$\underline{X} = (X_1, \dots, X_n)^T$
$\underline{Y} = (Y_1, \dots, Y_n)^T$
} continuous n-dimensional random vectors

we suppose $\exists g(\cdot)$ invertible s.t. $\exists$ the partial derivatives and the Jacobian matrix has determinant $\neq 0$.

$\underline{X} \in \mathbb{R}^n$ random vector

$f_{\underline{X}}(\underline{x})$ is the density function that describes the distribution of $\underline{X}$

We express the density function of $\underline{Y}$:

$$h_{\underline{Y}}(\underline{y}) = f_{\underline{X}}(g^{-1}(\underline{y})) \cdot \| J_{g^{-1}} \|$$

absolute value of the determinant of the Jacobian of the inverse of g

$$J_{g^{-1}} = \begin{bmatrix} \frac{\partial x_1}{\partial y_1} & \dots & \frac{\partial x_1}{\partial y_n} \\ \vdots & & \\ \frac{\partial x_n}{\partial y_1} & \dots & \frac{\partial x_n}{\partial y_n} \end{bmatrix}$$

we're defining a new random variable by applying some transfor. (new n-dim rand. var.)

GAMMA RANOM VARIABLES

Gamma function:

$$\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} \, dt$$

properties:

- $\Gamma(1) = 1$
- $\Gamma(\frac{1}{2}) = \sqrt{\pi}$
- $\Gamma(\alpha+1) = \alpha \Gamma(\alpha) \quad \forall \alpha > 0$
- $\Gamma(n+1) = n! \quad \forall n \in \{0, 1, \dots\}$
- $\Gamma(\alpha+1) = \int_0^\infty x^\alpha e^{-x} \, dx = \int_0^\infty x^\alpha \frac{d}{dx}(-e^{-x}) \, dx$

proof.

$$= \left[-x^\alpha e^{-x} \right]_0^{+\infty} + \alpha \int_0^\infty x^{\alpha-1} e^{-x} \, dx$$

$$= \alpha \Gamma(\alpha)$$

We want to define a random variable which density is proportional to: $t^{\alpha-1} e^{-t}$.

X random variable s.t. $f_X(x) \propto x^{\alpha-1} e^{-x} \, \mathbb{1}_{(0,\infty)}(x)$

$$f_X(x) = \frac{x^{\alpha-1} e^{-x} \, \mathbb{1}_{(0,\infty)}(x)}{\int_0^\infty t^{\alpha-1} e^{-t} \, dt}$$

$$= \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x} \, \mathbb{1}_{(0,\infty)}(x) \implies X \sim \text{Gamma}(\alpha, 1) \qquad \alpha \in \mathbb{R}^+$$

random var. taking values in $(0, \infty)$

We want to introduce another parameter to governate the shape. We'll introduce it by a transformation of $X \sim \text{Gamma}(\alpha, 1)$.

$$U := \frac{X}{\beta}, \quad \beta > 0$$

$$X = \beta U \implies dx = \beta \, du$$

The density of U becomes:

$$f_U(u) = f_X(\beta u)\cdot \beta \, \mathbb{1}_{(0,\infty)}(u)$$

$$= \frac{1}{\Gamma(\alpha)}(\beta u)^{\alpha-1} e^{-\beta u}\cdot \beta \, \mathbb{1}_{(0,\infty)}(u)$$

$$= \frac{1}{\Gamma(\alpha)} \beta^{\alpha} u^{\alpha-1} e^{-\beta u} \, \mathbb{1}_{(0,\infty)}(u) \longrightarrow U \sim \text{Gamma}(\alpha, \beta)$$

α = shape parameter
β = rate parameter
$1/\beta$ = scale parameter $[\cdot\cdot] \, e^{-\frac{u}{\beta}}$

random var. taking values in $(0,\infty)$

To study the impact of the parameters:

$$\mathbb{E}[U^n] = \int_{\mathbb{R}} u^n \frac{\beta^{\alpha}}{\Gamma(\alpha)} u^{\alpha-1} e^{-\beta u} \mathbb{1}_{(0,\infty)}(u) \, du$$

$$= \frac{\beta^{\alpha}}{\Gamma(\alpha)} \int_0^{\infty} u^{\alpha+n-1} e^{-\beta u} \, du$$

kernel of a Gamma random var: $\text{Gamma}(\alpha+n, \beta)$

$$= \frac{\beta^{\alpha}}{\Gamma(\alpha)} \frac{\Gamma(\alpha+n)}{\beta^{\alpha+n}} \int_0^{\infty} \frac{\beta^{\alpha+n}}{\Gamma(\alpha+n)} u^{\alpha+n-1} e^{-\beta u} \, du$$

$$= \frac{1}{\beta^n} \frac{\Gamma(\alpha+n)}{\Gamma(\alpha)} = \frac{1}{\beta^n} \cdot \boxed{(n+\alpha-1)(n+\alpha-2)\cdots\alpha} = \frac{(\alpha)_n}{\beta^n}$$

$:=$ rising factorial in n

RISING FACTORIAL of a in b:
$\frac{\Gamma(a+b)}{\Gamma(a)} = (a)_b$ = truncation of a factorial term

$$\longrightarrow \mathbb{E}[U] = \frac{\alpha}{\beta}$$

$$\text{Var}(U) = \mathbb{E}[U^2] - \mathbb{E}^2[U] = \frac{\alpha}{\beta^2}$$

Special cases of the Gamma distribution:

1. $\text{Gamma}(1, \beta) \sim \mathcal{E}(\beta)$
2. $\text{Gamma}\left(\frac{n}{2}, \frac{1}{2}\right) \sim \chi^2(n)$
3. $U \sim \text{Gamma}(n, \beta) \longrightarrow 2\beta U \sim \text{Gamma}\left(\frac{2n}{2}, \frac{1}{2}\right) \sim \chi^2(2n)$! (E)
4. $Z = U^{-1}$, $U \sim \text{Gamma}(\alpha, \beta) \longrightarrow Z \sim$ inverse-gamma (α, β)

Properties:

1. Additivity: $U_1, \dots, U_k \overset{iid}{\sim} \text{Gamma}(\alpha_j, \beta)$ $j = 1, \dots, k$ $\left(U_j \sim \text{Gamma}(\alpha_j, \beta_j)\right)$

$$\longrightarrow \sum_{j=1}^{k} U_j \sim \text{Gamma}\left(\sum_{j=1}^{k} \alpha_j, \beta\right)$$

2. $U_1 \sim \text{Gamma}(\alpha_1, \beta)$, $U_2 \sim \text{Gamma}(\alpha_2, \beta)$ $\longrightarrow$

$$(U_1 + U_2) \perp\!\!\!\perp \frac{U_1}{U_1 + U_2}$$

$$(U_1 + U_2) \perp\!\!\!\perp \frac{U_2}{U_1 + U_2}$$

BETA RANDOM VARIABLES

Beta function:

$$\beta(\alpha_1,\alpha_2) = \int_0^1 t^{\alpha_1-1}(1-t)^{\alpha_2-1}\,dt = \frac{\Gamma(\alpha_1)\Gamma(\alpha_2)}{\Gamma(\alpha_1+\alpha_2)}$$

Similarly to the previous case:
We want to define a random var. R:

$$f_R(x) \propto x^{\alpha_1-1}(1-x)^{\alpha_2-1}\,\mathbb{1}_{(0,1)}(x)$$

$$\Rightarrow f_R(x) = \frac{x^{\alpha_1-1}(1-x)^{\alpha_2-1}}{\int_0^1 t^{\alpha_1-1}(1-t)^{\alpha_2-1}\,dt}\,\mathbb{1}_{(0,1)}(x)$$

$$= \frac{1}{\beta(\alpha_1,\alpha_2)}\,x^{\alpha_1-1}(1-x)^{\alpha_2-1}\,\mathbb{1}_{(0,1)}(x) \Rightarrow R \sim \text{Beta}(\alpha_1,\alpha_2)$$

random var. taking values in $(0,1)$

Property:

- $U_1 \sim \text{Gamma}(\alpha_1,\beta)$, $U_2 \sim \text{Gamma}(\alpha_2,\beta)$ $\Rightarrow$
  - $\frac{U_1}{U_2+U_1} \sim \text{Beta}(\alpha_1,\alpha_2)$
  - $\frac{U_2}{U_2+U_1} \sim \text{Beta}(\alpha_2,\alpha_1)$

Again: n-th moment:

$$\mathbb{E}[R^n] = \int r^n \frac{1}{\beta(\alpha_1,\alpha_2)} r^{\alpha_1-1}(1-r)^{\alpha_2-1}\,\mathbb{1}_{(0,1)}(r)\,dr$$

$$= \frac{1}{\beta(\alpha_1,\alpha_2)}\int_0^1 r^{\alpha_1+n-1}(1-r)^{\alpha_2-1}\,dr \quad [..]$$

$$= \frac{\beta(\alpha_1+n,\alpha_2)}{\beta(\alpha_1,\alpha_2)} = \frac{(\alpha_1)_n}{(\alpha_1+\alpha_2)_n} = \frac{(\alpha_1+n-1)(\alpha_1+n-2)\cdots\alpha_1}{(\alpha_1+\alpha_2+n-1)(\alpha_1+\alpha_2+n-2)\cdots(\alpha_1+\alpha_2)}$$

$$\Rightarrow \mathbb{E}[R] = \frac{\alpha_1}{\alpha_1+\alpha_2}$$

$$\text{Var}(R) = \mathbb{E}[R^2] - \mathbb{E}^2[R] = \frac{\alpha_1\alpha_2}{(\alpha_1+\alpha_2+1)(\alpha_1+\alpha_2)^2}$$

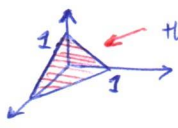
Special case:

1. $\text{Beta}(1,1) \sim \mathcal{U}([0,1])$

Properties:

1. $X \sim \text{Beta}(a,b) \Rightarrow 1-X \sim \text{Beta}(b,a)$
2. if $\alpha_1 > 1$, $\alpha_2 > 1$ $\Rightarrow$ mode $= \frac{\alpha_1-1}{\alpha_1+\alpha_2-1}$

Betas are useful to model probabilities (since it takes values in $(0,1)$).
If we want to model more than one outcome (more than one probability) the Dirichlet distribution is one of the possible multivariate extensions of the Beta.
it takes values in the simplex:



DIRICHLET DISTRIBUTION

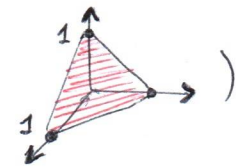
We start from considering more than 2 gammas.

$U_1, \dots, U_k \overset{iid}{\sim} \text{Gamma}(\alpha_j, \beta)$ $\quad (U_j \sim \text{Gamma}(\alpha_j, \beta))$

We define: $X_j = \dfrac{U_j}{U_1 + \dots + U_k}$.

The vector $(X_1, \dots, X_k) \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_k)$

We impose: $\sum_{j=1}^k U_j = 1$ (and so $(X_1, \dots, X_k)$ takes values in:



$\Longrightarrow (X_1, \dots, X_k) \in \Delta_{k-1} = \left\{ \underline{x} \in \mathbb{R}^k : \sum_{j=1}^k x_j = 1, \quad x_j \in [0,1] \right\}$

$k-1$ simplex

Density of a Dirichlet distribution:

$$f_X(\underline{x}) = \frac{1}{B(\underline{\alpha})} \prod_{j=1}^k x_j^{\alpha_j - 1} \mathbb{1}_{\Delta_{k-1}}(\underline{x}) \qquad B(\underline{\alpha}) = \frac{\Gamma(\alpha_1) \cdots \Gamma(\alpha_k)}{\Gamma(\alpha_1 + \dots + \alpha_k)}$$

$$= \frac{1}{B(\underline{\alpha})} \left[\prod_{j=1}^{k-1} x_j^{\alpha_j - 1} \right] \left(1 - \sum_{j=1}^{k-1} x_j \right)^{\alpha_k - 1} \mathbb{1}_{\Delta_{k-1}}(\underline{x})$$

For simplicity: $\alpha_0 = \sum_{j=1}^k \alpha_j$

Property:

$$\mathbb{E}[X_j] = \frac{\alpha_j}{\alpha_0} \qquad j = 1, \dots, k$$

because $X_j = \dfrac{U_j}{U_1 + \dots + U_k} = \dfrac{U_j}{U_j + \sum_{\substack{l=1 \\ l \neq j}}^k U_l}$

and so we're in the case of gammas (we have the property)

$$\text{Var}(X_j) = \frac{\alpha_j (\alpha_0 - \alpha_j)}{\alpha_0^2 (\alpha_0 + 1)}$$

$$\text{Cov}(X_j, X_l) = -\frac{\alpha_l \alpha_j}{\alpha_0^2 (\alpha_0 + 1)} \qquad j \neq l$$

(Other) properties:

1. The marginal distribution is: $X_j \sim \text{Beta}(\alpha_j, \alpha_0 - \alpha_j)$
2. Aggregation property:

$(X_1, \dots, X_n) \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_n)$

$(Z_1, \dots, Z_m) := \left(\sum_{i \in C_1} X_i, \dots, \sum_{i \in C_m} X_i \right)$

where $\begin{cases} C_i \subseteq \{1, \dots, n\} & \forall i \\ C_i \cap C_j = \emptyset & \forall i \neq j \\ \bigcup_{i=1}^m C_i = \{1, \dots, n\} \end{cases}$

we sum up some dimensions ($m \leq n$)

$\Longrightarrow (Z_1, \dots, Z_m) \sim \text{Dirichlet}(\alpha_1^*, \dots, \alpha_m^*)$

$\alpha_j^* = \sum_{i \in C_j} \alpha_j$

3. $(X_1, \dots, X_k) \sim \text{Dir}(\alpha_1, \dots, \alpha_k) \longrightarrow \begin{cases} X_1 \sim \text{Beta}(\alpha_1, \alpha_0 - \alpha_1) \\ X_1 \perp\!\!\!\perp \left(\frac{X_2}{X_1}, \dots, \frac{X_k}{X_1} \right) \\ \left(\frac{X_2}{1 - X_1}, \dots, \frac{X_k}{1 - X_1} \right) \sim \text{Dir}(\alpha_2, \dots, \alpha_k) \end{cases}$

we can take out one dimension

proof that we can take out a dimension:

We do a transformation:

$$g(x_1, \dots, x_{k-1}) = \begin{cases} Q_1 = x_1 & \in (0,1) \\ Q_2 = \dfrac{x_2}{1-x_1} & \\ \vdots & \\ Q_{k-1} = \dfrac{x_{k-1}}{1-x_1} & \in \Delta_{k-2} \end{cases}$$

We have that $Q_k = \sum_{j=2}^{k-1} Q_j$.

$$g^{-1}(Q_1, \dots, Q_{k-1}) = \begin{cases} x_1 = Q_1 \\ x_2 = Q_2(1-Q_1) \\ \vdots \\ x_{k-1} = Q_{k-1}(1-Q_1) \end{cases}$$

$$J_{g^{-1}} = \begin{bmatrix} \frac{\partial x_1}{\partial Q_1} & \dots & \frac{\partial x_1}{\partial Q_{k-1}} \\ \vdots & & \\ \frac{\partial x_{k-1}}{\partial Q_1} & \dots & \frac{\partial x_{k-1}}{\partial Q_{k-1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ -Q_2 & 1-Q_1 & 0 & 0 \\ \vdots & & 1-Q_1 & \vdots \\ & & & 0 \\ & & & 1-Q_1 \end{bmatrix}$$

$$\| J_{g^{-1}} \| = 1 \cdot (1-Q_1)^{k-2}$$

Starting from the density of X we want to derive the density of Q:

$$f_{\underline{Q}}(q) = f_{\underline{X}}(g^{-1}(q)) \cdot \| J_{g^{-1}} \|$$

$$= \frac{1}{\beta(\underline{\alpha})} \left[\prod_{j=2}^{k-1} (q_j (1-q_1))^{\alpha_j - 1} \right] \cdot q_1^{\alpha_1 - 1} \cdot \left(1 - \sum_{j=2}^{k-1} q_j (1-q_1) - q_1\right)^{\alpha_k - 1} \cdot (1-q_1)^{k-2}$$

($\| J_{g^{-1}} \|$)

$$= \frac{1}{\beta(\underline{\alpha})} q_1^{\alpha_1 - 1} (1-q_1)^{\sum_{j=2}^{k-1} \alpha_j} (1-q_1)^{\alpha_k - 1} \left[\prod_{j=2}^{k-1} q_j\right] \left[1 - \sum_{j=2}^{k-1} q_j\right]^{\alpha_k - 1} (1-q_1)^{k-2}$$

$$= \frac{1}{\beta(\underline{\alpha})} q_1^{\alpha_1 - 1} (1-q_1)^{\sum_{j=1}^{k} \alpha_j - 1} (1-q_1)^{k-2} \left[\prod_{j=2}^{k-1} q_j\right] \left[1 - \sum_{j=2}^{k-1} q_j\right]^{\alpha_k - 1}$$

$$= \frac{1}{\beta(\underline{\alpha})} q_1^{\alpha_1 - 1} (1-q_1)^{\alpha_0 - \alpha_1 - 1} \left[\prod_{j=2}^{k-1} q_j\right] \left[1 - \sum_{j=2}^{k-1} q_j\right]^{\alpha_k - 1}$$

Beta random var. up to a normalization constant

$\text{Beta}(\alpha_1, \alpha_0 - \alpha_1)$

Dirichlet distribution defined on the $k-2$ dimensional simplex: the parameter α_1 is not contributing in any way to the shape of the distribution

■

let's now see some discrete distributions.

BERNOULLI DISTRIBUTION

$X \in \{0,1\}$ describing the success/failure.

We want to model the randomness behind the trial.

We consider $\theta \in (0,1)$; (θ random var.)

$$p_X(x) = \theta^x (1-\theta)^{1-x} \qquad (\theta = \mathbb{P}(\text{success})) \implies X \sim Be(\theta)$$

probability mass function

$$\mathbb{E}[X] = \theta$$
$$Var(X) = \theta(1-\theta)$$

More than one trial:

BINOMIAL DISTRIBUTION

$X \in \{0, 1, \dots, n\}$, $\theta \in (0,1)$

$$p_X(x) = \binom{n}{x} \theta^x (1-\theta)^{n-x} \implies X \sim Bi(n, \theta)$$

(it describes the outcome of a sequence of trials)

$$\mathbb{E}[X] = n\theta$$
$$Var(X) = n\theta(1-\theta)$$

MULTINOMIAL DISTRIBUTION

multivariate extension of the Binomial distribution

$\underline{R}$ = k-dimensional random vector characterized by : $\underline{\lambda} = (\lambda_1, \dots, \lambda_k)$

where λ_j = probability of success for the j-th dimension

$$\sum_{j=1}^{k} \lambda_j = 1 \ , \quad \underline{\lambda} \in \Delta_{k-1}$$

$$p_{\underline{R}}(\underline{r}) = \frac{m!}{r_1! \cdots r_k!} \lambda_1^{r_1} \cdots \lambda_k^{r_k} \ \mathbb{1}\left\{ \begin{array}{l} \sum r_j = m, \\ r_j \in \{0, \dots, m\} \end{array} \right\}$$

r_j = # success in the j-th dimension

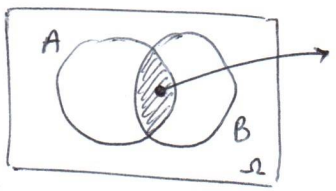
m = # trials (Total)

$$\mathbb{E}[R_j] = m \cdot \lambda_j$$
$$Var(R_j) = m \lambda_j (1-\lambda_j)$$
$$Cov(R_i, R_j) = -m \lambda_i \lambda_j \qquad i \neq j$$

Bayes theorem:

A, B events, $\mathbb{P}(B) > 0$: $\quad \mathbb{P}(A|B) = \dfrac{\mathbb{P}(B|A)\,\mathbb{P}(A)}{\mathbb{P}(B)}$

Intuitively: $A, B \in \Omega$:



$A \cap B$: $\quad \mathbb{P}(A|B) \propto \mathbb{P}(A \cap B)$

In the case of probability mass functions:

$$p_{X|Y}(x|y) = \frac{p_{Y|X}(y|x)\,p_X(x)}{p_Y(y)} = \frac{p_{Y|X}(y|x)\,p_X(x)}{\sum_x p_{X,Y}(x,y)}$$

$$= \frac{p_{Y|X}(y|x)\,p_X(x)}{m(y)} \propto p_{X,Y}(x,y)$$

we can update our believes just working proportionally to the joint distribution

Similarly, with density functions:

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)\,f_X(x)}{f_Y(y)} = \frac{f_{Y|X}(y|x)\,f_X(x)}{\int f_{X,Y}(x,y)\,dx}$$

$$= \frac{f_{Y|X}(y|x)\,f_X(x)}{m(y)}$$

The important thing is that we can use our prior belief to update the posterior (together with the data)

Remark: $\mathbb{P}(B|A_1, A) \propto \mathbb{P}(A_1, A_2|B)\,\mathbb{P}(B)$

$\propto \mathbb{P}(A_2|B, A_1)\,\mathbb{P}(A_1|B)\,\mathbb{P}(B)$

we can sequentially update our believes

Def. **Conjugacy**: The distribution of X is conjugate to the distribution of Y if $Y|X$ is in the same family of distributions of X.
(when we update our prior belief we stay in the same family of distributions)

BAYESIAN POINT ESTIMATION

We denote by Θ = set of parameters = set of all the possible states θ, $\theta \sim \pi(\theta)$.
We have a set of possible actions $\mathcal{A}$: $a \in \mathcal{A}$ is a single action.
We define a loss function: $\ell : \Theta \otimes \mathcal{A} \longrightarrow \mathbb{R}$, $\quad \ell : (\theta, a) \mapsto \ell(\theta, a)$.

meaning: when the state is θ, how much do we lose choosing the action a?

We work with:

$\mathbb{E}_\pi[\ell(\theta, a)]$ = prior expected loss.

The posterior expected loss will be:

$\mathbb{E}_\pi[\ell(\theta, a)|\underline{X}]$

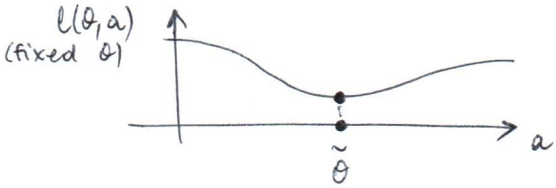
where: $\begin{cases} X_1, \dots, X_n|\theta \sim f(x, \theta) \\ \theta \sim \pi(\theta) \quad \text{s.t.} \quad \theta \in \Theta \subseteq \mathbb{R}^k \end{cases}$

So, we choose an action a to which corresponds a loss. Due to the fact that ϑ (based on which we choose a) is random, instead of minimizing $\ell(\theta, a)$ we have to take care also of the distribution of ϑ. And so, we work with prior and posterior expectation of $\ell(\theta, a)$ w.r.t. π.

A way to choose an optimal θ (in $\pi(\theta)$) is to set $\mathcal{A} = \Theta$:

$$\Longrightarrow \quad \arg\min_{a \in \Theta} \mathbb{E}_\pi[\ell(\theta, a) \mid \underline{X}] = \tilde{\theta}$$

We can choose the loss function in several ways.
A good choice is a convex function:



usually convex loss functions (unique minimum)

Different loss functions lead to different point estimations.

Exercise 1. (Quadratic loss function leads to posterior expectation)

$\theta \in \Theta \subseteq \mathbb{R}$, $\mathcal{A} \equiv \Theta$, $\ell(\theta, a) := (\theta - a)^2$

If we assume this loss function:

$$\Longrightarrow \tilde{\theta} = \mathbb{E}_\pi[\theta \mid \underline{X}] = \arg\min_{a \in \Theta} \mathbb{E}_\pi[\ell(\theta, a) \mid \underline{X}]$$

↑ we have to prove it

$$\mathbb{E}_\pi[\ell(\theta, a) \mid \underline{X}] = \int_\Theta \ell(\theta, a)\, \pi(\theta \mid \underline{X})\, d\theta$$
$$= \int_\Theta (\theta - a)^2\, \pi(\theta \mid \underline{X})\, d\theta$$
$$= \int_\Theta (\theta^2 - 2a\theta + a^2)\, \pi(\theta \mid \underline{X})\, d\theta$$

$\pi(\theta \mid \underline{X})$ = posterior distribution

$$\frac{d}{da}(\theta^2 - 2a\theta + a^2) = 2a - 2\theta$$

We have that the minimum is given by:

$$\int_\Theta (2a - 2\theta)\, \pi(\theta \mid \underline{X})\, d\theta = 0 \quad \Longrightarrow \quad 2a \underbrace{\int_\Theta \pi(\theta \mid \underline{X})\, d\theta}_{1} = 2 \underbrace{\int_\Theta \theta\, \pi(\theta \mid \underline{X})\, d\theta}_{\mathbb{E}[\theta \mid \underline{X}]}$$

$$\Longrightarrow a = \tilde{\theta} = \mathbb{E}_\pi[\theta \mid \underline{X}] = \text{the point estimate is the posterior mean}$$

Exercise 2. (Quadratic loss function leads to posterior expectation (higher dimension))

Same as before but: $\ell(\underline{\theta}, \underline{a}) = (\underline{\theta} - \underline{a})^T Q (\underline{\theta} - \underline{a})$, Q = simmetric positive semidefinite matrix

(we have higher dimensions $\underline{\theta} \in \Theta \subseteq \mathbb{R}^p$, $\mathcal{A} \equiv \Theta$)

$$\Longrightarrow \tilde{\underline{\theta}} = \mathbb{E}_\pi[\underline{\theta} \mid \underline{X}] = \arg\min_{\underline{a} \in \Theta} \mathbb{E}_\pi[\ell(\underline{\theta}, \underline{a}) \mid \underline{X}]$$

↑ we want to prove this

We directly start from:

$$\frac{d}{d\underline{a}} \mathbb{E}_\pi[\ell(\underline{\theta}, \underline{a}) \mid \underline{X}] = \frac{d}{d\underline{a}} \int_\Theta \underbrace{(\underline{\theta} - \underline{a})^T Q (\underline{\theta} - \underline{a})}_{:= F(\underline{a})}\, \pi(\underline{\theta} \mid \underline{X})\, d\underline{\theta}$$

$$F(\underline{a}) = \sum_{i,j=1}^{p} q_{ij} (\theta_i - a_i)(\theta_j - a_j) \qquad \text{where } q_{ij} = [Q]_{ij}$$

$$\frac{d}{da_\ell} F(\underline{a}) = \sum_{i,j=1}^{p} \frac{d}{da_\ell} q_{ij} (\theta_i - a_i)(\theta_j - a_j)$$

$$= \sum_{i,j=1}^{p} q_{ij} \frac{d}{da_\ell}\left[(\theta_i - a_i)(\theta_j - a_j)\right] + \sum_{i,j=1}^{p} q_{ij} (\theta_i - a_i) \frac{d}{da_\ell}(\theta_j - a_j)$$

$$= -\sum_{j=1}^{p} q_{\ell j} (\theta_j - a_j) - \sum_{i=1}^{p} q_{i\ell} (\theta_i - a_i)$$

$$= -2 \sum_{j=1}^{p} q_{\ell j} (\theta_j - a_j)$$

$$= \left[-2Q(\underline{\theta} - \underline{a})\right]_\ell$$

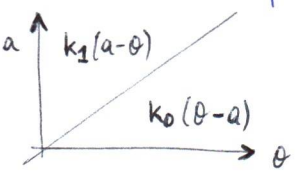
$$\Rightarrow \frac{d}{d\underline{a}} \mathbb{E}_\pi\left[\ell(\underline{\theta}, \underline{a}) \mid \underline{X}\right] = \int_\Theta -2Q(\underline{\theta} - \underline{a})\, \pi(\underline{\theta} \mid \underline{X})\, d\underline{\theta} = 0$$

$$\Rightarrow -2Q \underbrace{\int_\Theta \underline{\theta}\, \pi(\underline{\theta} \mid \underline{X})\, d\underline{\theta}}_{\mathbb{E}_\pi[\underline{\theta} \mid \underline{X}]} = -2Q \int_\Theta \underline{a}\, \pi(\underline{\theta} \mid \underline{X})\, d\underline{\theta}$$

$$\rightarrow \underline{a} = \tilde{\underline{\theta}} = \int_\Theta \underline{\theta}\, \pi(\underline{\theta} \mid \underline{X})\, d\underline{\theta} = \mathbb{E}_\pi[\underline{\theta} \mid \underline{X}]$$

Exercise 3. (Linear loss function leads to quantiles)

$\theta \in \Theta \subseteq \mathbb{R}$, $\mathcal{A} = \Theta$, $\ell(\theta, a) = \begin{cases} k_0(\theta - a) & a < \theta \\ k_1(a - \theta) & a \geq \theta \end{cases}$



With this loss function our point estimate will be:

$\tilde{\theta}$ = quantile of $\pi(\theta \mid \underline{X})$ of order $\frac{k_0}{k_1 + k_0}$ ← we want to prove this

We start from:

$$\mathbb{E}_\pi\left[\ell(\theta, a) \mid \underline{X}\right] = \int_\Theta \ell(\theta, a)\, \pi(\theta \mid \underline{X})\, d\theta$$

$$= \int_\Theta k_0(\theta - a)\, \mathbb{1}_{(a<\theta)}\, \pi(\theta \mid \underline{X})\, d\theta + \int_\Theta k_1(a - \theta)\, \mathbb{1}_{(a \geq \theta)}\, \pi(\theta \mid \underline{X})\, d\theta$$

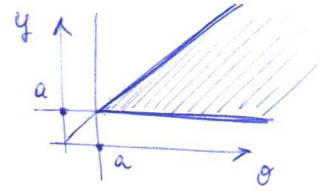
$$= \int_a^\infty k_0(\theta - a)\, \pi(\theta \mid \underline{X})\, d\theta + \int_{-\infty}^a k_1(a - \theta)\, \pi(\theta \mid \underline{X})\, d\theta$$

$$= k_0 I_0 + k_1 I_1$$

$$I_0 = \int_a^\infty (\theta - a)\, \pi(\theta \mid \underline{X})\, d\theta = \int_a^\infty \int_a^\theta dy\, \pi(\theta \mid \underline{X})\, d\theta$$

$$= \int_a^\infty \int_y^\infty \pi(\theta \mid \underline{X})\, d\theta\, dy$$

$$= \int_a^\infty \mathbb{P}(\theta > y \mid \underline{X})\, dy$$



$$I_1 = \int_{-\infty}^a (a - \theta)\, \pi(\theta \mid \underline{X})\, d\theta = \int_{-\infty}^a \int_\theta^a dy\, \pi(\theta \mid \underline{X})\, d\theta$$

$$= [\ldots] = \int_{-\infty}^a \mathbb{P}(\theta < y \mid \underline{X})\, dy$$

$$\Longrightarrow \quad \mathbb{E}_\pi[\ell(\theta,a)|\underline{X}] = k_0 \int_a^{\infty} \mathbb{P}(\theta > y|\underline{X})\, dy + k_1 \int_{-\infty}^{a} \mathbb{P}(\theta < y|\underline{X})\, dy$$

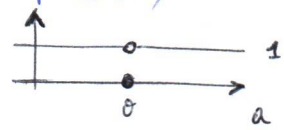
$$\frac{d}{da}\mathbb{E}_\pi[\ell(\theta,a)|\underline{X}] = -k_0\, \mathbb{P}(\theta > a|\underline{X}) + k_1\, \mathbb{P}(\theta < a|\underline{X})$$
$$\overset{!}{=} -k_0\,(1 - \mathbb{P}(\theta \le a|\underline{X})) + k_1\, \mathbb{P}(\theta < a|\underline{X}) = 0$$

$$\Longrightarrow \quad \mathbb{P}(\theta < a|\underline{X}) = \frac{k_0}{k_1+k_0} \quad \Longrightarrow \quad \mathbb{P}(\theta < \tilde{\theta}|\underline{X}) = \frac{k_0}{k_1+k_0}$$

$\tilde{\theta}$ is the quantile of order $\frac{k_0}{k_1+k_0}$

Exercise 4. (0-1 loss function leads to the mode of the posterior)

$\theta \in \Theta \subseteq \mathbb{R}$, $\mathcal{A} \equiv \Theta$, $\ell(\theta,a) = \begin{cases} 0 & \theta = a \\ 1 & \theta \neq a \end{cases}$:



we're penalizing everything in the same way except one point

We separate two cases: discrete, continuous.

- Discrete case:

$$\begin{aligned}
\arg\min_{a\in\Theta} \mathbb{E}_\pi[\ell(\theta,a)|\underline{X}] &= \arg\min_{a\in\Theta} \sum_{a\in\Theta} \mathbb{1}_{(\theta\neq a)}\, \pi(\theta|\underline{X}) \\
&= \arg\min_{a\in\Theta} \left(1 - \sum_{a\in\Theta} \mathbb{1}_{(\theta=a)}\, \pi(\theta|\underline{X})\right) \\
&= \arg\min_{a\in\Theta} \left(1 - \pi(a|\underline{X})\right) \\
&= \arg\max_{a\in\Theta} \pi(a|\underline{X}) \quad \Longrightarrow
\end{aligned}$$

the point estimate with the 0-1 loss function will be the posterior mode (the mode of the posterior distribution) = point where the posterior has more mass

- Continuous case:

$$\ell_\varepsilon(\theta,a) = \begin{cases} 0 & |\theta-a| < \varepsilon \\ 1 & |\theta-a| \ge \varepsilon \end{cases}$$

$$\begin{aligned}
\arg\min_{a\in\Theta} \mathbb{E}_\pi[\ell(\theta,a)|\underline{X}] &= \arg\min_{a\in\Theta} 1 - \int_\Theta \mathbb{1}_{(a-\varepsilon,a+\varepsilon)}(\theta)\, \pi(\theta|\underline{X})\, d\theta \\
&= \arg\max_{a\in\Theta} \int_\Theta \mathbb{1}_{(a-\varepsilon,a+\varepsilon)}(\theta)\, \pi(\theta|\underline{X})\, d\theta \\
&= \arg\max_{a\in\Theta} \pi(da|\underline{X}) \quad \text{for } \varepsilon \to 0
\end{aligned}$$

$\Longrightarrow$ corresponds again to the posterior mode (mode of the posterior distribution)

Remark: the multinomial distribution is conjugate to the dirichlet distribution

Consider a set of observations: $N_1, \dots, N_k$; $n := \sum_{i=1}^k N_i$
(coming from a k-dimensional multinomial distribution)
We set as likelihood: $(N_1, \dots, N_k) \sim$ multinomial $(\lambda_1, \dots, \lambda_k)$.
We set as prior for $\lambda_1, \dots, \lambda_k$:

$$(\lambda_1, \dots, \lambda_k) \sim \text{dirichlet}(\alpha_1, \dots, \alpha_k)$$

likelihood:

$$p(N_1, \dots, N_k \mid \lambda_1, \dots, \lambda_k) = \frac{n!}{N_1! \cdots N_k!} \lambda_1^{N_1} \cdots \lambda_k^{N_k}$$

Prior:

$$\pi(\lambda_1, \dots, \lambda_k) = \frac{1}{\beta(\underline{\alpha})} \left[\prod_{j=1}^k \lambda_j^{\alpha_j - 1} \right] \mathbb{1}_{\Delta_{k-1}}(\underline{\lambda})$$

We want to show that $\underline{\lambda} \mid \underline{N} \sim$ Dirichlet:

$$\pi(\underline{\lambda} \mid \underline{N}) = \frac{p(\underline{N} \mid \underline{\lambda})\, \pi(\underline{\lambda})}{m(\underline{N})} \propto p(\underline{N} \mid \underline{\lambda})\, \pi(\underline{\lambda}) =$$

$$= \left[\prod_{j=1}^k \lambda_j^{N_j} \right] \left[\prod_{j=1}^k \lambda_j^{\alpha_j - 1} \mathbb{1}_{\Delta_{k-1}}(\underline{\lambda}) \right]$$

$$= \prod_{j=1}^k \lambda_j^{\alpha_j + N_j - 1} \mathbb{1}_{\Delta_{k-1}}(\underline{\lambda})$$

kernel of a dirichlet distribution
$\sim$ dirichlet $(\alpha_1 + N_1, \dots, \alpha_k + N_k)$

Exercise.
We have an urn full of balls of different colors.
We have $k = 3$ different colors. We do a prior guess of the composition:

- 0.1 red balls
- 0.4 blue balls
- 0.5 green balls

We draw $n = 100$ balls with replacement and we get:

- $N_1 = 33$ red balls
- $N_2 = 15$ blue balls
- $N_3 = 52$ green balls

$$\Longrightarrow \begin{cases} \pi_1(\underline{\lambda}) \overset{d}{=} \text{Dirichlet}(0.1, 0.4, 0.5) \\ \pi_1(\underline{\lambda} \mid \underline{N}) \overset{d}{=} \text{Dirichlet}(0.1 + 33,\ 0.4 + 15,\ 0.5 + 52) = \text{Dir}(\alpha_1^*, \alpha_2^*, \alpha_3^*) \end{cases}$$

We want to study a bit this distribution:

$\Longrightarrow$ point estimate under the quadratic loss function: = posterior mean

$$\mathbb{E}_{\pi_1}[\lambda_\ell \mid \underline{N}] = \frac{\alpha_\ell^*}{\alpha_1^* + \alpha_2^* + \alpha_3^*} \Longrightarrow \begin{cases} \hat{\lambda_1} = 0.328 \\ \hat{\lambda_2} = 0.152 \\ \hat{\lambda_3} = 0.520 \end{cases}$$

posterior point estimate for the composition of our urn

Another prior guess:

- 10 red balls
- 40 blue balls
- 50 green balls

This prior it not only the composition, it also assumes the number of balls in the urn!

$$\Longrightarrow \begin{cases} \pi_2(\underline{\lambda}) = \text{Dir}(10, 40, 50) \\ \pi_2(\underline{\lambda} \mid \underline{N}) = \text{Dir}(10+33,\ 40+15,\ 50+52) \end{cases}$$

Point estimate: $\mathbb{E}_{\pi_2}[\lambda_j \mid \underline{N}] = \dfrac{\alpha_j^*}{\alpha_1^* + \alpha_2^* + \alpha_3^*}$

$$\Longrightarrow \begin{cases} \hat{\lambda_1} = 0.215 \\ \hat{\lambda_2} = 0.275 \\ \hat{\lambda_3} = 0.510 \end{cases} \quad \neq \text{previous ones}$$

Exercise 1. NORMAL - INV- GAMMA MODEL (continuous data)

$$X_1, \dots, X_n \mid \mu, \sigma^2 \sim N(\mu, \sigma^2)$$

$$(\mu, \tau^2) \sim \text{Normal-inv-gamma}(m_0, k_0, a_0, b_0)$$

$$\mu \mid \sigma^2 \sim N\left(m_0, \frac{\sigma^2}{k_0}\right) \overset{d}{=} (2\pi)^{-\frac{1}{2}} \left(\frac{\sigma^2}{k_0}\right)^{-1/2} e^{\frac{k_0}{2\sigma^2}(\mu - m_0)^2} \mathbb{1}_{(-\infty,\infty)}(\mu)$$

$$\sigma^2 \sim \text{inv-gamma}(a_0, b_0) \overset{d}{=} \frac{b_0^{a_0}}{\Gamma(a_0)} (\sigma^2)^{-a_0-1} e^{-\frac{b_0}{\sigma^2}} \mathbb{1}_{(0,\infty)}(\sigma^2)$$

A normal-inverse-gamma, conditionally to a sample that is Gaussian distributed is still a normal-inverse-gamma.

Likelihood:

$$L(\mu, \sigma^2; \underline{X}) = \prod_{i=1}^n f_{X_i}(x_i, \mu, \sigma^2) = [\dots]$$

Prior:

$$\pi(\mu, \sigma^2)$$

Posterior:

$$\pi(\mu, \sigma^2 \mid \underline{X}) \propto \pi(\mu, \sigma^2)\, L(\mu, \sigma^2, \underline{X})$$

$$\sim \text{normal-inverse-gamma}(m_n, k_n, a_n, b_n)$$

$$\begin{cases} m_n = \dfrac{k_0 m_0 + n\bar{x}}{k_0 + n} \\ k_n = k_0 + n \\ a_n = a_0 + \dfrac{n}{2} \\ b_n = b_0 + \dfrac{1}{2}\displaystyle\sum_{i=1}^n (x_i - \bar{x}) + \dfrac{1}{2}\dfrac{k_0 n}{k_0 + n}(\bar{x} - m_0)^2 \end{cases}$$

How to choose m_0, k_0, a_0, b_0?
One way is to rely on the summaries of the posterior distributions and see what changes when we modify the parameters of the priors.
For example: let's elicit the parameters of the distribution part of μ.

Elicit the distribution of μ:

$$\begin{cases} \mu \mid \sigma^2 \sim N\left(m_0, \dfrac{\sigma^2}{k_0}\right) & \text{(prior)} \\ \mu \mid \sigma^2, \underline{X} \sim N\left(m_n, \dfrac{\sigma^2}{k_n}\right) & \text{(posterior)} \end{cases}$$

Taking for instance the post. mean:

$$\Rightarrow \mathbb{E}[\mu \mid \sigma^2, \underline{X}] = m_n = \frac{k_0 m_0 + n\bar{x}}{k_0 + n} = \frac{k_0}{k_0+n} m_0 + \frac{n}{k_0+n}\bar{x}$$

m_0: **prior guess** (completely driven by prior distribution)

$\bar{x}$: **maximum likelihood estimator** (completely driven by data)

We see that the parameter k_0 takes the same role as n: we can think about an hypothetical sintetic sample (that plays the role of the prior sample) and the observed sample. The hypothetical sintetic is of size k_0 and with "maximum likelihood estimator" m_0. This is how we translate the prior.

Elicit the distribution of σ^2:

$$\begin{cases} \sigma^2 \sim \text{inv-gamma}(a_0, b_0) \\ \sigma^2 \mid \underline{X} \sim \text{inv-gamma}(a_n, b_n) \end{cases}$$

$$\Longrightarrow \mathbb{E}[\sigma^2 | \underline{x}] = \frac{b_n}{a_n - 1} = \frac{1}{a_0 + \frac{n}{2} - 1}\left[b_0 + \frac{1}{2} S_n^2 + \frac{1}{2}\frac{k_0 n}{k_0 + n}(\bar{x} - m_0)^2 \right]$$

S_n^2 → $S_n^2 = \sum_{i=1}^n (x_i - \bar{x})^2$

$$= \frac{1}{a_0 + \frac{n}{2} - 1} b_0 + \frac{n-1}{a_0 + \frac{n}{2} - 1} S^2 + \frac{1}{2}\frac{k_0 n}{k_0 + n}(\bar{x} - m_0)^2 \cdot \frac{1}{a_0 + \frac{n}{2} - 1}$$

we discard this part ($n \to \infty$ this is vanishing)

S^2 → unbiased estimator of the variance

$$= \frac{a_0 - 1}{a_0 + \frac{n}{2} - 1} \cdot \frac{b_0}{a_0 - 1} + \frac{\frac{n-1}{2}}{a_0 + \frac{n}{2} - 1} S^2$$

$\frac{b_0}{a_0 - 1}$: expectation of the prior distribution on the variance term

S^2: sample variance

$a_0 - 1$ plays the same role as $(n-1)/2$, and $b_0/(a_0 - 1)$ plays the same role as S^2 (the rest of the interpretation is as before)

Exercise 2. BETA - BERNOULLI MODEL (0-1 data)

Elections in Florida.
They interviewed $n = 509$ people, $r = 279$ said they'll vote for Bush.
We model the situation as a Bernoulli model: success = vote for Bush.

$$X_1, \dots, X_n | \theta \sim Be(\theta) \qquad \theta = \mathbb{P}(\text{voting for Bush})$$

The probability function in this case is:

$$p_X(x|\theta) = \theta^x (1-\theta)^x \qquad x \in \{0,1\}$$

Likelihood:

$$L(\theta; \underline{x}) = \prod_{i=1}^n p_X(x_i|\theta) = \theta^r (1-\theta)^{n-r}$$

$$\Longrightarrow \hat{\theta}_{MLE} = \arg\max_{\theta \in (0,1)} L(\theta, \underline{x}) = \frac{r}{n} = 0.548 \Longrightarrow$$ according to this Bush is going to win

What if we include some priors? Always look at the likelihood and try to find something of a similar form (maybe we'll obtain a conjugate prior!)

$$\theta \sim Beta(a,b) : \quad \pi(\theta) = \frac{1}{B(a,b)} \theta^{a-1} (1-\theta)^{b-1}$$

Posterior:

$$\pi(\theta|\underline{x}) \propto \pi(\theta) L(\theta, \underline{x}) = \underbrace{\theta^{a+r-1} (1-\theta)^{b+n-r-1}}$$

kernel of a $Beta(a+r, b+n-r)$
where $r = \sum_{i=1}^n x_i$

1. What kind of model are we assuming for the data? (Discarding θ)
= What is the marginal distribution of the data?

$$m(\underline{x}) = \int_\Theta L(\theta, \underline{x}) \pi(d\theta) = \int_\Theta \theta^r (1-\theta)^{n-r} \frac{1}{B(a,b)} \theta^{a-1} (1-\theta)^{b-1} \mathbb{1}_{(0,1)}(\theta) \, d\theta$$

$$= \frac{B(a_n, b_n)}{B(a,b)} \int_\Theta \frac{1}{B(a_n, b_n)} \theta^{a_n - 1} (1-\theta)^{b_n - 1} \mathbb{1}_{(0,1)}(\theta) \, d\theta = 1$$

$$= \frac{B(a_n, b_n)}{B(a,b)} \qquad \begin{cases} a_n = a_0 + r \\ b_n = b_0 + n - r \end{cases}$$

2. Choose a_0 and b_0.

We look at the posterior distr: $\theta|\underline{X} \sim \text{Beta}(a_n, b_n)$

$$\mathbb{E}[\theta|\underline{X}] = \frac{a_n}{a_n+b_n} = \frac{a_0+r}{a_0+b_0+n} = \frac{a_0}{a_0+b_0+n} + \frac{r}{a_0+b_0+n}$$

$$= \frac{a_0+b_0}{a_0+b_0+n} \cdot \underbrace{\frac{a_0}{a_0+b_0}}_{\mathbb{E}[\theta]\ \text{(prior)}} + \frac{n}{a_0+b_0+n} \cdot \underbrace{\frac{r}{n}}_{\hat{\theta}_{MLE}\ \text{(empirical)}}$$

To synthetize:

	prior	empirical
sample size	a_0+b_0	n
# success	a_0	r

Suppose that we have some previous informations:
(a previous round of interviews)

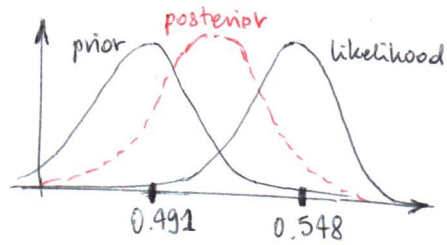
- Percentage of preference for Bush: 0.491
- standard deviation: 0.022

Let's use the informations of a previous round of interviews for the elicitation ↓

$$\Longrightarrow \begin{cases} \mathbb{E}[\theta] = \dfrac{a_0}{a_0+b_0} = 0.491 \\ \text{Var}(\theta) = \dfrac{a_0 b_0}{(a_0+b_0)^2(a_0+b_0+1)} = 0.22^2 \end{cases}$$

$$\Longrightarrow [..] \Longrightarrow \begin{cases} a_0 \simeq 253 = r^{old} \\ b_0 \simeq 262 = n^{old} \end{cases}$$

$$\Longrightarrow \theta|\underline{X} \sim \text{Beta}(253+279,\ 262+(504-279)) \sim \text{Beta}(532, 492)$$



We can compute summaries of $\pi(\theta|\underline{X})$:
For example: how much mass is on the part of the support $> \frac{1}{2}$?

$$\mathbb{P}(\theta > \tfrac{1}{2}|\underline{X}) = \int_{0.5}^{1} \pi(d\theta|\underline{X}) = 1 - \text{CDF}_{0.5}(\text{Beta}(532, 492)) \simeq 0.8947$$

We started from a prior distribution that was centered in a part of the support, but in force of the empirical evidence of the new sample we moved in another part.

3. Predict the outcome for $n+1$.

$$p(x_{n+1}|x_1,..,x_n) = \frac{p(x_1,..,x_n,x_{n+1})}{p(x_1,..,x_n)} = \frac{\text{Beta}(a_n+x_{n+1},\ b_n+1-x_{n+1})}{\text{Beta}(a_n, b_n)}$$

$$= \frac{\Gamma(a_n+x_{n+1})\,\Gamma(b_n+1-x_{n+1})}{\Gamma(a_n+b_n+1)} \cdot \frac{\Gamma(a_n+b_n)}{\Gamma(a_n)\,\Gamma(b_n)}$$

Suppose we want to predict if the next person is voting for Bush:

$$\mathbb{P}(x_{n+1}=1|\underline{X}) = \frac{\Gamma(a_n+1)\,\cancel{\Gamma(b_n)}}{\Gamma(a_n+b_n+1)} \cdot \frac{\Gamma(a_n+b_n)}{\Gamma(a_n)\,\cancel{\Gamma(b_n)}}$$

$$\mathbb{P}(X_{n+1}=1 \mid \underline{X}) = \frac{a_n \cancel{\Gamma(a_n)}\,\Gamma(a_n+b_n)}{(a_n+b_n)\,\Gamma(a_n+b_n)\,\cancel{\Gamma(a_n)}} = \frac{a_n}{a_n+b_n} = \mathbb{E}[\theta \mid \underline{X}] = 0.5159$$

What if we have to predict the outcome for more than 1 person?

$$p(x_{n+1},\dots,x_{n+m} \mid x_1,\dots,x_n) = \frac{p(x_1,\dots,x_n,x_{n+1},\dots,x_{n+m})}{p(x_1,\dots,x_n)}$$

$$= \frac{\text{Beta}(a_n+r_m,\ b_n+(m-r_m))}{\text{Beta}(a_n,b_n)} \qquad r_m = \sum_{i=n+1}^{m} x_i$$

$$= \frac{\Gamma(a_n+r_m)\,\Gamma(b_n+(m-r_m))}{\Gamma(a_n+b_n+m)} \cdot \frac{\Gamma(a_n+b_n)}{\Gamma(a_n)\,\Gamma(b_n)}$$

$$= \frac{(a_n+r_m-1)\cdots(a_n)\,\cancel{\Gamma(a_n)}\,(b_n+(m-r_m)-1)\cdots(b_n)\,\cancel{\Gamma(b_n)}}{(a_n+b_n+m-1)\cdots(a_n+b_n)\,\cancel{\Gamma(a_n+b_n)}} \cdot \frac{\cancel{\Gamma(a_n+b_n)}}{\cancel{\Gamma(a_n)}\,\cancel{\Gamma(b_n)}}$$

$$= \frac{(a_n)_{r_m}\,(b_n)_{m-r_m}}{(a_n+b_n)_m} = \text{predictive distribution of } r_m \text{ successes and } m \text{ further sampling conditioned on the first } n \text{ results}$$

$(a)_b = (a+b-1)(a+b-2)\cdots a = \frac{\Gamma(a+b)}{\Gamma(a)}$

Exercise 3. POISSON-GAMMA MODEL (counting data)

We suppose to have data: $X_1,\dots,X_n$, where $X_i \in \{0,1,2,\dots\}$

$$X_1,\dots,X_n \mid \lambda \overset{iid}{\sim} P(\lambda);$$

$$X_i \mid \lambda \sim P(\lambda) \overset{d}{=} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!} \qquad \lambda \in (0,\infty) \qquad : \begin{cases} \mathbb{E}[X_i] = \lambda \\ \text{Var}(X_i) = \lambda \end{cases}$$

Likelihood:

$$L(\lambda, \underline{X}) = \prod_{i=1}^{n} p(x_i,\lambda) = \frac{e^{-n\lambda}\lambda^{\sum_{i=1}^n x_i}}{\prod_{i=1}^n x_i!} \propto e^{-n\lambda}\lambda^{S_n}, \qquad S_n = \sum_{i=1}^n x_i$$

Prior for λ?

Looking at the likelihood we choose a gamma distribution.

$$\lambda \sim \text{Gamma}(a_0, b_0)$$

$$\pi(\lambda) \overset{d}{=} \frac{b_0^{a_0}}{\Gamma(a_0)}\lambda^{a_0-1}e^{-b_0\lambda}\mathbb{1}_{(0,\infty)}(\lambda) \qquad \left(\mathbb{E}[\lambda] = \frac{a_0}{b_0},\ \text{Var}(\lambda) = \frac{a_0}{b_0^2}\right)$$

In this way the posterior is:

$$\pi(\lambda \mid \underline{X}) \propto \pi(\lambda)\,L(\lambda,\underline{X}) = \lambda^{a_0-1}e^{-b_0\lambda}\lambda^{S_n}e^{-n\lambda}$$

$$= \lambda^{(a_0+S_n)-1}e^{-(b_0+n)\lambda} \sim \text{Gamma}(a_0+S_n, b_0+n) \sim \text{Gamma}(a_n, b_n)$$

1. Elicitation of the params of the prior?

$$\mathbb{E}[\lambda \mid \underline{X}] = \frac{a_n}{b_n} = \frac{a_0+S_n}{b_0+n} = \frac{b_0}{b_0+n}\frac{a_0}{b_0} + \frac{n}{b_0+n}\frac{S_n}{n}$$

$\mathbb{E}[\lambda]$ (prior) — $\hat{\lambda}_{MLE}$ (empirical)

b_0 is playing the role of the sample size of a prior (hypothetical) sample while a_0 plays the role of the sum of all the counts

2. Marginal distribution? (of the data)

$$m(x) = \frac{p(x|\lambda)\,\pi(\lambda)}{\pi(\lambda|\underline{x})} = \frac{\left(\frac{1}{x!}e^{-\lambda}\lambda^{x}\right)\left(\frac{b_0^{a_0}}{\Gamma(a_0)}\lambda^{a_0-1}e^{-b_0\lambda}\right)}{\frac{b_n^{a_n}}{\Gamma(a_n)}\lambda^{a_n-1}e^{-b_n\lambda}}$$

$$= \frac{1}{x!}\frac{b_0^{a_0}}{b_n^{a_n}}\frac{\Gamma(a_n)}{\Gamma(a_0)} = \binom{a_0+x-1}{x}\left(\frac{b_0}{b_0+1}\right)^{a_0}\left(\frac{1}{b_0+1}\right)^{x} \Rightarrow X \sim \text{Neg-Bin}(a,b)$$

marginal distribution for just one observation

3. Prediction: $n+1$?

$$p(x_{n+1}|\underline{x}) = \int_\Lambda p(x_{n+1}|\lambda)\,\pi(\lambda|\underline{x})\,d\lambda$$

$$= \int_\Lambda \frac{1}{x_{n+1}!}e^{-\lambda}\lambda^{x_{n+1}}\frac{b_n^{a_n}}{\Gamma(a_n)}\lambda^{a_n-1}e^{-b_n\lambda}\,d\lambda$$

$$= \frac{1}{x_{n+1}!}\frac{b_n^{a_n}}{\Gamma(a_n)}\frac{\Gamma(a_n+x_{n+1})}{(b_n+1)^{a_n+x_{n+1}}}\int_\Lambda [\,\cdot\,]\,d\lambda$$

$$= \binom{a_n+x_{n+1}-1}{x_{n+1}}\left(\frac{b_n}{b_n+1}\right)^{a_n}\left(\frac{1}{b_n+1}\right)^{\lambda_n+1}$$

$\Rightarrow$ The predictive distribution of the $n+1$ observation given $x_1,\dots,x_n$ is still a negative binomial distribution

$$\sim \text{Neg-Bin}(a_n, b_n)$$

$$X \sim \text{Neg-Bin}(a,b) \overset{d}{=} \binom{a+x-1}{x}\left(\frac{b}{b+1}\right)^{a}\left(\frac{1}{b+1}\right)^{x}$$

HYPOTHESIS TESTING

9/10/2020

H_0 vs. H_1 : $\begin{cases} H_0 : \theta \in \Theta_0 \\ H_1 : \theta \in \Theta_1 \end{cases}$ $\quad \Theta_0 \cap \Theta_1 = \emptyset$ $\quad \Theta_0 \cup \Theta_1 = \Theta$ = parameters space

In frequentist statistic we assume the null hypothesis to be true and we look at the empirical evidence and we state if we can reject or not H_0. The test is not exchangeable w.r.t. the hypothesis. In bayesian statistic the framework it different and we can switch the hypothesis: we're not assuming any of the hypothesis true, we just test the empirical evidence in favor of one hypothesis or the other. The test is equivalent if we switch the hypothesis.

We define a prior:

$$\pi(\theta) = \pi_0 \, g_0(\theta) \, \mathbb{1}_{\Theta_0} + (1-\pi_0) \, g_1(\theta) \, \mathbb{1}_{\Theta_1}$$

π_0, $(1-\pi_0)$: prior weights that we give to the sets Θ_0 and Θ_1

$g_0(\theta)$, $g_1(\theta)$ are the restrictions of the prior to Θ_0, Θ_1

What we can do is compute:

- Prior odds$_{01}$ $:= \dfrac{\pi_0}{1-\pi_0}$
- Posterior odds$_{01}$ $:= \dfrac{\mathbb{P}(\Theta_0 \mid \underline{X})}{\mathbb{P}(\Theta_1 \mid \underline{X})}$
- Bayesian Factor: $BF_{01} := \dfrac{\text{posterior odds}_{01}}{\text{prior odds}_{01}}$ ← This quantity measures how much the empirical evidence is shifting the mass in favor of the null hypothesis discarding the prior guess

BF_{01}	$2 \log BF_{01}$	evidence
<1	<0	in favor of H_1
1-3	0-2	weak evidence for H_0
3-12	2-5	in favor of H_0
12-150	5-10	strong in favor of H_0
>150	>10	very strong in favor of H_0

How to compute the Bayes factor:

$$BF_{01} = \frac{\mathbb{P}(\Theta_0 \mid \underline{X})}{\mathbb{P}(\Theta_1 \mid \underline{X})} \cdot \frac{\mathbb{P}(\Theta_1)}{\mathbb{P}(\Theta_0)} = \frac{\mathbb{P}(\Theta_0, \underline{X})}{\cancel{\mathbb{P}(\underline{X})}} \cdot \frac{\cancel{\mathbb{P}(\underline{X})}}{\mathbb{P}(\Theta_1, \underline{X})} \cdot \frac{\mathbb{P}(\Theta_1)}{\mathbb{P}(\Theta_0)}$$

$$= \frac{\mathbb{P}(\underline{X} \mid \Theta_0)}{\mathbb{P}(\underline{X} \mid \Theta_1)}$$

we can define the Bayes factor also in the case in which Θ_0 is not a set but a single point

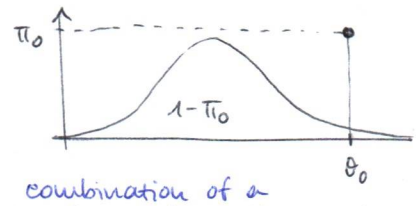
$H_0 : \theta = \theta_0$

$H_1 : \theta \neq \theta_0$

$$\begin{cases} H_0 : \theta = \theta_0 \\ H_1 : \theta \neq \theta_0 \end{cases}$$

We define a prior:

$$\pi_1(\theta) = \pi_0 \, \delta_{\theta_0} + (1-\pi_0)\, g(\theta)\, \mathbb{1}_{\Theta \setminus \{\theta_0\}} \longrightarrow$$



combination of a continuous distribution and a single mass point

Within this framework:

$$\mathbb{P}(\Theta_0) = \mathbb{P}(\theta = \theta_0) = \pi_0$$
$$\mathbb{P}(\Theta_1) = \mathbb{P}(\theta \neq \theta_0) = 1 - \pi_0$$

Marginal distribution under this prior:

$$m_1(\underline{x}) = m_1(x_1, \dots, x_n) = \int_\Theta f(\underline{x}|\theta)\, \pi_1(\theta)\, d\theta$$
$$= \int_\Theta f(\underline{x}|\theta)\, \pi_0\, \delta_{\theta_0}\, d\theta + \int_\Theta f(\underline{x}|\theta)\, \pi(\theta)\, \mathbb{1}_{\Theta \setminus \{\theta_0\}}\, d\theta$$

prior for θ (global) = starting prior for θ

$$= f(\underline{x}|\theta_0)\, \pi_0 + (1-\pi_0)\, m(\underline{x})$$

marginal of $\underline{x}$ under the general prior $\pi(\theta)$

$$\mathbb{P}(\theta = \theta_0 | \underline{x}) = \frac{f(\underline{x}|\theta_0)\, \pi_0}{f(\underline{x}|\theta_0)\pi_0 + (1-\pi_0)\, m(\underline{x})}$$

Now we can compute:

$$BF_{01} = \frac{\text{posterior odds}_{01}}{\text{prior odds}_{01}} = \frac{\mathbb{P}(\theta = \theta_0 | \underline{x})}{\mathbb{P}(\theta \neq \theta_0 | \underline{x})} \cdot \frac{1-\pi_0}{\pi_0}$$

$$= \frac{\frac{f(\underline{x}|\theta_0)\,\pi_0}{m_1(\underline{x})}}{1 - \frac{f(\underline{x}|\theta_0)\,\pi_0}{m_1(\underline{x})}} \cdot \frac{1-\pi_0}{\pi_0}$$

with the def. of $m_1(\underline{x})$

$$= \frac{\frac{f(\underline{x}|\theta_0)\,\pi_0}{m_1(\underline{x})}}{\frac{m_1(\underline{x}) - f(\underline{x}|\theta_0)\,\pi_0}{m_1(\underline{x})}} \cdot \frac{1-\pi_0}{\pi_0}$$

$$= \frac{(1-\pi_0)\, f(\underline{x}|\theta_0)}{f(\underline{x}|\theta_0)\pi_0 + (1-\pi_0)\, m(\underline{x}) - f(\underline{x}|\theta_0)\pi_0}$$

$$= \frac{f(\underline{x}|\theta_0)}{m(\underline{x})}$$

Doing Hypothesis Testing through Bayes Factor is equivalent to a model comparison:

Model 0: $\begin{cases} X_1, \dots, X_n | \theta \sim \pi_0(\theta) \\ \pi_0(\,) \overset{d}{=} \delta_{\theta_0} \end{cases}$

Model 1: $\begin{cases} X_1, \dots, X_n | \theta \sim \pi_1(\theta) \\ \pi_1(\theta) \overset{d}{=} \pi(\theta) \end{cases}$ ← starting prior that we use

Here we comparing a single mass point against a set, we can use it more generally: (with a restricted set)

$H_0 : \theta = \theta_0$

$H_1 : \theta < \theta_0$

$\pi^*(\theta) \propto \pi(\theta)\, \mathbb{1}_{(\theta < \theta_0)}$

Exercise 1.

Back to Beta-Bernoulli model: (for the elections (Bush))

$$\begin{cases} X_1, \dots, X_n \mid \theta \sim Be(\theta) \\ \pi(\theta) \sim Beta(a,b) \end{cases}$$

We found out:

$$\mathbb{P}(\theta > 0.5 \mid \underline{x}) = 0.8928$$

Suppose we want to test:

$$\begin{cases} H_0 : \theta \leq 0.5 \\ H_1 : \theta > 0.5 \end{cases}$$

We compute the Bayes factor:

$$BF_{01} = \frac{\mathbb{P}(\theta \leq 0.5 \mid \underline{x})}{\mathbb{P}(\theta > 0.5 \mid \underline{x})} \cdot \frac{\mathbb{P}(\theta > 0.5)}{\mathbb{P}(\theta \leq 0.5)} = \frac{1 - 0.8928}{0.8928} \cdot \frac{1 - 0.6543}{0.6543} = 0.0634$$

$$2 \log(BF_{01}) = -5.517 \implies \text{empirical evidence in favor of } H_1$$

Exercise 2.

We flip a coin $n = 1000$ times, obtaining 452 heads.
Question: is the coin balanced?

$$\begin{cases} H_0 : p = 0.5 \\ H_1 : p \neq 0.5 \end{cases} \qquad p = \mathbb{P}(\text{success})$$

We assume a binomial distribution.

X = # successes

$$X \sim Bin(n,p) \overset{d}{=} \binom{n}{x} p^x (1-p)^{n-x}$$

We start from:

$$p(x \mid p_0) = \binom{n}{x} p_0^x (1-p_0)^{n-x} \overset{H_0}{=} \binom{n}{x} p_0^n \qquad (\text{since } H_0 : p_0 = 1 - p_0 = \tfrac{1}{2})$$

We need the marginal for the data:
we first need to assume a prior for p:

$$\pi(p) \sim Beta(1,1) \sim \mathcal{U}([0,1])$$

with this assumption:

$$m(\underline{x}) = \binom{n}{x} \frac{Beta(1+x, 1+(n-x))}{Beta(1,1)}$$

Now we can compute:

$$BF_{01} = \frac{p(x \mid \theta_0)}{m(\underline{x})} = \frac{\binom{n}{x} p_0^n \, Beta(1,1)}{\binom{n}{x} Beta(1+x, 1+n-x)}$$

$$= p_0^n \frac{\binom{n}{x} (n+1)!}{n!} = 0.2512$$

$$2 \log(BF_{01}) = -2.763 < 0 \implies \text{evidence for } H_1$$

$$\implies \text{The coin is } \underline{\text{not}} \text{ balanced}$$

Exercise 3.

$$f(x|\theta) = 2\theta x e^{-\theta x^2} \mathbb{1}_{(0,+\infty)}(x)$$

$$X_1, \dots, X_n | \theta \overset{iid}{\sim} f(x,\theta)$$

1. Find the family of prior conjugate to $f(x|\theta)$ and compute the posterior.

We start from the likelihood:

$$L(\theta, \underline{x}) = \prod_{i=1}^n f(x_i,\theta) = 2^n \theta^n \left(\prod_{i=1}^n x_i\right) e^{-\theta \sum_{i=1}^n x_i^2}$$

$:= S_{n,2}$

we're interested only in θ: $\propto \theta^n e^{-\theta S_{n,2}}$

it reminds a kernel of a gamma distribution

We try with: $\theta \sim \text{Gamma}(a,b)$

$$\pi(\theta|\underline{x}) \propto \pi(\theta) L(\theta, \underline{x}) = \left[\theta^{a-1} e^{-b\theta}\right]\left[\theta^n e^{-\theta S_{n,2}}\right]$$

$$= \theta^{a+n-1} e^{-(b+S_{n,2})\theta} \sim \text{Gamma}(a+n, b+S_{n,2})$$

2. Write the Bayes factor to test: $H_0: \theta = 1$, $H_1: \theta \neq 1$

It's a pretual (single valued) test.

