Fitting football models and visualizing predictions with the `footBayes` package

Leonardo Egidi, Vasilis Palaskas

2022-11-18

- Modeling football outcomes
  - Bivariate-Poisson model
    - Likelihood approach
    - Bayesian approach
- Installing the package
- Model fit
  - Static fit
    - Changing default priors
  - Dynamic fit
- Model estimates and visualization tools
  - Plotting and interpreting team-specific abilities
- Model checking
- Predictions and predictive accuracy
  - Posterior out-of-sample probabilities
  - Home-win posterior probabilities
  - Rank-league reconstruction
  - Model comparisons: LOOIC and WAIC
- Extensions in next versions
- References

Modeling football outcomes

Modeling football outcomes became incredibly popular over the last years. However, an **encompassing computational tool** able to fit in one step many alternative football models is missing yet.

With the `footBayes` package we want to fill the gap and to give the possibility to **fit, interpret and graphically explore** the following goal-based Bayesian football models using the underlying Stan (Stan Development Team (2020)) environment:

- Double Poisson: Baio and Blangiardo (2010), Groll, Schauberger, and Tutz (2015), Egidi, Pauli, and Torelli (2018)
- Bivariate Poisson: Karlis and Ntzoufras (2003)
- Skellam for goals' differences: Karlis and Ntzoufras (2009)
- Zero-Inflated Skellam for goals' differences: Karlis and Ntzoufras (2009)
- Student-t for goals' differences: Gelman (2014)
- Diagonal-Inflated Bivariate Poisson: Karlis and Ntzoufras (2003) .

Precisely, we'll learn how to:

- fit **static** maximum likelihood and Bayesian models;
- fit **dynamic** Bayesian models;

- change the prior distributions and perform some **sensitivity tests**;
- interpret **parameters' estimates**;
- retrieve predictive intervals for team-specific **football abilities**;
- check the models through graphical **posterior predictive checks**;
- obtain out-of-sample **predictions**;
- reconstruct the final **rank's league**;
- compare models.

The package is also available at the following link:

https://github.com/LeoEgidi/footBayes

In my opinion building packages is like an art's work, likely to be *never-ending*. To say in art's terms, I love this quote from Antoine de Saint-Exupéry (partially rephrased):

> "A (software) designer knows that he has reached perfection not when there is nothing more to add, but when there is nothing left to take away!"

Bivariate-Poisson model

One main concern with the double Poisson model relies on the fact that the goals scored during a match by two competing teams are conditionally independent. However, in team sports, such as football, water-polo, handball, hockey, and basketball it is reasonable to assume that the two outcome variables are **correlated** since **the two teams interact during the game**. Consider, for instance, the realistic football case of the home team leading with 1-0, when only ten minutes are left to play. The away team can then become more determined and can take more risk in an effort to score and achieve the draw within the end of the match. Or, even when one of the two teams is leading say with 3-0, or 4-0, it is likely it will be relaxing a bit, and the opposing team could score at least one goal quite easily. To this aim, goals' correlation due to a change in the behaviour of the team or both teams could be captured by a dependence parameter, accounting for positive correlation. Positive parametric goals' dependence is made possible by using a **bivariate Poisson distribution**.

Consider random variables $X_r, r = 1, 2, 3$, which follow independent Poisson distributions with parameters $\lambda_r > 0$. Then the random variables $X = X_1 + X_3$ and $Y = X_2 + X_3$ jointly follow a bivariate Poisson distribution $\text{BP}(\lambda_1, \lambda_2, \lambda_3)$, with joint probability function

$$\begin{aligned} P_{X,Y}(x, y) &= \Pr(X = x, Y = y) \\ &= \exp\{-(\lambda_1 + \lambda_2 + \lambda_3)\}\frac{\lambda_1^x}{x!}\frac{\lambda_2^y}{y!}\times \\ &\sum_{k=0}^{\min(x,y)}\binom{x}{k}\binom{y}{k}k!\left(\frac{\lambda_3}{\lambda_1\lambda_2}\right)^k. \end{aligned}$$

Marginally each random variable follows a Poisson distribution with $\text{E}(X) = \lambda_1 + \lambda_3,\ \text{E}(Y) = \lambda_2 + \lambda_3$, and $\text{cov}(X, Y) = \lambda_3$; λ_3 acts as a measure of dependence between the goals scored by the two competing teams. If $\lambda_3 = 0$ then the two variables are conditionally independent and the bivariate Poisson distribution reduces to the product of two independent Poisson distributions, the double Poisson case.

Let (x_n, y_n) denote the observed number of goals scored by the home and the away team in the n-th game, respectively. A general bivariate Poisson model allowing for goals' correlation, as in Karlis and Ntzoufras (2003) is the following:

$$
\begin{aligned}
X_n, Y_n | \lambda_{1n}, \lambda_{2n}, \lambda_{3n} &\sim \mathsf{BivPoisson}(\lambda_{1n}, \lambda_{2n}, \lambda_{3n}) \\
\log(\lambda_{1n}) &= \mu + \text{home} + \text{att}_{h_n} + \text{def}_{a_n} \\
\log(\lambda_{2n}) &= \mu + \text{att}_{a_n} + \text{def}_{h_n} \\
\log(\lambda_{3n}) &= \beta_0 + \gamma_1 \beta^{\text{home}}_{h_n} + \gamma_2 \beta^{\text{away}}_{a_n} + \gamma_3 \boldsymbol{\beta} w_n,
\end{aligned}
$$

where $\lambda_{1n}, \lambda_{2n}$ represent the **scoring rates** for the home and the away team, respectively; μ represents the **constant intercept**; home represents the **home-effect**, i.e. the well-known advantage of the team hosting the game; att_t and def_t represent the **attack** and the **defence** abilities, respectively, for each team t, $t = 1, \dots, T$; the nested indexes $h_n, a_n = 1, \dots, T$ denote the home and the away team playing in the n-th game, respectively; β_0 is a constant parameter; $\beta^{\text{home}}_{h_n}$ and $\beta^{\text{away}}_{a_n}$ are parameters that depend on the home and away team respectively, w_n is a vector of covariates for the n-th match used to model the covariance term and $\boldsymbol{\beta}$ is the corresponding vector of regression coefficients. The parameters γ_1, γ_2 and γ_3 are dummy binary indicators taking values 0 or 1 which may activate distinct sources of the linear predictor. Hence when $\gamma_1 = \gamma_2 = \gamma_3 = 0$ we consider constant covariance as in Egidi and Torelli (2020) , whereas when $(\gamma_1, \gamma_2, \gamma_3) = (1, 1, 0)$ we assume that the covariance depends on the teams' parameters only but not on further match covariates, and so on.

The case $\lambda_{3n} = 0$ (the scores' correlation parameter equals zero) reduces to the double Poisson model, as in Baio and Blangiardo (2010) .

To achieve model's identifiability, attack/defence parameters are imposed a **sum-to-zero** constraint:

$$
\sum_{t=1}^{T} \text{att}_t = 0, \;\; \sum_{t=1}^{T} \text{def}_t = 0.
$$

Another identifiability constraint, largely proposed in the football literature, is the **corner**-constraint, which assumes the abilities for the T-th team are equal to the negative sum of the others, and then achieves a sum-to-zero as well:

$$
\text{att}_T = -\sum_{t=1}^{T-1} \text{att}_t, \;\; \text{def}_T = -\sum_{t=1}^{T-1} \text{def}_t.
$$

The current version of the package allows for the fit of diagonal-inflated Bivariate Poisson Karlis and Ntzoufras (2003) and zero-inflated Skellam models to better capture the probability of draw occurrences. A draw between two teams is represented by the outcomes on the diagonal of the probability table. To correct for the excess of draws we may add an inflation component on the diagonal of the probability function. This model is an extension of the simple zero-inflated model that allows only for an excess in (0,0) draws. Let's focus on the diagonal-inflated bivariate Poisson model, specified as:

$$
P_{X,Y}(x,y) = \Pr(X = x, Y = y) = \begin{cases} (1-p)\text{BP}(\lambda_1, \lambda_2, \lambda_3) & \text{if } x \neq y \\ (1-p)\text{BP}(\lambda_1, \lambda_2, \lambda_3) + pD(x, \eta) & \text{if } x = y, \end{cases}
$$

where $D(x, \eta)$ is a discrete distribution with parameter vector η.

Now it's time to fit and interpret the models, and we'll mainly focus on the bivariate Poisson case. Classical estimates for BP models are provided, among the others, by Karlis and Ntzoufras (2003) (MLE through an EM algorithm) and Koopman and Lit (2015); in the following, we quickly revise the maximum likelihood approach, but we'll deeply focus on Bayesian estimation with the underlying `rstan` software to better capture:

- parameters' uncertainty;
- predictions' uncertainty;
- model checking.

Likelihood approach

Given the parameter-vector $\boldsymbol{\theta} = (\{\text{att}_t, \text{def}_t, t = 1, \ldots, T\}, \mu, \text{home}, \beta^{\text{home}}_{h_n}, \beta^{\text{away}}_{a_n}, \beta_0, \boldsymbol{\beta})$, the likelihood function of the bivariate Poisson model above takes the following form:

$$L(\boldsymbol{\theta}) = \prod_{n=1}^{N} \exp\{-(\lambda_{1n} + \lambda_{2n} + \lambda_{3n})\} \frac{\lambda_{1n}^{x_n}}{x_n!} \frac{\lambda_{2n}^{y_n}}{y_n!} \times \sum_{k=0}^{\min(x_n, y_n)} \binom{x_n}{k} \binom{y_n}{k} k! \left(\frac{\lambda_{3n}}{\lambda_{1n}\lambda_{2n}} \right)^k.$$

Maximum-likelihood parameters estimation can be performed by searching the MLE $\hat{\boldsymbol{\theta}}$ such that:

$$\hat{\boldsymbol{\theta}} = \underset{\theta \in \Theta}{\text{argmax}}\, L(\boldsymbol{\theta}),$$

by imposing the following system of partial (log)-likelihood equations:

$$l'(\boldsymbol{\theta}) = 0.$$

Wald and deviance-confidence intervals may be constructed for the MLE $\hat{\boldsymbol{\theta}}$. A 95% Wald-type interval satisfies:

$$\hat{\boldsymbol{\theta}} \pm 1.96 \text{ se}(\hat{\boldsymbol{\theta}}).$$

As we'll see, the `footBayes` package allows the MLE computational approach (along with Wald-type and profile-likelihood confidence intervals) for static models only, i.e. when the model complexity is considered acceptable. As the parameters' space grows—as it commonly happens when adding dynamic patterns—MLE becomes computationally expensive and less reliable.

Bayesian approach

The goal of the Bayesian analysis is to carry out inferential conclusions from the joint posterior distribution $\pi(\boldsymbol{\theta}|\mathcal{D})$, where $\mathcal{D} = (x_n, y_n)_{n=1,\ldots,N}$ denotes the set of observed data for the N matches. The joint posterior satisfies

$$\pi(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{p(\mathcal{D})} \propto p(\mathcal{D}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}),$$

where $p(\mathcal{D}|\boldsymbol{\theta})$ is the model sampling distribution (proportional to the likelihood function), $\pi(\boldsymbol{\theta})$ is the joint prior distribution for $\boldsymbol{\theta}$, and $p(\mathcal{D}) = \int_{\Theta} p(\mathcal{D}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\theta$ is the marginal likelihood that does not depend on θ.

In the majority of the cases, $\pi(\boldsymbol{\theta}|\mathcal{D})$ does not have a closed form and for such reason we need to approximate it by simulation. The most popular class of algorithms designed to achieve this task is named **Markov Chain Monte Carlo Methods** (see Robert and Casella (2013) for a deep theoretical overview). These methods allow to sample weak correlated samples from some Markov chains whose stationary and limiting distribution coincide with the posterior distribution that we wish to approximate and sample from.

The `footBayes` package relies on a sophisticated MCMC enginery, namely the **Hamiltonian Monte Carlo** performed by the Stan software: the HMC borrow its name from the Hamiltonian dynamics of physics and is aimed at suppressing random-walk and wasteful behaviours in the exploration of the posterior distribution which typically arise when using the Gibbs sampling and the Metropolis-Hastings algorithm. For a deep and great summary about HMC, you may read the paper Betancourt (2017).

In terms of inferential conclusions, we are usually interested in summaries from the marginal posterior distributions of the single parameters: posterior means, medians, credibility intervals, etc.. We can write out the formula for the posterior distribution of the bivariate Poisson model above as:

$$\pi(\boldsymbol{\theta}|\mathcal{D}) \propto \pi(\boldsymbol{\theta}) \prod_{n=1}^{N} \mathsf{BivPoisson}(\lambda_{1n}, \lambda_{2n}, \lambda_{3n}),$$

where $\pi(\boldsymbol{\theta}) = \pi(\text{att})\pi(\text{def})\pi(\mu)\pi(\text{home})\pi(\beta_{h_n}^{\text{home}})\pi(\beta_{a_n}^{\text{away}})\pi(\beta_0)\pi(\boldsymbol{\beta})$ is the joint ptior distribution under the assumption of a-priori independent parameters' components.

The standard approach is to assign some **weakly-informative** prior distributions to the team-specific abilities. These parameters are considered **exchangeable** from two common (prior) distributions:

$$\begin{aligned}\text{att}_t &\sim \mathcal{N}(\mu_{\text{att}}, \sigma_{\text{att}})\\ \text{def}_t &\sim \mathcal{N}(\mu_{\text{def}}, \sigma_{\text{def}}), \quad t = 1, \ldots, T,\end{aligned}$$

with **hyperparameters** $\mu_{\text{att}}, \sigma_{\text{att}}, \mu_{\text{def}}, \sigma_{\text{def}}$. The model formulation is completed by assigning some weakly-informative priors to the remaining parameters. In what follows, some priors' options will be handled directly by the user.

Installing the package

Let's install the `footBayes` package from Github:

```
library(devtools)
install_github("LeoEgidi/footBayes")
```

and load the following required packages (please, install them on your laptops):

```
library(footBayes)
library(engsoccerdata)
library(bayesplot)
library(loo)
library(ggplot2)
library(dplyr)
library(tidyverse)
```

Model fit

Static fit

To start with some analysis, let's now import some data about the Italian Serie A, season 2000/2001, through the `engsoccerdata` package: the season consists of $T = 18$ teams, we start fitting a static bivariate Poisson model using:

- the likelihood approach: the `mle_foot` function returns the MLE estimates along with 95% profile-likelihood deviance confidence intervals (by default) and Wald-type confidence intervals. The user can specify the desired confidence interval with the optional argument `interval = c("profile", "Wald")`.
- the Bayesian approach: the `stan_foot` function produces an Hamiltonian Monte Carlo posterior sampling by using the underlying `rstan` ecosystem. The user can choose the number of iterations (`iter`), the number of Markov chains (`chains`), and other optional arguments values. With the `print` function, the usual Bayesian model summaries can be obtained: posterior means, medians, standard deviations, percentiles at 2.5%, 25%, 75%, 97.5% level, effective sample size (`n_eff`) and Gelman-Rubin statistic (`Rhat`).

