#Parameters
x = np.linspace(mu1-5*sigma1,mu1+5*sigma1,100)
```

```
mu_s1 = np.mean(s1)
std_s1 = np.std(s1)
S1 = norm.pdf(x, mu_s1, std_s1)

mu_s2 = np.mean(s2)
std_s2 = np.std(s2)
S2 = norm.pdf(x, mu_s2, std_s2)

#Make a plot
plt.plot(x, S1, linewidth=2, label="s1")
plt.plot(x, S2, linewidth=2, label="s2")
plt.legend()
plt.show()

#*************************************************************************

print("Q4.3")

time_start = time.clock()
#run your code
time_elapsed = (time.clock() - time_start)

# Gibbs sampling
K = 1000
s1 = np.zeros(K)
s2 = np.zeros(K)
t = np.zeros(K)
s1[0] = 25
s2[0] = 25
t[0] = 0
for k in range(K-1):
  muAB = sigmaAB@(np.linalg.inv(sigmaA)@mu+np.transpose(M)*(t[k]/sigma3**2))
  muAB2 = np.ravel(muAB)
  s1[k+1], s2[k+1] = np.random.multivariate_normal(muAB2, sigmaAB)
  my_mean = s1[k+1] - s2[k+1]
  a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std
  t[k+1] = truncnorm.rvs(a, b, my_mean, my_std)

time_elapsed = (time.clock() - time_start)
print(time_elapsed, "seconds to run with", K, "samples")

#Parameters
x = np.linspace(mu1-5*sigma1,mu1+5*sigma1,100)
mu_s1 = np.mean(s1)
std_s1 = np.std(s1)
S1 = norm.pdf(x, mu_s1, std_s1)

mu_s2 = np.mean(s2)
std_s2 = np.std(s2)
S2 = norm.pdf(x, mu_s2, std_s2)

#Make a plot
plt.hist(s1, label="s1", bins =50, density=True)
plt.plot(x, S1, linewidth=2, label="Approx. Gaussian")
plt.legend()
plt.show()
plt.hist(s2, label="s2", bins =50, density=True)
plt.plot(x, S2, linewidth=2, label="Approx. Gaussian")
plt.legend()
```

```
plt.show()

#*************************************************************************

print("Q4.4")

# Gibbs sampling
K = 1000
s1 = np.zeros(K)
s2 = np.zeros(K)
t = np.zeros(K)
s1[0] = 1
s2[0] = 1
t[0] = 0
for k in range(K-1):
  muAB = sigmaAB@(np.linalg.inv(sigmaA)@mu+np.transpose(M)*(t[k]/sigma3**2))
  muAB2 = np.ravel(muAB)
  s1[k+1], s2[k+1] = np.random.multivariate_normal(muAB2, sigmaAB)
  my_mean = s1[k+1] - s2[k+1]
  a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std
  t[k+1] = truncnorm.rvs(a, b, my_mean, my_std)

x = np.linspace(mu1-5*sigma1,mu1+5*sigma1,100)

#Parameters to set
mu_s1 = np.mean(s1)
std_s1 = np.std(s1)
S1 = norm.pdf(x, mu_s1, std_s1)
S1_start = norm.pdf(x, mu1, sigma1)

mu_s2 = np.mean(s2)
std_s2 = np.std(s2)
S2 = norm.pdf(x, mu_s2, std_s2)
S2_start = norm.pdf(x, mu2, sigma2)

#Make a plot
plt.plot(x,S1, linewidth=2, label = "posterior")
plt.plot(x,S1_start, linewidth = 2, label = "prior")
plt.title('S1')
plt.legend()
plt.show()

plt.plot(x,S2, linewidth=2,label = "posterior")
plt.plot(x,S2_start, linewidth = 2,label = "prior")
plt.title('S2')
plt.legend()
plt.show()

#*************************************************************************

print("Q5")

def GibbsSampling(mu1,sigma1,mu2,sigma2):
    M = np.array([[1, -1]])
    sigma3 = (25/6)

    mu = np.array([[mu1], [mu2]])
    sigmaA = np.array([[sigma1**2, 0], [0, sigma2**2]])
    sigmaAB = np.linalg.inv(np.linalg.inv(sigmaA)+np.transpose(M)*(1/sigma3**2)@M)
```

```
    myclip_a = 0
    myclip_b = 1000
    my_mean = mu1 - mu2
    my_std = sigma3
    a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std