Marginal of the data:

$$m(\underline{x}) = \int_\Theta L(\theta, \underline{x}) \pi(\theta) d\theta$$

$$= \int_\Theta 2^n \theta^n \left(\prod_{i=1}^n x_i\right) e^{-\theta S_{n,2}} \frac{b^a}{\Gamma(a)} \theta^{a-1} e^{-b\theta} d\theta$$

$$= 2^n \left(\prod_{i=1}^n x_i\right) \frac{b^a}{\Gamma(a)} \frac{\Gamma(a+n)}{(b+S_{n,2})^{a+n}} \int_\Theta \dots d\theta$$

we can now compute:

$$BF_{01} = \frac{f(\underline{x}|\theta_0)}{m(\underline{x})} = \frac{2^n \theta_0^n \left(\prod_{i=1}^n x_i\right) e^{-\theta_0 S_{n,2}}}{2^n \left(\prod_{i=1}^n x_i\right) \frac{b^a}{\Gamma(a)} \frac{\Gamma(a+n)}{(b+S_{n,2})^{a+n}}}$$

$$= \frac{\theta_0^n e^{-\theta_0 S_{n,2}} \Gamma(a) (b+S_{n,2})^{a+n}}{b^a \Gamma(a+n)}$$

$\theta_0 = 1$

$$= \frac{e^{-S_{n,2}} \Gamma(a) (b+S_{n,2})^{a+n}}{b^a \Gamma(a+n)}$$

3. Suppose that we have a sample of $n = 4$, with: $S_{4,2} = 5.71$

a. Choose a, b s.t. $\mathbb{E}[\theta] = 1$, $\text{var}(\theta) = 10$

b. Test $H_0: \theta_0 = 1$, $H_1: \theta_0 \neq 1$

a. $\begin{cases} \mathbb{E}[\theta] = \frac{a}{b} = 1 \\ \text{Var}(\theta) = \frac{a}{b^2} = 10 \end{cases} \longrightarrow \begin{cases} a = 0.1 \\ b = 0.1 \end{cases} \Rightarrow \pi(\theta) \overset{d}{=} \text{Gamma}(0.1, 0.1)$

b. $$BF_{01} = \frac{e^{-5.71} \Gamma(0.1)}{0.1^{0.1} \Gamma(0.1+4)} (0.1+5.71)^{0.1+4}$$

$$= 7.91 \longrightarrow \text{in favor of } H_0 \text{ (since } > 1)$$

9/10/2020

3/3

4. Compute: $\mathbb{P}(X_5 > 1 \mid \underline{x})$.

First we derive the CDF of X:

$$F(x) = \int_{-\infty}^{x} 2\vartheta t\, e^{-\vartheta t^2} \mathbb{1}_{(0,+\infty)}(t)\, dt$$

$$= [\dots] = 1 - e^{-\vartheta x^2}$$

$$\mathbb{P}(X_5 > 1 \mid \underline{x}) = \int_{\Theta} \mathbb{P}(X_5 > 1 \mid \vartheta)\, \pi(\vartheta \mid \underline{x})\, d\vartheta$$

$$= \int_0^{\infty} (1 - F(1))\, \pi(\vartheta \mid \underline{x})\, d\vartheta$$

$$= \int_0^{\infty} e^{-\vartheta} \frac{(b + S_{n,2})^{a+n}}{\Gamma(a+n)} \vartheta^{a+n-1} e^{-(b+S_{n,2})\vartheta}\, d\vartheta$$

$$= \frac{(b+S_{n,2})^{a+n}}{(b+S_{n,2}+1)^{a+n}} \int_{\Theta} \dots\, d\vartheta$$

$$= \left(\frac{b+S_{n,2}}{b+S_{n,2}+1}\right)^{a+n} = 0.5215$$

JEFFREY'S PRIOR – Binomial case

= prior proportional to the square root of the Fisher information matrix

We suppose to have:

$$X \sim \text{Bin}(n,p) \quad : \quad p(x|p) = \binom{n}{x} p^x (1-p)^{n-x}$$

Fisher information matrix:

$$\mathbf{I}(\underline{\vartheta}) = \mathbb{E}\left[\left(\frac{\partial}{\partial \underline{\vartheta}} \log f(x, \underline{\vartheta})\right)^2 \middle| \underline{\vartheta}\right] = -\mathbb{E}\left[\frac{\partial^2}{\partial \underline{\vartheta}^2} \log f(x, \underline{\vartheta}) \middle| \underline{\vartheta}\right]$$

So:

$$\log p(x|p) \propto x \log(p) + (n-x)\log(1-p)$$

$$\frac{d}{dp} \log p(x|p) = \frac{x}{p} - \frac{n-x}{1-p}$$

$$\frac{d^2}{dp^2} \log p(x|p) = -\frac{x}{p^2} - \frac{n-x}{(1-p)^2}$$

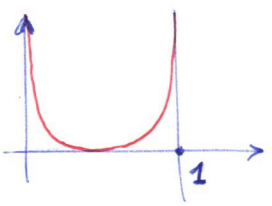
$$I(p) = -\mathbb{E}\left[-\frac{x}{p^2} - \frac{n-x}{(1-p)^2}\right] = \frac{np}{p^2} + \frac{n-np}{(1-p)^2}$$

$$= \frac{n - np + np}{p(1-p)} = \frac{n}{p(1-p)}$$

$\Longrightarrow$ Jeffrey's prior:

$$\pi_J(p) \propto \sqrt{I(p)} \propto \sqrt{\frac{1}{p(1-p)}} = p^{-1/2}(1-p)^{-1/2}$$

$$\text{Beta}\left(\tfrac{1}{2}, \tfrac{1}{2}\right)$$



JEFFREY'S PRIOR

family of objective priors
= non informative priors

$$\text{prior} \propto \sqrt{|I(\theta)|}, \quad I(\theta) = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\log f(x,\theta)\right)^2 \middle| \theta\right] = \mathbb{E}\left[\left(\frac{f'(x,\theta)}{f(x,\theta)}\right)^2 \middle| \theta\right]$$
$$= -\mathbb{E}\left[\frac{\partial^2}{\partial\theta^2}\log f(x,\theta) \middle| \theta\right]$$

Moreover: $I_n(\theta) = n\, I(\theta)$

Univariate Normal Distribution

(with prior on both parameters)

$$X_1, \dots, X_n \mid \mu, \sigma^2 \sim N(\mu, \sigma^2)$$

$$f(x, \mu, \sigma^2) \propto (\sigma^2)^{-1/2} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \qquad \log f(x,\mu,\sigma^2) \propto -\frac{1}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}(x-\mu)^2$$

$$\frac{d}{d\mu}\log f(x,\mu,\sigma^2) = \frac{(x-\mu)}{2\sigma^2}$$

$$\frac{d^2}{d\mu^2}\log f(x,\mu,\sigma^2) = -\frac{1}{2\sigma^2}$$

$$\frac{d}{d\sigma^2}\log f(x,\mu,\sigma^2) = -\frac{1}{2\sigma^2} + \frac{(x-\mu)^2}{2(\sigma^2)^2}$$

$$\frac{d^2}{d(\sigma^2)^2}\log f(x,\mu,\sigma^2) = \frac{1}{2(\sigma^2)^2} - \frac{(x-\mu)^2}{(\sigma^2)^3}$$

$$\frac{d^2}{d\mu\, d\sigma^2}\log f(x,\mu,\sigma^2) = -\frac{(x-\mu)}{2(\sigma^2)^2}$$

$$I(\theta) = \mathbb{E}_X\begin{bmatrix} -\frac{\partial^2}{\partial\mu^2} & -\frac{\partial^2}{\partial\mu\partial\sigma^2} \\ -\frac{\partial^2}{\partial\mu\partial\sigma^2} & -\frac{\partial^2}{\partial(\sigma^2)^2} \end{bmatrix} = \mathbb{E}_X\begin{bmatrix} \frac{1}{2\sigma^2} & \frac{X-\mu}{2(\sigma^2)^2} \\ \frac{X-\mu}{2(\sigma^2)^2} & \frac{1}{2(\sigma^2)^2} - \frac{(X-\mu)^2}{(\sigma^2)^3} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2\sigma^2} & 0 \\ 0 & \frac{1}{2(\sigma^2)^2} - \frac{\sigma^2}{(\sigma^2)^3} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sigma^2} & 0 \\ 0 & \frac{1}{2(\sigma^2)^2} \end{bmatrix}$$

$$|I(\theta)| = \det I(\theta) = \frac{1}{\sigma^2}\cdot\frac{1}{2(\sigma^2)^2} = \frac{1}{2(\sigma^2)^3}$$

$$\Longrightarrow \sqrt{|I(\theta)|} \propto (\sigma^2)^{-3/2}$$

$$\Longrightarrow \underbrace{\pi_J(\theta)}_{\text{Jeffrey's prior}} \propto (\sigma^2)^{-3/2}$$

← proportional on something depending on σ^2
(⇒ for the location parameter it proportional to a constant (when the only parameter we consider is the location parameter))
Moreover, this is an improper prior (it not integrating up to 1): it not normalized and cannot be normalized)

Quick remark for the

location family case

$X_1, \dots, X_n | \mu \overset{iid}{\sim} f(x, \mu)$

$f(x, \mu) \in \mathcal{F}_\mu = \{ g(x-\mu) : \mu \in \mathbb{R},\ g(\cdot) \text{ density} \}$ ← the act of the location parameter is in term of translation (we just translate x by μ)

We have that:

$$\frac{d}{d\mu} g(x-\mu) = -g'(x-\mu)$$

Moreover:

$$\frac{f'(x,\mu)}{f(x,\mu)} = -\frac{g'(x-\mu)}{g(x-\mu)}$$

$$\Rightarrow I(\theta) = I(\mu) = \mathbb{E}\left[\left(\frac{f'(x,\mu)}{f(x,\mu)} \right)^2 \middle| \mu \right]$$

$$= \int \frac{(g'(x-\mu))^2}{(g(x-\mu))^2} g(x-\mu)\, dx$$

because we're taking the expectation w.r.t. x

$$= \int \frac{(g'(z))^2}{g(z)}\, dz$$

$\propto$ constant $:= k$

since it does not depend on μ

$$\Rightarrow \pi_J(\mu) \propto \text{constant}$$

$\pi_J(\mu)$: Jeffrey's prior

15/10/2020
2/4

Multivariate Normal Distribution

Normal likelihood with Normal-Inverse-Wishart prior

Riccardo Corradin
October 14, 2020

N - NIW

$\boldsymbol{X}$ is a random vector, absolutely continuous wrt the Lebesgue measure on $\mathbb{R}^p$, distributed as a multivariate Normal distribution, $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \Sigma)$, with $\boldsymbol{X} \in \mathbb{R}^p$, $\boldsymbol{\mu} \in \mathbb{R}^p$ and $\Sigma \in \mathbb{M}_{p\times p}$, with $\mathbb{M}_{p\times p}$ denoting the space of positive definite symmetric matrices $p \times p$. The density function of $\boldsymbol{X}$ is

$$f(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) = (2\pi)^{-d/2}|\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\mu)^\intercal \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\}$$

with $\mathbb{E}[\boldsymbol{X}] = \boldsymbol{\mu}$ and $Cov(X_i, X_j) = \Sigma_{[ij]}$. We assume as prior distributions $\pi(\boldsymbol{\mu}, \Sigma)$ on $(\boldsymbol{\mu}, \Sigma)$ a Normal-Inverse-Wishart distribution, $(\boldsymbol{\mu}, \Sigma) \sim NIW(\boldsymbol{m}_0, \kappa_0, \nu_0, \Lambda_0)$, which corresponds to assume that

$$\begin{aligned} \boldsymbol{\mu} \mid \Sigma &\sim N(\boldsymbol{m}_0, \Sigma/\kappa_0), \qquad \boldsymbol{m}_0 \in \mathbb{R}^p, \kappa_0 \in \mathbb{R}^+ \\ \Sigma &\sim IW(\nu_0, \Lambda_0), \qquad \nu_0 \in \mathbb{R}^+ . \nu_0 > p-1, \Lambda_0 \in \mathbb{M}_{p\times p} \end{aligned}$$

N - NIW

Recall if $\Sigma \sim IW(\nu_0, \Lambda_0)$, with $\Sigma, \Lambda_0 \in \mathbb{M}_{p\times p}$, $\nu_0 \in \mathbb{R}^+$ and $\nu_0 > p-1$, its density is equal to

$$f(\Sigma; \nu_0, \Lambda_0) = \frac{|\Lambda_0|^{\nu_0/2}}{2^{\nu_0 p/2}\Gamma_p(\nu_0/2)}|\Sigma|^{-(\nu_0+p+1)/2} \exp\left\{-\frac{1}{2}\text{tr}(\Lambda_0\Sigma^{-1})\right\}$$

where $\Gamma_p(\cdot)$ denotes the multivariate Gamma function of order p with

$$\Gamma_p(\nu_0/2) = \pi^{p(p-1)/4}\prod_{j=1}^{p}\Gamma\left(\frac{\nu_0}{2} + \frac{1-j}{2}\right).$$

We further recall that $\mathbb{E}[\Sigma] = \Lambda_0/(\nu_0 - p - 1)$ and

$$Cor(\Sigma_{[ij]}, \Sigma_{[kl]}) = \frac{2\Lambda_{0[ij]}\Lambda_{0[kl]} + (\nu_0 - p - 1)(\Lambda_{0[ik]}\Lambda_{0[jl]} + \Lambda_{0[il]}\Lambda_{0[jk]})}{(\nu_0 - p)(\nu_0 - p - 1)^2(\nu_0 - p - 3)}$$

N - NIW

(Recall:)
Multivariate location-scale t-Student distribution

We further recall that $\boldsymbol{Y} \sim t_\nu(\boldsymbol{\mu}, \Sigma)$ denotes a multivariate location-scale t-Student distribution, with $\alpha > 0$ degree of freedom, $\boldsymbol{\mu} \in \mathbb{R}^p$ location parameter and $\Sigma \in \mathbb{M}_{p\times p}$ scale parameter, and its density is equal to

$$f(\boldsymbol{y}; \alpha, \boldsymbol{\mu}, \Sigma) = \frac{\Gamma_p\left(\frac{\alpha+p}{2}\right)}{\Gamma\left(\frac{\alpha}{2}\right)\alpha^{p/2}\pi^{p/2}|\Sigma|^{1/2}}\left[1 + \frac{1}{\alpha}(\boldsymbol{y}-\boldsymbol{\mu})^\intercal\Sigma^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right]^{-(\alpha+p)/2}$$

Details (Normal-Inverse-Wishart)

N - NIW

The full model specification is then

$$\begin{aligned} \boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \mid \boldsymbol{\mu}, \Sigma &\sim N(\boldsymbol{\mu}, \Sigma) \\ \boldsymbol{\mu} \mid \Sigma &\sim N(\boldsymbol{m}_0, \Sigma/\kappa_0) \\ \Sigma &\sim IW(\nu_0, \Lambda_0) \end{aligned}$$

where a priori we have

$$\mathbb{E}_\pi[\boldsymbol{\mu}] = \mathbb{E}_\Sigma[\mathbb{E}_\mu[\boldsymbol{\mu} \mid \Sigma]] = \mathbb{E}_\Sigma[\boldsymbol{m}_0] = \boldsymbol{m}_0$$

$$\begin{aligned} Var_\pi(\boldsymbol{\mu}) &= Var_\Sigma(\mathbb{E}_\mu[\boldsymbol{\mu} \mid \Sigma]) + \mathbb{E}_\Sigma[\mathrm{Var}_\mu(\boldsymbol{\mu} \mid \Sigma)] \\ &= Var_\Sigma(\boldsymbol{m}_0) + \mathbb{E}_\Sigma[\Lambda_0/\kappa_0] = \frac{1}{\kappa_0}\left(\frac{\Lambda_0}{\nu_0 - p - 1}\right) \end{aligned}$$

and

$$\mathbb{E}_\pi[\Sigma] = \Lambda_0/(\nu_0 - p - 1); \qquad Var_\pi(\Sigma) = \textit{..see above}$$

N - NIW

The likelihood is equal to

$$\begin{aligned} L(\boldsymbol{\mu}, \Sigma; \boldsymbol{X}_{1:n}) &= \prod_{i=1}^n f_X(\boldsymbol{X}_i; \boldsymbol{\mu}, \Sigma) \\ &= \prod_{i=1}^n (2\pi)^{-p/2} |\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu})^\intercal \Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})\right\} \\ &= (2\pi)^{-np/2} |\Sigma|^{-n/2} \exp\left\{-\frac{1}{2}\sum_{i=1}^n (\boldsymbol{x}_i - \boldsymbol{\mu})^\intercal \Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})\right\} \end{aligned}$$

we further set $\boldsymbol{S}_n = \sum_{i=1}^n \boldsymbol{x}_i$.

N - NIW

The posterior distribution of $(\boldsymbol{\mu}, \Sigma)$ given a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ is equal to

$$\begin{aligned} \pi(\boldsymbol{\mu}, \Sigma \mid \boldsymbol{X}_1, \ldots, \boldsymbol{X}_n) &\propto L(\boldsymbol{\mu}, \Sigma; \boldsymbol{X}_{1:n}) \pi(\boldsymbol{\mu}, \Sigma) \\ &\propto \underbrace{|\Sigma|^{-1/2} |\Sigma|^{-(\nu_0 + n + p + 1)/2}}_{A} \exp\left\{-\frac{1}{2}\sum_{i=1}^n (\boldsymbol{X}_i - \boldsymbol{\mu})^\intercal \Sigma^{-1}(\boldsymbol{X}_i - \boldsymbol{\mu})\right. \\ &\quad \left. -\frac{1}{2}(\boldsymbol{\mu} - \boldsymbol{m}_0)^\intercal \left(\frac{\Sigma}{\kappa_0}\right)^{-1} (\boldsymbol{\mu} - \boldsymbol{m}_0) - \frac{1}{2}\mathrm{tr}(\Lambda_0 \Sigma^{-1})\right\} \\ &= A \exp\left\{-\frac{1}{2}\left[\sum_{i=1}^n \boldsymbol{X}_i^\intercal \Sigma^{-1} \boldsymbol{X}_i - 2\boldsymbol{\mu}^\intercal \Sigma^{-1} \boldsymbol{S}_n + n\boldsymbol{\mu}^\intercal \Sigma^{-1} \boldsymbol{\mu}\right.\right. \\ &\quad \left.\left. + \kappa_0 \boldsymbol{\mu}^\intercal \Sigma^{-1} \boldsymbol{\mu} - 2\kappa_0 \boldsymbol{\mu}^\intercal \Sigma^{-1} \boldsymbol{m}_0 + \kappa_0 \boldsymbol{m}_0^\intercal \Sigma^{-1} \boldsymbol{m}_0 + \mathrm{tr}(\Lambda_0 \Sigma^{-1})\right]\right\} \end{aligned}$$

N - NIW

We set $\kappa_n = \kappa_0 + n$ and $\boldsymbol{m}_n = (\kappa_0 \boldsymbol{m}_0 + \boldsymbol{S}_n)/\kappa_n$

$$\begin{aligned} &= A \exp\left\{-\frac{1}{2}\left[-2\boldsymbol{\mu}^\intercal \left(\frac{\Sigma}{\kappa_n}\right)^{-1} \boldsymbol{m}_n + \boldsymbol{\mu}^\intercal \left(\frac{\Sigma}{\kappa_n}\right)^{-1} \boldsymbol{\mu} \pm \boldsymbol{m}_n^\intercal \left(\frac{\Sigma}{\kappa_n}\right)^{-1} \boldsymbol{m}_n\right.\right. \\ &\quad \left.\left. + \sum_{i=1}^n \boldsymbol{X}_i^\intercal \Sigma^{-1} \boldsymbol{X}_i + \kappa_0 \boldsymbol{m}_0^\intercal \Sigma^{-1} \boldsymbol{m}_0 + \mathrm{tr}(\Lambda_0 \Sigma^{-1})\right]\right\} \\ &= A \exp\left\{-\frac{1}{2}\left[(\boldsymbol{\mu} - \boldsymbol{m}_n)^\intercal \left(\frac{\Sigma}{\kappa_n}\right)^{-1} (\boldsymbol{\mu} - \boldsymbol{m}_n)\right.\right. \\ &\quad \left.\left. + \mathrm{tr}\left(\left(\sum_{i=1}^n \boldsymbol{X}_i \boldsymbol{X}_i^\intercal + \kappa_0 \boldsymbol{m}_0 \boldsymbol{m}_0^\intercal - \kappa_n \boldsymbol{m}_n \boldsymbol{m}_n^\intercal + \Lambda_0\right) \Sigma^{-1}\right)\right]\right\} \end{aligned}$$

N - NIW

$$
\begin{aligned}
&= |\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}\left[(\mu - \boldsymbol{m}_n)^\intercal \left(\frac{\Sigma}{\kappa_n}\right)^{-1}(\mu - \boldsymbol{m}_n)\right]\right\} \\
&\times |\Sigma|^{-(\nu_n+p+1)/2} \exp\left\{-\frac{1}{2}\left[\mathrm{tr}\left(\left(\Lambda_0 + \sum_{i=1}^{n}(\boldsymbol{X}_i - \overline{\boldsymbol{X}})(\boldsymbol{X}_i - \overline{\boldsymbol{X}})^\intercal \right.\right.\right. \\
&\quad \left.\left.\left. + \frac{\kappa_0 n}{\kappa_n}(\overline{\boldsymbol{X}} - \boldsymbol{m}_0)(\overline{\boldsymbol{X}} - \boldsymbol{m}_0)^\intercal\right)\Sigma^{-1}\right)\right]\right\}
\end{aligned}
$$

which is the kernel of a NIW distribution with updated parameters

$$
\kappa_n = \kappa_0 + n, \qquad \boldsymbol{m}_n = (\kappa_0 \boldsymbol{m}_0 + \boldsymbol{S}_n)/\kappa_n, \qquad \nu_n = \nu_0 + n,
$$

$$
\Lambda_n = \Lambda_0 + \sum_{i=1}^{n}(\boldsymbol{X}_i - \overline{\boldsymbol{X}})(\boldsymbol{X}_i - \overline{\boldsymbol{X}})^\intercal + \frac{\kappa_0 n}{\kappa_n}(\overline{\boldsymbol{X}} - \boldsymbol{m}_0)(\overline{\boldsymbol{X}} - \boldsymbol{m}_0)^\intercal
$$

N - NIW

The marginal distribution of μ is equal to

$$
\begin{aligned}
\pi(\mu) &= \int_{\mathbb{M}_{p\times p}} \pi(\mu \mid \Sigma)\pi(d\Sigma) \\
&= \int_{\mathbb{M}_{p\times p}} (2\pi)^{-p/2}\left|\frac{\Sigma}{\kappa_0}\right|^{-1/2} \exp\left\{-\frac{1}{2}(\mu - \boldsymbol{m}_0)^\intercal \left(\frac{\Sigma}{\kappa_0}\right)^{-1}(\mu - \boldsymbol{m}_0)\right\} \\
&\times \frac{|\Lambda_0|^{\nu_0/2}}{2^{(\nu_0 p)/2}\Gamma_p(\frac{\nu_0}{2})}|\Sigma|^{-(\nu_0+p+1)/2} \exp\left\{-\frac{1}{2}\mathrm{tr}(\Lambda_0\Sigma^{-1})\right\} d\Sigma \\
&\propto |\Lambda_0|^{\nu_0/2} \int_{\mathbb{M}_{p\times p}} |\Sigma|^{-(\nu_0+p+1)/2} \exp\{ \\
&\quad -\frac{1}{2}\mathrm{tr}\left[(\kappa_0(\mu - \boldsymbol{m}_0)(\mu - \boldsymbol{m}_0)^\intercal + \Lambda_0)\Sigma^{-1}\right]\Big\} d\Sigma
\end{aligned}
$$

N - NIW

$$
\begin{aligned}
&\propto \frac{|\Lambda_0|^{\nu_0/2} 2^{(\nu_0+1)p/2}\Gamma_p(\frac{\nu+1}{2})}{|\Lambda_0 + \kappa_0(\mu - \boldsymbol{m}_0)(\mu - \boldsymbol{m}_0)^\intercal|^{(\nu_0+1)/2}} \\
&\propto \left|\Lambda_0 + (\mu - \boldsymbol{m}_0)^\intercal \left(\frac{\Lambda_0}{\kappa_0}\right)^{-1}(\mu - \boldsymbol{m}_0)\Lambda_0\right|^{-(\nu_0+1)/2} \\
&\propto \left|1 + \frac{1}{(\nu_0 - p + 1)}(\mu - \boldsymbol{m}_0)^\intercal \left(\frac{\Lambda_0}{(\nu_0 - p + 1)\kappa_0}\right)^{-1}(\mu - \boldsymbol{m}_0)\right|^{-(\nu_0+1)/2}
\end{aligned}
$$

which is the kernel of a t-Student distribution, $\mu \sim t_{\nu_0 - p+1}\left(\boldsymbol{m}_0, \frac{\Lambda_0}{(\nu_0 - p+1)\kappa_0}\right)$.

N - NIW

Given $\boldsymbol{X}_{1:n} = \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\}$ a set of n observations, the predictive distribution of the $n+1$ observation is equal to

$$
\begin{aligned}
f(\boldsymbol{X}_{n+1} \mid \boldsymbol{X}_{1:n}) &= \int_{\mathbb{R}^p \times \mathbb{M}_{p\times p}} f(\boldsymbol{X}_{n+1} \mid \mu, \Sigma, \boldsymbol{X}_{1:n})\pi(d\mu, d\Sigma) \\
&= \int_{\mathbb{R}^p \times \mathbb{M}_{p\times p}} f(\boldsymbol{X}_{n+1} \mid \mu, \Sigma)\pi(d\mu, d\Sigma \mid \boldsymbol{X}_{1:n}) \\
&\propto \int_{\mathbb{R}^p \times \mathbb{M}_{p\times p}} |\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_{n+1} - \mu)^\intercal \Sigma^{-1}(\boldsymbol{x}_{n+1} - \mu)\right\} \\
&\times \left|\frac{\Sigma}{\kappa_n}\right|^{-1/2} \exp\left\{-\frac{1}{2}(\mu - \boldsymbol{m}_n)^\intercal \left(\frac{\Sigma}{\kappa_n}\right)^{-1}(\mu - \boldsymbol{m}_n)\right\} \\
&\times |\Sigma|^{-(\nu_n+p+1)/2} \exp\left\{-\frac{1}{2}\mathrm{tr}(\Lambda_n\Sigma^{-1})\right\} d\Sigma d\mu
\end{aligned}
$$

$$\propto C \int_{\mathbb{R}^p \times \mathbb{M}_{p\times p}} C^{-1} |\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\mu - \boldsymbol{m}_{n+1})^\intercal \left(\frac{\Sigma}{\kappa_{n+1}}\right)^{-1} (\mu - \boldsymbol{m}_{n+1})\right\}$$
$$\times |\Sigma|^{-(\nu_{n+1}+p+1)/2} \exp\left\{-\frac{1}{2}\mathrm{tr}(\Lambda_{n+1}\Sigma^{-1})\right\} d\Sigma d\mu$$

which is the integral of the kernel of a NIW, up to a normalization constant C, the part of C depending on $\boldsymbol{X}_{n+1}$ is

$$C \propto \left|\Lambda_n + \frac{\kappa_n}{\kappa_n + 1}(\boldsymbol{x}_{n+1} - \boldsymbol{\mu}_n)(\boldsymbol{x}_{n+1} - \boldsymbol{\mu}_n)^\intercal\right|^{-(\nu_n+1)/2}$$
$$\propto \left|1 + \frac{1}{\nu_n - p + 1}(\boldsymbol{x}_{n+1} - \boldsymbol{\mu}_n)^\intercal \left(\frac{(\kappa_n + 1)\Lambda_n}{\kappa_n(\nu_n - p + 1)}\right)^{-1} (\boldsymbol{x}_{n+1} - \boldsymbol{\mu}_n)\right|^{-(\nu_n+1)/2}$$

that describes the kernel of a multivariate t-Student distribution, $\boldsymbol{X}_{n+1} \mid \boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \sim t_{\nu_n - p+1}\left(\boldsymbol{m}_n, \left(\frac{(\kappa_n+1)\Lambda_n}{\kappa_n(\nu_n - p+1)}\right)\right)$.

Recap:

- Normal-Inverse-Wishart prior
- Normal likelihood
- Normal-Inverse-Wishart posterior
- The marginal on μ is a t-Student (multivariate)
- The predictive is a t-student (multivariate)

Exercise 1. 15/10/2020 4/4

3 components working in parallel ($\Rightarrow \perp\!\!\!\perp$).
They're of the same kind. We denote by Y_1, Y_2, Y_3 the failure time of each component (expressed in hours) and we assume: $Y_i \sim \text{Beta}(a, 1)$, $a>0$ $\forall i$.
We collected a sample of $n = 120$ and we obtained $\sum_{i=1}^{n} \log(x_i) = -4a$,
where X_i = lifetime of the i-th system.

1. Derive the density of X (= lifetime of the entire system).

$$F_X(x) = \mathbb{P}(X \le x) = \mathbb{P}(\max\{Y_1, Y_2, Y_3\} \le x) = \mathbb{P}(Y_1 \le x, Y_2 \le x, Y_3 \le x)$$
$$\overset{\perp\!\!\!\perp}{=} \mathbb{P}(Y_1 \le x)\,\mathbb{P}(Y_2 \le x)\,\mathbb{P}(Y_3 \le x) = (\mathbb{P}(Y \le x))^3$$

$$Y_i \sim \text{Beta}(a,1) \Longrightarrow f(y,a) = a y^{a-1} \mathbb{1}_{(0,1)}(y)$$
$$\Longrightarrow F_Y(y) = \int_0^y a t^{a-1}\,dt = y^a$$

$$\Longrightarrow F_X(x) = (\mathbb{P}(Y \le x))^3 = (x^a)^3 = x^{3a}$$
$$\Longrightarrow f(x,a) = \frac{d}{dx} F_X(x) = 3a x^{3a-1} \Longrightarrow X|a \sim \text{Beta}(3a, 1)$$

2. Find the conjugate distribution and the update rule of the hyperparameters.

$$L(a, \underline{x}) = \prod_{i=1}^{n} f(x_i, a) = \prod_{i=1}^{n} (3a x_i^{3a-1}) = 3^n a^n \left(\prod_{i=1}^{n} x_i^{-1}\right) e^{3a \sum_{i=1}^{n} \log(x_i)}$$
$$\propto a^n e^{3a \sum_{i=1}^{n} \log(x_i)} = a^n e^{3a S_{\ell x}}$$

$$\Longrightarrow \pi(a) \overset{d}{=} \text{Gamma}(\alpha, \beta) \overset{d}{=} \frac{\beta^\alpha}{\Gamma(\alpha)} a^{\alpha-1} e^{-\beta a}$$

$$\Longrightarrow \pi(a|\underline{x}) \propto L(a, \underline{x})\,\pi(a) \propto a^{\alpha+n-1} e^{-(\beta - 3S_{\ell x})a} \sim \text{Gamma}(\alpha+n, \beta - 3S_{\ell x})$$

3. Suppose that we have a previous study with $m = 10$ observations:
$z_1, \dots, z_{10}$ s.t. $\sum_{i=1}^{m} z_i = -1.73$.
Use the equivalence sample principle to specify α and β.
We start from the posterior expectation:

$$\mathbb{E}[a|\underline{x}] = \left(\frac{\beta - 3S_{\ell x}}{\alpha + n}\right)^{-1} = \left(\frac{\alpha}{\alpha+n}\frac{\beta}{\alpha} + \frac{n}{\alpha+n}\left(-\frac{3S_{\ell x}}{n}\right)\right)^{-1}$$

So n is playing the role of α,
$-3S_{\ell x}$ is playing the role of β.
A way to elicitate the hyperparameters is:

$$\alpha = m = 10$$
$$\beta = -3S_{\ell x} = 5.19$$

The parameters of the posterior distribution are:

$$\alpha_n = \alpha + n = m + n = 10 + 120 = 130$$
$$\beta_n = \beta - 3S_{\ell x} = 5.19 - \dots = 154.95$$

} parameters of the posterior distr. assuming as prior elicitation the specification according to the equivalence sample principle with the prior sample $z_1, \dots, z_m$

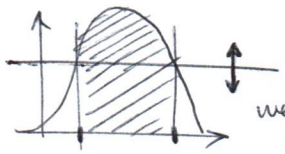
4. Calculate the predictive probability that a system works at most 30 minutes.

$$\mathbb{P}(X_{n+1} \le 0.5 | \underline{X}) = \int_0^\infty \mathbb{P}(X_{n+1} \le 0.5 | a)\,\pi(a|\underline{x})\,da$$
$$\overset{\perp\!\!\!\perp}{=} \int_0^\infty \underbrace{(0.5)^{3a}}_{F_X(0.5)} a^{\alpha+n-1} e^{-(\beta - 3S_{\ell x})a} \frac{\beta_n^{\alpha_n}}{\Gamma(\alpha_n)}\,da$$

$$\mathbb{P}(X_{n+1} \le 0.5 \mid \underline{X}) = \frac{\beta_n^{\alpha_n}}{\Gamma(\alpha_n)} \cdot \frac{\Gamma(\alpha_n)}{(\beta_n - 3\log(0.5))^{\alpha_n}} \int_0^{\infty} \frac{(\beta_n - 3\log(0.5))^{\alpha_n}}{\Gamma(\alpha_n)} a^{\alpha_n - 1} e^{-(\beta_n - 3\log(0.5))a} \, da$$

$$= \frac{\beta_n^{\alpha_n}}{(\beta_n - 3\log(0.5))^{\alpha_n}} = 0.1768$$

5. Use an opportune approximation to derive a 0.95 HPD credible interval.



we have a distribution, we move the line until we have a mass of 0.95 inside the interval

For large values of α, by **CLT**:

$$\text{Gamma} \xrightarrow{d} N(\mathbb{E}[a], \text{Var}(a))$$

In this case: $\mathbb{E}[a \mid \underline{X}] = \frac{\alpha_n}{\beta_n} = \frac{130}{154,..} = 0.8390$

$$\text{Var}(a \mid \underline{X}) = \frac{\alpha_n}{\beta_n^2} = (0.0736)^2$$

$$\Rightarrow CI_{0.95}(a) = \mathbb{E}[a \mid \underline{X}] \pm z_{0.025} \sqrt{\text{Var}(a \mid \underline{X})} = (0.7028, 0.9772)$$

6. Test $H_0: a \ge 1$ vs. $H_1: a < 1$ using the Bayes Factor.
[Hint: CDF $\chi^2_{20}(10.38) = 0.0393$]

$$BF_{01} = \frac{\text{posterior odds}_{01}}{\text{prior odds}_{01}} = \frac{\mathbb{P}(a \ge 1 \mid \underline{X})}{\mathbb{P}(a < 1 \mid \underline{X})} \cdot \frac{\mathbb{P}(a < 1)}{\mathbb{P}(a \ge 1)}$$

A priori: $a \sim \text{Gamma}(\alpha, \beta) = \text{Gamma}(10, 5.19) = \text{Gamma}(\frac{20}{2}, 5.19)$

And so: $c \cdot a \sim \text{Gamma}(\frac{20}{2}, \frac{1}{2})$ if $c = 2\beta = 2 \cdot 5.19 = 10.38$

$\left(c \cdot a \sim \text{Gamma}(\alpha, \frac{\beta}{c})\right)$

$$\Rightarrow \mathbb{P}(a < 1) = \mathbb{P}\left(\text{Gamma}\left(\tfrac{20}{2}, \tfrac{1}{2}\right) < 10.38\right) = \mathbb{P}(\chi^2_{20} < 10.38) = 0.0393$$

For the posterior:

$$\mathbb{P}(a < 1 \mid \underline{X}) \simeq \Phi\left(\frac{1 - 0.8390}{0.0736}\right) = 0.9856$$

$$\Rightarrow BF_{01} = \frac{1 - 0.9856}{0.9856} \cdot \frac{0.0393}{1 - 0.0393} = 0.0005 \ldots$$

$\Rightarrow$ strong evidence in support for H_1

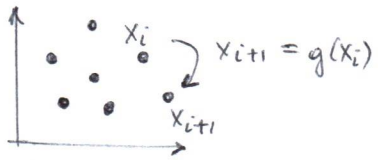
$\Rightarrow$ $a < 1$

RANDOM NUMBERS

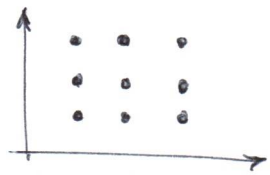
We're dealing with pseudo random numbers.
How do we obtain random numbers?

- Sample in some way a sequence from a uniform: $U_n \sim \mathcal{U}([0,1])$
- Something cool
- Deal with other distributions

There is a distinction: pseudo random vs. quasi random



pseudo random: x_{i+1} is generated deterministically



quasi random: equally distributed forming a grid

RNG (Random Numbers Generator)

Defined by a structure: (S, S_0, f, U, g):

- S = support = finite set of state
- S_0 = initial condition
- f = transition kernel: $f: S \to S$
- U = set of outputs
- g = output function: $g: S \to U$

Pseudocode:
1. Initialize S_0
2. for $i = 1, \dots, n_{rep}$:
   - $S_i = f(S_{i-1})$
   - $U_i = g(S_i)$

Period of a random number generator: smallest m s.t. $S_{i+m} = S_i \quad \forall i$

Example: linear congruential Generators

$$X_n = (a X_{n-1} + c) \bmod m$$

Exercise 1.

Suppose we have $X_1, \dots, X_n$ taking values in $\mathbb{R}^+$.

$$X_1, \dots, X_n \mid \theta, \beta \sim \text{Weibull}(\theta, \beta)$$
$$f(x; \theta, \beta) = \frac{\beta}{\theta} x^{\beta-1} \exp\left\{-\frac{x^\beta}{\theta}\right\} \mathbb{1}_{\mathbb{R}^+}(x)$$

Trick for the conj prior (general variable θ):
$L(\theta, \dots) \propto \theta^n e^{-\theta} \Rightarrow$ gamma
$\propto \frac{1}{\theta^n} e^{-\theta} \Rightarrow$ IG inverse gamma

1. Suppose β known. Derive the conjugate prior $\pi(\theta)$ and write the update rule for the hyperparameters.

$$L(\theta; \underline{x}, \beta) = \prod_{i=1}^n f(x; \theta, \beta) = \frac{\beta^n}{\theta^n} \left[\prod_{i=1}^n x_i^{\beta-1}\right] e^{-\sum_{i=1}^n \frac{x_i^\beta}{\theta}} \mathbb{1}_{\mathbb{R}^+}(x_i)$$

$$= \frac{\beta^n}{\theta^n} \left[\prod_{i=1}^n x_i^{-1}\right] e^{-\sum_{i=1}^n \frac{x_i^\beta}{\theta} + \beta \sum_{i=1}^n \log(x_i)} \mathbb{1}_{\mathbb{R}^+}(x_i)$$

$$\propto \frac{1}{\theta^n} e^{-\frac{1}{\theta} \sum_{i=1}^n x_i^\beta} := S(\beta, \underline{x})$$

$$\propto \frac{1}{\theta^n} e^{-\frac{1}{\theta} S(\beta, \underline{x})}$$

$\Longrightarrow$ looks like the kernel of an inverse gamma distribution, we set:

$$\theta \sim IG(a, b)$$

$\theta \sim IG(a,b)$:

$$\pi(\theta) = \frac{b^a}{\Gamma(a)} \theta^{-a-1} e^{-\frac{b}{\theta}}$$

$$\pi(\theta|\underline{x}) \propto L(\theta; \underline{x}, \beta)\, \pi(\theta) \propto \theta^{-a-n-1} e^{-\frac{1}{\theta}(b + S(\beta,\underline{x}))}$$

$$\Longrightarrow \theta|\underline{x} \sim IG(a_n, b_n) \qquad a_n = a+n, \quad b_n = b + S(\beta,\underline{x})$$

$$\Longrightarrow \begin{cases} X|\beta,\theta & \text{Weibull} \\ \theta|\beta & \text{Inverse Gamma } (a,b) \\ \theta|X,\beta & \text{Inverse Gamma } (a_n,b_n) \end{cases}$$

2. Suppose we have a sample collected from a previous study : $\underline{z} = (z_1, .., z_m)$. Specify the hyperparameters of $\pi(\theta)$ using the Equivalent Sample Principle (ESP).

We start from the MLE : (for θ)

$$\log f(\theta; \underline{x}, \beta) \propto -n \log(\theta) - \frac{S(\beta,\underline{x})}{\theta}$$

$$\Longrightarrow \frac{\partial}{\partial\theta} \log L(\theta; \underline{x}, \beta) = -\frac{n}{\theta} + \frac{S(\beta,\underline{x})}{\theta^2} = 0 \Longrightarrow \hat{\theta}_{MLE} = \frac{S(\beta,\underline{x})}{n}$$

Moreover we have :

$$\mathbb{E}[\theta|\underline{x}] = \frac{b_n}{a_{n-1}} = \frac{b + S(\beta,\underline{x})}{a+n-1} = \frac{a-1}{a+n-1} \cdot \underbrace{\frac{b}{a-1}}_{\mathbb{E}[\theta]} + \frac{n}{a+n-1} \cdot \underbrace{\frac{S(\beta,\underline{x})}{n}}_{\hat{\theta}_{MLE}}$$

$a-1$ plays the role of the sample size
b plays the role of $S(\beta,\underline{x})$

$$\Longrightarrow \text{Elicitation} : \begin{cases} a-1 = m \\ b = S(\beta,\underline{z}) \end{cases} \Longrightarrow \begin{cases} a_1 = m+1 \\ b_1 = S(\beta,\underline{z}) \end{cases}$$

2.' Choose the hyperparameters of the conjugate prior assuming $a_2 > 0$ fixed and $b_2 = \arg\max_b \underbrace{m(\underline{z}; a_2, b)}_{\text{marginal distribution}}$

We need the marginal distribution :

$$m(\underline{z}; a_2, b) = \int L(\theta; \underline{z}, \beta)\, \pi(\theta)\, d\theta$$

$$= \int \frac{\beta^m}{\theta^m} \left(\prod_{i=1}^m z_i^{\beta-1}\right) e^{-\frac{S(\beta,\underline{z})}{\theta}} \frac{b^{a_2}}{\Gamma(a_2)} \theta^{-a_2-1} e^{-\frac{b}{\theta}} d\theta$$

$$= \left(\prod_{i=1}^m z_i^{\beta-1}\right) \beta^m \frac{b^{a_2}}{\Gamma(a_2)} \cdot \frac{\Gamma(a_2+m)}{(b+S(\beta,\underline{z}))^{a_2+m}} \int \theta^{-a_2-m-1} e^{-\frac{1}{\theta}(b+S(\beta,\underline{z}))} \cdot \frac{(b+S(\beta,\underline{z}))^{a_2+m}}{\Gamma(a_2+m)} d\theta$$

$$\frac{d}{db} m(\underline{z}; a_2, b) \propto \frac{d}{db}\left(\frac{b^{a_2}}{(b+S(\beta,\underline{z}))^{a_2+m}}\right) = \frac{a_2 b^{(a_2-1)} (b+S(\beta,\underline{z}))^{a_2+m} - b^{a_2}(a_2+m)(b+S(\beta,\underline{z}))^{a_2+m-1}}{(b+S(\beta,\underline{z}))^{2(a_2+m)}}$$

$$\frac{d}{db}\left(\frac{b^{a_2}}{(b+S(\beta,\underline{z}))^{a_2+m}}\right) = 0 \Longrightarrow b_2 = \frac{a_2 S(\beta,\underline{z})}{m}$$

3. Use the Bayes factor to compare, based on the sample $\underline{x}$, two models with the same likelihood and :

$$M_1 : \pi_1(\theta) = \pi(\theta; a_1, b_1) \qquad a_1 = m+1, \quad b_1 = S(\beta,\underline{z})$$

$$M_2 : \pi_2(\theta) = \pi(\theta; a_2, b_2) \qquad a_2 = m, \quad b_2 = \frac{a_2 S(\beta,\underline{z})}{m}$$

The idea is to test which specification of the hyperparameter is better.

$$BF_{12} = \frac{m(\underline{x}, a_1, b_1)}{m(\underline{x}, a_2, b_2)} = \frac{\beta^n \left(\prod_{i=1}^n x_i \beta^{-1}\right)}{\beta^n \left(\prod_{i=1}^n x_i \beta^{-1}\right)} \cdot \frac{\frac{\Gamma(a_1+n)}{\Gamma(a_1)} \cdot \frac{b_1^{a_1}}{(b_1 + S(\beta,\underline{x}))^{a_1+n}}}{\frac{\Gamma(a_2+n)}{\Gamma(a_2)} \cdot \frac{b_2^{a_2}}{(b_2 + S(\beta,\underline{x}))^{a_2+n}}}$$

$$= \frac{\Gamma(m+n+1)\left(S(\beta,\underline{z})\right)^{m+1}}{\Gamma(m+1)\left(S(\beta,\underline{z}) + S(\beta,\underline{x})\right)^{m+n+1}} \cdot \frac{\Gamma(m)\left(S(\beta,\underline{z}) + S(\beta,\underline{x})\right)^{m+n}}{\Gamma(m+n)\left(S(\beta,\underline{z})\right)^{m}}$$

$$= \frac{m+n}{m} \cdot \frac{S(\beta,\underline{z})}{S(\beta,\underline{z}) + S(\beta,\underline{x})}$$

Suppose: $\underline{z} = (1.51, 0.50, 1.20, 0.32)$

$\underline{x} = (0.54, 0.69, 5.47, 5.29, 4.13, 4.51, 3.68, 4.96, 3.70, 1.54)$

$\beta = 2$

and so: $S(\beta, \underline{z}) = 4.0725$

$S(\beta, \underline{x}) = 150.2753$

$$\Longrightarrow BF_{12} = \frac{4+10}{4} \cdot \frac{4.0725}{4.0725 + 150.2753} = 0.0923$$

$$\Longrightarrow 2\log(BF_{12}) = -4.7654$$

$\Longrightarrow$ we prefer the second prior

Exercise 2.

We consider a M/M/1 model.

$X_i = T_i - T_{i-1}$ = inter arrival time : $X_1, \dots, X_n | \lambda \sim \mathcal{E}(\lambda)$ $\quad \mathbb{E}[X] = \frac{1}{\lambda}$

Y_i = service time : $Y_1, \dots, Y_n | \mu \sim \mathcal{E}\left(\frac{1}{\mu}\right)$ probabilmente è $\mathcal{E}(\mu)$

Data consists in pairs: $(X_1, Y_1), \dots, (X_n, Y_n)$.

Crucial information: $(\underline{X}) \perp\!\!\!\perp (\underline{Y})$.

1. Define the likelihood and derive the conjugate prior $\pi(\lambda, \mu)$. Moreover find the hyperparameters of the posterior.

$$L(\lambda, \mu | \underline{x}, \underline{y}) = \prod_{i=1}^n f(x_i, y_i; \lambda, \mu) = \prod_{i=1}^n f(x_i; \lambda) f(y_i; \mu)$$

$$= \prod_{i=1}^n \lambda \mu e^{-\lambda x_i} e^{-\mu y_i} = \lambda^n \mu^n e^{-\sum \lambda x_i} e^{-\sum \mu y_i}$$

$$= \lambda^n \mu^n e^{-\lambda S_x} e^{-\mu S_y}$$

$\pi(\lambda, \mu) = \pi(\mu)\pi(\lambda)$: $\lambda \sim \text{Gamma}(a, b)$, $\mu \sim \text{Gamma}(c, d)$

$$\pi(\lambda) \overset{d}{=} \frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda}$$

$$\pi(\mu) \overset{d}{=} \frac{d^c}{\Gamma(c)} \mu^{c-1} e^{-d\mu}$$

Posterior: $\lambda | \underline{X}, \underline{Y} \sim \lambda | \underline{X} \sim \text{Gamma}(a+n, b+S_x)$

$\mu | \underline{X}, \underline{Y} \sim \mu | \underline{Y} \sim \text{Gamma}(c+n, d+S_y)$

1' Elicitation of the prior parameters considering an old sample ($m = 8$):

$\underline{x}^{old} = (2, 1.8, 1.7, 2.5, 2.1, 2.0, 1.9, 2.2)$

$\underline{y}^{old} = (1.7, 1.5, 0.9, 1.4, 1.8, 1.6, 1.0, 0.8)$

$$\mathbb{E}[\lambda|\underline{X}] = \frac{a_n}{b_n} = \frac{a+n}{b+S_x} = \left(\frac{b+S_x}{a+n}\right)^{-1} = \left(\frac{a}{a+n}\,\frac{b}{a} + \frac{n}{a+n}\,\frac{S_x}{n}\right)^{-1}$$

a plays the same role as n
b plays the same role as S_x
we have the same with $\mathbb{E}[\mu|\underline{Y}]$

$\Longrightarrow$ Elicitation:
$a = c = n$
$b = S_{x\,old}$
$d = S_{y\,old}$

2. Test the hypothesis: $H_0 : \lambda = 0.5,\ \mu = 1$, $H_1 : \lambda \neq 0.5,\ \mu \neq 1$

$$BF_{01} = \frac{\prod_{i=1}^{n} f(x,y|\lambda=0.5,\mu=1)}{m(\underline{x},\underline{y})} \quad \leftarrow \text{likelihood with } H_0$$

$$m(\underline{x},\underline{y}) = \iint \prod_{i=1}^{n} f(x_i,y_i|\lambda,\mu)\,\pi(\lambda,\mu)\,d\mu\,d\lambda$$

$$= \frac{b^a d^c}{\Gamma(a)\Gamma(c)} \iint \lambda^n \mu^n \lambda^{a-1} \mu^{c-1} e^{-(b+S_x)\lambda - (d+S_y)\mu}\,d\mu\,d\lambda$$

$$= \frac{b^a d^c}{\Gamma(a)\Gamma(c)} \frac{\Gamma(a+n)\Gamma(c+n)}{(b+S_x)^{a+n}(d+S_y)^{c+n}} \iint \ldots d\mu\,d\lambda$$

Considering: $n = 12$, $S_x = 22.6$, $S_y = 18.4$

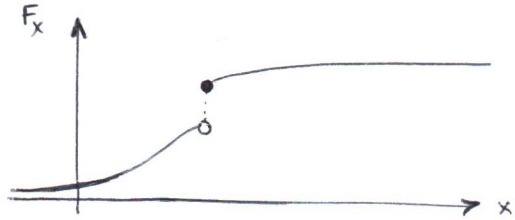
$\Longrightarrow BF_{01} = \dfrac{L(\lambda=0.5,\mu=1|\underline{x},\underline{y})}{m(\underline{x},\underline{y})} = 0.7363 \Longrightarrow$ since it's close to **1** there is no big evidence in case of H_0 or H_1

3. We have that a system is stable if $\frac{\lambda}{\mu} < 1$.
Compute the posterior expectation of $\frac{\lambda}{\mu}$.

$$\mathbb{E}\left[\frac{\lambda}{\mu}\middle|\underline{X},\underline{Y}\right] = \underbrace{\mathbb{E}[\lambda|\underline{X}]}_{\text{gamma}} \cdot \underbrace{\mathbb{E}\left[\frac{1}{\mu}\middle|\underline{Y}\right]}_{\text{inverse gamma}} = \left(\frac{a+n}{b+S_x}\right)\left(\frac{d+S_y}{c+n-1}\right)$$

INVERSE TRANSFORM METHOD

We want to sample $X \sim F_X(\cdot)$ with $F_X(\cdot)$ = cumulative distr. function :
$F: \mathbb{R} \to [0,1]$ monotonically increasing, right continuous, $F(-\infty)=0$, $F(+\infty)=1$



(we can have discontinuities)

$F(x) \sim \mathcal{U}([0,1])$

We consider the generalized inverse of $F(x)$: $U = F^{-1}(X) = \inf\{x : F(x) \geq U\}$

Prop. If $U \sim \mathcal{U}([0,1]) \implies X = F^{-1}(U) \sim F_X$

proof.

$x_0 \in \mathbb{R}$. $\quad A_1 := \{u \in [0,1] : F^{-1}(u) \leq x_0\}$
$\quad A_2 := \{u \in [0,1] : F(x_0) \geq u\}$

The proposition says: $A_1 = A_2$.
consider $u \in A_1$: $F^{-1}(u) \leq x_0$. for right continuity we can reverse:
$u \leq F(x_0) \implies u \in A_2 \implies A_1 \subseteq A_2$
Reversely, consider $u \in A_2$: $u \leq F(x_0)$. By the def. of F: $F^{-1}(u) \leq x_0$
$\implies u \in A_1 \implies A_2 \subseteq A_1 \implies A_1 = A_2$.

$$\mathbb{P}(X \leq x_0) = \mathbb{P}(F^{-1}(U) \leq x_0) = \mathbb{P}(U \leq F(x_0)) = F_X(x_0)$$

(construction; $A_1 = A_2$; $U \sim \mathcal{U}([0,1])$) ∎

Algorithm:
for $i = 1, \dots, n_{rep}$:
1. sample $U_i \sim \mathcal{U}([0,1])$
2. set $X_i = F^{-1}(U_i)$

Example 1. (Exponential distribution)

$X \sim \mathcal{E}(\lambda)$: $f(x) = \lambda e^{-\lambda x} \mathbb{1}_{(0,\infty)}(x)$ continuous w.r.t. the lebesgue measure
We want to sample from this distribution.

$$F(x) = \int_{-\infty}^{x} \lambda e^{-\lambda t} \mathbb{1}_{(0,\infty)}(t)\, dt = 1 - e^{-\lambda x}$$

We impose:

$$U = F(X) = 1 - e^{-\lambda X} \implies 1 - U = e^{-\lambda X}$$
$$\implies \log(1-U) = -\lambda X$$
$$\implies X = -\frac{1}{\lambda}\log(1-U) \overset{d}{=} -\frac{1}{\lambda}\log(U)$$

$U \sim \mathcal{U}([0,1]) = 1-U \sim \mathcal{U}([0,1])$

Example 2. (Gamma distribution)

$X \sim \text{Gamma}(m, b)$, m = integer

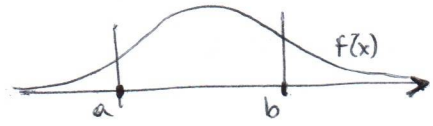
Remark 1: $X_1, \dots, X_m \sim \text{Gamma}(a_i, b) \implies \sum_{i=1}^m X_i \sim \text{Gamma}\left(\sum_{i=1}^m a_i, b\right)$

Remark 2: $\text{Gamma}(1, b) = \mathcal{E}(b)$

Algorithm:
For $i = 1, \dots, n_{rep}$:
1. sample $Y_1, \dots, Y_m \sim \text{Gamma}(1, b) = \mathcal{E}(b)$ (Example 1.)
2. set $X_i = \sum_{j=1}^m Y_j$

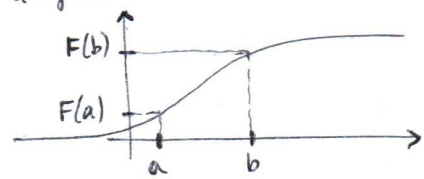
Example 3. (Truncated distribution)

Truncated distribution.
We have $X \sim F_X(\cdot)$ absolutely continuous.
We want to sample from $Y = X|_{X \in A}$,
where $A = (a,b)$.



$$F_Y(y) = F_{X|_{X\in A}}(y) = \mathbb{P}(X \le y \mid X \in A) = \frac{\mathbb{P}(X \le y, X \in A)}{\mathbb{P}(X \in A)}$$

$$= \begin{cases} \dfrac{F_X(y) - F_X(a)}{F_X(b) - F_X(a)} & a < y \le b \\ 0 & y \le a \\ 1 & y > a_2 \end{cases}$$



What is the density?

$$f_Y(y) = \frac{dF_Y}{dy} = \frac{1}{\mathbb{P}(y \in A)} f_X(y) \mathbb{1}_{(a,b)}(x)$$

How to sample?

$$U = F_Y(y) = \frac{F_X(y) - F_X(a)}{F_X(b) - F_X(a)} := \frac{F_X(y) - F_1}{F_2 - F_1} \Longrightarrow \underbrace{(F_2 - F_1)U + F_1}_{\sim \mathcal{U}([F_1, F_2])} = F_X(y)$$

$\Rightarrow$ Instead of sampling from $\mathcal{U}([0,1])$, for a truncated version we sample from $\mathcal{U}([F(a), F(b)])$

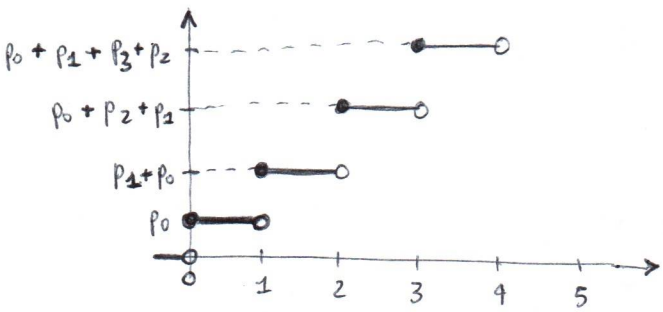
Example 4. (Discrete distribution)

We want to sample from a discrete random variable.

$X \sim F_X(\cdot)$, $X \in \{0, 1, 2, \dots, n\}$, $n \in \mathbb{N}$

and so: $p_X(x) = \sum_{j=0}^{n} p_j \delta_j(x)$.

Graphically:



$$F_X(x) = \sum_{j=0}^{n} p_j \mathbb{1}_{\{j \le x\}}$$

$$F_X^{-1}(U) \overset{(*)}{=} \begin{cases} 0 & U < p_0 \\ 1 & p_0 \le U < p_0 + p_1 \\ 2 & p_0 + p_1 \le U < p_0 + p_1 + p_2 \\ \vdots & \vdots \\ n & \sum_{j=0}^{n-1} \le U \le \sum_{j=0}^{n} p_j \end{cases}$$

Algorithm:

for $i = 1, \dots, n_{rep}$:

1. Sample $U \sim \mathcal{U}([0,1])$
2. Procede according to $(*)$

However, we cannot use this method to sample from a gaussian.

BOX-MULLER METHOD

We want to sample from $X \sim N(0,1)$.
We start from:

$$(X_1, X_2) \overset{iid}{\sim} N(0,1) \quad : \quad f(x_1,x_2) = (2\pi)^{-1/2} e^{-\frac{1}{2}(x_1^2 + x_2^2)}$$

We move to polar coordinates:

$$\begin{cases} \rho = x_1^2 + x_2^2 & \in [0,\infty) \\ \varphi = \tan^{-1}\left(\frac{x_1}{x_2}\right) & \in [0, 2\pi) \end{cases} \iff \begin{cases} x_1 = \sqrt{\rho}\cos(\varphi) \\ x_2 = \sqrt{\rho}\sin(\varphi) \end{cases}$$

$$J = \begin{bmatrix} \frac{\partial x_1}{\partial \rho} & \frac{\partial x_1}{\partial \varphi} \\ \frac{\partial x_2}{\partial \rho} & \frac{\partial x_2}{\partial \varphi} \end{bmatrix} = \begin{bmatrix} -\frac{1}{2\sqrt{\rho}}\cos(\varphi) & -\sqrt{\rho}\sin(\varphi) \\ -\frac{1}{2\sqrt{\rho}}\sin(\varphi) & \sqrt{\rho}\cos(\varphi) \end{bmatrix}$$

$$\Rightarrow f(\rho,\varphi) = f_{\underline{X}}\left(\sqrt{\rho}\cos(\varphi), \sqrt{\rho}\sin(\varphi)\right)|J|\, \mathbb{1}_{[0,\infty)}(\rho)\, \mathbb{1}_{[0,2\pi)}(\varphi)$$

$$= (2\pi)^{-1/2} e^{-1/2\rho}\,|J|$$

$$= \underbrace{(2\pi)^{-1/2}}_{\sim \mathcal{U}([0,2\pi])} \underbrace{e^{-1/2\rho}\,\frac{1}{2}}_{\sim \mathcal{E}(\frac{1}{2})}$$

Algorithm:
for $i = 1, \dots, n_{rep}/2$:

1. sample $U_1, U_2 \sim \mathcal{U}([0,1])$
2. set $\rho = -2\log(U_1)$, $\varphi = (2\pi)U_2$
3. set $X_1 = \sqrt{\rho}\cos(\varphi)$, $X_2 = \sqrt{\rho}\sin(\varphi)$

Variants:

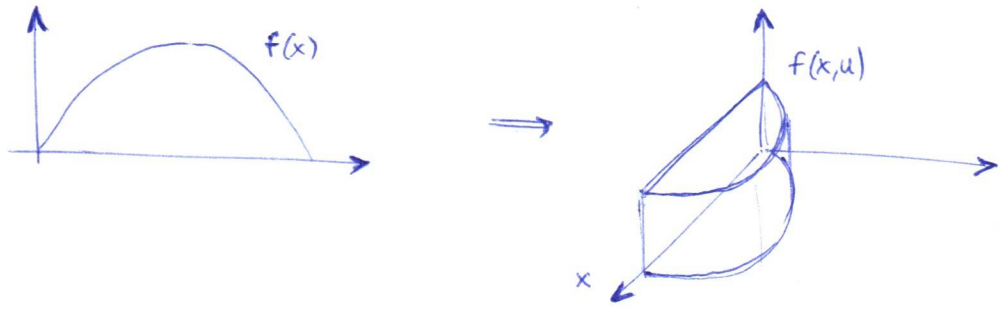
- $Y \sim N(\mu, \sigma^2)$: sample $X \sim N(0,1) \Rightarrow Y = \sigma X + \mu$
- $\underline{Y} \sim N_k(\mu, \Sigma)$, $\Sigma = A^T A$: sample $\underline{X} \sim N_k(0, I) \Rightarrow \underline{Y} = A\underline{X} + \underline{\mu}$

FUNDAMENTAL THEOREM OF SIMULATIONS

Suppose we want to simulate from $X \sim f(x)$, where $f(x)$ is a density function defined on an arbitrary space.

Since $f(x) = \int_0^{f(x)} du \Rightarrow f(x)$ is the marginal of $(X,U) \sim \mathcal{U}\left(\{(x,u) : 0 < u < f(x)\}\right)$

It's a trick: augment the dimension to simplify the problem:

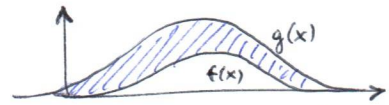


One method based on this theorem is for instance the Acceptance/Rejection method.

ACCEPTANCE / REJECTION METHOD

We are interested in sampling from $X \sim f(x)$.
We don't know how to do it so we use an instrumental random variable $Y \sim g(y)$.
Y has to satisfy the following:

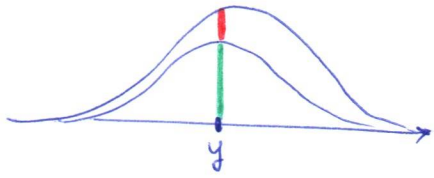
1. $S_X \subseteq S_Y \subseteq \mathbb{R}^k$ (support)
2. $f(x) \leq M g(x) \quad \forall x$



we sample using $g(\cdot)$ and we correct by the difference of the area (///)

Prop. Assume $f(x)$ and $g(x)$ densities s.t. 1. and 2. holds. Then to simulate from $f(x)$ we can simulate from $Y \sim g(y)$ and from $U|Y=y \sim \mathcal{U}([0, Mg(y)])$ until $0 < U < f(y)$.

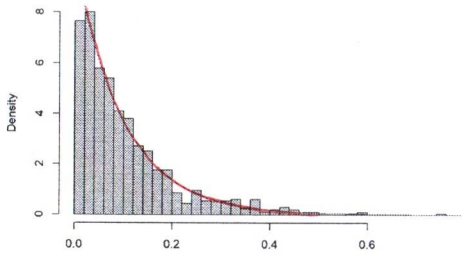
First we simulate on the support; obtaining y.