At this stage, we are currently ignoring any time-dependence in our parameters, considering them to be **static** across distinct match-times.

```
### Use Italian Serie A 2000/2001

## with 'tidyverse' environment
#
#library(tidyverse)
#italy <- as_tibble(italy)
#italy_2000<- italy %>%
#  dplyr::select(Season, home, visitor, hgoal,vgoal) #%>%
#  dplyr::filter(Season=="2000")
#italy_2000

## alternatively, you can use the basic 'subsetting' code,
## not using the 'tidyverse' environment:
data(italy)
italy <- as.data.frame(italy)
italy_2000 <- subset(italy[, c(2,3,4,6,7)],
                     Season =="2000")
head(italy_2000)
#>       Season           home        visitor hgoal vgoal
#> 23421   2000 Udinese Calcio Brescia Calcio     4     2
#> 23422   2000        AS Roma     Bologna FC     2     0
#> 23423   2000     AC Perugia       US Lecce     1     1
#> 23424   2000 Reggina Calcio          Inter     2     1
#> 23425   2000     SSC Napoli       Juventus     1     2
#> 23426   2000       AC Milan Vicenza Calcio     2     0

### Fit Stan models
## no dynamics, no predictions
## 4 Markov chains, 'n_iter' iterations each

n_iter <- 200    # number of MCMC iterations
fit1_stan <- stan_foot(data = italy_2000,
                       model="biv_pois",
                       chains = 4,
                       #cores = 4,
                       iter = n_iter) # biv poisson
#>
#> SAMPLING FOR MODEL '02e61e6dbc2773b8f36869e8591b3648' NOW (CHAIN 1).
#> Chain 1:
#> Chain 1: Gradient evaluation took 0 seconds
#> Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
#> Chain 1: Adjust your expectations accordingly!
#> Chain 1:
#> Chain 1:
#> Chain 1: WARNING: There aren't enough warmup iterations to fit the
#> Chain 1:          three stages of adaptation as currently configured.
#> Chain 1:          Reducing each adaptation stage to 15%/75%/10% of
#> Chain 1:          the given number of warmup iterations:
#> Chain 1:            init_buffer = 15
#> Chain 1:            adapt_window = 75
#> Chain 1:           term_buffer = 10
#> Chain 1: 
#> Chain 1: Iteration:   1 / 200 [  0%]  (Warmup)
#> Chain 1: Iteration:  20 / 200 [ 10%]  (Warmup)
#> Chain 1: Iteration:  40 / 200 [ 20%]  (Warmup)
#> Chain 1: Iteration:  60 / 200 [ 30%]  (Warmup)
#> Chain 1: Iteration:  80 / 200 [ 40%]  (Warmup)
#> Chain 1: Iteration: 100 / 200 [ 50%]  (Warmup)
#> Chain 1: Iteration: 101 / 200 [ 50%]  (Sampling)
#> Chain 1: Iteration: 120 / 200 [ 60%]  (Sampling)
#> Chain 1: Iteration: 140 / 200 [ 70%]  (Sampling)
#> Chain 1: Iteration: 160 / 200 [ 80%]  (Sampling)
#> Chain 1: Iteration: 180 / 200 [ 90%]  (Sampling)
#> Chain 1: Iteration: 200 / 200 [100%]  (Sampling)
#> Chain 1: 
#> Chain 1:  Elapsed Time: 1.235 seconds (Warm-up)
#> Chain 1:                2.014 seconds (Sampling)
#> Chain 1:                3.249 seconds (Total)
#> Chain 1: 
#> 
#> SAMPLING FOR MODEL '02e61e6dbc2773b8f36869e8591b3648' NOW (CHAIN 2).
#> Chain 2: 
#> Chain 2: Gradient evaluation took 0 seconds
#> Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
#> Chain 2: Adjust your expectations accordingly!
#> Chain 2: 
#> Chain 2: 
#> Chain 2: WARNING: There aren't enough warmup iterations to fit the
#> Chain 2:          three stages of adaptation as currently configured.
#> Chain 2:          Reducing each adaptation stage to 15%/75%/10% of
#> Chain 2:          the given number of warmup iterations:
#> Chain 2:            init_buffer = 15
#> Chain 2:            adapt_window = 75
#> Chain 2:            term_buffer = 10
#> Chain 2: 
#> Chain 2: Iteration:   1 / 200 [  0%]  (Warmup)
#> Chain 2: Iteration:  20 / 200 [ 10%]  (Warmup)
#> Chain 2: Iteration:  40 / 200 [ 20%]  (Warmup)
#> Chain 2: Iteration:  60 / 200 [ 30%]  (Warmup)
#> Chain 2: Iteration:  80 / 200 [ 40%]  (Warmup)
#> Chain 2: Iteration: 100 / 200 [ 50%]  (Warmup)
#> Chain 2: Iteration: 101 / 200 [ 50%]  (Sampling)
#> Chain 2: Iteration: 120 / 200 [ 60%]  (Sampling)
#> Chain 2: Iteration: 140 / 200 [ 70%]  (Sampling)
#> Chain 2: Iteration: 160 / 200 [ 80%]  (Sampling)
#> Chain 2: Iteration: 180 / 200 [ 90%]  (Sampling)
#> Chain 2: Iteration: 200 / 200 [100%]  (Sampling)
#> Chain 2: 
#> Chain 2:  Elapsed Time: 1.013 seconds (Warm-up)
#> Chain 2:                0.809 seconds (Sampling)
#> Chain 2:                1.822 seconds (Total)
#> Chain 2: 
#> 
#> SAMPLING FOR MODEL '02e61e6dbc2773b8f36869e8591b3648' NOW (CHAIN 3).
#> Chain 3:
#> Chain 3: Gradient evaluation took 0 seconds
#> Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
#> Chain 3: Adjust your expectations accordingly!
#> Chain 3:
#> Chain 3:
#> Chain 3: WARNING: There aren't enough warmup iterations to fit the
#> Chain 3:          three stages of adaptation as currently configured.
#> Chain 3:          Reducing each adaptation stage to 15%/75%/10% of
#> Chain 3:          the given number of warmup iterations:
#> Chain 3:            init_buffer = 15
#> Chain 3:            adapt_window = 75
#> Chain 3:            term_buffer = 10
#> Chain 3:
#> Chain 3: Iteration:   1 / 200 [  0%]  (Warmup)
#> Chain 3: Iteration:  20 / 200 [ 10%]  (Warmup)
#> Chain 3: Iteration:  40 / 200 [ 20%]  (Warmup)
#> Chain 3: Iteration:  60 / 200 [ 30%]  (Warmup)
#> Chain 3: Iteration:  80 / 200 [ 40%]  (Warmup)
#> Chain 3: Iteration: 100 / 200 [ 50%]  (Warmup)
#> Chain 3: Iteration: 101 / 200 [ 50%]  (Sampling)
#> Chain 3: Iteration: 120 / 200 [ 60%]  (Sampling)
#> Chain 3: Iteration: 140 / 200 [ 70%]  (Sampling)
#> Chain 3: Iteration: 160 / 200 [ 80%]  (Sampling)
#> Chain 3: Iteration: 180 / 200 [ 90%]  (Sampling)
#> Chain 3: Iteration: 200 / 200 [100%]  (Sampling)
#> Chain 3:
#> Chain 3:  Elapsed Time: 0.998 seconds (Warm-up)
#> Chain 3:                0.976 seconds (Sampling)
#> Chain 3:                1.974 seconds (Total)
#> Chain 3:
#>
#> SAMPLING FOR MODEL '02e61e6dbc2773b8f36869e8591b3648' NOW (CHAIN 4).
#> Chain 4:
#> Chain 4: Gradient evaluation took 0.001 seconds
#> Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 10 seconds.
#> Chain 4: Adjust your expectations accordingly!
#> Chain 4:
#> Chain 4:
#> Chain 4: WARNING: There aren't enough warmup iterations to fit the
#> Chain 4:          three stages of adaptation as currently configured.
#> Chain 4:          Reducing each adaptation stage to 15%/75%/10% of
#> Chain 4:          the given number of warmup iterations:
#> Chain 4:            init_buffer = 15
#> Chain 4:            adapt_window = 75
#> Chain 4:            term_buffer = 10
#> Chain 4:
#> Chain 4: Iteration:   1 / 200 [  0%]  (Warmup)
#> Chain 4: Iteration:  20 / 200 [ 10%]  (Warmup)
#> Chain 4: Iteration:  40 / 200 [ 20%]  (Warmup)
#> Chain 4: Iteration:  60 / 200 [ 30%]  (Warmup)
#> Chain 4: Iteration:  80 / 200 [ 40%]  (Warmup)
#> Chain 4: Iteration: 100 / 200 [ 50%]  (Warmup)
#> Chain 4: Iteration: 101 / 200 [ 50%]  (Sampling)
#> Chain 4: Iteration: 120 / 200 [ 60%]  (Sampling)
#> Chain 4: Iteration: 140 / 200 [ 70%]  (Sampling)
#> Chain 4: Iteration: 160 / 200 [ 80%]  (Sampling)
#> Chain 4: Iteration: 180 / 200 [ 90%]  (Sampling)
#> Chain 4: Iteration: 200 / 200 [100%]  (Sampling)
#> Chain 4: 
#> Chain 4:  Elapsed Time: 1.271 seconds (Warm-up)
#> Chain 4:                1.553 seconds (Sampling)
#> Chain 4:                2.824 seconds (Total)
#> Chain 4: 

## print of model summary for parameters:
## home, sigma_att, sigma_def

print(fit1_stan, pars =c("home", "rho", "sigma_att",
                         "sigma_def"))
#> Inference for Stan model: 02e61e6dbc2773b8f36869e8591b3648.
#> 4 chains, each with iter=200; warmup=100; thin=1; 
#> post-warmup draws per chain=100, total post-warmup draws=400.
#> 
#>            mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
#> home       0.29    0.01 0.07  0.14  0.24  0.29  0.34  0.42   178 1.01
#> rho       -1.78    0.04 0.38 -2.60 -2.00 -1.72 -1.51 -1.20    83 1.05
#> sigma_att  0.21    0.01 0.07  0.10  0.16  0.20  0.25  0.37   127 1.01
#> sigma_def  0.22    0.01 0.07  0.10  0.17  0.21  0.26  0.38   102 1.03
#> 
#> Samples were drawn using NUTS(diag_e) at Fri Nov 18 14:56:11 2022.
#> For each parameter, n_eff is a crude measure of effective sample size,
#> and Rhat is the potential scale reduction factor on split chains (at 
#> convergence, Rhat=1).
```

The **Gelman-Rubin statistic** $\hat{R}$ (`Rhat`) is below the threshold 1.1 for all the parameters, whereas the **effective sample size** (`n_eff`), measuring the approximate number of iid replications from the Markov chains, does not appear to be problematic. Thus, HMC sampling reached the convergence.

As we could expect, there is a positive effect from the home-effect (posterior mean about 0.3), and this implies that if two teams are equally good (meaning that their attack and defence abilities almost coincide), assuming that the constant intercept $\mu \approx 0$, we get that the average number of goals for the home-team will be $\lambda_1 = \exp\{0.3\} \approx 1.35$, against $\lambda_2 = \exp\{0\} = 1$.

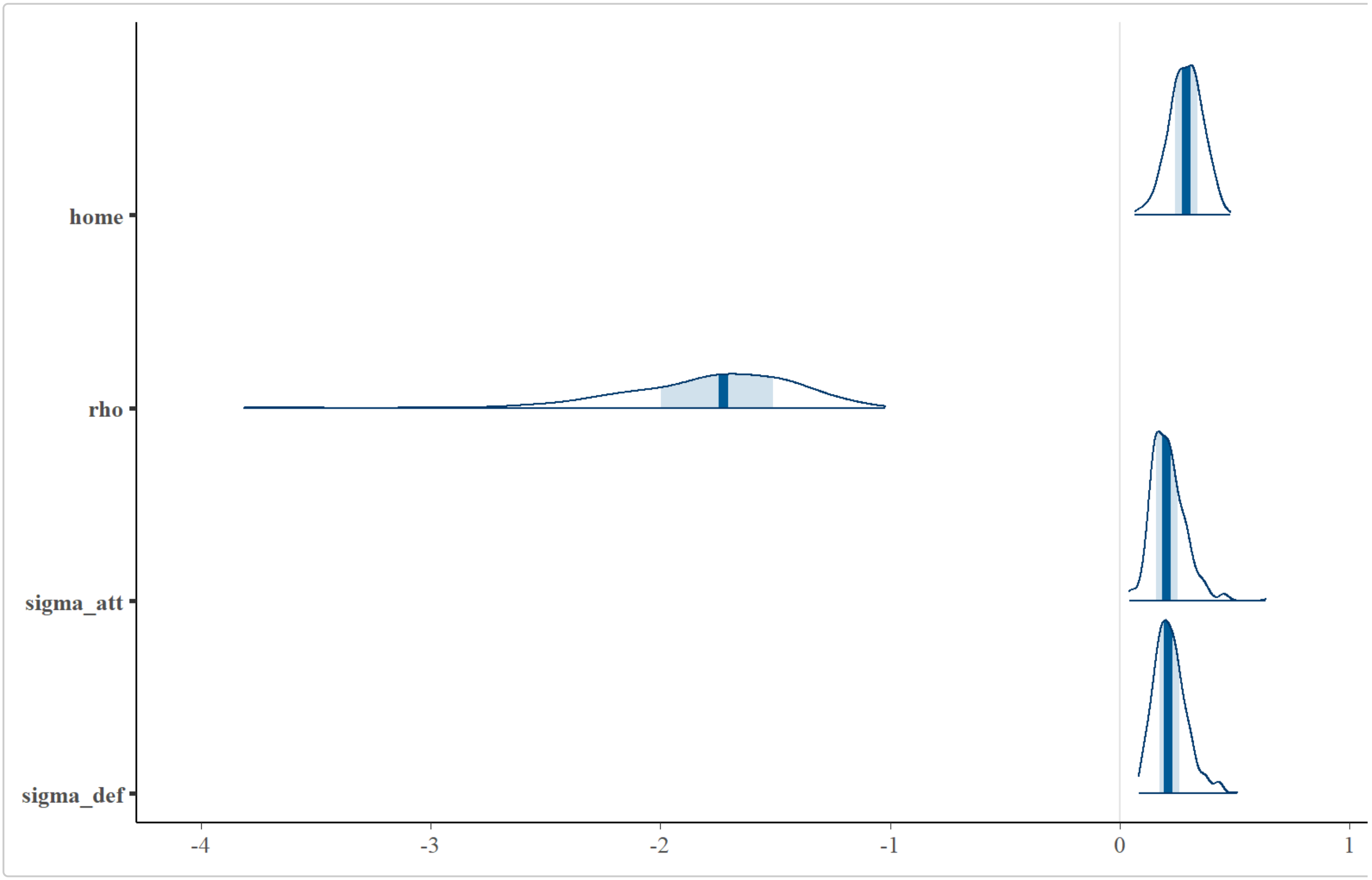
In the model above, we are assuming that the covariance λ_{3n} is constant and not depending on the match and/or on teams characteristics/further covariates:

$$\begin{aligned} \lambda_{3n} = & \exp\{\rho\} \\ \rho \sim & \mathcal{N}^+(0,1), \end{aligned}$$

where ρ is assigned an half-Gaussian distribution with standard deviation equal to 1. According to the fit above, this means that in the model above we get an estimate of $\lambda_{3n} = \exp\{-4.25\} \approx 0.014$, suggesting a low, despite non-null, amount of goals-correlation existing in the 2000/2001 Italian Serie A. Of course, in the next package's version, the user will be allowed to specify a more general linear predictor for $\log(\lambda_{3n})$, as outlined in the BP presentation above, along with some prior distributions for the parameters involved in the covariance formulation.