    # Gibbs sampling
    K = 1000
    s1 = np.zeros(K)
    s2 = np.zeros(K)
    t = np.zeros(K)
    s1[0] = 1
    s2[0] = 1
    t[0] = 0
    for k in range(K-1):
      muAB = sigmaAB@(np.linalg.inv(sigmaA)@mu+np.transpose(M)*(t[k]/sigma3**2))
      muAB2 = np.ravel(muAB)
      s1[k+1], s2[k+1] = np.random.multivariate_normal(muAB2, sigmaAB)
      my_mean = s1[k+1] - s2[k+1]
      a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std
      t[k+1] = truncnorm.rvs(a, b, my_mean, my_std)

    #Parameters
    mu_s1 = np.mean(s1)
    std_s1 = np.std(s1)

    mu_s2 = np.mean(s2)
    std_s2 = np.std(s2)

    return(mu_s1, std_s1, mu_s2, std_s2)


data = pd.read_csv("SerieA.csv")
teams = data['team1'].drop_duplicates()
data = data.sample(frac=1).reset_index(drop=True) #gives random data. comment out to
team_list = dict()

for team in teams:
    team_list[team] = (25,(25/3)) #default (mean,variance) for each team

for i in range(data.shape[0]):
    score = data.iloc[i]['score1'] - data.iloc[i]['score2']
    if score > 0:
        winner = data.iloc[i]['team1']
        looser = data.iloc[i]['team2']
        mu1, sigma1, mu2, sigma2 = GibbsSampling(team_list[winner][0], team_list[win
        team_list[winner] = [mu1, sigma1]
        team_list[looser] = [mu2, sigma2]

    elif score < 0:
        winner = data.iloc[i]['team2']
        looser = data.iloc[i]['team1']
        mu1, sigma1, mu2, sigma2 = GibbsSampling(team_list[winner][0], team_list[win
        team_list[winner] = [mu1, sigma1]
        team_list[looser] = [mu2, sigma2]

df = pd.DataFrame.from_dict(team_list, orient='index', columns=['Skill', 'Standard D
df.sort_values(by='Skill', inplace=True, ascending = False)
```

```
df['Rank'] = range(1, len(df) + 1)
print(team_list)

#*************************************************************************

print("Q6")

def GibbsSampling(mu1,sigma1,mu2,sigma2):
    M = np.array([[1, -1]])
    sigma3 = (25/6)

    mu = np.array([[mu1], [mu2]])
    sigmaA = np.array([[sigma1**2, 0], [0, sigma2**2]])
    sigmaAB = np.linalg.inv(np.linalg.inv(sigmaA)+np.transpose(M)*(1/sigma3**2)@M)

    myclip_a = 0
    myclip_b = 1000
    my_mean = mu1 - mu2
    my_std = sigma3
    a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std

    # Gibbs sampling
    K = 1000
    s1 = np.zeros(K)
    s2 = np.zeros(K)
    t = np.zeros(K)
    s1[0] = 1
    s2[0] = 1
    t[0] = 0
    for k in range(K-1):
      muAB = sigmaAB@(np.linalg.inv(sigmaA)@mu+np.transpose(M)*(t[k]/sigma3**2))
      muAB2 = np.ravel(muAB)
      s1[k+1], s2[k+1] = np.random.multivariate_normal(muAB2, sigmaAB)
      my_mean = s1[k+1] - s2[k+1]
      a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std
      t[k+1] = truncnorm.rvs(a, b, my_mean, my_std)

    #Parameters
    mu_s1 = np.mean(s1)
    std_s1 = np.std(s1)

    mu_s2 = np.mean(s2)
    std_s2 = np.std(s2)

    return(mu_s1, std_s1, mu_s2, std_s2)


def predict(mu1, sigma1, mu2, sigma2):
    L = 1000 #number of samples

    s1 = np.random.normal(mu1, sigma1, L)
    s2 = np.random.normal(mu2, sigma2, L)

    t = np.mean(s1 - s2)
    y = np.sign(t)
    result_list.append(y)

    return(result_list)
```

```
data = pd.read_csv("SerieA.csv")
teams = data['team1'].drop_duplicates()
team_list = dict()
result_list = list()

for team in teams:
    team_list[team] = (25,(25/3))

for i in range(data.shape[0]):
    score = data.iloc[i]['score1'] - data.iloc[i]['score2']
    predict(team_list[data.iloc[i]['team1']][0], team_list[data.iloc[i]['team1']][1]
    if score > 0:
        winner = data.iloc[i]['team1']
        looser = data.iloc[i]['team2']
        mu1, sigma1, mu2, sigma2 = GibbsSampling(team_list[winner][0], team_list[win
        team_list[winner] = [mu1, sigma1]
        team_list[looser] = [mu2, sigma2]