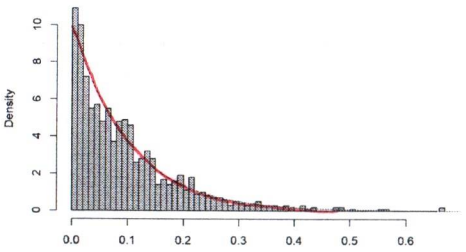
Then we simulate vertically (uniformly). If we fall in the acceptance zone we accept. If we fall in the rejection zone we reject

We augmented a dimension since we simulate twice (we sample horizontally and then vertically before accepting).

```
# if not already installed
# install microbenchmark and RcppArmadillo
# about armadillo, take a look on:
#       http://arma.sourceforge.net/docs.html

library(microbenchmark)
library(RcppArmadillo)
library(Rcpp)
library(ggplot2)
library(gridExtra)

#---------------------------
#
# INVERSE TRANSFORM
#
#---------------------------

#---------------------------
# Exponential(Lambda)
#---------------------------

exp_custom <- function(n, lambda){
  results <- c()
  for(i in 1:n){
    u <- runif(1)
    results[i] <- -log(u) / lambda
  }
  return(results)
}

# plots comparison against the R base implementation
hist(rexp(1000, 10), breaks = 50, freq = F, main = "", xlab = "")
lines(seq(0.001, 1, length.out = 100), dexp(seq(0.001, 1, length.out = 100), 10), col = 2)
```



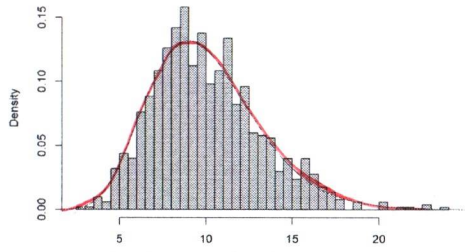

```
hist(exp_custom(1000, 10), breaks = 50, freq = F, main = "", xlab = "")
lines(seq(0.001, 1, length.out = 100), dexp(seq(0.001, 1, length.out = 100), 10), col = 2)
```



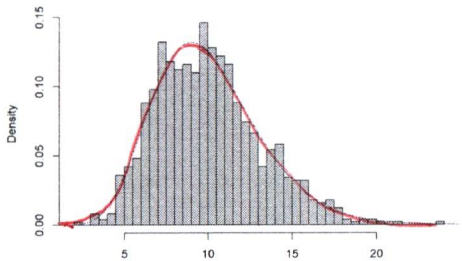

```
#---------------------------
# c++ (using armadillo) implementation
cppFunction("arma::vec exp_custom_CPP(int n, double lambda){
             arma::vec results(n);
             double u;
             for(int i = 0; i < n; i++){
               u = arma::randu();
               results(i) = - log(u) / lambda;
             }
             return results;
           }", depends = "RcppArmadillo")

# comparison
microbenchmark(R = exp_custom(100, 1), CPP = exp_custom_CPP(100, 1), default = rexp(100, 1), times = 1000
)
```

```
## Unit: microseconds
##     expr   min     lq     mean median     uq    max neval
##        R 198.5 207.65 303.0347 230.00 323.75 8228.9  1000
##      CPP   7.0   7.90  12.9705   8.65  10.10 1598.5  1000
##  default   5.7   6.70   8.5299   7.20   8.05   96.4  1000
```

```
#---------------------------
# Gamma RV
#---------------------------

gamma_custom <- function(n, m, b){
  results <- c()
  for(i in 1:n){
    result_temp <- c()
    for(j in 1:m){
      u <- runif(1)
      result_temp[j] <- -log(u) / b
    }
    results[i] <- sum(result_temp)
  }
  return(results)
}

# plots
hist(rgamma(1000, shape = 10, rate = 1), breaks = 50, freq = F, main = "", xlab = "")
lines(seq(0.001, 30, length.out = 1000), dgamma(seq(0.001, 30, length.out = 1000), shape = 10, rate = 1),
col = 2)
```



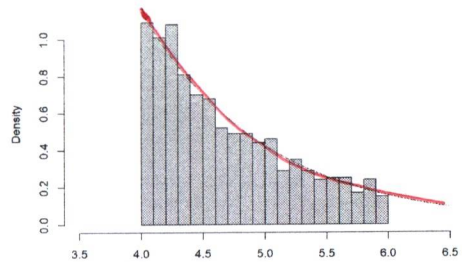

```
hist(gamma_custom(1000, 10, 1), breaks = 50, freq = F, main = "", xlab = "")
lines(seq(0.001, 30, length.out = 1000), dgamma(seq(0.001, 30, length.out = 1000), shape = 10, rate = 1),
col = 2)
```




```
#---------------------------
# c++ (using armadillo) implementation
cppFunction("arma::vec gamma_custom_CPP(int n, int m, double b){
              arma::vec results(n);
              double u;
              arma::vec results_temp(m);
              for(int i = 0; i < n; i++){
                for(int j = 0; j < m; j++){
                  u = arma::randu();
                  results_temp(j) = - log(u) / b;
                }
                results(i) = arma::accu(results_temp);
              }
              return results;
            }", depends = "RcppArmadillo")

# comparison
microbenchmark(R = gamma_custom(100, 10, 1),
               CPP = gamma_custom_CPP(100, 10, 1),
               default = rgamma(100, shape = 10, rate = 1),
               times = 1000)
```

```
## Unit: microseconds
##     expr    min      lq      mean  median      uq     max neval
##        R 2236.1 2378.75 3042.5992 2568.15 2915.65 14596.7  1000
##      CPP   54.9   57.10   70.6677   64.40   71.80  2242.8  1000
##  default   12.6   14.60   21.5196   19.80   24.70   120.5  1000
```

```
#---------------------------
# Truncated RV
#---------------------------
# sample an exponential(1) on the subset of the support (8, 11)

trunc_exp_custom <- function(n, b, a1, a2){
  results <- c()
  for(i in 1:n){
    u <- runif(1) * (exp(-b * a2) - exp(-b * a1)) + exp(-b * a1)
    results[i] <- -log(u) / b
  }
  return(results)
}

hist(trunc_exp_custom(1000, 1, 4, 6), freq = F, xlim = c(3.5, 6.5), breaks = 25, main = "", xlab = "")
curve(dexp(x, 1) / (pexp(6, 1) - pexp(4, 1)), col = 2, add = T)
```




```
#---------------------------
# c++ (using armadillo) implementation
cppFunction("arma::vec trunc_exp_custom_CPP(int n, double b, double a1, double a2){
              arma::vec results(n);
              double u;
              for(int i = 0; i < n; i++){
                u = arma::randu() * (exp(-b * a2) - exp(-b * a1)) + exp(-b * a1);
                results(i) = - log(u) / b;
              }
              return results;
            }", depends = "RcppArmadillo")

# comparison
microbenchmark(R = trunc_exp_custom(100, 1, 4, 6), CPP = trunc_exp_custom_CPP(100, 1, 4, 6), times = 1000
)
```

```
## Unit: microseconds
##  expr   min     lq     mean median     uq     max neval
##     R 238.8 255.75 356.3538  277.6 301.20 11070.9  1000
##   CPP  21.5  22.80  29.7548   24.6  26.15  2206.4  1000
```

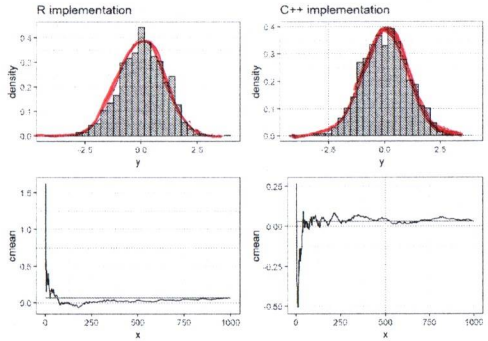
```
#---------------------------
# Discrete RV
#---------------------------

custom_sample <- function(n, wei){
  results <- c()
  cumulate_wei <- cumsum(wei / sum(wei))

  for(i in 1:n){
    u <- runif(1)
    for(j in 1:length(wei)){
      if(u < cumulate_wei[j]){
        results[i] <- j
        break
      }
    }
  }
  return(results)
}

microbenchmark(not_sort = custom_sample(1000, c(0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                                                 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                                                 0.01, 0.8)),
               sort = custom_sample(1000, c(0.8, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                                            0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                                            0.01, 0.01)), times = 1000)
```

```
## Unit: milliseconds
##       expr    min      lq     mean  median      uq     max neval
##  not_sort 3.3268 3.50565 4.433857 3.76540 4.35295 72.5212  1000
##       sort 2.5336 2.68765 3.389816 2.84015 3.28665 14.9865  1000
```

```
#---------------------------
# c++ (using armadillo) implementation
cppFunction("arma::vec custom_sample_CPP(int n, arma::vec wei){
             arma::vec results(n);
             double u;
             arma::vec cumulate_wei = arma::cumsum(wei / accu(wei));

             for(int i = 0; i < n; i++){
               u = arma::randu();
               for(int j = 0; j < wei.n_elem; j++){
                 if(u < cumulate_wei(j)){
                   results(i) = j + 1;
                   break;
                 }
               }
             }
             return results;
           }", depends = "RcppArmadillo")

microbenchmark(R = custom_sample(1000, c(0.1, 0.9, 0.1)),
               CPP = custom_sample_CPP(1000, c(0.1, 0.9, 0.1)),
               default = sample(1:3, size = 1000, replace = T, prob = c(0.1, 0.9, 0.1)), times = 1000)
```

```
## Unit: microseconds
##     expr    min      lq      mean  median      uq     max neval
##        R 2494.7 2638.65 3392.8053 2829.00 3288.4 17122.6  1000
##      CPP   22.4   24.70   40.2625   33.60   40.9  3696.7  1000
##  default   22.6   25.60   38.1405   35.15   43.1   246.1  1000
```

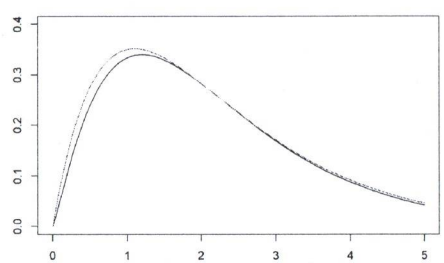
```
#---------------------------
# SAMPLING A GAUSSIAN DISTRIBUTION
#---------------------------

norm_custom <- function(n, mean, sigma2){
  results <- matrix(0, ncol = 2, nrow = n/2)
  for(i in 1:(n/2)){
    u <- runif(2)
    rho = -2 * log(u[1])
    phi = 2 * pi * u[2]
    results[i,1] = sqrt(rho) * cos(phi)
    results[i,2] = sqrt(rho) * sin(phi)
  }
  return(as.vector(results) * sqrt(sigma2) + mean)
}

cppFunction("arma::vec norm_custom_CPP(int n, double mean, double sigma2){
             arma::vec results(n);
             double u1, u2, rho, phi;
             for(int i = 0; i < n/2; i++){
               u1 = arma::randu();
               u2 = arma::randu();
               rho = -2 * log(u1);
               phi = 2 * M_PI * u2;
               results(i) = sqrt(rho) * cos(phi);
               results(i+(n/2)) = sqrt(rho) * sin(phi);
             }
             return (results*sqrt(sigma2) + mean);
           }", depends = "RcppArmadillo")

# plots

sample_R <- norm_custom_CPP(1000, 0, 1)
df1_R <- data.frame(y = sample_R)
df2_R <- data.frame(x = seq(-4, 4, length.out = 100),
                    dens = dnorm(seq(-4, 4, length.out = 100),0,1))
df3_R <- data.frame(cmean = cumsum(sample_R) / 1:length(sample_R),
                    x = 1:length(sample_R))

p1 <- ggplot() +
  geom_histogram(data = df1_R, mapping = aes(y, stat(density)), alpha = 0.3, col = 1, bins = 30) +
  geom_line(data = df2_R, mapping = aes(x = x, y = dens), col = 2)+
  theme_bw() +
  ggtitle("R implementation")
l1 <- ggplot() +
  geom_line(data = df3_R, mapping = aes(x = x, y = cmean)) +
  geom_line(data = df3_R, mapping = aes(x = x, y = mean(sample_R)), col = 2) +
  theme_bw()

sample_CPP <- norm_custom_CPP(1000, 0, 1)
df1_CPP <- data.frame(y = sample_CPP)
df2_CPP <- data.frame(x = seq(-4, 4, length.out = 100),
                      dens = dnorm(seq(-4, 4, length.out = 100),0,1))
df3_CPP <- data.frame(cmean = cumsum(sample_CPP) / 1:length(sample_R),
                      x = 1:length(sample_CPP))

p2 <- ggplot() +
  geom_histogram(data = df1_CPP, mapping = aes(y, stat(density)), alpha = 0.3, col = 1, bins = 30) +
  geom_line(data = df2_CPP, mapping = aes(x = x, y = dens), col = 2)+
  theme_bw() +
  ggtitle("C++ implementation")
l2 <- ggplot() +
  geom_line(data = df3_CPP, mapping = aes(x = x, y = cmean)) +
  geom_line(data = df3_CPP, mapping = aes(x = x, y = mean(sample_CPP)), col = 2) +
  theme_bw()

grid.arrange(p1, p2, l1, l2, nrow = 2)
```



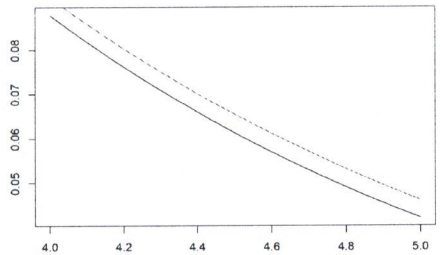

```
# Comparison

microbenchmark(R = norm_custom(100, 0, 1),
               CPP = norm_custom_CPP(100, 0, 1),
               default = rnorm(100, mean = 0, sd = 1),
               times = 100)
```

```
## Unit: microseconds
##     expr    min     lq    mean median     uq     max neval
##        R 124.0 129.55 378.687  145.7 163.15 22024.4   100
##      CPP  10.5  11.25  15.292   11.9  13.25   150.1   100
##  default   9.8  10.35  12.927   10.9  11.90    91.1   100
```

```
#--------------------------
# ACCEPTANCE-REJECTION
#--------------------------

a <- 2.2
b <- round(a) / a
C = gamma(round(a)) / gamma(a) * b^(-round(a)) * ((a - round(a)) / (1 - b))^(a - round(a)) * exp(-(a - ro
und(a)))

curve(dgamma(x, shape = 2.2, rate = 1), xlim = c(0, 5), ylim = c(0, 0.4), xlab = "", ylab = "")
curve(C * dgamma(x, shape = 2, rate = 2/2.2), add = T, col = 2)
```



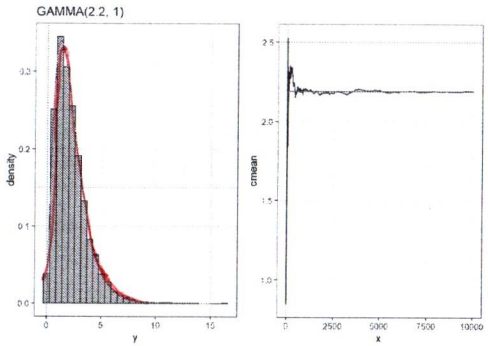
```
curve(dgamma(x, shape = 2.2, rate = 1), xlim = c(4, 5), xlab = "", ylab = "")
curve(C * dgamma(x, shape = 2, rate = 2/2.2), add = T, col = 2)
```



```
#--------------------------

gamma_AR <- function(n, a, b){
  acc <- 0
  k <- 0
  a_star <- round(a)
  b_star <- a_star / a
  M <- gamma(a_star) / gamma(a) * b_star^(-a_star) * ((a - a_star) / (1 - b_star))^(a - a_star) * exp(-
 (a - a_star))
  results <- c()
  while(acc < n){
    y <- rgamma(1, shape = a_star, rate = b_star)
    u <- runif(1)
    if(u <= dgamma(y, shape = a, rate = b) / (M * dgamma(y, shape = a_star, rate = b_star))){
      acc = acc + 1
      results <- c(results, y)
    }
    k <- k + 1
  }
  print(paste0("Acc. ratio = ", round(acc / k, digits = 3)))
  return(results)
}

# plots

sample_GAMMA <- gamma_AR(10000, 2.2, 1)
```

```
## [1] "Acc. ratio = 0.95"
```

```
df1 <- data.frame(y = sample_GAMMA)
df2 <- data.frame(x = seq(0, 15, length.out = 100),
                  dens = dgamma(seq(0, 15, length.out = 100),shape = 2.2, rate = 1))
df3 <- data.frame(cmean = cumsum(sample_GAMMA) / 1:length(sample_GAMMA),
                  x = 1:length(sample_GAMMA))

p <- ggplot() +
  geom_histogram(data = df1, mapping = aes(y, stat(density)), alpha = 0.3, col = 1, bins = 30) +
  geom_line(data = df2, mapping = aes(x = x, y = dens), col = 2)+
  theme_bw() +
  ggtitle("GAMMA(2.2, 1)")
l <- ggplot() +
  geom_line(data = df3, mapping = aes(x = x, y = cmean)) +
  geom_line(data = df3, mapping = aes(x = x, y = mean(sample_GAMMA)), col = 2) +
  theme_bw() +
  ggtitle("")

grid.arrange(p, l, nrow = 1)
```




```
#---------------------------

cppFunction("arma::vec gamma_AR_CPP(int n, double a, double b){
             arma::vec results(n, arma::fill::zeros);
             int acc = 0;
             int k = 0;
             double y, u;
             double a_star = round(a);
             double b_star = a_star / a;
             double M = pow(b_star, -a_star) * pow(((a - a_star) / (1 - b_star)), (a - a_star)) *
                        exp(- (a - a_star) + lgamma(a_star) - lgamma(a));
             while(acc < n - 1){
               y = arma::randg(arma::distr_param(a, 1 / b));
               u = arma::randu();
               if(u < pow(b_star, - a_star) * pow(y, a - a_star) *
                      exp(- y * (1 - b_star) + lgamma(a_star) - lgamma(a)) / M){
                 results(acc) = y;
                 acc += 1;
                       }
               k += 1;
             }
             Rcpp::Rcout << (double) acc / (double) k;
             return (results);
           }", depends = "RcppArmadillo")

microbenchmark(R = gamma_AR(100, 2.2, 1),
               CPP = gamma_AR_CPP(100, 2.2, 1),
               times = 1000)
```

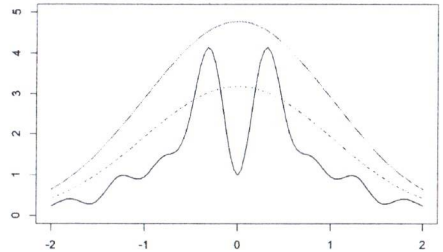
```
## [1] "Acc. ratio = 0.952"
## [1] "Acc. ratio = 0.952"
## 0.9339620.9519230.970588[1] "Acc. ratio = 0.901"
## 0.9611650.970588[1] "Acc. ratio = 0.926"
## ..
```

```
## Unit: microseconds
##  expr  min      lq      mean  median      uq      max neval
##     R 1013 1108.05 1516.3743 1183.85 1393.25 103459.6  1000
##   CPP  339  363.35  407.6909  385.95  415.60   1655.6  1000
```

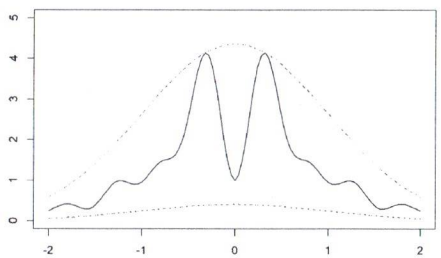
```
#---------------------------
# ACCEPTANCE-REJECTION ~-~ 2
#---------------------------

fx <- function(x) exp(-x^2 / 2) * (sin(6 * x)^2 + 3 * cos(x)^2*sin(4*x)^2 + 1)
integrate(fx, -Inf, Inf)$value
```

```
## [1] 5.89434
```

```
curve(fx, xlim = c(-2, 2), ylim = c(0, 5), xlab = "", ylab = "")
curve(dnorm(x)*8, col = 2, lty = 2, add = T)
curve(dnorm(x)*12, col = 2, lty = 1, add = T)
```



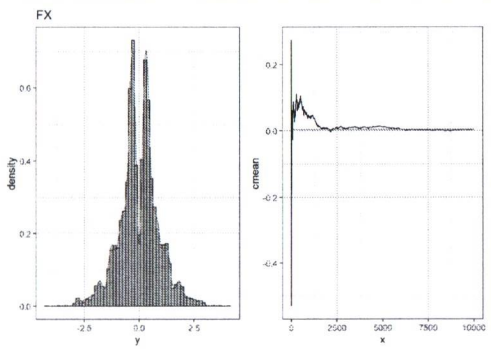
```
M <- -optim(0, fn = function(x) -fx(x) / dnorm(x), method = "Brent", lower = -10, upper = 10)$value
curve(fx, xlim = c(-2, 2), ylim = c(0, 5), xlab = "", ylab = "")
curve(dnorm(x), col = 2, lty = 2, add = T)
curve(dnorm(x)*M, col = 3, lty = 2, add = T)
```



```
fx_AR <- function(n, M){
  acc <- 0
  k <- 0
  results <- c()
  while(acc < n){
    y <- rnorm(1)
    u <- runif(1)
    if(u <= fx(y) / (M * dnorm(y))){
      acc = acc + 1
      results <- c(results, y)
    }
    k <- k + 1
  }
  print(paste0("Acc. ratio = ", round(acc / k, digits = 3)))
  return(results)
}

# plots

fx_sample <- fx_AR(10000, M)
```

```
## [1] "Acc. ratio = 0.546"
```

```
df1 <- data.frame(y = fx_sample)
df2 <- data.frame(x = seq(-4, 4, length.out = 1000),
                  dens = fx(seq(-4, 4, length.out = 1000)) / integrate(fx, -Inf, Inf)$value)
df3 <- data.frame(cmean = sapply(1:length(fx_sample), function(x) mean(fx_sample[1:x])),
                  x = 1:length(fx_sample))

p <- ggplot() +
  geom_histogram(data = df1, mapping = aes(y, stat(density)), alpha = 0.3, col = 1, bins = 60) +
  geom_line(data = df2, mapping = aes(x = x, y = dens), col = 2)+
  theme_bw() +
  ggtitle("FX")
l <- ggplot() +
  geom_line(data = df3, mapping = aes(x = x, y = cmean)) +
  geom_line(data = df3, mapping = aes(x = x, y = mean(fx_sample)), col = 2) +
  theme_bw() +
  ggtitle("")

grid.arrange(p, l, nrow = 1)
```



ACCEPTANCE/REJECTION METHOD

We want to sample from $X \sim f_X$.

Prop. let $X \sim f_X$ and $Y \sim g_X$ s.t.:

1. $S_{f_X} \subseteq S_{g_X} \subseteq \mathbb{R}^k$
2. $\exists M > 0 : \; f_X \leq M g_X$

$\rightarrow$ To simulate from X is sufficient to simulate from $Y \sim g_X$ and $U|Y=y \sim \mathcal{U}([0, M g_X(y)])$ until $0 < U < f(y)$

The proposition is saying: $\mathbb{P}\left(Y \leq x \mid U \leq \frac{f_X(Y)}{M g_X(Y)}\right) = \mathbb{P}(X \leq x)$

proof. $(u = u)$

$$\mathbb{P}\left(Y \leq x \mid U \leq \frac{f_X(Y)}{M g_X(Y)}\right) = \frac{\mathbb{P}\left(Y \leq x, \; U \leq \frac{f_X(Y)}{g_X(Y) \cdot M}\right)}{\mathbb{P}\left(U \leq \frac{f_X(Y)}{M g_X(Y)}\right)}$$

$$= \frac{\mathbb{E}_Y\left[\mathbb{P}\left(Y \leq x, U \leq \frac{f_X(Y)}{M g_X(Y)} \mid Y = y\right)\right]}{\mathbb{E}_Y\left[\mathbb{P}\left(U \leq \frac{f_X(Y)}{M g_X(Y)} \mid Y = y\right)\right]}$$

$$= \frac{\int \mathbb{P}\left(Y \leq x, U \leq \frac{f_X(Y)}{M g_X(Y)} \mid Y = y\right) \cdot g(y)\, dy}{\int \mathbb{P}\left(U \leq \frac{f_X(Y)}{M g_X(Y)} \mid Y = y\right) g(y)\, dy}$$

$$= \frac{\int \mathbb{1}_{(-\infty, x)}(y) \; \mathbb{P}\left(U \leq \frac{f_X(y)}{M g_X(y)}\right) g(y)\, dy}{\int \mathbb{P}\left(U \leq \frac{f_X(y)}{M g_X(y)}\right) g(y)\, dy}$$

Here $U \sim \mathcal{U}([0,1])$ (because of the convection), we use its cdf

$$= \frac{\int \mathbb{1}_{(-\infty, x)}(y) \; \frac{f_X(y)}{M \cancel{g_X(y)}} \cancel{g_X(y)}\, dy}{\int \frac{f_X(y)}{M \cancel{g_X(y)}} \cancel{g_X(y)}\, dy}$$

$$= \frac{\int \mathbb{1}_{(-\infty, x)}(y) \; f_X(y)\, dy}{\int f_X(y)\, dy} = \mathbb{P}(X \leq x) \qquad \blacksquare$$

$\int f_X(y)\, dy = 1$

Algorithm:

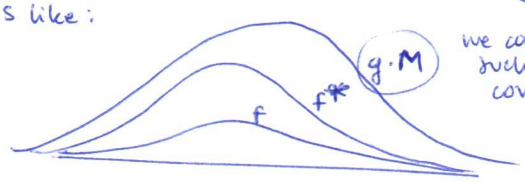
for $i = 1, \ldots, n_{rep}$:

1. Sample $Y \sim g_X$
2. Sample $U \sim \mathcal{U}([0,1])$
3. If $U \leq \frac{f(y)}{M g(y)}$ set $X_i = Y$, otherwise go back to 1.

Remark 1. The algorithm works also with not normalized densities. Suppose to have: $f_X = a f_X^*$, $g_X = b g_X^*$, where f_X^* and g_X^* are not normalized densities.

$\Longrightarrow \; f_X = M g_X \iff f_X^* = M^* g_X^*$ — we can always choose another M^* (s.t. the algorithm works)

It's like:



we can always find an M such that $M \cdot g$ (or in case $M^* \cdot g^*$) covers both f and f^*

Remark 2. What is the probability of accepting a proposed value?

$$\mathbb{P}\left(U \le \frac{f_X(Y)}{M g_X(Y)}\right) = \int \mathbb{P}\left(U \le \frac{f_X(Y)}{M g_X(Y)} \mid Y=y\right) g(y)\,dy$$

$$= \int \frac{f_X(y)}{M g_X(y)} g(y)\,dy$$

$$= \frac{1}{M}\int f_X(y)\,dy = \frac{1}{M}$$

if we want to optimize the algorithm we have to maximize this probability (and so we have to reduce M)

Example.

We want to sample from $f_X \overset{d}{=} \text{Gamma}(\alpha, 1)$, $\alpha > 1$.

We use as proposal $g_X \overset{d}{=} \text{Gamma}(m, b)$, with $m = \lfloor \alpha \rfloor$.

Find M (if $\exists$) s.t. $f_X \le M g_X$.

We study:

$$\frac{f_X}{g_X} = \frac{\frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}}{\frac{1}{\Gamma(m)} b^m x^{m-1} e^{-bx}} = \frac{\Gamma(m)}{\Gamma(\alpha)} b^{-m} \underbrace{x^{\alpha-m} e^{-(1-b)x}}_{\text{Gamma}(\alpha-m+1,\ 1-b)} \quad (*)$$

(Conditions)

If $1-b>0 \Rightarrow b<1$

If $\alpha - m + 1 > 0 \Rightarrow \alpha - \lfloor \alpha \rfloor - 1 > 0$ (always)

The only condition is $b \in (0,1)$ (to guarantee a proper density).

For $b \in (0,1)$ the maximum of the Gamma is in its mode:

$$\text{mode}\left(\text{Gamma}(\alpha-m+1, 1-b)\right) = \frac{\alpha - \lfloor \alpha \rfloor}{1-b} = x \text{ in } (*)$$

$\Rightarrow$ the maximum value of the ratio is in $\frac{\alpha-\lfloor\alpha\rfloor}{1-b}$ (ratio: $\frac{f(\cdot)}{g(\cdot)}$)

$$\frac{f_X}{g_X} \le \frac{\Gamma(\lfloor\alpha\rfloor)}{\Gamma(\alpha)} b^{-\lfloor\alpha\rfloor} \left(\frac{\alpha-\lfloor\alpha\rfloor}{1-b}\right)^{\alpha-\lfloor\alpha\rfloor} e^{-(1-b)\left(\frac{\alpha-\lfloor\alpha\rfloor}{1-b}\right)} := M(b)$$

since α is fixed, the expression is a function of b

We can optimize it in terms of b.

Notice: it's simpler to study $M^{-1}(b)$: (not the inverse, $\frac{1}{M(b)}$)

We have that: $\frac{1}{M(b)} \propto b^{\lfloor\alpha\rfloor}(1-b)^{\alpha-\lfloor\alpha\rfloor} \propto \text{Beta}(\lfloor\alpha\rfloor+1,\ \lfloor\alpha\rfloor-\alpha+1)$

Optimize in terms of b = minimize $M(b)$ = maximize $M^{-1}(b)$.

TO maximize $\frac{1}{M(b)} \overset{d}{=} \text{Beta}(\cdot,\cdot)$ $\longrightarrow$ we consider the mode of $\text{Beta}(\cdot,\cdot)$

The mode of $\text{Beta}(\cdot,\cdot)$ $\exists$ only if both params are > 1.

The mode of $\text{Beta}(\cdot,\cdot)$ is in $\frac{\lfloor\alpha\rfloor}{\alpha}$

$$\Rightarrow b^{\text{optimal}} = \frac{\lfloor\alpha\rfloor}{\alpha} \Rightarrow M^{\text{optimal}} = M(b^{\text{optimal}})$$

IMPORTANCE SAMPLING

Let f_X be a (regular enough) density and h_X a measurable function.
We're interested in: $\mathbb{E}_f[h(X)]$?

$$\mathbb{E}_f[h(X)] = \int_{\mathbb{R}} h(x) f(dx)$$

A first trivial approach: Monte Carlo method:

$$\mathbb{E}_f[h(X)] \approx \frac{1}{m} \sum_{i=1}^{m} h(X_i) \qquad \text{with} \quad X_1, \dots, X_m \overset{iid}{\sim} f_X .$$

By the (strong) law of large Numbers ((S)LLN):

$$\frac{1}{m} \sum_{i=1}^{m} h(X_i) \xrightarrow{a.s.} \mathbb{E}_f[h(X)]$$

Here we said we're not able to sample from f_X, we still can evaluate f_X where we want

However, in some scenarios we're not able to sample from f_X.
To overcome the problem we introduce g_X, an instrumental random variable.

- We generate $X_1, \dots, X_m \sim g_X$
- $\mathbb{E}_f[h(X)] \approx \frac{1}{m} \sum_{i=1}^{m} \frac{f(X_i)}{g(X_i)} h(X_i)$

We introduce the weigh function: $w(x) = \frac{f(x)}{g(x)}$ $\left(w(X_i) = \frac{f(X_i)}{g(X_i)} \right)$
We have that:

$$\frac{1}{m} \sum_{i=1}^{m} \frac{f(X_i)}{g(X_i)} h(X_i) \xrightarrow{a.s.} \mathbb{E}_g\left[\frac{f(X)}{g(X)} h(X) \right] = \int \frac{f(x)}{g(x)} h(x) g(x) dx$$

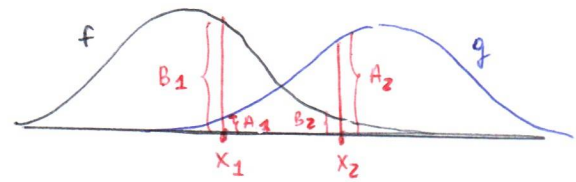
$$= \int f(x) h(x) dx = \mathbb{E}_f[h(X)]$$

Remark: we can use importance sampling to sample from a distribution.
We can approximate $f(x)$:

$$\tilde{f}(x) = \frac{1}{m} \sum_{i=1}^{m} \frac{f(x_i)}{g(x_i)} \delta_{x_i}(x)$$

$$= \frac{1}{m} \sum_{i=1}^{m} w(x_i) \delta_{x_i}(x)$$

→ when we're not able to sample from $f(x)$ we can sample from $g(x)$ and re-weight the probability by $w(x)$.



$w_1 = \frac{B_1}{A_1}$, $w_2 = \frac{B_2}{A_2}$

Algorithm:
for $i = 1, \dots, n_{rep}$:

1. Sample $X_1, \dots, X_m \sim g$ (m large → good (global) sample of f)
2. Compute $(w_1, \dots, w_m)$ with $w_j = \frac{f(x_j)}{g(x_j)}$
3. Sample from $X_1, \dots, X_m$ with probabilities $(w_1, \dots, w_m)$

Theorem. The optimal choice of g_X (that minimizes the variance of the importance sampling estimator) is:

$$g_X^*(x) = \frac{|h(x)| f(x)}{\int |h(t)| f(t) dt}$$

proof.

$$\mathrm{Var}_g\left(h(x) \frac{f(x)}{g(x)} \right) = \mathbb{E}_g\left[\left(h(x) \frac{f(x)}{g(x)} \right)^2 \right] - \mathbb{E}_g^2\left[h(x) \frac{f(x)}{g(x)} \right]$$

Notice that: $\mathbb{E}_g\left[h(x) \frac{f(x)}{g(x)} \right] = \int h(x) \frac{f(x)}{g(x)} g(x) dx = \int h(x) f(x) dx$ // $g(x)$

$\Longrightarrow$ $g(x)$ that minimizes $\mathrm{Var}_g(\cdot)$ = $g(x)$ that minimizes $\mathbb{E}_g[(\cdot)^2]$

$$\mathbb{E}_g\left[h^2(x) \frac{f^2(x)}{g^2(x)} \right] \geq \left(\mathbb{E}_g\left[|h(x)| \frac{f(x)}{g(x)} \right] \right)^2 = \left(\int |h(x)| f(x) dx \right)^2 \Longrightarrow g^*(x) = \frac{|h(x)| f(x)}{\int |h(x)| f(x) dx}$$

$f(\cdot), g(\cdot) \geq 0$

with this we minimize $\mathbb{E}_g[(\cdot)^2]$

Suppose we want to evaluate the tails of a distribution:

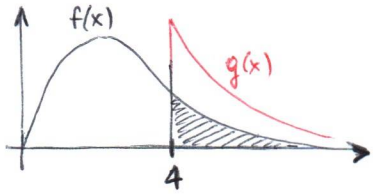
Example.

$X \sim \text{Weibull}(2, 2)$.

We want to evaluate: $\mathbb{P}(X > 4)$?

$$f(x) = \frac{k}{\lambda} x^{k-1} e^{-(\frac{x}{\lambda})^k}$$

We want to evaluate:



We want to treat it as an importance sampling problem:

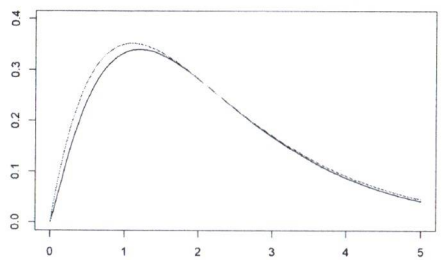
$f(x) \sim \text{Weibull}(2, 2)$

$h(x) = \mathbb{1}_{(4,\infty)}(x)$

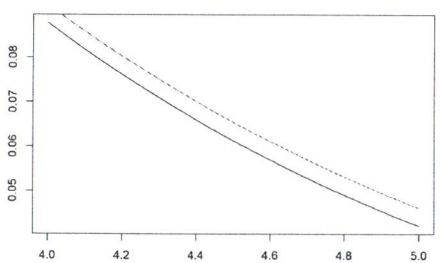
$g(x) \sim \mathcal{E}(\lambda) + 4$

```
library(microbenchmark)
library(RcppArmadillo)
library(Rcpp)
library(ggplot2)
library(gridExtra)
library(ggpubr)

#--------------------------
# ACCEPTANCE-REJECTION
#--------------------------

a <- 2.2
b <- round(a) / a
C = gamma(round(a)) / gamma(a) * b^(-round(a)) * ((a - round(a)) / (1 - b))^(a - round(a)) * exp(-(a - ro
und(a)))

curve(dgamma(x, shape = 2.2, rate = 1), xlim = c(0, 5), ylim = c(0, 0.4), xlab = "", ylab = "")
curve(C * dgamma(x, shape = 2, rate = 2/2.2), add = T, col = 2)
```

: Gamma (·,·)



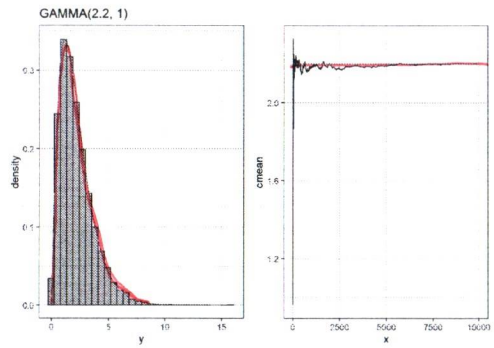

```
curve(dgamma(x, shape = 2.2, rate = 1), xlim = c(4, 5), xlab = "", ylab = "")
curve(C * dgamma(x, shape = 2, rate = 2/2.2), add = T, col = 2)
```




```
#---------------------------

gamma_AR <- function(n, a, b){
  acc <- 0
  k <- 0
  a_star <- round(a)
  b_star <- a_star / a
  M <- gamma(a_star) / gamma(a) * b_star^(-a_star) * ((a - a_star) / (1 - b_star))^(a - a_star) * exp(-
(a - a_star))
  results <- c()

  while(acc < n){
    y <- rgamma(1, shape = a_star, rate = b_star)
    u <- runif(1)
    if(u <= dgamma(y, shape = a, rate = b) / (M * dgamma(y, shape = a_star, rate = b_star))){
      acc = acc + 1
      results <- c(results, y)
    }
    k <- k + 1
  }

  print(paste0("Acc. ratio = ", round(acc / k, digits = 3)))
  return(results)
}

# plots

sample_GAMMA <- gamma_AR(10000, 2.2, 1)
```

```
## [1] "Acc. ratio = 0.951"
```

```
df1 <- data.frame(y = sample_GAMMA)
df2 <- data.frame(x = seq(0, 15, length.out = 100),
                  dens = dgamma(seq(0, 15, length.out = 100),shape = 2.2, rate = 1))
df3 <- data.frame(cmean = sapply(1:length(sample_GAMMA), function(x) mean(sample_GAMMA[1:x])),
                  x = 1:length(sample_GAMMA))

p <- ggplot() +
  geom_histogram(data = df1, mapping = aes(y, stat(density)), alpha = 0.3, col = 1, bins = 30) +
  geom_line(data = df2, mapping = aes(x = x, y = dens), col = 2)+
  theme_bw() +
  ggtitle("GAMMA(2.2, 1)")
l <- ggplot() +
  geom_line(data = df3, mapping = aes(x = x, y = cmean)) +
  geom_line(data = df3, mapping = aes(x = x, y = mean(sample_GAMMA)), col = 2) +
  theme_bw() +
  ggtitle("")

grid.arrange(p, l, nrow = 1)
```



```
#---------------------------

cppFunction("arma::vec gamma_AR_CPP(int n, double a, double b){
              arma::vec results(n, arma::fill::zeros);
              int acc = 0;
              int k = 0;
              double y, u;
              double a_star = round(a);
              double b_star = a_star / a;
              double M = pow(b_star, -a_star) * pow(((a - a_star) / (1 - b_star)), (a - a_star)) *
                         exp(- (a - a_star) + lgamma(a_star) - lgamma(a));
              while(acc < n - 1){
                y = arma::randg(arma::distr_param(a, 1 / b));
                u = arma::randu();
                if(u < pow(b_star, - a_star) * pow(y, a - a_star) *
                        exp(- y * (1 - b_star) + lgamma(a_star) - lgamma(a)) / M){
                  results(acc) = y;
                  acc += 1;
                       }
                k += 1;
              }
              Rcpp::Rcout << (double) acc / (double) k;
              return (results);
            }", depends = "RcppArmadillo")

microbenchmark(gamma_AR(100, 2.2, 1),
               gamma_AR_CPP(100, 2.2, 1),
               times = 100)
```

```
## [1] "Acc. ratio = 0.935"
## 0.961165[1] "Acc. ratio = 0.962"
## 0.9801980.951923[1] "Acc. ratio = 0.935"
## ..
```

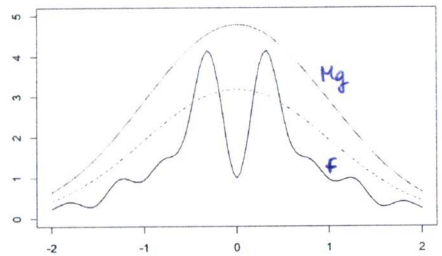
```
## Unit: microseconds
##                       expr      min        lq      mean   median        uq
##      gamma_AR(100, 2.2, 1) 1038.901 1190.0505 1676.8950 1522.052 1712.3510
##  gamma_AR_CPP(100, 2.2, 1)  344.101  374.7015  434.1711  400.751  431.9005
##       max neval
##  8215.901   100
##  1950.900   100
```

```
#---------------------------
# ACCEPTANCE-REJECTION ~~~ 2
#---------------------------

fx <- function(x) exp(-x^2 / 2) * (sin(6 * x)^2 + 3 * cos(x)^2*sin(4*x)^2 + 1)
integrate(fx, -Inf, Inf)$value
```

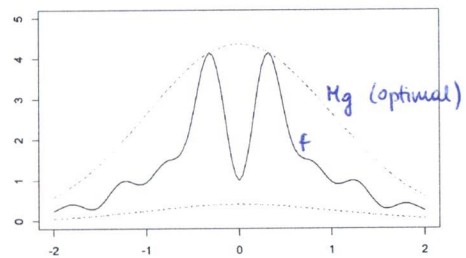
Not normalized densities

```
## [1] 5.89434
```

```
curve(fx, xlim = c(-2, 2), ylim = c(0, 5), xlab = "", ylab = "")
curve(dnorm(x)*8, col = 2, lty = 2, add = T)
curve(dnorm(x)*12, col = 2, lty = 1, add = T)
```



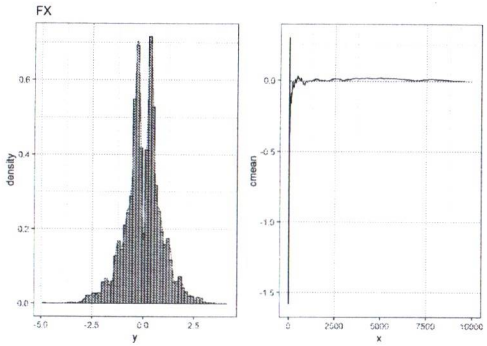

```
M <- -optim(0, fn = function(x) -fx(x) / dnorm(x), method = "Brent", lower = -10, upper = 10)$value
curve(fx, xlim = c(-2, 2), ylim = c(0, 5), xlab = "", ylab = "")
curve(dnorm(x), col = 2, lty = 2, add = T)
curve(dnorm(x)*M, col = 3, lty = 2, add = T)
```




```
fx_AR <- function(n, M){
  acc <- 0
  k <- 0
  results <- c()

  while(acc < n){
    y <- rnorm(1)
    u <- runif(1)
    if(u <= fx(y) / (M * dnorm(y))){
      acc = acc + 1
      results <- c(results, y)
    }
    k <- k + 1
  }
  print(paste0("Acc. ratio = ", round(acc / k, digits = 3)))
  return(results)
}

# plots

fx_sample <- fx_AR(10000, M)
```

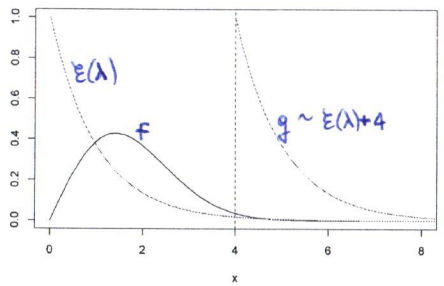
```
## [1] "Acc. ratio = 0.535"
```

```
df1 <- data.frame(y = fx_sample)
df2 <- data.frame(x = seq(-4, 4, length.out = 1000),
                  dens = fx(seq(-4, 4, length.out = 1000)) / integrate(fx, -Inf, Inf)$value)
df3 <- data.frame(cmean = sapply(1:length(fx_sample), function(x) mean(fx_sample[1:x])),
                  x = 1:length(fx_sample))

p <- ggplot() +
  geom_histogram(data = df1, mapping = aes(y, stat(density)), alpha = 0.3, col = 1, bins = 60) +
  geom_line(data = df2, mapping = aes(x = x, y = dens), col = 2)+
  theme_bw() +
  ggtitle("FX")
l <- ggplot() +
  geom_line(data = df3, mapping = aes(x = x, y = cmean)) +
  geom_line(data = df3, mapping = aes(x = x, y = mean(fx_sample)), col = 2) +
  theme_bw() +
  ggtitle("")

grid.arrange(p, l, nrow = 1)
```

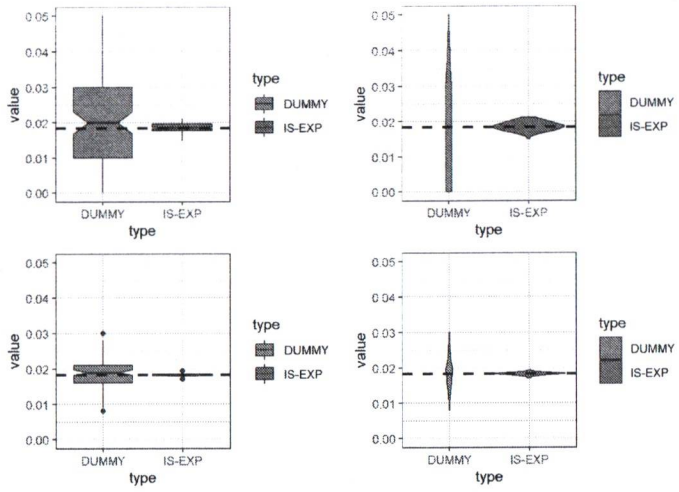



```
#--------------------------
# IMPORTANCE SAMPLING
#--------------------------
# evaluate the probability on the right tail of a weibull(2,2) distribution

curve(dweibull(x, 2, 2), xlim = c(0, 8), ylim = c(0, 1), main = "", ylab = "")
abline(v = 4, lty = 2)
lines(seq(0, 10, length.out = 100), dexp(seq(0, 10, length.out = 100)), col = 4)
lines(seq(4, 10, length.out = 100), dexp(seq(4, 10, length.out = 100) - 4), col = 2)
```

Right tail of a Weibull(·,·)




```
#--------------------------
# dummy sampler: counts how many values are greater than a threshold

dummy_wei <- function(n, a, b, c){
  temp <- rweibull(n, a, b)
  return(mean(temp > c))
}

#--------------------------
# importance sampling: sample form an exponential distribution translated by the threshold

imp_wei <- function(n, a, b, c){
  wei <- c()
  for(i in 1:n){
    temp <- rgamma(1, 1, 1) + c
    wei[i] <- dweibull(temp, a, b) / dgamma(temp - c, 1, 1)
  }
  return(mean(wei))
}

#--------------------------
# replicate the evaluation of the tail one hundred of times

results <- rep(0, 200)
for(i in 1:100){
  results[i] <- dummy_wei(100, 2, 2, 4)
  results[i + 100] <- imp_wei(100, 2, 2, 4)
}

df <- data.frame(res = results, type = rep(c("DUMMY", "IS-EXP"), each = 100))

p1 <- ggplot(df) +
  geom_boxplot(aes(x = type, y = res, fill = type), notch=T) +
  geom_hline(yintercept = pweibull(4, 2, 2, lower.tail = F), col = 1, lwd = 1, lty = 2) +
  theme_bw() +
  ylim(c(0, 0.05)) +
  ylab("value")

p2 <- ggplot(df) +
  geom_violin(aes(x = type, y = res, fill = type)) +
  geom_hline(yintercept = pweibull(4, 2, 2, lower.tail = F), col = 1, lwd = 1, lty = 2) +
  theme_bw() +
  ylim(c(0, 0.05)) +
  ylab("value")

#----------------------------

results <- rep(0, 200)
for(i in 1:100){
  results[i] <- dummy_wei(1000, 2, 2, 4)
  results[i + 100] <- imp_wei(1000, 2, 2, 4)
}

df <- data.frame(res = results, type = rep(c("DUMMY", "IS-EXP"), each = 100))

p3 <- ggplot(df) +
  geom_boxplot(aes(x = type, y = res, fill = type), notch=T) +
  geom_hline(yintercept = pweibull(4, 2, 2, lower.tail = F), col = 1, lwd = 1, lty = 2) +
  theme_bw() +
  ylim(c(0, 0.05)) +
  ylab("value")

p4 <- ggplot(df) +
  geom_violin(aes(x = type, y = res, fill = type)) +
  geom_hline(yintercept = pweibull(4, 2, 2, lower.tail = F), col = 1, lwd = 1, lty = 2) +
  theme_bw() +
  ylim(c(0, 0.05)) +
  ylab("value")

ggarrange(p1, p2, p3, p4, ncol = 2, nrow = 2)
```


0.05
0.04
0.03
0.02
0.01
0.00
value
DUMMY
IS-EXP
type
type
DUMMY
IS-EXP

MCMC

We want to sample from $f_X(x)$ with support S_X.
We use a sequence $\{x^{(t)}\}$ with ergodic distribution equivalent to $f_X(x)$.
How to generate this sequence $(\{x^{(t)}\})$?
A possible strategy is the use of the METROPOLIS-HASTINGS ALGORITHM.
We use a proposal distribution to propose a candidate at the t-step:

$$q(y|x^{t-1}) = q(y|x) \qquad \text{(notations)}.$$

We need some requirements:

- $q(\cdot|x)$ is a density $\forall x \in S_X$
- $q(y|\cdot)$ is measurable $\forall y$
- $S_X \subseteq S_q \subseteq \mathbb{R}^k$ (the support of the proposal must cover the support of the target var)

How does it work?
We propose a value for $f(x)$ by the conditional distribution $q(y|x)$ and we have a probability of accepting (rejecting):

$$\alpha(y,x) = \min\left\{1, \frac{f(y)\,q(x|y)}{f(x)\,q(y|x)}\right\}$$

$\Longrightarrow$
1. We propose a value $y \sim q(y|x)$
2. We accept with probability $\alpha(x,y)$ (= probability to move from x to y, while $1-\alpha(y,x)$ is the probability of staying in x)

Algorithm:

- Set $t=0$, $y^0 = x_0$ = initial value
- for $t = 1, \dots, n_{rep}$:
  1. Sample $Y \sim q(y|x^{t-1})$
  2. Sample $U \sim \mathcal{U}([0,1])$
  3. If $U \le \alpha(x,y) \Longrightarrow X^t = Y$
     otherwise $\Longrightarrow X^t = X^{t-1}$

Theorem.
Let $\{x^{(t)}\}$ be a chain produced by a Metropolis-Hastings algorithm.

1. The kernel associated with the chain satisfies the "detailed balanced condition":
   $\exists f: \quad k(y,x)\,f(y) = k(x,y)\,f(x)$
2. f is the ergodic distribution of the chain

About the kernel:

$$k(y,x) = \alpha(y,x)\,q(y|x) + r(x)\,\delta(x)$$
$$r(x) = 1 - \int \alpha(y,x)\,q(y|x)\,dy$$

Remark 1. We can use this strategy to sample from the posterior distribution

$$f(\theta) = \pi(\theta|x_1, \dots, x_n)$$

Remark 2. It works also up to a constant:

$$\frac{f(y)\,q(x|y)}{f(x)\,q(y|x)} = \frac{\cancel{M} f^*(y)\,q(x|y)}{\cancel{M} f^*(x)\,q(y|x)}$$

f^* = not normalized function

= we can sample from not normalized functions
(In fact recall that, instead of computing exactly the posterior distribution we just have:

$$\pi(\theta|\underline{x}) \propto L(\theta,\underline{x})\,\pi(\theta) \quad \leftarrow \text{not normalized})$$

We can tune the algorithm choosing different $q(y|x)$.
For instance:

- Independent Metropolis-Hastings : $q(y|x) \overset{d}{=} q(y) \quad (\perp\!\!\!\perp x)$
- Random-Walk Metropolis-Hastings : $q(y|x) \overset{d}{=} q(y-x)$

Example 1.

Suppose to have $Y_1, \dots, Y_n | \mu, \lambda \overset{iid}{\sim} \text{Cauchy}(\mu, \lambda)$:

$$f_Y(y) = \frac{e^{-\lambda}}{\pi\left(1+(y-\mu)^2 e^{-2\lambda}\right)} \mathbb{1}_{(-\infty,\infty)}(y) \qquad \lambda, \mu \in \mathbb{R}$$

We want to make posterior inference on μ and λ.
We assume as prior:

$$\pi(\mu, \lambda) = N(\mu, m_1 \sigma_1^2) \cdot N(\lambda, m_2 \sigma_2^2)$$

The Cauchy is "the bad guy" because: $\nexists\, \mathbb{E}[Y]$, $\nexists\, \text{Var}(Y)$.
Posterior distribution:

$$\pi(\mu, \lambda | Y_1, \dots, Y_n) \propto \prod_{i=1}^n f_Y(y_i)\, \pi(\lambda)\, \pi(\lambda)$$

$$\propto \frac{e^{-n\lambda}}{\pi^n \cdot \prod_{i=1}^n \left(1+(y_i-\mu)^2 e^{-2\lambda}\right)} \pi(\mu)\, \pi(\lambda)$$

We define a MH algorithm using as proposal:

$$q(\mu^*, \lambda^* | \mu^{t-1}, \lambda^{t-1}) = N_2\left(\begin{pmatrix}\mu^{t-1}\\ \lambda^{t-1}\end{pmatrix}, \Sigma\right)$$

$$\alpha(\mu^*, \lambda^*, \mu^{t-1}, \lambda^{t-1}) = \frac{\pi(\mu^*, \lambda^* | \underline{Y})\, q(\mu^{t-1}, \lambda^{t-1} | \lambda^*, \mu^*)}{\pi(\mu^{t-1}, \lambda^{t-1} | \underline{Y})\, q(\mu^*, \lambda^* | \mu^{t-1}, \lambda^{t-1})}$$

In case of random walk Metropolis Hastings we can cancel this because we have a symmetric distribution

$$\log(\alpha(-,-)) = \log(\pi(\mu^*, \lambda^* | \underline{Y})) - \log(\pi(\mu^{t-1}, \lambda^{t-1} | \underline{Y}))$$

$$\propto \sum_{i=1}^n \log f(y_i | \mu^*, \lambda^*) + \log \pi(\mu^*) + \log \pi(\lambda^*) +$$

$$- \sum_{i=1}^n \log f(y_i | \mu^{t-1}, \lambda^{t-1}) - \log \pi(\mu^{t-1}) - \log \pi(\lambda^{t-1})$$

Example 2.

Sample of a mixture of 2 gaussian distributions.

$$f(x | \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \beta) \overset{d}{=} \beta N(\mu_1, \sigma_1^2) + (1-\beta) N(\mu_2, \sigma_2^2)$$

We want to use as proposal:

$$q(y | x^{t-1}) \overset{d}{=} N(y; x^{t-1}, \sigma_q^2)$$

We'll consider: $\beta = 0.2$, $\mu_1 = -2.5$, $\mu_2 = 2.5$, $\sigma_1^2 = \sigma_2^2 = 1$

ADAPTIVE METROPOLIS HASTINGS

Instead of considering a proposal: $q(y|x)$
we consider a family of proposals: $\{q_\gamma(y|x)\}_{\gamma \in \Gamma}$.

In this way the proposal may change during the simulation

Associated with the family of proposals we have a family of transition kernels:
$\{k_\gamma(y;x)\}_{\gamma \in \Gamma}$

Theorem.
An adaptive scheme on $\{k_\gamma\}_{\gamma \in \Gamma}$ will converge in total variation, i.e.

$$\lim_{n \to \infty} \| X(x) - f(x) \|_{TV} = 0$$

where $X(x)$ is the distribution of the chain, if:

1. $f(x)$ is the stationary distribution for all $\gamma \in \Gamma$
2. diminishing adaptation:
$$\lim_{m \to \infty} \sup_x \| k_{\gamma_m}(x) - k_{\gamma_{m+1}}(x) \|_{TV} = 0$$
3. $\forall x_n, \forall k_\gamma$ we reach the stationary distribution in a bounded number of steps M
(we don't have degenerate situations)

Remark: The optimal acceptance ratio for a MH algorithm is in $(0.25, 0.50)$

Example.
We want to sample from a multivariate t-student.

$$f_X(\underline{x}) = \frac{\Gamma\left(\frac{a+p}{2}\right)}{\Gamma\left(\frac{a}{2}\right) a^{p/2} \pi^{p/2} |\Sigma|^{1/2}} \left(1 + (\underline{x}-\underline{\mu}) \Sigma^{-1} (\underline{x}-\underline{\mu})\right)^{-\left(\frac{a+p}{2}\right)}$$

p = dimension
a = degree of freedom
$\underline{\mu}$ = location parameter
Σ = scale parameter

As proposal we choose:

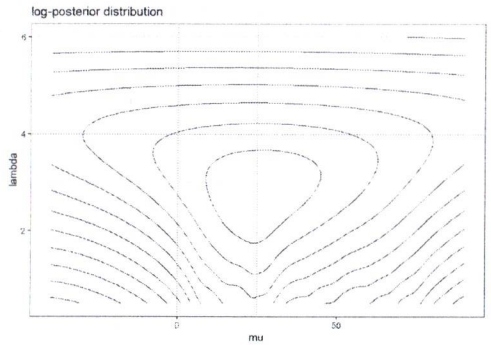
$$\underline{Y}|\underline{X} = \underline{X} + (1-\beta) N\left(\underline{0}, \frac{(2.38)^2}{p} \Sigma\right) + \beta N\left(\underline{0}, (0.1)^2 \frac{I}{d}\right)$$

where Σ = variance of the target distribution
(we actually replace Σ with $\hat{\Sigma}$, which is an estimation from the sample that we produce (at each step we update $\hat{\Sigma}$))

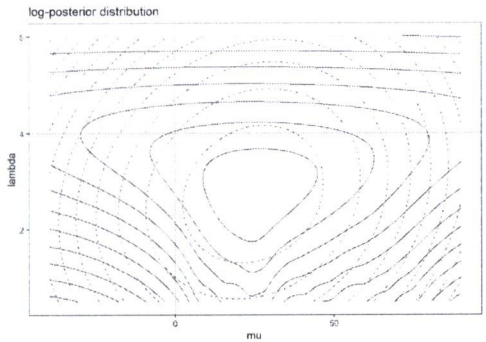
```
library(LearnBayes)
library(mvtnorm)
library(coda)
library(ggplot2)
library(ggpubr)

##########################################
### METROPOLIS-HASTINGS ALGORITHM     #####
##########################################
### CAUCHY DISTRIBUTION               #####
##########################################

data(darwin)
data = darwin$difference
data
```

```
##  [1] -67 -48   6   8  14  16  23  24  28  29  41  49  67  60  75
```

```
#-----------------------------------------
# prior distribution: product of two independent gaussian distribution
# specify the hyperparameters

mu1 <- 0
sig1 <- 100
mu2 <- 0
sig2 <- sqrt(10)

#-----------------------------------------
# log-posterior function

cauchy_log_post <- function(theta, y, mu1, sig1, mu2, sig2){
  mu <- theta[1]
  lambda <- theta[2]
  n <- length(y)
  out <- 0

  # log-likelihood for each obs
  for(i in 1:n){
    out <- out - lambda - log(1 + exp(-2*lambda) * (y[i]-mu)^2)
  }

  # Add the prior in log scale
  out <- out + dnorm(mu, mean = mu1, sd = sig1, log = T)
  out <- out + dnorm(lambda, mean = mu2, sd = sig2, log = T)

  # return the log-posterior
  return(out)
}

#-----------------------------------------
# plot the log-posterior

grid_mu  <- seq(-40, 90, length = 100)
grid_lam <- seq(0.5, 6, length = 100)
expanded_grid <- as.matrix(expand.grid(grid_mu, grid_lam))

log_post <- c()
for(i in 1:nrow(expanded_grid)){
    log_post[i] <- cauchy_log_post(theta = expanded_grid[i,], y = data,
                                   mu1 = mu1, sig1 = sig1, mu2 = mu2, sig2 = sig2)
}


plot_df <- data.frame(mu = expanded_grid[,1], lam = expanded_grid[,2], lpost = log_post)
ggplot(plot_df, mapping = aes(x = mu, y = lam, z = lpost)) +
  geom_contour() +
  theme_bw() +
  xlab("mu") +
  ylab("lambda") +
  ggtitle("log-posterior distribution")
```




```
#-----------------------------------------
# PROPOSAL: we use a bivariate Gaussian distribution on (mu, lambda)
# smart way: first look to the maximum a posteriori
# using the optim function

optimal <- optim(par = c(mean(data), log(sd(data))),
                 fn = cauchy_log_post, y = data,
                 mu1 = mu1, sig1 = sig1, mu2 = mu2, sig2 = sig2,
                 hessian = T,
                 control = list(fnscale=-1))

# maximum a posterirori
MAP <- optimal$par

# the optim give back also an "estimate" of the Hessian matrix around the maximum
# Through this estimate we define the matrix of variance and covariance of the proposal.
# Indeed, the second derivative of the negative log-likelihood evaluated at the maximum likelihood
# estimates (MLE) is the observed Fisher information and the inverse of the Fisher information matrix
# is an estimator of the covariance matrix
Sigma <- - solve(optimal$hessian)

# PROPOSAL: Normal(X_current, Sigma)

# NOTE: we could also scale the variance and covariance matrix with a
# factor influencing how far I move in the next step.

# gamma = ...
# Sig = gamma*Sig

# Of course we expect that if \gamma is too big the next step will be far away (most likely)
# and therefore it will be accepted less easily. If \gamma is very small the move will be accepted
# with high probability since is close to the current value but I will explore slowly the state space!

#-----------------------------------------
# Visualize the proposal (fixing the position parameter equal to the MAP)

log_proposal <- c()
for(i in 1:nrow(expanded_grid)){
  log_proposal[i] <- dmvnorm(expanded_grid[i,], mean = MAP, sigma = Sigma, log = T)
}

plot_df_2 <- data.frame(mu = expanded_grid[,1], lam = expanded_grid[,2], lpost = log_proposal)
ggplot(plot_df, mapping = aes(x = mu, y = lam, z = lpost)) +
  geom_contour() +
  geom_contour(data = plot_df_2, col = 2, lty = 2) +
  theme_bw() +
  xlab("mu") +
  ylab("lambda") +
  ggtitle("log-posterior distribution")
```



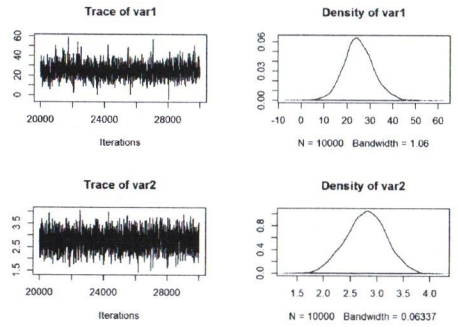
log posterior + proposal
(——) (- - - - -)

```
#-----------------------------------------
# Metropolis-Hastings Algorithm
# The algorithm produces a trajectory of
# the Markov chain with stationary distribution equal
# To the posterior of the model we are considering.
# In particular, we use a RANDOM WALK.
# The "noise" is normally distributed with the var-cov matrix
# equal to the inverse of the Hessian of the target distribution estimated
# Around its max.

cauchy_MH <- function(niter, burnin,
                      theta0, Sigma, mu1, mu2, sig1, sig2){
  # niter, burnin: iterations
  # theta0 initial state of the chain
  # Sig: proposal's var-cov matrix
  # mu1, mu2, sig2, sig2: hyperparameters

  # define the matrix that contains the output MCMC sample
  theta_output <- matrix(nrow = (niter-burnin), ncol = 2)
  # number of accepted moves
  nacp = 0

  pb <- txtProgressBar(min = 1, max = niter, initial = 1, style = 3)
  # Start from theta0
  for(i in 1:(niter))
  {
    theta_temp <- as.vector(rmvnorm(1, mean = theta0, sigma = Sigma)) # propose theta_temp: the mean is g
iven by the previous value

    ## Compute the logarithm of the probability of accepting the transition from th0 to theta_temp
    # First consider the log of accept/reject ratio. Numerator:
    lacp <- cauchy_log_post(th = theta_temp, y = data, mu1 = mu1, sig1 = sig1, mu2 = mu2, sig2 = sig2)
    # Denominator:
    lacp <- lacp - cauchy_log_post(th = theta0, y = data, mu1 = mu1, sig1 = sig1, mu2 = mu2, sig2 = sig2)
    lacp <- min(0, lacp)
    # Note: The proposal is symmetrical and therefore does not appear in the value of acceptance / reject
ion!
    lgu <- log(runif(1))

    ## if u < acp accept the move
    if(lgu < lacp)
    {
      theta0 <- theta_temp
      nacp = nacp + 1
    }
    # otherwise remain in the same state

    if(i > burnin )
    {
      theta_output[(i-burnin),] = theta0
    }
    setTxtProgressBar(pb, i)
  }
  close(pb)
  cat("Acceptance rate =", nacp/niter, "\n")
  return(theta_output)
}

#-------------------------------------
# RUN
# fix the seed
set.seed(42)

# fix the parameters of the Metropolis
niter = 30000
burnin = 20000

# At the end the chain will contain
# (niter-burnin) = 5000 observations
theta0 = MAP
theta_post <- cauchy_MH(niter = niter, burnin = burnin, theta0 = theta0, Sigma = Sigma,
                        mu1 = mu1, mu2 = mu2, sig1 = sig1, sig2 = sig2)
```

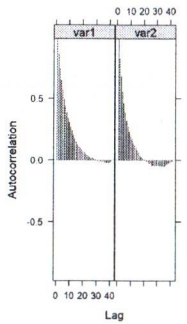
```
# CODA
mcmc_theta_post <- mcmc(theta_post, start = burnin + 1, end = niter)
summary(mcmc_theta_post)
```

```
##
## Iterations = 20001:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##         Mean     SD Naive SE Time-series SE
## [1,] 25.143 6.7532 0.067532         0.2577
## [2,]  2.794 0.3772 0.003772         0.0122
##
## 2. Quantiles for each variable:
##
##        2.5%    25%    50%    75%  97.5%
## var1 12.526 20.833 24.782 29.288 39.387
## var2  2.036  2.542  2.803  3.055  3.519
```

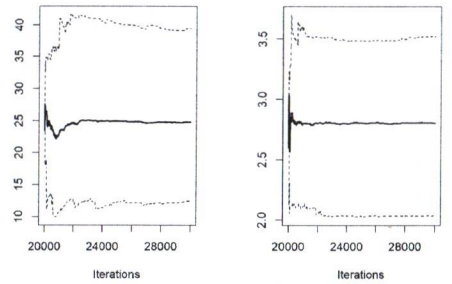
```
#  The "naive" standard error is the standard error of the mean,
#  which captures simulation error of the mean rather than posterior
#  uncertainty.
#
#    naiveSE = posteriorSD/sqrt(n)
#
#  The time-series standard error adjust the "naive" standard error for
#  autocorrelation. ---> more precise estimate

plot(mcmc_theta_post)
```




```
acfplot(mcmc_theta_post)
```




```
cumuplot(mcmc_theta_post)
```



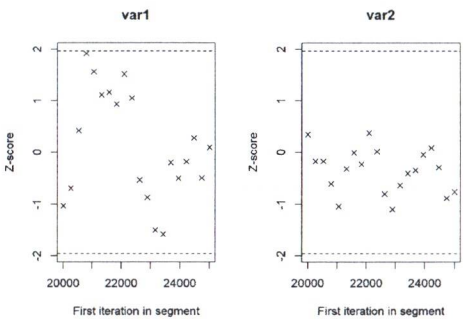

```
# in cumuplot, the dashed lines correspond to the quantiles
# 2.5% and 97.5%. The thick central lines represents the median of the chain
# all up to the i-th iteration i=burnin+1...niter

effectiveSize(mcmc_theta_post)
```

```
##     var1     var2
## 686.5122 956.7678
```

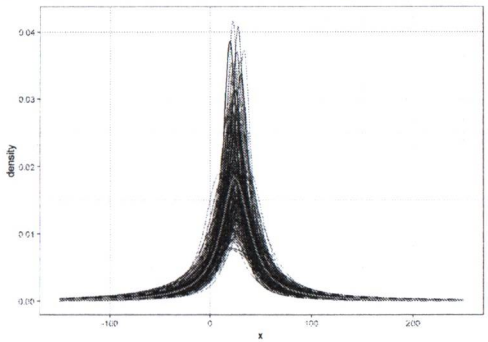
```
# Geweke (1992) proposed a convergence diagnostic for Markov chains
# based on a test for equality of the means of the first and last
# part of a Markov chain (by default the first 10% and the last
# 50%).  If the samples are drawn from the stationary distribution
# of the chain, the two means are equal and Geweke's statistic has
# an asymptotically standard normal distribution.

geweke.diag(mcmc_theta_post)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    var1    var2
## -1.0324  0.3482
```

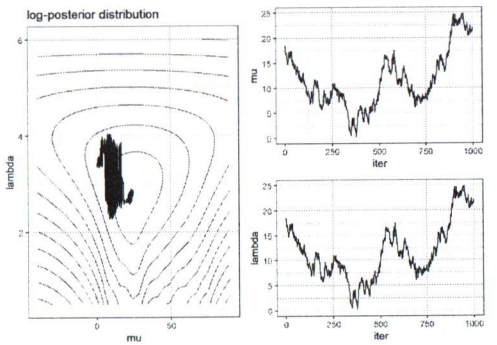
```
geweke.plot(mcmc_theta_post, frac1 = 0.1, frac2 = 0.5, nbins = 20)
```