We can also depict the marginal posterior distributions for ρ (and eventually for the other fixed-effects parameters) using the `bayesplot` package for Bayesian visualizations:

```r
## Marginal posterior with bayesplot

posterior1 <- as.matrix(fit1_stan)
mcmc_areas(posterior1, regex_pars=c("home", "rho",
  "sigma_att", "sigma_def"))
```



We can also access the original BP Stan code by typing:

```r
### Model's code extraction

fit1_stan@stanmodel
#> S4 class stanmodel '02e61e6dbc2773b8f36869e8591b3648' coded as follows:
#> 
#>     functions{
#> 
#>       real bipois_lpmf(int[] r , real mu1,real mu2,real mu3) {
#>         real ss;
#>         real log_s;
#>         real mus;
#>         int  miny;
#> 
#>         miny = min(r[1], r[2]);
#> 
#>         ss = poisson_lpmf(r[1] | mu1) + poisson_lpmf(r[2] | mu2) -
#>           exp(mu3);
#>         if(miny > 0) {
#>           mus = -mu1-mu2+mu3;
#>           log_s = ss;
#>
#>           for(k in 1:miny) {
#>             log_s = log_s + log(r[1] - k + 1) + mus
#>             + log(r[2] - k + 1)
#>             - log(k);
#>             ss = log_sum_exp(ss, log_s);
#>           }
#>         }
#>         return(ss);
#>       }
#>
#>     }
#>     data{
#>       int N;   // number of games
#>       int y[N,2];
#>       int nteams;
#>       int team1[N];
#>       int team2[N];
#>       real ranking[nteams];
#>
#>       // priors part
#>       int<lower=1,upper=4> prior_dist_num;    // 1 gaussian, 2 t, 3 cauchy, 4 laplace
#>       int<lower=1,upper=4> prior_dist_sd_num; // 1 gaussian, 2 t, 3 cauchy, 4 laplace
#>
#>       real hyper_df;
#>       real hyper_location;
#>
#>       real hyper_sd_df;
#>       real hyper_sd_location;
#>       real hyper_sd_scale;
#>     }
#>     parameters{
#>       vector[nteams] att_raw;
#>       vector[nteams] def_raw;
#>       real<lower=0> sigma_att;
#>       real<lower=0> sigma_def;
#>       real beta;
#>       real rho;
#>       real home;
#>       real gamma;
#>     }
#>     transformed parameters{
#>       vector[nteams] att;
#>       vector[nteams] def;
#>       vector[3] theta[N];
#>
#>       for (t in 1:nteams){
#>         att[t] = att_raw[t]-mean(att_raw);
#>         def[t] = def_raw[t]-mean(def_raw);
#>       }
#>
#>       for (n in 1:N){
#>         theta[n,1] = exp(home+att[team1[n]]+def[team2[n]]+
#>                          (gamma/2)*(ranking[team1[n]]-ranking[team2[n]]));
#>         theta[n,2] = exp(att[team2[n]]+def[team1[n]]-
#>                          (gamma/2)*(ranking[team1[n]]-ranking[team2[n]]));
#>         theta[n,3] = exp(rho);
#>       }
#>     }
#>     model{
#>       // log-priors for team-specific abilities
#>       for (t in 1:(nteams)){
#>         if (prior_dist_num == 1){
#>           target+= normal_lpdf(att_raw[t]|hyper_location, sigma_att);
#>           target+= normal_lpdf(def_raw[t]|hyper_location, sigma_def);
#>         }
#>         else if (prior_dist_num == 2){
#>           target+= student_t_lpdf(att_raw[t]|hyper_df, hyper_location, sigma_att);
#>           target+= student_t_lpdf(def_raw[t]|hyper_df, hyper_location, sigma_def);
#>         }
#>         else if (prior_dist_num == 3){
#>           target+= cauchy_lpdf(att_raw[t]|hyper_location, sigma_att);
#>           target+= cauchy_lpdf(def_raw[t]|hyper_location, sigma_def);
#>         }
#>         else if (prior_dist_num == 4){
#>           target+= double_exponential_lpdf(att_raw[t]|hyper_location, sigma_att);
#>           target+= double_exponential_lpdf(def_raw[t]|hyper_location, sigma_def);
#>         }
#>       }
#>
#>
#>       // log-hyperpriors for sd parameters
#>       if (prior_dist_sd_num == 1 ){
#>         target+=normal_lpdf(sigma_att|hyper_sd_location, hyper_sd_scale);
#>         target+=normal_lpdf(sigma_def|hyper_sd_location, hyper_sd_scale);
#>       }
#>       else if (prior_dist_sd_num == 2){
#>         target+=student_t_lpdf(sigma_att|hyper_sd_df, hyper_sd_location, hyper_sd_scale);
#>         target+=student_t_lpdf(sigma_def|hyper_sd_df, hyper_sd_location, hyper_sd_scale);
#>       }
#>       else if (prior_dist_sd_num == 3){
#>         target+=cauchy_lpdf(sigma_att|hyper_sd_location, hyper_sd_scale);
#>         target+=cauchy_lpdf(sigma_def|hyper_sd_location, hyper_sd_scale);
#>       }
#>       else if (prior_dist_sd_num == 4){
#>         target+=double_exponential_lpdf(sigma_att|hyper_sd_location, hyper_sd_scale);
#>         target+=double_exponential_lpdf(sigma_def|hyper_sd_location, hyper_sd_scale);
#>       }
#>
#>       // log-priors fixed effects
#>       target+=normal_lpdf(rho|0,1);
#>       target+=normal_lpdf(home|0,5);
#>       target+=normal_lpdf(gamma|0,1);
#>
#>       // likelihood
#>       for (n in 1:N){
#>          target+=poisson_lpmf(y[n,1]|theta[n,1]+theta[n,3]);
#>          target+=poisson_lpmf(y[n,2]|theta[n,2]+theta[n,3]);
#>        //  target+=bipois_lpmf(y[n,]| theta[n,1],
#>           //                   theta[n,2], theta[n,3]);
#>       }
#>     }
#>     generated quantities{
#>       int y_rep[N,2];
#>       vector[N] log_lik;
#>       int diff_y_rep[N];
#>
#>       //in-sample replications
#>       for (n in 1:N){
#>         y_rep[n,1] = poisson_rng(theta[n,1]+theta[n,3]);
#>         y_rep[n,2] = poisson_rng(theta[n,2]+theta[n,3]);
#>         diff_y_rep[n] = y_rep[n,1] - y_rep[n,2];
#>         log_lik[n] = poisson_lpmf(y[n,1]|theta[n,1]+theta[n,3])+
#>                      poisson_lpmf(y[n,2]|theta[n,2]+theta[n,3]);
#>        //log_lik[n] =bipois_lpmf(y[n,]| theta[n,1],
#>                 //                 theta[n,2], theta[n,3]);
#>       }
#>     }
```

We fit now the same model under the MLE approach with Wald-type confidence intervals. We can then print the MLE estimates, e.g for the parameters ρ and home:

```
### Fit MLE models
## no dynamics, no predictions
## Wald intervals

fit1_mle <- mle_foot(data = italy_2000,
                     model="biv_pois",
                     interval = "Wald") # mle biv poisson
fit1_mle$home
#>      2.5% mle 97.5%
#> [1,]  0.2 0.3  0.39
```

We got a very similar estimate to the Bayesian model for the home-effect.

Changing default priors

One of the common practices in Bayesian statistics is to change the priors and perform some **sensitivity tests**. The default priors for the team-specific abilities and their related team-level standard deviations are:

$$\begin{aligned}\text{att}_t &\sim \mathcal{N}(\mu_{\text{att}}, \sigma_{\text{att}}),\\ \text{def}_t &\sim \mathcal{N}(\mu_{\text{def}}, \sigma_{\text{def}}),\\ \sigma_{\text{att}}, \sigma_{\text{def}} &\sim \textsf{Cauchy}^+(0,5),\end{aligned}$$

where $\textsf{Cauchy}^+$ denotes the half-Cauchy distribution with support $[0, +\infty)$. However, the user is free to elicit some different priors, possibly choosing one among the following distributions: Gaussian (`normal`), student-t (`student_t`), Cauchy (`cauchy`) and Laplace (`laplace`). The `prior` optional argument allows to specify the priors for the team-specific parameters att and def, whereas the optional argument `prior_sd` allows to assign a prior to the group-level standard deviations $\sigma_{\text{att}}, \sigma_{\text{def}}$. For instance, for each team $t, t = 1, \dots, T$, we could consider:

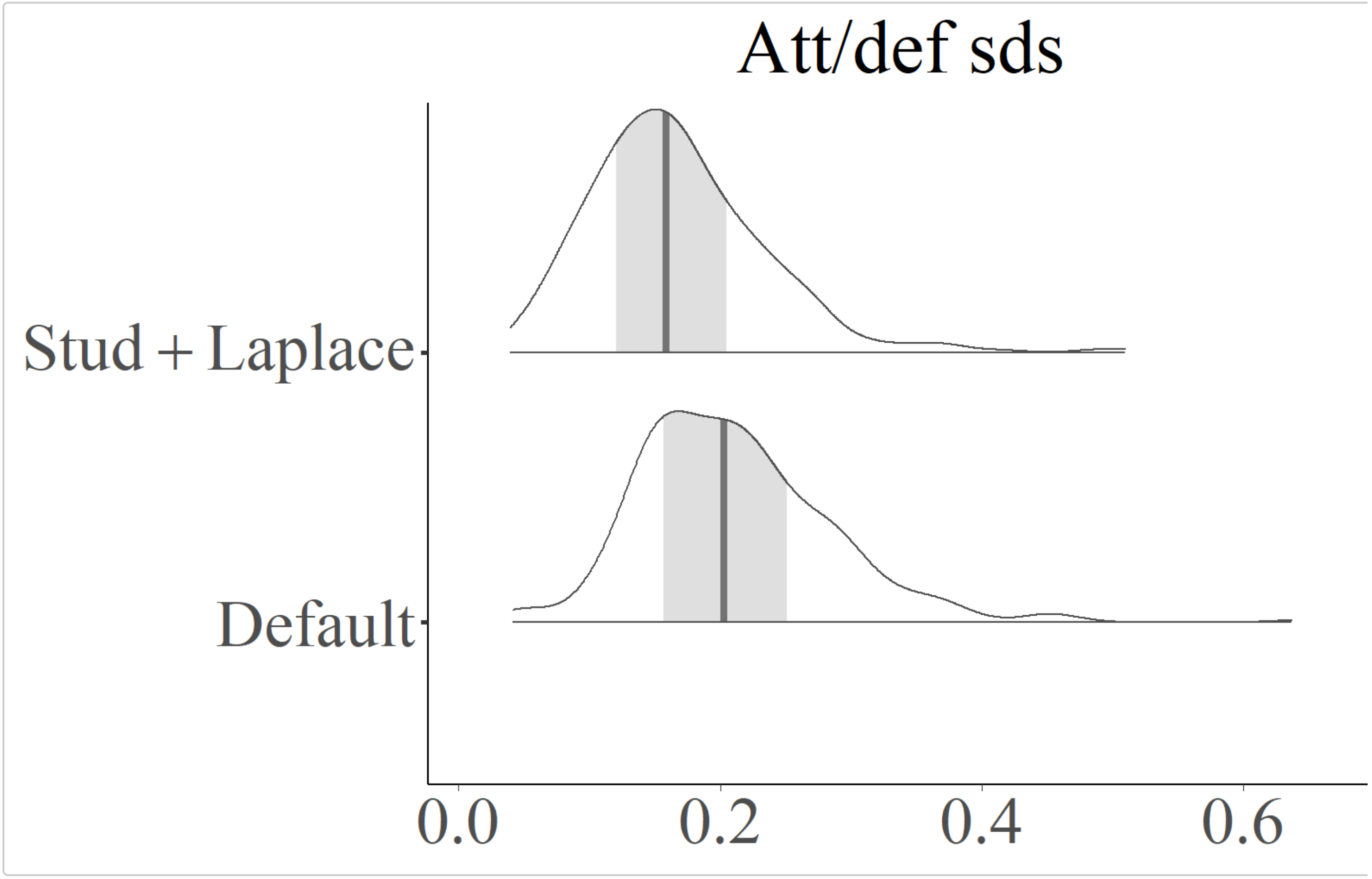
$$\begin{aligned} \text{att}_t &\sim t(4, \mu_{\text{att}}, \sigma_{\text{att}}), \\ \text{def}_t &\sim t(4, \mu_{\text{def}}, \sigma_{\text{def}}), \\ \sigma_{\text{att}}, \sigma_{\text{def}} &\sim \mathsf{Laplace}^+(0,1), \end{aligned}$$

where $t(\text{df}, \mu, \sigma)$ denotes a student-t distribution with df degrees of freedom, location μ and scale σ, whereas $\mathsf{Laplace}^+$ denotes a half-Laplace distribution.

```r
### Fit Stan models
## changing priors
## student-t for team-specific abilities, laplace for sds

fit1_stan_t <- stan_foot(data = italy_2000,
                         model="biv_pois",
                         chains = 4,
                         prior = student_t(4,0,NULL),
                         prior_sd = laplace(0,1),
                         #cores = 4,
                         iter = n_iter) # biv poisson
```

Then, we can compare the marginal posteriors from the two models, the one with Gaussian team-specific abilities and the default $\mathsf{Cauchy}(0,5)$ for the team-level sds, and the other one specified above, with student-t distributed team-specific abilities and the $\mathsf{Laplace}^+(0,1)$ for the team-level sds. We depict here the comparison for the attack team-level sds only:

```r
## comparing posteriors

posterior1_t <- as.matrix(fit1_stan_t)
model_names <- c("Default", "Stud+Laplace")
color_scheme_set(scheme = "gray")
gl_posterior <- cbind(posterior1[,"sigma_att"],
                      posterior1_t[,"sigma_att"])
colnames(gl_posterior)<-c("sigma_att", "sigma_att_t")
mcmc_areas(gl_posterior, pars=c("sigma_att", "sigma_att_t"))+
  xaxis_text(on =TRUE, size=ggplot2::rel(2.9))+
  yaxis_text(on =TRUE, size=ggplot2::rel(2.9))+
  scale_y_discrete(labels = ((parse(text= model_names))))+
  ggtitle("Att/def sds")+
  theme(plot.title = element_text(hjust = 0.5, size =rel(2.6)))
```



The student+laplace prior induces a lower amount of group-variability in the σ_{att} marginal posterior distribution (then, a larger shrinkage towards the grand mean μ_{att}).

