Bayesian Analysis mit JAGS

Langyan Zang

October 8, 2022

1 Bayesian Analysis

1.1 Introduction

In 1763, a prominent theorem was proposed by mathematician Thomas Bayes. This simple rule spawned a vast ramificaitions for statistical inference, which are composed of a lot of researchers and believers called Bayesian. One of Bayes' successors, Pierre-Simon Laplace rediscovered and extensively developed the methods after Bayes' death.

Another branch of statistics is called frequentist refering to people who believe in data explaining everything and do not use Bayes' rule for statistical inference or colusion. This branch is originally founded by Ronald Fisher, who was born 200 years later than Bayes. Fisher (1922)[1] said "The discussion of theoretical statistics may be regarded as alternating between problems of estimation and problems of distribution. In the first place a method of calculating one of the population parameters is devised from common-sense considerations: we next require to know its probable error, and therefore an approximate solution of the distribution, in samples, of the statistics calculated.". One of greatest work by Ronald Fisher is his delineation to the concept of likelihood. The importance of likelihood is hardly overstated as one of the primary concepts underlying statistical inference (Paolella, 2018)[2]. It is the most important and major part which forms the ground of both frequentist and Bayesian schools inference. A brief introduction is given below.

Likelihood function $\mathcal{L}(\boldsymbol{\theta};\mathbf{Y})$ is the joint density of a sample $\mathbf{Y} = (Y_1, \dots, Y_n)$, viewed as a function of the k-dimentional parameter $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^k$ (Paolella, 2018)[2]. From a frequentist's view, we assume the sample $\mathbf{Y}$ is following a specific distribution, and it is the parameters of the distribution that we really concern. Maximum likelihood comes to play after assumption. The m.l.e., standing for maximum likelihood estimation as a point estimator of the unknown parameter, $\boldsymbol{\theta}_{\mathrm{ML}} = argmax\ \mathcal{L}(\boldsymbol{\theta};\mathbf{Y})$ over $\boldsymbol{\theta} \in \Theta$.

1.2 Bayes' Rule

The Bayes' Rule is rather simple. It is comprised of two major components, one for the data and one for the parameter. Unlike maximum likelihood method, it adds credibility to distribution parameter, called **prior**, rather than merely maximising the likelihood function, which means the distribution parameters also have tendency. This gives the famous Bayes' Rule as follows:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\sum_{\theta^*} p(D|\theta^*)p(\theta^*)} = \frac{p(D|\theta)p(\theta)}{\int_{\theta^*} p(D|\theta^*)p(\theta^*)} \tag{1}$$

The likelihood is $p(D|\theta)$ given θ under a specific distribution, and this is for the data. The other one $p(\theta)$ prior is for the parameter's belief. The multiplication of both forms the scaled **posterior** distribution. To make the posterior distribution a real distribution, i.e. the integral or sum of its probabilities equals to 1, the conditional probability is used.

1.3 Inference Framework

$$p(\theta|D) = p(\beta_0, \beta_1, \sigma|y, x) \Leftarrow \begin{cases} \textit{Likelihood:} & y \sim N(\mu, \sigma^2) \\ & \mu = \beta_0 + \beta_1 x \\ \textit{Prior:} & \beta_0 \sim N(0, 10000) \\ & \beta_1 \sim N(0, 10000) \\ & \sigma \sim U(0, 100) \end{cases}$$

1.4 Markov Chain Monte Carlo

Since the posterior distribution is generated by prior and likelihood, the analytical form of it is not always available. If available, it could also be ugly and ill. In addition, the derivation of the posterior distribution is rather time-consuming and not promising. Hence, a method called Markov Chain Monte Carlo (MCMC) comes into play. I will use MCMC as syntax for Markov Chain Monte Carlo method in the rest of this document. MCMC is especially useful in simulating a distribution without a exact distribution function provided. This means the denominator of Bayes' rule is not necessary to be calculated when simulating the posterior distribution. Therefore, with the help of MCMC, the Bayes' rule is reduced to $p(\theta|D) \propto p(D|\theta)p(\theta)$. The most commonly used MCMC are as follows:

1. Metropolis-Hastings algorithm
2. Gibbs Sampling
3. Hamiltonian Monte Carlo (HMC)

Details of the implementation of these algorithms can be viewed and obtained on my Github repository https://github.com/7lang2yan/Markov-Chain-Monte-Carlo-MCMC.git.