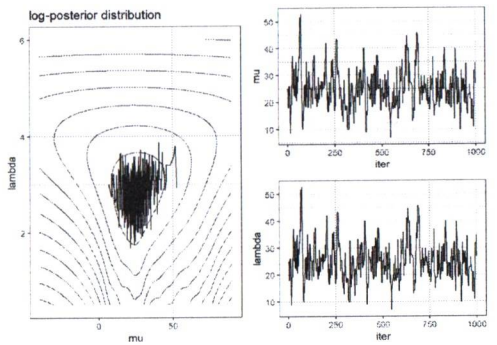
```
# Note: If the chain has converged, the majority of points should fall within
# 2 standard deviations of zero. This plot shows
# what happens to Geweke's Z-score when successively larger numbers
# of iterations are discarded from the beginning of the chain. To
# preserve the asymptotic conditions required for Geweke's
# diagnostic, the plot never discards more than half the chain.

#-----------------------------------
# predictive distribution
# p(X_(n+1)|X_1, X_2, ..., X_n):
# It is the integral of (likelihood * posterior) d\theta.
# ---------> Monte Carlo integration

mu_post <- theta_post[,1]
lam_post <- theta_post[,2]

S <- length(lam_post) # number of final iterations
grid_x = seq(-150, 250, length = 500) # grid on which I want to evaluate the predictive
Dens_pred_matrix <- matrix(0, ncol = 500, nrow = S) # Matrix that will contain the values

# Scale parameter:
sig_post = exp(lam_post)
# Column by column
for(i in 1:500){
  z <- (grid_x[i] - mu_post)/sig_post
  # Density in the logarithmic scale
  Dens_pred_matrix[,i] <- -log(pi) - lam_post - log(1+z^2)
}
Dens_pred_matrix <- exp(Dens_pred_matrix)

# plot
p <- ggplot() +
  geom_line(data = data.frame(x = grid_x, y = Dens_pred_matrix[1,]),
            mapping = aes(x = x, y = y), alpha = 0.3) +
  theme_bw() +
  ylab("density")

# just the first 250
for(i in 2:250){
  p <- p + geom_line(data = data.frame(x = grid_x, y = Dens_pred_matrix[i,]),
                mapping = aes(x = x, y = y), alpha = 0.3)
}

# posterior mean of the density and quantiles
Dens_pred <- apply(Dens_pred_matrix, 2, mean)
Dens_pred_quant <- apply(Dens_pred_matrix, 2, quantile, prob = c(0.05,0.95))

# add the lasts on the plot
p <- p + geom_line(data = data.frame(x = grid_x, y = Dens_pred),
                   mapping = aes(x = x, y = y), col = 3, lwd = 1) +
  geom_line(data = data.frame(x = grid_x, y = Dens_pred_quant[1,]),
            mapping = aes(x = x, y = y), col = 3, lty = 2, lwd = 1) +
  geom_line(data = data.frame(x = grid_x, y = Dens_pred_quant[2,]),
            mapping = aes(x = x, y = y), col = 3, lty = 2, lwd = 1)
p
```



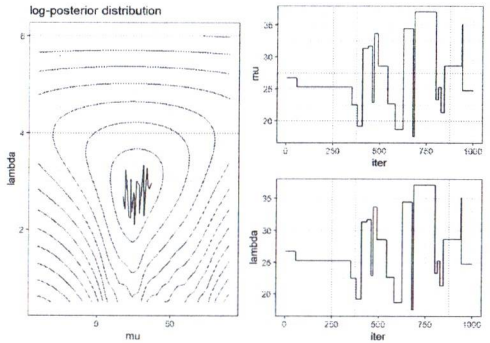

```
#-----------------------------------
# check for different constants gamma
# fix the seed

gamma_vec = c(0.01, 1, 100)
niter = 11000
burnin = 10000

for(l in 1:length(gamma_vec)){
  gamma = gamma_vec[l]
  cat("--------- gamma = ", gamma, "-------------\n")
  theta_post_temp <- cauchy_MH(niter = niter, burnin = burnin, theta0 = theta0, Sigma = gamma * Sigma,
                               mu1 = mu1, mu2 = mu2, sig1 = sig1, sig2 = sig2)

  p1 <- ggplot() +
    geom_contour(data = plot_df, mapping = aes(x = mu, y = lam, z = lpost)) +
    geom_line(data = as.data.frame(theta_post_temp), mapping = aes(x = V1, y = V2)) +
    theme_bw() +
    xlab("mu") +
    ylab("lambda") +
    ggtitle("log-posterior distribution")
  p2 <- ggplot()+
    geom_line(data = data.frame(x = 1:(niter - burnin), y = theta_post_temp[,1]),
              mapping = aes(x = x, y = y)) +
    theme_bw() +
    xlab("iter") +
    ylab("mu")
  p3 <- ggplot()+
    geom_line(data = data.frame(x = 1:(niter - burnin), y = theta_post_temp[,1]),
              mapping = aes(x = x, y = y)) +
    theme_bw() +
    xlab("iter") +
    ylab("lambda")
  print(ggarrange(p1, ggarrange(p2, p3, nrow = 2), ncol = 2))

  Sys.sleep(10)
}
```



gamma = 0.01

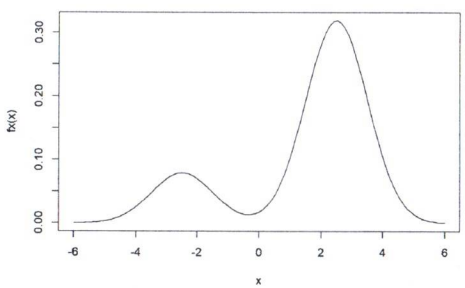


gamma = 1



gamma = 100

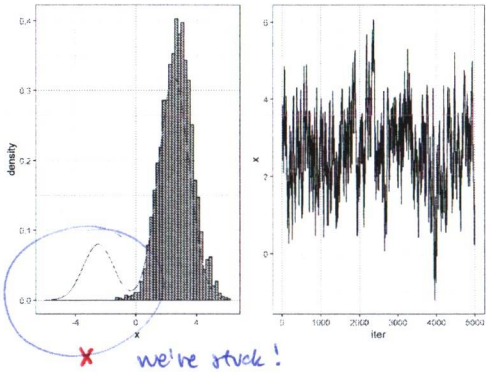
```
##########################################
### METROPOLIS-HASTINGS ALGORITHM      #####
##########################################
### BIMODAL DISTRIBUTION               #####
##########################################

fx <- function(x){
  # mixture of two normal distributions
  return(0.2 * dnorm(x, -2.5, 1) + 0.8 * dnorm(x, 2.5, 1))
}
curve(fx, xlim = c(-6, 6))
```




```
#-----------------------------------
# MH algo with N(x_old, s2) proposal

bimodalN_MH <- function(niter, burnin,
                        x0, s2){
  # initialize the algo
  x_output <- c()
  nacp = 0
  pb <- txtProgressBar(min = 1, max = niter, initial = 1, style = 3)

  for(i in 1:(niter))
  {
    # propose a new value
    x_temp <- rnorm(1, mean = x0, sd = sqrt(s2))
    # compute the acceptance ratio
    acc_ratio <- min(1, fx(x_temp) / fx(x0))

    if(runif(1) <= acc_ratio){
      # if accepted update x0 and nacp
      x0 <- x_temp
      nacp = nacp + 1
    }

    # if the burnin phase is ended
    if(i > burnin){
      # return the value
      x_output <- c(x_output, x0)
    }

    setTxtProgressBar(pb, i)
  }
  close(pb)
  cat("Acceptance rate =", nacp/niter, "\n")
  return(x_output)
}

#-------------------------------------------
# plots

set.seed(42)
curve_df <- data.frame(x = seq(-6, 6, length.out = 1000), y = fx(seq(-6, 6, length.out = 1000)))
plot_df <- data.frame(x = bimodalN_MH(15000, 10000, 0, 0.1))
```

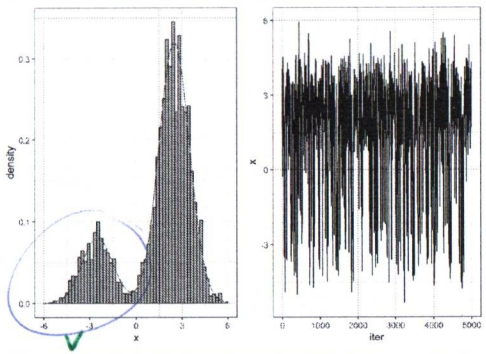
```
plot_df2 <- data.frame(x = bimodalN_MH(15000, 10000, 0, 6))
```

```
p1 <- ggplot() +
  geom_histogram(data = plot_df, mapping = aes(x, stat(density)), alpha = 0.3, col = 1, bins = 60) +
  geom_line(data = curve_df, mapping = aes(x = x, y = y), col = 2)+
  theme_bw()
p2 <- ggplot()+
  geom_line(data = plot_df, mapping = aes(x = 1:5000, y = x)) +
  theme_bw() +
  xlab("iter") +
  ylab("x")
ggarrange(p1, p2, ncol = 2)
```

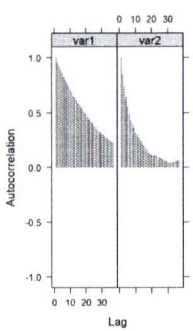


X we're stuck !

```
p3 <- ggplot() +
  geom_histogram(data = plot_df2, mapping = aes(x, stat(density)), alpha = 0.3, col = 1, bins = 60) +
  geom_line(data = curve_df, mapping = aes(x = x, y = y), col = 2)+
  theme_bw()
p4 <- ggplot()+
  geom_line(data = plot_df2, mapping = aes(x = 1:5000, y = x)) +
  theme_bw() +
  xlab("iter") +
  ylab("x")
ggarrange(p3, p4, ncol = 2)
```




```
#-----------------------------------
# in coda

coda_obj <- mcmc(cbind(plot_df$x, plot_df2$x))
acfplot(coda_obj)
```



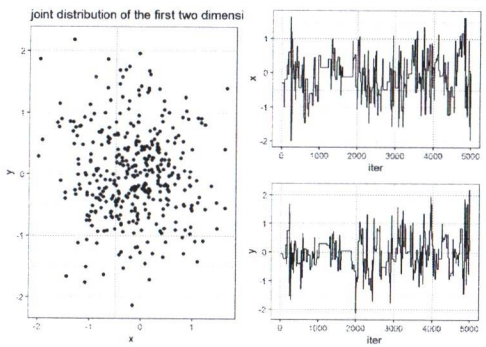

```
effectiveSize(coda_obj)
```

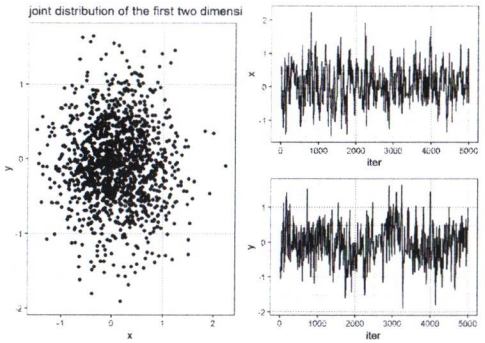
```
##     var1     var2
##  93.1467 308.6322
```

```
################################################
### ADAPTIVE METROPOLIS-HASTINGS ALGORITHM      #####
################################################

# proportional to a multivariate t-student
f_multi <- function(x, m, S, gdl){
  (1 + t(x - m) %*% solve(S) %*% (x - m) / gdl)^(- (gdl + length(x))/2)
}

# we want to simulate from a multivariate distribution,
# optimizing the acceptance rate
# we know (R&R) that in a multivariate case,
# an good choice for the variance of a Gaussian RW is 2.38^2 * Sigma / d
# but usually we don't know Sigma

adapt_MH <- function(niter, burnin, m, S, gdl, x0, Sigma = NULL, beta = 0.05, adapt = T){

  # initialize the algo
  d <- length(x0)
  x_res_temp <- matrix(x0, ncol = d)
  nacp = 0
  pb <- txtProgressBar(min = 1, max = niter, initial = 1, style = 3)

  for(i in 1:(niter))
  {
    # propose a new value
    if(adapt == F){
      x_temp <- as.vector(rmvnorm(1, mean = x0, sigma = Sigma))
    } else {
      if(i <= 2*d){
        Sigma_temp <- (0.1)^2 * diag(1, d) / d
      } else {
        Sigma_temp <- (1 - beta)^2 * (2.38)^2 / d * var(x_res_temp) + beta^2 * (0.1)^2 * rep(1, d) / d
      }
      x_temp <- as.vector(rmvnorm(1, mean = x0, sigma = Sigma_temp))
    }

    # compute the acceptance ratio
    acc_ratio <- min(1, f_multi(x_temp, m, S, gdl) / f_multi(x0, m, S, gdl))
    if(runif(1) <= acc_ratio){
      # if accepted update x0 and nacp
      x0 <- x_temp
      nacp = nacp + 1
    }
    x_res_temp <- rbind(x_res_temp, x0)

    setTxtProgressBar(pb, i)
  }
  close(pb)
  cat("Acceptance rate =", nacp/niter, "\n")
  return(x_res_temp[-c(1:(burnin+1)), ])
}

niter = 15000
burnin = 10000
sample_no_adapt <- adapt_MH(niter, burnin, m = rep(0, 10), S = diag(0.25, 10),
                            gdl = 15, x0 = rep(0, 10), Sigma = diag(0.5, 10), adapt = F)
```

```
sample_adapt <- adapt_MH(niter, burnin, m = rep(0, 10), S = diag(0.254, 10),
                         gdl = 15, x0 = rep(0, 10), adapt = T)
```

```r
# not adaptive
plot_df_1 <- data.frame(x = sample_no_adapt[,1], y = sample_no_adapt[,2], iter = 1:(niter - burnin))
p1_no <- ggplot() +
  geom_point(data = plot_df_1, mapping = aes(x = x, y = y)) +
  theme_bw() +
  xlab("x") +
  ylab("y") +
  ggtitle("joint distribution of the first two dimensions")
p2_no <- ggplot()+
  geom_line(data = plot_df_1, mapping = aes(x = iter, y = x)) +
  theme_bw() +
  xlab("iter") +
  ylab("x")
p3_no <- ggplot()+
  geom_line(data = plot_df_1, mapping = aes(x = iter, y = y)) +
  theme_bw() +
  xlab("iter") +
  ylab("y")
ggarrange(p1_no, ggarrange(p2_no, p3_no, nrow = 2), ncol = 2)
```






```r
# adaptive
plot_df_2 <- data.frame(x = sample_adapt[,1], y = sample_adapt[,2], iter = 1:(niter - burnin))
p1_y <- ggplot() +
  geom_point(data = plot_df_2, mapping = aes(x = x, y = y)) +
  theme_bw() +
  xlab("x") +
  ylab("y") +
  ggtitle("joint distribution of the first two dimensions")
p2_y <- ggplot()+
  geom_line(data = plot_df_2, mapping = aes(x = iter, y = x)) +
  theme_bw() +
  xlab("iter") +
  ylab("x")
p3_y <- ggplot()+
  geom_line(data = plot_df_2, mapping = aes(x = iter, y = y)) +
  theme_bw() +
  xlab("iter") +
  ylab("y")
ggarrange(p1_y, ggarrange(p2_y, p3_y, nrow = 2), ncol = 2)
```

GIBBS SAMPLER

It works with multivariate distributions.
We consider a simple case: $d = 2$.
We want to sample from $f(x,y)$.
Instead of sampling from $f(x,y)$ we sample from $f(x|y)$ and $f(y|x)$.

Algorithm:

- initialize x_0
- for $i = 1, \dots, n_{rep}$:
  1. Sample $Y_i \sim f(y|x_{i-1})$
  2. Sample $X_i \sim f(x|y_i)$

The Gibbs sampler is a special case of the Metropolis-Hastings:
consider the case where we're moving from x to x^*:
the proposal distribution is $f(x^*|y)$,
the acceptance ratio is:

$$\alpha = \min\left\{1, \frac{f(x^*,y)}{f(x,y)} \frac{f(x|y)}{f(x^*|y)}\right\}$$

$$= \min\left\{1, \frac{f(x^*|y)}{f(x|y)} \frac{f(y)}{f(y)} \frac{f(x|y)}{f(x^*|y)}\right\} = 1$$

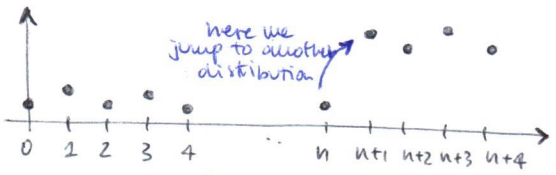
The Gibbs sampler is a Metropolis-Hastings where we accept all the proposed values ($\Rightarrow$ we have all the properties of the Metropolis-Hastings)

Special cases of the Gibbs sampler:

1. Full Gibbs sampler: we sample $X_i | \underline{X}_{i-1} \quad \forall i$
   (for all the dimensions we sample one dimension given the others)
2. Blocked Gibbs sampler: we work with a subset of the quantity of interest:
   if $\underline{X}$ is the quantity of interest we work with $X_1, X_2 | X_3, \dots, X_d$
3. Collapsed Gibbs sampler: we can marginalize out some dimensions: $X_1 | X_3, \dots, X_d$

Example.
We have a sample $X_1, \dots, X_n, X_{n+1}, \dots, X_{n+m}$, $\quad X_i \in \{0,1,2,\dots\}$

$$\begin{cases} X_1, \dots, X_n | \lambda_1 \overset{iid}{\sim} P(\lambda_1) \\ X_{n+1}, \dots, X_{n+m} | \lambda_2 \overset{iid}{\sim} P(\lambda_2) \end{cases}$$



$$f(x|\lambda_j) = \frac{\lambda_j^x e^{-\lambda_j}}{x!} \mathbb{1}_{\{0,1,2,\dots\}}(x)$$

Prior on λ_j:

$$\pi(\lambda_j) = \frac{b^a}{\Gamma(a)} \lambda_j^{a-1} e^{-b\lambda_j} \qquad j = 1, 2$$

One more assumption: the change-distribution point is uniformly distributed between 1 and $n+m$ and so:

$$\mathbb{P}(n = j) = \frac{1}{n+m} \mathbb{1}_{\{1,2,\dots,n+m\}}(j)$$

We want to sample from:

$$\pi(\lambda_1, \lambda_2, n | \underline{X}) \propto L(\lambda_1, \lambda_2, n | \underline{X}) \pi(\lambda_1) \pi(\lambda_2) \pi(n)$$

To keep it simple we discard the quantities $\mathbb{1}_{\{\lambda_1, \lambda_2, n\}}$:

$$L(\lambda_1, \lambda_2, n \mid \underline{x}) = \prod_{i=1}^{n} f(x_i \mid \lambda_1) \prod_{i=1}^{m} f(x_{n+i} \mid \lambda_2)$$
$$\propto e^{-n\lambda_1} e^{-m\lambda_2} \lambda_1^{\sum_{i=1}^{n} x_i} \lambda_2^{\sum_{i=1}^{n} x_{n+i}}$$

And so the posterior is:

$$\pi(\lambda_1, \lambda_2, n \mid \underline{x}) \propto \left[e^{-n\lambda_1} e^{-m\lambda_2} \lambda_1^{\sum_{i=1}^{n} x_i} \lambda_2^{\sum_{i=1}^{m} x_{i+n}}\right]\left[\lambda_1^{a-1} \lambda_2^{a-1} e^{-b\lambda_1} e^{-b\lambda_2}\right]\left[\frac{1}{n+m} \mathbb{1}_{\{1,\dots,n+m\}}^{(n)}\right]$$

Let's prepare it for the Gibbs sampler:

$$\pi(\lambda_1 \mid -) \propto e^{-n\lambda_1} e^{-b\lambda_1} \lambda_1^{a-1} \lambda_1^{\sum_{i=1}^{n} x_i} = e^{-(n+b)\lambda_1} \lambda_1^{a+\sum_{i=1}^{n} x_i - 1}$$

$$\Longrightarrow \lambda_1 \mid - \sim \text{Gamma}\left(a + \textstyle\sum_{i=1}^{n} x_i,\ b+n\right)$$

$$\pi(\lambda_2 \mid -) \propto e^{-(m+b)\lambda_2} \lambda_2^{a+\sum_{i=1}^{m} x_{i+n} - 1}$$

$$\Longrightarrow \lambda_2 \mid - \sim \text{Gamma}\left(a + \textstyle\sum_{i=1}^{m} x_{i+n},\ b+m\right)$$

$$\mathbb{P}(n=j \mid -) \propto e^{-n\lambda_1} \lambda_1^{\sum_{i=1}^{n} x_i} \lambda_2^{\sum_{i=1}^{m} x_{i+n}}$$

still discrete, not uniform

AUGMENTED GIBBS SAMPLER

We want to sample from θ = parameter of interest.
We have:

$$\begin{cases} Y_1, \dots, Y_n \mid \theta \overset{iid}{\sim} f(y \mid \theta) \\ \theta \sim \pi(\theta) \end{cases}$$

We want to sample from $\pi(\theta \mid y)$.
We can write:

$$\pi(\theta \mid y) = \int_x \pi(\theta, x \mid y)\, dx$$

we augment with one dimension and we rewrite the same distribution as marginalizing out the augmented distribution

thanks to this notation we can sample by:

$$\begin{cases} \pi(\theta \mid x, y) \\ \pi(x \mid \theta, y) \end{cases}$$

Example.

Assume we collect a sample with some missing observations:

$$\begin{cases} \underline{Y}_1, \dots, \underline{Y}_n \mid \Sigma \sim N_p(\underline{0}, \Sigma) & \underline{Y} \in \mathbb{R}^p \\ \Sigma \sim \text{Inverse-Wishart}(\nu_0, \Psi_0) & \nu_0 > p-1,\ \Psi \in \mathbb{R}^{p \times p} \end{cases}$$

$$\Longrightarrow \Sigma \mid \underline{Y} \sim \text{Inverse Wishart}(\nu_0 + n,\ \Psi_0 + S), \quad S = \underline{Y}^T \underline{Y}$$

Suppose dimension = 2:

Y_1	1	1	-1	2	N/A	N/A
Y_2	1	-1	N/A	N/A	2	2

$$\underline{Y}^{\text{observed}} = \{Y_{11}, Y_{21}, Y_{31}, Y_{41}, Y_{12}, Y_{22}, Y_{52}, Y_{62}\}$$
$$\underline{Y}^{\text{missed}} = \{Y_{51}, Y_{61}, Y_{32}, Y_{42}\}$$

We're interested in sampling: $\pi(\Sigma \mid \underline{Y}^{\text{observed}}) = L(\Sigma \mid \underline{Y}^{\text{obs.}})\, \pi(\Sigma)$
However we don't know the distribution.
We can augment the problem: $\pi(\Sigma, \underline{Y}^{\text{missed}} \mid \underline{Y}^{\text{obs}})$.
Thanks to that the strategy becomes:

sample by: $\begin{cases} \pi(\Sigma \mid \underline{Y}^{\text{obs}}, \underline{Y}^{\text{missed}}) \\ \pi(\underline{Y}^{\text{missed}}, \Sigma, \underline{Y}^{\text{obs}}) \end{cases}$

we sample (through Gibbs Sampler) the parameter of interest and then we can update the missed value then again the param of interest and then again the missed value

Algorithm:

- Initialize Σ, $\underline{Y}^{miss}$
- for $i = 1, \dots, n_{rep}$:
  1. Sample from $\Sigma \mid \underline{Y}^{obs}, \underline{Y}^{miss} \sim IW(\lambda_0 + n, \Psi_0 + S)$
  2. Sample from $\underline{Y}^{miss} \mid \Sigma, \underline{Y}^{obs} \sim \prod_{i \in C_1} f_{Y_1|Y_2}(y_1|y_2,\Sigma) \prod_{i \in C_2} f_{Y_2|Y_1}(y_2|y_1,\Sigma)$

C_1: where we miss the 1st dimension; C_2: where we miss the 2nd dimension

Property of the Gaussian:

$$\underline{X} \sim N_2(\mu, \Sigma), \quad \Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{bmatrix}$$

$$\Rightarrow X_1 | X_2 = y \sim N\left(\mu_1 + \frac{\sigma_{12}}{\sigma_1^2}(y-\mu_2),\ \sigma_1^2 - \frac{\sigma_{12}^2}{\sigma_2^2}\right)$$

HAMILTONIAN MONTE CARLO

(sampler used in Stan)

— alternative to Metropolis-Hastings where instead of using a proposal distribution we use a system dinamics to propose a new value

We want to sample $\underline{X} \sim f(\underline{x})$, $\underline{X} \in \mathbb{R}^d$.

We can think about $\underline{X}$ as a position in a specific system.

We want to augment the problem with a latent augmented variable (that in terms of Hamiltonian dynamics plays the role of the momentum).

It's like we want to spin the position, check the trajectory w.r.t. a dynamics and use the new point as a new proposed point.

Problem augmented: $(\underline{X}, \underline{p})$ = (position, momentum):

1. Update $(\underline{X}, \underline{p}) \in \mathbb{R}^d \times \mathbb{R}^d$
2. Perform a Metropolis Hasting to accept a new value

The function describing the system is:

$H(\underline{x}, \underline{p})$ = Hamiltonian function

We update the trajectories by looking at the partial derivatives of $H(\underline{x}, \underline{p})$ w.r.t. the time:

$$\begin{cases} \frac{dx_i}{dt} = \frac{dH}{dp_i} & i = 1, \dots, d \\ \frac{dp_i}{dt} = -\frac{dH}{dx_i} & i = 1, \dots, d \end{cases}$$

We consider: $H(\underline{x}, \underline{p}) = U(\underline{x}) + K(\underline{p})$

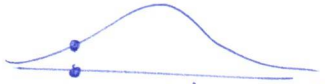
energy = potential energy + kinetic energy

We just consider: $K(\underline{p}) = \underline{p}^T M^{-1} \underline{p}$ — this proposal leads to a gaussian proposal to spin the point

$$\Rightarrow \begin{cases} \frac{dx_i}{dt} = [M^{-1}\underline{p}]_i & i = 1, \dots, d \\ \frac{dp_i}{dt} = \frac{\partial U}{\partial x_i} & i = 1, \dots, d \end{cases}$$

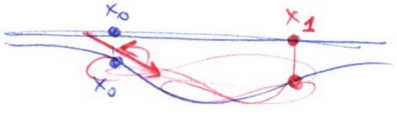
$\frac{\partial U}{\partial x_i}$ — distribution of interest

Idea:



x_0, we have a distribution and an initial point

→ We flip the distribution and following the potential energy and the spin we move to another point:



Sketch on how it works:

$$\begin{cases} p_i(t + \frac{\varepsilon}{2}) = p_i(t) - (\frac{\varepsilon}{2}) \frac{\partial U}{\partial x_i}(x_i(t)) \\ x_i(t+\varepsilon) = x_i(t) + \varepsilon [M^{-1} p(t + \frac{\varepsilon}{2})]_i \\ p_i(t+\varepsilon) = p_i(t + \frac{\varepsilon}{2}) - (\frac{\varepsilon}{2}) \frac{\partial U}{\partial x_i}(x_i(t+\varepsilon)) \end{cases}$$

ε = size of the step that we want to make

— To study the trajectory on the flipped distribution we use HAMILTONIAN DYNAMICS

How can we use it?

We start from some density that can be written as:

$$f_x(\underline{x}) = C \cdot e^{-E(\underline{x})}$$

We augment it with $\underline{p}$:

$$f_{xp}(\underline{x}, \underline{p}) = C \cdot e^{-H(\underline{x}, \underline{p})}$$

we assume: $H(\underline{x}, \underline{p}) = U(\underline{x}) + K(\underline{p})$:

$$f_{xp}(\underline{x}, \underline{p}) = C_1 e^{-U(\underline{x})} \cdot C_2 e^{-K(\underline{p})}$$

Algorithm:

- Initialize $\underline{x}_0, \varepsilon, L$
- for $i = 1, \dots, n_{rep}$:
  1. Sample $\underline{p}_0 \sim f_p(\underline{p}) = C_2 e^{-K(\underline{p})}$
  2. Set $\underline{x}_0^{temp} = \underline{x}_{i-1}$ (we initialize the system in the previous point)
  3. for $j = 1, \dots, L$ (where L = number of steps that we use to integrate the system)
     - 3.1. $\underline{p}_j = \underline{p}_{j-1} - \frac{\varepsilon}{2} \nabla U(\underline{x}_{j-1}^{temp})$
     - 3.2. $\underline{x}_j^{temp} = \underline{x}_{j-1}^{temp} + \varepsilon M^{-1} \underline{p}_j$
     - 3.3. $\underline{p}_j = \underline{p}_j - \frac{\varepsilon}{2} \nabla U(\underline{x}_j^{temp})$
  4. Set $\underline{p}_L = -\underline{p}_L$
  5. Perform an MH step from $\underline{x}_0^{temp}$ to $\underline{x}_L^{temp}$

We'll see an example where we want to sample from a highly correlated gaussian distribution:

$$\underline{X} \sim N(\underline{\mu}, \Sigma) : \quad \underline{\mu} = [2, 2]^T, \quad \Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$

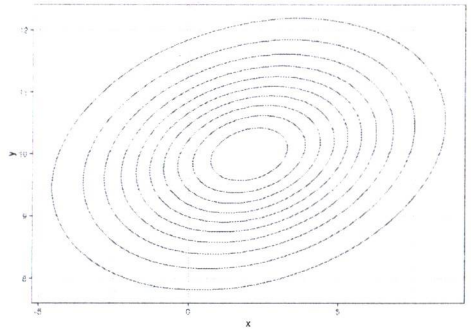
$$\Rightarrow f_x(\underline{x}) = C_2 e^{-U(\underline{x})} : \quad U(\underline{x}) = \frac{1}{2} (\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu})$$

$$\nabla U(\underline{x}) = \Sigma^{-1} (\underline{x} - \underline{\mu})$$

```
library(LearnBayes)
library(mvtnorm)
library(coda)
library(ggplot2)
library(ggpubr)

##########################################
### GIBBS SAMPLER ALGORITHM          #####
##########################################
### BIVARIATE
### GAUSSIAN DISTRIBUTION
##########################################

## Build an R function to sample from the distribution with parameters:
## - the vector of the mean   mu = (mu_1,mu_2)
## - the vector of the sd     sig = (sig_1,sig_2) = sqrt(sig_11,sig_22)
## - the correlation          rho = sig_12/sqrt(sig_11+sig_22)
##
## Parameters of the MCMC chain:
## - nr iterations:  niter
## - burnin
## - thinning
## - initial state of the chain

biv_normal_gibbs <- function(niter, burnin, thin, x0, mu, sig, rho)
{
  results <- matrix(0, nrow = round((niter - burnin) / thin), ncol = 2)
  x <- x0

  pb <- txtProgressBar(min = 1, max = niter, initial = 1, style = 3)
  for(j in 1:niter)
  {
    x[1] <- rnorm(1, mean = mu[1] + sig[1] / sig[2] * rho * (x[2] - mu[2]),
                  sd = sqrt(1 - rho^2) * sig[1])
    x[2] <- rnorm(1, mean = mu[2] + sig[2] / sig[1] * rho * (x[1] - mu[1]),
                  sd = sqrt(1 - rho^2) * sig[2])

    if(j > burnin & (j - burnin) %% thin==0)
    {
      results[(j - burnin) / thin, ] <- x
    }
    setTxtProgressBar(pb, j)
  }
  close(pb)
  return(results)
}

## set the parameters
mu <- c(2, 10)
sig <- c(3, 1)
rho <- 0.25

## var-cov matrix
Sig <- matrix(c(sig[1]^2, rho * sig[1] * sig[2],
                rho * sig[1] * sig[2], sig[2]^2), byrow = T, nrow = 2)
Sig
```

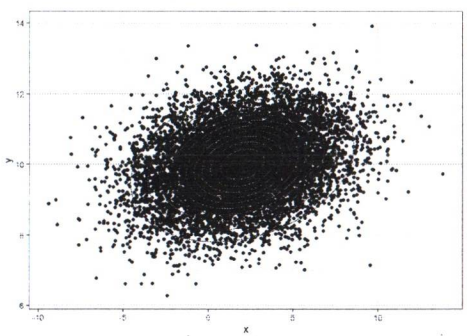
```
##      [,1] [,2]
## [1,] 9.00 0.75
## [2,] 0.75 1.00
```

```
# Visualize the target distribution
# Grids
gr_x1 <- seq(mu[1] - 3 * sig[1], mu[1] + 3 * sig[1], length = 100)
gr_x2 <- seq(mu[2] - 3 * sig[2], mu[2] + 3 * sig[2], length = 100)
ex_grid <- expand.grid(gr_x1, gr_x2)
dens <- apply(ex_grid, 1, function(x) dmvnorm(x, mean = mu, sigma = Sig))
plot_df <- data.frame(x = ex_grid[,1], y = ex_grid[,2], dens = dens)

ggplot(plot_df) +
  geom_contour(mapping = aes(x = x, y = y, z = dens)) +
  theme_bw()
```

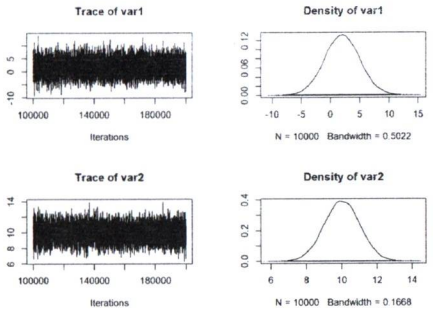


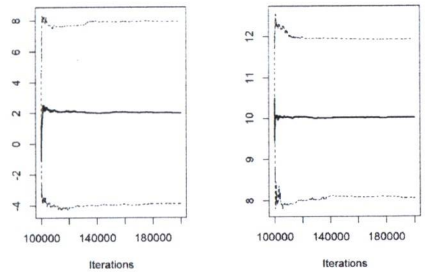

```
### ------------------------------------------------------------------
### RUN THE GIBBS SAMPLER
# set the parameters
niter <- 200000
burnin <- 100000
thin <- 10
x0 <- c(1, 1)

# simulation
set.seed(42)
X <- biv_normal_gibbs(niter = niter, burnin = burnin, thin = thin,
                      x0 = x0, mu = mu, sig = sig, rho = rho)
```

```
### ------------------------------------------------------------------
# plot the sampled values
ggplot(plot_df) +
  geom_point(data = data.frame(x = X[,1], y = X[,2]), mapping = aes(x = x, y = y)) +
  geom_contour(mapping = aes(x = x, y = y, z = dens)) +
  theme_bw()
```




```
### ------------------------------------------------------------------
### DIAGNOSTIC IN CODA
X_mc <- mcmc(data = X, start = burnin + 1, end = niter, thin = thin)
plot(X_mc)
```






```
summary(X_mc)
```

```
##
## Iterations = 100001:199991
## Thinning interval = 10
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean     SD Naive SE Time-series SE
## [1,]  1.969 2.9896 0.029896       0.029896
## [2,] 10.012 0.9927 0.009927       0.009927
##
## 2. Quantiles for each variable:
##
##        2.5%      25%    50%   75%  97.5%
## var1 -3.893 -0.05432  1.971  3.99  7.898
## var2  8.058  9.33829 10.019 10.69 11.945
```

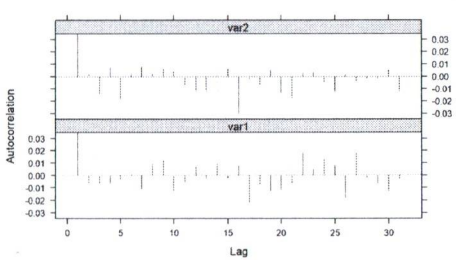
```
# Effective sample size:
effectiveSize(X_mc)
```

```
##  var1  var2
## 10000 10000
```

```
dim(X_mc)[1]
```

```
## [1] 10000
```

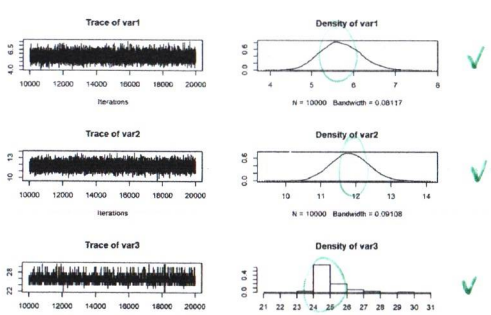
```
# Autocovariances
acfplot(X_mc, lag.max = 30)
```




```
# (0.025, 0.5, 0.975) quantiles
cumuplot(X_mc)
```

```
##########################################
### GIBBS SAMPLER ALGORITHM          #####
##########################################
### CHANGE-POINT MODEL
##########################################

set.seed(1)
x <- c(rpois(25, lambda = 5), rpois(40, lambda = 12))

sample_log_probs <- function(vals, probs){
  t_probs <- sapply(1:length(probs), function(x) 1 / sum(exp(probs - probs[x])) )
  sample(vals, size = 1, prob = t_probs)
}

change_gibbs <- function(niter, burnin, thin, data, lambda10, lambda20, n0, a, b)
{
  results <- matrix(0, nrow = round((niter - burnin) / thin), ncol = 3)
  temp <- c(lambda10, lambda20, n0)
  n_m <- length(data)

  pb <- txtProgressBar(min = 1, max = niter, initial = 1, style = 3)
  for(j in 1:niter)
  {
    m <- n_m - temp[3]
    temp[1] <- rgamma(1, a + sum(data[1:temp[3]]), b + temp[3])
    temp[2] <- rgamma(1, a + sum(data[(temp[3] + 1):n_m]), b + m)
    temp_log_probs <- sapply(2:(n_m - 1),
                             function(x) sum(data[1:x]) * log(temp[1]) - x * temp[1] +
                               sum(data[(x+1):n_m]) * log(temp[2]) - (n_m - x) * temp[2])
    temp[3] <- sample_log_probs(2:(n_m - 1), prob = temp_log_probs)

    if(j > burnin & (j - burnin) %% thin==0)
    {
      results[(j - burnin) / thin, ] <- temp
    }
    setTxtProgressBar(pb, j)
  }
  close(pb)
  return(results)
}

### ----------------------------------------------------------------------------
# RUN
niter <- 20000
burnin <- 10000
thin <- 1

sample_change <- change_gibbs(niter = niter, burnin = burnin, thin = thin, data = x,
                              lambda10 = 1, lambda20 = 1, n0 = 20, a = 10, b = 1)
```

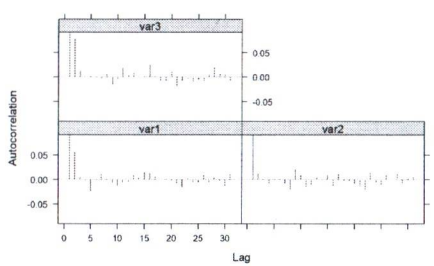
```
### ----------------------------------------------------------------------------
### DIAGNOSTIC IN CODA
change_mc <- mcmc(data = sample_change, start = burnin + 1, end = niter, thin = thin)
plot(change_mc)
```



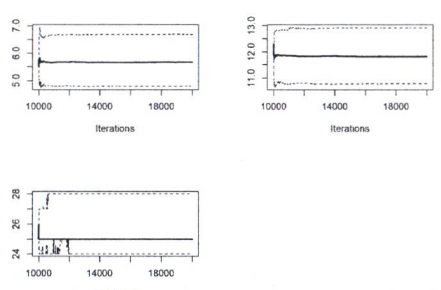

```
summary(change_mc)
```

```
##
## Iterations = 10001:20000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean     SD Naive SE Time-series SE
## [1,]  5.687 0.4832 0.004832       0.005104
## [2,] 11.823 0.5421 0.005421       0.005421
## [3,] 25.422 0.8649 0.008649       0.009369
##
## 2. Quantiles for each variable:
##
##        2.5%   25%    50%    75%  97.5%
## var1  4.798  5.35  5.663  6.009  6.686
## var2 10.789 11.45 11.814 12.186 12.910
## var3 24.000 25.00 25.000 26.000 28.000
```

```
# Autocovariances
acfplot(change_mc, lag.max = 30)
```




```
# (0.025, 0.5, 0.975) quantiles
cumuplot(change_mc)
```




```
# Effective sample size:
effectiveSize(change_mc)
```

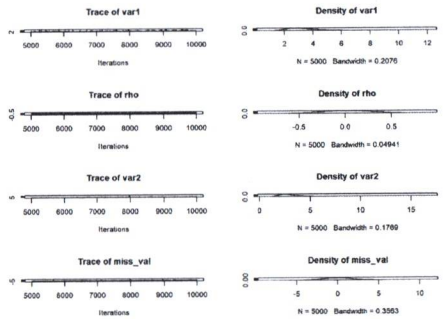
```
##      var1      var2      var3
##  8962.611 10000.000  8522.428
```

```
dim(change_mc)[1]
```

```
## [1] 10000
```

```
##########################################
### GIBBS SAMPLER ALGORITHM          #####
##########################################
### MISSING DATA
##########################################

gibbs_missing <- function(niter, burnin, thin, data, Sigma0, n0, Psi0){

  y_aug <- data
  Sigma <- Sigma0

  # about the missing data
  miss <- which(is.na(y_aug), arr.ind = T)
  miss_rows <- miss[,1]
  miss_cols <- miss[,2]
  n_mis <- length(miss_rows)

  # results
  rho <- var1 <- var2 <- miss_val <- vector(length = (niter - burnin) / thin)

  pb <- txtProgressBar(min = 1, max = niter, initial = 1, style = 3)
  for(iter in 1:niter)
  {
    ## Step 1: simulate the missing data
    for(i in 1:n_mis)
    {
      if(miss_cols[i] == 1)
      {
        mu1 <- Sigma[1,2] / Sigma[2,2] * y_aug[miss_rows[i], 2]
        s2_1 <- Sigma[1, 1] - Sigma[1, 2]^2 / Sigma[2, 2]
        y_aug[miss_rows[i], 1] <- rnorm(1, mean = mu1, sd = sqrt(s2_1))
      }
      else
      {
        mu2 <- Sigma[2, 1] / Sigma[1, 1] * y_aug[miss_rows[i], 1]
        s2_2 <- Sigma[2, 2] - Sigma[2, 1]^2 / Sigma[1, 1]
        y_aug[miss_rows[i], 2] <- rnorm(1, mean = mu2, sd = sqrt(s2_2))
      }
    }

    # Compute S
    S <- t(y_aug) %*% y_aug

    # Sample Sigma
    Sigma <- solve(rWishart(1, df = n + n0, Sigma = solve(S + Psi0))[,,1])

    if(iter>burnin & (iter-burnin) %% thin == 0)
    {
      ## Compute the correlation parameter at each iteration
      rho[(iter-burnin) / thin] <- (Sigma[1,2] / sqrt(Sigma[1,1] * Sigma[2,2]))
      var1[(iter-burnin) / thin] <- Sigma[1,1]
      var2[(iter-burnin) / thin] <- Sigma[2,2]
      miss_val[(iter-burnin) / thin] <- y_aug[20, 2]
    }
    setTxtProgressBar(pb, value = iter)
  }
  close(pb)
  return(cbind(var1, rho, var2, miss_val))
}

### ---------------------------------------------------------------------
# RUN THE ALGORITHM

set.seed(42)
y <- rmvnorm(20, c(0,0), sigma = diag(2,2))
y[12:16, 1] <- y[17:20, 2] <- NA
n <- nrow(y)
colnames(y) <- c("y1", "y2")

n0 <- 4
Sigma0 <- Psi0 <- 2* diag(2)
niter <- 10000
burnin <- 5000
thin <- 1

sample_missing <- gibbs_missing(niter = niter, burnin = burnin, thin = thin,
                                data = y, Sigma0 = Sigma0, n0 = n0, Psi0 = Psi0)
```

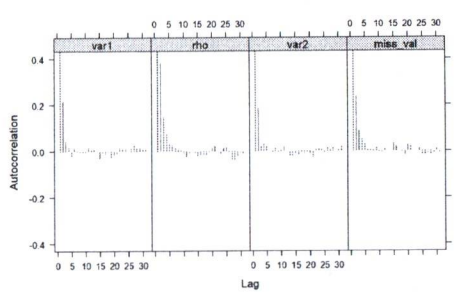
```
### ---------------------------------------------------------------------
# diagnostic in CODA

missing_mc <- mcmc(sample_missing, start = burnin + 1, end = niter, thin = thin)
plot(missing_mc)
```




```
summary(missing_mc)
```

```
##
## Iterations = 5001:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean    SD Naive SE Time-series SE
## var1      3.35131 1.226 0.017333        0.02161
## rho       0.01964 0.256 0.003621        0.00540
## var2      2.97991 1.092 0.015443        0.01892
## miss_val -0.03838 1.907 0.026970        0.03634
##
## 2. Quantiles for each variable:
##
##             2.5%     25%      50%    75%  97.5%
## var1      1.7062  2.5078  3.10763 3.9493 6.3496
## rho      -0.4707 -0.1653  0.01910 0.2087 0.4979
## var2      1.5465  2.2431  2.75765 3.4717 5.6839
## miss_val -3.7083 -1.2842 -0.05871 1.1900 3.8184
```

```
# Autocovariances
acfplot(missing_mc, lag.max = 30)
```




```
# Effective sample size:
effectiveSize(missing_mc)
```

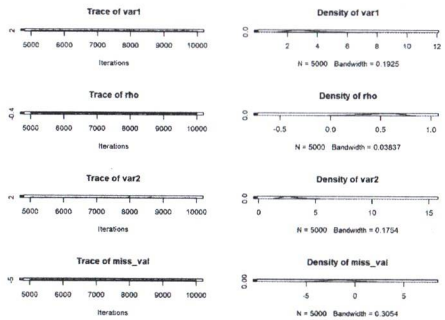
```
##     var1      rho     var2 miss_val
## 3217.089 2247.441 3330.684 2753.747
```

```
dim(missing_mc)[1]
```

```
## [1] 5000
```

```
### ----------------------------------------------------------------------
# RUN THE ALGORITHM (correlated)

set.seed(42)
yC <- rmvnorm(20, c(0,0), sigma = matrix(c(2, 1, 1, 2), ncol = 2))
yC[12:16, 1] <- yC[17:20, 2] <- NA
n <- nrow(yC)
colnames(yC) <- c("y1", "y2")

n0 <- 4
Sigma0 <- Psi0 <- 2* diag(2)
niter <- 10000
burnin <- 5000
thin <- 1

sample_missing_C <- gibbs_missing(niter = niter, burnin = burnin, thin = thin,
                                  data = yC, Sigma0 = Sigma0, n0 = n0, Psi0 = Psi0)
```

```
### ----------------------------------------------------------------------
# diagnostic in CODA

missing_mc_C <- mcmc(sample_missing_C, start = burnin + 1, end = niter, thin = thin)
plot(missing_mc_C)
```

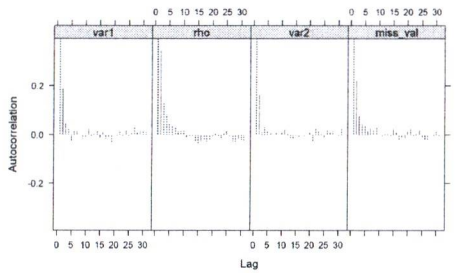


✓

```
summary(missing_mc_C)
```

```
##
## Iterations = 5001:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean     SD Naive SE Time-series SE
## var1      3.2357 1.1744 0.016608       0.020428
## rho       0.4717 0.1992 0.002818       0.004266
## var2      2.9719 1.0657 0.015071       0.018053
## miss_val -1.5000 1.6562 0.023422       0.031450
##
## 2. Quantiles for each variable:
##
##               2.5%    25%     50%     75%  97.5%
## var1      1.688622  2.435  3.0071  3.7723 6.1855
## rho       0.001918  0.352  0.5041  0.6185 0.7763
## var2      1.554310  2.243  2.7558  3.4609 5.6087
## miss_val -4.699409 -2.573 -1.5310 -0.4521 1.8670
```

```
# Autocovariances
acfplot(missing_mc_C, lag.max = 30)
```



✓

```
# Effective sample size:
effectiveSize(missing_mc_C)
```

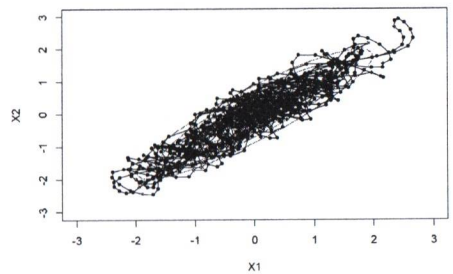
```
##     var1      rho     var2 miss_val
## 3304.811 2181.002 3484.853 2773.181
```

```
dim(missing_mc_C)[1]
```

```
## [1] 5000
```

Bivariate Gaussian distribution with high correlation:
(Hamiltonian strategy vs. Metropolis Hastings Random Walk)

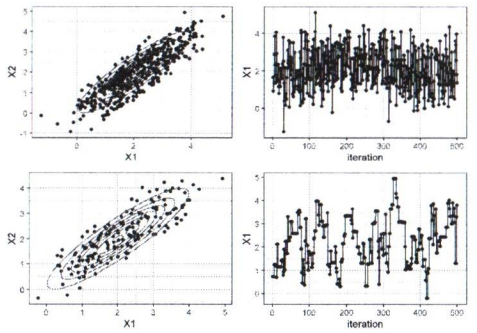
```
##########################################
### HAMILTONIAN MONTE CARLO          #####
##########################################
# First a RW MH to make some comparisons

RW_MH_biv_gaus <- function(niter, burnin, x0, mu, Sigma){
  results <- matrix(ncol = 2, nrow = niter - burnin)
  x_old <- x0

  pb <- txtProgressBar(0, niter, style = 3)
  for(i in 1:niter){
    x_new <- rmvnorm(1, mean = x_old, sigma = 2.38 * diag(1,2) / 2)[1,]
    acc_ratio <- min(1, dmvnorm(x_new, mu, Sigma) / dmvnorm(x_old, mu, Sigma))

    if(runif(1) <= acc_ratio){
      x_old <- x_new
    }

    if(i > burnin){
      results[i - burnin, ] <- c(x_old)
    }

    setTxtProgressBar(pb, i)
  }
  close(pb)
  return(results)
}

##########################################
### HAMILTONIAN MONTE CARLO - a step  #####
##########################################

# How is it working? Take a look on a single step

HMC_step_norm <-  function(epsilon, L, q_old, A){

  q = q_old
  p = rnorm(length(q),0,1) # independent standard normal variates
  p_old = p

  # Make a half step for momentum at the beginning
  p=p - (epsilon / 2) * solve(A) %*% q

  # Alternate full steps for position and momentum
  for (i in 1:L){
    q_plot <- q
    q=q + epsilon * p
    if(i != L) p = p - epsilon * solve(A) %*% q

    points(q[1], q[2], pch = 20)
    segments(q_plot[1], q_plot[2], q[1], q[2])
  }

  # Make a half step for momentum at the end.
  p = p - (epsilon / 2) * solve(A) %*% q
  # Negate momentum at end of trajectory to make the proposal symmetric
  p = - p

  # Evaluate potential and kinetic energies at start and end of trajectory
  current_U = t(q_old) %*% solve(A) %*% q_old / 2
  current_K = sum(p_old^2) / 2
  proposed_U = t(q) %*% solve(A) %*% q / 2
  proposed_K = sum(p^2) / 2

  # Accept or reject the state at end of trajectory, returning either
  # the position at the end of the trajectory or the initial position
  if (runif(1) < exp(current_U-proposed_U+current_K-proposed_K)){
    return (q) # accept
  }
  else{
    return (q_old) # reject
  }
}

#---------------------------------------------------
# plot some iterations
q_old <- c(-1.5,-1)
A <- matrix(c(1,0.9,0.9,1), ncol = 2)

eg <- expand.grid(seq(-4, 4, length.out = 50), seq(-4, 4, length.out = 50))
dns <- apply(eg, 1, function(x) dmvnorm(x = x, sigma = A))
plot(q_old[1], q_old[2], xlim = c(-3,3), ylim = c(-3,3), pch = 20, xlab = "X1", ylab = "X2")
contour(seq(-4, 4, length.out = 50), seq(-4, 4, length.out = 50), matrix(dns, ncol = 50), add = T, col = 2)

set.seed(42)
for(i in 1:100){
  q_old <- HMC_step_norm(0.2, 10, q_old, A)
  points(q_old[1], q_old[2], col = 2, pch = 20)
  Sys.sleep(2)
}
```



```
##########################################
### HAMILTONIAN MONTE CARLO          #####
##########################################

HMC_norm <-  function(niter, burnin, x0, mu, Sigma, epsilon, L){

  results <- matrix(ncol = 2, nrow = niter - burnin)
  x_old <- x0

  pb <- txtProgressBar(0, niter, style = 3)
  for(i in 1:niter){

    x = x_old
    p = rnorm(length(x),0,1)
    p_old = p

    p = p - (epsilon / 2) * solve(Sigma) %*% (x - mu)
    for (j in 1:L){
      x = x + epsilon * p
      if(j != L) p = p - epsilon * solve(Sigma) %*% (x - mu)
    }
    p = p - (epsilon / 2) * solve(Sigma) %*% (x - mu)
    p = - p

    current_U = t(x_old - mu) %*% solve(Sigma) %*% (x_old - mu) / 2
    current_K = sum(p_old^2) / 2
    proposed_U = t(x - mu) %*% solve(Sigma) %*% (x - mu) / 2
    proposed_K = sum(p^2) / 2

    if (runif(1) < exp(current_U-proposed_U+current_K-proposed_K)){
      x_old <- x
    }

    if(i > burnin){
      results[i - burnin, ] <- c(x_old)
    }

    setTxtProgressBar(pb, i)
  }
  close(pb)
  return(results)
}

#------------------------------------------------------------------
# RUN: HMC vs RW-MH

x0 <- c(3,3)
mu <- c(2,2)
Sigma <- matrix(c(1,0.9,0.9,1), ncol = 2)
niter <- 10000
burnin <- 1000

set.seed(123)
HMC_sample <- HMC_norm(niter = niter, burnin = burnin, x0 = x0,
                       mu = mu, Sigma = Sigma, epsilon = 0.2, L = 10)
```

```
MH_sample <- RW_MH_biv_gaus(niter = niter, burnin = burnin, x0 = x0,
                            mu = mu, Sigma = Sigma)
```

```
#------------------------------------------------------------------
# plots
# a common dataset for the two methods
plt_data <- data.frame(x1 = HMC_sample[,1], x2 = HMC_sample[,2],
                       y1 = MH_sample[,1], y2 = MH_sample[,2],
                       ind = 1:(niter - burnin))

# construct the data.frame for the contour plot
eg <- expand.grid(seq(-2, 6, length.out = 50), seq(-2, 6, length.out = 50))
dns_data <- data.frame(x = eg[,1], y = eg[,2],
                       z = apply(eg, 1, function(x) dmvnorm(x = x, mean = c(2,2), sigma = A)))
plt_data <- plt_data[1:500,]

# plots for HMC
p1 <- ggplot(plt_data) +
  geom_point(aes(x = x1, y = x2)) +
  geom_contour(data = dns_data, aes(x = x, y = y, z = z)) +
  theme_bw() +
  xlab("X1") +
  ylab("X2")

p2 <- ggplot(plt_data) +
  geom_point(aes(x = ind, y = x1)) +
  geom_line(aes(x = ind, y = x1)) +
  theme_bw() +
  xlab("iteration") +
  ylab("X1")

# plots for MH
p3 <- ggplot(plt_data) +
  geom_point(aes(x = y1, y = y2)) +
  geom_contour(data = dns_data, aes(x = x, y = y, z = z)) +
  theme_bw() +
  xlab("X1") +
  ylab("X2")

p4 <- ggplot(plt_data) +
  geom_point(aes(x = ind, y = y1)) +
  geom_line(aes(x = ind, y = y1)) +
  theme_bw() +
  xlab("iteration") +
  ylab("X1")

# all the plots
ggarrange(p1, p2, p3, p4, ncol = 2, nrow = 2)
```

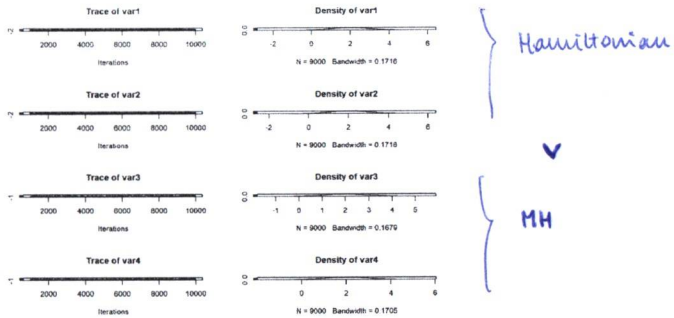


Hamiltonian

∨

Metropolis-Hastings
Random-Walk
(here the sampler it a bit stuck due to the high correlation)

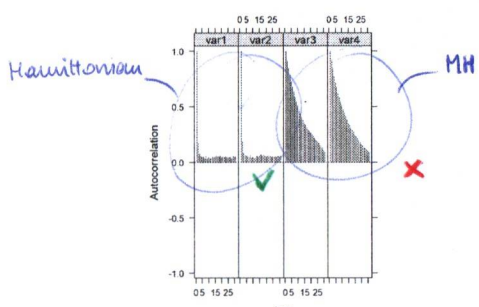
```
#------------------------------------------------------------------
# Diagnostic in CODA

# import the results in coda (all together)
coda_ALL <- mcmc(cbind(HMC_sample, MH_sample), start = burnin + 1, end = niter)

# plot of the chains and summaries
plot(coda_ALL)
```




```
summary(coda_ALL)
```

```
##
## Iterations = 1001:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 9000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##       Mean     SD Naive SE Time-series SE
## [1,] 2.003 0.9999  0.01054        0.02871
## [2,] 2.005 1.0016  0.01056        0.02784
## [3,] 2.004 0.9784  0.01031        0.05458
## [4,] 2.002 0.9935  0.01047        0.05454
##
## 2. Quantiles for each variable:
##
##         2.5%   25%   50%   75% 97.5%
## var1 0.04869 1.331 1.989 2.683 3.946
## var2 0.03096 1.330 1.997 2.686 3.989
## var3 0.07265 1.333 2.002 2.673 3.911
## var4 0.05078 1.312 2.033 2.714 3.888
```

```
# Autocovariances
acfplot(coda_ALL, lag.max = 30)
```




```
# Effective sample size:
effectiveSize(coda_ALL)
```

```
##      var1      var2      var3      var4
## 1212.9373 1294.2718  321.3608  331.8334
```

```
dim(coda_ALL)[1]
```

```
## [1] 9000
```

5/11/2020
1/11

Introduction to STAN: hierarchical models and regression.

November 4, 2020

Gelman and Rubin Multiple Sequence Diagnostic

Idea: starting from several chains with different starting points, if the convergence is reached the trajectories should be similar.

Steps (for each parameter):

1) **Run** $m \geq 2$ chains of length $2n$ from overdispersed starting values
2) **Discard** the first n draws in each chain
3) **Calculate** the within-chain and between-chain variance
4) **Calculate** the estimated variance of the parameter as a weighted sum of the within-chain and between-chain variance
5) **Calculate** the potential scale reduction factor

Note: here W, the within chain variance, is the mean of the variances of each chain and B, the between chain variance, is the variance of the chain means multiplied by n.

in Stan is a popular thing to run more than one chain (and so we can do more diagnostic)

Gelman and Rubin Multiple Sequence Diagnostic[2]

We can **estimate the variance** of the stationary distribution as a weighted average of W and B.

The **potential scale reduction** factor is

$$\hat{R} = \sqrt{\frac{\hat{Var}(\theta)}{W}}$$

when is ~1 is good because it means that all the chains look alike (all the chains look from the same distribution)

When $\hat{R}$ is high (greater than 1.2) then we should run our chains out longer to improve convergence to the stationary distribution.

We can see how the potential scale reduction factor changes through the iterations using the **gelman.plot()** function.

A brief intro to STAN

http://mc-stan.org/

- STAN is a (very new!) probabilistic programming language implementing **full Bayesian statistical inference with MCMC sampling** (NUTS, HMC) and penalized maximum likelihood estimation with Optimization (BFGS).
- STAN is **coded in C++** and runs on all major platforms (Linux, Mac, Windows).
- STAN is an **open-source** software developed mainly at Columbia University.
- STAN can be accessed through **several interfaces**: RStan (R), PyStan (Python), MatlabStan (Matlab) and also CmdStan (shell), http://mc-stan.org/users/interfaces/cmdstan

Sample from the posterior of the model

Multiple Markov chain Monte Carlo (MCMC) algorithms and optimization algorithms are implemented for the inference:

1 MCMC algorithms:

- ☐ Hamiltonian Monte Carlo (HMC) (default)
- ☐ No-U-Turn sampler

2 Optimization algorithms:

- ☐ BFGS algorithm (default), Nesterov's accelerated gradient descent algorithm, Newton's method

Typical workflow of using RStan

- Represent a **statistical model** by writing its log-posterior density (up to a normalization constant independent from the parameters); this can be done using STAN modeling language
- Translate the model coded in **STAN to C++** code using the function stanc
- **Compile the C++ code** for the model using a C++ compiler (such as g++) to create a Dynamic Shared Object that can be loaded by R
- Run the DSO to **sample** from the posterior
- **Diagnose convergence** of the MCMC chains of sample
- Conduct model **inference**

[Don't worry! A single rstan call performs implicitly steps 2, 3 and 4.]

we can just write the likelihood and the prior, stan will do the rest

done with a simple command in R

STAN has specific blocks...

- **DATA**:
  - Given input data
  - Executed first and load
- **TRANSFORMED DATA**:
  - Transform variables for convenience
- **PARAMETERS**:
  - Result output parameters
  - Updated at each iteration
- **TRANFORM. PARAMETERS**:
  - Transform parameters for convenience
- **MODEL**:
  - Describe the model
- **GENERATED QUANTITIES**:
  - Generate quantities for monitoring convergence

describes the data, what we feed the model with

quantities of interest

we specify the prior distribution and the likelihood

... the order must be kept,
the blocks are optional (except model block)

Variable and expression types

- **Primitive**: **int** and **real**
- **Matrix**: **vector[n]**, **row_vector[n]**, **matrix[m, n]**
- **Bounded**: primitive or matrix type, with $< lower = L >$, $< upper = U >$, $< lower = L, upper = U >$
- **Constrained**: **unit_vector** for unit-length vectors, **simplex** for unit simplexes, **ordered** for ordered vectors, **corr_matrix** and **cov_matrix** for symmetric and positive definite matrices.
- **Arrays**: collection of object of any type

5/11/2020
2/11

Variable and expression types

- **Sampling**: $y \sim normal(mu, sigma)$
- **Increments log-probability**: *increment_log_prob(lp)*
  add the value lp to the log-density
- **For/while loop**: *for(n in 1 : N)*, *while(cond)*
- **Block**:{.....} (allows local variables)
- **Print**: *print("TH =", theta)*

Math functions

- All the built-in C + + functions and operators are available.
- For the matrices functions of basic arithmetics, solvers, decompositions, .. are available (check the manual).
- Each distribution of the library has:
  - pseudo random number generator
  - log density or mass function
  - cumulative distribution

Methods of the class S4 stanfit

Printing, plotting, and summarizing:

- **show** Print the default summary for the model.
- **print** Print a customizable summary for the model. See print.stanfit.
- **plot** Create various plots summarizing the fitted model. See plot,stanfit-method.
- **summary**Summarize the distributions of estimated parameters and derived quantities using the posterior draws. See summary,stanfit-method.
- **get_posterior_mean** Get the posterior mean for parameters of interest (using pars to specify a subset of parameters). Returned is a matrix with one column per chain and an additional column for all chains combined.

....

see https://mc-stan.org/rstan/reference/stanfit-class.html for a summary

Example: hierarchical model

The data

The data^2

We have a set of 341 observation divided into 12 genus. For each observation we have the **body weight** *BoW*, the **length from the tail to the nose** *SVL* and the **brain weight** *BrW*.

We are interested into study a feature variable R which is measuring the relation between the body weight and the brain weight:

$$R = log\left(\frac{BrW/BoW}{1-(BrW/BoW)}\right)$$

Remark: $BrW \in (0, BoW)$, $BoW \in \mathbb{R}^+$, $R \in \mathbb{R}$

The model

Consider a **hierarchical model** as follows:

genus. stan

$$R_j = R_1, \dots, R_{n_j} \mid \theta_j, \sigma^2 \sim N(\theta_j, \sigma^2), \quad j = 1, \dots, M \quad \text{(group)}$$

$$\theta_1, \dots, \theta_M \mid \mu, \tau^2 \sim N(\mu, \tau^2)$$

$$(\mu, \tau^2) \sim N(m_0, s_0^2) \times IG(\alpha_0, \beta_0) \quad = \text{normal}(m_0, s_0^2) \cdot \text{inverse-gamma}(\alpha_0, \beta_0)$$

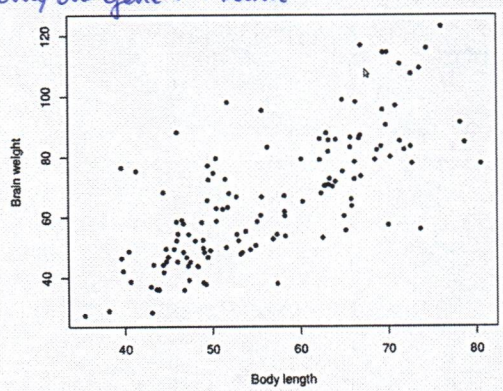
$$\sigma^2 \sim IG(a_0, b_0)$$

where the R_j's are the the brain/body weight indexes in the j-th genus, $\theta_1, \dots, \theta_M$ are the mean values for the index in the M groups, then we have the priors for the parameter (μ, τ^2) and for σ^2.

The target distribution $\propto (\theta_1, \dots, \theta_M, \sigma^2, \mu, \tau^2 \mid \vec{R})$

The data

We want to study the relation Body-Brain weight in only one gene: "Rana"



5/11/2020
3/11

The data[2]

We have a set of 133 observation for the *Rana* genus. For each observation we have the **length from the tail to the nose** X and the **brain weight** Y.

We are interested into study the relation between the brain weight Y and the length of the body Y, model of the kind $Y = g(X)$.

We consider two models:

- $Y = \beta_0 + \beta_1 X$
- $Y = \beta_0 + \beta_1 X + \beta_2 X^2$

16/19

The regression model I

Consider a **regression model** as follows:

regression.stan

$$\mathbf{Y} = Y_1, \ldots, Y_n \mid \beta_0, \beta_1, \sigma^2 \sim N(\beta_0 + \beta_1 X, \sigma^2)$$

$$\beta_j \mid \tau_j^2 \sim N(0, \tau_j^2), \quad j = 1, 2$$

$$\sigma^2 \sim IG(a_0, b_0)$$

where the $\mathbf{Y}$ are the the brain weights, β_0, β_1 are the regression coefficients, then we have the prior for σ^2.

The target distribution is $\mathcal{L}(\beta_0, \beta_1, \sigma^2, \mid \mathbf{Y}, \mathbf{X})$.

17/19

The regression model II

Consider a **regression model** as follows:

regression2.stan

$$\mathbf{Y} = Y_1, \ldots, Y_n \mid \beta_0, \beta_1, \beta_2, \sigma^2 \sim N(\beta_0 + \beta_1 X + \beta_2 X^2, \sigma^2)$$

$$\beta_j \mid \tau_j^2 \sim N(0, \tau_j^2), \quad j = 1, 2, 3$$

$$\sigma^2 \sim IG(a_0, b_0)$$

where the $\mathbf{Y}$ are the the brain weights, $\beta_0, \beta_1, \beta_2$ are the regression coefficients, then we have the prior for σ^2.

The target distribution is $\mathcal{L}(\beta_0, \beta_1, \beta_2, \sigma^2, \mid \mathbf{Y}, \mathbf{X})$.