When specifying the prior for the team-specific parameters through the argument `prior`, you are not allowed to fix the group-level standard deviations $\sigma_{\text{att}}, \sigma_{\text{def}}$ to some numerical values. Rather, they need to be assigned a reasonable prior distribution. For such reason, the most appropriate specification for the `prior` argument is `prior = 'dist'(0, NULL)`, where the scale argument is set to `NULL` (otherwise, a warning message is occurring).

Dynamic fit

A structural limitation in the previous models is the assumption of static team-specific parameters, namely teams are assumed to have a constant performance across time, as determined by the attack and defence abilities (att, def). However, teams' performance tend to be *dynamic* and change across different years, if not different weeks. Many factors contribute to this football aspect:

i. teams act during the summer/winter players' transfermarket, by dramatically changing their rosters;
ii. some teams' players could be injured in some periods, by affecting the global quality of the team in some matches;
iii. coaches could be dismissed from their teams due to some non satisfactory results;
iv. some teams could improve/worsen their attitudes due to the so-called turnover;

and many others. Perhaps, we could assume **dynamics** in the attach/defence abilities as in Owen (2011) and Egidi, Pauli, and Torelli (2018) in terms of weeks or seasons. In such framework, for a given number of times $1, \ldots, \mathcal{T}$, the models above would be unchanged, but the priors for the abilities parameters at each time $\tau, \tau = 2, \ldots, \mathcal{T}$, would be **auto-regressive** of order 1:

$$\begin{aligned}\text{att}_{t,\tau} &\sim \mathcal{N}(\text{att}_{t,\tau-1}, \sigma_{\text{att}})\\ \text{def}_{t,\tau} &\sim \mathcal{N}(\text{def}_{t,\tau-1}, \sigma_{\text{def}}),\end{aligned}$$

whereas for $\tau = 1$ we have:

$$\text{att}_{t,1} \sim \mathcal{N}(\mu_{\text{att}}, \sigma_{\text{att}})$$
$$\text{def}_{t,1} \sim \mathcal{N}(\mu_{\text{def}}, \sigma_{\text{def}}),$$

with hyperparameters $\mu_{\text{att}}, \mu_{\text{def}}, \sigma_{\text{att}}, \sigma_{\text{def}}$ and with the standard deviations capturing the time's/evolution's variation of both the teams' skills (here assumed to be constant across time and teams). Of course, the identifiability constraint must be imposed for each time τ.

We can use the `dynamic_type` argument in the `stan_foot` function, with possible options `'seasonal'` or `'weekly'` in order to consider more seasons (no examples are given in this course) or more week-times within a single season, respectively. Let's fit a weekly-dynamic parameters model on the Serie A 2000/2001 season:

```
### Fit Stan models
## seasonal dynamics, no predictions
## 4 Markov chains, 'n_iter' iterations each

fit2_stan <- stan_foot(data = italy_2000,
                       model="biv_pois",
                       dynamic_type ="weekly",
                       #cores = 4,
                       iter = n_iter) # biv poisson
print(fit2_stan, pars =c("home", "rho", "sigma_att",
                        "sigma_def"))
```

From the printed summary, we may note that the degree of goals' correlation seems to be again very small here. Moreover, the Gelman-Rubin statistic for σ_{att} is relatively high, whereas the effective sample sizes for σ_{att} and σ_{def} are quite low. This is suggesting possible inefficiencies during the HMC sampling and that a *model-reparametrization* could be suited and effective at this stage. Another option is to play a bit with the prior specification for σ_{att} and σ_{def}, for instance by specifying a prior inducing less shrinkage in the team-specific abilities.

To deal with these issues and broaden the set of candidate models, let's fit also a dynamic double-Poisson model with the `double_pois` option for the argument `model`:

```
### Fit Stan models
## weekly dynamics, no predictions
## 4 chains, 'n_iter' iterations each

fit3_stan <- stan_foot(data = italy_2000,
                       model="double_pois",
                       dynamic_type = "weekly",
                       #cores = 4,
                       iter = n_iter)  # double poisson
print(fit3_stan, pars =c("home", "sigma_att",
                        "sigma_def"))
```

The fitting problems mentioned above remain also for the double Poisson model…Thus, it's time to play a little bit with the prior distributions. Also in the dynamic approach we can change the default priors for the team-specific abilities and their standard deviations, respectively, through the optional arguments `prior` and `prior_sd`. The specification follows almost analogously the static case: with the first argument we may specify the prior's family for the team-specific abilities and the specific priors for $\text{att}_{t,1}, \text{def}_{t,1}$ along with the hyper-prior location $\mu_{\text{att}}, \mu_{\text{def}}$, whereas σ_{att} and σ_{def} need to be assigned some proper prior distribution. Assume to fit the same double Poisson model, but here we suppose student-t distributed team-specific abilities with 4 degrees of freedom to eventually capture more extreme team-specific abilities (more variability, i.e. less shrinkage), along with a $\text{Cauchy}^+(0, 25)$ for their standard deviations (to better capture a possible larger evolution variability):

$$\begin{aligned} \text{att}_{t,\tau} &\sim t(4, \text{att}_{t,\tau-1}, \sigma_{\text{att}}) \\ \text{def}_{t,\tau} &\sim t(4, \text{def}_{t,\tau-1}, \sigma_{\text{def}}) \\ \sigma_{\text{att}}, \sigma_{\text{def}} &\sim \text{Cauchy}^+(0, 25), \end{aligned}$$

```
### Fit Stan models
## weekly dynamics, no predictions
## 4 chains, 'n_iter' iterations each

fit3_stan_t <- stan_foot(data = italy_2000,
                model="double_pois",
                prior = student_t(4,0, NULL), # 4 df
                prior_sd = cauchy(0,25),
                dynamic_type = "weekly",
                #cores = 4,
                iter = n_iter)  # double poisson
print(fit3_stan_t, pars =c("home", "sigma_att",
                           "sigma_def"))
```

As we may conclude, the situation has been only slightly improved.

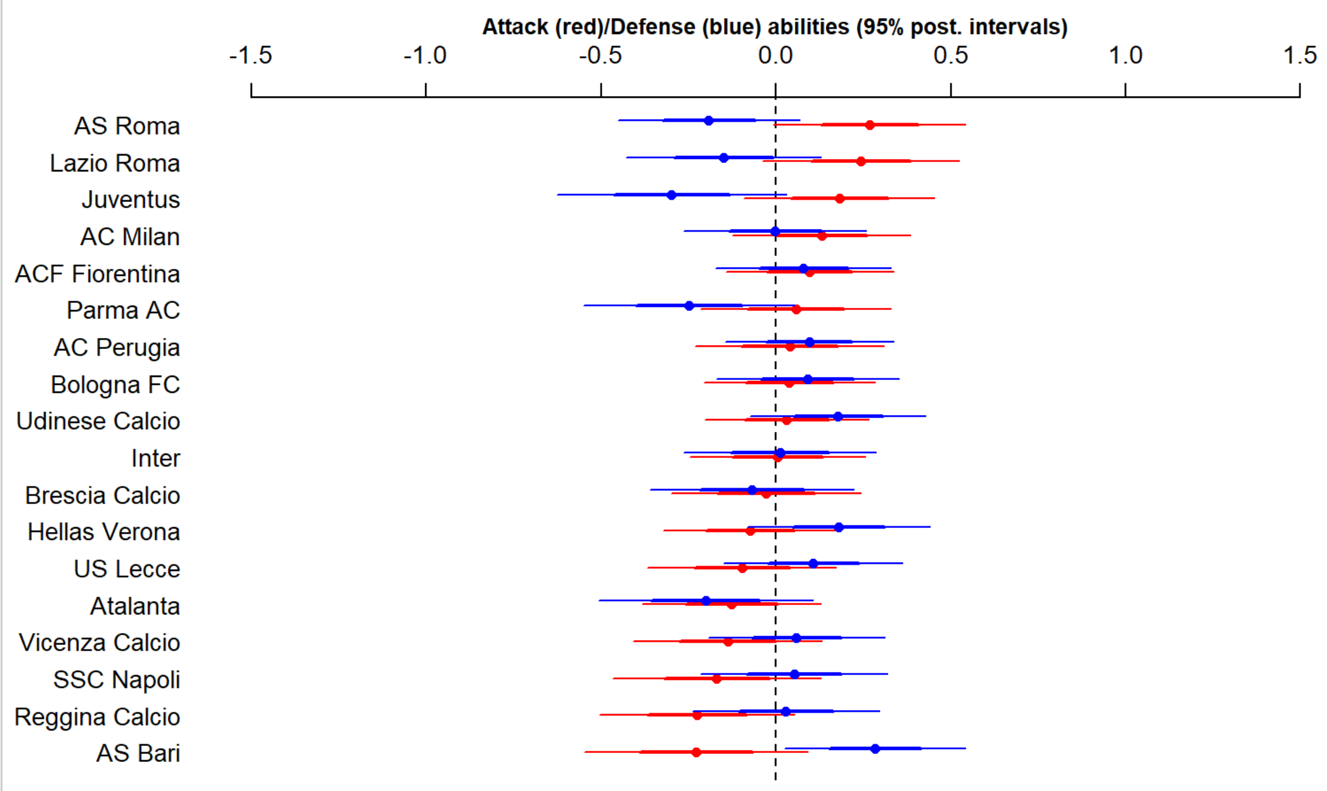
Model estimates and visualization tools

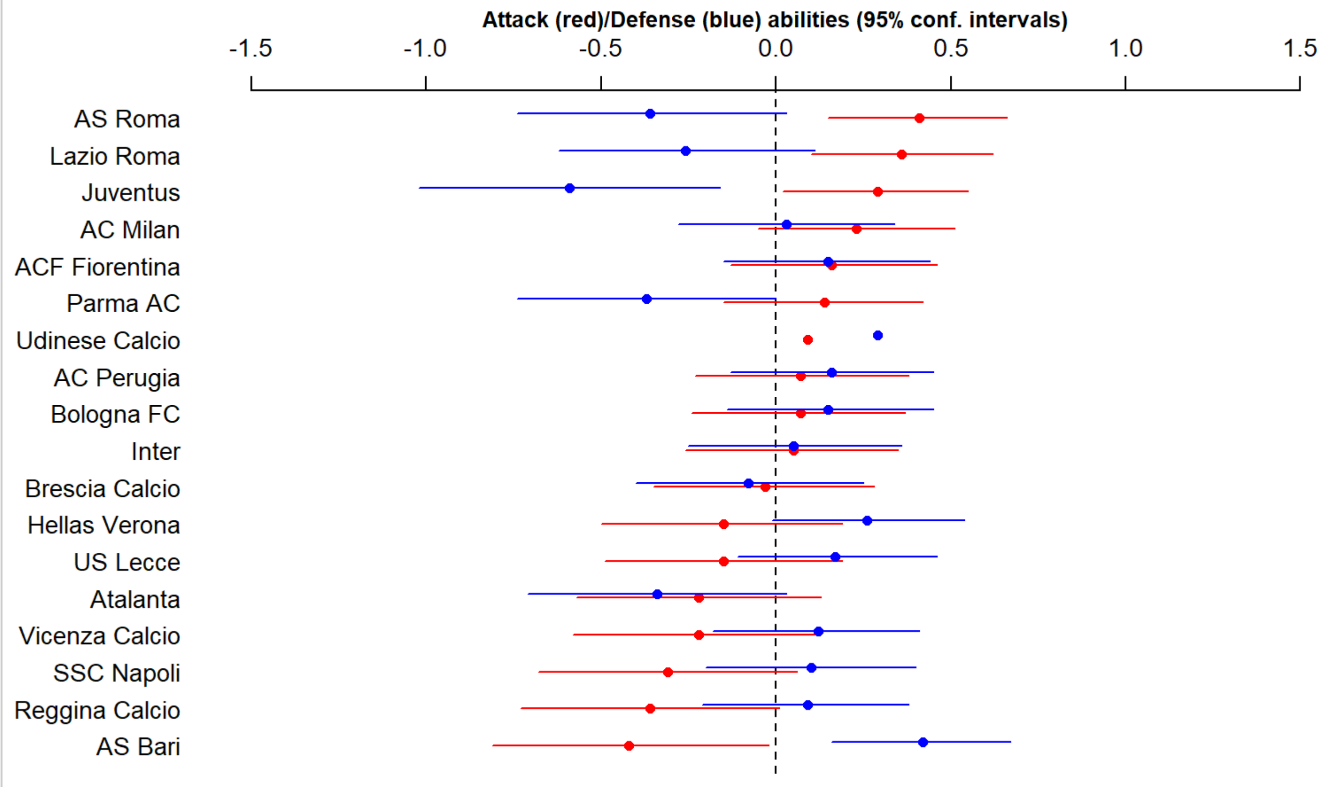
Plotting and interpreting team-specific abilities

Once the model has been fitted, there is a large amount of interesting summaries to explore. The function `foot_abilities` allows to depict posterior/confidence intervals for global attack and defense abilities on the considered data (attack abilities are plotted in red, whereas defense abilities in blue colors). The *higher the attack and the lower the defence* for a given team, and *the better is the overall team's strength*.