2 JAGS Basic

JAGS stands for *Just another Gibbs sampler*, as it is a successor of the pioneering system BUGS which is the abbreviation for *Bayesian inference using Gibbs sampling*. JAGS uses a mixture of Metropolis and Gibbs algorithms to sample the posterior distribution. JAGS is written in C++, I will not introduce the use of JAGS C++ interface, but rather use it from R via the **rjags** and **runjags** packages.

2.1 Installation

To install JAGS, go to https://mcmc-jags.sourceforge.io/ and click on the files page link. Press on the *Download Latest Version* button and the downloading will shortly start automaticly.

To use it from the R, `rjags` and `runjags` two R packages are needed. Open R or R studio, go to the command line, type in `install.packages("rjags")` and `install.packages("runjags")` to install both packages, such that you can communicate with JAGS from R now.

2.2 JAGS Syntax

A JAGS model in defined in a text file using a dialect of the BUGS language. The model definition starts with the keyword `model` followed by curly brackets `{}`.

```
model {
        for (i in 1:N) {
                Y[i]  ~ dnorm(mu[i], tau)
                mu[i] <- alpha + beta * (x[i] - x.bar)
        }
        x.bar <- mean(x)
        alpha ~ dnorm(0.0, 1.0E-4)
        beta ~ dnorm(0.0, 1.0E-4)
        sigma <- 1.0 / sqrt(tau)
        tau ~ dgamma(1.0E-3, 1.0E-3)
}
```

Listing 1: Simple Linear Regression Example

2.2.1 Relations

Relations in JAGS can be of two types. One for *stochastic relation* and the other one for *deterministic relation*. The *stochastic relation* is defined by `~` representing a *random variable*, such as `alpha ~ dnorm(0.0, 1.0E-4)` meaning `alpha` following a Normal distribution with 0 mean(μ) and $1.0e-4$ precision(τ) (Note: the Normal distribution in JAGS does not take variance(σ^2) or standard deviation(σ) as parameter, but rather take precision(τ) as its parameter. The relationship between variance and precision: $\sigma^2 = \frac{1}{\tau}$. Details will be elaborated in the distribution section.). Whereas the *deterministic relation* is defined by `<-`, representing the variable on the left hand side exactly equals to the value on the right hand side. For instance, `sigma <- 1.0 / sqrt(tau)` means `sigma` equals to `1.0 / sqrt(tau)`. It is rather similar to the assigning operation in other programming language, but not quite an assigning operation. It always stands for a relation between both sides.

2.2.2 Constructing and Subsetting Vectors and Matrices

The vector construction syntax is generally same as in R, such as `y <- c(x[1], x[2], x[3])`. The subsetting is also the same as in R. For example, vector subsetting: `x[1:n]` and matrix subsetting: `A[1:M,1:N]` or `A[r,]` for the r^{th} row of matrix A or `A[,c]` for the c^{th} column.

2.2.3 For loops

The for loops in JAGS is exactly same as in R.

```
for (1:n) {
        y[i] ~ dnorm(0.0,1.0E-3)
}
```

2.2.4 Data Sampling and Transformation

The data sampling and transformation can be done in JAGS within the `data` block. For example,

```
data {
        for (i in 1:N) {
                y[i] ~ dnorm(1.0, 0.001)
        }
}
model {
        fake <- 0
}
```

Listing 2: Sampling in JAGS

```
data {
        for (i in 1:N) {
                y[i] <- sqrt(x[i])
        }
}
model {
        for (i in 1:N) {
                y[i] ~ dnorm(mu,tau)
        }
}
```