18/19

WAIC

for MODEL COMPARISON

$$WAIC = -2\, lppd + 2 p_{WAIC}$$

where **lppd** is the log pointwise predictive density

$$lppd = \sum_{i=1}^{n} \log \left(\int p(Y_i \mid \theta) \pi(\theta \mid \mathbf{Y}) d\theta \right) \simeq \sum_{i=1}^{n} \log \left(\frac{1}{S} \sum_{j=1}^{S} p(Y_i \mid \theta^{(j)}) \right)$$

Log pointwise predictive density (the integral is computed in a Montecarlo strategy)

and

$$p_{WAIC} = \sum_{i=1}^{n} Var(\log p(Y_i \mid \theta) \mid \mathbf{Y}) \simeq \sum_{i=1}^{n} \frac{1}{S-1} \sum_{j=1}^{S} \left(\log p(Y_i \mid \theta^{(j)}) - \overline{\log p(Y_i \mid \theta)} \right)^2$$

19/19

5/11/2020

4/11

```
#####################################################################
### genus.stan
#####################################################################

# data  we've giving to the model
# {
#   int<lower = 0> M;       // number of genus
#   int<lower = 0> N[M];    // numerosities in each genus: array of M cells
#   vector<lower = -25, upper = 25>[sum(N)] Y;     // Log(ratio / (1 - ratio)) ---> vector
#   int<lower = 1, upper = M> genus_idx[sum(N)];   // which genus the datum belongs to!
# }
#
# parameters
# {
#   real theta[M];      // array of the mean into each genus   (dim(theta) = M)
#   real<lower = 0> sigma2;        (σ² > 0)
#   real mu;
#   real<lower = 0> tau2;          (τ² > 0)
# }
#
# model
# {
#   // Prior:
#   sigma2 ~ inv_gamma(2., 1.);
#   mu ~ normal(-5., 10.); // mean and std dev
#   tau2 ~ inv_gamma(2., 1.);
#   theta ~ normal(mu, pow(tau2, 0.5));  // iid sample
#
#   // Likelihood:
#   for(i in 1:(sum(N)))
#   {
#       Y[i] ~ normal(theta[genus_idx[i]], pow(sigma2, 0.5));
#   }
#
# }
```

```
#####################################################################
### regression.stan
#####################################################################

# data
# {
#   int<lower = 0> N;       // number of genus
#   vector[N] Y;        // response
#   vector[N] X;        // covariate
# }
#
# parameters
# {
#   real<lower = 0> sigma2;
#   real beta0;
#   real beta1;
# }
#
# transformed parameters
# {
#   vector[N] mu;
#       mu = beta0 + beta1 * X;
# }
#
# model
# {
#   // Prior:
#   sigma2 ~ inv_gamma(2., 10.);
#   beta0 ~ normal(0., 1.);
#   beta1 ~ normal(0., 1.);
#
#   // Likelihood:
#   Y ~ normal(mu, pow(sigma2, 0.5));
#
# }
#
# generated quantities
# {
#       vector[N] log_lik;
#       for (j in 1:N)
#   {
#           log_lik[j] = normal_lpdf(Y[j] | mu[j], pow(sigma2, 0.5));
#       }
# }
```

```
#########################################################################
### regression2.stan
#########################################################################

# data
# {
#   int<lower = 0> N;       // number of obs
#   int<lower = 0> K;       // number of covariates (including the intercept)
#   vector[N] Y;        // response
#   matrix[N, K] X;     // covariates
#
#   // we pass also the parameters for the priors
#   vector[K] mean_beta;
#   vector[K] var_beta;
#   real<lower = 0> scale_s2;
# }
#
# parameters
# {
#   real<lower = 0> sigma2;
#   vector[K] beta;
# }
#
# transformed parameters
# {
#   vector[N] mu;
#       mu = X * beta;
# }
#
# model
# {
#   // Prior:
#   sigma2 ~ inv_gamma(2., scale_s2);
#   for(k in 1:K)
#   {
#       beta[k] ~ normal(mean_beta[k], pow(var_beta[k], 0.5));
#   }
#
#   // Likelihood:
#   Y ~ normal(mu, pow(sigma2, 0.5));
#
# }
#
# generated quantities
# {
#       vector[N] log_lik;
#       for (j in 1:N)
#   {
#           log_lik[j] = normal_lpdf(Y[j] | mu[j], pow(sigma2, 0.5));
#       }
# }
```

5/11/2020
5/11

```
################################################
##### EXAMPLE                          #########
##### Bayesian hierarchical model      #########
################################################

# R & STAN are friends!
library(rstan)
```

```
library(coda)
```

```
# for plots
library(ggplot2)
library(tidyr)
```

```
library(dplyr)
```

```
library(purrr)
library(ggsci)

# dataset: body weights and brain weights of frogs and salamander
# variable of interest log(r / (1 - r)) where r = brain_w / body_w

Y <- read.csv("data.csv")
Y <- Y[complete.cases(Y),]
head(Y)
```

```
##   Order         Family   Genus Species     BoW   SVL   BrW
## 1 Anura Bombinatoridae Bombina  maxima 18543.9 60.24 45.30
## 2 Anura Bombinatoridae Bombina  maxima 11701.3 61.54 52.98
## 3 Anura Bombinatoridae Bombina  maxima 22450.8 59.80 58.70
## 4 Anura Bombinatoridae Bombina  maxima 23549.8 60.39 33.80
## 5 Anura Bombinatoridae Bombina  maxima 23091.0 60.51 40.00
## 6 Anura Bombinatoridae Bombina  maxima 20670.2 61.89 41.00
```

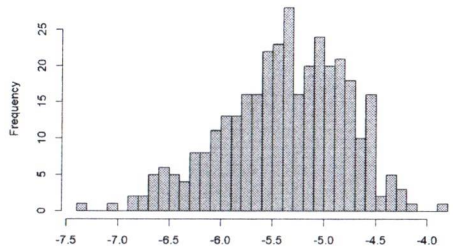
```
dim(Y)
```

```
## [1] 341   7
```

```
# Compute M, number of Genus, and the vector N,
# with the numerosities in each Genus:

m <-  length(unique(Y[,3]))
num_genus <- as.vector(table(Y[,3]))
table(Y[,3])
```

```
##
##     Amolops     Bombina       Bufo   Chaparana  Fejervarya       Hyla
##          26          12          45           6          25          23
##     Kaloula    Nanorana         Paa Polypedates        Rana Rhacophorus
##          17           6           6          16         133          26
```

```
genus_data <- list(M = m, N = num_genus,
                   Y = log((Y[,7]/Y[,5]) / (1 - Y[,7]/Y[,5])),
                   genus_idx = as.numeric(as.factor(Y$Genus)))
hist(genus_data$Y, breaks = 40, main = "", xlab = "")
```




```
# FIT THE MODEL ----------------------------------------------------------
# Fit a model defined in the Stan modeling language
# and return the fitted result in an instance of class stanfit
# Steps performed by the function:
# (i) translate the model into c++ code
# (ii) compile the C++ code into a binary shared object
# (iii) sampling

?stan
```

```
## starting httpd help server ...
```

```
##  done
```

```
fit1 <- stan(file = "genus.stan",
             data = genus_data,
             iter = 5000, thin = 5,
             chains = 2, warmup = 1000,
             algorithm = 'NUTS',
             diagnostic_file = 'diagnostic.txt',
             verbose = TRUE,
             seed = 42)
```

adaptive algorithm

to reproduce the results

```
# Arguments:
# chains = number of chains that are randomly initialized, default is 4
# (iter, thin, warmup are setted for each chain)
# algorithm = 'NUTS', 'HMC'
# sample_file, diagnostic_file = write files with samples or diagnostic data
# save_dso = whether save the dso object or not
# control = set the parameters of the algorithm and the adaptation parameters

is(fit1)
```

```
## [1] "stanfit"
```

```
# We can split the 3 steps into 3 different functions:
# (i) stanc: translate stan model into C++ code

step1 <- stanc(file = 'genus.stan', model_name = 'genus_ex')
names(step1)
```

```
## [1] "status"        "model_cppname" "cppcode"       "model_name"
## [5] "model_code"
```

```
# (ii) stan_model: construct a stan model tha can be used to draw samples from the model
# The C++ code is compiled into a Dynamic Shared Object.
# DSO = A dynamic shared object (DSO) is an object file that is meant to be
# used simultaneously (or shared) by multiple applications while they are executing.

step2 <- stan_model(stanc_ret = step1, verbose = FALSE)
is(step2)
```

```
## [1] "stanmodel"
```

```
# (iii) sampling: draw samples from a stan-model
step3 <- sampling(step2,
                  data = genus_data, chains = 2, iter = 2000,
                  warmup = 500, thin = 1, seed = 42)
```

```
##
## SAMPLING FOR MODEL 'genus_ex' 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: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  501 / 2000 [ 25%]  (Sampling)
## Chain 1: Iteration:  700 / 2000 [ 35%]  (Sampling)
## Chain 1: Iteration:  900 / 2000 [ 45%]  (Sampling)
## Chain 1: Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1: Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1: Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1: Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1: Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.229 seconds (Warm-up)
## Chain 1:                0.469 seconds (Sampling)
## Chain 1:                0.698 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'genus_ex' 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: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  501 / 2000 [ 25%]  (Sampling)
## Chain 2: Iteration:  700 / 2000 [ 35%]  (Sampling)
## Chain 2: Iteration:  900 / 2000 [ 45%]  (Sampling)
## Chain 2: Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2: Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2: Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2: Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2: Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.262 seconds (Warm-up)
## Chain 2:                0.486 seconds (Sampling)
## Chain 2:                0.748 seconds (Total)
## Chain 2:
```

```
?sampling
is(step3)
```

```
## [1] "stanfit"
```

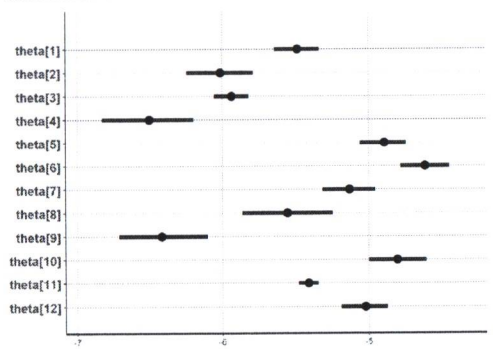
```
# POST PROCESSING ----------------------------------------------------

print(fit1,
      probs = c(0.025, 0.5, 0.975),
      par = c('mu', 'tau2', 'sigma2'))
```

```
## Inference for Stan model: genus.
## 2 chains, each with iter=5000; warmup=1000; thin=5;
## post-warmup draws per chain=800, total post-warmup draws=1600.
##
##         mean se_mean   sd  2.5%   50% 97.5% n_eff Rhat
## mu     -5.49    0.01 0.21 -5.91 -5.48 -5.08  1656    1
## tau2    0.49    0.00 0.20  0.23  0.45  1.02  1673    1
## sigma2  0.16    0.00 0.01  0.14  0.16  0.19  1571    1
##
## Samples were drawn using NUTS(diag_e) at Sat Dec 26 00:30:08 2020.
## 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).
```

```
# Output:
# Info about the iterations and the used sampler
# Summary about parameters includes: the mean, the stardard error of the mean
# (st_dev/sqrt(n)),
# standard deviation and quantiles, the effective sample size and the split Rhat
# All of them are computed without the warmup samples.

# The split rate R:
# One way to monitor whether a chain has converged to the equilibrium distribution
# is to compare its behavior to other randomly initialized chains.
# The R statistic measures the ratio of the average variance of samples within each
# chain to the variance of the pooled samples across chains; if all chains are at
# equilibrium, these will be the same and R will be one.
# If the chains have not converged to a common distribution, the R stat will be greater than 1.

plot(fit1, ask = T, pars = "theta", ci_level = 0.95, fill_color = "blue")
```

```
## ci_level: 0.95 (95% intervals)
```

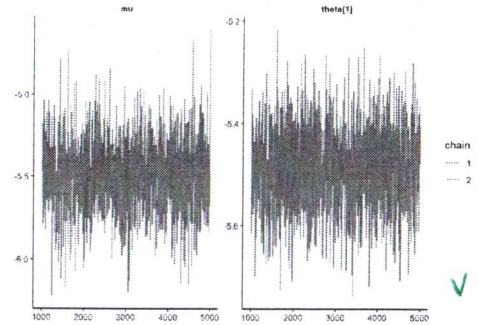
```
## outer_level: 0.95 (95% intervals)
```



mean (posterior) + credible interval for each parameter

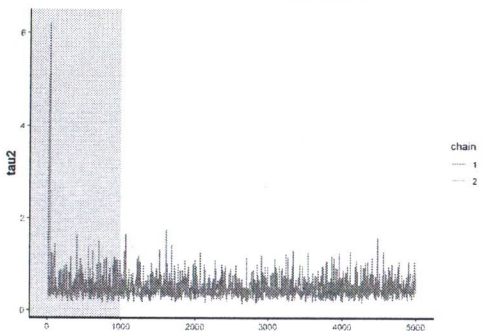
```
# Draw the traceplot corresponding to one or more Markov chains,
# providing a visual way to inspect
# sampling behavior and assess mixing across chains and convergence.

rstan::traceplot(fit1, pars = c('mu', 'theta[1]'), inc_warmup = FALSE)
```

5/11/2020
6/11



spikes ✓
no trend ✓
moreover the two chains overlap ✓

```
rstan::traceplot(fit1, pars='tau2', inc_warmup = TRUE)
```




```
# Extract the values of the chains, discarding the warmup (burnin)
# permuted = T is mixing together different chains

par <- rstan::extract(fit1, pars = c("mu", "tau2", "sigma2"), permuted = TRUE)
is(par)
```

```
## [1] "list"   "vector"
```

```
names(par)
```

```
## [1] "mu"     "tau2"   "sigma2"
```

```
mu <- par$mu
sig2 <- par$sigma2
tau2 <- par$tau2
length(mu)
```

```
## [1] 1600
```

```
# permuted = T
theta <- rstan::extract(fit1, pars='theta', permuted = TRUE)
theta <- as.data.frame(theta)
head(theta)
```

```
##     theta.1   theta.2   theta.3   theta.4   theta.5   theta.6   theta.7
## 1 -5.564416 -6.020027 -5.947470 -6.540032 -4.989690 -4.664250 -5.155128
## 2 -5.517353 -5.974061 -5.857821 -6.515771 -4.842932 -4.532301 -5.187231
## 3 -5.541215 -6.146815 -5.910979 -6.360873 -4.911621 -4.574073 -5.059364
## 4 -5.446020 -6.001836 -5.988937 -6.403410 -4.965873 -4.383605 -5.146782
## 5 -5.383781 -6.042250 -5.998785 -6.783995 -4.993508 -4.641451 -5.082626
## 6 -5.687307 -5.740500 -5.819411 -6.226673 -4.902599 -4.446042 -5.144108
##     theta.8   theta.9  theta.10  theta.11  theta.12
## 1 -5.576948 -6.531898 -4.894100 -5.353026 -5.113530
## 2 -5.377250 -6.559967 -4.713586 -5.429696 -4.949506
## 3 -5.416739 -6.722691 -4.828463 -5.389647 -5.032335
## 4 -5.911699 -6.339601 -4.909600 -5.337732 -5.163152
## 5 -5.707826 -6.179845 -4.806417 -5.401022 -5.022910
## 6 -5.653977 -6.636936 -4.861800 -5.416102 -5.152136
```

```
dim(theta)
```

```
## [1] 1600   12
```

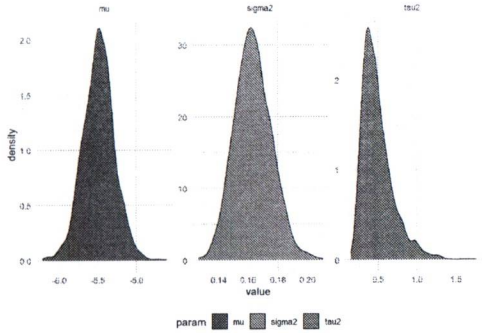
```
# Other useful methods:
post_mean <- get_posterior_mean(fit1)
rownames(post_mean)[1:12] <- levels(as.factor(Y$Genus))
post_mean
```

```
##             mean-chain:1 mean-chain:2 mean-all chains
## Amolops       -5.4860630   -5.4867829       -5.4864230
## Bombina       -6.0196932   -6.0110154       -6.0153543
## Bufo          -5.9358331   -5.9371755       -5.9365043
## Chaparana     -6.5052676   -6.5051963       -6.5052319
## Fejervarya    -4.8962140   -4.8898551       -4.8930346
## Hyla          -4.6107428   -4.6110062       -4.6108745
## Kaloula       -5.1306869   -5.1328643       -5.1317756
## Nanorana      -5.5519425   -5.5645864       -5.5582645
## Paa           -6.4229788   -6.4158082       -6.4193935
## Polypedates   -4.8032051   -4.8067025       -4.8049538
## Rana          -5.4120805   -5.4133209       -5.4127007
## Rhacophorus   -5.0270369   -5.0250899       -5.0260634
## sigma2         0.1631791    0.1634428        0.1633109
## mu            -5.4891632   -5.4862205       -5.4876919
## tau2           0.4908686    0.4932224        0.4920455
## lp__         138.8598433  138.9770334      138.9184384
```

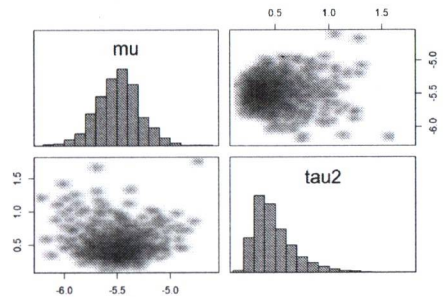
```
# plot the posterior distributions

plot_post <- fit1 %>%
  rstan::extract(c("mu", "tau2", "sigma2")) %>%
  as_tibble() %>%
  map_df(as_data_frame, .id = 'param')
```

```
## Warning: `as_data_frame()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="bottom")
```




```
pairs(fit1, pars = c('mu', 'tau2'), condition = 0.5)
```




```
# CODA ----------------------------------------------------------
# Thus, you can then get standard rjags/coda-style summary tables and plots.

coda_chain <- As.mcmc.list(fit1)
summary(coda_chain)
```

```
##
## Iterations = 1005:5000
## Thinning interval = 5
## Number of chains = 2
## Sample size per chain = 800
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean      SD  Naive SE Time-series SE
## theta[1]   -5.4864 0.07841 0.0019603      0.0019061
## theta[2]   -6.0154 0.11572 0.0028930      0.0028916
## theta[3]   -5.9365 0.06063 0.0015157      0.0015623
## theta[4]   -6.5052 0.16184 0.0040460      0.0042665
## theta[5]   -4.8930 0.08163 0.0020408      0.0020399
## theta[6]   -4.6109 0.08509 0.0021273      0.0021280
## theta[7]   -5.1318 0.09592 0.0023980      0.0023986
## theta[8]   -5.5583 0.16005 0.0040012      0.0038481
## theta[9]   -6.4194 0.15637 0.0039093      0.0046422
## theta[10]  -4.8050 0.10032 0.0025080      0.0025084
## theta[11]  -5.4127 0.03475 0.0008688      0.0008690
## theta[12]  -5.0261 0.08103 0.0020258      0.0020263
## sigma2      0.1633 0.01254 0.0003134      0.0003135
## mu         -5.4877 0.20636 0.0051591      0.0051587
## tau2        0.4920 0.20418 0.0051044      0.0051059
## lp__      138.9184 2.77361 0.0693403      0.0732548
##
## 2. Quantiles for each variable:
##
##               2.5%      25%      50%      75%    97.5%
## theta[1]   -5.6432  -5.5409  -5.4861  -5.4321  -5.3372
## theta[2]   -6.2472  -6.0941  -6.0133  -5.9353  -5.7879
## theta[3]   -6.0567  -5.9754  -5.9366  -5.8967  -5.8159
## theta[4]   -6.8276  -6.6138  -6.5039  -6.3930  -6.1999
## theta[5]   -5.0583  -4.9503  -4.8911  -4.8357  -4.7415
## theta[6]   -4.7785  -4.6676  -4.6119  -4.5530  -4.4410
## theta[7]   -5.3158  -5.1964  -5.1316  -5.0639  -4.9539
## theta[8]   -5.8647  -5.6634  -5.5580  -5.4532  -5.2413
## theta[9]   -6.7123  -6.5292  -6.4202  -6.3136  -6.1028
## theta[10]  -4.9992  -4.8740  -4.8048  -4.7402  -4.6051
## theta[11]  -5.4802  -5.4350  -5.4120  -5.3902  -5.3445
## theta[12]  -5.1876  -5.0775  -5.0243  -4.9716  -4.8714
## sigma2      0.1402   0.1547   0.1627   0.1715   0.1887
## mu         -5.9054  -5.6219  -5.4848  -5.3621  -5.0807
## tau2        0.2294   0.3481   0.4488   0.5835   1.0164
## lp__      132.3481 137.3404 139.3401 140.9368 143.1369
```

If the 2 are similar the sample seems to come from an independent distribution ✓

```
# Compute the gelman and rubin's convergence diagnostic
gelman.diag(coda_chain, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)
```

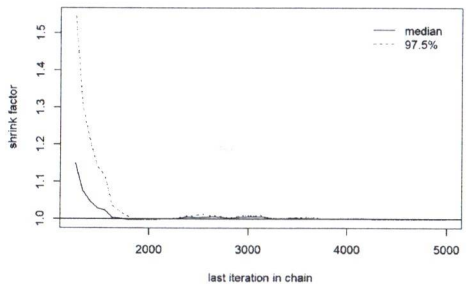
← we want it close to 1

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## theta[1]       0.999      0.999
## theta[2]       1.004      1.023
## theta[3]       1.001      1.003
## theta[4]       1.018      1.035
## theta[5]       1.003      1.018
## theta[6]       1.001      1.001
## theta[7]       1.006      1.025
## theta[8]       1.005      1.026
## theta[9]       1.001      1.002
## theta[10]      1.001      1.002
## theta[11]      0.999      1.000
## theta[12]      1.000      1.000
## sigma2         0.999      0.999
## mu             1.002      1.005
## tau2           0.999      0.999
## lp__           1.005      1.029
##
## Multivariate psrf
##
## 1.02
```

✓

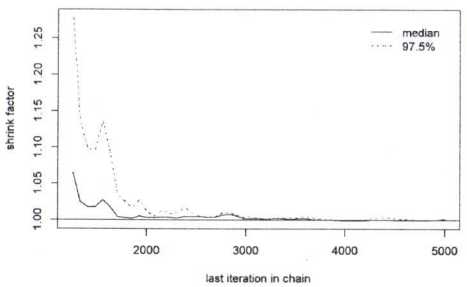
5/11/2020
4/11

```
# â potential scale reduction factorâ is calculated for each variable in x,
# together with upper and lower confidence limits. Approximate convergence is
# diagnosed when the upper limit is close to 1. For multivariate chains, a multivariate
# value is calculated that bounds above the potential scale reduction factor for
# any linear combination of the (possibly transformed) variables

# This plot shows the evolution of Gelman and Rubinâs shrink factor as the number
# of iterations increases

gelman.plot(coda_chain[,1], confidence = 0.95)
```



√

```
gelman.plot(coda_chain[,10], confidence = 0.95)
```



√

```
# Geweke's convergence diagnostic

geweke.diag(coda_chain, frac1=0.1, frac2=0.5)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  theta[1]  theta[2]  theta[3]  theta[4]  theta[5]  theta[6]  theta[7]  theta[8]
##   1.03137   0.48831  -0.51213   2.00768   1.62820  -0.68911   0.06433  -0.31644
##  theta[9] theta[10] theta[11] theta[12]    sigma2        mu      tau2       lp__
##  -1.29011   2.55167   0.95158   0.19520  -1.19261   0.75953   0.37998   0.18273
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  theta[1]  theta[2]  theta[3]  theta[4]  theta[5]  theta[6]  theta[7]  theta[8]
##  -2.23074   1.87965   0.62554  -0.83404  -0.50976   2.06815   1.26767   0.39706
##  theta[9] theta[10] theta[11] theta[12]    sigma2        mu      tau2       lp__
##   0.08466   0.52401  -1.90652  -0.45058   1.00894  -0.30276   0.48057   1.10584
```

```
# OPTIMIZATION: obtain a point estimate by maximing the joint posterior
# from the model defined by class stanmodel

is(step2)
```

```
## [1] "stanmodel"
```

```
opt <- optimizing(step2,
                  data = genus_data,
                  algorithm = 'Newton')
opt
```

```
## $par
##   theta[1]   theta[2]   theta[3]   theta[4]   theta[5]   theta[6]   theta[7]
## -5.4859567 -6.0168279 -5.9370624 -6.4901737 -4.8938234 -4.6147617 -5.1338464
##   theta[8]   theta[9]  theta[10]  theta[11]  theta[12]     sigma2         mu
## -5.5575901 -6.4089653 -5.8052538 -5.4125007 -5.0287780  0.1559170 -5.4820056
##       tau2
##  0.3478813
##
## $value
## [1] 149.4373
##
## $return_code
## [1] 0
##
## $theta_tilde
##       theta[1]  theta[2]  theta[3]  theta[4]  theta[5]  theta[6]  theta[7]
## [1,] -5.485957 -6.016828 -5.937062 -6.490174 -4.893823 -4.614762 -5.133846
##      theta[8]  theta[9] theta[10] theta[11] theta[12]   sigma2        mu
## [1,] -5.55759 -6.408965 -5.805254 -5.412501 -5.028778 0.155917 -5.482006
##           tau2
## [1,] 0.3478813
```

```
# Arguments:
# algorithm = 'LBFGS', 'BFGS', 'Newton'

# RUN MULTIPLE CHAINS IN PARALLEL ----------------------------------------
# Specifying the argument chains, rstan runs chains sequentially using one R
# process, which means that  rstan does not support sampling in parallel
# directly. However, the function SFLIST2STANFIT merges a list of stanfit objects
# into one.
#
# Argument: a list of stanfit objects. The stanfit objects to be merged
#           need to have the same configuration of iteration, warmup and thinning
#
# Output: an object of stanfit consolidated from all the input stanfit objects
#
# We can run multiple chains in parallel using using other approaches or packages
# and then use this function to merge them.

library(parallel)

# Create the model:

fit2 <- stan_model(file = 'genus.stan')

# Set parameters of the chain:

IT = 1000
WU = 1000
TH = 5

seed <- 42


# Note: the following does not work on Windows systems (use function parLapply or
# package foreach)
# mc.cores = The number of cores to use, i.e. at most how many child processes will
# be run simultaneously. The option is initialized from environment variable MC_CORES
# if set. Must be at least one, and parallelization requires at least two cores.

detectCores()
```

```
## [1] 4
```

```
# # multicore
# system.time(
# sflist <- mclapply(1:4,
#                    function(i)
#                      sampling(step2,
#                               data = genus_data, chains = 1, iter = IT*TH+WU,
#                               warmup = WU, thin= TH,
#                               chain_id = i,
#                               refresh = -1,
#                               pars = c('mu', 'sigma2', 'tau2')),
#                    mc.cores = 4)
# )

# single core
system.time(
  sflist <- mclapply(1:4,
                     function(i)
                       sampling(step2,
                                data = genus_data, chains = 1, iter = IT*TH+WU,
                                warmup = WU, thin= TH,
                                chain_id = i,
                                refresh = -1,
                                pars = c('mu', 'sigma2', 'tau2')),
                     mc.cores = 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:
## Chain 1:  Elapsed Time: 0.468 seconds (Warm-up)
## Chain 1:                1.883 seconds (Sampling)
## Chain 1:                2.351 seconds (Total)
## Chain 1:
## 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:
## Chain 2:  Elapsed Time: 0.64 seconds (Warm-up)
## Chain 2:                1.924 seconds (Sampling)
## Chain 2:                2.564 seconds (Total)
## Chain 2:
## 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:
## Chain 3:  Elapsed Time: 0.414 seconds (Warm-up)
## Chain 3:                1.536 seconds (Sampling)
## Chain 3:                1.95 seconds (Total)
## Chain 3:
## 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:
## Chain 4:  Elapsed Time: 0.391 seconds (Warm-up)
## Chain 4:                2.039 seconds (Sampling)
## Chain 4:                2.43 seconds (Total)
## Chain 4:
```

```
##    user  system elapsed
##    8.95    0.02    9.44
```

```
is(sflist)
```

```
## [1] "list"   "vector"
```

```
# Summary for 4 different chains
print(sflist)
```

```
## [[1]]
## Inference for Stan model: genus_ex.
## 1 chains, each with iter=6000; warmup=1000; thin=5;
## post-warmup draws per chain=1000, total post-warmup draws=1000.
##
##          mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## mu      -5.49    0.01 0.22  -5.93  -5.63  -5.50  -5.36  -5.02   839    1
## sigma2   0.16    0.00 0.01   0.14   0.15   0.16   0.17   0.19  1101    1
## tau2     0.50    0.01 0.23   0.24   0.36   0.45   0.60   1.05   993    1
## lp__   138.91    0.09 2.81 132.41 137.30 139.22 140.98 143.41   999    1
##
## Samples were drawn using NUTS(diag_e) at Sat Dec 26 00:31:13 2020.
## 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).
##
## [[2]]
## Inference for Stan model: genus_ex.
## 1 chains, each with iter=6000; warmup=1000; thin=5;
## post-warmup draws per chain=1000, total post-warmup draws=1000.
##
##          mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## mu      -5.48    0.01 0.20  -5.87  -5.61  -5.49  -5.36  -5.09   938    1
## sigma2   0.16    0.00 0.01   0.14   0.15   0.16   0.17   0.19  1037    1
## tau2     0.49    0.01 0.21   0.22   0.35   0.45   0.58   1.04   957    1
## lp__   139.04    0.09 2.72 133.08 137.26 139.40 141.09 143.46   870    1
##
## Samples were drawn using NUTS(diag_e) at Sat Dec 26 00:31:16 2020.
## 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).
##
## [[3]]
## Inference for Stan model: genus_ex.
## 1 chains, each with iter=6000; warmup=1000; thin=5;
## post-warmup draws per chain=1000, total post-warmup draws=1000.
##
##          mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## mu      -5.48    0.01 0.20  -5.86  -5.60  -5.47  -5.34  -5.10   759    1
## sigma2   0.16    0.00 0.01   0.14   0.15   0.16   0.17   0.19   882    1
## tau2     0.49    0.01 0.22   0.22   0.34   0.43   0.58   1.04   950    1
## lp__   138.93    0.09 2.76 132.77 137.21 139.36 140.93 143.49   931    1
##
## Samples were drawn using NUTS(diag_e) at Sat Dec 26 00:31:18 2020.
## 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).
##
## [[4]]
## Inference for Stan model: genus_ex.
## 1 chains, each with iter=6000; warmup=1000; thin=5;
## post-warmup draws per chain=1000, total post-warmup draws=1000.
##
##          mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## mu      -5.49    0.01 0.21  -5.89  -5.63  -5.48  -5.35  -5.07  1134    1
## sigma2   0.16    0.00 0.01   0.14   0.15   0.16   0.17   0.19  1053    1
## tau2     0.50    0.01 0.23   0.23   0.36   0.46   0.58   1.10  1109    1
## lp__   138.96    0.09 2.79 132.79 137.26 139.27 140.89 143.41   907    1
##
## Samples were drawn using NUTS(diag_e) at Sat Dec 26 00:31:20 2020.
## 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).
```

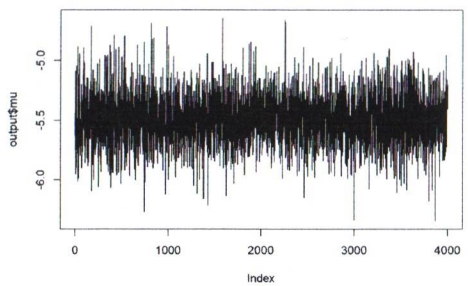
```
# Merge the chains: we will obtain IT*NUM.CORES iterations
fit_p <- sflist2stanfit(sflist)
print(fit_p)
```

```
## Inference for Stan model: genus_ex.
## 4 chains, each with iter=6000; warmup=1000; thin=5;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## mu      -5.48    0.00 0.21  -5.89  -5.62  -5.48  -5.35  -5.08  3762    1
## sigma2   0.16    0.00 0.01   0.14   0.15   0.16   0.17   0.19  4024    1
## tau2     0.50    0.00 0.22   0.23   0.35   0.45   0.59   1.06  4040    1
## lp__   138.96    0.05 2.77 132.78 137.25 139.32 140.98 143.44  3782    1
##
## Samples were drawn using NUTS(diag_e) at Sat Dec 26 00:31:20 2020.
## 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).
```

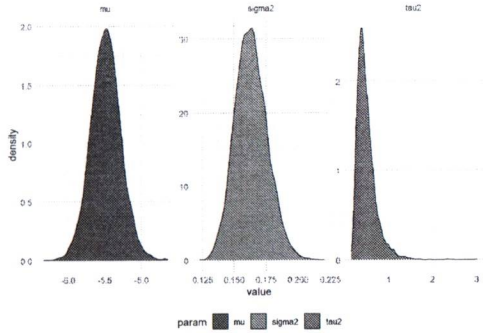
```
# Plot a chain
output <- rstan::extract(fit_p, pars=c('mu', 'tau2', "sigma2") )
length(output$mu)
```

```
## [1] 4000
```

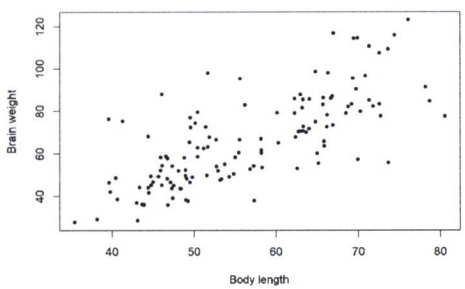
```
plot(output$mu, type='l', lwd=1)
```




```
# plot the posterior distributions

plot_post <- output %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="bottom")
```




```
################################################
##### EXAMPLE                          #########
##### Bayesian regression              #########
################################################

regression_data <- list(N = sum(Y$Genus == "Rana"),
                        Y = Y$BrW[Y$Genus == "Rana"],
                        X = Y$SVL[Y$Genus == "Rana"])
plot(regression_data$X, regression_data$Y, pch = 20,
     xlab = "Body length", ylab = "Brain weight")
```




```
# FIT THE MODEL ----------------------------------------------------------

fit2 <- stan(file = "regression.stan",
             data = regression_data,
             iter = 5000, thin = 5,
             chains = 2, warmup = 1000,
             algorithm = 'NUTS',
             diagnostic_file = 'diag_reg.txt',
             verbose = FALSE,
             seed = 42)
```

```
##
## SAMPLING FOR MODEL 'regression' 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: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1: Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1: Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1: Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.132 seconds (Warm-up)
## Chain 1:                0.456 seconds (Sampling)
## Chain 1:                0.588 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'regression' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.001 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 10 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2: Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 2: Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 2: Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2: Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 2: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.133 seconds (Warm-up)
## Chain 2:                0.444 seconds (Sampling)
## Chain 2:                0.577 seconds (Total)
## Chain 2:
```

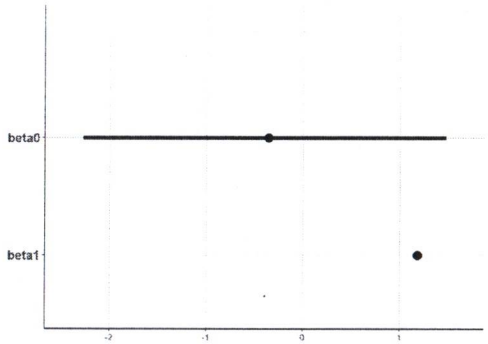
```
print(fit2, par = c('beta0', 'beta1', 'sigma2'))
```

```
## Inference for Stan model: regression.
## 2 chains, each with iter=5000; warmup=1000; thin=5;
## post-warmup draws per chain=800, total post-warmup draws=1600.
##
##          mean se_mean    sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## beta0   -0.37    0.02  0.97  -2.27  -1.02  -0.35   0.29   1.48  1683    1
## beta1    1.18    0.00  0.03   1.13   1.17   1.18   1.20   1.24  1577    1
## sigma2 209.07    0.67 25.55 164.93 191.20 207.29 224.52 264.24  1473    1
##
## Samples were drawn using NUTS(diag_e) at Sat Dec 26 00:32:14 2020.
## 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).
```

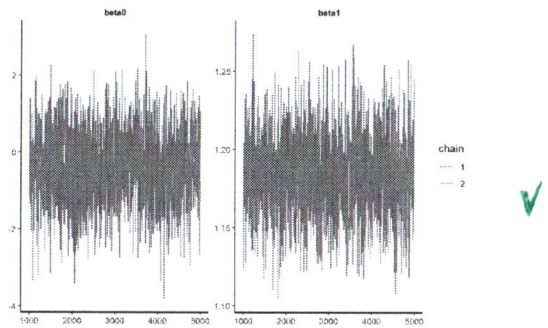
```
plot(fit2, ask = T, pars = c("beta0", "beta1"), ci_level = 0.95, fill_color = "blue")
```

```
## ci_level: 0.95 (95% intervals)
## outer_level: 0.95 (95% intervals)
```

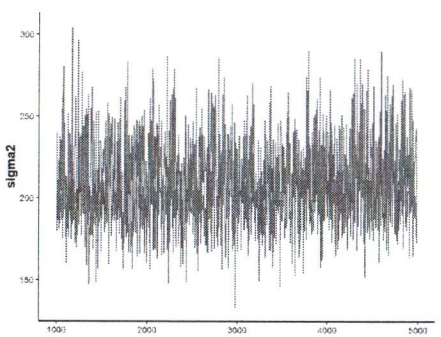
5/11/2020
9/11




```
# Draw the traceplot corresponding to one or more Markov chains,
# providing a visual way to inspect
# sampling behavior and assess mixing across chains and convergence.

rstan::traceplot(fit2, pars = c("beta0", "beta1"), inc_warmup = FALSE)
```




```
rstan::traceplot(fit2, pars='sigma2', inc_warmup = FALSE)
```



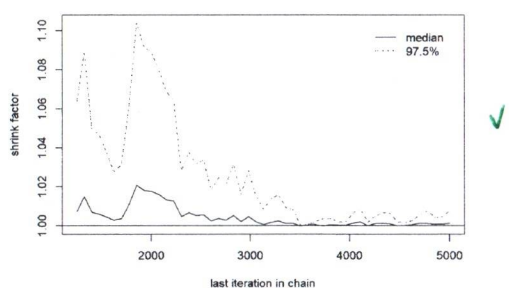

```
#-----------------------------------------------------------------------
# DIAGNOSTIC IN CODA ---------------------------------------------------
#-----------------------------------------------------------------------

coda_chain_reg <- As.mcmc.list(fit2, pars = c("beta0", "beta1", "sigma2"))
summary(coda_chain_reg)
```

```
##
## Iterations = 1005:5000
## Thinning interval = 5
## Number of chains = 2
## Sample size per chain = 800
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean      SD  Naive SE Time-series SE
## beta0   -0.3662  0.9727 0.0243167      0.0243234
## beta1    1.1841  0.0268 0.0006699      0.0007176
## sigma2 209.0738 25.5544 0.6388601      0.6388825
##
## 2. Quantiles for each variable:
##
##           2.5%     25%      50%      75%   97.5%
## beta0   -2.270  -1.019  -0.3539   0.2915   1.477
## beta1    1.132   1.165   1.1837   1.2026   1.237
## sigma2 164.927 191.196 207.2946 224.5227 264.237
```

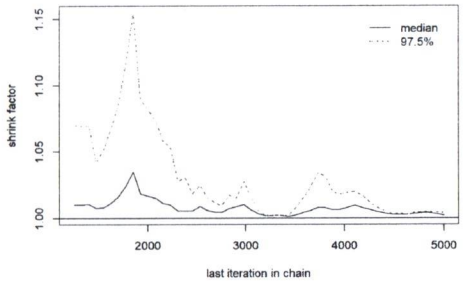
```
# Compute the gelman and rubin's convergence diagnostic

gelman.diag(coda_chain_reg, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)
```

```
## Potential scale reduction factors:
##
##        Point est. Upper C.I.
## beta0           1       1.01
## beta1           1       1.00
## sigma2          1       1.00
##
## Multivariate psrf
##
## 1
```

```
# â potential scale reduction factorâ is calculated for each variable in x,
# together with upper and lower confidence limits. Approximate convergence is
# diagnosed when the upper limit is close to 1. For multivariate chains, a multivariate
# value is calculated that bounds above the potential scale reduction factor for
# any linear combination of the (possibly transformed) variables

# This plot shows the evolution of Gelman and Rubinâs shrink factor as the number
# of iterations increases

gelman.plot(coda_chain_reg[,1], confidence = 0.95)
```




```
gelman.plot(coda_chain_reg[,2], confidence = 0.95)
```

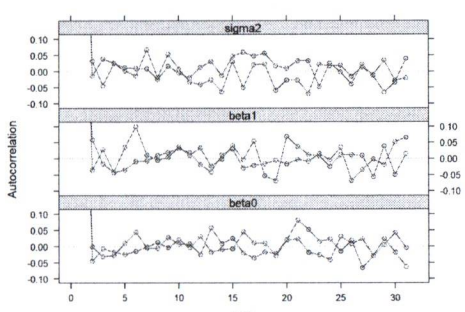


✓

```
# Geweke's convergence diagnostic
geweke.diag(coda_chain_reg, frac1=0.1, frac2=0.5)
```

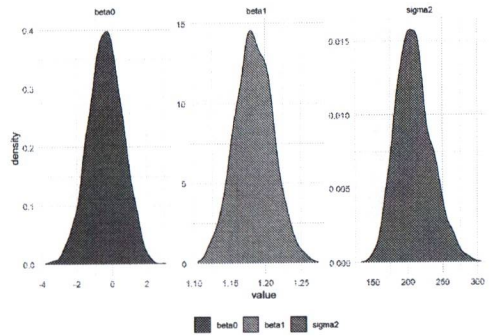
```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  beta0  beta1 sigma2
## -1.632  1.707  4.030
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   beta0   beta1  sigma2
##  0.5899 -0.4188 -1.6639
```

```
# autocorrelation and plot
acfplot(coda_chain_reg, lag.max = 30)
```



✓

```
# plots of the posteriors
plot_post <- fit2 %>%
  rstan::extract(c("beta0", "beta1", "sigma2")) %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="bottom") +
  labs(fill = "")
```




```
################################################
##### EXAMPLE                          #########
##### Bayesian regression (square)     #########
################################################

regression_data_full <- list(N = sum(Y$Genus == "Rana"),
                             K = 3,
                             Y = Y$BrW[Y$Genus == "Rana"],
                             X = cbind(rep(1, sum(Y$Genus == "Rana")),
                                       Y$SVL[Y$Genus == "Rana"],
                                       (Y$SVL[Y$Genus == "Rana"])^2),
                             scale_s2 = 10)

reg_lm <- summary(lm(regression_data_full$Y ~ regression_data_full$X[,2:3]))
regression_data_full$mean_beta <- reg_lm$coefficients[,1]
regression_data_full$var_beta <- (reg_lm$coefficients[,2])^2

# FIT THE MODEL (again) ------------------------------------------------------------

fit3 <- stan(file = "regression2.stan",
             data = regression_data_full,
             iter = 5000, thin = 5,
             chains = 2, warmup = 1000,
             algorithm = 'NUTS',
             diagnostic_file = 'diag_reg.txt',
             verbose = FALSE,
             seed = 42)
```

5/11/2020
10/11

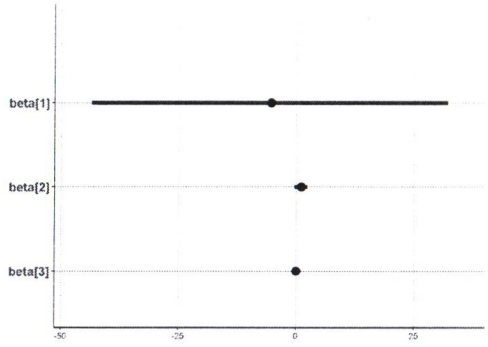
```
##
## SAMPLING FOR MODEL 'regression2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1: Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1: Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1: Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 1.793 seconds (Warm-up)
## Chain 1:                4.193 seconds (Sampling)
## Chain 1:                5.986 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'regression2' 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: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2: Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 2: Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 2: Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2: Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 2: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 2.541 seconds (Warm-up)
## Chain 2:                5.224 seconds (Sampling)
## Chain 2:                7.765 seconds (Total)
## Chain 2:
```

```
print(fit3, par = c('beta', 'sigma2'))
```

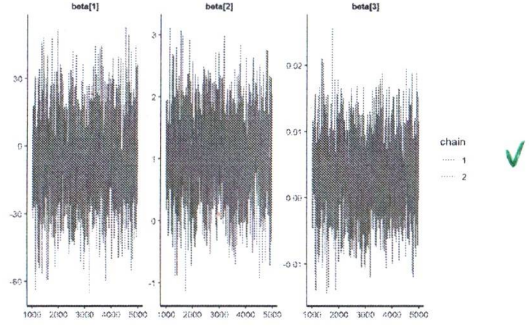
```
## Inference for Stan model: regression2.
## 2 chains, each with iter=5000; warmup=1000; thin=5;
## post-warmup draws per chain=800, total post-warmup draws=1600.
##
##          mean se_mean    sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## beta[1]  -5.52    0.57 19.33 -43.46 -18.29  -5.30   6.90  32.05  1158    1
## beta[2]   1.06    0.02  0.68  -0.29   0.60   1.05   1.50   2.37  1146    1
## beta[3]   0.00    0.00  0.01  -0.01   0.00   0.00   0.01   0.02  1137    1
## sigma2  202.07    0.65 24.93 158.35 184.92 200.25 217.72 256.21  1469    1
##
## Samples were drawn using NUTS(diag_e) at Sat Dec 26 00:33:22 2020.
## 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).
```

```
plot(fit3, ask = T, pars = c('beta'), ci_level = 0.95, fill_color = "blue")
```

```
## ci_level: 0.95 (95% intervals)
## outer_level: 0.95 (95% intervals)
```

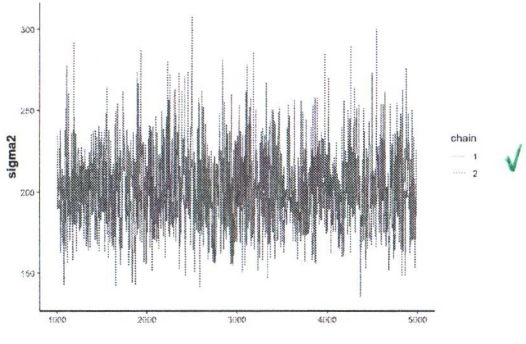



```
# Draw the traceplot corresponding to one or more Markov chains,
# providing a visual way to inspect
# sampling behavior and assess mixing across chains and convergence.

rstan::traceplot(fit3, pars = c("beta"), inc_warmup = FALSE)
```



✓

```
rstan::traceplot(fit3, pars='sigma2', inc_warmup = FALSE)
```



✓

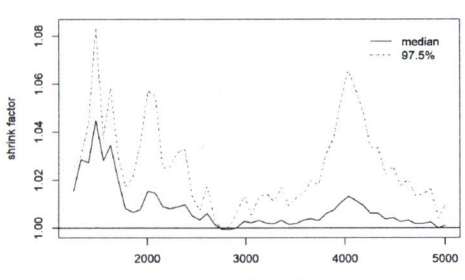
```
#---------------------------------------------------------------------------
# DIAGNOSTIC IN CODA ------------------------------------------------------
#---------------------------------------------------------------------------

coda_chain_reg2 <- As.mcmc.list(fit3, pars = c("beta", "sigma2"))
summary(coda_chain_reg2)
```

```
##
## Iterations = 1005:5000
## Thinning interval = 5
## Number of chains = 2
## Sample size per chain = 800
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean        SD  Naive SE Time-series SE
## beta[1]  -5.517964 19.334166 0.4833541      0.5668650
## beta[2]   1.059473  0.676141 0.0169035      0.0197404
## beta[3]   0.003549  0.005871 0.0001468      0.0001657
## sigma2  202.065899 24.930122 0.6232531      0.6459738
##
## 2. Quantiles for each variable:
##
##               2.5%        25%        50%       75%     97.5%
## beta[1] -43.463927 -1.829e+01  -5.297031 6.905e+00  32.05434
## beta[2]  -0.285381  6.042e-01   1.053805 1.500e+00   2.37464
## beta[3]  -0.008291 -2.265e-04   0.003608 7.464e-03   0.01529
## sigma2  158.354818  1.849e+02 200.247164 2.177e+02 256.20640
```

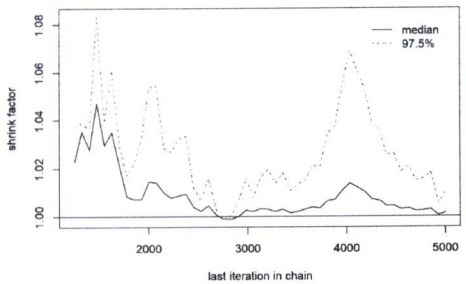
```
# Compute the gelman and rubin's convergence diagnostic

gelman.diag(coda_chain_reg2, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)
```

```
## Potential scale reduction factors:
##
##         Point est. Upper C.I.
## beta[1]      1.001       1.01
## beta[2]      1.001       1.01
## beta[3]      1.002       1.01
## sigma2       0.999       1.00
##
## Multivariate psrf
##
## 1
```

```
# â potential scale reduction factorâ is calculated for each variable in x,
# together with upper and lower confidence limits. Approximate convergence is
# diagnosed when the upper limit is close to 1. For multivariate chains, a multivariate
# value is calculated that bounds above the potential scale reduction factor for
# any linear combination of the (possibly transformed) variables

# This plot shows the evolution of Gelman and Rubinâs shrink factor as the number
# of iterations increases

gelman.plot(coda_chain_reg2[,1], confidence = 0.95)
```



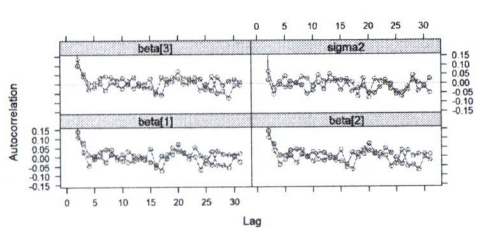

```
gelman.plot(coda_chain_reg2[,2], confidence = 0.95)
```




```
# Geweke's convergence diagnostic
geweke.diag(coda_chain_reg2, frac1=0.1, frac2=0.5)
```

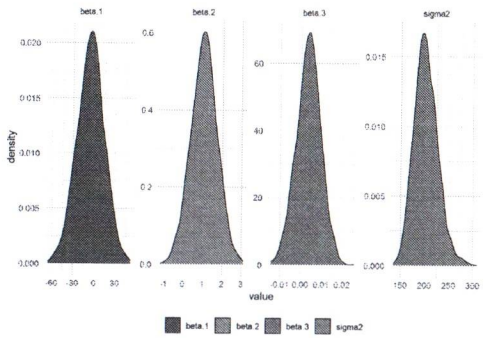
```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## beta[1] beta[2] beta[3]  sigma2
## -0.4626  0.3668 -0.4147 -0.8712
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## beta[1] beta[2] beta[3]  sigma2
## -1.5764  1.2696 -0.8863 -0.6318
```

```
# autocorrelation and plot
acfplot(coda_chain_reg2, lag.max = 30)
```



5/11/2020
11/12

```
# plots of the posteriors
plot_post <- fit3 %>%
  rstan::extract(c("beta", "sigma2")) %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free', nrow = 1) +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="bottom") +
  labs(fill = "")
```




```
#------------------------------------------------------------------------
# COMPARE THE MODELS ----------------------------------------------------
#------------------------------------------------------------------------

loo(fit2)
```

leave one out cross validation

```
##
## Computed from 1600 by 133 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -546.5  8.6
## p_loo         2.1  0.4
## looic      1093.0 17.2
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
loo(fit3)
```

lower is better

```
##
## Computed from 1600 by 133 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -545.0  9.3
## p_loo         3.9  0.7
## looic      1090.1 18.7
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

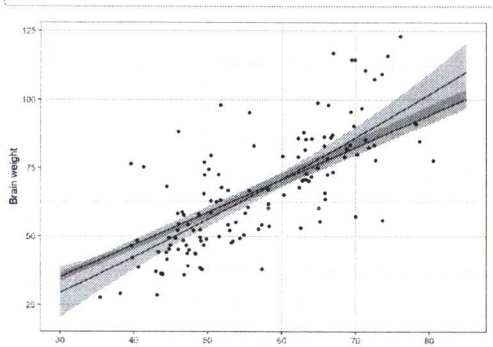
```
# look at loo-ic
# lower is better, second model is preferred

my_WAIC <- function(fit, param){

  llik   <- rstan::extract(fit, param)[[1]]
  p_WAIC <- sum(apply(llik, 2, var))
  lppd   <- sum(apply(llik, 2, function(x) log(mean(exp(x)))))
  WAIC   <- - 2 * lppd + 2 * p_WAIC
  return(WAIC)

}

my_WAIC(fit2, "log_lik")
```

```
## [1] 1093.025
```

```
my_WAIC(fit3, "log_lik")
```

```
## [1] 1090.068
```

```
# asymptotically equivalent

#------------------------------------------------------------------------
# NICE PLOT -------------------------------------------------------------
#------------------------------------------------------------------------

xseq <- seq(30, 85, length.out = 100)

df_temp <- t(apply(as.data.frame(rstan::extract(fit2, pars = c('beta0', 'beta1'), permuted = TRUE)),
                   1, function(x) x[1] + x[2] * xseq))
df_temp2 <- t(apply(as.data.frame(rstan::extract(fit3, pars = c('beta'), permuted = TRUE)),
                    1, function(x) x[1] + x[2] * xseq + x[3] * xseq^2))

df_plot <- data.frame(x = xseq, y = colMeans(df_temp),
                      ylow = apply(df_temp, 2, function(x) quantile(x, 0.025)),
                      yup  = apply(df_temp, 2, function(x) quantile(x, 0.975)),
                      z = colMeans(df_temp2),
                      zlow = apply(df_temp2, 2, function(x) quantile(x, 0.025)),
                      zup  = apply(df_temp2, 2, function(x) quantile(x, 0.975)))

ggplot(data.frame(x = regression_data$X, y = regression_data$Y)) +
  theme_bw() +
  geom_line(data = df_plot, aes(x = x, y = y), col = "blue") +
  geom_ribbon(data = df_plot, aes(x = x, ymin = ylow, ymax = yup), fill = "blue", alpha = 0.2) +
  geom_line(data = df_plot, aes(x = x, y = z), col = "red") +
  geom_ribbon(data = df_plot, aes(x = x, ymin = zlow, ymax = zup), fill = "red", alpha = 0.2) +
  geom_point(aes(x = x, y = y)) +
  xlab("SVL") +
  ylab("Brain weight")
```



model + uncertainty (in credible bands)

13/11/2020
1/9

Variable selection and shrinkage

Riccardo Corradin
November 13, 2020

Introduction

The model choice problem

Strategy: fix **prior probability** masses to all the possible models, then apply the Bayes theorem to compute the **posterior distribution**.

The **best models** (more than one) are those with highest posterior probability.

Let m be the index describing L models $M_1, \dots, M_L$:

- $P(m = j)$, **prior probability to choose model** M_j (typically $1/L$)
- Model M_j:
  - likelihood $p(\boldsymbol{Y} \mid \theta_j, M_j)$
  - prior $\pi(\theta_j \mid M_j) = \pi(\theta_j \mid m = j)$
  - θ_j vector of parameters

We set a prior on each model and then we check the posterior
How can we do it?

We set a prior conditioned on the model (and the likelihood too)

Idea: scan the entire model space.

The model choice problem^2

How it works...:

1. For any model M_j compute the posterior distribution of the parameter θ_j

$$\pi(\theta_j \mid \boldsymbol{Y}, M_j) = \frac{p(\boldsymbol{Y} \mid \theta_j, M_j)\pi(\theta_j \mid M_j)}{m(\boldsymbol{Y} \mid M_j)}$$

2. Compute the posterior probability masses for $M_1, \dots, M_L$

$$P(m = j \mid \boldsymbol{Y}) = \frac{m(\boldsymbol{Y} \mid M_j)P(m = j)}{m(\boldsymbol{Y})}, \quad j = 1, \dots, L$$

with

$$m(\boldsymbol{Y}) = \sum_{j=1}^{L} m(\boldsymbol{Y} \mid M_j)P(m = j)$$

3. Choose the model(s) with the highest $P(m = j \mid \boldsymbol{Y})$ and eventually compare these few models via predictive goodness-of-fit criteria

Problem of variable selection

In a regression world: given a dependent variable Y and a set of K potential regressors $X_1, \ldots, X_K$, the problem is to find the **best** model of the form

$$g(\mathbb{E}[Y \mid \boldsymbol{X}, \beta]) = \beta_1 X_1 + \cdots + \beta_K X_K$$

i.e. linear models or generalized linear models, where we have at most K different covariates.

Using the previous exhaustive strategy, **we have to estimate** 2^K models to scan the entire space. With $K = 12$ we need to estimate 4096. The problem is **intractable**.

We need **alternative strategies** to deal with the model choice problem.

Spike and slab in a nutshell

A Hierarchical Mixture Model for Variable Selection

We index each of these possible 2^K subset choices by the vector

$$\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_K)^\intercal$$

where $\gamma_i = 0$ **if** $\beta_i = 0$, $\gamma_i = 1$ **if** $\beta_i \neq 0$, and $\{\boldsymbol{\gamma}_s, s = 1, \ldots, 2^K\}$ is the set of all possible models. We model the uncertainty underlying variable selection augmenting the problem with $\boldsymbol{\gamma}$, using a mixture prior

$$\pi(\beta, \boldsymbol{\gamma}) = \pi(\beta \mid \boldsymbol{\gamma})\pi(\boldsymbol{\gamma})$$

where

$$\gamma_j \sim \text{Bernoulli}(\theta_j), \quad j = 1, \ldots, K, \quad \text{independent}$$

and θ_j is the probability that β_j is large enough to justify including X_j in the model.

Specifying the prior - Spike&Slab

This prior can also be used to put increased weight on parsimonious models by setting the θ_j small. The components of $\boldsymbol{\gamma}$ are apriori independent. Consider

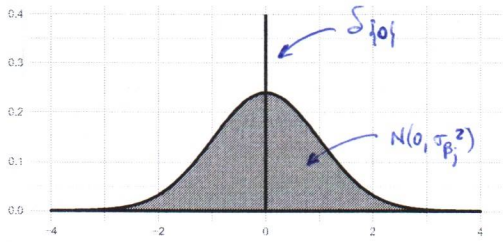
$$\begin{aligned} \beta_j \mid \gamma_j &\sim (1 - \gamma_j)\delta_{\{0\}} + \gamma_j N(0, \sigma^2_{\beta_j}) \\ \gamma_j \mid \theta_j &\sim \text{Bern}(\theta_j) \\ \theta_j &\sim \pi(\theta_j) \end{aligned}$$

→ $\gamma_j = 0 \Rightarrow$ all the mass in 0
$\gamma_j = 1 \Rightarrow$ all the mass in $N(0, \sigma^2_{\beta_j})$

where $\delta_{\{0\}}$ is a Dirac measure with mass in 0 (you may also consider approximations of it!) and $N(0, \sigma^2_{\beta_j})$ is a diffuse distribution.

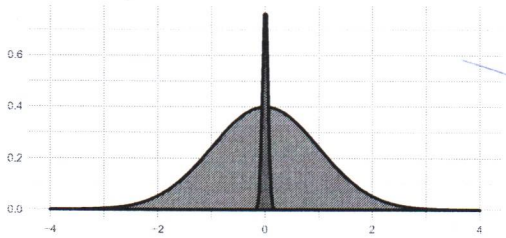
A common choice for $\pi(\theta_j)$ is an Unif$(0, 1)$ distribution.

What are we doing? - Spike&Slab



Remark: in **STAN** is not possible to define a Spike&Slab prior we can use only continuous distributions for the parameters.

Relax the mixture - SSVS



For example we can set them both as gaussians

We can relax the mixture prior distribution on the coefficient by considering a combination of two continuous distribution: one super concentrated and the other slab.

Specifying the prior - SSVS

Consider a **prior distribution** for β_j as

$$\begin{aligned} \beta_j \mid \sigma_j^2 &\sim N(0, \sigma_j^2) \\ \sigma_j^2 \mid c_j, \tau_j, \gamma_j &\sim (1-\gamma_j)\delta_{\tau_j^2} + \gamma_j \delta_{c_j^2 \tau_j^2} \\ \gamma_j \mid \theta_j &\sim \text{Bern}(\theta_j) \end{aligned}$$

→ if $\gamma_j = 1$ then the variance is $c_j^2 \tau_j^2$ otherwise it just τ_j^2
(c_j^2 = how much we amplify the variance in the slab component)

where $\delta_{\{x\}}$ is a Dirac measure with mass in x, and $N(0, \sigma_j^2)$ is a Gaussian (**diffuse**) distribution. We are combining together two diffuse distribution: a Gaussian with variance τ_j^2 and a Gaussian with variance $c_j^2 \tau_j^2$.

This prior looks like a relaxed version of the previous Spike&Slab prior.

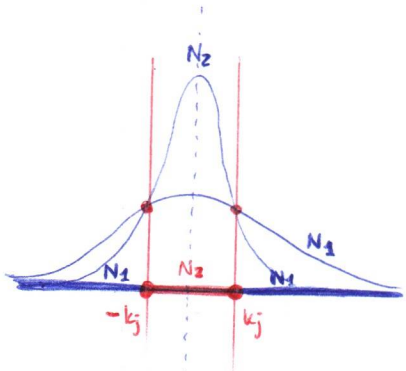
Specifying the prior - SSVS^2

Within the previous priors setup we have a mixture of two Gaussian distributions, that intersect at the points $\pm\kappa_j = \tau_j \epsilon_j$ where

$$\epsilon_j = \sqrt{2 \frac{\log(c_j) c_j^2}{c_j^2 - 1}}$$

The point κ_j can be regarded as a **threshold** for declaring practical significance in that all coefficients falling into the interval $(-\kappa_j, \kappa_j)$ can be interpreted as **'practically zero'**.

We use the previous priors in a practical example.



For both the cases (Spike & Slab / SSVS) we procede similarly:

Model estimation

Let

- $\pi(\gamma)$: prior for the model index parameter γ
- $\pi(\beta \mid \gamma)$: prior for coefficients within each model
- $p(\boldsymbol{Y} \mid \beta_\gamma, \gamma)$: likelihood (consider the product for the n subjects)

Model/covariate choice is made via the posterior of the model index parameter:

$$\pi(\gamma_j \mid \boldsymbol{Y}) = \frac{p(\boldsymbol{Y} \mid \gamma_j)\pi(\gamma_j)}{\sum_k p(\boldsymbol{Y} \mid \gamma_k)\pi(\gamma_j)}$$

= posterior distribution of including or not a variable in the model

Where

$$\pi(\boldsymbol{Y} \mid \gamma_j) = \int p(\boldsymbol{Y} \mid \gamma_j, \beta_\gamma)\pi(d\beta_\gamma \mid \gamma_j)$$

is the marginal density of $\boldsymbol{Y}$ in a model indexed by γ_j.

Criteria for selecting covariates

Given (a sample from) the posterior distribution $\pi(\gamma \mid Y)$ we have different strategies to select the covariates:

- **HPD**: choose the model with the **highest posterior probability**

$$\hat{\pi}(\gamma_j \mid \boldsymbol{Y}) = \frac{1}{n_{rep}} \sum_{t=1}^{n_{rep}} \mathbb{I}_{[\gamma^{(t)} = \gamma_j]}$$

= we include those which prob. (posterior) is higher than a threshold

- **MPM**: pick all variables with estimated marginal posterior inclusion probabilities

$$\hat{\pi}(\gamma_j \mid \boldsymbol{Y}) = \frac{1}{n_{rep}} \sum_{t=1}^{n_{rep}} \mathbb{I}_{[\gamma_j^{(t)} = 1]}$$

- **HS**: hard shrinkage, pick all variables such that 0 does **NOT** belong to the marginal posterior **credible interval** of the corresponding regression parameter, or to a CI centered around the posterior mean $\pm$ sd

Example:

Reach dataset

The Rotterdam Early Arthritis Cohort (dataset) study was initiated in 2004 to **investigate the development of rheumatoid arthritis** in patients with early manifestations of joint impairment.

Information regarding basic patient characteristics, serological measurements, and patterns of disease involvement at baseline has been gathered in **681 recruited patients**.

It is of interest to know which of the following 12 factors are potentially **associated with the development** of rheumatoid arthritis considered as a binary (yes/no) outcome.

Covariates:

Reach dataset[2]

- ACCP (cyclic citrullinated peptide antibody)
- age
- ESR (erythrocyte sedimentation rate, is the rate at which red blood cells sediment in a period of one hour)
- DC (duration of complaints in days)
- stiffness (duration of morning stiffness in minutes)
- RF (rheumatoid factor)
- gender
- Sym (symmetrical pattern of joint inflammation; yes/no)
- SJC (swollen joint count)
- TJC (tender joint count)
- BCPH (bilateral compression pain in hands; yes/no)
- BCPF (bilateral compression pain in feet; yes/no)

Reach dataset[3]

The standard approach to analyze these data would be to use **probit regression** combined with some off-the-shelf variable selection method. The F-to-out backward selection with $p_D \leq 0.05$ yields a model with the following variables:

ACCP, ESR, DC, Sym, SJC, and BCPH

The model with the most favorable value of the AIC selected after an exhaustive model evaluation contains two extra variables: RF and stiffness.

We consider a Bayesian probit regression, where the coefficients are selected via Spike&Slab priors.

$$\text{probit}(\mathbb{E}[Y_i \mid \boldsymbol{X}_i]) = \boldsymbol{X}_i^\intercal \boldsymbol{\beta}$$
$$\text{probit}(a) = \Phi^{-1}(a) \implies \mathbb{E}[Y_i \mid \boldsymbol{X}_i] = \Phi(\boldsymbol{X}_i^\intercal \boldsymbol{\beta})$$
$\Phi(\cdot)$ denote the CDF of a Gaussian distribution

15/26

Recap on the model

Model specification:

$$Y_i \mid \boldsymbol{X}_i, \boldsymbol{\beta} \sim \text{Bern}\left(\Phi(\boldsymbol{X}_i^\intercal \boldsymbol{\beta})\right), \quad i = 1, \ldots, n$$
$$\boldsymbol{\beta}_\gamma \mid \gamma \sim \ldots$$
$$\gamma_j \mid \theta_j \sim \text{Bern}(\theta_j), \quad j = 1, \ldots, K$$

depending on if we decide to consider Spike & Slab or SSVS

Prior elicitation:

- we set $\boldsymbol{b} = 0$ (vector), expected value for the distribution on β, i.e. considering the symmetry of the distribution we are not saying that a priori the effect is positive or negative.
- working with marginally standardized covariates
- eventual hyperpriors to elicitate the model

16/26

See the code!