    elif score < 0:
        winner = data.iloc[i]['team2']
        looser = data.iloc[i]['team1']
        mu1, sigma1, mu2, sigma2 = GibbsSampling(team_list[winner][0], team_list[win
        team_list[winner] = [mu1, sigma1]
        team_list[looser] = [mu2, sigma2]


def compare_result(prediction, data):
    true_result = list()
    count = 0
    score = data['score1'] - data['score2']
    for i in range(data.shape[0]):
        if score[i] > 0:
            true_result.append(1)
        elif score[i] < 0:
            true_result.append(-1)
        else:
            true_result.append(0)

    for k in range(len(true_result)):
        if true_result[k] == prediction[k]:
            count += 1

    draw = true_result.count(0)
    pred_rate = count/380
    pred_rate_discard_zeros = count/(380-draw)
    print("Number of correct predictions:", count)
    print("Prediction rate:", pred_rate)
    print("Prediction rate if disregarding draws:", pred_rate_discard_zeros)
    return()

compare_result(result_list, data)

#*************************************************************************

print("Q8")

def multiplyGauss(m1,s1,m2,s2):
    s = 1/(1/s1+1/s2)
```

```
    m = (m1/s1+m2/s2)*s
    return m, s

def divideGauss(m1,s1,m2,s2):
    m,s = multiplyGauss(m1,s1,m2,-s2)
    return m, s

def truncGaussMM(my_a,my_b,m1,s1):
    a, b = (my_a - m1) / np.sqrt(s1), (my_b - m1) / np.sqrt(s1)
    m = truncnorm.mean(a, b, loc=m1, scale=np.sqrt(s1))
    s = truncnorm.var(a, b, loc=0, scale=np.sqrt(s1))
    return m, s

# Defining the 5 hyperparameters
# y0 = 1 indicates that player 1 won a game
m1 = 25
m2 = 25
v1 = (25/3)**2
v2 = (25/3)**2
v3 = (25/6)**2
y0 = 1

#--------STARTING THE MESSAGE PASSING FROM THE TOP------

# Message from factor f_s1 to node s1
mu_fs1_s1_m = m1
mu_fs1_s1_v = v1

# Message from factor f_s2 to node s2
mu_fs2_s2_m = m2
mu_fs2_s2_v = v2

# Message from node s1 to factor f_s1s2w
mu_s1_s1s2w_m = mu_fs1_s1_m
mu_s1_s1s2w_v = mu_fs1_s1_v

# Message from node s2 to factor f_s1s2w
mu_s2_fs1s2w_m = mu_fs2_s2_m
mu_s2_fs1s2w_v = mu_fs2_s2_v

# Message from factor f_s1s2w to node w
mu_fs1s2w_w_m = mu_fs1_s1_m - mu_fs2_s2_m
mu_fs1s2w_w_v = mu_fs1_s1_v + mu_fs2_s2_v

# Message from node w to factor f_wt
mu_w_fwt_m = mu_fs1s2w_w_m
mu_w_fwt_v = mu_fs1s2w_w_v

# Message from factor f_wt to node t
mu_fwt_t_m = mu_w_fwt_m
mu_fwt_t_v = mu_w_fwt_v + v3

# Do moment matching of the marginal of t
if y0==1:
    a, b = 0, 1000
else:
    a, b = -1000, 0

#Turning the truncated Gaussian into a Gaussian
```

```
pt_m, pt_v = truncGaussMM(a,b,mu_fwt_t_m,mu_fwt_t_v)

#Compute the updated message from f_yt to t
mu_fyt_t_m, mu_fyt_t_v = divideGauss(pt_m,pt_v,mu_fwt_t_m,mu_fwt_t_v)

#-------SENDING MESSAGES BACK UPWARDS----------
# Compute the message from node t to factor f_wt
mu_t_fwt_m = mu_fyt_t_m
mu_t_fwt_v = mu_fyt_t_v

# Compute the message from factor f_wt to node w
mu_fwt_x_m = mu_t_fwt_m
mu_fwt_x_v = mu_t_fwt_v + v3

# Compute the message from node w to factor f_s1s2w
mu_w_fs1s2w_m = mu_fwt_x_m
mu_w_fs1s2w_v = mu_fwt_x_v

# Compute the message from factor f_s1s2w to node s1
mu_fs1s2w_s1_m = mu_w_fs1s2w_m + mu_w_fs1s2w_m
mu_fs1s2w_s1_v = mu_w_fs1s2w_v + mu_s2_fs1s2w_v