Listing 3: Data Transformation in JAGS

2.3 JAGS Distributions

2.3.1 Discrete Distributions

Table 1: Built-In Discrete Distributions

Distribution	*Usage*	*Density*	*Range*
Beta Binomial	`dbetabin(a,b,n)`, $a > 0, b > 0, n \in \mathbb{N}^*$	$\binom{a+x-1}{x}\binom{b+n-x-1}{n-x}\binom{a+b+n-1}{n}^{-1}$	(0,1)
Bernoulli	`dbern(p)`, $0 < p < 1$	$p^x(1-p)^{1-x}$	(0,1)
Binomial	`dbin(p,n)`, $0 < p < 1, n \in \mathbb{N}^*$	$\binom{n}{x}p^x(1-p)^{n-x}$	(0,n)
Categorical	`dcat(pi)`, $\pi \in (\mathbb{R}^+)^N$	$\frac{\pi_x}{\sum_i \pi_i}$	(1,N)
Hypergeometric	`dhyper(n1,n2,m1,psi)`, $n_i \geq 0, 0 < m_1 < n_+$	$\frac{\binom{n_1}{x}\binom{n_2}{m_1-x}\psi^x}{\sum_i \binom{n_1}{i}\binom{n_2}{m_1-i}\psi^i}$	$(\max(0, n_+ - m_1),$ $\min(n_1, m_1))$
Negative Binomial	`dnegbin(p,r)`, $0 < p \leq 1, r \geq 0$	$\binom{x+r-1}{x}p^r(1-p)^x$	$(0, \infty)$
Poisson	`dpois(lambda)`, $\lambda > 0$	$\frac{e^{-\lambda}\lambda^x}{x!}$	$(0, \infty)$

2.3.2 Continuous Distributions

Table 2: Built-In Continuous Distributions

Distribution	*Usage*	*Density*	*Range*
Beta	`dbeta(a,b)`, $a > 0, b > 0$	$\frac{x^{a-1}(1-x)^{b-1}}{B(a,b)}$	(0,1)
χ^2	`dchisqr(k)`, $k > 0$	$\frac{x^{\frac{k}{2}-1}e^{-\frac{x}{2}}}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})}$	$(0,\infty)$
Double Exponential	`ddexp(mu,tau)`, $\tau > 0$	$\frac{\tau e^{-\tau\vert x-\mu\vert}}{2}$	$(0,\infty)$
Exponential	`dexp(lambda)`, $\lambda > 0$	$\lambda e^{-\lambda x}$	$(0,\infty)$
F	`df(n,m)`, $n > 0, m > 0$	$\frac{\Gamma(\frac{n+m}{2})}{\Gamma(\frac{n}{2})\Gamma(\frac{m}{2})}\left(\frac{n}{m}\right)^{\frac{n}{2}} x^{\frac{n}{2}-1}\left(1+\frac{nx}{m}\right)^{-\frac{n+m}{2}}$	$(0,\infty)$
Gamma	`dgamma(r, lambda)`, $r > 0, \lambda > 0$	$\frac{\lambda^r x^{r-1}e^{-\lambda x}}{\Gamma(r)}$	$(0,\infty)$
Generalized Gamma	`dgen.gamma(r,lambda,b)`, $r > 0, \lambda > 0, b > 0$	$\frac{b\lambda^{br}x^{br-1}e^{-(\lambda x)^b}}{\Gamma(r)}$	$(0,\infty)$
Logistic	`dlogis(mu,tau)`, $\tau > 0$	$\frac{\tau e^{\tau(x-\mu)}}{(1+e^{\tau(x-\mu)})^2}$	$(0,\infty))$
Log-normal	`dlnorm(mu,tau)`, $\tau > 0$	$\left(\frac{\tau}{2\pi}\right)^{\frac{1}{2}} x^{-1}e^{-\tau(\log(x)-\mu)^2/2}$	$(0,\infty)$
Noncentral χ^2	`dnchisqr(k,delta)`, $k > 0, \delta \geq 0$	$\sum_{r=0}^{\infty}\frac{e^{-\frac{\delta}{2}}(\frac{\delta}{2})^r}{r!}\frac{x^{\frac{k}{2}+r-1}e^{-\frac{x}{2}}}{2^{\frac{k}{2}+r}\Gamma(\frac{k}{2}+r)}$	$(0,\infty)$
Normal	`dnorm(mu,tau)`, $\tau > 0$	$\left(\frac{\tau}{2\pi}\right)^{\frac{1}{2}} e^{-\frac{\tau(x-\mu)^2}{2}}$	$(0,\infty)$
Pareto	`dpar(alpha,c)`, $\alpha > 0, c > 0$	$\alpha c^{\alpha}x^{-(\alpha+1)}$	$(0,\infty)$
Student t	`dt(mu,tau,k)`, $\tau > 0, k > 0$	$\frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})}\left(\frac{\tau}{k\pi}\right)^{\frac{1}{2}}\left(1+\frac{\tau(x-\mu)^2}{k}\right)^{-\frac{k+1}{2}}$	(c,∞)
Uniform	`dunif(a,b)`, $a < b$	$\frac{1}{b-a}$	(a,b)
Weibull	`dweib(v,lambda)`, $\nu > 0, \lambda > 0$	$\mu\lambda x^{\nu-1}e^{-\lambda x^{\nu}}$	$(0,\infty)$