Regularization

Brief introduction to regularization methods

Data: $\{(X_1, Y_1), \ldots, (X_n, Y_n)\}$. Model specification:

$$g(\mathbb{E}[Y_i \mid X_i]) = \boldsymbol{X}_i^{\intercal}\beta$$

Problem: the regression coefficients may explode, the best fit can be poor in terms of interpretability.

Maximum likelihood estimation:

$$\hat{\beta}_{ML} = \arg\max_{\beta} \left\{ \sum_{i=1}^{n} \log f\left(Y_i \mid \boldsymbol{X}_i\beta\right) \right\}$$

Penalized estimation:

$$\hat{\beta}_{penalized} = \arg\max_{\beta} \left\{ \sum_{i=1}^{n} \log f\left(Y_i \mid \boldsymbol{X}_i\beta\right) + \underbrace{h(\beta)}_{\text{shrinkage penalty}} \right\}$$

by including this term we can push the value of the parameters to zero

Brief introduction to regularization methods[2]

Different choices for the **shrinkage penalty** lead to different regularization methods:

- A L_1 norm leads to the **Lasso** method:

$$h(\beta) = \lambda \|\beta\|_1 = \lambda \sum_{j=1}^{K} |\beta_j|$$

- A L_2 norm leads to the **Ridge** method:

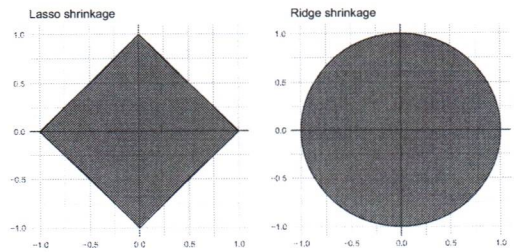
$$h(\beta) = \lambda \|\beta\|_2 = \lambda \sum_{j=1}^{K} \beta_j^2$$

- Combining together L_1 and L_2 norms we get the **Elastic Net**:

$$h(\beta) = \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2 = \lambda_1 \sum_{j=1}^{K} |\beta_j| + \lambda_2 \sum_{j=1}^{K} \beta_j^2$$

Brief introduction to regularization methods[3]

But what are we doing practically?



With the **Lasso** shrinkage we are forcing the parameter to be inside a square, with the **Ridge** inside a circle. With the **Elastic Net** we are combining the two penalties.

Brief introduction to regularization methods[4]

The **Lasso** penalty is known for its sparsity property, i.e. is forcing some coefficients quickly to zero.

The **Ridge** penalty is known for its grouping property, in the sense that is able to retain groups of strongly correlated variables.

The **Elastic Net** is a good compromise between this two properties.

But what happens in **Bayesian** statistics?

Regularization methods with Bayes

The choice of β which minimize the **Ridge** penalty is also the value which maximize the probability of independent random variables distributed as a **Gaussian distribution**. Let assume that $\beta \mid \lambda \sim N(0, \boldsymbol{I}_K/\lambda)$.

$$\begin{aligned}\tilde{\beta} &= \arg\max_{\beta} \left\{ (2\pi)^{-K/2} \lambda^{K/2} e^{-\frac{\lambda}{2}\beta^T\beta} \right\} \\ &= \arg\max_{\beta} \left\{ -\frac{\lambda}{2}\beta^T\beta \right\} \\ &= \arg\min_{\beta} \left\{ \lambda \sum_{j=1}^{K} \beta_j^2 \right\}\end{aligned}$$

where λ is a common penalty.

Regularization methods with Bayes[2]

Thus the **Ridge** regression correspond to the maximum a posteriori (MAP) estimation of a model with a Gaussian distribution on the regression coefficients and a particular structure for the variance.

The specification for the model is then

$$\begin{aligned} Y_i \mid \boldsymbol{X}_i\beta &\sim p(Y_i \mid \boldsymbol{X}_i, \beta), \quad i = 1, \dots n \\ \beta_j \mid \lambda &\sim N(0, 1/\lambda), \quad j = 1, \dots K \end{aligned}$$

About the parameter λ:

- we can fix the parameter to a specific value
- we can specify an hyperprior distribution, for example

$$\lambda \sim Gamma(a_\lambda, b_\lambda)$$

Regularization methods with Bayes[3]

The choice of β which minimize the **Lasso** penalty is also the value which maximize the probability of independent random variables distributed as a **Laplace distribution**. Let assume that $\beta_j \mid \lambda \sim Lap(0, 1/\lambda_j)$.

$$\begin{aligned}\tilde{\beta} &= \arg\max_{\beta} \left\{ \prod_{j=1}^{K} \frac{\lambda_j}{2} e^{-\lambda_j|0-\beta_j|} \right\} \\ &= \arg\max_{\beta} \left\{ \sum_{j=1}^{K} -\lambda_j|\beta_j| \right\} \\ &= \arg\min_{\beta} \left\{ \sum_{j=1}^{K} \lambda_j|\beta_j| \right\} \\ &= \arg\min_{\beta} \left\{ \lambda \sum_{j=1}^{K} |\beta_j| \right\}\end{aligned}$$

where the last holds assuming a common penalty.

Regularization methods with Bayes[4]

Thus the **Lasso** regression correspond to the maximum a posteriori (MAP) estimation of a model with a Laplace distribution on the regression coefficients.

The specification for the model is then

$$\begin{aligned} Y_i \mid \boldsymbol{X}_i\beta &\sim p(Y_i \mid \boldsymbol{X}_i, \beta), \quad i = 1, \dots n \\ \beta_j \mid \lambda &\sim Lap(0, 1/\lambda), \quad j = 1, \dots K \end{aligned}$$

About the parameter λ:

- we can fix the parameter to a specific value
- we can specify an hyperprior distribution, Park and Casella suggested

$$\lambda^2 \sim Gamma(a_\lambda, b_\lambda)$$

Regularization methods with Bayes[3]

In a similar spirit, the **Elastic Net** in the Bayesian field correspond to a particular choice of the prior distribution on the regression coefficients, such that the MAP estimation of the parameters is equivalent to the ones that minimize the **Elastic Net** penalty.

The prior distribution for β is an hierarchical one, defined as:

$$\beta_j \mid \tau_j, \sigma^2 \sim N\left(0, \left[\frac{\lambda_2}{\sigma^2}\frac{\tau_j}{\tau_j - 1}\right]^{-1}\right)$$
$$\tau_j \mid \sigma^2 \sim \text{trGamma}\left(0.5, \frac{\lambda_1^2}{8\lambda_2\sigma^2}, 1, \infty\right)$$

where trGamma(a, b, c, d) correspond to the Gamma distribution with shape parameter a and rate parameter b, truncated on (c, d).

Regularization methods with Bayes[2]

The **Elastic Net** correspond to the maximum a posteriori (MAP) estimation of a model with the previous prior specification on the regression coefficients.

The specification for the model is then

$$Y_i \mid \boldsymbol{X}_i\beta \sim p(Y_i \mid \boldsymbol{X}_i, \beta), \quad i = 1, \ldots n$$
$$\ldots \sim \ldots$$

plus the previous prior for the coefficients β. About the parameter $\lambda_1.\lambda_2$:

- we can fix the parameter to a specific value
- we can specify an hyperprior distribution, Park and Casella suggested

$$\lambda_j^2 \sim Gamma(a_{\lambda_j}, b_{\lambda_j}), \quad j = 1, 2$$

See the code!

```
## SSVS_probit.txt

# ############################################################
# # STOCHASTIC SEARCH VARIABLE SELECTION: probit REGRESSION ###
# ############################################################
#
# model {
#
# #likelihood
# for (i in 1:N){
#   mu[i] <- beta0 + inprod(X[i,], beta)
#       z[i]  <- phi(mu[i])
#   Y[i] ~ dbern(z[i]) # usual probit regression model
# }
#
# # tracing the visited model
# for (j in 1:p){
#   TempIndicator[j] <- g[j] * pow(2, j)
# }
# mdl<- 1 + sum(TempIndicator[]) # model index in binary coding
#
#
# c1 <- 1 / (pow(tau_ss, 2)) #reciprocal of the spike variance (precision)
# c2 <- c1 / (pow(c_ss, 2)) # reciprocal of the slab variance
#
# ## Since the Gaussian distribution
# ## is parametrized in terms of precision parameter
# beta0 ~ dnorm(0, 0.001)
#
# for(j in 1:p){
#   bprior[j]<-0
#   tprior[j] <-equals(g[j],0) * c1 + equals(g[j],1) * c2
#       # this is c1 if the first condition is met,
#       #  c2 if the second is met
#   beta[j] ~ dnorm(bprior[j],tprior[j])
#   g[j] ~ dbern(0.5)
# }
# }
```

$g_j = \gamma_j$

```
## SpSi_probit.txt

# ############################################################
# # STOCHASTIC SEARCH VARIABLE SELECTION: probit REGRESSION ###
# ############################################################
#
# model {
#
# #likelihood
# for (i in 1:N){
#   mu[i] <- beta0 + inprod(X[i,], beta)
#       z[i]  <- phi(mu[i])
#   Y[i] ~ dbern(z[i]) # usual probit regression model
# }
#
# # tracing the visited model
# for (j in 1:p){
#   TempIndicator[j]<- g[j] * pow(2, j)
# }
# mdl <- 1 + sum(TempIndicator[]) # model index in binary coding
#
# ##Since the Gaussian distribution
# ## is parametrized in terms of precision parameter
# beta0 ~ dnorm(0, 0.001)
#
# for(j in 1:p){
#   tprior[j] <- 1 / var_beta[j]
#   bprior[j] <- 0
# }
#
# for(j in 1:p){
#   beta_temp[j] ~ dnorm(bprior[j], tprior[j])
#   g[j] ~ dbern(theta[j])
#   theta[j] ~ dunif(0,1)
#   beta[j] <- g[j] * beta_temp[j]
# }
# }
```

$g_j = \gamma_j$

```
## BL_probit.stan

# //################################################
# //### BAYESIAN LASSO: PROBIT REGRESSION ##########
# //################################################
#
# /////////////////////////// DATA /////////////////////////////////////
# data {
#   int<lower = 0> N;       // number of data
#   int<lower = 0> p;       // number of covariates (without the intercept)
#   int<lower = 0, upper = 1> Y[N];   // response vector
#   matrix[N, p] X;           // design matrix
#
# }
#
# ////////////////////// PARAMETERS ///////////////////////////////////
# parameters {
#   vector[p] beta;        // regression coefficients
#   real beta0;            // intercept
#   real<lower = 0> lambda2;
# }
#
#
# ////////////////// MODEL /////////////////////////
# model {
#
# // Likelihood
#   for (s in 1:N)
#   {
#       // print(" s = ", s);
#       Y[s] ~ bernoulli(Phi(beta0 + row(X, s)*beta));
#   }
#
# // Prior
#   // beta
#   beta0 ~ normal(0.0, 50.0);   // vague prior: (mean, sd)
#
#   for (j in 1:p)
#   {
#       beta[j] ~ double_exponential(0.0, 1.0 / (pow(lambda2,0.5)));
#   }
#
#   lambda2 ~ exponential(0.1);
# }
```

```
## BEN_probit.stan
#
# //#########################################################
# //### BAYESIAN ELASTIC NET: PROBIT REGRESSION ############
# //#########################################################
#
# /////////////////////////// DATA /////////////////////////////////////
# data {
#   int<lower = 0> N;       // number of data
#   int<lower = 0> p;       // number of covariates (without the intercept)
#   int<lower = 0, upper = 1> Y[N];   // response vector
#   matrix[N, p] X;           // design matrix
#
# }
#
# ////////////////////// PARAMETERS ///////////////////////////////////
# parameters {
#   vector[p] beta;        // regression coefficients
#   real beta0;            // intercept
#   vector<lower = 1>[p] tau;
#   real<lower = 0> a1;
#   real<lower = 0> a2;
# }
#
#
# ////////////////// MODEL /////////////////////////
# model {
#
# // Likelihood
#   for (s in 1:N)
#   {
#       // print(" s = ", s);
#       Y[s] ~ bernoulli(Phi(beta0 + row(X, s)*beta));
#   }
#
# // Prior
#   // beta
#   beta0 ~ normal(0.0, 50.0);   // vague prior: (mean, sd)
#
#   for (j in 1:p)
#   {
#       beta[j] ~ normal(0.0, pow((tau[j]-1)/(tau[j]*a2), 0.5) );
#   }
#
#   // tau
#   for (m in 1:p)
#   {
#       tau[m] ~ gamma(0.5, (pow(a1,2)) / (8*a2) ) T[1, ];   // truncated distribution!!! (see the manua
l)
#   }
#
#   a1 ~ exponential(0.1);  // mean: 10
#   a2 ~ exponential(0.1);
#
# }
```

```
## BL_probit2.stan

# //################################################
# //### BAYESIAN LASSO: PROBIT REGRESSION ##########
# //################################################
#
# /////////////////////////// DATA /////////////////////////////////////
# data {
#   int<lower = 0> N;       // number of data
#   int<lower = 0> p;       // number of covariates (without the intercept)
#   int<lower = 0, upper = 1> Y[N];   // response vector
#   matrix[N, p] X;           // design matrix
#   real<lower = 0> lambda;
# }
#
# ////////////////////// PARAMETERS ///////////////////////////////////
# parameters {
#   vector[p] beta;        // regression coefficients
#   real beta0;            // intercept
# }
#
#
# ////////////////// MODEL //////////////////////////
# model {
#
# // Likelihood
#   for (s in 1:N)
#   {
#       // print(" s = ", s);
#       Y[s] ~ bernoulli(Phi(beta0 + row(X, s)*beta));
#   }
#
# // Prior
#   // beta
#   beta0 ~ normal(0.0, 50.0);   // vague prior: (mean, sd)
#
#   for (j in 1:p)
#   {
#       beta[j] ~ double_exponential(0.0, 1.0 / lambda);
#   }
#
# }
```

```
## BR_probit2.stan

# //#####################################################
# //### BAYESIAN ELASTIC NET: PROBIT REGRESSION ##########
# //#####################################################
#
# /////////////////////////// DATA /////////////////////////////////////
# data {
#   int<lower = 0> N;       // number of data
#   int<lower = 0> p;       // number of covariates (without the intercept)
#   int<lower = 0, upper = 1> Y[N];   // response vector
#   matrix[N, p] X;           // design matrix
#   real<lower = 0> lambda;
# }
#
# ////////////////////// PARAMETERS ///////////////////////////////////
# parameters {
#   vector[p] beta;        // regression coefficients
#   real beta0;            // intercept
# }
#
#
# ////////////////// MODEL //////////////////////////
# model {
#
# // Likelihood
#   for (s in 1:N)
#   {
#       // print(" s = ", s);
#       Y[s] ~ bernoulli(Phi(beta0 + row(X, s)*beta));
#   }
#
# // Prior
#   // beta
#   beta0 ~ normal(0.0, 50.0);   // vague prior: (mean, sd)
#
#   for (j in 1:p)
#   {
#       beta[j] ~ normal(0.0, pow(1.0 / lambda, 0.5));
#   }
#
# }
```

```
#####################################################
###########   VARIABLE SELECTION              #####
#####################################################

library(doParallel)

# R & STAN are friends!
library(rstan)
library(rjags)
library(coda)

# for plots
library(ggplot2)
library(tidyr)
library(dplyr)
library(purrr)
library(ggsci)
require(gplots)
require(ggpubr)

################## REACH data ###################################
## Rheumatoid arthritis is an autoimmune disease characterized by
## chronic synovial inflammation and destruction of cartilage and bone
## in the joints.
# The Rotterdam Early Arthritis Cohort (REACH) study was
## initiated in 2004 to investigate the development of rheumatoid arthritis
## in patients with early manifestations of joint impairment (compromissione articolare).
## Information regarding basic patient characteristics, serological
## measurements, and patterns of disease involvement at baseline has
## been gathered in 681 recruited patients. It is of interest
## to know which of the following 12 factors are potentially
## associated with the development of rheumatoid arthritis considered as
## a binary (yes/no) outcome:
## ACCP (cyclic citrullinated peptide antibody),
## age,
## ESR (erythrocyte sedimentation rate, is the rate at which red blood cells sediment in a period of one
 hour),
## DC (duration of complaints in days),
## stiffness (duration of morning stiffness in minutes),
## RF (rheumatoid factor),
## gender,
## Sym (symmetrical pattern of joint inflammation; yes/no),
## SJC (swollen joint count),
## TJC (tender joint count),
## BCPH (bilateral compression pain in hands; yes/no),
## BCPF (bilateral compression pain in feet; yes/no).
##
## The standard approach to analyze these data would be to use
## logistic/probit regression combined with some off-the-shelf variable
## selection method. The F-to-out backward selection with p_D 0.05 yields a
## model with the following variables:
## ---------------> ACCP, ESR, DC, Sym, SJC, and BCPH.
## The model with the most favorable value of the AIC selected after an
## exhaustive model evaluation contains two extra variables: RF and stiffness.
## Which of these models provide the best approximation to the true
## underlying relationships is, if at all possible, difficult to assess.

reach <- read.table("REACH_data.txt", header=T)
names(reach)
head(reach)
reach$gender <- reach$gender - 1

sub.idx = 1:12
num.data = 681

X <- as.matrix(reach[1:num.data,sub.idx])
Y <- as.vector(reach[1:num.data,13])

N <- dim(X)[1]
p <- dim(X)[2]

#########################################################################
################### Spike and Slab ######################################
#########################################################################

## data to pass to JAGS (see the code in SSVS_probit.bug)
data_JAGS_SpSl <-list(N = N, p = p, Y = Y, X = as.matrix(X),
                      var_beta = rep(1, p))

## A list of initial value for the MCMC algorithm
# that WinBUGS will implement
inits = function() {
  list(beta0 = 0.0, beta_temp = rep(0,p), g = rep(0,p), theta = rep(0.5, p),
       .RNG.seed = 321, .RNG.name = 'base::Wichmann-Hill')
}

model=jags.model("SpSl_probit.bug",
                 data = data_JAGS_SpSl,
                 n.adapt = 1000,
                 inits = inits,
                 n.chains = 1)

# if we want to perform a larger burn in with not adaptation.
update(model,n.iter=500)

# Posterior parameters WinBUGS  has to save NO "sigma2" here
param <- c("beta0", "beta", "g", "mdl")

# number of iterations & thinning
nit <- 50000
thin <-10

##The command coda.samle() calls jags from R passing the data and initial value just defined
output <- coda.samples(model = model,
                       variable.names = param,
                       n.iter = nit,
                       thin = thin)

# save(output,file='SpSl.dat')   # we save the chain
load('SpSl.dat')

# the output is an mcmc object of the library coda
str(output)

###We give a look at the trace plot and at the density summary
### of the posterior chains for each of the parameters

plot(output,ask=T)

# summary
summary(output)

### To work with the posterior chains it is better
### to cast the output to be an array object of R

output <- as.matrix(output)

###Some variable selection thecniques:
# The median probability moodel (MPM)
# pick variables with estimated posterior inclusion probabilities
# higher than 0.5
# Notice that the estimated posterior inclusion probabilities are the
# posterior means of the gamma variables (in the code we called g)
head(output)

##We save the posterior chain of the inclusion variable in post_g
post_g <-as.matrix(output[,14:25])
apply(post_g,2,"mean")
post_mean_g <- apply(post_g,2,"mean")

p1 <- data.frame(value = post_mean_g, var = colnames(X)) %>%
  ggplot(aes(y = value, x = var, fill = var)) +
  geom_bar(stat="identity") +
  geom_hline(mapping = aes(yintercept = .5), col = 2, lwd = 1.1) +
  coord_flip() +
  theme_minimal() +
  theme(legend.position="none") +
  ylab("posterior inclusion probabilities") +
  xlab("")
p1
```

← posterior probability of including a specific covariate in the model (we take/consider ≥ 0.5)

```
# we will compare later the model with other methods
mp_SpSl <- as.vector(which(post_mean_g > 0.5))
```

```
post_mean_g[mp_SpSl]

# ANOTHER WAY TO CHOOSE THE MODEL:
# Highest posterior density model (HPD)
# pick a model with the highest estimated posterior probability

# Recall that we have represented the model index
# using a binary coding as
# mdl=1+2^g1+...+2^gp
# the visited model are saved in the chain

plot(output[,"mdl"], pch = 20)

# for example at iteration 10 the chain explored the
# model

output[10,"mdl"]

### We start to analyze how many models have been visited
## by the posterior chain:
length(unique( output[,"mdl"]))

# model visited out of 4096
## Now we compute the posterior frequency of the visited  models
visited_models<-table(output[,"mdl"])
visited_models

## HPD model
# getting the unique profiles
# and sort the results
unique_model <- unique(post_g, MARGIN  = 1)
freq <- apply(unique_model, 1, function(b) sum(apply(post_g, MARGIN = 1, function(a) all(a == b))))
cbind(unique_model[order(freq,decreasing = T),], sort(freq,decreasing = T))

# the HPD model is
colnames(X)[as.logical(unique_model[which.max(freq),])]
HDP_SpSl <- c(1:12)[as.logical(unique_model[which.max(freq),])]

#########################################################################
################### SSSVS ###############################################
#########################################################################

###Parameters of the  spike slab prior Set 1
c_ss <- 100
intersect <- 0.05
tau_ss <- intersect / sqrt(2 * log(c_ss) * c_ss^2/(c_ss^2 - 1))

## With this choice of hyperameter, c_ss and intersection
## the variance of the quasi-spike prior is
tau_ss^2

##While the variance of the slab is
(tau_ss*c_ss)^2

#---
curve(dnorm(x, 0, tau_ss), xlim = c(-5, 5))
curve(dnorm(x, 0, tau_ss * c_ss), xlim = c(-5, 5), add = T, col = 2)
# the spike is super concentrated

## data to pass to JAGS (see the code in SSVS_probit.bug)
data_JAGS_1 <-list(N = N, p = p, Y = Y, X = as.matrix(X),
                   tau_ss = tau_ss, c_ss = c_ss)

## A list of initial value for the MCMC algorithm
# that WinBUGS will implement
inits = function() {
  list(beta0 = 0.0, beta = rep(0,p), g = rep(0,p),
       .RNG.seed = 321, .RNG.name = 'base::Wichmann-Hill')
}

model=jags.model("SSVS_probit.bug",
                 data = data_JAGS_1,
                 n.adapt = 1000,
                 inits = inits,
                 n.chains = 1)

# if we want to perform a larger burn in with not adaptation.
update(model,n.iter=500)

# Posterior parameters WinBUGS  has to save NO "sigma2" here
param <- c("beta0", "beta", "g", "mdl")

# Some other information sfor the
# MCMC algorihm
# number of iterations
nit <- 50000

#thinning
thin <-10

##The command coda.samle() calls jags from R passing the data and initial value just defined
output <- coda.samples(model = model,
                       variable.names = param,
                       n.iter = nit,
                       thin = thin)

# save(output,file='ssvs_1.dat')   # we save the chain
load('ssvs_1.dat')

# the output is an mcmc object of the library coda
str(output)

###We give a look at the trace plot and at the density summary
### of the posterior chains for each of the parameters

plot(output,ask=T)

# summary
summary(output)

### To work with the posterior chains it is better
### to cast the output to be an array object of R

output <- as.matrix(output)

###Some variable selection thecniques:
# The median probability moodel (MPM)
# pick variables with estimated posterior inclusion probabilities
# higher than 0.5
# Notice that the estimated posterior inclusion probabilities are the
# posterior means of the gamma variables (in the code we called g)
head(output)

##We save the posterior chain of the inclusion variable in post_g
post_g <-as.matrix(output[,14:25])
post_mean_g <- apply(post_g,2,"mean")

p2 <- data.frame(value = post_mean_g, var = colnames(X)) %>%
  ggplot(aes(y = value, x = var, fill = var)) +
  geom_bar(stat="identity") +
  geom_hline(mapping = aes(yintercept = .5), col = 2, lwd = 1.1) +
  coord_flip() +
  theme_minimal() +
  theme(legend.position="none") +
  ylab("posterior inclusion probabilities") +
  xlab("")
p2

# we will compare later the model with other methods
mp_SSV1 <- as.vector(which(post_mean_g > 0.5))
post_mean_g[mp_SSV1]

# ANOTHER WAY TO CHOOSE THE MODEL:
# Highest posterior density model (HPD)
# pick a model with the highest estimated posterior probability

# Recall that we have represented the model index
# using a binary coding as
# mdl=1+2^g1+...+2^gp
# the visited model are saved in the chain
plot(output[,"mdl"], pch = 20)

# for example at iteration 10 the chain explored the
# model
```

```
output[10,"mdl"]

### We start to analyze how many models have been visited
## by the posterior chain:
length(unique( output[,"mdl"]))

# model visited out of 4096
## Now we compute the posterior frequency of the visited  models
visited_models<-table(output[,"mdl"])
visited_models

# HPD model
# getting the unique profiles
# and sort the results
unique_model <- unique(post_g, MARGIN  = 1)
freq <- apply(unique_model, 1, function(b) sum(apply(post_g, MARGIN = 1, function(a) all(a == b))))
cbind(unique_model[order(freq,decreasing = T),], sort(freq,decreasing = T))

# the HPD model is
colnames(X)[as.logical(unique_model[which.max(freq),])]
HDP_SSV1 <- c(1:12)[as.logical(unique_model[which.max(freq),])]

#---------------------- SCENARIO 2 ----------------------------------

###Parameters of the  spike slab prior Set 2
c_ss <- 100
intersect <- 0.1
tau_ss <- intersect / sqrt(2 * log(c_ss) * c_ss^2 / (c_ss^2 - 1))

## With this choice of hyperameter, c_ss and intersection
## the variance of the quasi-spike prior is
tau_ss^2

##While the variance of the slab is
(tau_ss*c_ss)^2

#---
curve(dnorm(x, 0, tau_ss), xlim = c(-5, 5))
curve(dnorm(x, 0, tau_ss * c_ss), xlim = c(-5, 5), add = T, col = 2)
# the spike is super concentrated

## data to pass to JAGS (see the code in SSVS_probit.bug)
data_JAGS_2 <-list(N = N, p = p, Y = Y, X = as.matrix(X),
                   tau_ss = tau_ss, c_ss = c_ss)

## A list of initial value for the MCMC algorithm
# that WinBUGS will implement
inits_2 = function() {
  list(beta0 = 0.0, beta = rep(0,p), g = rep(0,p),
       .RNG.seed = 321, .RNG.name = 'base::Wichmann-Hill')
}

model=jags.model("SSVS_probit.bug",
                 data = data_JAGS_2,
                 n.adapt = 1000,
                 inits = inits_2,
                 n.chains = 1)

# if we want to perform a larger burn in with not adaptation.
update(model,n.iter=500)

# Posterior parameters WinBUGS  has to save NO "sigma2" here
param <- c("beta0", "beta", "g", "mdl")

# Some other information sfor the
# MCMC algorihm
# number of iterations
nit <- 50000

#thinning
thin <-10

##The command coda.samle() calls jags from R passing the data and initial value just defined
output <- coda.samples(model = model,
                       variable.names = param,
                       n.iter = nit,
                       thin = thin)

# save(output,file='ssvs_2.dat')   # we save the chain
load('ssvs_2.dat')

output <- as.matrix(output)

## We save the posterior chain of the inclusion variable in post_g
post_g <-as.matrix(output[,14:25])
apply(post_g,2,"mean")
post_mean_g <- apply(post_g,2,"mean")

p3 <- data.frame(value = post_mean_g, var = colnames(X)) %>%
  ggplot(aes(y = value, x = var, fill = var)) +
  geom_bar(stat="identity") +
  geom_hline(mapping = aes(yintercept = .5), col = 2, lwd = 1.1) +
  coord_flip() +
  theme_minimal() +
  theme(legend.position="none") +
  ylab("posterior inclusion probabilities") +
  xlab("")
p3

# we will compare later the model with other methods
mp_SSV2 <- as.vector(which(post_mean_g > 0.5))
post_mean_g[mp_SSV2]

### We start to analyze how many models have been visited
## by the posterior chain:
length(unique( output[,"mdl"]))

# model visited out of 4096
## Now we compute the posterior frequency of the visited  models
visited_models <- table(output[,"mdl"])
visited_models

# HPD model
# getting the unique profiles
# and sort the results
unique_model <- unique(post_g, MARGIN  = 1)
freq <- apply(unique_model, 1, function(b) sum(apply(post_g, MARGIN = 1, function(a) all(a == b))))
cbind(unique_model[order(freq,decreasing = T),], sort(freq,decreasing = T))

# the HPD model is
colnames(X)[as.logical(unique_model[which.max(freq),])]
HDP_SSV2 <- c(1:12)[as.logical(unique_model[which.max(freq),])]

#---------------------------------------------------------------------

ggarrange(p1, p2, p3, nrow = 1)

#########################################################################
################### BAYESIAN LASSO ######################################
#########################################################################

data_win <- list(N = N, p = p, Y = Y, X = as.matrix(X))
inits <- function()
{
  list(beta0 = 0, beta = rep(0.1,p), lambda2 = 20)
}


################## FIT THE MODEL
# WARNING: the model is a bit slow to run...
BL_model = stan(file = "BL_probit.stan", data = data_win,
                chains = 1, iter = 25000, warmup = 5000, thin= 5, seed = 199,
                init = inits, control = list(max_treedepth = 20, adapt_delta = 0.95))
# save(BL_model, file="BL_model.dat")
load("BL_model.dat")

is(BL_model)
names(BL_model)
summary(BL_model)

# shrinkage
shr_param <- rstan::extract(BL_model, pars = c("lambda2"), permuted = TRUE)
rstan::traceplot(BEN_model, pars = c("lambda2"))
```

```
ggplot(data.frame(x = shr_param, y = as.factor("lambda2"))) +
  geom_density(aes(lambda2, fill = y)) +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")

# Extract values of the chain
par <- rstan::extract(BL_model, pars = c("beta0", "beta"), permuted = TRUE)
is(par)
names(par)
beta0 <- par$beta0;
length(beta0)
beta <- as.matrix(par$beta)
dim(beta)

#### Variable selection:
# compute the 95% posterior credible interval for beta
CI_beta = apply(beta, 2, quantile, c(0.025, 0.975))
CI_beta

# If the credibility interval does not contain 0
# then I keep the variable

idx_cov_BL = NULL
for(l in 1:p){
  if(CI_beta[1,l]<0 && CI_beta[2,l]>0)
  {
    cat("*** variable ", colnames(reach)[l], " excluded \n")
  }
  else
  {
    cat("*** variable ", colnames(reach)[l], " included \n")
    idx_cov_BL = c(idx_cov_BL, l)
  }

}

mean_beta_post <- apply(beta, 2, "mean")
mean_beta_post

#-------------
# boxplot

data.frame(x = as.vector(beta), var = rep(colnames(reach)[1:12], each = nrow(beta))) %>%
  ggplot(aes(x = var, y = x, fill = var)) +
  geom_boxplot() +
  theme_bw() +
  theme(legend.position = "null") +
  geom_hline(aes(yintercept = 0), col = 2, lty = 2) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  xlab("") +
  ylab("")

## PREDICTION in the PROBABILITY SCALE:
Xnew <- X[1,]
G <- 4000 # num iteration
out <- numeric(G)
for(g in 1:G) out[g] <- pnorm(sum(Xnew*beta[g,]), mean = 0, sd = 1 )

data.frame(out = out) %>%
  ggplot(aes(out)) +
  geom_histogram(bins = 40, alpha = 0.6, col = 1) +
  geom_vline(mapping = aes(xintercept = 1), col = 2, lwd = 1.1) +
  theme_minimal() +
  theme(legend.position="none") +
  ylab("predicted probability") +
  xlab("")

# EVALUATE THE PREDICTION ERROR ON THE OBSERVED SAMPLE:
error = 0
subN = 100

plot(1:subN, rep(0.5, subN), col = "white", ylim = c(-0.01, 1.01), xlab = "", ylab = "")
for(i in 1:subN){
  Xnew = X[i,]
  for(g in 1:G) # loop over the iterations
    out[g] = pnorm(sum(Xnew*beta[g,]), mean = 0, sd = 1 )
  error = error + abs(Y[i] - ifelse(mean(out) > 0.5, 1, 0))
  segments(i, quantile(out, 0.05), i, quantile(out, 0.95), col = "royalblue", lwd = 2)
  points( i, Y[i], col = 'darkred', pch = 19 )
  points( i, mean(out), col = 'royalblue', pch = 19 )
  message("*** Subject ", i)
}
abline(h = 0.5)
errBL <- error/subN

##############################################################################
################### BAYESIAN ELASTIC NET #####################################
##############################################################################

data_win <- list(N = N, p = p, Y = Y, X = as.matrix(X))
inits <- function()
{
  list(beta0 = 0, beta = rep(0.1,p), tau = rep(2, p), a1 = 10, a2 = 20)
}


################## FIT THE MODEL
# WARNING: the model is a bit slow to run...
BEN_model = stan(file = "BEN_probit.stan", data = data_win,
     chains = 1, iter = 25000, warmup = 5000, thin= 5, seed = 199,
     init = inits, control = list(max_treedepth = 20, adapt_delta = 0.95))
# save(BEN_model, file="BEN_model.dat" )
load("BEN_model.dat")

is(BEN_model)
names(BEN_model)
summary(BEN_model)

# shrinkage
shr_param <- rstan::extract(BEN_model, pars = c("a1", "a2"), permuted = TRUE)
rstan::traceplot(BEN_model, pars = c("a1", "a2"))

plot_post <- shr_param %>%
  as_tibble() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="bottom")

# Extract values of the chain
par <- rstan::extract(BEN_model, pars = c("beta0", "beta"), permuted = TRUE)
is(par)
names(par)
beta0 <- par$beta0;
length(beta0)
beta <- as.matrix(par$beta)
dim(beta)

#### Variable selection:
# compute the 95% posterior credible interval for beta
CI_beta = apply(beta, 2, quantile, c(0.025, 0.975))
CI_beta

# If the credibility interval does not contain 0
# then I keep the variable

idx_cov_BEN = NULL
for(l in 1:p){
  if(CI_beta[1,l]<0 && CI_beta[2,l]>0)
  {
    cat("*** variable ", colnames(reach)[l], " excluded \n")
  }
  else
  {
    cat("*** variable ", colnames(reach)[l], " included \n")
    idx_cov_BEN = c(idx_cov_BEN, l)
  }
```

```
}

mean_beta_post <- apply(beta, 2, "mean")
mean_beta_post

#--------
# boxplot

data.frame(x = as.vector(beta), var = rep(colnames(reach)[1:12], each = nrow(beta))) %>%
  ggplot(aes(x = var, y = x, fill = var)) +
  geom_boxplot() +
  theme_bw() +
  theme(legend.position = "null") +
  geom_hline(aes(yintercept = 0), col = 2, lty = 2) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  xlab("") +
  ylab("")

# NOTE: I can also choose a different interval (80%, 90%)
# ---> the smaller the degree is, the less I am restrictive in choosing covariates

## PREDICTION in the PROBABILITY SCALE:
Xnew <- X[1,]
G <- 4000 # num iteration
out <- numeric(G)
for(g in 1:G) out[g] <- pnorm(sum(Xnew*beta[g,]), mean = 0, sd = 1 )

data.frame(out = out) %>%
  ggplot(aes(out)) +
  geom_histogram(bins = 40, alpha = 0.6, col = 1) +
  geom_vline(mapping = aes(xintercept = 1), col = 2, lwd = 1.1) +
  theme_minimal() +
  theme(legend.position="none") +
  ylab("predicted probability") +
  xlab("")

# EVALUATE THE PREDICTION ERROR ON THE OBSERVED SAMPLE:
error = 0
subN = 100

plot(1:subN, rep(0.5, subN), col = "white", ylim = c(-0.01, 1.01))
for(i in 1:subN){
  Xnew = X[i,]
  for(g in 1:G) # loop over the iterations
    out[g] = pnorm(sum(Xnew*beta[g,]), mean = 0, sd = 1 )
  error = error + abs(Y[i] - ifelse(mean(out) > 0.5, 1, 0))
  segments(i, quantile(out, 0.05), i, quantile(out, 0.95), col = "royalblue", lwd = 2)
  points( i, Y[i], col = 'darkred', pch = 19 )
  points( i, mean(out), col = 'royalblue', pch = 19 )
  message("*** Subject ", i)
}
abline(h = 0.5)
errBEN <- error/subN

##-----------------------------------------------------------------------

mp_SpSl             # median probability model (SpSl)
HDP_SpSl            # Highest posterior density model (HPD) (SpSl)

mp_SSV1             # median probability model (SSVS1)
HDP_SSV1            # Highest posterior density model (HPD) (SSVS1)

mp_SSV2             # median probability model (SSVS2)
HDP_SSV2            # Highest posterior density model (HPD) (SSVS1)

idx_cov_BL          # Bayesian lasso
idx_cov_BEN         # Bayesian elastic net

# errors
c(errBL, errBEN)

#########################################################################
################### BAYESIAN LASSO 2 ####################################
#########################################################################

build_model <- stanc(file = 'BL_probit2.stan', model_name = 'BL_stan2')
compiled_model <- stan_model(stanc_ret = build_model, verbose = FALSE)
```

```
#------- run in parallel for different lambda
no_cores <- 1
registerDoParallel(cores=no_cores)
cl <- makeCluster(no_cores)
lambda_seq <- seq(0.1, 150, length.out = 50)

model_list <- foreach(i = 1:length(lambda_seq)) %dopar% {
  data_win <- list(N = N, p = p, Y = Y, X = as.matrix(X), lambda = lambda_seq[i])
  BL_model = sampling(compiled_model,
           data = data_win, chains = 1, iter = 10000,
           warmup = 5000, thin = 1, seed = 42)
  BL_model
}
stopCluster(cl)
# save(model_list, file="BL_model_multi.dat" )
load("BL_model_multi.dat")

df <- data.frame(x = unlist(lapply(model_list, function(x) as.vector(rstan::get_posterior_mean(x)[1:12,1
]))),
                 par = rep(colnames(reach)[1:12], length(lambda_seq)),
                 lambda = rep(lambda_seq, each = 12))
pBL <- ggplot(df) +
  geom_line(aes(x = lambda, y = x, group = par, col = par)) +
  theme_bw() +
  ylab("posterior mean") +
  scale_color_discrete(name = "variable") +
  ggtitle("BAYESIAN LASSO")

#########################################################################
################### BAYESIAN RIDGE 2 ####################################
#########################################################################

build_model <- stanc(file = 'BR_probit2.stan', model_name = 'BR_stan2')
compiled_model <- stan_model(stanc_ret = build_model, verbose = FALSE)

#------- run in parallel for different lambda
no_cores <- 1
registerDoParallel(cores=no_cores)
cl <- makeCluster(no_cores)
lambda_seq <- seq(0.1, 150, length.out = 50)

model_list <- foreach(i = 1:length(lambda_seq)) %dopar% {
  data_win <- list(N = N, p = p, Y = Y, X = as.matrix(X), lambda = lambda_seq[i])
  BR_model = sampling(compiled_model,
                      data = data_win, chains = 1, iter = 10000,
                      warmup = 5000, thin = 1, seed = 42)
  BR_model
}
stopCluster(cl)
# save(model_list, file="BR_model_multi.dat" )
load("BR_model_multi.dat")

df2 <- data.frame(x = unlist(lapply(model_list, function(x) as.vector(rstan::get_posterior_mean(x)[1:12,1
]))),
                  par = rep(colnames(reach)[1:12], length(lambda_seq)),
                  lambda = rep(lambda_seq, each = 12))
pBR <- ggplot(df2) +
  geom_line(aes(x = lambda, y = x, group = par, col = par)) +
  theme_bw() +
  ylab("posterior mean") +
  scale_color_discrete(name = "variable") +
  ggtitle("BAYESIAN RIDGE")

#---------------
# comparison

ggarrange(pBL, pBR, legend = "right", common.legend = T)
```

19/11/2020
1/8

Mixing the models and Time series data

Riccardo Corradin
November 19, 2020

Introduction

GLM

Let Y be the response variable, $Y \in \mathcal{Y}$, and $\boldsymbol{x}$ a p-dimensional vector of covariates, with $\boldsymbol{x} \in \mathcal{X}$. Assume that $f_Y \in \mathcal{F}$ **exponential family** of distribution.

Moreover, let $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\xi}) \in \Theta$ a vector of parameters, with Θ parameter space, where $\boldsymbol{\beta}$ denotes the regression coefficients and $\boldsymbol{\xi}$ denotes the other parameters, such that the following holds

$$\mathbb{E}[Y \mid \boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\xi}] = g^{-1}(\boldsymbol{x}^\intercal \boldsymbol{\beta})$$

A **G**eneralized **L**inear **M**odel is composed by three terms:

- A **probability distribution** f_Y (belonging to the exponential family)
- A **linear predictor** $\eta = \boldsymbol{x}^\intercal \boldsymbol{\beta}$
- A **link function** $g(\cdot)$ such that $\mathbb{E}[Y \mid \boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\xi}] = g^{-1}(\eta)$

GLM^2

Different **probability distributions** and **link functions** lead to different **models**. Denote by $\mu = \mathbb{E}[Y \mid \boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\xi}]$. Examples of models:

- **Normal** distribution for Y
  - Identity link function, leads to the linear regression
  $$\boldsymbol{x}^\intercal \boldsymbol{\beta} = \eta = g(\mu) = \mu$$
- **Bernoulli** distribution for Y (or **categorical**)
  - Logit link function, leads to the logistic regression
  $$\boldsymbol{x}^\intercal \boldsymbol{\beta} = \eta = g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$$
  - Probit link function, leads to the probit regression
  $$\boldsymbol{x}^\intercal \boldsymbol{\beta} = \eta = g(\mu) = \Phi^{-1}(\mu)$$
- **Poisson** distribution for Y
  - Log link function, leads to the Poisson regression
  $$\boldsymbol{x}^\intercal \boldsymbol{\beta} = \eta = g(\mu) = \log(\mu)$$

GLM with Bayes

Assume that we have an exchangeable sample $\{(Y_1, \boldsymbol{x}_1), \dots, (Y_m, \boldsymbol{x}_n)\}$. We denote by

$$\mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{X}, \theta) = \prod_{i=1}^{n} f_Y(Y_i \mid \boldsymbol{x}_i, \theta)$$

the **likelihood** of our data, where $f_Y(Y \mid \boldsymbol{x}, \theta)$ corresponds to the probability distribution of a **GLM** model, and by $\pi(\theta)$ the prior distribution on the vector of parameters θ.

We are interested into performing inference on the parameters θ, characterizing the model. By **Bayes' theorem**, we have that

$$\pi(\theta \mid \boldsymbol{Y}, \boldsymbol{X}) = \frac{\mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{X}, \theta)\pi(\theta)}{\int_\Theta \mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{X}, \theta)\pi(d\theta)}$$

We can use the previous to perform models estimation and inference. Usually we don't have **conjugacy** $\implies$ the estimation can be tedious, and computational intensive!

Mixed or mixture?

The difference

By now, we set $\theta = (\beta, \boldsymbol{\lambda}, \boldsymbol{\xi}) \in \Theta$.

Assume that our data are divided into **groups**. Let G a variable denoting the **allocation** of the observations to a specific group, i.e. $G_i = j \implies$ the observation i-th belongs to the group j.

- **Mixed models**: the values $G_1, \dots, G_n$ are known and fixed, denoting a group structure of the data. As consequence, our data are **partially exchangeable** (see the next slide). GLM within this framework are defined by a relation as the following

  The data are partially exchangeable: they're ex. intra-groups

$$g(\mu) = \boldsymbol{x}^\intercal \beta + \boldsymbol{z}^\intercal \boldsymbol{\lambda}_j$$

  where $\boldsymbol{x}$ denotes the covariates for the fixed effects, β the coefficients for the fixed effects, $\boldsymbol{z}$ denotes the covariates for the random effects, and $\boldsymbol{\lambda}$ the coefficients for the random effects.

Data are fully exch.

- **Mixture models**: the values $G_1, \dots, G_n$ are unknown and random, object of interest for our inference. The data are exchangeable.

Partial exchangeability

→ observations are exchangeable within groups

Let K be the number of groups, and $Y_{1,1}, \dots, Y_{n_1,1}, \dots, Y_{1,K}, \dots, Y_{n_K,K}$ a set of data divided into K groups.

We say that $Y_{1,1}, \dots, Y_{n_1,1}, \dots, Y_{1,K}, \dots, Y_{n_K,K}$ are **partially exchangeable** if for any set of k permutation $\sigma_1, \dots, \sigma_k$, where σ_j is a permutation of $1, \dots, n_j$, we have

$$\begin{aligned}&\mathscr{L}(Y_{1,1}, \dots, Y_{n_1,1}, \dots, Y_{1,K}, \dots, Y_{n_K,K}) \\ &\quad = \mathscr{L}(Y_{\sigma_1(1),1}, \dots, Y_{\sigma_1(n_1),1}, \dots, Y_{\sigma_K(1),K}, \dots, Y_{\sigma_K(n_K),K})\end{aligned}$$

- De Finetti representation theorem for partially exchangeable data
- The likelihood becomes

$$\begin{aligned}&\mathscr{L}(\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_K \mid \boldsymbol{X}, \boldsymbol{G}, \beta, \boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_K) \\ &\qquad = \prod_{j=1}^{K} \prod_{i:G_i=j}^{n} f_Y(Y_i \mid \boldsymbol{x}_i, \boldsymbol{z}_i, \beta, \boldsymbol{\lambda}_j)\end{aligned}$$

GLMM

GLMM - introduction

! Our data are divided into K **groups**. We know the group allocation. We observe:

- $Y_{i,j}$ i-th response variable in the j-th group
- $\boldsymbol{x}_{i,j}$ i-th covariates in the j-th group for the **fixed effects** → associated to global coefficients
- $\boldsymbol{z}_{i,j}$ i-th covariates in the j-th group for the **random effects** → associated to coefficients which are group specific

We denote by

- $\pi(\boldsymbol{\beta})$ the prior for the regression coefficients corresponding to the **fixed effects** $\boldsymbol{\beta}$
- $\prod_{j=1}^{K} \pi_j(\boldsymbol{\lambda}_j)$ the prior for the regression coefficients corresponding to the **random effects** $\boldsymbol{\lambda}_j$, for the groups $j = 1, \ldots, K$
- $\pi(\boldsymbol{\xi})$ the prior for the remaining parameters of the model

GLMM - model specification

Model specification:

$$\begin{aligned} Y_{i,j} \mid \mu_{i,j}, \boldsymbol{\xi} &\sim f_Y(Y_{i,j} \mid \mu_{i,j}, \boldsymbol{\xi}), && i = 1 : n_j,\ j = 1 : K \\ \mu_{i,j} &= g^{-1}\left(\boldsymbol{x}_{i,j}^T \boldsymbol{\beta} + \boldsymbol{z}_{i,j}^T \boldsymbol{\lambda}_j\right) && i = 1 : n_j,\ j = 1 : K \\ \boldsymbol{\lambda}_j &\sim \pi_j(\boldsymbol{\lambda}_j), && j = 1, \ldots, K \\ \boldsymbol{\beta} &\sim \pi(\boldsymbol{\beta}) \\ \boldsymbol{\xi} &\sim \pi(\boldsymbol{\xi}) \end{aligned}$$

Prior elicitation:

- We assume independent distribution with expected value equal to 0 for all the coefficients, as for example

$$\beta_\ell \sim N(0, \sigma^2_{\beta_\ell})$$

- We may relax the model by assuming hyperprior distributions on the variances of the priors for the coefficients.

GLMM - Identifiability

Let $\mathcal{F} = \{f(\cdot, \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}$ our statistical model, indexed by $\boldsymbol{\theta}$. We say that a **model** is **identifiable** if

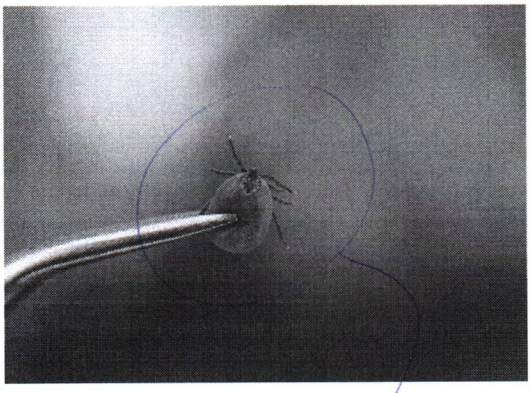
$$f(\cdot, \boldsymbol{\theta}_1) = f(\cdot, \boldsymbol{\theta}_2) \Longrightarrow \boldsymbol{\theta}_1 = \boldsymbol{\theta}_2, \qquad \forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \Theta$$

- The GLMMs **are not always identifiable**
- A solution to the identifiability problem is given by setting a 0-mean distribution on the random effects (as we have done)
- When you go down in the hierarchies, you can suppress the common intercept term (is helping)

Example: random intercepts

The data

A tick:



The data[2]

A red grouse:

The data[3] and the model

We have a set of 403 **observations**, where each observation correspond to a particular grouse. For each animal we observe:

- Height, above sea level (meters), denoted by X, where the animal lives, continuous variable
- Year (three different years), denoted by G, where $G = 1$ is 1994, $G = 2$ is 1995 and $G = 3$ is 1996. We assume a specific random effect for each year
- Number of ticks Y on the heads of the red grouse

We assume a Poisson distribution on the data, then the **model** becomes:

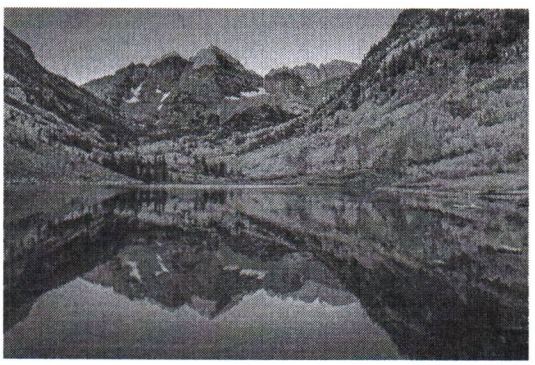
$$
\begin{aligned}
Y_{i,j} \mid \mu_{i,j} &\sim Poisson(\mu_{i,j}) \\
\mu_{i,j} &= e^{\beta_0 + \beta_1 x_{i,j} + \lambda_j} \\
\beta_\ell &\sim N(0, \sigma^2_{\beta_\ell}) \\
\lambda\ell &\sim N(0, \sigma^2_{\lambda_\ell}) \\
\sigma^2_{\ldots} &\sim InvGamma(1, 10)
\end{aligned}
$$

19/11/2020
3/8

Example: random coefficients

The data

Aspen:



The data[2]

An ant:

The data[3]

We have a set of 60 **observations**, where each observation correspond to a particular tree. The tree. are divided into *High* and *Low* altitude trees. For each spot we observe:

- Trap days, different lengths of time surveying each tree, denoted by X
- Elevation, G. If $G = 1$, then the spot is *High*, if $G = 2$ then the spot is *Low*.
- Valley, three different valleys for each elevation, denoted by SG, nested within each elevation
- Summer precipitation (avg), assuming a random effect, denoted by Z_1
- Summer temperature (avg), assuming a random effect, denoted by Z_2
- Number of ants, assuming a random effect, denoted by Y

The model

We assume a Poisson distribution on the data, then the **model** becomes:

$$
\begin{aligned}
Y_{i,j,r} \mid \mu_{i,j,r} &\sim Poisson(\mu_{i,j,r}) \\
\mu_{i,j,r} &= \exp\left(\beta_1 x_{i,j} + \theta_j + \gamma_{j,1} z_{1,i,j} + \gamma_{j,2} z_{2,i,j} + \alpha_{j,r}\right) \\
\beta_1 &\sim N(0, \sigma^2_{\beta_1}) \\
\theta_j &\sim N(0, \sigma^2_{\theta_j}) \\
\gamma_{j,t} &\sim N(0, \sigma^2_{\gamma_{j,t}}) \\
\alpha_{j,r} &\sim N(0, \sigma^2_{\alpha_{j,r}}) \\
\sigma^2_{\cdots} &\sim InvGamma(1, 10)
\end{aligned}
$$

where we have two nested hierarchies, with β_1 common regression coefficient shared across different groups, θ_j is the random effect for the j-th group, $\alpha_{j,r}$ is the random effect of the r-th subgroup nested in the j-th group and $\gamma_{j,1}, \gamma_{j,2}$ are the random coefficients for the j-th group.

See the code!

Models for time series

Univariate time series

Let Y be a random variable, with $Y \in \mathcal{Y} \subseteq \mathbb{R}$. We further assume that we observed the random variable at different times $t = 1, \dots, T$ (**for simplicity we assume discrete and equally spaced times**).

Different possible modeling strategies:

- **AR(p)** - **A**uto-**R**egressive model of order p: = regression on the past

$$Y_t = \alpha + \sum_{j=1}^{p} \phi_j Y_{t-j} + \varepsilon_t, \quad \varepsilon_t \sim f_\varepsilon \text{ noise}$$

- **MA(p)** - **M**oving **A**verage model of order p: = we work with the past error terms

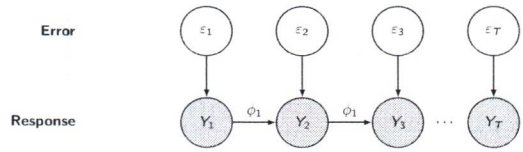
$$Y_t = \alpha + \sum_{j=1}^{p} \theta_j \varepsilon_{t-j}$$

- **ARMA(p,q)** - Auto-Regressive Moving Average model of order (p, q), combining together the previous strategies.

19/11/2020
4|8

AR(1)

We consider an AR(1) model.



We complete the model specification by assuming a Gaussian distribution for the error and with the following priors:

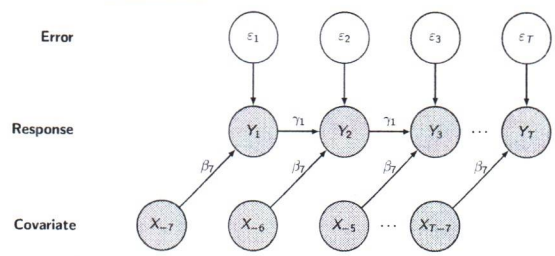
We can translate the model in a Bayesian framework:

$$
\begin{aligned}
Y_t \mid \alpha, \phi_1, \sigma_Y^2 &\sim N(\alpha + \phi_1 Y_{t-1}, \sigma_Y^2) \\
\alpha &\sim N(0, \sigma_\alpha^2) \\
\phi_1 &\sim N(0, \sigma_\beta^2) \\
\sigma_Y^2 &\sim InvGamma(a_Y, b_Y)
\end{aligned}
$$

See the code!

DLR model

We might want to include a lagged covariate to increase the prediction performance of the model. We consider a **D**ynamic **R**egression **M**odel.



DLR model

We complete the model specification assuming the following priors:

$$
\begin{aligned}
Y_t \mid \alpha, \gamma_1, \beta_7, \sigma_Y^2, \boldsymbol{X} &\sim N(\alpha + \gamma_1 Y_{t-1} + \beta_7 X_{t-7}, \sigma_Y^2) \\
\alpha &\sim N(0, \sigma_\alpha^2) \\
\phi_1 &\sim N(0, \sigma_\beta^2) \\
\sigma_Y^2 &\sim InvGamma(a_Y, b_Y)
\end{aligned}
$$

See the code!

```
## StSp_model.stan

# //#####################################################
# //### State-space univariate model ####################
# //#####################################################
#
# /////////////////////////// DATA //////////////////////////
# data {
#   int<lower = 0> N;       // number of data
#   real Y[N];      // response vector
#   real sigma2phi;
#   real a_sigma2;
#   real b_sigma2;
#   real sigma2m0;
# }
#
# ////////////////////// PARAMETERS //////////////////////////
# parameters {
#   real phi;
#   real sigma2;
#   real m0;
# }
#
#
# ////////////////// MODEL ///////////////////////////////
# model {
#
#   // Likelihood
#   for (t in 2:N)
#   {
#       Y[t] ~ normal(m0 + phi * Y[t - 1], pow(sigma2, 0.5));
#   }
#   m0 ~ normal(0, sigma2m0);
#   phi ~ normal(0, sigma2phi);
#   sigma2 ~ inv_gamma(a_sigma2, b_sigma2);
# }
```

```
## DLR_model.stan

# //#####################################################
# //### State-space univariate model ####################
# //#####################################################
#
# /////////////////////////// DATA //////////////////////////
# data {
#   int<lower = 0> N;       // number of data
#   real Y[N];      // response vector
#   real X[N];      // response vector
#   real sigma_coef[3];
#   real sigma2Y_par[2];
# }
#
# ////////////////////// PARAMETERS //////////////////////////
# parameters {
#   real sigma2Y;
#   real m0Y;
#   real beta;
#   real gamma;
# }
#
#
# ////////////////// MODEL ///////////////////////////////
# model {
#
#   // Likelihood
#   for (t in 2:N)
#   {
#       Y[t] ~ normal(m0Y + beta * X[t] + gamma * Y[t - 1], pow(sigma2Y, 0.5));
#   }
#
#   m0Y ~ normal(0, sigma_coef[1]);
#   beta ~ normal(0, sigma_coef[2]);
#   gamma ~ normal(0, sigma_coef[3]);
#   sigma2Y ~ inv_gamma(sigma2Y_par[1], sigma2Y_par[2]);
# }
```

```
## Poi_GLM.stan

# //#####################################################
# //### GLM - POISSON REGRESSION ########################
# //#####################################################
#
# /////////////////////////// DATA //////////////////////////////////////
# data {
#   int<lower = 0> N;       // number of data
#   int<lower = 0> p_fix;   // number of covariates, fixed effects
#
#   int<lower = 0> Y[N];    // response vector
#   matrix[N, p_fix] X;     // design matrix (fixed effects)
# }
#
# ////////////////////// PARAMETERS ////////////////////////////////////
# parameters {
#   vector[p_fix] beta;         // regression coefficients
#   vector[p_fix] sigma2_beta;  // variances for the prior on beta
# }
#
# ////////////////////// TRANSFORMED PARAMETERS ////////////////////////////////////
# transformed parameters
# {
#   vector[N] mu;
#   for(i in 1:N){
#     mu[i] = exp(row(X, i) * beta);
#   }
# }
#
# ////////////////// MODEL ////////////////////////
# model {
#
#   // Likelihood
#   for (s in 1:N)
#   {
#       // print(" s = ", s);
#       Y[s] ~ poisson(mu[s]);
#   }
#
#   for (j in 1:p_fix)
#   {
#       beta[j] ~ normal(0.0, pow(sigma2_beta[j], 0.5));
#       sigma2_beta[j] ~ inv_gamma(2., 10.);
#   }
# }
#
# ////////////////// GENERATED QUANTITIES ////////////////////////
# generated quantities
# {
#       vector[N] log_lik;
#       for (j in 1:N)
#   {
#           log_lik[j] = poisson_lpmf(Y[j] | mu[j]);
#       }
# }
```

Poi_GLMM.stan

```
# //////////////////////////////////////////////////
# //### GLMM - POISSON REGRESSION ################
# //////////////////////////////////////////////////
#
# /////////////////////////// DATA /////////////////////////////////////
# data {
#   int<lower = 0> N;       // number of data
#   int<lower = 0> p_fix;   // number of covariates, fixed effects
#   int<lower = 0> ngr;  // number of groups
#
#   int<lower = 0> Y[N];    // response vector
#   matrix[N, p_fix] X;     // design matrix (fixed effects)
#   matrix[N, ngr] G;       // gropus (dummy) allocation
# }
#
# ////////////////////// PARAMETERS ///////////////////////////////////
# parameters {
#   vector[p_fix] beta;         // regression coefficients
#   vector[ngr] theta;          // (group specific) random effects
#   vector[p_fix] sigma2_beta;  // variances for the prior on beta
#   vector[ngr] sigma2_theta;   // variances for the prior on theta
# }
#
# ////////////////////// TRANSFORMED PARAMETERS ///////////////////////////////
# transformed parameters
# {
#   vector[N] mu;
#   for(i in 1:N){
#     mu[i] = exp(row(X, i) * beta + row(G, i) * theta);
#   }
# }
#
# ////////////////// MODEL /////////////////////////
# model {
#
#   // Likelihood
#   for (s in 1:N)
#   {
#       // print(" s = ", s);
#       Y[s] ~ poisson(mu[s]);
#   }
#
#   for (j in 1:p_fix)
#   {
#       beta[j] ~ normal(0.0, pow(sigma2_beta[j], 0.5));
#       sigma2_beta[j] ~ inv_gamma(2., 10.);
#   }
#
#   for (j in 1:ngr)
#   {
#       theta[j] ~ normal(0.0, pow(sigma2_theta[j], 0.5));
#       sigma2_theta[j] ~ inv_gamma(2., 10.);
#   }
#
# }
#
# ////////////////// GENERATED QUANTITIES /////////////////////////
# generated quantities
# {
#       vector[N] log_lik;
#       for (j in 1:N){
#           log_lik[j] = poisson_lpmf(Y[j] | mu[j]);
#       }
# }
```