We can produce the team-specific abilities for the two static fits above, `fit1_mle` (MLE) and `fit1_stan` (Bayes), with red bars for the attack and blue bars for the defence, respectively:

```
## Plotting abilities: credible and confidence 95% intervals

foot_abilities(object = fit1_stan, data = italy_2000, cex.var = 1)
foot_abilities(object = fit1_mle, data = italy_2000, cex.var = 1)
```

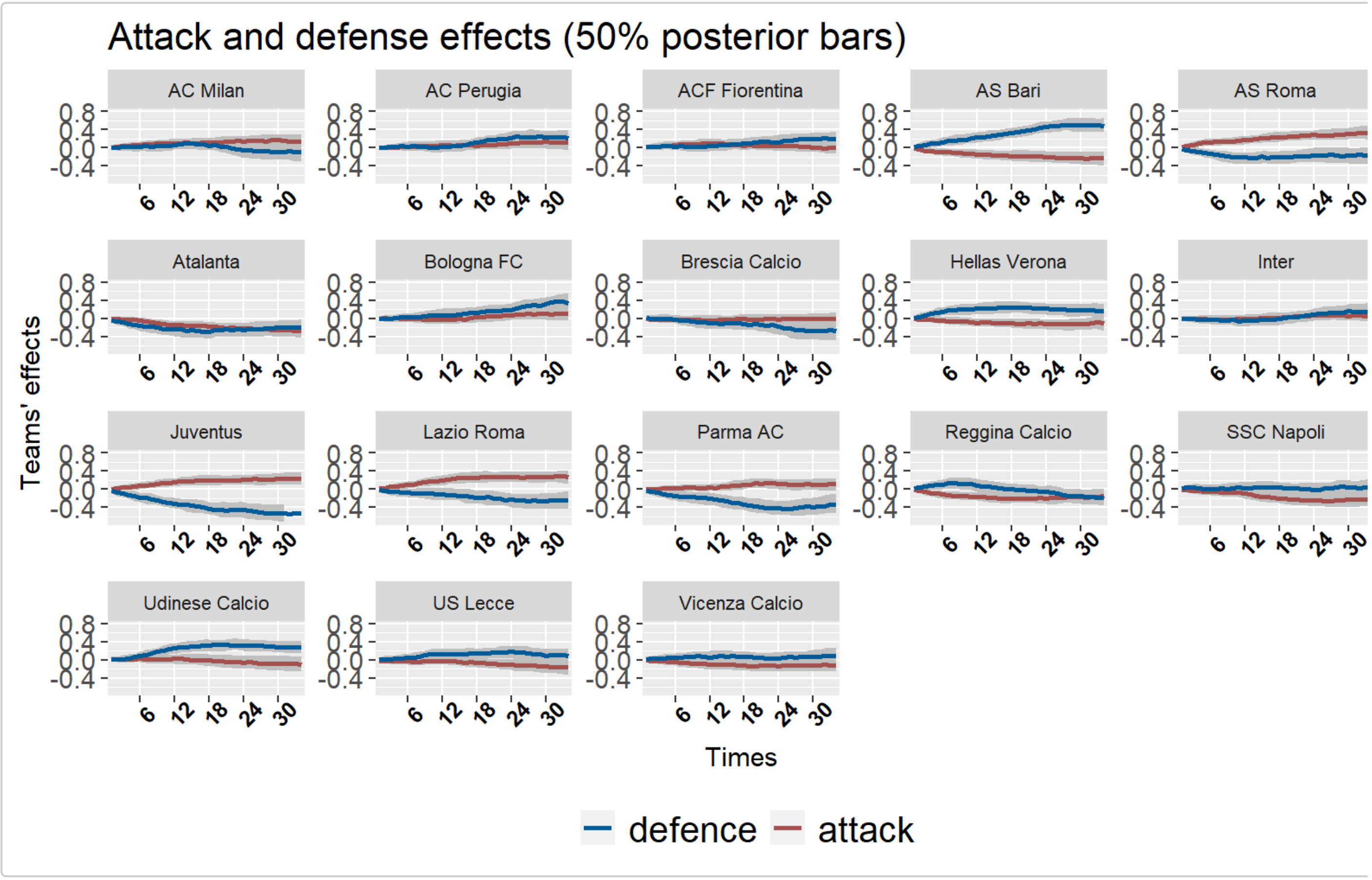




AS Roma, the team winning the Serie A 2000/2001, is associated with the highest attack ability and the lowest defence ability according to both the models. In general, the models seem to well capture the static abilities: AS Roma, Lazio Roma and Juventus (1st, 3rd and 2nd at the end of that season, respectively) are rated as the best teams in terms of their abilities, whereas AS Bari, SSC Napoli and Vicenza Calcio (all relegated at the end of the season) have the worst abilities.

We can also depict the team-specific dynamic plots for the dynamic models:

```
## Plotting abilities: credible and confidence 95% intervals

foot_abilities(fit2_stan, italy_2000)
```



As we can see, dynamic abilities naturally evolve over the time: better teams (AS Roma, Lazio Roma, Juventus, Parma) are associated with increasing attack abilities and decreasing defence abilities, whereas the worst ones (AS Bari, SSC NApoli, and Hellas Verona) exhibit decreasing attacking skills and increasing defensive skills. The reason for these increasing/decreasing behaviours is straightforward: at the beginning, all the attack/defence parameters have been initialized to have location equal to 0. The user is free to change the location, and in the final package's version he will also have the possibility to elicit different team-specific hyper-prior locations.

Model checking

Checking the model fit is a relevant and vital statistical task. To this purpose, we can evaluate hypothetical replications $\mathcal{D}^{\text{rep}}$ under the **posterior predictive distribution**

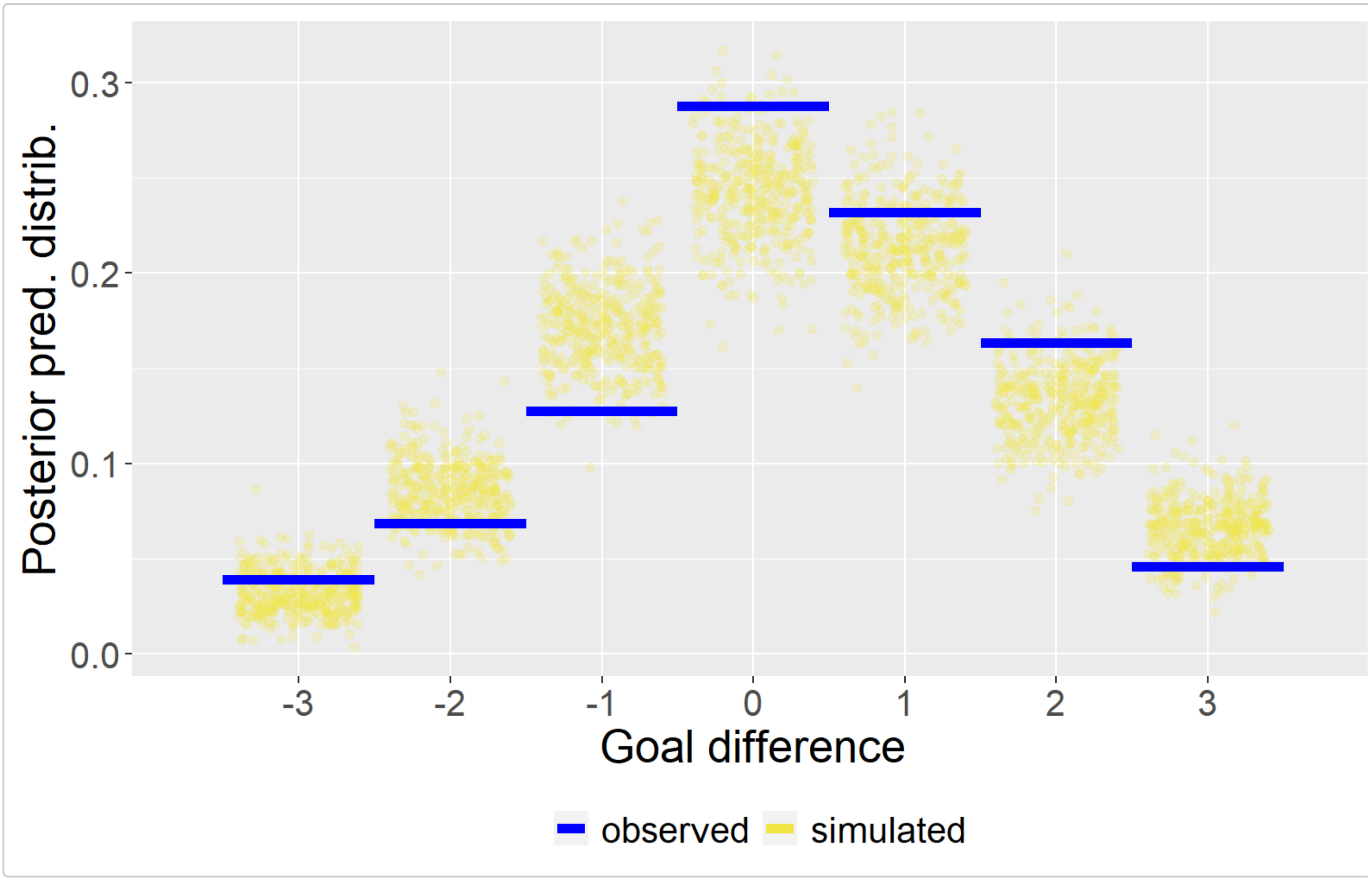
$$p(\mathcal{D}^{\text{rep}}|\mathcal{D}) = \int p(\mathcal{D}^{\text{rep}}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta},$$

and check whether these replicated values are somehow close to the observed data $\mathcal{D}$. These methods comparing hypothetical replications with the observed data are named **posterior predictive checks** and have great theoretical and applied appeal in Bayesian inference. See Gelman et al. (2014) for an overview.

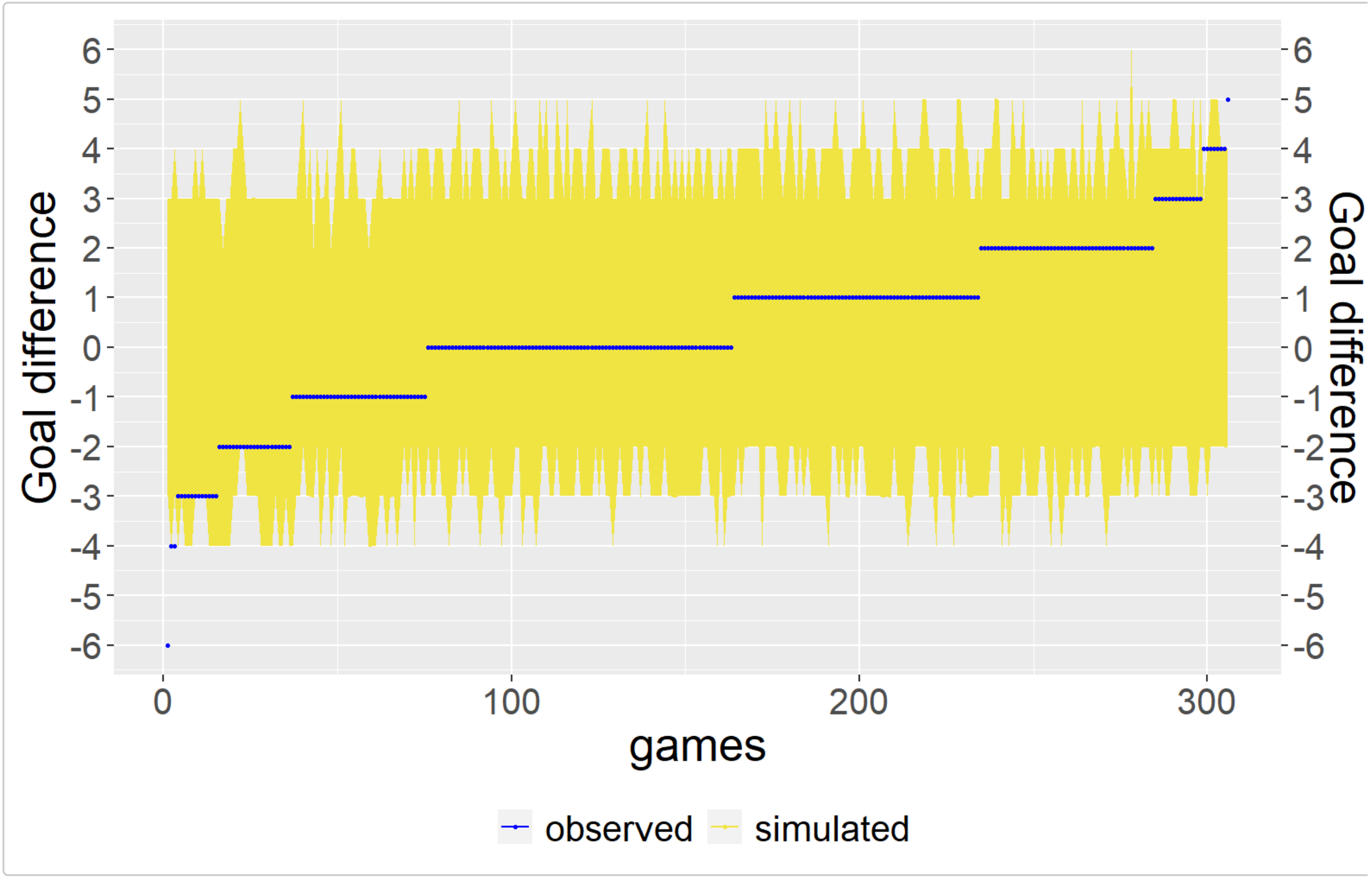
The function `pp_foot` allows to obtain:

- an aggregated plot depicting the observed frequencies of the observed goal differences $Z_n = X_n - Y_n,\ n = 1, \ldots, N$ plotted against the replicated ones;
- a visualization of the match-ordered goal differences along with their 50% and 95% credible intervals.

```
## PP checks: aggregated goal's differences and ordered goal differences

pp_foot(data = italy_2000, object = fit1_stan,
        type = "aggregated")
#> $pp_plot
```



```
#>
#> $pp_table
#>   goal diff. Bayesian p-value
#> 1         -3            0.322
#> 2         -2            0.865
#> 3         -1            0.983
#> 4          0            0.050
#> 5          1            0.242
#> 6          2            0.082
#> 7          3            0.930

pp_foot(data = italy_2000, object = fit1_stan,
        type = "matches")
#> $pp_plot
```