3 Workflow of JAGS model in R

There are various ways to run JAGS models in R, I will only illustrate one way, just to set the ball rolling. Load the required package before everything happens `library(runjags)`.

3.1 Prepare Data

First step, data need to be prepared for JAGS' use. I will be using simulated data here as an example.

```
# Generate random data
Ntotal = 18
x = seq(-5,5,by=0.001)
x = sample(x,Ntotal)
y = (3 + x) + rnorm(Ntotal,0,6)
# Add outliers
x = c(x,4,4.2)
y = c(y,35,36)

# Prepare the data for JAGS
dat = dump.format(list(y=y, x=x, Ntotal=Ntotal))
```

A `list` function is used here to prepare variables with specific names such that JAGS will recognize. The `dump.format` function is to package the data and make it ready for the shipment to JAGS.

3.2 Define Model

The JAGS models should all be defined in a `.txt` file. Here I will show how you can write a `.txt` file in R.

```
linear_regression_model = "
model {

        beta0 ~ dnorm(0, 0.00001) # prior intercept
        beta1 ~ dnorm(0, 0.00001) # prior slope
        sigma ~ dunif(0.00001, 100) # prior for the standard deviation

        for (i in 1:Ntotal) {
                mu[i] <- beta0 + beta1*x[i]
                y[i] ~ dnorm(mu[i], 1/sigma^2)
        }

}
"

writeLines( linear_regression_model, con = "models/linearRegr.txt" )
```

3.3 Initialize Chains

Before running JAGS, the starting point for the posterior distribution sampling, i.e. Markov chain, is needed. If the starting point is not provided, JAGS will use same starting points for all chains. Also, JAGS needs to be told which parameter to monitor.

```
# Initialize chains
inits1 <- dump.format(list(beta0=0, beta1=0, sigma=3,
                 .RNG.name="base::Super-Duper", .RNG.seed=99999 ))
inits2 <- dump.format(list(beta0=1, beta1=1, sigma=2,
                 .RNG.name="base::Wichmann-Hill", .RNG.seed=1234 ))
inits3 <- dump.format(list(beta0=2, beta1=2, sigma=4,
                 .RNG.name="base::Mersenne-Twister", .RNG.seed=6666 ))

# Tell JAGS which latent variables to monitor
monitor = c("beta0", "beta1", "sigma", "deviance")
```

3.4 Run Jags

Calling JAGS is done by function `run.jags`. It needs to be provided with multiple parameters, i.e. `model`: model file location, `monitor`: monitoring parameters, `data`: packaged data, `n.chains`: number of chains, `inits`: starting points / initial points for each chain, `plots`: whether to plot, `burnin`: the number of burnin iterations, `sample`: the total number of (additional) samples to take, `thin`: the thinning interval to be used in JAGS.

```
# Run the function that fits the models using JAGS
results <- run.jags(model="models/linearRegr.txt",
                    monitor=monitor,
                    data=dat,
                    n.chains=3,
                    inits=c(inits1, inits2, inits3),
                    plots = FALSE,
                    burnin=4000,
                    sample=5000,
                    thin=5)
```