# Compute the message from factor f_s1s2w to node s2
mu_fs1s2w_s2_m = mu_w_fs1s2w_m - mu_w_fs1s2w_m
mu_fs1s2w_s2_v = mu_w_fs1s2w_v + mu_s1_s1s2w_v

# Compute the marginal of s1 and s2
ps1_m, ps1_v = multiplyGauss(mu_fs1_s1_m,mu_fs1_s1_v,mu_fs1s2w_s1_m,mu_fs1s2w_s1_v)
ps2_m, ps2_v = multiplyGauss(mu_fs2_s2_m,mu_fs2_s2_v,mu_fs1s2w_s2_m,mu_fs1s2w_s2_v)

#--------MAKING THE PLOT-------
#Parameters
L = 100 #number of samples
x = np.linspace(m1-v1, m1+v1, 100)

#Draw values from the Gaussian distributions
s1_norm = np.random.normal(ps1_m, np.sqrt(ps1_v), L)
s2_norm = np.random.normal(ps2_m, np.sqrt(ps2_v), L)

mu_s1 = np.mean(s1_norm)
mu_s2 = np.mean(s2_norm)
var_s1 = np.var(s1_norm)
var_s2 = np.var(s2_norm)

#Making pdf:s
s1_pdf = norm.pdf(x, mu_s1, np.sqrt(var_s1))
s2_pdf = norm.pdf(x, mu_s2, np.sqrt(var_s2))

#Make a plot
plt.hist(s1, label="s1, Gibbs sampling", bins =50, density=True)
plt.plot(x, s1_pdf, linewidth=2, label="s1, message passing")
plt.legend()
plt.show()
plt.hist(s2, label="s2, Gibbs sampling", bins =50, density=True)
plt.plot(x, s2_pdf, linewidth=2, label="s2, message passing")
plt.legend()
plt.show()

#**********************************************************************
```

```
print("Q9")

def GibbsSampling(mu1,sigma1,mu2,sigma2):
    M = np.array([[1, -1]])
    sigma3 = (25/6)

    mu = np.array([[mu1], [mu2]])
    sigmaA = np.array([[sigma1**2, 0], [0, sigma2**2]])
    sigmaAB = np.linalg.inv(np.linalg.inv(sigmaA)+np.transpose(M)*(1/sigma3**2)@M)

    myclip_a = 0
    myclip_b = 1000
    my_mean = mu1 - mu2
    my_std = sigma3
    a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std

    # Gibbs sampling
    K = 1000
    s1 = np.zeros(K)
    s2 = np.zeros(K)
    t = np.zeros(K)
    s1[0] = 1
    s2[0] = 1
    t[0] = 0
    for k in range(K-1):
      muAB = sigmaAB@(np.linalg.inv(sigmaA)@mu+np.transpose(M)*(t[k]/sigma3**2))
      muAB2 = np.ravel(muAB)
      s1[k+1], s2[k+1] = np.random.multivariate_normal(muAB2, sigmaAB)
      my_mean = s1[k+1] - s2[k+1]
      a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std
      t[k+1] = truncnorm.rvs(a, b, my_mean, my_std)

    #Parameters
    s1 = np.delete(s1, 2, 0) #remove burn-in
    s2 = np.delete(s2, 2, 0) #remove burn-in
    mu_s1 = np.mean(s1)
    std_s1 = np.std(s1)

    mu_s2 = np.mean(s2)
    std_s2 = np.std(s2)

    return(mu_s1, std_s1, mu_s2, std_s2)


data = pd.read_csv("NBA1314_Games.csv")
teams = data['team1'].drop_duplicates()
data = data.sample(frac=1).reset_index(drop=True) #gives random data. comment out to
team_list = dict()

for team in teams:
    team_list[team] = (25,(25/3)) #default (mean,variance) for each team

for i in range(data.shape[0]):
    score = data.iloc[i]['score1'] - data.iloc[i]['score2']
    if score > 0:
        winner = data.iloc[i]['team1']
        #print(winner)
        looser = data.iloc[i]['team2']
```