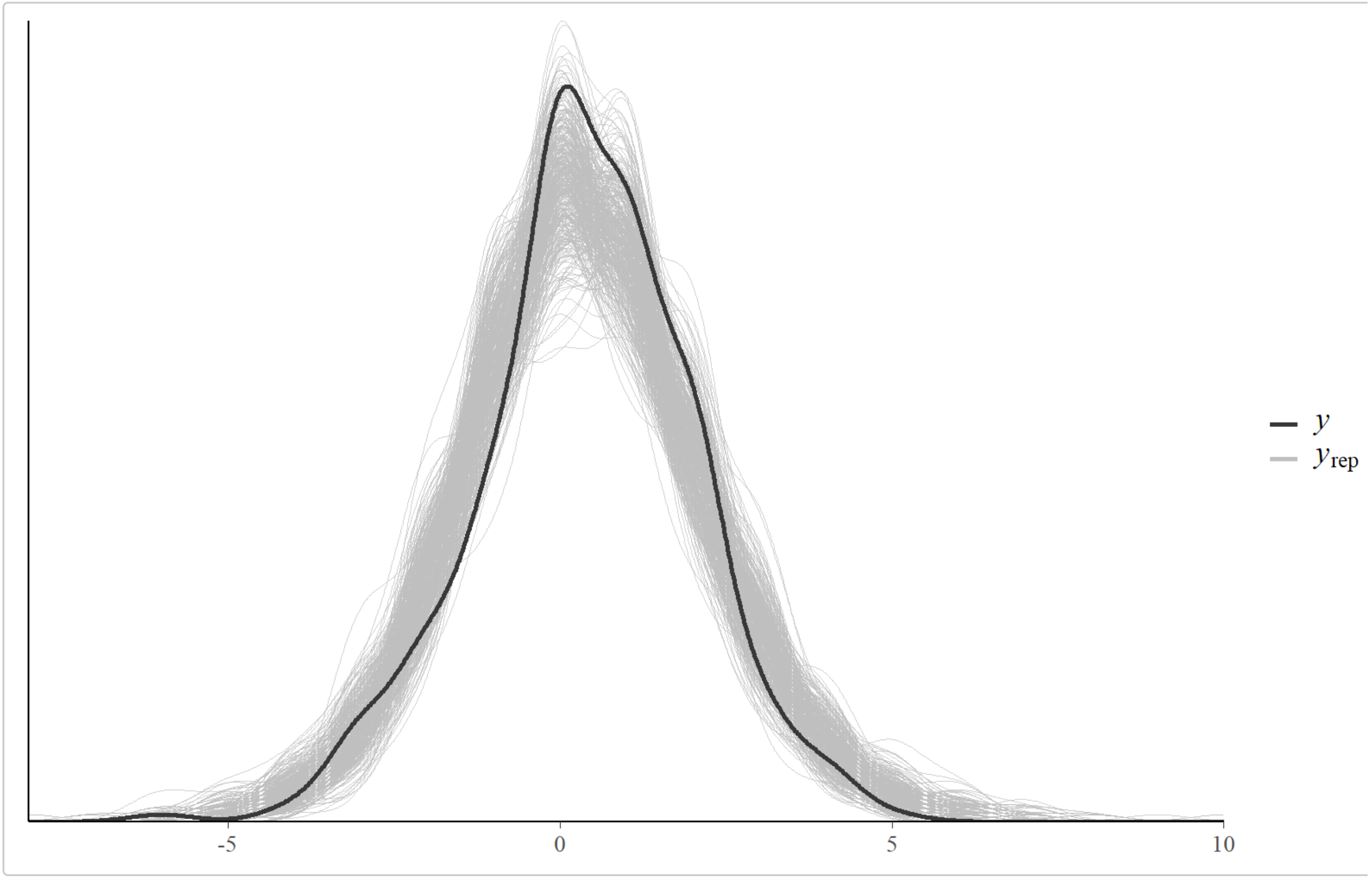
```
#>
#> $pp_table
#>   1-alpha emp. coverage
#> 1    0.95          0.951
```

The aggregated goal difference frequencies seem to be decently captured by the model's replications: in the first plot, blue horizontal lines denote the observed goal differences frequencies registered in the dataset, whereas yellow jittered points denote the correspondent replications. Goal-difference of 0, corresponding to the draws occurrences, is only slightly underestimated by the model. However, in general there are no particular clues of model's misfit.

In the second plot, the ordered observed goal differences are plotted against their replications (50% and 95% credible intervals), and from this plot also we do not have particular signs of model's misfits.

Other useful PP checks, such as the overlap between data density and replicated data densities to check eventual inconsistencies, can be obtained through the standard use of the `bayesplot` package, in this case providing an approximation to a continuous distribution using an input kernel choice (`bw = 0.5` in the `ppc_dens_overlay` used below):

```
## PPC densities overlay with the bayesplot package

# extracting the replications

sims <-rstan::extract(fit1_stan)
goal_diff <- italy_2000$hgoal-italy_2000$vgoal

# plotting data density vs replications densities

ppc_dens_overlay(goal_diff, sims$y_rep[,,1]-sims$y_rep[,,2], bw = 0.5)
```



From this plot above we have the empirical confirmation that the goal difference is well captured by the static bivariate Poisson model.

Predictions and predictive accuracy

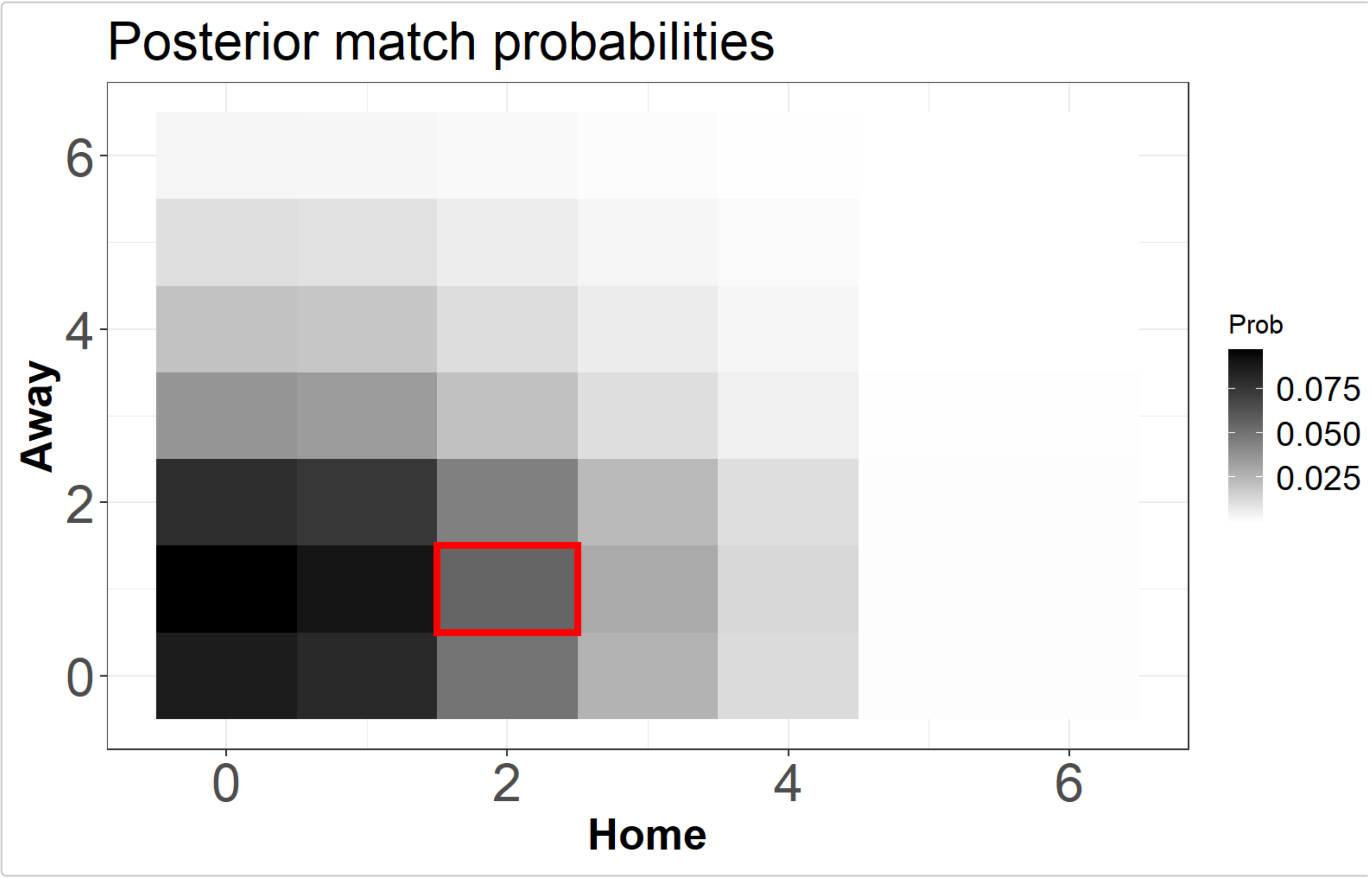
Posterior out-of-sample probabilities

The hottest feature in sports analytics is to obtain future predictions. By considering the **posterior predictive distribution** for future and observable data $\tilde{\mathcal{D}}$, we acknowledge the whole model's prediction uncertainty (which propagates from the posterior model's uncertainty) and we can then generate observable values $\tilde{D}$ conditioned on the posterior model's parameters estimates:

$$p(\tilde{\mathcal{D}}|\mathcal{D}) = \int p(\tilde{\mathcal{D}}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}.$$

We may then predict test set matches by using the argument `predict` of the `stan_foot` function, for instance considering the last four weeks of the 2000/2001 season as the test set, and then computing posterior-results probabilities using the function `foot_prob` for two teams belonging to the test set, such as Reggina Calcio and AC Milan:

```
### Fit Stan models
## weekly dynamics, predictions of last four weeks
## 4 chains 'n_iter' iterations each

fit4_stan <- stan_foot(data = italy_2000,
                       model="biv_pois",
                       predict = 36,
                              dynamic_type = "weekly",
                              #cores = 4,
                              iter = n_iter)  # biv poisson
foot_prob(object = fit4_stan, data = italy_2000,
          home_team = "Reggina Calcio",
          away_team= "AC Milan")
```



Darker regions are associated with higher posterior probabilities, whereas the red square corresponds to the actual observed result, 2-1 for Reggina Calcio. According to the posterior-results probabilities, this final observed result had in principle about a 5% probability to happen! (Remember, football is about rare events…).

Home-win posterior probabilities

We can also use the out-of-sample posterior-results probabilities to compute some aggregated **home/draw/loss** probabilities (based then on the S draws from the MCMC method) for a given match:

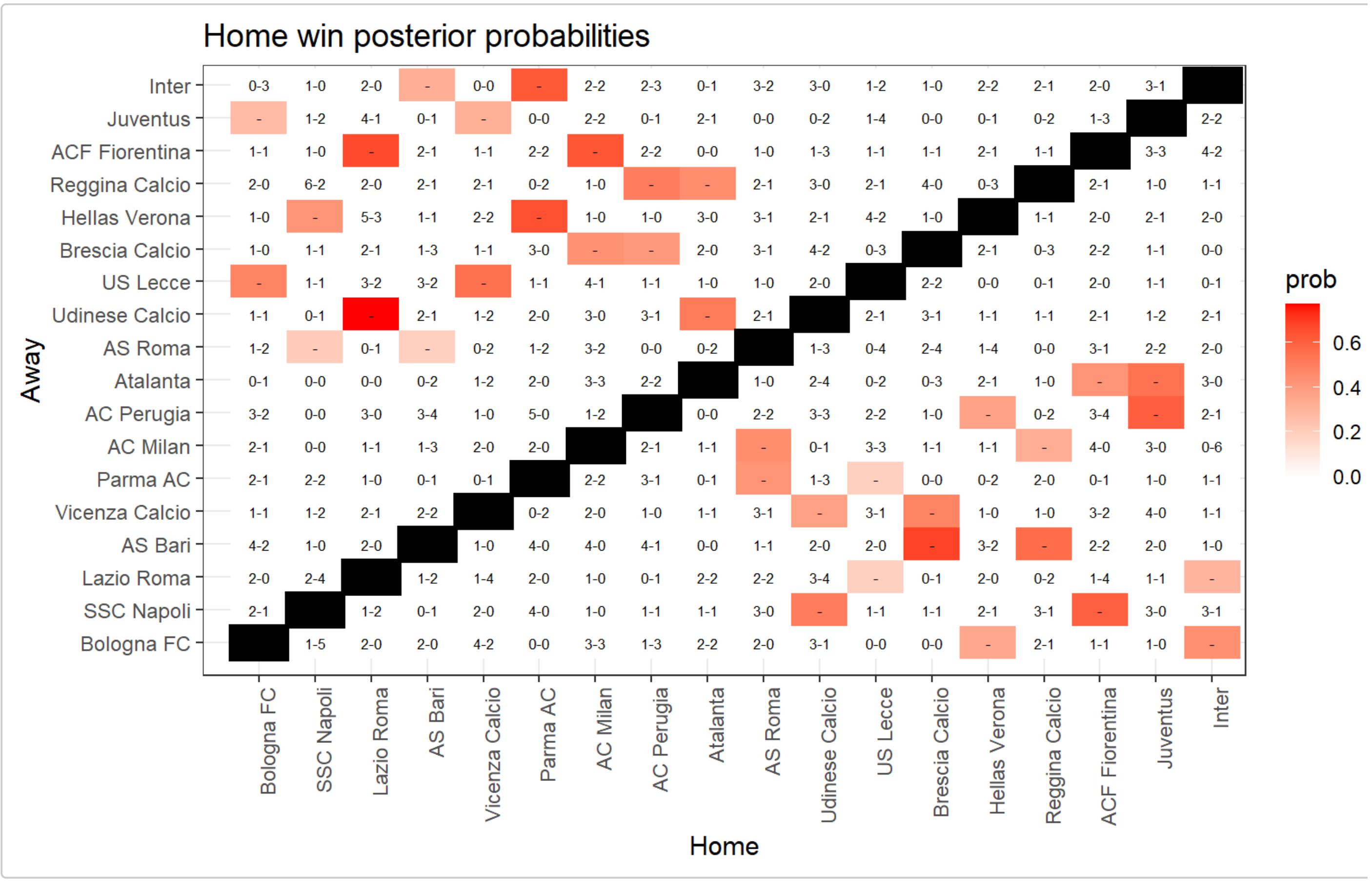
$$p_{\text{home}} = \Pr(X > Y) = \frac{1}{S}\sum_{s=1}^{S} |\tilde{x}^{(s)} > \tilde{y}^{(s)}|$$

$$p_{\text{draw}} = \Pr(X = Y) = \frac{1}{S}\sum_{s=1}^{S} |\tilde{x}^{(s)} = \tilde{y}^{(s)}|$$

$$p_{\text{loss}} = \Pr(X < Y) = \frac{1}{S}\sum_{s=1}^{S} |\tilde{x}^{(s)} < \tilde{y}^{(s)}|,$$

where $(\tilde{x}^{(s)}, \tilde{y}^{(s)})$ represents the s-th MCMC pair of the future home goals and away goals for a given match, respectively. According to this scenario, we can depict the home-win posterior probabilities of a given test set through the function `foot_round_robin`:

```r
## Home win out-of-sample probabilities

foot_round_robin(data = italy_2000, object = fit4_stan)
#> $round_plot
```



```
#>
#> $round_table
#> # A tibble: 36 x 4
#>    Home           Away           Home_prob Observed
#>    <chr>          <chr>          <chr>     <chr>
#>  1 Hellas Verona  Bologna FC     0.35      -
#>  2 Inter          Bologna FC     0.438     -
#>  3 Udinese Calcio SSC Napoli     0.507     -
#>  4 ACF Fiorentina SSC Napoli     0.603     -
#>  5 US Lecce       Lazio Roma     0.192     -
#>  6 Inter          Lazio Roma     0.3       -
#>  7 Brescia Calcio AS Bari        0.685     -
#>  8 Reggina Calcio AS Bari        0.565     -
#>  9 Udinese Calcio Vicenza Calcio 0.378     -
#> 10 Brescia Calcio Vicenza Calcio 0.482     -
#> # ... with 26 more rows
```

Red cells denote more likely home-wins (close to 0.6), such as: Lazio Roma - Fiorentina (observed result: 3-0, home win), Lazio Roma - Udinese (observed result: 3-1, home win), Juventus - AC Perugia (observed result: 1-0, home win), Brescia Calcio - AS Bari (observed result: 3-1, home win). Conversely, lighter cells denote more likely away wins (close to 0.6), such as: AS Bari - AS Roma (observed result: 1-4, away win), AS Bari - Inter (observed result: 1-2, away win).