3.5 Plot Chains

After running JAGS, all results are stored in the variable `results`. The chains are stored in the struct variable within `results`. Retrieve it and plot it. The `plot` in R support this type of data, i.e. `mcmc.list`.

```
# plot the chains
plot(results$mcmc)
```

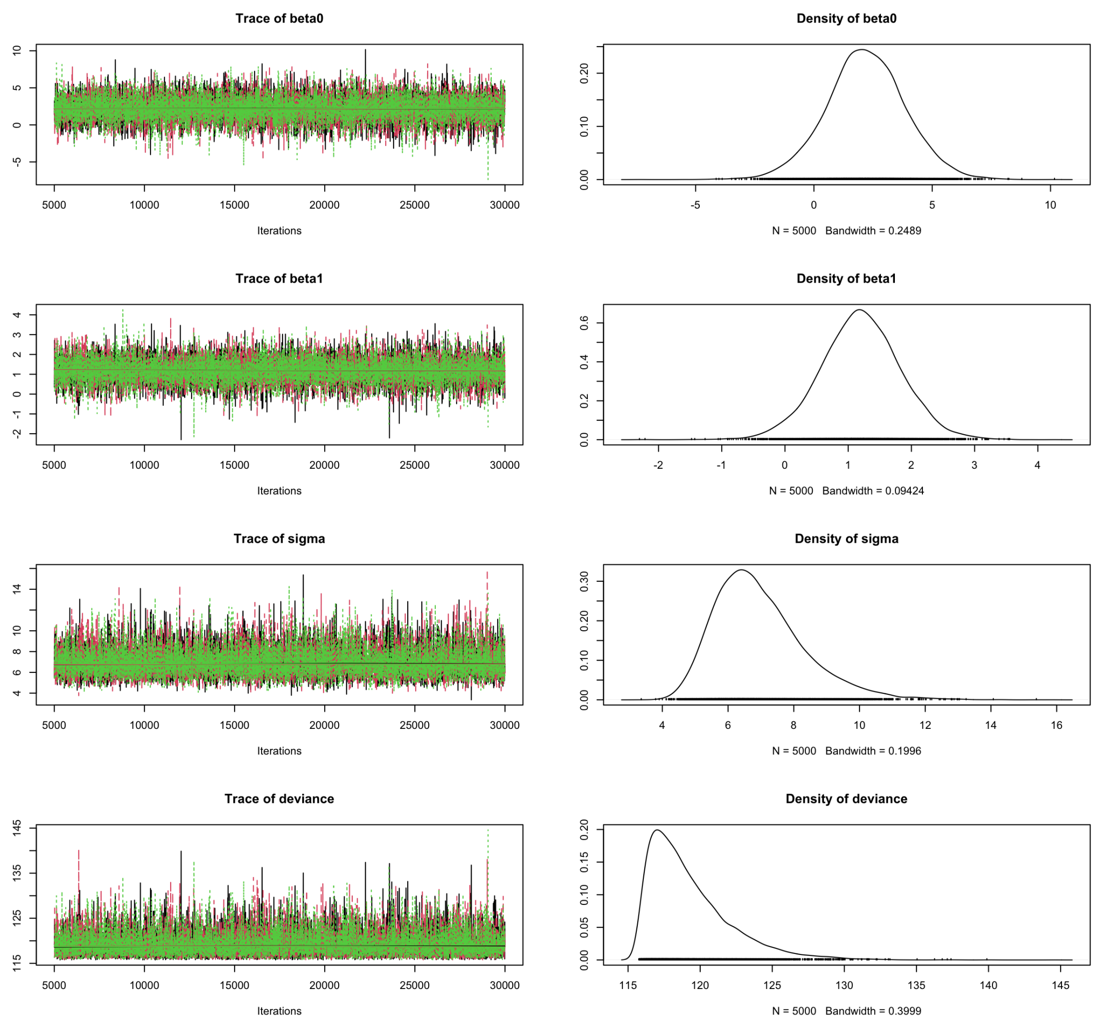

Figure 1: Posterior Distribution Sampling for Each Monitored Parameters

4 Reinforcement Learning - Iowa Card Game

4.1 Iowa Gambling Task

The Iowa Gambling Task(IGT) is a psychological task to simulate real-life decision making. This simulation is introduced by Antoine Bechara, Antonio Damasio, Hanna Damasio and Steven Anderson.