Poi_GLMM_ran.stan

```
# //////////////////////////////////////////////////
# //### GLMM - POISSON REGRESSION ################
# //////////////////////////////////////////////////
#
# /////////////////////////// DATA /////////////////////////////////////
# data{
#   int<lower = 0> N;        // number of data
#   int<lower = 0> ngr;   // number of groups
#   int<lower = 0> nsubgr;    // number of groups
#   int<lower = 0> p_fix;    // number of covariates, fixed effects
#   int<lower = 0> p_ran;    // number of covariates, random effects
#   int<lower = 0> Y[N];      // response vector
#   vector[N] X;      // design matrix (fixed effects)
#   matrix[N, p_ran] Z;       // design matrix (random effects)
#   int<lower = 1, upper = ngr> group[N];
#   int<lower = 1, upper = nsubgr> subgroup[N];
# }
#
# ////////////////////// PARAMETERS ///////////////////////////////////
# parameters{
#   real beta;                      // regression coefficients (fixed)
#   vector[ngr] theta;                // (group specific) random effects
#   matrix[p_ran, ngr] gamma;       // regression coefficients (random)
#   vector[nsubgr] alpha;           // (group specific) random effects
#   real sigma2_beta;                 // variances for the prior on beta
#   vector[ngr] sigma2_theta;         // variances for the prior on theta
#   matrix[ngr, p_ran] sigma2_gamma; // (group specific) random effects
#   vector[nsubgr] sigma2_alpha;    // (group specific) random effects
# }
#
# ////////////////////// TRANSFORMED PARAMETERS ///////////////////////////////
# transformed parameters{
#   vector[N] mu;
#   for(i in 1:N){
#     mu[i] = exp(X[i] * beta + theta[group[i]] +
#                 row(Z, i) * col(gamma, group[i]) + alpha[subgroup[i]]);
#   }
# }
#
# ////////////////// MODEL /////////////////////////
# model{
#   // Likelihood
#   for (s in 1:N){
#       Y[s] ~ poisson(mu[s]);
#   }
#       // Priors
#   beta ~ normal(0.0, pow(sigma2_beta, 0.5));
#   sigma2_beta ~ inv_gamma(2., 10.);
#   for (j in 1:ngr){
#       theta[j] ~ normal(0.0, pow(sigma2_theta[j], 0.5));
#       sigma2_theta[j] ~ inv_gamma(2., 10.);
#   }
#   for (j in 1:nsubgr){
#       alpha[j] ~ normal(0.0, pow(sigma2_alpha[j], 0.5));
#       sigma2_alpha[j] ~ inv_gamma(2., 10.);
#   }
#   for (j in 1:p_ran) {
#     for(i in 1:ngr){
#       gamma[j,i] ~ normal(0.0, pow(sigma2_gamma[j,i], 0.5));
#       sigma2_gamma[j,i] ~ inv_gamma(2., 10.);
#     }
#   }
# }
#
# ////////////////// GENERATED QUANTITIES /////////////////////////
# generated quantities{
#       vector[N] log_lik;
#       for (j in 1:N){
#           log_lik[j] = poisson_lpmf(Y[j] | mu[j]);
#       }
# }
```

```
####################################
###########   GLMM           #####
####################################

# R & STAN are friends!
library(rstan)
library(coda)

# for plots
library(ggplot2)
library(tidyr)
library(dplyr)
library(purrr)
library(ggsci)
require(gplots)
require(ggpubr)

load("data_gro.dat")
head(data_grouse)

X <- data_grouse[,2]
temp <- as.factor(data_grouse$YEAR)
G <- model.matrix(~temp, contrasts.arg=list(temp=contrasts(temp, contrasts=F)))[,-1]
colnames(G) <- c("95", "96", "97")
Y <- as.vector(data_grouse[,1])

N <- length(X)
p_fix <- 2
ngr <- length(unique(data_grouse$YEAR))

# useful function ------------------------------------------------------

my_WAIC <- function(fit, param){

  llik   <- rstan::extract(fit, param)[[1]]
  p_WAIC <- sum(apply(llik, 2, var))
  lppd   <- sum(apply(llik, 2, function(x) log(mean(exp(x)))))
  WAIC   <- - 2 * lppd + 2 * p_WAIC
  return(WAIC)

}

#########################################################################
################### POI regression ######################################
#########################################################################

data_POI <-list(N = N,
                p_fix = 2,
                ngr = 3,
                Y = Y,
                X = as.matrix(cbind(rep(1, N), X)),
                G = as.matrix(G))

inits <- function()
{
  list(beta = rep(0.1, p_fix),
       theta = rep(0.1, ngr),
       sigma2_beta = rep(10, p_fix),
       sigma2_theta = rep(10, ngr))
}

################### FIT THE MODEL

Poi_GLMM <- stan(file = "Poi_GLMM.stan",
                 data = data_POI,
                 chains = 2,
                 iter = 30000,
                 warmup = 10000,
                 thin= 20,
                 seed = 42,
                 init = inits,
                 algorithm = 'NUTS')
# save(Poi_GLMM, file="Poi_GLMM_est.dat")
load("Poi_GLMM_est.dat")

rstan::traceplot(Poi_GLMM, pars = "sigma2_beta", inc_warmup = TRUE)
rstan::traceplot(Poi_GLMM, pars= "sigma2_theta", inc_warmup = TRUE)

rstan::traceplot(Poi_GLMM, pars = "beta", inc_warmup = FALSE)
rstan::traceplot(Poi_GLMM, pars= "theta", inc_warmup = FALSE)

# Diagnostic -----------------------------------------------------------

coda_chain <- As.mcmc.list(Poi_GLMM, pars = c("beta", "theta"))
summary(coda_chain)

# Compute the gelman and rubin's convergence diagnostic
# â??potential scale reduction factorâ??
# And quoting Gelman:
# â??Inference is normal science. Model-checking is revolutionary science.â??
gelman.diag(coda_chain, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)

# Geweke's convergence diagnostic
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

# autocorrelation and plot
acfplot(coda_chain, lag.max = 30)

# Posterior distributions ----------------------------------------------

# beta
plot_post <- Poi_GLMM %>%
  rstan::extract("beta") %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")

# theta
plot_post <- Poi_GLMM %>%
  rstan::extract("theta") %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  xlim(c(-6, 8)) +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  coord_flip() +
  theme(legend.position="none")

# PLOT -----------------------------------------------------------------

params <- rstan::extract(Poi_GLMM, c("beta", "theta"), perm = T)
x_seq <- seq(380, 570, length.out = 100)
jpar <- cbind(params[[1]], params[[2]])
tres <- array(0, dim = c(nrow(jpar), length(x_seq), 3))
for(i in 1:nrow(jpar)){
  for(j in 1:3){
    tres[i,,j] <- exp(jpar[i,1] + jpar[i,2] * x_seq + jpar[i,2 + j])
  }
}

df_plot <- data.frame(x = rep(x_seq, 3),
                      ymean = as.vector(apply(tres, c(2,3), mean)),
                      group = as.factor(c(rep(95, 100), rep(96, 100), rep(97, 100))))
pdf_plot <- data.frame(x = X, y = Y, group = as.factor(data_grouse$YEAR))

ggplot() +
  geom_point(data = pdf_plot, mapping = aes(x = x, y = y, col = group)) +
  geom_line(data = df_plot, mapping = aes(x = x, y = ymean, col = group)) +
  scale_color_locuszoom() +
  theme_minimal() +
  theme(legend.position="none") +
```

```
  ylab("Numbers of ticks on red grouse") +
  xlab("")

#------------------------------------------------------------------------
### COMPARE WITH GLM
#------------------------------------------------------------------------

data_POI <-list(N = N,
                p_fix = 2,
                Y = Y,
                X = as.matrix(cbind(rep(1, N), X)))

inits <- function()
{
  list(beta = rep(0.1, p_fix),
       sigma2_beta = rep(10, p_fix))
}

################### FIT THE MODEL

Poi_GLM <- stan(file = "Poi_GLM.stan",
                data = data_POI,
                chains = 2,
                iter = 30000,
                warmup = 10000,
                thin= 20,
                seed = 42,
                init = inits,
                algorithm = 'NUTS')
# save(Poi_GLM, file="Poi_GLM_est.dat")
load("Poi_GLM_est.dat")

rstan::traceplot(Poi_GLM, pars = "sigma2_beta", inc_warmup = TRUE)
rstan::traceplot(Poi_GLM, pars = "beta", inc_warmup = FALSE)

# Diagnostic ------------------------------------------------------------

coda_chain <- As.mcmc.list(Poi_GLM, pars = c("beta"))
summary(coda_chain)

# Compute the gelman and rubin's convergence diagnostic
# â potential scale reduction factorâ
gelman.diag(coda_chain, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)

# Geweke's convergence diagnostic
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

# autocorrelation and plot
acfplot(coda_chain, lag.max = 30)

# WAIC GLMM vs GLM -------------------------------------------------------

my_WAIC(Poi_GLMM, "log_lik")
my_WAIC(Poi_GLM, "log_lik")

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------
# ------------------------------------------------------------------------
# ------------------------------------------------------------------------
# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

AntData <- read.csv("AntAbundance.csv")
head(AntData)

# num.fpods ~ b0_j * elevation + b1_j * valley + b2_j * summer.precip + b3_j * summer.temp + b4 * trap.da
ys +
table(AntData$num.fpodz)

Y <- AntData$num.fpodz
X <- as.matrix(AntData$trap.days)
group <- as.numeric(as.factor(AntData$elevation))
subgroup <- rep(1:6, each = 10)
Z <- as.matrix(cbind(AntData[,c(8,10)]))
N <- length(Y)

p_fix <- 1
p_ran <- 2
ngr <- 2
nsubgr <- 6

#------------------------------------------------------------------------

data_POI <-list(N = length(Y),
                p_fix = 1,
                p_ran = 2,
                ngr = 2,
                nsubgr = 6,
                Y = Y,
                X = as.vector(X),
                Z = Z,
                group = group,
                subgroup = subgroup)

inits <- function()
{
  list(beta = 0.1,
       sigma2_beta = 10,
       theta = rep(0.1, ngr),
       sigma2_theta = rep(10, ngr),
       gamma = matrix(0.1, nrow = p_ran, ncol = ngr),
       sigma2_gamma = matrix(10, nrow = p_ran, ncol = ngr),
       alpha = rep(0.1, nsubgr),
       sigma2_alpha = rep(10, nsubgr))
}

################### FIT THE MODEL

Poi_GLMM_ran <- stan(file = "Poi_GLMM_ran.stan",
                     data = data_POI,
                     chains = 2,
                     iter = 30000,
                     warmup = 10000,
                     thin= 20,
                     seed = 42,
                     init = inits,
                     algorithm = 'NUTS')
# save(Poi_GLMM_ran, file="Poi_GLMM_ran.dat")
load("Poi_GLMM_ran.dat")

rstan::traceplot(Poi_GLMM_ran, pars = "sigma2_beta", inc_warmup = TRUE)
rstan::traceplot(Poi_GLMM_ran, pars = "beta", inc_warmup = FALSE)

rstan::traceplot(Poi_GLMM_ran, pars = "sigma2_theta", inc_warmup = TRUE)
rstan::traceplot(Poi_GLMM_ran, pars = "theta", inc_warmup = FALSE)

rstan::traceplot(Poi_GLMM_ran, pars = "sigma2_gamma", inc_warmup = TRUE)
rstan::traceplot(Poi_GLMM_ran, pars = "gamma", inc_warmup = FALSE)

rstan::traceplot(Poi_GLMM_ran, pars = "sigma2_alpha", inc_warmup = TRUE)
rstan::traceplot(Poi_GLMM_ran, pars = "alpha", inc_warmup = FALSE)

# Diagnostic ------------------------------------------------------------

coda_chain <- As.mcmc.list(Poi_GLMM_ran, pars = c("beta", "theta", "gamma", "alpha"))
summary(coda_chain)

# Compute the gelman and rubin's convergence diagnostic
# â potential scale reduction factorâ
gelman.diag(coda_chain, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)

# Geweke's convergence diagnostic
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

# autocorrelation and plot
acfplot(coda_chain, lag.max = 30)

# theta
plot_post <- Poi_GLMM_ran %>%
  rstan::extract("beta") %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
```

```
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")

# theta
plot_post <- Poi_GLMM_ran %>%
  rstan::extract("theta") %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  coord_flip() +
  xlim(c(-180, 20)) +
  theme(legend.position="none")

# alpha
plot_post <- Poi_GLMM_ran %>%
  rstan::extract("alpha") %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  coord_flip() +
  xlim(c(-10,8)) +
  theme(legend.position="none")

# theta
plot_post <- Poi_GLMM_ran %>%
  rstan::extract("gamma") %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  coord_flip() +
  xlim(c(-1, 12)) +
  theme(legend.position="none")

#------------------------------------------------------------------------
# Compare with a GLM (Poisson regression - no valley)

Y <- AntData$num.fpodz
X <- as.matrix(cbind(AntData$trap.days, as.numeric(as.factor(AntData$elevation)), AntData[,c(8,10)]))
N <- length(Y)
p <- 4

#------------------------------------------------------------------------

data_POI <-list(N = length(Y),
                p_fix = p,
                Y = Y,
                X = as.matrix(X))

inits <- function()
{
  list(beta = rep(0.1, p),
       sigma2_beta = rep(10, p))
}

################### FIT THE MODEL
```

```
Poi_GLM2 <- stan(file = "Poi_GLM.stan",
                 data = data_POI,
                 chains = 2,
                 iter = 30000,
                 warmup = 10000,
                 thin= 20,
                 seed = 42,
                 init = inits,
                 algorithm = 'NUTS')
# save(Poi_GLM2, file="Poi_GLM2.dat")
load("Poi_GLM2.dat")

rstan::traceplot(Poi_GLM2, pars = "sigma2_beta", inc_warmup = TRUE)
rstan::traceplot(Poi_GLM2, pars = "beta", inc_warmup = FALSE)

# Diagnostic ----------------------------------------------------------

coda_chain <- As.mcmc.list(Poi_GLM2, pars = c("beta"))
summary(coda_chain)

# Compute the gelman and rubin's convergence diagnostic
# â potential scale reduction factorâ
gelman.diag(coda_chain, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)

# Geweke's convergence diagnostic
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

# autocorrelation and plot
acfplot(coda_chain, lag.max = 30)

# WAIC GLMM vs GLM -----------------------------------------------------

my_WAIC(Poi_GLMM_ran, "log_lik")
my_WAIC(Poi_GLM2, "log_lik")


########################################################################
########################################################################
########################################################################
########################################################################
################### AR1 model ##########################################
########################################################################

load("data.Rda")
Y <- COVID_data$ricoveri[COVID_data$data > as.Date("2020-10-01") & COVID_data$regione == "Lombardia"]
N <- length(Y)

data_StSp <-list(N = N,
                 Y = Y,
                 sigma2phi = 10,
                 a_sigma2 = 2,
                 b_sigma2 = 10,
                 sigma2m0 = 10)

inits <- function()
{
  list(phi = 0,
       m0 = 0,
       sigma2 = var(Y))
}

################### FIT THE MODEL

StSp_model <- stan(file = "StSp_model.stan",
                   data = data_StSp,
                   chains = 2,
                   iter = 30000,
                   warmup = 10000,
                   thin= 20,
                   seed = 42,
                   init = inits,
                   algorithm = 'NUTS')
# save(StSp_model, file="StSp_model_est.dat")
load("StSp_model_est.dat")
rstan::traceplot(StSp_model, pars = c("phi", "m0", "sigma2"), inc_warmup = TRUE)
```

```
# Diagnostic ----------------------------------------------------------

coda_chain <- As.mcmc.list(StSp_model, pars = c("phi", "m0", "sigma2"))
summary(coda_chain)

# Compute the gelman and rubin's convergence diagnostic
# â potential scale reduction factorâ
# And quoting Gelman:
# âInference is normal science. Model-checking is revolutionary science.â
gelman.diag(coda_chain, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)

# Geweke's convergence diagnostic
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

# autocorrelation and plot
stan_ac(StSp_model)

# Posterior distributions ---------------------------------------------

# beta
plot_post <- StSp_model %>%
  rstan::extract(c("phi", "m0", "sigma2")) %>%
  as_tibble() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")


# PLOT ----------------------------------------------------------------

params <- data.frame(rstan::extract(StSp_model, c("phi", "m0", "sigma2"), perm = T))

resultStSp <- matrix(NA, nrow = nrow(params), ncol = 7)
for(i in 1:7){
  if(i == 1){
    resultStSp[,i] <- apply(params, 1, function(x) rnorm(1, x[2] + x[1] * Y[length(Y)], sqrt(x[3])))
  } else {
    resultStSp[,i] <- apply(cbind(params, resultStSp[,i-1]), 1, function(x) rnorm(1, x[2] + x[1] * x[4],
 sqrt(x[3])))
  }
}

ggplot(data.frame(x = (length(Y)):(length(Y) + 6),
                  y = colMeans(resultStSp),
                  ylow = apply(resultStSp, 2, quantile, p = 0.05),
                  yup = apply(resultStSp, 2, quantile, p = 0.95))) +
  geom_line(aes(x = x, y = y), lty = 2) + theme_bw() +
  geom_line(data = data.frame(Y = Y, x = 1:length(Y)), aes(x = x, y = Y)) +
  geom_ribbon(aes(ymin = ylow, ymax = yup, x = x), alpha = 0.2) +
  geom_vline(aes(xintercept = length(Y)), col = 2, lty = 2) +
  ggtitle("AR1 model - COVID data Lombardia")

#########################################################################
################### DRM model ###########################################
#########################################################################

load("data.Rda")
Y <- COVID_data$ricoveri[COVID_data$data >= as.Date("2020-10-01") & COVID_data$regione == "Lombardia"]
X <- COVID_data$casi[COVID_data$data >= as.Date("2020-9-24") & COVID_data$regione == "Lombardia"]
N <- length(Y)

data_DRM <-list(N = N,
                Y = Y,
                X = X[1:length(Y)],
                sigma_coef = rep(10, 3),
                sigma2X_par = c(2, 10),
                sigma2Y_par = c(2, 10))

inits <- function()
{
  list(sigma2Y = var(Y),
       m0Y = 0,
       beta = 0,
       gamma = 0)
}

################## FIT THE MODEL

DRM_model <- stan(file = "DLR_model.stan",
                  data = data_DRM,
                  chains = 2,
                  iter = 30000,
                  warmup = 10000,
                  thin= 20,
                  seed = 42,
                  init = inits,
                  algorithm = 'NUTS')
# save(DRM_model, file="DRM_model_est.dat")
load("DRM_model_est.dat")
rstan::traceplot(DRM_model, pars = c("beta", "gamma"), inc_warmup = TRUE)
rstan::traceplot(DRM_model, pars = c("m0Y", "sigma2Y"), inc_warmup = TRUE)

# Diagnostic ----------------------------------------------------------

coda_chain <- As.mcmc.list(DRM_model, pars = c("beta", "gamma"))
summary(coda_chain)

# Compute the gelman and rubin's convergence diagnostic
# â potential scale reduction factorâ
# And quoting Gelman:
# âInference is normal science. Model-checking is revolutionary science.â
gelman.diag(coda_chain, confidence = 0.95,  autoburnin = TRUE, multivariate=TRUE)

# Geweke's convergence diagnostic
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

# autocorrelation and plot
stan_ac(DRM_model)

# Posterior distributions ---------------------------------------------

# coefs
plot_post <- DRM_model %>%
  rstan::extract(c("beta", "gamma")) %>%
  as_tibble() %>%
  map_df(as_tibble, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")

# intercepts
plot_post <- DRM_model %>%
  rstan::extract(c("m0Y", "sigma2Y")) %>%
  as_tibble() %>%
  map_df(as_tibble, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")

# PLOT ----------------------------------------------------------------

params <- data.frame(rstan::extract(DRM_model, c("beta", "m0Y", "gamma", "sigma2Y"), perm = T))
Xtemp <- X[(length(Y) + 1):(length(Y) + 7)]
resultDLR <- matrix(NA, nrow = nrow(params), ncol = 7)
for(i in 1:7){
  if(i == 1){
    resultDLR[,i] <- apply(params, 1, function(x) rnorm(1, x[2] + x[1] * Xtemp[i] + x[3] * Y[length(Y)],
 sqrt(x[4])))
```

```
  } else {
    resultDLR[,i] <- apply(cbind(params, resultDLR[,i-1]), 1, function(x) rnorm(1,  x[2] + x[1] * Xtemp
[i] + x[3] * x[5], sqrt(x[4])))
  }
}

ggplot(data.frame(x = (length(Y)):(length(Y) + 6),
                  y = colMeans(resultDLR),
                  ylow = apply(resultDLR, 2, quantile, p = 0.05),
                  yup = apply(resultDLR, 2, quantile, p = 0.95))) +
  geom_line(data = data.frame(x = (length(Y)):(length(Y) + 6),
                              y = colMeans(resultStSp),
                              ylow = apply(resultStSp, 2, quantile, p = 0.05),
                              yup = apply(resultStSp, 2, quantile, p = 0.95)),
            aes(x = x, y = y), lty = 2, col = 3) +
  geom_ribbon(data = data.frame(x = (length(Y)):(length(Y) + 6),
                                y = colMeans(resultStSp),
                                ylow = apply(resultStSp, 2, quantile, p = 0.05),
                                yup = apply(resultStSp, 2, quantile, p = 0.95)),
              aes(ymin = ylow, ymax = yup, x = x), alpha = 0.2, fill = 3) +
  geom_line(aes(x = x, y = y), lty = 2) + theme_bw() +
  geom_line(data = data.frame(Y = Y, x = 1:length(Y)), aes(x = x, y = Y)) +
  geom_ribbon(aes(ymin = ylow, ymax = yup, x = x), alpha = 0.2) +
  geom_vline(aes(xintercept = length(Y)), col = 2, lty = 2) +
  ggtitle("AR1 model & DLR - COVID data Lombardia")
```

27/11/2020
1/9

Survival data

Riccardo Corradin
November 27, 2020

Survival data

Introduction

Let T be a **RV** denoting a positive time to event, i.e. with support $\mathbb{R}^+$, object of interest. Further we denote by $F(t) = P[T \leq t]$ the **cdf** of T and by $f(t) = \frac{d}{dt}F(t)$ its **pdf**.

Related quantities of interest

- Survival function - surviving at least until t

$$S(t) = P[T > t] = 1 - F(t)$$

- Hazard function - event rate at t conditionally on survival until t = probability that the event is happening during this infinitesimal t (δt)

$$h(t) = \lim_{\delta t \to 0} \frac{P[t \leq T < t + \delta t \mid T \geq t]}{\delta t}$$

- Cumulative hazard function - cumulative hazard until time t

$$H(t) = \int_0^t h(x)dx$$

Introduction^2

Useful relations

- Hazard - survival

$$h(t) = -\frac{S'(t)}{S(t)} = -\frac{d}{dt}log(S(t))$$

- Survival - cumulative hazard

$$S(t) = e^{-H(t)}$$

- Density - survival & hazard

$$f(t) = h(t)S(t)$$

Goal: we want to model the time to event, i.e. the survival function related to T or the cdf.

Right censoring

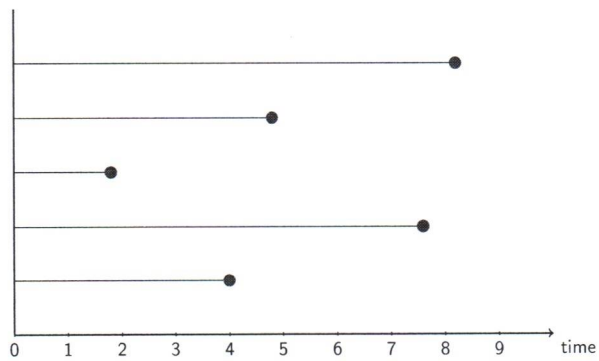
In an perfect world, we run the studies until the end of *"Life, the Universe and Everything"* and we observe the occurrences of all the events. But in real world, we do not.

In practice, three scenarios may occur:

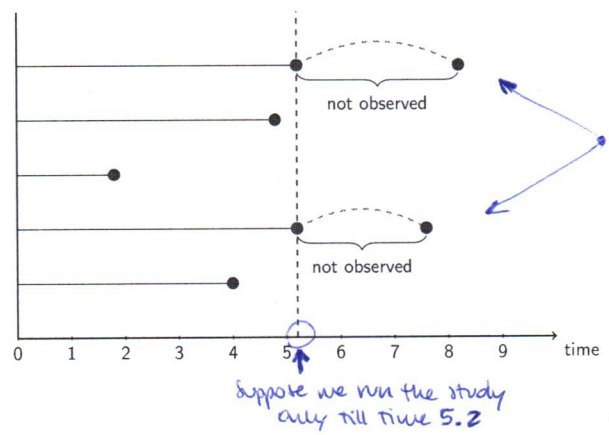
- An event occurs
- We are monitoring an individual/thing/what-else, but the event does not occur before the end of the study
- We are monitoring an individual/thing/what-else, but is leaving the study before the event occurs

Suppose that we observe $T_1, \ldots, T_n$ of time to events.

Right censoring[2]



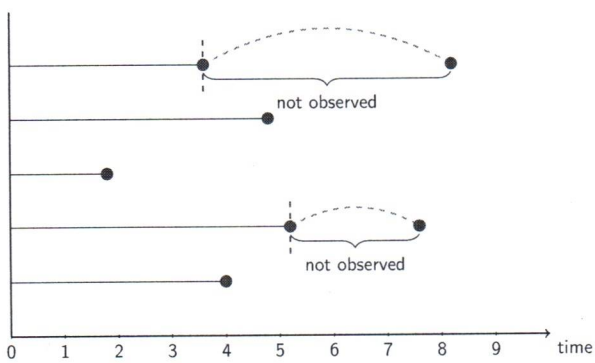
Right censoring[3] : FIXED CENSORING



In this case, for these two we only know that the event did not happen until the end of the study

Suppose we run the study only till time 5.2

Right censoring[4] : RANDOM CENSORING = the censoring time is different individual to individual



The likelihood

For simplicity we assume a common censoring time c, such that $T_i = \min\{Y_i, c\}$. Let δ_i denotes if an observation is censored, i.e.

$$\delta_i = \begin{cases} 1 : Y_i \leq c \\ 0 : Y_i > c \end{cases}$$

We denote by the generic $g(T_i \mid \cdot)$ the contribution to the likelihood of the i-th observation

- Each observation not truncated contributes to the likelihood by the fact that it survived until T_i and **the event occurs at** T_i, i.e.

$$g(T_i \mid \delta_i = 1, \theta) = f(T_i, \theta) = h(T_i, \theta) S(T_i, \theta)$$

- Each truncated observation contributes to the likelihood by the fact that **the event occurs after** T_i (i.e. survived until T_i without occurrence)

$$g(T_i \mid \delta_i = 0, \theta) = S(T_i, \theta)$$

The likelihood^2

Within the previous setup, the **likelihood** of an n-sized sample becomes

$$\begin{aligned} \mathcal{L}(T_1, \ldots, T_n \mid \delta_1, \ldots, \delta_n, \theta) &= \prod_{i=1}^{n} g(T_i \mid \delta_i, \theta) \\ &= \prod_{i=1}^{n} [f(T_i \mid \theta)]^{\delta_i} [S(T_i \mid \theta)]^{1-\delta_i} \\ &= \prod_{i=1}^{n} S(T_i \mid \theta) h(T_i \mid \theta)^{\delta_i} \end{aligned}$$

where it is apparent the contribution of the two situations:

- all the observations contributed in terms of surviving until the last observed time
- the uncensored observations contribute also by the risk of observe the events

Different parametric models

Different choices for the parametric model $f(T \mid \theta)$ lead to **different models** for the time to events. Common choices:

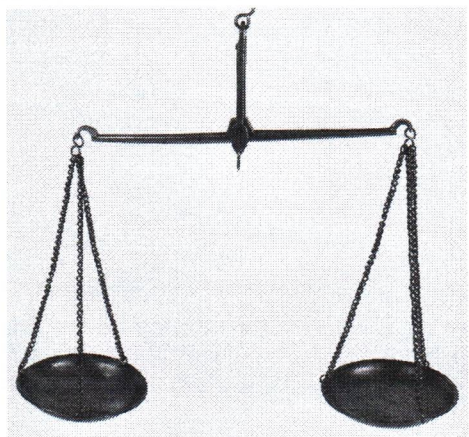
- **Exponential** $f(T_i \mid \theta) = \theta e^{-\theta T_i}$ $\quad T_i > 0,\ \theta > 0$
- **Weibull** $f(T_i \mid \theta, \kappa) = \theta \kappa t^{\kappa - 1} e^{-\theta t^{\kappa}}$ $\quad T_i > 0,\ \theta > 0,\ \kappa > 0$
- **Log-logistic**
  $f(T_i \mid \alpha, \beta) = \dfrac{\beta T_i^{\beta-1}/\alpha^{\beta}}{(1 + (T_i/\alpha)^{\beta})^2}$ $\quad T_i > 0,\ \alpha > 0,\ \beta > 0$
- **Log-normal** $f(T_i \mid \mu, \sigma) =$
  $\dfrac{1}{T_i \sigma \sqrt{2\pi}} exp\left(-\dfrac{1}{2}\left(\dfrac{log T_i - \mu}{\sigma}\right)^2\right)$ $\quad T_i > 0,\ \mu \in \mathbb{R},\ \sigma > 0$

Example: weights

— here we decide to treat the censored data as missing values data (instead of doing ✱)

A potato

A scale



The weights example

It is **not only about the time**, we can censor other stuffs! For example, the same mechanism works also if you consider the weights problem:

- We have a set of **potatoes**, with weights distributed as $W_i \sim Exp(\theta)$
- We have a scale, with an upper limit c
- We have censored observations for the potatoes with weights larger than c

We are interested about inference for the parameter θ

- **Conjugate prior** $\pi(\theta) \sim Gamma(a_0, b_0)$
- **Likelihood**

$$\mathcal{L}(W_1, \dots W_n \mid \delta_1, \delta_n, \theta) = \prod_{i:\delta_i=1} \theta e^{-\theta W_i} \times \prod_{i:\delta_i=0} e^{-\theta W_i}$$

Augment the problem

We have a sample $\{(W_1, \delta_1), \dots, (W_n, \delta_n)\}$ of **censored weights** and **censor variables**.

- If the i-th observation is censored, we do not know what is the i-th weight
- We handle the i-th weight as a missing value

Core idea behind the algorithm:
a **Gibbs sampler** for $\pi(\theta, \mathbf{W}^{(miss)} \mid \mathbf{W}^{(obs)})$ where

1. We sample $\mathbf{W}^{(miss)} \sim f(W \mid theta)$
2. We sample $\theta \sim \pi(\theta \mid \mathbf{W}^{(miss)}, \mathbf{W}^{(obs)})$

27/11/2020
3/9

See the code!

Example 2.

Kevlar fiber

Spool

The data

We have a set of data composed by 108 **kevlar fiber lifetimes**, coming from the combination of **eight different spools** and **four levels of stress** (pressure).

A subset of the data (11 observations) are censored.

We are interested into model the lifetimes of the kevlar fibers, taking into account the different spools and the stress levels.

First let's see some preliminary analysis..

See the code!

PH vs AFT models

Two main family of models for survival data

- **Proportional hazard models**: the effect of the covariates is multiplicative on the hazard rate

$$h(T_i \mid \beta, \mathbf{x}_i) = e^{\beta^\intercal \mathbf{x}_i} h_0(t_i)$$

where $h_0(t_i)$ is a baseline hazard rate

- **Accelerated failure time models**: the effect of the covariate is multiplicative in the lifetime, which in terms of the hazard rate can be written as

$$h(T_i \mid \beta, \mathbf{x}_i) = e^{\beta^\intercal \mathbf{x}_i} h_0\left(e^{\beta^\intercal \mathbf{x}_i} t_i\right)$$

If $\beta^\intercal \mathbf{x}_i < 0$ the effect of the covariates is to decelerate the time, whereas a $\beta^\intercal \mathbf{x}_i > 0$ accelerates the time

distortion at hazard level

distortion at hazard level and lifetime level (time level)

The Weibull case

We consider a **Weibull distribution** with the following parametrization

$$X \sim Wei(\kappa, \theta), \qquad \kappa, \theta \in \mathbb{R}^+$$

and the following related quantities

$$\begin{aligned}
f(x; \kappa, \theta) &= \frac{\kappa}{\theta}\left(\frac{x}{\theta}\right)^{\kappa-1} \mathrm{e}^{-\left(\frac{x}{\theta}\right)^{\kappa}} \\
F(x; \kappa, \theta) &= 1 - \mathrm{e}^{-\left(\frac{x}{\theta}\right)^{\kappa}} \\
S(x; \kappa, \theta) &= \mathrm{e}^{-\left(\frac{x}{\theta}\right)^{\kappa}} \\
h(x; \kappa, \theta) &= \frac{\kappa}{\theta}\left(\frac{x}{\theta}\right)^{\kappa-1} \\
\mathbb{E}[X] &= \theta \Gamma\left(1 + \frac{1}{\kappa}\right)
\end{aligned}$$

The Weibull case

When we use a **Weibull baseline hazard** $h_0(t)$, i.e.

$$h_0(t) = \frac{\kappa}{\theta^\kappa} t^{\kappa-1}$$

in this case the two approaches (proportional hazard models and accelerated failure time models) coincide !

both the proportional hazards and the accelerated life models coincide, provided the covariates are constant (see Cox and Oakes, 1984, p.71).

Using for example a **Proportional hazard models** we have

$$h(T_i \mid \beta, \mathbf{x}_i) = e^{\beta^\intercal \mathbf{x}_i} \frac{\kappa}{\theta^\kappa} t^{\kappa-1} = e^{\beta^\intercal \mathbf{x}_i - \kappa log(\theta)} \kappa t^{\kappa-1}$$

A multiplicative effect on the hazard rate is equivalent to an effect on the parameter θ as in the previous, where now we have that

$$T_i \sim Weibull(\kappa, e^{\mu - \frac{\beta^\intercal \mathbf{x}_i}{\kappa}})$$

by setting $\mu = log(\theta)$, with $\mu \in \mathbb{R}$.

27/11/2020
419

The Weibull case: the model

We have data for different spools, we use a random effects model, with a specific intercept for each spool. **Model specification** :

$$
\begin{aligned}
T_i \mid g_j, x_i, \beta, \mu_j, \kappa &\sim \textit{Weibull}\left(\kappa, e^{\mu_j - \frac{\beta^\intercal x_i}{\kappa}}\right) && i = 1, \dots, n_j,\ j = 1, \dots G \\
\mu_j \mid m_0, \sigma^2_\mu &\sim N(m_0, \sigma^2_\mu), && j = 1, \dots, G \\
\beta \mid b_0, \sigma^2_\beta &\sim N(b_0, \sigma^2_\beta) \\
\sigma^2_\beta \mid a_\beta, b_\beta &\sim IG(a_\beta, b_\beta) \\
\kappa \mid a_\kappa, b_\kappa &\sim \textit{Gamma}(a_\kappa, b_\kappa)
\end{aligned}
$$

We compare the model with

- the case without random effects
- the case without covariates (only intercept)

See the code!

Example 3.

The data

We have a set of data composed by 456 **people**, enrolled in drug addiction rehabilitation treatment.

A subset of the data (111 observations) are censored.

We are interested into model the time to events, represented by the possibly relapse of the drug. People lost to follow up are considered as events, people out of the study without events as censored.

We known the drugs usage history of the patients, and we have four different groups: Heroin and Cocaine, Heroin only, Cocaine only, no one of the previous. We use these groups to include random effects in the model.

We want to include in the model also the **age** and the **length of treatment** as covariates, possibly with random regression coefficients.

The Weibull case: the model

We use a random effects model with random regression coefficients. **Model specification** :

$$
\begin{aligned}
T_i \mid g_j, x_i, \beta, \mu_j, \kappa &\sim \textit{Weibull}\left(\kappa, e^{\mu_j - \frac{\beta_j^\intercal x_i}{\kappa}}\right) && i = 1, \dots, n_j,\ j = 1, \dots G \\
\mu_j \mid m_0, \sigma^2_\mu &\sim N(m_0, \sigma^2_\mu), && j = 1, \dots, G \\
\beta_j \mid b_0, \sigma^2_\beta &\sim N(b_0, \sigma^2_\beta), && j = 1, \dots, G \\
\sigma^2_\beta \mid a_\beta, b_\beta &\sim IG(a_\beta, b_\beta) \\
\kappa \mid a_\kappa, b_\kappa &\sim \textit{Gamma}(a_\kappa, b_\kappa)
\end{aligned}
$$

We compare the model with

- the case without random effects
- the case without covariates (only intercept)

Drugs_dummy.stan

```
data {
    int<lower = 0> N;                   // number of observations
    int<lower = 0, upper = 1> cens[N];    // censoring
    real<lower = 0> time[N];              // Failure times
}

parameters {
    real mu;            // random effects
    real<lower=0> lambda;           // variance on the mu
    real<lower=0> kappa;            // second parameter of the Weibull (shape)
}

model {
    // likelihood failure times as exact or censored
    for ( j in 1:N )
    {
        if (cens[j] == 0)
        {
            time[j] ~ weibull(kappa, exp(mu));
        }
        else
        {
            target += weibull_lccdf(time[j]| kappa, exp(mu));
        }
    }


    mu ~ normal(0, pow(lambda, 0.5)); // iid
    lambda ~ inv_gamma(5, 10);
    kappa ~ gamma(1, 0.2);
}

generated quantities {
    real log_lik[N];
    for (i in 1:N) {
        if (cens[i] == 0)
        {
            log_lik[i] = weibull_lpdf(time[i] | kappa, exp(mu));
        }
        else
        {
            log_lik[i] = weibull_lccdf(time[i]| kappa, exp(mu));
        }
    }
}
```

Hide

Drugs_fixed.stan

```
data {
    int<lower = 0> N;                   // number of observations
    int<lower = 0> p;           // number of covariates

    int<lower = 0, upper = 1> cens[N];    // censoring
    real<lower = 0> time[N];              // Failure times
    matrix[N, p] X;             // design matrix
}

parameters {
    vector[p] beta;         // coefficient for log(stress)
    real mu;            // random effects
    real<lower=0> lambda;           // variance on the mu
    real<lower=0> kappa;            // second parameter of the Weibull (shape)
}

model {
    // likelihood failure times as exact or censored
    for ( j in 1:N )
    {
        if (cens[j] == 0)
        {
            time[j] ~ weibull(kappa, exp(mu - row(X, j) * beta / kappa ));
        }
        else
        {
            target += weibull_lccdf(time[j]| kappa, exp(mu - row(X, j) * beta / kappa ));
        }
    }


    mu ~ normal(0, pow(lambda, 0.5)); // iid
    for(k in 1:p){
        beta[k] ~ normal(0, 10);
    }
    lambda ~ inv_gamma(5, 10);
    kappa ~ gamma(1, 0.2);
}

generated quantities {
    real log_lik[N];
    for (i in 1:N) {
        if (cens[i] == 0)
        {
            log_lik[i] = weibull_lpdf(time[i] | kappa, exp(mu - row(X, i) * beta / kappa ));
        }
        else
        {
            log_lik[i] = weibull_lccdf(time[i]| kappa, exp(mu - row(X, i) * beta / kappa ));
        }
    }
}
```

Hide

```
## Drugs_random.stan

data {
    int<lower = 0> N;                   // number of observations
    int<lower = 0> ngr;          // number of groups
    int<lower = 0> p;            // number of covariates

    int<lower = 0, upper = 1> cens[N];      // censoring
    real<lower = 0> time[N];                // Failure times

    int<lower = 1, upper = 8> group[N];     // group allocation
    matrix[N, p] X;              // design matrix
}

parameters {
    matrix[p, ngr] beta;             // coefficient for log(stress)
    vector[ngr] mu;              // random effects
    real<lower=0> lambda;            // variance on the mu
    real<lower=0> kappa;             // second parameter of the Weibull (shape)
}

model {
    // likelihood failure times as exact or censored
    for ( j in 1:N )
    {
        if (cens[j] == 0)
        {
            time[j] ~ weibull(kappa, exp(mu[group[j]] - row(X, j) * col(beta, group[j]) / kappa ));
        }
        else
        {
            target += weibull_lccdf(time[j]| kappa, exp(mu[group[j]] - row(X, j) * col(beta, group[j]) /
 kappa ));
        }
    }


    // Prior values
    for(g in 1:ngr){
        mu[g] ~ normal(0, pow(lambda, 0.5)); // iid
        for(k in 1:p){
            beta[k,g] ~ normal(0, 10);
        }
    }
    lambda ~ inv_gamma(5, 10);
    kappa ~ gamma(1, 0.2);
}

generated quantities {
    real log_lik[N];
    for (i in 1:N) {
        if (cens[i] == 0)
        {
            log_lik[i] = weibull_lpdf(time[i] | kappa, exp(mu[group[i]] - row(X, i) * col(beta, group[i])
/ kappa ));
        }
        else
        {
            log_lik[i] = weibull_lccdf(time[i]| kappa, exp(mu[group[i]] - row(X, i) * col(beta, group[i])
/ kappa ));
        }
    }
}
```

Hide

```
## kevlar_dummy.stan

data {
    int<lower = 0> N;                   // number of observations
    int<lower = 0, upper = 1> cens[N];     // censoring
    real<lower = 0> time[N];               // Failure times
}

parameters {
    real mu;         // random effects
    real<lower=0> lambda;    // variance on the mu
    real<lower=0> kappa;     // second parameter of the Weibull (shape)
}

model {
    // likelihood failure times as exact or censored
    for ( j in 1:N )
    {
        if (cens[j] == 0)
        {
            time[j] ~ weibull(kappa, exp(mu));
        }
        else
        {
            target += weibull_lccdf(time[j]| kappa, exp(mu));  // right-censored data
        }
    }


    // Prior values
    kappa ~ gamma(1, 0.2);
    mu ~ normal(0, pow(lambda, 0.5));
    lambda ~ inv_gamma(10, 12);
}

generated quantities {
    real log_lik[N];
    for (i in 1:N) {
        if (cens[i] == 0)
        {
            log_lik[i] = weibull_lpdf(time[i] | kappa, exp(mu));
        }
        else
        {
            log_lik[i] = weibull_lccdf(time[i]| kappa, exp(mu));
        }
    }
}
```

Hide

```
## kevlar_fixed.stan

data {
    int<lower = 0> N;                // number of observations
    int<lower = 0, upper = 1> cens[N];    // censoring
    real<lower = 0> time[N];              // Failure times
    real<lower = 0> stress[N];            // Pressure
}

parameters {
    real beta;          // coefficient for log(stress)
    real mu;        // fixed effects
    real<lower=0> lambda;   // variance on the mu
    real<lower=0> kappa;    // second parameter of the Weibull (shape)
}

model {
    // likelihood failure times as exact or censored
    for ( j in 1:N )
    {
        if (cens[j] == 0)
        {
            time[j] ~ weibull(kappa, exp(mu - beta*log(stress[j]) / kappa));
        }
        else
        {
            target += weibull_lccdf(time[j]| kappa, exp(mu - beta*log(stress[j]) / kappa));  // right-cen
sored data
        }
    }

    // Prior values
    beta ~ normal(0, 10);
    kappa ~ gamma(1, 0.2);
    mu ~ normal(0, pow(lambda, 0.5));
    lambda ~ inv_gamma(10, 12);
}

generated quantities {
    real log_lik[N];
    for (i in 1:N) {
        if (cens[i] == 0)
        {
            log_lik[i] = weibull_lpdf(time[i] | kappa, exp(mu - beta*log(stress[i]) / kappa));
        }
        else
        {
            log_lik[i] = weibull_lccdf(time[i]| kappa, exp(mu - beta*log(stress[i]) / kappa));
        }
    }
}
```

Hide

```
## kevlar_random.stan

data {
    int<lower = 0> N;                // number of observations
    int<lower = 0, upper = 1> cens[N];    // censoring
    real<lower = 0> time[N];              // Failure times
    int<lower = 1, upper = 8> spool[N];   // Spool indicators
    real<lower = 0> stress[N];            // Pressure
}

parameters {
    real beta;          // coefficient for log(stress)
    vector[8] mu;       // random effects
    real<lower=0> lambda;   // variance on the mu
    real<lower=0> kappa;    // second parameter of the Weibull (shape)
}

model {
    // likelihood failure times as exact or censored
    for ( j in 1:N )
    {
        if (cens[j] == 0)
        {
            time[j] ~ weibull(kappa, exp(mu[ spool[j] ] - beta*log(stress[j]) / kappa ));
        }
        else
        {
            target += weibull_lccdf(time[j]| kappa, exp(mu[ spool[j] ] - beta*log(stress[j]) / kappa ));
// right-censored data
            // The log of the Weibull complementary cumulative distribution (survival) function of y give
n shape mu and scale sigma
        }
    }

    //## Prior for the spool effect
    mu ~ normal(0, pow(lambda, 0.5)); // iid
    lambda ~ inv_gamma(5, 10);

    // Prior values
    beta ~ normal(0, 10);
    kappa ~ gamma(1, 0.2);
}

generated quantities {
    real log_lik[N];
    for (i in 1:N) {
        if (cens[i] == 0)
        {
            log_lik[i] = weibull_lpdf(time[i] | kappa, exp(mu[ spool[i] ] - beta*log(stress[i]) / kappa
));
        }
        else
        {
            log_lik[i] = weibull_lccdf(time[i]| kappa, exp(mu[ spool[i] ] - beta*log(stress[i]) / kappa
));
        }
    }
}
```

```r
# R & STAN are friends!
library(rstan)
library(coda)
library(survival)

# for plots
library(ggplot2)
library(tidyr)
library(dplyr)
library(purrr)
library(ggsci)
require(gplots)
require(ggpubr)

# useful function ----------------------------------------------------

my_WAIC <- function(fit, param){

  llik   <- rstan::extract(fit, param)[[1]]
  p_WAIC <- sum(apply(llik, 2, var))
  lppd   <- sum(apply(llik, 2, function(x) log(mean(exp(x)))))
  WAIC   <- - 2 * lppd + 2 * p_WAIC
  return(WAIC)

}

# ---------------------------------
# example on censored data:

load("potatoes.dat")
head(potatoes)

# ---------------------------------
# GIBBS SAMPLER strategy
# augment the data

gibbs_surv <- function(data, cens, niter, burnin, thin, a0, b0){

  # initialize the quantities
  results <- c()
  data_cens <- matrix(0, nrow = (niter - burnin) / thin, ncol = sum(!cens))
  aug_data <- data
  an <- a0 + length(data)

  pb <- txtProgressBar(0, niter, style = 3)
  for(i in 1:niter){
    # sample theta from the posterior
    bn <- b0 + sum(aug_data)
    theta <- rgamma(1, an, rate = bn)

    # sample the missing data
    aug_data[!cens] <- data[!cens] + rexp(sum(!cens), rate = theta)

    # save the results
    if(i > burnin & ((i - burnin) %% thin == 0)){
      results[(i - burnin)/thin] <- theta
      data_cens[(i - burnin)/thin,] <- aug_data[!cens]
    }
    setTxtProgressBar(pb, i)
  }
  close(pb)
  return(list(results, data_cens))
}

# ---------------------------------
# SAMPLE A CHAIN
# a0 - "prior sample size"
# b0 - "prior guess on theta"

sample1 <- gibbs_surv(potatoes$potatoes, potatoes$cens, 10000, 1000, 5, 1, 50)
sample_coda <- as.mcmc(sample1[[1]])

#---------------------------------
# DIAGNOSTIC
summary(sample_coda)
plot(sample_coda)
acfplot(sample_coda)
geweke.diag(sample_coda, frac1=0.1, frac2=0.5)

#---------------------------------
# POSTERIOR

p1 <- ggplot(data.frame(value = sample1[[1]]), aes(value)) +
  geom_density(fill = "orange", alpha = 0.5) +
  theme_minimal() +
  theme(legend.position="none") +
  geom_vline(aes(xintercept = 1 / mean(potatoes$potatoes)), col = 2, lty = 2)
p1

#---------------------------------
# AUGMENTED DATA

plot_df <- sample1[[2]] %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_df %>%
  ggplot(aes(y = value, group = param)) +
  geom_boxplot(colour = "black", fill = "orange") +
  coord_flip() +
  theme(legend.position="none") +
  ggtitle("Boxplot of augmented weights by single obs") +
  scale_y_continuous(breaks = c(250, 500, 750, 1000, 1250), name = "Augmented values") +
  theme_minimal() +
  theme(axis.text.y=element_text(colour="white", size = 12))

# ---------------------------------
# GIBBS SAMPLER strategy
# do not augment the data

gibbs_surv2 <- function(data, niter, burnin, thin, a0, b0){

  # initialize the quantities
  results <- c()
  an <- a0 + length(data)

  pb <- txtProgressBar(0, niter, style = 3)
  for(i in 1:niter){
    # sample theta from the posterior
    bn <- b0 + sum(data)
    theta <- rgamma(1, an, rate = bn)

    # save the results
    if(i > burnin & ((i - burnin) %% thin == 0)){
      results[(i - burnin)/thin] <- theta
    }
    setTxtProgressBar(pb, i)
  }
  close(pb)
  return(results)
}

sample2 <- gibbs_surv2(potatoes$potatoes, 10000, 1000, 5, 1, 50)
sample_coda <- as.mcmc(sample2)

#---------------------------------
# DIAGNOSTIC
summary(sample_coda)
plot(sample_coda)
acfplot(sample_coda)
geweke.diag(sample_coda, frac1=0.1, frac2=0.5)

#---------------------------------
# POSTERIOR

p2 <- ggplot(data.frame(value = sample2), aes(value)) +
  geom_density(fill = "blue", alpha = 0.5) +
  theme_minimal() +
  theme(legend.position="none")+
  geom_vline(aes(xintercept = 1 / mean(potatoes$potatoes)), col = 2, lty = 2)
ggarrange(p1, p2)

# ---------------------------------
```

```
# ----------------------------------
# KEVLAR DATA
# no Bayes :(
# ----------------------------------
# ----------------------------------

dati = read.table("kevlar.txt", header = T)
attach(dati)
unique(Spool) # 8 different spools
unique(Stress) # 4 levels of pressure
dim(dati)

# ----------------------------------
# Add the information about censoring: the last 11 observations are censored
cens <- c(rep(1, 97), rep(0, 11))
# 1 ---> non censored, 0 ---> censored

# ----------------------------------
# Function SURV: Create a survival object, frequently used as a response
# variable in a model formula (here, you can specify different types of
# censoring: to the right, to the left, interval..)
# Divide Time and stress for a constant to avoid numerical problems
tempi <- Surv(dati$Time/(10^3), cens)
tempi

# ----------------------------------
# Kaplan - Maier estimator of the survival function
# based on data, without considering covariate information
fit0 <- survfit(tempi ~ 1)
plot(fit0, col = "darkgreen", lwd = 2)

# - Every jump corresponds to one (or more) observation
# - Steep decreasing slope at the beginning

# ----------------------------------
# Kaplan - Meier estimator when stress is taken as a covariate
fit1 = survfit(tempi ~ Stress)
summary(fit1)

# Returns a list containing the survival curve, confidence limits for the curve, and other information.
# - time: the timepoints on the curve.
# - n.risk: the number of subjects at risk
# - n.event:  it is the cumulative number of events that have occurred since the last time

# ----------------------------------
# NOTE: If log = T, the survival function on the y-axis is showed in log scale
colors = rainbow(length(unique(Stress)))
plot(fit1, col = colors, lwd = 2)
legend("topright", legend = unique(Stress)[4:1], col = colors, lwd = 2)
# It seems that if the pressure increases, survival times fall dramatically!

# ----------------------------------
# For the details on the option "fun" see the help plot.survfit
# --> "cloglog" creates a complimentary log-log survival plot
# (f(y) = log(-log(y)) ).
## Drawing the lines of the log-log graph, estimating the regressions with Weibull error,
# Within each group identified by the value of Stress
fit1 = survfit(Surv(Time) ~ Stress, data = dati)
plot(fit1, fun = "cloglog", xlab='survival times', ylab='log H(t)', col = colors)
str = unique(Stress)
for(j in 1:4){
  sel <- (dati$Stress == str[j])
  time.sub <- dati$Time[sel]
  cens.sub <- cens[sel]
  reg.sub <- survreg(Surv(time.sub,cens.sub) ~ 1, dist = "weibull")
  lines(Time, 1/reg.sub$scale*(log(Time) - reg.sub$coefficients), col="gray", lwd = 2)
}

# ----------------------------------
# Create 8 dummy variables to consider the different spools
# --> we have a regression with 7 parameters alpha_2,...alpha_8,
# corrisponding to the covariate Spool
# We use the function "survreg", that runs a regression where the response is log(survival time)
# [with Weibull distribution, logarithmic link is automatically considered]
fit2 <- survreg(formula = tempi ~ 0 + log(Stress/23.4)  + factor(Spool), dist = "weibull")
coefML <- fit2$coefficients[2:9]
coefMLs <- fit2$coefficients[1]
summary(fit2)

# ----------------------------------
# Plot the residuals:
w = fit2$scale * (log(dati$Time / 10^3) - fit2$linear)
par(mfrow = c(1,2))
plot(w)
abline(h = 0, col = "red")
title(paste("Generalized residuals"))
hist(w, nclass = "fd")

# ----------------------------------
# ----------------------------------
# KEVLAR DATA
# with Bayes :)
# ----------------------------------
# ----------------------------------
dev.off()

# Divide Time and stress for a constant to avoid numerical problems
dati_list = list(N = 108, time = Time/(10^3),
                 spool = Spool,
                 cens = c(rep(0, 97), rep(1,11)),
                 stress = Stress/(23.4))
hist(log(Time/(10^3)), nclass = 30, freq = F, xlab = "time", main = "")

#-----------------------------------
# STAN model

inits <- function() {
  list(beta = 1, kappa = 2, lambda = 10, mu = rep(0,8))
}

model = stan(file = "kevlar_random.stan",
             data = dati_list,
             chains = 1,
             iter = 50000,
             warmup = 10000,
             thin= 10,
             seed = 1919,
             init = inits,
             control = list(max_treedepth = 20, adapt_delta = 0.99))
# save(model, file="model.dat")
load("model.dat")

# Final iterations: 4000
# Posterior mean:
get_posterior_mean(model, pars = c("beta", "mu", "lambda", "kappa"))
is(model)

#-----------------------------------
# What about the chains
rstan::traceplot(model, pars = "beta", inc_warmup = TRUE)
rstan::traceplot(model, pars = "mu", inc_warmup = TRUE)
rstan::traceplot(model, pars = "lambda", inc_warmup = TRUE)
rstan::traceplot(model, pars = "kappa", inc_warmup = TRUE)

#-----------------------------------
# Random effects
plot(model, ask = T, pars = c("mu"))

#-----------------------------------
# Diagnostic
coda_chain <- As.mcmc.list(model, pars = c("beta", "mu", "lambda", "kappa"))
summary(coda_chain)
acfplot(coda_chain)
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

#-----------------------------------
# Posterior
plot_post <- model %>%
  rstan::extract(c("kappa", "lambda", "beta")) %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
```

```
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")

plot_post <- model %>%
  rstan::extract(c("mu")) %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free', ncol = 4) +
  theme_minimal() +
  xlim(c(0,7)) +
  theme(legend.position="none") +
  geom_vline(data = data.frame(param = unique(plot_post$param), val = coefML),
             mapping = aes(xintercept = coefML, group = param), col = 2, lty = 2)

#-----------------------------------
# Bayesian residuals
coda_df <- as.data.frame(as.matrix(coda_chain))
attach(coda_df)

names(coda_df)
mu_temp <- coda_df[, 2:9]
nit_saved <- dim(mu_temp)[1]

N = nrow(dati)
W <- matrix(c(rep(0, (N - 11) * nit_saved)), nit_saved, (N - 11))
time_new = Time/(10^3)
stress_new = Stress/(23.4)

for (i in 1 :(N - 11)){
  sp <- Spool[i]
  for (j in 1:nit_saved){
    W[j,i] = (log(time_new[i]) - (mu_temp[j, sp] -
           (coda_df[j,1]*log(stress_new[i])) / kappa[j] + lgamma(1 + 1 / kappa[j]))) ## calcolo residuo
  }
}

#-----------------------------------
# Mean residuals
resbay <- colMeans(W)
summary(resbay)
par(mfrow = c(1,2))
plot(resbay)
abline(h = 0, col = "red")
title(paste("Bayesian residuals"))
hist(resbay, nclass = "fd")
dev.off()

#-----------------------------------
# COMPUTE THE PREDICTIVE FOR A RANDOM SPOOL
## --> when the spool is not known
## Level of pressure: 23.4

ngr <- 200 # grid points
TMAX = 10
temp <- seq(0, TMAX, length.out = ngr)    ## ngr times in (0, TMAX)

## pressure 1 = 23.4
press1 <- 23.4/23.4
pred1 <- matrix(0, nit_saved, ngr)       ## matrix: cdf for each time and iteration

# Keep all values so you can calculate the median and quantiles
for(j in 1 : nit_saved){
  newspool_effect <- rnorm(1, mean = 0, sd = sqrt(lambda[j])) # Random Effect of a new spool from the sam
e population
  etanew = exp(newspool_effect - (coda_df[j,1] * log(press1) / kappa[j]))
  pred1[j,] <- exp(-(temp/etanew)^kappa[j])    ## CDF evaluated in the grid, conditioning on the state of
the chain at iteration j
}

## pressure 2 = 22.5
press2 <- 22.5/23.4
pred2 <- matrix(0, nit_saved, ngr)
for(j in 1 : nit_saved){
  newspool_effect <- rnorm(1, mean=0, sd=sqrt(lambda[j]))
  etanew = exp(newspool_effect - (coda_df[j,1] * log(press2) / kappa[j]))
  pred2[j,] <- exp(-(temp/etanew)^kappa[j])
  ## CDF evaluated in the grid, conditioning on the state of the chain at iteration j
}

## pressure 3 = 28.5
press3 <- 28.5/23.4
pred3 <- matrix(0, nit_saved, ngr)
for(j in 1 : nit_saved){
  newspool_effect <- rnorm(1, mean=0, sd=sqrt(lambda[j]))
  etanew = exp(newspool_effect - (coda_df[j,1] * log(press3) / kappa[j]))
  pred3[j,] <- exp(-(temp/etanew)^kappa[j])
}

#-----------------------------------
# COMPUTE THE PREDICTIVE FOR A RANDOM SPOOL
## --> when the spool is not known
## Level of pressure: 23.4

df_plot <- data.frame(mean = apply(t(cbind(pred1, pred2, pred3)), 1, mean),
                      press = rep(c("23.4", "22.5", "28.5"), each = 200),
                      x = rep(temp, 3))
ggplot(df_plot) +
  geom_line(aes(x = x, y = mean, col = press)) +
  theme(legend.position="none") +
  theme_minimal()+
  ylab("Survival")

# ZOOM
ggplot(df_plot) +
  geom_line(aes(x = x, y = mean, col = press)) +
  theme(legend.position="none") +
  theme_minimal() +
  xlim(c(0.5, 1.5))+
  ylab("Survival")
# With a smaller prssure (blue) the survival function is higher

#-----------------------------------
# PREDICTIVE FOR A KNOWN SPOOL
## new pressure = 24.5
press_new <- 24.5/23.4
pred_new <- matrix(c(rep(0, (ngr)*nit_saved)),nit_saved, (ngr))

#------------ Let's consider spool number 6 ------------
for(j in 1 : nit_saved){
  ## from the spool population
  etanew = exp(coda_df[j,4] - coda_df[j,1] * log(press_new) / kappa[j])    ## compute etanew
  pred_new[j,] <- exp(-(temp/etanew)^kappa[j])
}
pred_newpost <- colMeans(pred_new)

## SURVIVAL FUNCTION
df_new_press <- data.frame(mean = colMeans(pred_new),
                           low = apply(pred_new, 2, quantile, 0.1),
                           up = apply(pred_new, 2, quantile, 0.9),
                           x = temp)

ggplot(df_plot) +
  geom_line(aes(x = x, y = mean, col = press)) +
  theme(legend.position="none") +
  theme_minimal() +
  geom_line(data = df_new_press, mapping = aes(x = x, y = mean), lty = 2) +
  geom_ribbon(data = df_new_press, mapping = aes(x = x, ymin = low, ymax = up), alpha = 0.2) +
  ylab("Survival")

#-----------------------------------
# MODEL COMPARISON - only stress
dati_list = list(N = 108, time = Time/(10^3),
                 cens = c(rep(0, 97), rep(1,11)),
                 stress = Stress/(23.4))

#-----------------------------------
# STAN model
```

```
inits <- function() {
  list(beta = 1, kappa = 2, lambda = 10)
}

model_fix = stan(file = "kevlar_fixed.stan",
                 data = dati_list,
                 chains = 1,
                 iter = 50000,
                 warmup = 10000,
                 thin= 10,
                 seed = 1919,
                 init = inits,
                 control = list(max_treedepth = 20, adapt_delta = 0.99))
# save(model_fix, file="model_fix.dat")
load("model_fix.dat")

#-----------------------------------
# DIAGNOSTIC
rstan::traceplot(model_fix, pars = c("kappa", "mu", "beta"), inc_warmup = TRUE)

#-----------------------------------
# CODA
coda_chain_fix <- As.mcmc.list(model_fix, pars = c("lambda", "mu", "kappa", "beta"))
summary(coda_chain_fix)
acfplot(coda_chain_fix)
geweke.diag(coda_chain_fix, frac1=0.1, frac2=0.5)

#-----------------------------------
# Posterior
plot_post <- model_fix %>%
  rstan::extract(c("kappa", "mu", "beta")) %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")

#-----------------------------------
# MODEL COMPARISON - dummy
dati_list = list(N = 108, time = Time/(10^3),
                 cens = c(rep(0, 97), rep(1,11)))

#-----------------------------------
# STAN model
inits <- function() {
  list(kappa = 2, lambda = 10)
}

model_dummy = stan(file = "kevlar_dummy.stan",
                   data = dati_list,
                   chains = 1,
                   iter = 50000,
                   warmup = 10000,
                   thin= 10,
                   seed = 1919,
                   init = inits,
                   control = list(max_treedepth = 20, adapt_delta = 0.99))
# save(model_dummy, file="model_dummy.dat")
load("model_dummy.dat")

#-----------------------------------
# DIAGNOSTIC
rstan::traceplot(model_dummy, pars = c("kappa", "mu"), inc_warmup = TRUE)

#-----------------------------------
# Diagnostic
coda_chain_dummy <- As.mcmc.list(model_dummy, pars = c("kappa", "mu"))
summary(coda_chain_dummy)
acfplot(coda_chain_dummy)
geweke.diag(coda_chain_dummy, frac1=0.1, frac2=0.5)

#-----------------------------------
```

```
# Posterior
plot_post <- model_dummy %>%
  rstan::extract(c("kappa", "mu")) %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free') +
  scale_fill_locuszoom() +
  theme_minimal() +
  theme(legend.position="none")

#-----------------------------------
# MODEL COMPARISON - WAIC

my_WAIC(model, "log_lik")
my_WAIC(model_fix, "log_lik")
my_WAIC(model_dummy, "log_lik")

#-----------------------------------
#--------
#-----------------------------------
#--------
#-----------------------------------
#--------
# AGE   Age at Enrollment   Years
# HC    Heroin/Cocaine Use During         1 = Heroin & Cocaine
# 3 Months Prior to Admission           2 = Heroin Only
#                                       3 = Cocaine Only
#                                       4 = Neither Heroin nor Cocaine
# LEN.T Length of Stay in Treatment Days
#                   (Admission Date to Exit Date)
# TIME (log scale)  Time to Drug Relapse    Days
#                   (Measured from Admission Date)
# CENSOR                                        1 = Returned to Drugs (or lost)
#                                           0 = Otherwise

load(file = "data_drugs.RData")
head(data_drugs)

dati_list = list(N = nrow(data_drugs),
                 ngr = 4,
                 p = 2,
                 time = round(exp(data_drugs$time)),
                 group = data_drugs$group,
                 cens = 1 - data_drugs$cens,
                 X = data_drugs[,4:5])

#-----------------------------------
# STAN model

model_drugs = stan(file = "Drugs_random.stan",
             data = dati_list,
             chains = 1,
             iter = 50000,
             warmup = 10000,
             thin= 10,
             seed = 1919,
             control = list(max_treedepth = 20, adapt_delta = 0.99))
# save(model_drugs, file="model_drugs.dat")
load("model_drugs.dat")

# Final iterations: 4000
# Posterior mean:
get_posterior_mean(model_drugs, pars = c("beta", "mu", "lambda", "kappa"))
is(model_drugs)

#-----------------------------------
# What about the chains
rstan::traceplot(model_drugs, pars = "beta", inc_warmup = TRUE)
rstan::traceplot(model_drugs, pars = "mu", inc_warmup = TRUE)
rstan::traceplot(model_drugs, pars = "lambda", inc_warmup = TRUE)
rstan::traceplot(model_drugs, pars = "kappa", inc_warmup = TRUE)

#-----------------------------------
```

```
# Random effects
plot(model_drugs, ask = T, pars = c("mu"))
plot(model_drugs, ask = T, pars = c("beta"))

#-----------------------------------
# Diagnostic
coda_chain <- As.mcmc.list(model_drugs, pars = c("beta", "mu", "lambda", "kappa"))
summary(coda_chain)
acfplot(coda_chain)
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

#-----------------------------------
# Posterior
plot_post <- model_drugs %>%
  rstan::extract(c("beta")) %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')
cv <- rep(c("AGE", "LENGTH", "AGE", "LENGTH", "AGE", "LENGTH", "AGE", "LENGTH"), each = 4000)
gr <- rep(c("gr 1", "gr 2", "gr 3", "gr 4"), each = 8000)

data.frame(plot_post, cv = cv, gr = gr) %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_grid(gr~cv, scales = 'free') +
  theme_minimal() +
  theme(legend.position="none")

plot_post <- model_drugs %>%
  rstan::extract(c("mu")) %>%
  as.data.frame() %>%
  map_df(as_data_frame, .id = 'param')

plot_post %>%
  ggplot(aes(value, fill = param)) +
  geom_density() +
  facet_wrap(~param, scales = 'free', ncol = 4) +
  theme_minimal() +
  xlim(c(4,6.5)) +
  theme(legend.position="none")

#---- model comparison

dati_list = list(N = nrow(data_drugs),
                 p = 2,
                 time = round(exp(data_drugs$time)),
                 cens = 1 - data_drugs$cens,
                 X = data_drugs[,4:5])

#-----------------------------------
# STAN model

model_drugs_fix = stan(file = "Drugs_fixed.stan",
                       data = dati_list,
                       chains = 1,
                       iter = 50000,
                       warmup = 10000,
                       thin= 10,
                       seed = 1919,
                       control = list(max_treedepth = 20, adapt_delta = 0.99))
# save(model_drugs_fix, file="model_drugs_fix.dat")
load("model_drugs_fix.dat")

#-----------------------------------
rstan::traceplot(model_drugs_fix, pars = "beta", inc_warmup = TRUE)
rstan::traceplot(model_drugs_fix, pars = "mu", inc_warmup = TRUE)
rstan::traceplot(model_drugs_fix, pars = "lambda", inc_warmup = TRUE)
rstan::traceplot(model_drugs_fix, pars = "kappa", inc_warmup = TRUE)

#-----------------------------------
# Diagnostic
coda_chain <- As.mcmc.list(model_drugs_fix, pars = c("beta", "mu", "lambda", "kappa"))
summary(coda_chain)
acfplot(coda_chain)
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)

#-----------------------------------
#-----------------------------------
```

```
#-----------------------------------

dati_list = list(N = nrow(data_drugs),
                 time = round(exp(data_drugs$time)),
                 cens = 1 - data_drugs$cens)

#-----------------------------------
# STAN model

model_drugs_dummy = stan(file = "Drugs_dummy.stan",
                         data = dati_list,
                         chains = 1,
                         iter = 50000,
                         warmup = 10000,
                         thin= 10,
                         seed = 1919,
                         control = list(max_treedepth = 20, adapt_delta = 0.99))
# save(model_drugs_dummy, file="model_drugs_dummy.dat")
load("model_drugs_dummy.dat")

#-----------------------------------
rstan::traceplot(model_drugs_dummy, pars = "mu", inc_warmup = TRUE)
rstan::traceplot(model_drugs_dummy, pars = "lambda", inc_warmup = TRUE)
rstan::traceplot(model_drugs_dummy, pars = "kappa", inc_warmup = TRUE)

#-----------------------------------
# Diagnostic
coda_chain <- As.mcmc.list(model_drugs_dummy, pars = c("mu", "lambda", "kappa"))
summary(coda_chain)
acfplot(coda_chain)
geweke.diag(coda_chain, frac1=0.1, frac2=0.5)


#-----------------------------------
# MODEL COMPARISON - WAIC
my_WAIC(model_drugs, "log_lik")
my_WAIC(model_drugs_fix, "log_lik")
my_WAIC(model_drugs_dummy, "log_lik")
```

4/12/2020
1/6

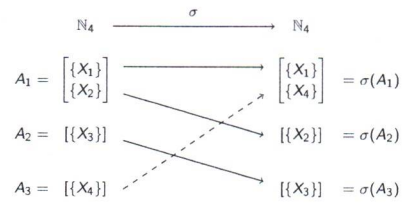
BNP1

Riccardo Corradin
December 4, 2020

de Finetti theorem and Pólya Urn

de Finetti Th. & Pólya urn

Let $\mathbb{X}$ be a Polish space, with $\mathscr{X}$ denoting its Borel σ-field. Let $\{X_1, \ldots, X_n\}$ be a finite sequence of $\mathbb{X}$-valued random elements defined on a common probability space $(\Omega, \mathscr{A}, \mathbb{P})$. We further assume the sequence exchangeable, in the sense that the distribution law of $\{X_1, \ldots, X_n\}$ is invariant w.r.t. any permutation of the elements $\sigma : \mathbb{N}_n \to \mathbb{N}_n$.



$$P(\{X_1, X_2\}, \{X_3\}, \{X_4\}) = P(\{X_1, X_4\}, \{X_2\}, \{X_3\})$$

de Finetti Th. & Pólya urn

We have a characterization of an exchangeable sequence thanks to the de Finetti representation theorem.

Theorem (de Finetti, 1937)

The sequence $\{X_1, \ldots, X_n\}$ is exchangeable if and only if there exists a probability measure Q such that

$$P[X_1 = x_1, \ldots, X_n = x_n] = \int \prod_{i=1}^{n} P[X_i = x_i \mid \theta] Q(\mathrm{d}\theta)$$

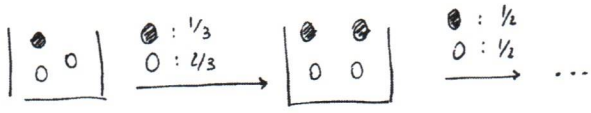
The theorem states that a sequence is exchangeable iff it is conditionally independent.

The measure Q is termed *de Finetti measure*, and it plays the role of prior distribution in Bayesian statistics.

is playing the role of prior distribution in Bayesian statistic

de Finetti Th. & Pólya urn

We consider a Pólya urn sampling scheme. Suppose we have an urn with balls of two colors, black and white. At each step we sample a ball, with probability proportional to the number of balls of the same color currently in the urn. Once we sampled a ball, we replace the ball in the urn with a new one of the same color.



We denote by:

- B the initial number of black balls;
- W the initial number of white balls;
- X_i the i-th Bernoulli sampling, where $X_i = \begin{cases} 1 : \text{black ball} \\ 0 : \text{white ball} \end{cases}$;
- S_n the number of black balls in n trials;
- $n - S_n$ the number of white balls in n trials;

de Finetti Th. & Pólya urn

Notice that a sequence sampled from a Pólya urn is exchangeable. We have:

$$P[\boldsymbol{X}_4 = \{1,0,1,1\}] = \frac{B}{B+W}\frac{W}{B+W+1}\frac{B+1}{B+W+2}\frac{B+2}{B+W+3}$$

$$P[\boldsymbol{X}_4 = \{1,1,0,1\}] = \frac{B}{B+W}\frac{B+1}{B+W+1}\frac{W}{B+W+2}\frac{B+2}{B+W+3}$$

and the two coincides. More in general, for a n-sized sample from a Pólya urn scheme with two distinct color we have

$$P[X_1, \dots X_n] = \frac{\Gamma(B+S_n)\Gamma(W+n-S_n)\Gamma(B+W)}{\Gamma(B)\Gamma(W)\Gamma(B+W+n)}$$

de Finetti Th. & Pólya urn

From the de Finetti theorem we have

$$P[X_1, \dots, X_n] = \int \prod_{i=1}^{n} \theta^{X_i}(1-\theta)^{1-X_i} Q(\mathrm{d}\theta) = \int \theta^{S_n}(1-\theta)^{n-S_n} Q(\mathrm{d}\theta)$$

Further, the measure Q is the cumulative distribution function of the limit frequency, which for the model we are considering corresponds to the cdf of

$$Y = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} X_i \sim Beta(B, W)$$

We can prove now that for an exchangeable sequence sampled from a Pólya urn sampling scheme, the de Finetti representation theorem holds.

de Finetti Th. & Pólya urn

proof.