Finally, computations of well-known predictive measures on test set data such as the **Brier score** and the **Epstein probability score** will be included in the next package's version, however they could be easily computed by the users.

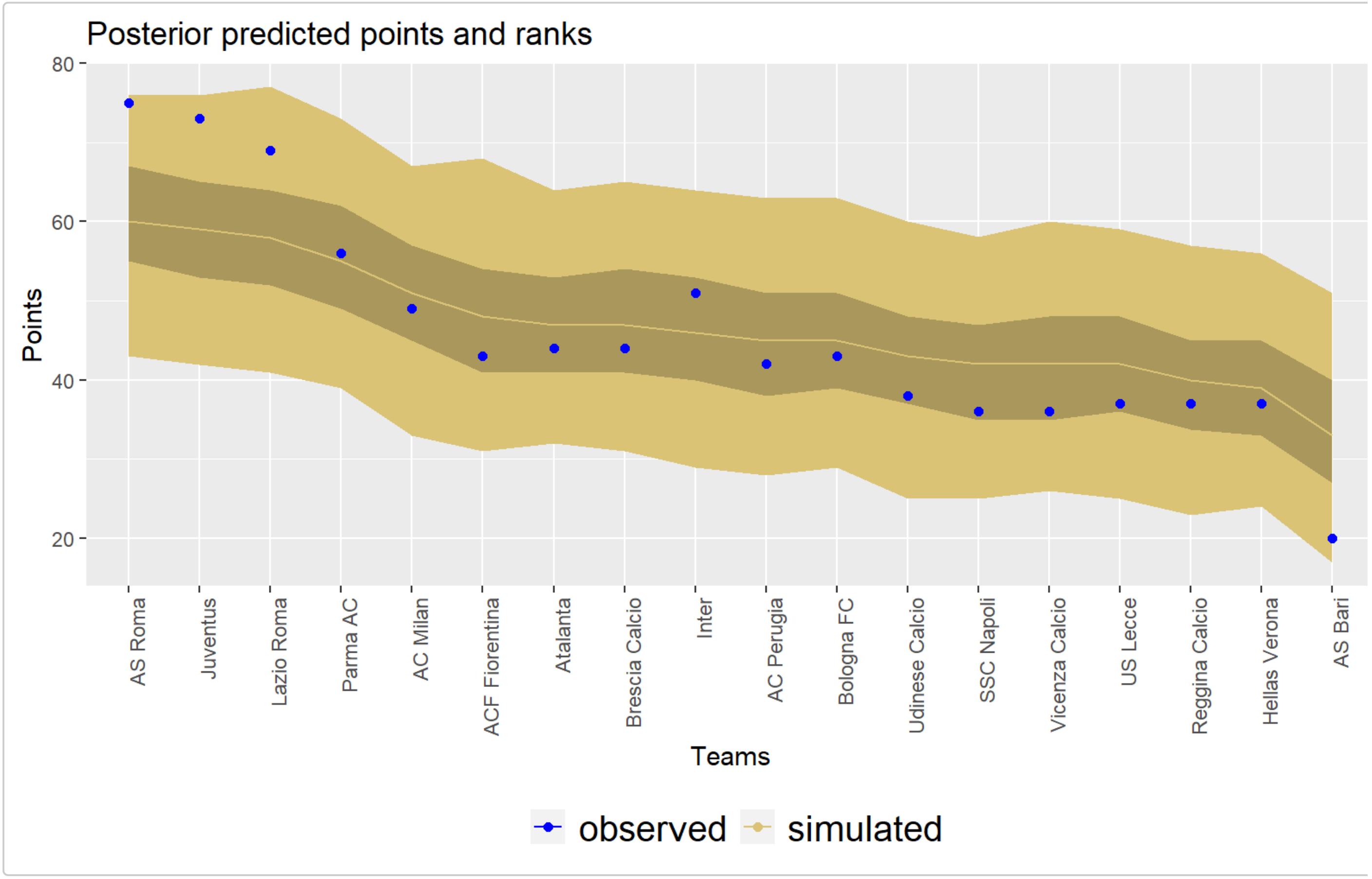
Rank-league reconstruction

Statisticians and football amateurs are much interested in the final rank-league predictions. However, predicting the final rank position (along with the teams' points) is often assimilated to an *oracle*, rather than a coherent statistical procedure.

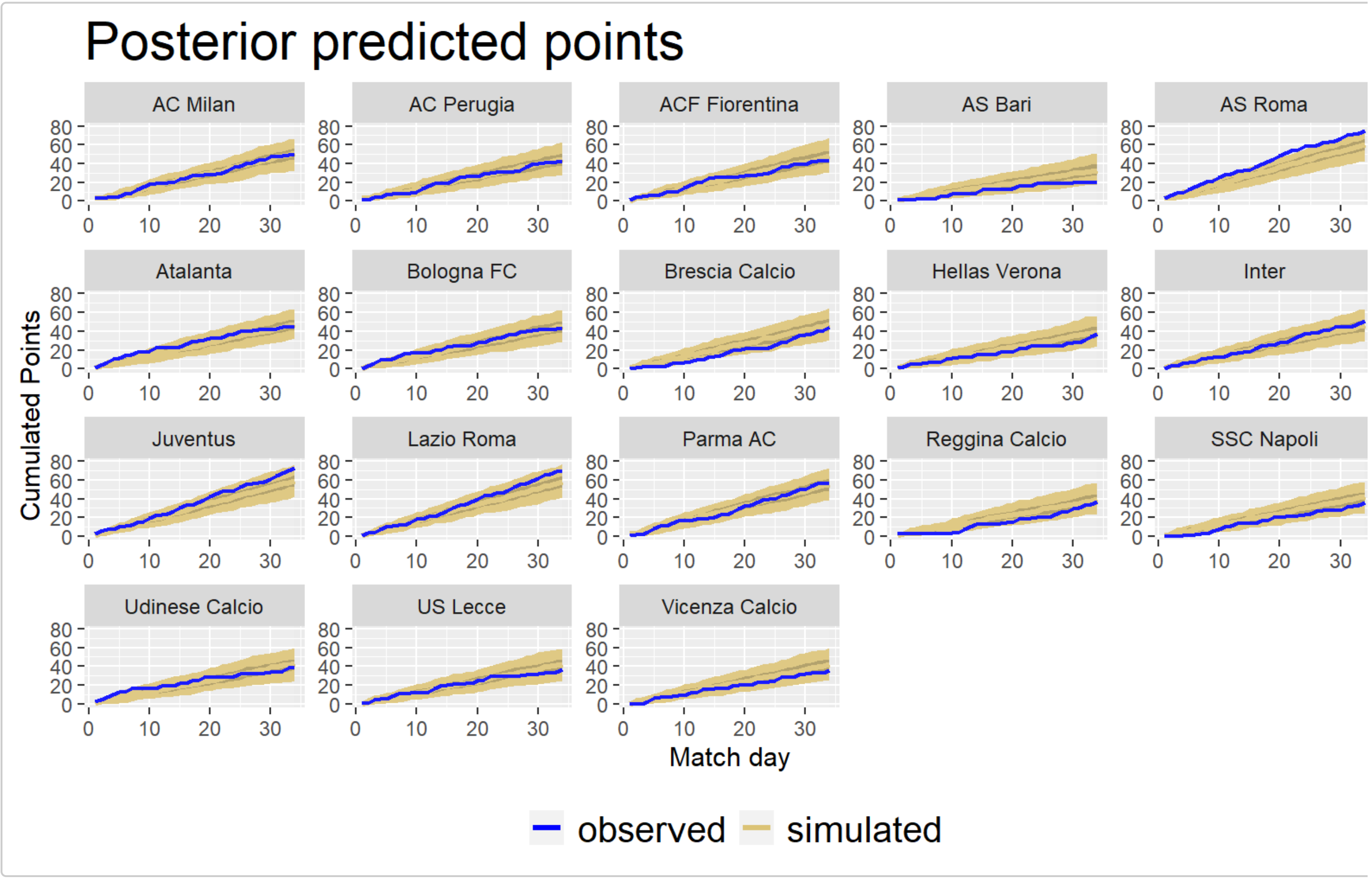
We can provide here:

- **rank-league reconstruction** for the first model `fit1_stan` by using the **in-sample** replications $\mathcal{D}^{\text{rep}}$ (yellow ribbons for the credible intervals, solid blue lines for the observed cumulated points):

```
## Rank league reconstruction

# aggregated plot

foot_rank(data = italy_2000, object = fit1_stan)
#> $rank_table
#>             teams mid obs lo hi
#> 1         AS Roma  60  75 55 67
#> 2        Juventus  59  73 53 65
#> 3      Lazio Roma  58  69 52 64
#> 4        Parma AC  55  56 49 62
#> 5        AC Milan  51  49 45 57
#> 6  ACF Fiorentina  48  43 41 54
#> 7        Atalanta  47  44 41 53
#> 8  Brescia Calcio  47  44 41 54
#> 9           Inter  46  51 40 53
#> 10     AC Perugia  45  42 38 51
#> 11     Bologna FC  45  43 39 51
#> 12 Udinese Calcio  43  38 37 48
#> 13     SSC Napoli  42  36 35 47
#> 14 Vicenza Calcio  42  36 35 48
#> 15       US Lecce  42  37 36 48
#> 16 Reggina Calcio  40  37 34 45
#> 17  Hellas Verona  39  37 33 45
#> 18        AS Bari  33  20 27 40
#>
#> $rank_plot
```



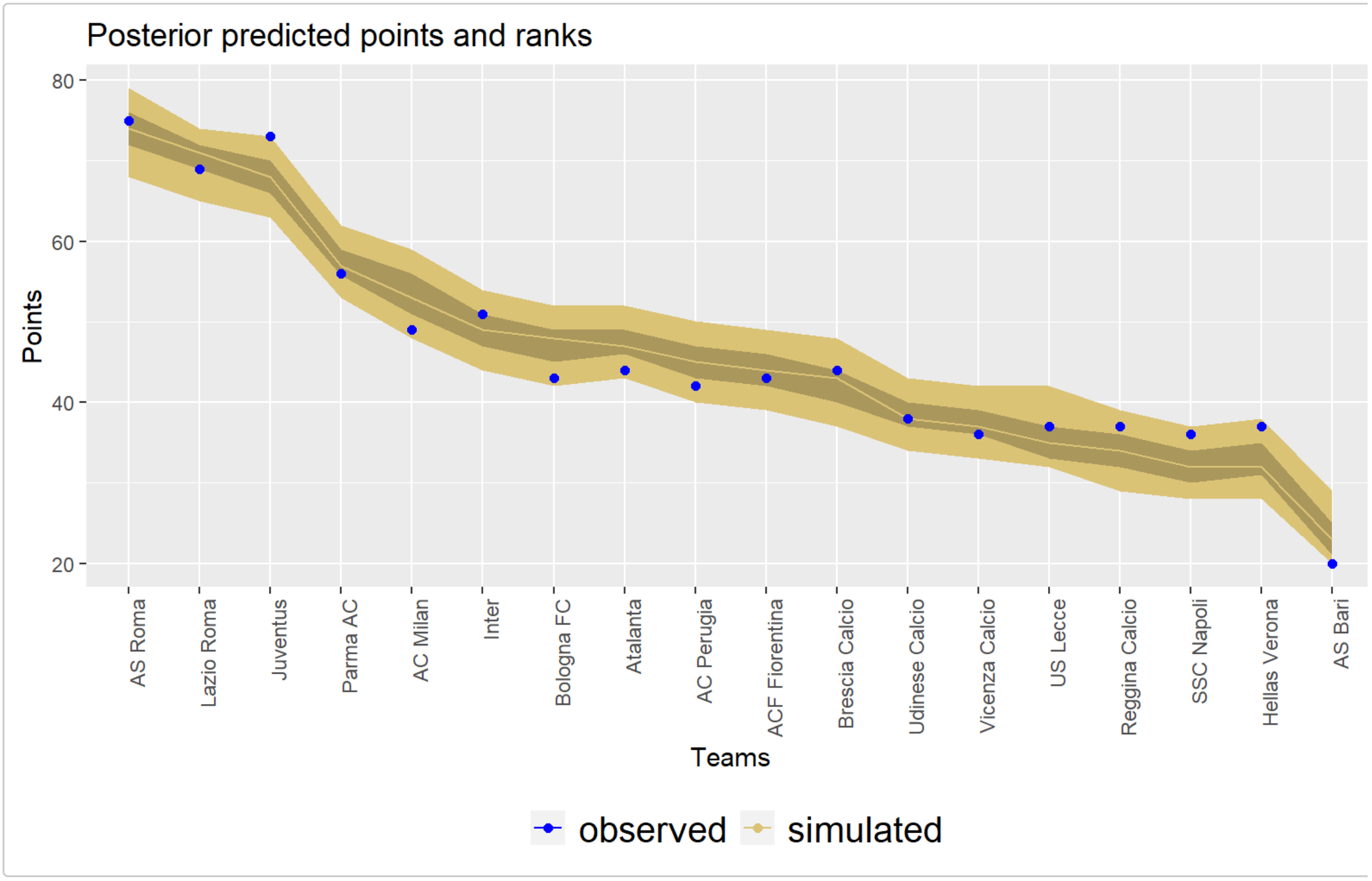
```
# team-specific plot

foot_rank(data = italy_2000, object = fit1_stan, visualize = "individual")
#> $rank_plot
```