The game is designed as picking one card from four cards (A,B,C,D) without knowing the probability of payoff of each card. The payoff distribution of the IGT is as follows.

Table 3: IGT Payoff Distribution

Deck	A	B	C	D
Gain from each trial	1.00	1.00	0.50	0.50
Number of loss(es) in each set of 10 trials	5	1	5	1
Loss amount in each set of 10 trials	-1.50	-12.50	-0.25	-2.50
	-2.00		-0.50	
	-2.50		-0.50	
	-3.00		-0.50	
	-3.50		-0.75	
Expected value of 10 trials	-2.50	-2.50	2.50	2.50

4.2 Learning Model

We want a model to describe the learning process of each human participants, since they do not know the information of each deck shown in the table above. Participants are expected to learn by trial and error.

The **Reward** after picking one deck can be denoted with the following equation:

$$R_t = win(t) - loss(t) \tag{2}$$

For example, if we pick deck B, we would win \$1.00, and this time we happen to encounter the case that we need to lose \$ 12.50. Then our reward from the action of picking deck B is $R_t = 1.00 - 12.50 = -11.50$.

With this information after each trial, we can update our knowledge of the deck we pick in that trial. Let a_j denote the action of choosing deck j, $Q_t(a_j)$ denote the estimated value of choosing deck j in trial t. Hence we update our knowledge of deck j after trial t with reagrd to the difference between our previous knowledge of deck j and the reward. We want our knowledge to be close to the reward R_t, which means if our previous knowledge or guess about deck j $Q_{t-1}(a_j)$ is away from the reward R_t, we conisder it as a failure of estimate and re-estimate the value of deck j after trial t. However, we cannot fully accept the reward R_t after one trial, since the value of the deck is a probability problem when we do not have any information regarding the game. Therefore, a learning rate α is introduced in our model to depict the gradual acceptance process when human learning uncertainty.

$$Q_t(a_j) = Q_{t-1}(a_j) + \alpha \overbrace{(R_t - Q_{t-1}(a_j))}^{\textit{Prediction error}} \tag{3}$$

We call the difference between reward R_t and previous estimate $Q_{t-1}(a_j)$ as *Prediction error.*

We cannot always pick the deck with the current highest expected value, because our guess towards other decks might be incorrect currently which could possibly be higher than the one we think is the highest now, and we would want to update the other decks estimates. Hence, a stochastic policy needs to be introduced for the learning model to pick decks. The policy we want would instruct the learning model to pick the deck with current highest estimated value more likely, but still have the chance to pick other decks, and if two decks' estimated value are close, their probabilities of being chosen should also be close. There is a distribution would act exactly as we want, that is *Softmax* distribution.

$$\Pr_t(a_j) = \frac{e^{\beta Q_{t-1}(a_j)}}{\sum_i e^{\beta Q_{t-1}(a_j)}} \tag{4}$$

where β is the inverse temperature parameter.