```
        mu1, sigma1, mu2, sigma2 = GibbsSampling(team_list[winner][0], team_list[win
        team_list[winner] = [mu1, sigma1]
        team_list[looser] = [mu2, sigma2]

    elif score < 0:
        winner = data.iloc[i]['team2']
        #print(winner)
        looser = data.iloc[i]['team1']
        mu1, sigma1, mu2, sigma2 = GibbsSampling(team_list[winner][0], team_list[win
        team_list[winner] = [mu1, sigma1]
        team_list[looser] = [mu2, sigma2]


df = pd.DataFrame.from_dict(team_list, orient='index', columns=['Skill', 'Standard D
df.sort_values(by='Skill', inplace=True, ascending = False)
df['Rank'] = range(1, len(df) + 1)
print(team_list)

#*************************************************************************

print("Q10", "\n")


def GibbsSampling(mu1,sigma1,mu2,sigma2):
    M = np.array([[1, -1]])
    sigma3 = (25/6)

    mu = np.array([[mu1], [mu2]])
    sigmaA = np.array([[sigma1**2, 0], [0, sigma2**2]])
    sigmaAB = np.linalg.inv(np.linalg.inv(sigmaA)+np.transpose(M)*(1/sigma3**2)@M)

    myclip_a = 0
    myclip_b = 1000
    my_mean = mu1 - mu2
    my_std = sigma3
    a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std

    # Gibbs sampling
    K = 1000
    s1 = np.zeros(K)
    s2 = np.zeros(K)
    t = np.zeros(K)
    s1[0] = 1
    s2[0] = 1
    t[0] = 0
    for k in range(K-1):
      muAB = sigmaAB@(np.linalg.inv(sigmaA)@mu+np.transpose(M)*(t[k]/sigma3**2))
      muAB2 = np.ravel(muAB)
      s1[k+1], s2[k+1] = np.random.multivariate_normal(muAB2, sigmaAB)
      my_mean = s1[k+1] - s2[k+1]
      a, b = (myclip_a - my_mean) / my_std, (myclip_b - my_mean) / my_std
      t[k+1] = truncnorm.rvs(a, b, my_mean, my_std)

    #Parameters
    s1 = np.delete(s1, 2, 0) #remove burn-in
    s2 = np.delete(s2, 2, 0) #remove burn-in
    mu_s1 = np.mean(s1)
    std_s1 = np.std(s1)
```

```
    mu_s2 = np.mean(s2)
    std_s2 = np.std(s2)

    return(mu_s1, std_s1, mu_s2, std_s2)


def predict(mu1, sigma1, mu2, sigma2, i):
    L = 1000 #number of samples
    s1 = np.random.normal(mu1, sigma1, L)
    s2 = np.random.normal(mu2, sigma2, L)
    t = np.mean(s1 - s2)
    if i > 19 and abs(t) < 1:
        y = 0
        result_list.append(y)
    else:
        y = np.sign(t)
        result_list.append(y)

    return(result_list)


data = pd.read_csv("SerieA.csv")
teams = data['team1'].drop_duplicates()
team_list = dict()
result_list = list()


for team in teams:
    team_list[team] = (25,(25/3))

for i in range(data.shape[0]):
    score = data.iloc[i]['score1'] - data.iloc[i]['score2']
    predict(team_list[data.iloc[i]['team1']][0], team_list[data.iloc[i]['team1']][1]
    if score > 0:
        winner = data.iloc[i]['team1']
        looser = data.iloc[i]['team2']
        mu1, sigma1, mu2, sigma2 = GibbsSampling(team_list[winner][0], team_list[win
        team_list[winner] = [mu1, sigma1]
        team_list[looser] = [mu2, sigma2]

    elif score < 0:
        winner = data.iloc[i]['team2']
        looser = data.iloc[i]['team1']
        mu1, sigma1, mu2, sigma2 = GibbsSampling(team_list[winner][0], team_list[win
        team_list[winner] = [mu1, sigma1]
        team_list[looser] = [mu2, sigma2]


def compare_result(prediction, data):
    true_result = list()
    prediction_result = np.around(prediction)
    count = 0
    score = data['score1'] - data['score2']
    for i in range(data.shape[0]):
        if score[i] > 0:
            true_result.append(1)
        elif score[i] < 0:
            true_result.append(-1)
        else:
```

```
            true_result.append(0)

    for k in range(len(true_result)):
        if true_result[k] == prediction_result[k]:
            count += 1
    draw = true_result.count(0)
    pred_rate = count/380
    draw = true_result.count(0)
    pred_rate = count/380
    pred_rate_discard_zeros = count/(380-draw)
    print("Number of correct predictions:", count)
    print("Prediction rate:", pred_rate)
    print("Prediction rate if disregarding draws:", pred_rate_discard_zeros)

    return()

compare_result(result_list, data)
```