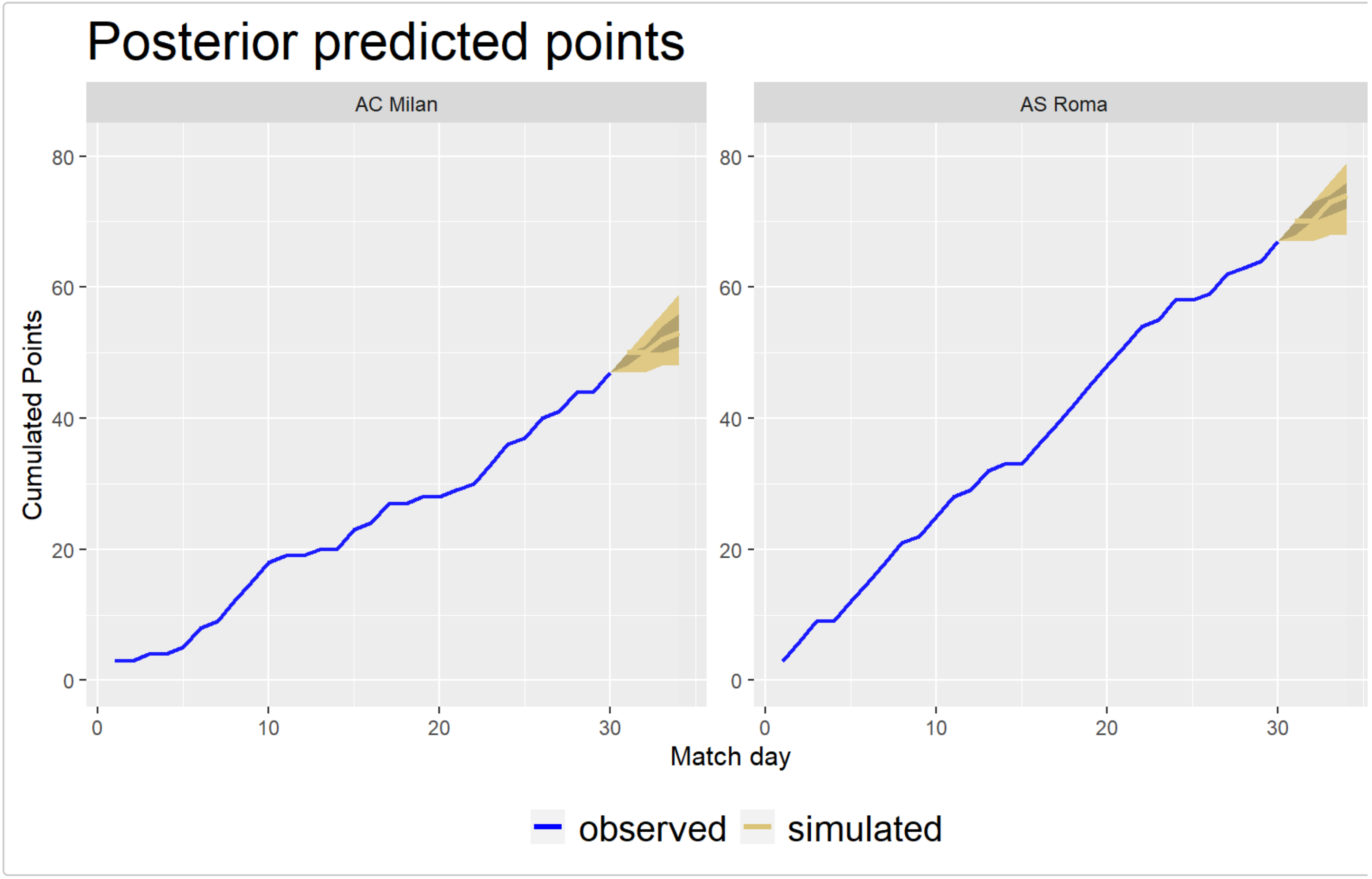
- **rank-league prediction** (aggregated or at team-level) for the fourth model `fit4_stan` by using the **out-of-sample** replications $\tilde{\mathcal{D}}$ (yellow ribbons for the credible intervals, solid blue lines for the observed cumulated points):

```
## Rank predictions for individual teams

# aggregated plot

foot_rank(data = italy_2000, object = fit4_stan)
#> $rank_table
#>              teams mid obs lo hi
#> 1          AS Roma  74  75 72 76
#> 2       Lazio Roma  71  69 69 72
#> 3         Juventus  68  73 66 70
#> 4         Parma AC  57  56 56 59
#> 5         AC Milan  53  49 51 56
#> 6            Inter  49  51 47 51
#> 7       Bologna FC  48  43 45 49
#> 8         Atalanta  47  44 46 49
#> 9       AC Perugia  45  42 43 47
#> 10 ACF Fiorentina  44  43 42 46
#> 11 Brescia Calcio  43  44 40 44
#> 12 Udinese Calcio  38  38 37 40
#> 13 Vicenza Calcio  37  36 36 39
#> 14       US Lecce  35  37 33 37
#> 15 Reggina Calcio  34  37 32 36
#> 16      SSC Napoli  32  36 30 34
#> 17   Hellas Verona  32  37 31 35
#> 18         AS Bari  23  20 21 25
```

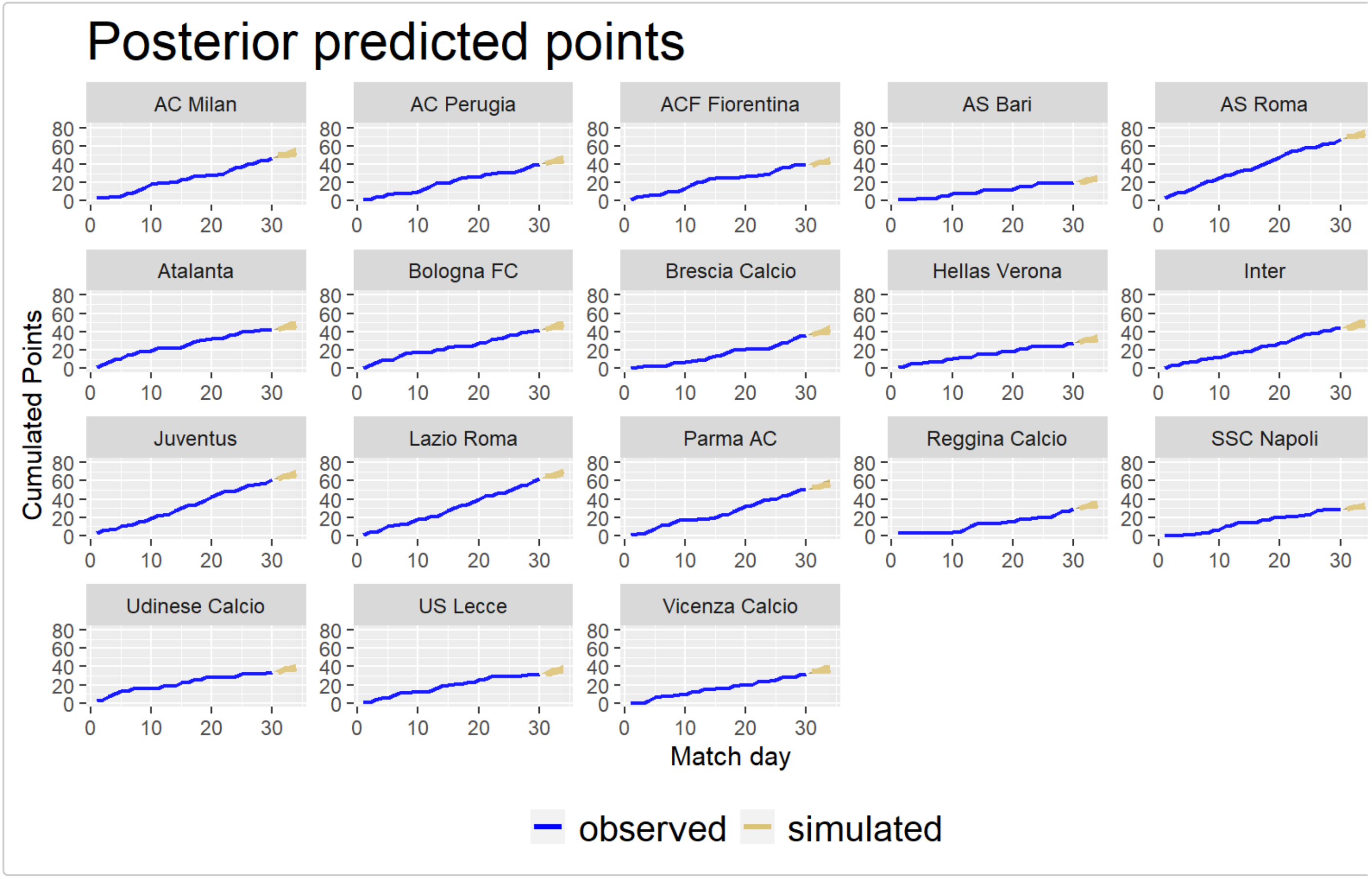
```
#>
#> $rank_plot
```



```
# team-specific plot

foot_rank(italy_2000, fit4_stan,
          team_sel = c("AC Milan", "AS Roma"),
          visualize = "individual")
#> $rank_plot
```



```r
foot_rank(italy_2000, fit4_stan,
          visualize = "individual")
#> $rank_plot
```



Model comparisons: LOOIC and WAIC

Comparing statistical models in terms of some **predictive information criteria** should conclude many analysis and can be carried out by using the Leave-one-out cross-validation criterion (**LOOIC**) and the Watanabe Akaike Information criterion (**WAIC**) performed by using the `loo` package. For more details about LOOIC and WAIC, read the paper Vehtari, Gelman, and Gabry (2017).

The general formulation for the predictive information criteria is the following:

$$\text{crit} = -2\widehat{\text{elpd}} = -2(\widehat{\text{lpd}} - \text{parameters penalty})$$

- $\widehat{\text{elpd}}$: estimate of the expected log predictive density of the fitted model.
- $\widehat{\text{lpd}}$ is a measure of the log predictive density of the fitted model.
- parameters penalty is a penalization accounting for the effective number of parameters of the fitted model.

The interpretation is the following: the lower is the value for an information criterion, and the better is the estimated model's predictive accuracy. Moreover, if two competing models share the same value for the log predictive density, the model with less parameters is favored.

This is the **Occam's Razor** occurring in statistics:

> "Frustra fit per plura quod potest fieri per pauciora"

We can perform Bayesian model comparisons by using the `loo` and `waic` functions of the `loo` package. We are going to compare the static and the weekly dynamic models on the Italian Serie A 2000/2001:

```
### Model comparisons
## LOOIC, loo function

# extract pointwise log-likelihood

log_lik_1 <- extract_log_lik(fit1_stan)
log_lik_1_t <- extract_log_lik(fit1_stan_t)
log_lik_2 <- extract_log_lik(fit2_stan)
log_lik_3 <- extract_log_lik(fit3_stan)
log_lik_3_t <- extract_log_lik(fit3_stan_t)

# compute loo

loo1 <- loo(log_lik_1)
loo1_t <- loo(log_lik_1_t)
loo2 <- loo(log_lik_2)
loo3 <- loo(log_lik_3)
loo3_t <- loo(log_lik_3_t)


# compare three looic

compare(loo1, loo1_t, loo2, loo3, loo3_t)
#>        elpd_diff se_diff elpd_loo p_loo  looic
#> loo1_t    0.0       0.0  -892.0     20.5 1784.1
#> loo1     -0.8       0.5  -892.8     21.8 1785.7
#> loo2     -2.8       2.7  -894.8     34.1 1789.6
#> loo3     -4.9       4.0  -897.0     35.9 1793.9
#> loo3_t   -6.8       4.0  -898.9     39.2 1797.7
```

According to the above model LOOIC comparisons, the weekly-dynamic double Poisson models attain the lowest LOOIC values and are then the favored models in terms of predictive accuracy. The static model's `fit1_stan` final looic is suggesting that the assumption of static team-specific parameters is too restrictive and oversimplified to capture teams' skills over time and make reliable predictions. Anyway, from model checking we have the suggestion that even the static model has a reliable goodness of fit and could be used for some simplified analysis not requiring complex dynamic patterns.

Extensions in next versions

Extensions and to-do list for the next package's versions:

1. **Data Web-scraping**: automatic routine to scrape data from internet;
2. **More numerical outputs**: posterior probabilities, confusion matrices, betting strategies, etc.;
3. **Diagnostics, pp checks** designed for football;
4. **Teams' statistics**
5. **More covariates** to be included in the model (possibly by users).
6. **More priors choices**

References

Baio, Gianluca, and Marta Blangiardo. 2010. "Bayesian Hierarchical Model for the Prediction of Football Results." *Journal of Applied Statistics* 37 (2): 253–64.

Betancourt, Michael. 2017. "A Conceptual Introduction to Hamiltonian Monte Carlo." *arXiv Preprint arXiv:1701.02434*.

Egidi, Leonardo, Francesco Pauli, and Nicola Torelli. 2018. "Combining Historical Data and Bookmakers' Odds in Modelling Football Scores." *Statistical Modelling* 18 (5-6): 436–59.

Egidi, Leonardo, and Nicola Torelli. 2020. "Comparing Goal-Based and Result-Based Approaches in Modelling Football Outcomes." *Social Indicators Research*, 1–13.

Gelman, Andrew. 2014. "Stan Goes to the World Cup." *Statistical Modeling, Causal Inference, and Social Science Blog*. https://statmodeling.stat.columbia.edu/2014/07/13/stan-analyzes-world-cup-data/.

Gelman, Andrew, John B Carlin, Hal S Stern, and Donald B Rubin. 2014. *Bayesian Data Analysis*. Vol. 2. Chapman & Hall/CRC Boca Raton, FL, USA.

Groll, Andreas, Gunther Schauberger, and Gerhard Tutz. 2015. "Prediction of Major International Soccer Tournaments Based on Team-Specific Regularized Poisson Regression: An Application to the FIFA World Cup 2014." *Journal of Quantitative Analysis in Sports* 11 (2): 97–115.

Karlis, Dimitris, and Ioannis Ntzoufras. 2003. "Analysis of Sports Data by Using Bivariate Poisson Models." *Journal of the Royal Statistical Society: Series D (The Statistician)* 52 (3): 381–93.

———. 2009. "Bayesian Modelling of Football Outcomes: Using the Skellam's Distribution for the Goal Difference." *IMA Journal of Management Mathematics* 20 (2): 133–45.

Koopman, Siem Jan, and Rutger Lit. 2015. "A Dynamic Bivariate Poisson Model for Analysing and Forecasting Match Results in the English Premier League." *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 178 (1): 167–86.

Owen, Alun. 2011. "Dynamic Bayesian Forecasting Models of Football Match Outcomes with Estimation of the Evolution Variance Parameter." *IMA Journal of Management Mathematics* 22 (2): 99–113.

Robert, Christian, and George Casella. 2013. *Monte Carlo Statistical Methods*. Springer Science & Business Media.

Stan Development Team. 2020. "RStan: The R Interface to Stan." https://mc-stan.org/.

Vehtari, Aki, Andrew Gelman, and Jonah Gabry. 2017. "Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and WAIC." *Statistics and Computing* 27 (5): 1413–32.