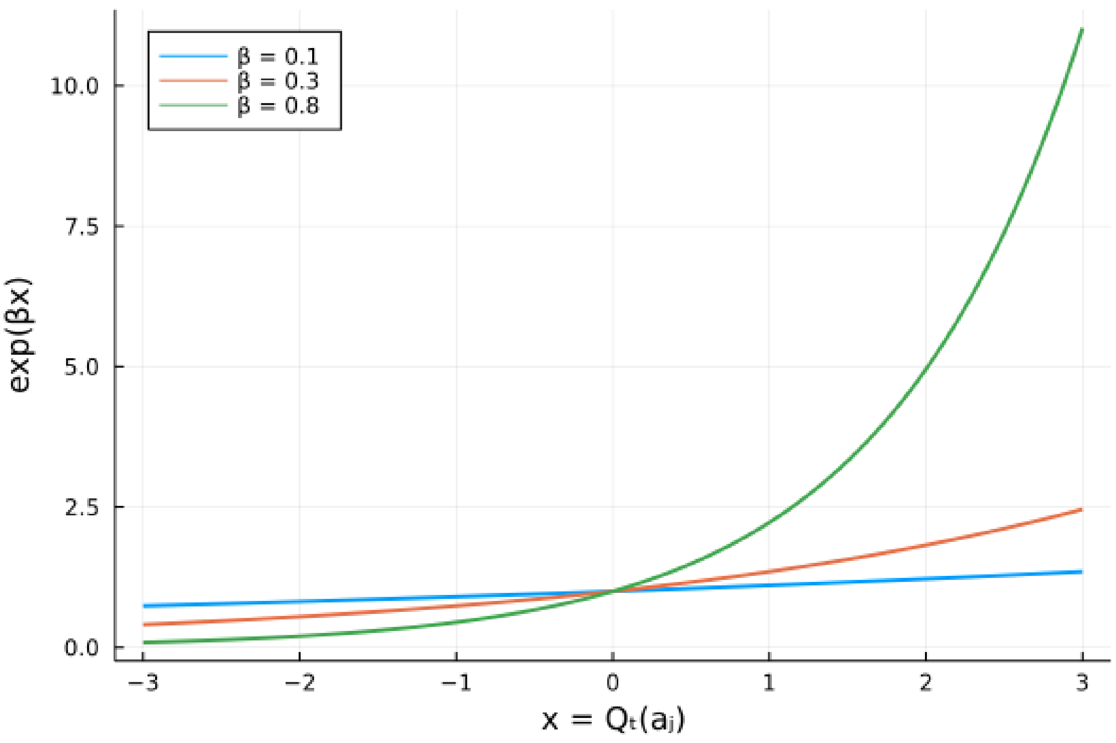

Figure 2: Inverse temperature parameter scaling of $Q_t(a_j)$

4.2.1 Gains and Losses Asymmetry

The initial model of IGT was *Expectancy-valance learning* model (EVL). This model allows for asymmetry in how gains and losses are evaluated.

$$R_t = (1 - W) \cdot win(t) - W \cdot loss(t) \tag{5}$$

4.2.2 Inverse Temparature Parameter

We want to explore more decks in the beginning of the trials, which requires relatively equal probabilities of choosing decks even after learning. Therefore, a trial-dependent temperature parameter would meet our requirement.

$$\beta(t) = \left(\frac{t}{10}\right)^c \tag{6}$$

4.2.3 Final Reinforcement Learning Model

$$R_t = (1 - W) \cdot win(t) - W \cdot loss(t) \tag{7}$$

$$Q_t(a_j) = Q_{t-1}(a_j) + \alpha\,(R_t - Q_{t-1}(a_j)) \tag{8}$$

$$\Pr_t(a_j) = \frac{e^{\beta Q_{t-1}(a_j)}}{\sum_i e^{\beta Q_{t-1}(a_j)}} \tag{9}$$

$$\beta(t) = \left(\frac{t}{10}\right)^c \tag{10}$$

The JAGS model for above simulation model is as follows:

```
model {
        for  (d in 1:D) {
                q[1,d] <- 0
        }

        for (t in 1:T) {
                beta[t] <- ifelse(t<=10, (t/10)^2, 1)
                for (d in 1:D) {
                        choice_prob[t,d] <- exp(beta*q[t,d])
                }
                choices[t] ~ dcat(choice_prob[t,])
                for (d in 1:D) {
                  q[t+1,d] <- ifelse(
                   d==choices[t], # if this is the chosen deck
                   # update the value of the deck if it is sampled in current trial
                   (1-alpha) * q[t,d] + alpha*((1-W)*gains[t] - W*losses[t]),
                   # set future expectation as current estimated value if not sampled
                   q[t,d]
                  )
                }
        }
}
```

Listing 4: JAGS model for EVL model to simulate human learning behavior in IGT

References

[1] Ronald A. Fisher. "On the Mathematical Foundations of Theoretical Statistics". In: *Philosophical Trans- actions of the Royal Society A* 222 (1922), pp. 309–368.

[2] Marc S. Paolella. *Fundamental Statistical Inference - A Computational Approach*. Probability and Statistics. John Wiley & Sons, 2018.