# 1 Robust Regression

Bayesian regression with gaussian errors, like ordinary least-square regression is quite sensitive to outliers[**?**]. To increase outlier robustness, a Bayesian regression model with Student-T errors is more effective[**?**]. This is the first model which while being relatively straightforward is quite effective in several real-world usecases.

$$y \sim StudentT(\nu, \mu, \sigma) \tag{1}$$

Here, $\nu$ is the degrees of freedom parameter. As $\nu$ approaches $\inf$, the distribution approaches normal. Here we choose prior for $\nu$ as:

$$\nu \sim Gamma(shape = 2, scale = 10)$$

$\mu$ is the mean of the regression and is given by:

$$\mu = \alpha + \beta * X$$

$\sigma$ is the variance of the $StudentT$ distribution and has a prior:

$$\sigma \sim Exponential(1)$$

And $\alpha, \beta, X$ are sampled from:

$$\alpha \sim Normal(0, 10), \quad \beta \sim Normal(0, 2.5), \quad X \sim Normal(0, 10)$$

## 1.1 Data Generation

PPLBench generates all the data internally. This allows for a more customizable way to evaluate the PPL implementations of the model. In this case, to test the robustness of the model implementations, we generate data by sampling the alphas and betas from the $\sigma = [10, 11]$ tails of the distribution. A total of $2N = 2000$ data points were generated, where each point being generated from a sample drawn for these tails of parameter distributions. We use 1000 of the generated data as evidence for the model implementations to facilitate convergence of sampling process. The other 1000 are used to compute the posterior predictive log-likelihood of the parameter samples.

## 1.2 PPL Implementations

The robust regression model was implemented in Stan and Jags PPLs, with the help of PyStan[**?**] and PyJAGS[**?**] libraries for interface. The compilation and inference times were recorded. The PPLBench is designed to accept an approximate time per PPL per iteration. The PPLBench estimates the number of posterior samples to obtain from the implementations based on this time constraint. The PPL's default warmup/burn-in and initialization parameters were left unchanged. The sampling is restricted to a single chain and any form of multi-threading is disabled. One noteworthy difference between the two would be that Jags uses Gibbs sampler while Stan uses NUTS HMC. This difference will be important when we analyze the results.

## 1.3 Results

Upon obtaining the posterior samples for each PPL implementation and for each iteration, we compute the posterior predictive log-likelihood w.r.t test data. We use this to then compute the average posterior predictive w.r.t time over each iteration, and obtain a graph of the min, max and mean. These are plotted on a graph for each PPL implementation so their convergence behavior can be visualized. The plot has a log-time axis to help accommodate larger timescales and vastly different convergence behaviors. This plot is designed to capture both the relative performance and accuracy of these implementations. Figure 1 shows the comparative performance without considering the inference time.

From the figure, we can make the following observations:

- Both implementations converge to the same posterior predictive log-likelihood given enough time.
- Without considering compile time, Stan is both faster to converge as well has a much lower time per sample.
- Stan's NUTS inference starts sampling from relatively high log-likelihood space and hence converges much quicker than Jags' Gibbs sampler, which has a much slower convergence.

The specifications of the benchmark are tabulated in table 1

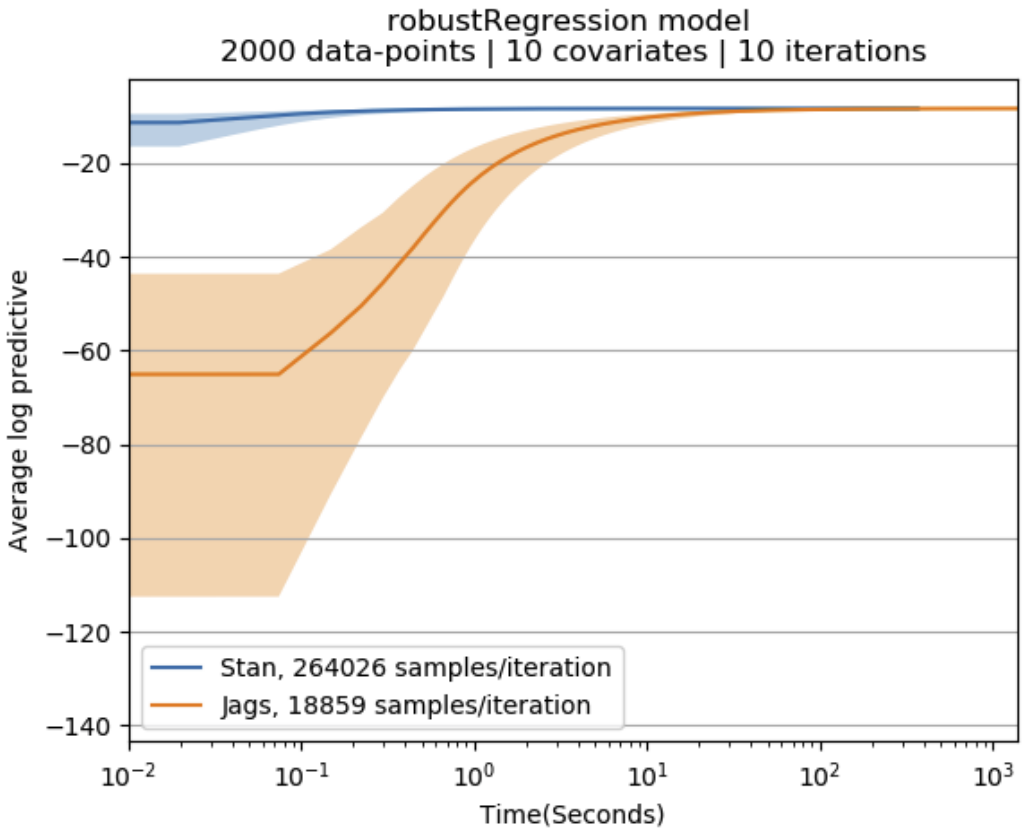


Figure 1: Posterior convergence behaviour of Jags and Stan for Robust Regression model

Table 1: Robust Regression Model

| Model Argument | Value | Notes |
|---|---|---|
| What | Goes | Here? |
| Axon | Output terminal | $\sim$10 |
| Soma | Cell body | up to $10^6$ |