We have

$$\begin{aligned} P[X_1, \dots, X_n] &= \int \theta^{S_n}(1-\theta)^{n-S_n} Q(\mathrm{d}\theta) \\ &= \int \theta^{S_n}(1-\theta)^{n-S_n} \frac{\theta^{B-1}(1-\theta)^{W-1}}{B(B,W)}\mathrm{d}\theta \\ &= \frac{B(B+S_n, W+n-S_n)}{B(B,W)} \int \frac{\theta^{B+S_n-1}(1-\theta)^{W+n-S_n-1}}{B(B+S_n, W+n-S_n)}\mathrm{d}\theta \\ &= \frac{\Gamma(B+S_n)\Gamma(W+n-S_n)\Gamma(B+W)}{\Gamma(B)\Gamma(W)\Gamma(B+W+n)}, \end{aligned}$$

we substitute to Q the corresponding measure that we stated for the limit frequency (in our case we're measuring w.r.t. a Beta)

← probability associated to an exchangeable sequence produced from a polya urn

which corresponds to the quantity of interest.

For an exchangeable sequence sampled from a Pólya urn the de Finetti theorem holds, and the de Finetti measure Q playing the role of prior distribution corresponds to a Beta distribution.

We can generalize the de Finetti representation theorem for a scenario with a infinite sequence and an infinite number of features. Let $X^{(\infty)}$ be an infinite sequence of observations, where the i-th element X_i of the sequence is taking values on a measurable space $(\mathbb{X}, \mathscr{X})$.

The sequence $X^{(\infty)}$ is exchangeable, for any $n \geq 1$ and any permutation σ of $\{1, \ldots, n\}$, $(X_1, \ldots, X_n)$ coincides in distribution with $(X_{\sigma(1)}, \ldots, X_{\sigma(n)})$.

Let $\mathbb{X}^{(\infty)} = \mathbb{X} \times \mathbb{X} \times \ldots$ and $\mathcal{P}$ be the space of probability measures on $(\mathbb{X}, \mathscr{X})$.

Theorem (de Finetti, 1937)

The sequence $X^{(\infty)}$ is exchangeable if and only if there exists a probability measure Q on $\mathbb{P}$ such that, for any $n \geq 1$, and $A = A_1 \times A_2 \times \cdots \times A_n \times \mathbb{X}^{(\infty)}$, with $A_i \in \mathcal{X}$, one has

$$\mathbb{P}\left[X^{(\infty)} \in A\right] = \int_{\mathbb{P}} \prod_{i=1}^{n} p(A_i) Q(\mathrm{d}p).$$

An introduction to the DP

distribution over distributions : distribution taking values o e space of probability distributions

DP - Introduction

Let $\mathbb{X}$ be a Polish space, with $\mathscr{X}$ denoting its Borel σ-field. We further denote by $\mathbb{P}$ the space of probability measures with support $\mathbb{X}$, endowed with its Borel σ-field $\mathscr{P}$.

Definition (Dirichlet process – Ferguson, 1973)

Let $\alpha(\cdot)$ be a finite, non-negative and diffuse measure defined on $(\mathbb{X}, \mathscr{X})$, with $\alpha(\mathbb{X}) < +\infty$. We say that P is distributed as a Dirichlet process with parameter $\alpha(\cdot)$ if for any partition $\{A_1, \ldots, A_k\}$ of $\mathbb{X}$ we have

$$(P(A_1), \ldots, P(A_k)) \sim Dir(\alpha(A_1), \ldots, \alpha(A_k)).$$

Notice that P is a random probability measure with support $\mathbb{P}$. We can further decompose the parameter of the DP in two terms $\alpha(\cdot) = aP_0(\cdot)$, with $P_0(\cdot) = \alpha(\cdot)/a$, which is termed base measure, and $a = \alpha(\mathbb{X})$, the strength / total mass parameter. Two different notations:

- $P \sim DP(\alpha(\cdot))$
- $P \sim DP(a, P_0(\cdot))$

DP - Moments

Let A be a measurable subset of $\mathbb{X}$, and $A^c = \mathbb{X} \setminus A$. Thanks to the definition of DP we have that

$$P(A) \sim Beta(\alpha(A), \alpha(A^c))$$

then we have:

- $\mathbb{E}[P(A)] = \frac{\alpha(A)}{\alpha(A)+\alpha(A^c)} = P_0(A)$
- $Var(P(A)) = \frac{\alpha(A)\alpha(A^c)}{(\alpha(A)+\alpha(A^c))^2(\alpha(A)+\alpha(A^c)+1)} = \frac{a^2 P_0(A)P_0(A^c)}{a^2(P_0(A)+P_0(A^c))(a+1)} = \frac{P_0(A)P_0(A^c)}{(a+1)}$

a plays the role of a precision parameter.

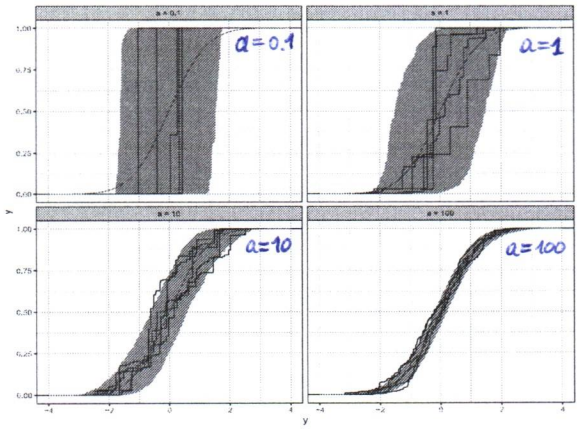
The Dirichlet process is a.s. discrete random probability measure with

$$P(\mathrm{d}t) = \sum_{j=1}^{\infty} W_j \delta_{X_j^*}(\mathrm{d}t)$$

double randomness inside the process: both $\{X_j^*\}_j$ and $\{W_j\}_j$ are random

with $\delta_{X_j^*}(\cdot)$ a Dirac measure in X_j^*, $\{W_j\}_{j\geq 1}$ a sequence of random weights, $\{X_j^*\}_{j\geq 1}$ a sequence of random elements i.i.d. from $P_0(\cdot)$, with $\{W_j\}_{j\geq 1} \perp\!\!\!\perp \{X_j^*\}_{j\geq 1}$

DP - Moments



if we increase a we increase the precision

DP - Posterior

Suppose we collected an exchangeable sample $X_1, \dots, X_n$ from a DP, showing $X_1^*, \dots, X_k^*$ distinct values with frequencies $n_1, \dots n_k$, i.e.

$$X_1, \dots, X_n \mid P \overset{\text{iid}}{\sim} P, \qquad P \sim DP(\alpha(\cdot)),$$

and we are interested into characterize the posterior distribution of P, given $X_1, \dots, X_n$. The posterior distribution is again a Dirichlet process, and is given by

$$P \mid X_1, \dots, X_n \sim DP\left(\alpha(\cdot) + \sum_{j=1}^{k} n_j \delta_{X_j^*}(\cdot)\right)$$

or equivalently $P \mid X_1, \dots, X_n \sim DP\left(aP_0(\cdot) + \sum_{j=1}^{k} n_j \delta_{X_j^*}(\cdot)\right)$

- posterior total mass

$$\int_{\mathbb{X}} aP_0(x) + \sum_{j=1}^{k} n_j \delta_{X_j^*}(x)\mathrm{d}x = a + n$$

- posterior expectation on a measurable set $A \subseteq \mathbb{X}$

$$\mathbb{E}[P(A) \mid X_1, \dots, X_n] = \frac{a}{a+n} P_0(A) + \frac{n}{a+n} \sum_{j=1}^{k} \frac{n_j}{n} \delta_{X_j^*}(A)$$

DP - Number of unique elements (aka number of clusters)

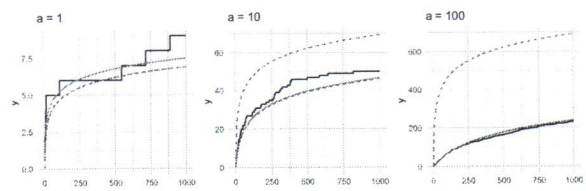
Let K_n be the number of unique elements out of a sample of size n. The expected number of elements out of a sample of size n is equal to

$$\mathbb{E}[K_n] = \sum_{j=1}^{n} \frac{a}{a+j-1} = a(\psi(a+n) - \psi(a)) \simeq a \log\left(\frac{a+n}{a}\right) \simeq a \log(n)$$

and

$$Var(K_n) = a(\psi(a+n) - \psi(a)) + a(\psi'(a+n) - \psi'(a)) \simeq a \log\left(\frac{a+n}{a}\right)$$

for $n \to \infty$, with $n > a \gg 0$ and $\psi(\cdot)$ denoting the digamma function.



DP - Self similarity

What happen when we restrict the DP to a subset of its support? For example, what is the distribution of $P\,|_B$ with $B \subset \mathbb{X}$?

Theorem (Self similarity)

Let $P \sim DP(a, P_0(\cdot))$ a Dirichlet process with support $\mathbb{P}_{\mathbb{X}}$, with $P_0(\cdot)$ a diffuse probability measure on $\mathbb{X}$. Let B a measurable subset of $\mathbb{X}$, and $B^c = \mathbb{X} \setminus B$. Then

$$P(B), \quad P(B^c), \quad P\,|_B, \quad P\,|_{B^c}$$

are mutually independent, for any $B : P_0(B) > 0$. Further

$$P\,|_B \sim DP(aP_0(B), P_0(\cdot)\,|_B)$$
$$P\,|_{B^c} \sim DP(aP_0(B^c), P_0(\cdot)\,|_{B^c})$$

!! A Dirichlet process restricted on a subset of its support is still distributed as a Dirichlet process, and is independent on what happens on the rest of the support.

DP - Self similarity

[PROOF] Consider a partition $A_1, \dots, A_m$ of B, and a partition $C_1, \dots, C_q$ of B^c. Then we have that over the two partitions joint together

$$\begin{aligned} \boldsymbol{X} &= (P(A_1), \dots, P(A_m), P(C_1), \dots, P(C_q)) \\ &\sim Dir(\alpha(A_1), \dots, \alpha(A_m), \alpha(C_1), \dots, \alpha(C_q)) \end{aligned}$$

Moreover, let

$$Z_1 = \sum_{j=1}^{m} X_i, \quad Z_2 = \sum_{j=m+1}^{m+q} X_i, \quad \boldsymbol{Y}_1 = \left(\frac{X_1}{Z_1}, \dots, \frac{X_m}{Z_1}\right), \quad \boldsymbol{Y}_2 = \left(\frac{X_{m+1}}{Z_2}, \dots, \frac{X_{m+q}}{Z_2}\right)$$

Thanks to the aggregation property of the Dirichlet distribution, the previous quantities are mutually independent. Further, $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ are defining two distinct Dirichlet process on B and B^c, with base measure $P_0(\cdot)\,|_B$ and $P_0(\cdot)\,|_{B^c}$ respectively, and total mass $aP_0(B)$ and $aP_0(B^c)$.

Chinese restaurant process = process of allocation of elements to different groups

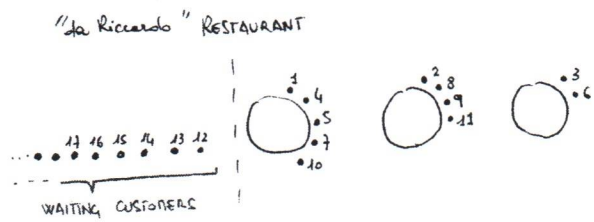
Suppose we have an infinite queue in front of a restaurant, with an infinite number of tables.

We are interested into modeling the table allocation of the customers, as far as they continue to enter in the restaurant.

Conditionally on the allocation of the first n customers, the next $n+1$ customer can sit to an already occupied table with probability proportional to the current number of customers at that table, or at a new table.

- For the j-th table denoting by n_j the current number of customers at table j, the probability that the $n+1$ customer will sit at the table j is proportional to n_j.
- The probability that the $n+1$ customer will sit at a new table is proportional to a.

Chinese restaurant process



The $n+1 = 12$ customer can sit at the first table with probability proportional to $n_1 = 5$, at the second table with probability proportional to $n_2 = 4$, at the third table with probability proportional to $n_3 = 2$, or at a new table with probability proportional to a.

I.e. we have $p_1 = \frac{n_1}{a+n_1+n_2+n_3} = \frac{5}{a+11}$,

and $p_2 = \frac{4}{a+11}$, $p_3 = \frac{2}{a+11}$, and $p_{new} = \frac{a}{a+11}$.

Chinese restaurant process

Let Z_i denotes the allocation of the i-th customer. We then have

$$P[Z_{n+1} = t \mid Z_1, \dots, Z_n] = \frac{a}{a+n}\delta_{k+1}(t) + \frac{n}{a+n}\sum_{j=1}^{k}\frac{n_j}{n}\delta_j(t)$$

→ probability of allocating the new customer to a new table

→ probability of allocating the new customer to one of the occupied tables (NOTICE: we've not making distinctions between groups)

Where k is the number of filled table by the first n customers, and n_j is the number of customers at the j-th table.

Remarks:

- CRP shows the "rich-gets-richer" phenomenon, where large clusters grow larger faster (the number of clusters grows logarithmic w.r.t. the number of observations);
- We are not interested in any specific feature characterizing each group;
- The allocation probability is in relation with a Dirichlet process;
- The probability associated to a specific allocation is

$$P(Z_1, \dots, Z_n) = \frac{a^k}{(a)_n}\prod_{j=1}^{k}(n_j - 1)!$$

with $(a)_b$ denoting the rising factorial.

Generalized Pólya urn

Suppose that we are now interested also in some features characterizing each group.

in the case of chinese Restaurant we were not interested in group's (table's) features and characteristics

For example, we are in an Pólya urn regime. At each step we can sample a ball of the j-th color with probability proportional to the current number of balls of color j.

But we have also a positive probability of sampling a ball of a new color, from another urn with an infinite number of balls $\alpha(\cdot)$, where each color is unique.

Given a sample $X_1, \dots, X_n$, with $X_1^*, \dots, X_k^*$ unique elements and $n_1, \dots, n_k$ frequencies, we have

$$P[X_{n+1} \in \mathrm{d}t \mid X_1, \dots, X_n] \propto \alpha(\mathrm{d}t) + \sum_{j=1}^{k} n_j \delta_{X_j^*}(\mathrm{d}t)$$

probability of sampling a specific color (or generally a specific value)

$\alpha(\mathrm{d}t)$ → P(new color)

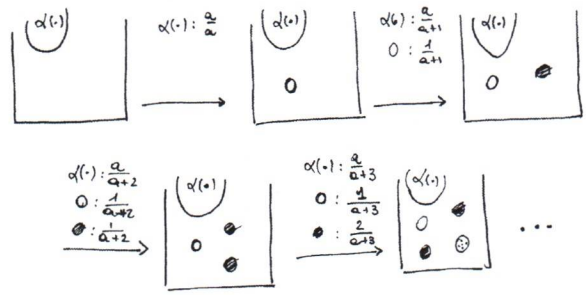
$\sum_{j=1}^{k} n_j \delta_{X_j^*}(\mathrm{d}t)$ → P(one of the already-there colors)

→ Generalization of the Pólya urn

The previous reminds the predictive distribution of a Dirichlet process.

Early studied by Blackwell and MacQueen.

Generalized Pólya urn



How the generalized Pólya urn works in practice. At each step we can sample an already observed color, or a new one from $\alpha(\cdot)$.

Generalized Pólya urn

Remarks:

- Then the joint law of $(X_1, \dots, X_n)$ can be decomposed as

$$P(X_1, \dots, X_n) = P(X_n \mid X_{n-1}, \dots, X_1) P(X_{n-1} \mid X_{n-2}, \dots, X_1) \dots P(X_1)$$
$$= \prod_{i=1}^{n} \left[\frac{a}{a+i-1} P_0(\cdot) + \frac{i-1}{a+i-1} \sum_{j=1}^{k_{[i]}} \frac{n_{j[i]}}{i-1} \delta_{X^*_{j[i]}} \right]$$

  with $k_{[i]}$ denoting the number of already observed distinct elements at the i-th sampling, $X^*_{1[i]}, \dots, X^*_{k_{[i]}[i]}$ the unique values, and $n_{1[i]}, \dots, n_{k_{[i]}[i]}$ their frequencies;

- The sampled sequence is exchangeable, i.e.

$$P(X_1, \dots, X_n) = P(X_{\sigma(1)}, \dots, X_{\sigma(n)})$$

  with $\sigma : \mathbb{N}_n \to \mathbb{N}_n$ a permutation of $1, \dots n$;

- By sampling from a generalized Pólya urn, we obtain a trajectory from a Dirichlet process;
- It can be view as a CRP where we assign to each table an unique dish chosen from P_0.

Stick-breaking

We want to represent $P = \sum_{j=1}^{\infty} W_j \delta_{X_j^*}$ in a useful way. First, we recall that $\{W_j\}_{j \geq 1}$ are independent of $\{X_j^*\}_{j>1}$.

We know that $\{X_j^*\}_{j \geq 1}$ are i.i.d from $P_0(\cdot)$.

We need to characterize the distribution of $\{W_j\}_{j \geq 1}$. Possible solution: stick-breaking construction: instead of characterizing the value of W_j, we focus on the distribution of the mass that we are stealing at the j-th step from the residual mass after $j-1$ steps.

Suppose we are stealing mass β at the first step on the atom X_1^*. From the posterior distribution of a DP we have

$$P \mid X_1^* \sim DP\left(a+1, \frac{aP_0(\cdot) + \delta_{X_1^*}(\cdot)}{a+1}\right)$$

We further know that defining a partition $(X_1^*, \mathbb{X} \setminus X_1^*)$ of $\mathbb{X}$ we have

$$(P(X_1^*), P(\mathbb{X} \setminus X_1^*)) \sim Dir\left((a+1)\frac{aP_0(X_1^*) + \delta_{X_1^*}(X_1^*)}{a+1}, (a+1)\frac{aP_0(\mathbb{X}) + \delta_{X_1^*}(\mathbb{X})}{a+1}\right)$$
$$\sim Dir(1, a)$$

4/12/2020
9/6

Stick-breaking

We then have that P has a point mass located in X_1^*, and can be rewritten as

$$G = V_1 \delta_{X_1^*} + (1 - V_1) P\,|_{\mathbb{X} \setminus X_1^*}$$

we decompose the process in 2 terms: one is a mass point in X_1^* with weight = V_1 and one is the process restricted to $\mathbb{X} \setminus X_1^*$.

Thanks to the self similarity, we know that $P\,|_{\mathbb{X} \setminus X_1^*}$ is still distributed as a DP with $P_0(\cdot)\,|_{\mathbb{X} \setminus X_1^*} \overset{a.s.}{=} P_0(\cdot)$, with mass a. Further, we know the marginal distribution for a dimension of a Dirichlet random variable, and we have that

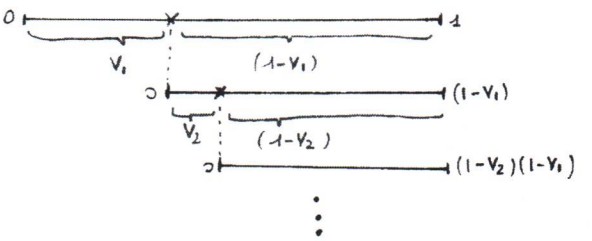
$$V_1 \sim Beta(1, a)$$

We then have

$$\begin{aligned}
&P \sim DP(a, P_0(\cdot)) \\
&P \sim V_1 \delta_{X_1^*} + (1 - V_1) P_1, \qquad \text{with } P_1 \sim DP(a, P_0(\cdot)) \\
&P \sim V_2 \delta_{X_1^*} + (1 - V_1)(V_2 \delta_{X_2^*} + (1 - V_2) P_2), \qquad \text{with } P_2 \sim DP(a, P_0(\cdot)) \\
&\vdots \\
&P = \sum_{j=1}^{\infty} W_j \delta_{X_j^*}
\end{aligned}$$

with $\{X_j^*\}_{j \geq 1}$ i.i.d from $P_0(\cdot)$, $W_j = V_j \prod_{\ell < j}(1 - V_\ell)$ and $V_j \sim Beta(1, a)$.

Stick-breaking



How the stick-breaking works in practice with the distribution of the weights. Each V_j is distributed according to a $Beta(1, a)$ distribution.

Stick-breaking

The stick-breaking construction is more general than the Dirichlet process case.

We need some regularity conditions:

$$\sum_{j=1}^{\infty} W_j = 1 \ a.s., \qquad IFF \qquad \sum_{j=1}^{\infty} \mathbb{E}[\log(1 - V_j)] = -\infty$$

We are dealing with a sequence $\{V_j\}_{j \geq 1} \overset{\text{iid}}{\sim} Beta(1, a)$.

It is sufficient that $P[V_1 > 0] > 0$.

The posterior distribution can also be represented in terms of stick-breaking construction, with updated weights.

$$\begin{aligned}
P \mid X_1, \ldots, X_n &= \sum_{j=1}^{\infty} W_j \delta_{X_j^*}(\cdot) \\
W_j &= V_j \prod_{\ell < j}(1 - V_\ell) \\
V_j &\overset{\text{ind}}{\sim} Beta(1 + n_j, a + m_j)
\end{aligned}$$

where $n_j = \sum_{i=1}^{n} \mathbb{I}_{[X_i = X_j^*]}$ and $m_j = n - \sum_{\ell=1}^{j} \sum_{i=1}^{n} \mathbb{I}_{[X_i = X_\ell^*]}$

Sampling from a Dirichlet process

we can sample a trajectory or a realization from the random prob. measure

Sampling from a DP: a trajectory

To sample a trajectory of size n from a Dirichlet process, we can exploit the Pólya urn sampling scheme. We start from X_1 where

$$P[X_1 \in \mathrm{d}t] = \frac{a}{a} P_0(\mathrm{d}t) \sim P_0(\mathrm{d}t)$$

then we have

$$P[X_2 \in \mathrm{d}t \mid X_1] = \frac{a}{a+1} P_0(\mathrm{d}t) + \frac{1}{a+1}\delta_{X_1}(\mathrm{d}t)$$

$$P[X_3 \in \mathrm{d}t \mid X_1, X_2] = \frac{a}{a+2} P_0(\mathrm{d}t) + \frac{2}{a+2}\sum_{j=1}^{k} \frac{n_j}{2}\delta_{X_j^*}$$

$$P[X_n \in \mathrm{d}t \mid X_1, \ldots, X_{n-1}] = \frac{a}{a+n-1} P_0(\mathrm{d}t) + \frac{n-1}{a+n-1}\sum_{j=1}^{k} \frac{n_j}{n-1}\delta_{X_j^*}$$

and the sequence produced $X_1, dots. X_n$ is an n-size sample from a Dirichlet process with mass parameter a and base measure P_0.

Sampling from a DP: updating a trajectory

Let $\boldsymbol{X}_{(i)} = \{X_1, \ldots, X_n\} \setminus X_i$. Let $\mathscr{L}(Y)$ denotes the distributional law of Y. Then we have

$$\begin{aligned}
\mathscr{L}(X_i \mid \boldsymbol{X}_{(i)}) &= \int_{\mathcal{P}_{\mathbb{X}}} \mathscr{L}(X_i, P \mid \boldsymbol{X}_{(i)}) \mathrm{d}P \\
&= \int_{\mathcal{P}_{\mathbb{X}}} \underbrace{\mathscr{L}(X_i \mid P, \boldsymbol{X}_{(i)})}_{X_i \mid P \sim P} \underbrace{\mathscr{L}(P \mid \boldsymbol{X}_{(i)})}_{P \mid \boldsymbol{X}_{(i)} \sim DP(\alpha(\cdot) + \sum_{1 \le j \ne i}^{n} n_j \delta_{X_j})} \mathrm{d}P \\
&= \int_{\mathcal{P}_{\mathbb{X}}} P \mathscr{L}(\mathrm{d}P \mid \boldsymbol{X}_{(i)}) = \mathbb{E}[P \mid \boldsymbol{X}_{(i)}] \\
&= \frac{a}{a+n-1} P_0(\cdot) + \frac{n-1}{a+n-1} \sum_{j=1}^{k_{(i)}} \frac{n_{j(i)}}{n-1} \delta_{X^*_{j(i)}}
\end{aligned}$$

(*) (**)

where $k_{(i)}$ are the number of distinct values in $\boldsymbol{X}_{(i)}$, $X^*_{1(i)}, \ldots, X^*_{k_{(i)}(i)}$ denote the distinct values out of $\boldsymbol{X}_{(i)}$, and $n_{j(i)}$ the frequency of the j-th distinct value.

we can update a trajectory updating element by element conditional to the rest of the sample and sampling for each element something that we already observed (*) or something new (**) with probabilities (*) and (**)

Sampling from a DP: a truncated realization

We might be interested into sampling a realization from a Dirichlet process.

WARNING: we have an infinite number of dimensions.

A first attempt: we can truncate the stick-breaking representation.

$$P(\mathrm{d}t) \simeq \tilde{P}(\mathrm{d}t) = \sum_{j=1}^{J} W_j \delta_{X_j^*}(\mathrm{d}t)$$

with

$$\begin{aligned}
W_j &= V_j \prod_{\ell<j} (1 - V_\ell), \qquad j = 1, \ldots, J-1 \\
V_j &\sim Beta(1, a), \qquad j = 1, \ldots, J-1 \\
V_J &= 1
\end{aligned}$$

Notice that:

- we introduce an approximation;
- we can control in some way the approximation;
- we can use a random truncation, for example $J \sim Poi(\lambda)$.

Sampling from a DP: a sliced realization

Instead of truncating truncate the stick-breaking construction, we can augment the problem and implement a slice sampling strategy, moving from $\mathscr{L}(P)$ to $\mathscr{L}(P, U)$, with $U \sim Unif(0, 1)$, such that

$$\mathscr{L}(P, U) = \sum_{j=1}^{\infty} \mathbb{I}_{[U \le W_j]} \delta_{X_j^*}$$

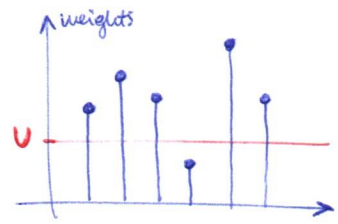
Notice that conditionally on a specific value for U, we can represent P with a finite number of component:

- Only the atoms with $W_j \ge U$ are included.
- Exists M such that $\sum_{j=M}^{\infty} W_j < U$, so that there are no weights after M which can be possibly larger than U.

Further

$$\int \mathscr{L}(P, U) \mathrm{d}u = \sum_{j=1}^{\infty} \int \mathbb{I}_{[u \le W_j]} \delta_{X_j^*} \mathbb{I}_{[0,1]}(u) du = \sum_{j=1}^{\infty} W_j \delta_{X_j^*}$$

If we marginalize wrt U, we obtain $\mathscr{L}(P)$.



Conditionally on the augmented variable $U \sim U[0,1]$ we select just a finite number of weights, all the W_j's s.t. $W_j > U$

```
# R & STAN are friends!
library(coda)
library(BNPmix)

# for plots
library(ggplot2)
library(tidyr)
library(dplyr)
```

```
library(purrr)
library(ggsci)
require(gplots)
```

```
require(ggpubr)
```

```
#----------------------------------------
# SAMPLING m TRAJECTORY FROM DP of Length n
# with Gaussian (0, 1) base measure
# using the predictive distribution

traj_sample <- function(m, n, alpha){
  result <- list()

  pb <- txtProgressBar(0, m, style = 3)
  for(j in 1:m){
    temp_seq <- c()

    for(i in 1:n){
      index <- sample(1:(length(temp_seq) + 1), 1, prob = c(temp_seq, alpha))
      if(index <= length(temp_seq)){
        temp_seq[index] <- temp_seq[index] + 1
      } else {
        temp_seq[index] <- 1
      }
    }
    temp_mat <- cbind(temp_seq, rnorm(length(temp_seq), 0, 1))
    colnames(temp_mat) <- c("frequency", "value")
    result[[j]] <- temp_mat
    setTxtProgressBar(pb, j)
  }
  close(pb)
  return(result)
}


#----------------------------------------
# DIFFERENT TRIALS
# A = (-2, 1)

a <- 1
sample1 <- traj_sample(100, 10000, a)
```

```
PA <- mean(
          unlist(
            lapply(sample1, function(x) sum(x[x[,2] < 1 && x[,2] > -2, 1]) / 10000)
          )
        )
PA
```

```
## [1] 0.82
```

```
PA * (1 - PA) / (a + 1)
```

```
## [1] 0.0738
```

```
PA_true <- pnorm(1) - pnorm(-2)
PA_true
```

```
## [1] 0.8185946
```

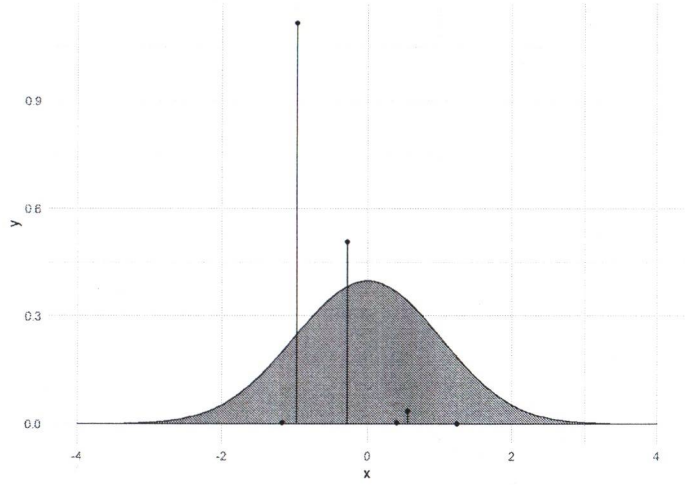
```
PA_true * (1 - PA_true) / (a + 1)
```

```
## [1] 0.07424874
```

```
sample1[[1]]
```

```
##      frequency      value
## [1,]      3041 -0.2760578
## [2,]      6691 -0.9641521
## [3,]       224  0.5612653
## [4,]        23  0.4112057
## [5,]        17 -1.1699911
## [6,]         4  1.2365513
```

```
data.frame(x = sample1[[1]][,2],
           y = sample1[[1]][,1]/6000) %>%
  ggplot() +
  geom_ribbon(data = data.frame(x = seq(-4, 4, by = 0.05),
                                ymin = rep(0, 161),
                                ymax = dnorm(seq(-4, 4, by = 0.05))),
              aes(x = x, ymin = ymin, ymax = ymax), fill = "orange", col = 1) +
  geom_point(aes(x = x, y = y)) +
  theme_minimal() +
  geom_segment(aes(x = x, y = rep(0, length(x)), xend = x, yend = y))
```




```
#--- increase the mass parameter a = 25
a <- 25
sample2 <- traj_sample(100, 10000, a)
```

```
PA <- mean(
          unlist(
            lapply(sample2, function(x) sum(x[x[,2] < 1 && x[,2] > -2, 1]) / 10000)
          )
        )

PA
```

```
## [1] 0.79
```

```
PA * (1 - PA) / (a + 1)
```

```
## [1] 0.006380769
```

```r
PA_true <- pnorm(1) - pnorm(-2)

PA_true
```

```
## [1] 0.8185946
```

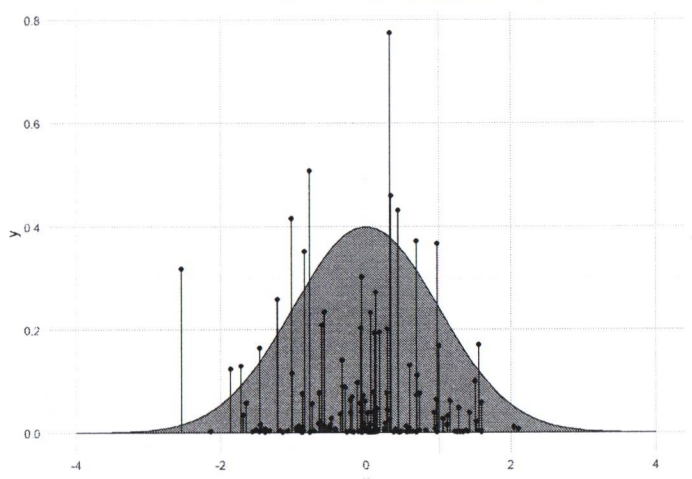
```r
PA_true * (1 - PA_true) / (a + 1)
```

```
## [1] 0.005711441
```

```r
sample2[[1]]
```

```
##        frequency        value
##   [1,]       459  0.341671668
##   [2,]       317 -2.540899383
##   [3,]        77 -0.638279770
##   [4,]        ..
```

```r
data.frame(x = sample2[[1]][,2], y = sample2[[1]][,1]/1000) %>%
  ggplot() +
  geom_ribbon(data = data.frame(x = seq(-4, 4, by = 0.05),
                                ymin = rep(0, 161),
                                ymax = dnorm(seq(-4, 4, by = 0.05))),
              aes(x = x, ymin = ymin, ymax = ymax), fill = "orange", col = 1) +
  geom_point(aes(x = x, y = y)) +
  theme_minimal() +
  geom_segment(aes(x = x, y = rep(0, length(x)), xend = x, yend = y))
```




```r
#--- increase the mass parameter a = 100
a <- 100
sample3 <- traj_sample(100, 10000, a)
```

```r
PA <- mean(
  unlist(
    lapply(sample3, function(x) sum(x[x[,2] < 1 && x[,2] > -2, 1]) / 10000)
  )
)

PA
```

```
## [1] 0.81
```

```r
PA * (1 - PA) / (a + 1)
```

```
## [1] 0.001523762
```

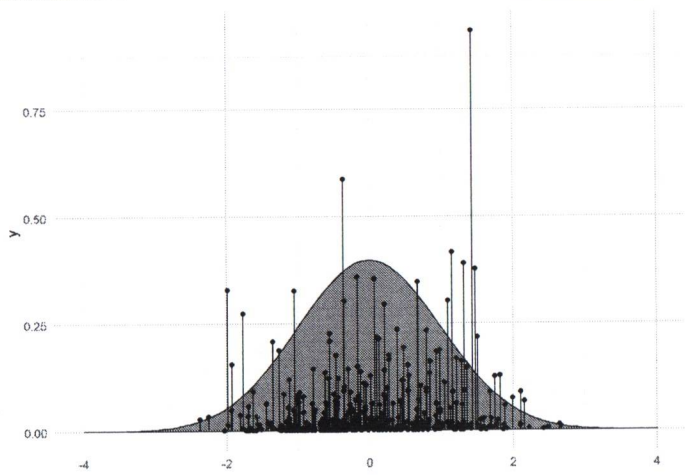
```r
PA_true <- pnorm(1) - pnorm(-2)

PA_true
```

```
## [1] 0.8185946
```

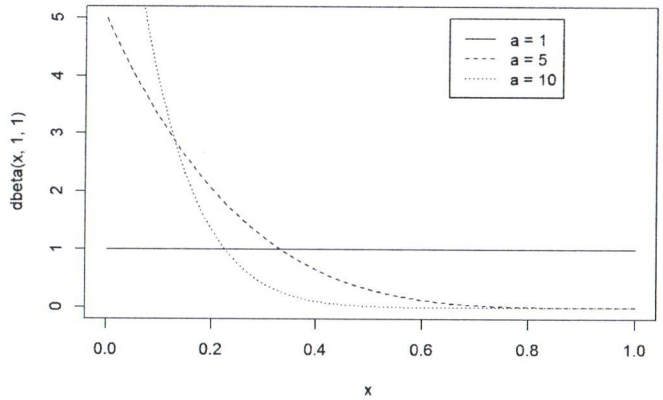
```r
PA_true * (1 - PA_true) / (a + 1)
```

```
## [1] 0.001470272
```

```r
data.frame(x = sample3[[1]][,2], y = sample3[[1]][,1]/500) %>%
  ggplot() +
  geom_ribbon(data = data.frame(x = seq(-4, 4, by = 0.05),
                                ymin = rep(0, 161),
                                ymax = dnorm(seq(-4, 4, by = 0.05))),
              aes(x = x, ymin = ymin, ymax = ymax), fill = "orange", col = 1) +
  geom_point(aes(x = x, y = y)) +
  theme_minimal() +
  geom_segment(aes(x = x, y = rep(0, length(x)), xend = x, yend = y))
```

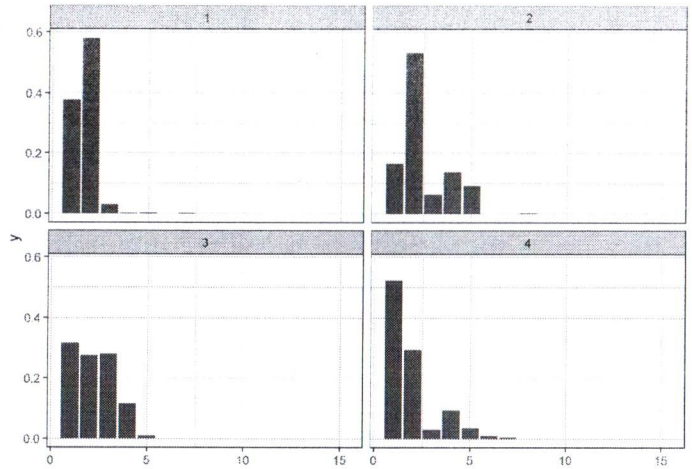



```r
#-----------------------------------------
#-----------------------------------------
#-----------------------------------------
# STICK BREAKING
#-----------------------------------------------------------
# About the weigths

curve(dbeta(x, 1, 1), ylim = c(0, 5))
curve(dbeta(x, 1, 5), add = T, lty = 2)
curve(dbeta(x, 1, 10), add = T, lty = 3)
legend(0.65, 5, lty = c(1,2,3), legend = c("a = 1", "a = 5", "a = 10"))
```



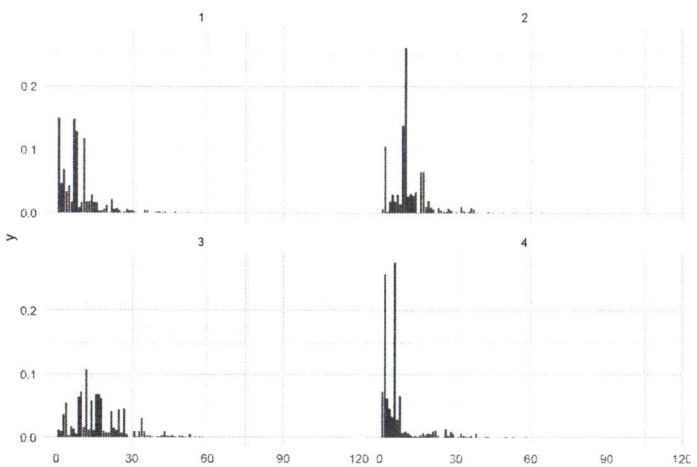

```
#---------
# SAMPLING
stick_eps <- function(eps, m, alpha){
  result <- list()

  pb <- txtProgressBar(0, m, style = 3)
  for(j in 1:m){
    temp_seq_V <- rbeta(1,1,alpha)
    temp_seq_W <- temp_seq_V[1]
    k <- 2
    while(sum(temp_seq_W) < 1 - eps){
      temp_seq_V[k] <- rbeta(1,1,alpha)
      temp_seq_W[k] <- prod(1 - temp_seq_V[1:(k - 1)]) * temp_seq_V[k]
      k <- k + 1
    }
    result[[j]] <- cbind(temp_seq_W, rnorm(length(temp_seq_W), 0, 1))
    setTxtProgressBar(pb, j)
  }
  close(pb)
  return(result)
}

# alpha = 1
sample_stick <- stick_eps(0.0001, 100, 1)
```

```
data.frame(y = c(sample_stick[[1]][,1], sample_stick[[2]][,1],
                 sample_stick[[3]][,1], sample_stick[[4]][,1]),
           group = as.factor(c(rep(1, length(sample_stick[[1]][,1])),
                               rep(2, length(sample_stick[[2]][,1])),
                               rep(3, length(sample_stick[[3]][,1])),
                               rep(4, length(sample_stick[[4]][,1])))),
           x = c(1:length(sample_stick[[1]][,1]),
                 1:length(sample_stick[[2]][,1]),
                 1:length(sample_stick[[3]][,1]),
                 1:length(sample_stick[[4]][,1]))) %>%
  ggplot() +
  geom_bar(aes(x = x, y = y), stat = "identity") +
  theme_bw() +
  facet_wrap(~group) + xlab("")
```



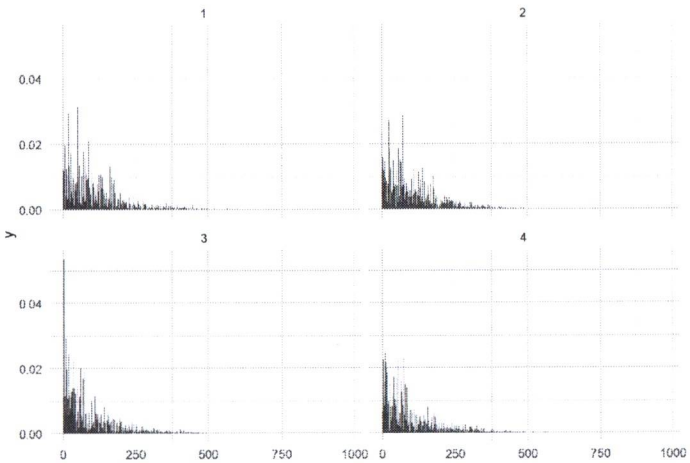

```
# alpha = 10
sample_stick <- stick_eps(0.0001, 100, 10)
```

```
data.frame(y = c(sample_stick[[1]][,1], sample_stick[[2]][,1],
                 sample_stick[[3]][,1], sample_stick[[4]][,1]),
           group = as.factor(c(rep(1, length(sample_stick[[1]][,1])),
                               rep(2, length(sample_stick[[2]][,1])),
                               rep(3, length(sample_stick[[3]][,1])),
                               rep(4, length(sample_stick[[4]][,1])))),
           x = c(1:length(sample_stick[[1]][,1]),
                 1:length(sample_stick[[2]][,1]),
                 1:length(sample_stick[[3]][,1]),
                 1:length(sample_stick[[4]][,1]))) %>%
  ggplot() +
  geom_bar(aes(x = x, y = y), stat = "identity") +
  theme_minimal() +
  facet_wrap(~group) + xlab("")
```




```
# alpha = 100
sample_stick <- stick_eps(0.0001, 100, 100)
```

```
data.frame(y = c(sample_stick[[1]][,1], sample_stick[[2]][,1],
                 sample_stick[[3]][,1], sample_stick[[4]][,1]),
           group = as.factor(c(rep(1, length(sample_stick[[1]][,1])),
                               rep(2, length(sample_stick[[2]][,1])),
                               rep(3, length(sample_stick[[3]][,1])),
                               rep(4, length(sample_stick[[4]][,1])))),
           x = c(1:length(sample_stick[[1]][,1]),
                 1:length(sample_stick[[2]][,1]),
                 1:length(sample_stick[[3]][,1]),
                 1:length(sample_stick[[4]][,1]))) %>%
  ggplot() +
  geom_bar(aes(x = x, y = y), stat = "identity") +
  theme_minimal() +
  facet_wrap(~group) + xlab("")
```

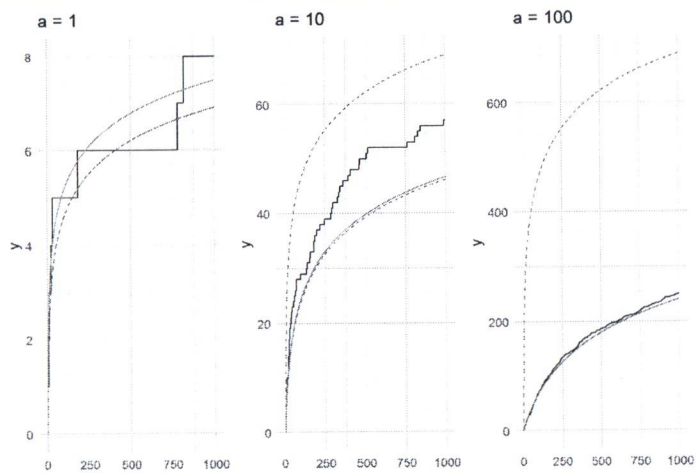



```
#------------------------------
# The number of clusters

traj_sample_unique <- function(n, alpha){
  result <- 1

  pb <- txtProgressBar(0, n, style = 3)
  for(i in 2:n){
    index <- sample(1:(length(unique(result)) + 1), 1, prob = c(table(result), alpha))
    result <- c(result, index)
    setTxtProgressBar(pb, i)
  }
  close(pb)
  return(result)
}

sample_one1 <- traj_sample_unique(10^3, 1)
```

```
sample_one2 <- traj_sample_unique(10^3, 10)
```

```
sample_one3 <- traj_sample_unique(10^3, 100)
```

```
ENOC <- function(alpha, n){
  sum(alpha / (alpha + 1:n - 1))
}

p1 <- data.frame(x = 1:10^3, y = sapply(1:10^3, function(x) length(unique(sample_one1[1:x]))),
                 z = sapply(1:10^3, function(x) ENOC(1, x))) %>%
  ggplot() +
  geom_line(aes(x = x, y = y)) +
  geom_line(aes(x = x, y = z), col = 3) +
  theme_minimal() + xlab("") +
  geom_line(data = data.frame(x = 1:10^3, y = 1 * log(1:10^3)),
            mapping = aes(x = x, y = y), col = 2, lty = 2) +
  geom_line(data = data.frame(x = 1:10^3, y = 1 * log((1 + 1:10^3) / 1)),
            mapping = aes(x = x, y = y), col = 4, lty = 2) +
  ggtitle("a = 1")

p2 <- data.frame(x = 1:10^3, y = sapply(1:10^3, function(x) length(unique(sample_one2[1:x]))),
                 z = sapply(1:10^3, function(x) ENOC(10, x))) %>%
  ggplot() +
  geom_line(aes(x = x, y = y)) +
  geom_line(aes(x = x, y = z), col = 3) +
  theme_minimal() + xlab("") +
  geom_line(data = data.frame(x = 1:10^3, y = 10 * log(1:10^3)),
            mapping = aes(x = x, y = y), col = 2, lty = 2)+
  geom_line(data = data.frame(x = 1:10^3, y = 10 * log((10 + 1:10^3) / 10)),
            mapping = aes(x = x, y = y), col = 4, lty = 2)+
  ggtitle("a = 10")

p3 <- data.frame(x = 1:10^3, y = sapply(1:10^3, function(x) length(unique(sample_one3[1:x]))),
                 z = sapply(1:10^3, function(x) ENOC(100, x))) %>%
  ggplot() +
  geom_line(aes(x = x, y = y)) +
  geom_line(aes(x = x, y = z), col = 3) +
  theme_minimal() + xlab("") +
  geom_line(data = data.frame(x = 1:10^3, y = 100 * log(1:10^3)),
            mapping = aes(x = x, y = y), col = 2, lty = 2)+
  geom_line(data = data.frame(x = 1:10^3, y = 100 * log((100 + 1:10^3) / 100)),
            mapping = aes(x = x, y = y), col = 4, lty = 2)+
  ggtitle("a = 100")

ggarrange(p1, p2, p3, ncol = 3)
```



10/12/2020
1/6

BNP2

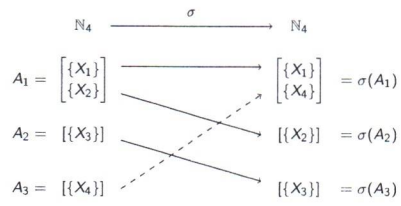
Riccardo Corradin
December 10, 2020

Recap : DIRICHLET PROCESS
combinatorial process and its realizations are sequences of atoms which possibile show ties. In general it's a probability measure with random jumps at random locations (we can have an infinite but countable jumps and locations)

Random partitions

Exchangeability

Let $\mathbb{X}$ be a Polish space, with $\mathscr{X}$ denoting its Borel σ-field. Let $\{X_1, \ldots, X_n\}$ be a finite sequence of $\mathbb{X}$-valued random elements defined on a common probability space $(\Omega, \mathscr{A}, \mathbb{P})$. We further assume the sequence exchangeable, in the sense that the distribution law of $\{X_1, \ldots, X_n\}$ is invariant w.r.t. any permutation of the elements $\sigma : \mathbb{N}_n \to \mathbb{N}_n$.



$$P(\{X_1, X_2\}, \{X_3\}, \{X_4\}) = P(\{X_1, X_4\}, \{X_2\}, \{X_3\})$$

Partitions

Let $\mathbb{N}_n = \{1, \ldots, n\}$ space of first n integer.

Definition

We say that $\psi_n = \{A_1, \ldots, A_k\}$ is a *partition* of $\mathbb{N}_n$ if the sets $\{A_1, \ldots, A_k\}$ are nonempty and such that $\mathbb{N}_n = \cup_i A_i$ and $A_i \cap A_j = \emptyset$ for all $i \neq j$.

Let $\mathcal{B}$ be the space of all partitions of $\mathbb{N}_n$ and $\mathscr{B}$ its discrete σ-field.

Definition

Let $(\Omega, \mathscr{F}, \mathrm{P})$ be a generic probability space. A random partition is any measurable function $\Psi_n : (\Omega, \mathscr{F}, \mathrm{P}) \to (\mathcal{B}, \mathscr{B})$.

The definition of Ψ_n induces a probability measure on the space of partitions $\mathcal{B}$.

Given a partition $\psi_n = \{A_1, \ldots, A_k\}$ of $\mathbb{N}_n$, the sequence of cardinalities of the blocks of ψ_n, that is $\mathbf{c}_n = (n_1, \ldots, n_k)$, with $n_i = |A_i|$, is named composition of n. A random partition Ψ_n induces a random composition $\mathcal{C}_n$. Note that the space of all compositions of n is the simplex

$$\triangle_n^k = \left\{ (n_1, \ldots, n_k) \in \mathbb{N}_n^k : \sum_{j=1}^{k} n_j = n, n_j > 0 \right\}.$$

Exchangeable partitions

Let σ be a permutation of $\mathbb{N}_n$ and $\sigma(A) = \{\sigma(i) : i \in A\}$.

Definition (Exchangeable partition)

We say that a random partition Ψ_n is exchangeable if, given any permutation $\sigma : \mathbb{N}_n \to \mathbb{N}_n$, the distribution of Ψ_n is invariant with respect to σ, i.e. for any partition $\{A_1, \ldots, A_k\}$ of $\mathbb{N}_n$ and for any permutation σ, the following holds

$$P[\Psi_n = \{A_1, \ldots, A_k\}] = P[\Psi_n = \{\sigma(A_1), \ldots \sigma(A_k)\}].$$

Equivalently, we say that a random partition Ψ_n is exchangeable if there exists a symmetric function $p_k^{(n)} : \mathcal{C}_n \to [0, 1]$ such that, for every partition $\{A_1, \ldots, A_k\}$ of $\mathbb{N}_n$

$$P[\Psi_n = \{A_1, \ldots, A_k\}] = p_k^{(n)}(|A_1|, \ldots, |A_k|) = p_k^{(n)}(n_1, \ldots, n_k),$$

The function $p_k^{(n)}$ is named exchangeable partition probability function (EPPF) of Ψ_n.

we can change the elements between the groups and also the labels, the only important thing is the cardinality of the groups (whatever the name of the group is, whatever the elements of the group are)

This function describes the probability of having a particular partition: if we have n elements the probability of dividing these elements in a specific partition depends only on the cardinalities

EPPFs

Remark: $p_k^{(n)}$ symmetric $\Rightarrow$ the cluster labels are exchangeable.

An EPPF satisfies the following properties:

probability of dividing one obs. in one group $\rightarrow$ 1

i) $p_1^{(1)} = 1$;

ii) for any $(n_1, \ldots, n_k) \in \triangle_n^k$, with $n \geq 1$ and $1 \leq k \leq n$,

$$p_k^{(n)}(n_1, \ldots n_k) = p_k^{(n)}(n_{\sigma(1)}, \ldots, n_{\sigma(k)}),$$

where $\sigma(\cdot)$ is a permutation of $(1, \ldots, k)$;

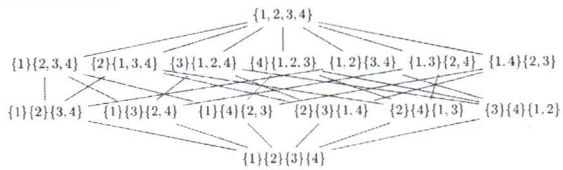
the probability is invariant w.r.t. permutations of the blocks

iii) for any $(n_1, \ldots, n_k) \in \triangle_n^k$, with $n > 1$ and $1 \leq k \leq n$, the following addition rule holds

$$p_k^{(n)}(n_1, \ldots, n_k) = \sum_{j=1}^{k} p_k^{(n+1)}(n_1, \ldots, n_j + 1, \ldots, n_k) + p_{k+1}^{(n+1)}(n_1, \ldots, n_k, 1). \quad (1)$$

EPPFs

Problem: the space of the possible partition of n elements is large, and is growing fast as far as n grows.



We have

$$S_{n,k} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (k - j)^n$$

(Stirling number of the second kind) ways to partition n elements in k groups, and

$$B_n = \sum_{k=1}^{n} S_{n,k}$$

(Bell number) ways to partition n elements. B_n is exploding as far as n grows.

Connection with the Dirichlet process

10/12/2020
2/6

DP - EPPF

Let P be a random probability measure distributed as a Dirichlet process with total mass $a > 0$ and base measure $P_0(\cdot)$, and let $X_1, \ldots, X_n$ be an exchangeable sample from P, i.e.

$$X_1, \ldots, X_n \mid P \overset{\text{iid}}{\sim} P$$
$$P \sim DP(a, P_0(\cdot))$$

Due to the a.s. discreteness of P, the sample can possibly show ties. We denote by $X_1^*, \ldots, X_k^*$ the unique values in $X_1, \ldots, X_n$, with frequencies $n_1, \ldots, n_k$, for $1 \leq k \leq n$.

TIES IN THE SEQUENCE $\Leftrightarrow$ LATENT PARTITION

The expectation of the distribution of a sequence $(X_1, \ldots, X_n)$ sampled from a Dirichlet process

$$\mathrm{E}[\mathscr{L}(X_1, \ldots, X_n)] = \mathrm{E}\left[\prod_{i=1}^{n} P(X_i)\right] = \mathrm{E}\left[\prod_{j=1}^{k} P^{n_j}(X_j^*)\right]$$
$$= \underbrace{p_k^{(n)}(n_1, \ldots, n_k)}_{\text{distribution of the partition}} \underbrace{\prod_{j=1}^{k} P_0(X_j^*)}_{\text{distribution of the atoms}}$$

7/26

DP - EPPF

In particular, for the Dirichlet process we have that the EPPF is equal to

$$p_k^{(n)}(n_1, \ldots, n_k) = \frac{a^k}{(a)_n} \prod_{j=1}^{k} (n_j - 1)!$$

$(a)_n = \frac{\Gamma(a+n)}{\Gamma(a)}$

Does it satisfy the previous properties?

i) $p_1^{(1)} = \frac{a^1}{(a)_1}(1-1)! = \frac{a}{a} = 1$

ii) $p_k^{(n)}(n_1, \ldots, n_k) = \frac{a^k}{(a)_n} \prod_{j=1}^{k} (n_j - 1)! = \frac{a^k}{(a)_n} \prod_{j=1}^{k} (n_{\sigma(j)} - 1)! = p_k^{(n)}(n_{\sigma(1)}, \ldots, n_{\sigma(k)})$

iii) Recall that $(a)_b = \frac{\Gamma(a+b)}{\Gamma(a)}$, then we have

$$\sum_{\ell=1}^{k} p_k^{(n+1)}(n_1, \ldots, n_\ell + 1, \ldots, n_k) + p_{k+1}^{(n+1)}(n_1, \ldots, n_k, 1)$$
$$= \sum_{\ell=1}^{k} \frac{a^k}{(a)_{n+1}} \prod_{j=1}^{k} (n_j - 1)! \frac{(n_\ell)!}{(n_\ell - 1)!} + \frac{a^{k+1}}{(a)_{n+1}} \prod_{j=1}^{k} (n_j - 1)!$$
$$= \left(\frac{a^k \Gamma(a)}{\Gamma(a+n)} \prod_{j=1}^{k} (n_j - 1)!\right) \left[\frac{\Gamma(a+n)}{\Gamma(a+n+1)} \left(a + \sum_{\ell=1}^{k} n_j\right)\right] = p_k^{(n)}(n_1, \ldots, n_k)$$

8/26

DP - EPPF

Notice that we can derive the allocation probabilities of the Chinese restaurant process starting from the EPPF. Suppose we have allocated $X_1, \ldots, X_n$ into k distinct groups, with frequencies $n_1, \ldots, n_k$.

The $n+1$ elements can be allocated to the j-th group with probability

$$\frac{p_k^{(n+1)}(n_1, \ldots, n_j + 1, \ldots, n_k)}{p_k^{(n)}(n_1, \ldots, n_k)} = \frac{\frac{a^k}{(a)_{n+1}} n_j \prod_{j=1}^{k} (n_j - 1)!}{\frac{a^k}{(a)_n} \prod_{j=1}^{k} (n_j - 1)!} = \frac{n_j}{a+n},$$

and to a new group with probability

$$\frac{p_{k+1}^{(n+1)}(n_1, \ldots, n_k, 1)}{p_k^{(n)}(n_1, \ldots, n_k)} = \frac{\frac{a^{k+1}}{(a)_{n+1}} \prod_{j=1}^{k} (n_j - 1)!}{\frac{a^k}{(a)_n} \prod_{j=1}^{k} (n_j - 1)!} = \frac{a}{a+n}.$$

The previous probability are equivalent to the allocation process described by a CRP.

9/26

Dirichlet process mixtures

Idea of mixtures of distributions:
if one model is not flexible enough
→ use more than one model

Introduction

Let $\mathbb{Y}$, Θ be Polish spaces, and $\mathscr{Y}$, $\mathscr{T}$ their Borel σ-field. Let $\mathcal{K}(\cdot;\cdot)$ be a (regular enough) kernel function, such that $\mathcal{K}: \times\Theta \to [0,1]$, $\mathcal{K}(\cdot;\theta)$ is a density function for all $\theta \in \Theta$, and $\mathcal{K}(A;\cdot)$ is measurable for all $A \subseteq Y$. We can define a mixture model in a general fashion as

$$f(y) = \int_\Theta \mathcal{K}(y;\theta) P(\mathrm{d}\theta)$$

Examples:

- $P = \sum_{j=1}^k p_j \delta_{x_j}(\cdot)$, finite number of elements with known and fixed weights p_j's and atoms x_j's;
- $P = \sum_{j=1}^k W_j \delta_{X_j^*}(\cdot)$, finite number of elements with unknown random weights W_j's and atoms X_j^*'s;
- $P = \sum_{j=1}^\infty W_j \delta_{X_j^*}(\cdot)$, infinite number of elements with unknown random weights W_j's and atoms X_j^*'s;

DP mixture

We set $P \sim DP(a, P_0(\cdot))$, with $a > 0$, and $P(\cdot) = \sum_{j=1}^\infty W_j \delta_{\theta_j^*}(\cdot)$. Then we obtain an infinite mixture model

$$\tilde{f}(y) = \int_\Theta \mathcal{K}(y;\theta) P(\mathrm{d}\theta) = \sum_{j=1}^\infty W_j \mathcal{K}(y, \theta_j^*)$$

This model is defining a random density (which is a distribution over the densities space) It's not a fixed model, it's a random density! Its realizations are densities themselves! The source of randomness is given by the weights and atoms.

where the last holds thanks to the a.s. discreteness of the Dirichlet process, further we know that $\{\theta_j^*\}_{j\geq 1} \overset{\text{iid}}{\sim} P_0(\cdot)$ and $\{W_j\}_{j\geq 1}$ is a sequence characterized by a stick-breaking representation $W_j = V_j \prod_{\ell<j}(1 - V_\ell)$, and $V_j \sim Beta(1, a)$ for all $j \geq 1$.

Suppose we collected a sample $Y_1, \ldots, Y_n$ distributed as the previous mixture. Then we have, in hierarchical form:

$$\begin{aligned} Y_i \mid \theta_i &\overset{\text{ind}}{\sim} \mathcal{K}(Y_i;\theta_i) \\ \theta_1, \ldots, \theta_n \mid P &\overset{\text{iid}}{\sim} P \\ P &\sim DP(a; P_0(\cdot)) \end{aligned}$$

Each Y_i is associated to a latent parameter θ_i. The sequence $(\theta_1, \ldots, \theta_n)$ can possibly show ties, and its distribution is governed by a Dirichlet process.

DP mixture

Choosing $\mathcal{K}(\cdot,\cdot)$ equal to a continuous density, a DP mixture is a random density, in the sense that its describing a random variable taking values on $\mathbb{S}_{\tilde{f}} \subseteq \mathbb{F}_{\mathbb{Y}}$, where $\mathbb{F}_{\mathbb{Y}}$ denotes the space of densities with support $\mathbb{Y}$.

Different choices for $\mathcal{K}(\cdot;\cdot)$ lead to different models. Some common choices:

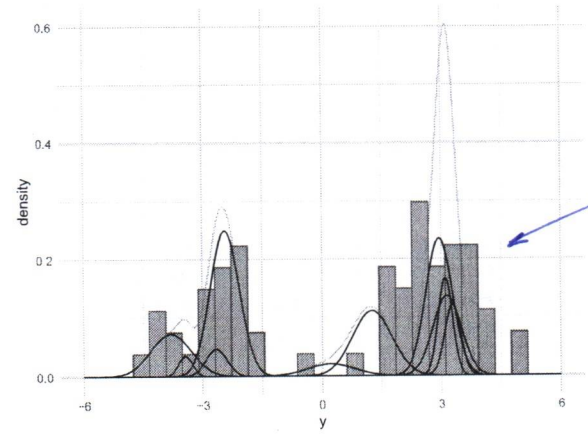
- $\mathcal{K}(y;\theta) \sim N(\mu, \sigma^2)$, with $\theta = \mu$, location Gaussian mixture model.
- $\mathcal{K}(y;\theta) \sim N(\mu, \sigma^2)$, with $\theta = (\mu, \sigma^2)$, location-scale Gaussian mixture model.
- $\mathcal{K}(y;\theta) \sim N(\boldsymbol{\mu}, \Sigma)$, with $\theta = (\boldsymbol{\mu}, \Sigma)$, multivariate location-scale Gaussian mixture model.

A mixture of Gaussian distribution is a really flexible model – dense in a space of regular densities (Lo 1984).

You can define a mixture model for several kind of data:

- Functional data – ex. $K(y,\theta) \sim GP(\mu(t), R_\theta(t,t'))$ with $\theta = (\mu(t), R_\theta(t,t'))$
- Graph data – ex. $K(y,\theta) \sim ERGM(\tau)$ with $\theta = \tau$ (Exponential Random Graph Model).
- ...

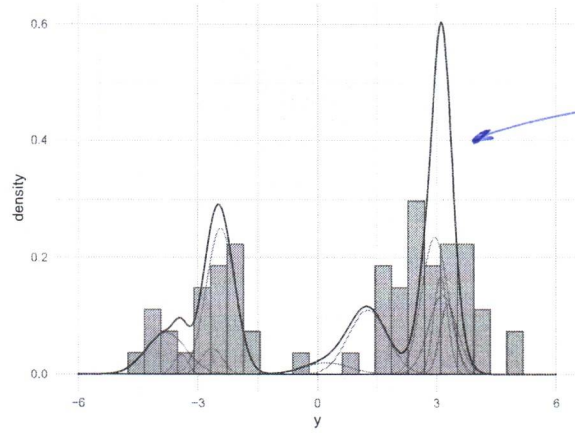
Example of location-scale Gaussian mixture



The black lines are different components in the mixture. Each observation is associated to a specific component. We have a finite number of different components with random weights, random location and random variances.

10/12/2020
3/6

Example of location-scale Gaussian mixture

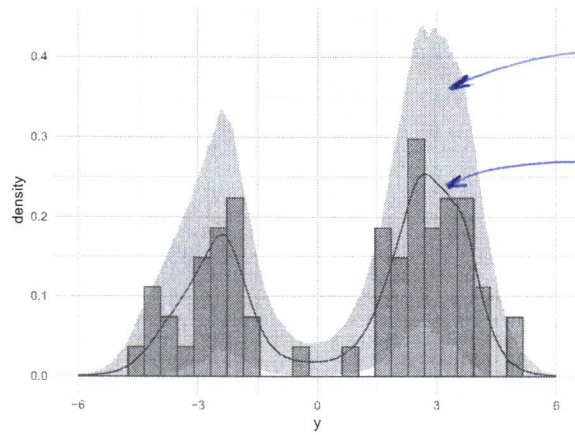


14/26

single realization from the random density that we induce with this mixture model (smoothed version of the previous components: we're taking the average of the distrib. weighted by their random weights)

⟹ A realization from the mixture model is a single mixture (which is a particular density)

Example of location-scale Gaussian mixture



15/26

credible bands (0.9)

posterior mean

See the GIF!

Latent partition in the mixtures

Let $Y_1, \ldots, Y_n$ be a set of random elements taking values on $\mathbb{Y}$. We can augment the problem by introducing a set of latent random variables $S_1, \ldots, S_n$, where the generic $S_i \in \mathbb{N}$ describe the allocation of the i-th observation to a specific group, i.e. $S_i = j$ if the i-th observation belongs to the j-th component of the mixture, equivalently

$$S_i = j \quad \text{if} \quad \theta_i = \theta_j^*$$

We are implicitly inducing equivalence classes in our data, such that

$$Y_i \sim Y_\ell \quad \text{if} \quad S_i = S_\ell$$

or

$$Y_i \sim Y_\ell \quad \text{if} \quad \theta_i = \theta_\ell$$

The group allocations $S_1, \ldots, S_n$ are unknown and object of interest.

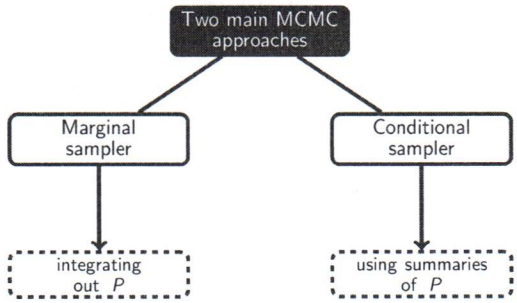
We are implicitly assuming a latent random partition of the data $\Psi_n = \{A_1, \ldots, A_k\}$ with $A_j = \{i : S_i = j\}$.

- Model based clustering → we assign the observations to the best model. ← standard procedure
- Product partition model: $\Psi_n = \{A_1, \ldots, A_k\}$ with $A_j = \{i : S_i = j\}$ ← our procedure: we're (sort of) considering ∞ clusters: when our clusters are not sufficient to describe the data we're taking a new cluster

$$P(\Psi_n \mid \boldsymbol{Y}, \boldsymbol{\theta}) \propto \pi(\Psi_n) P(\boldsymbol{\theta} \mid \Psi_n) L(\boldsymbol{Y} \mid \Psi, \theta)$$

16/26

Sampling the mixtures



Sampling the mixtures

We are interested into perform inference on the latent partition and the random density underlying the data.

Our target distribution is to sample from:

$$\begin{aligned}\mathscr{L}(\boldsymbol{Y}, \boldsymbol{\theta}, P) &= \mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{\theta}, P)\mathscr{L}(\boldsymbol{\theta} \mid P)\mathscr{L}(P) \\ &= \mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{\theta})\mathscr{L}(\boldsymbol{\theta} \mid P)\mathscr{L}(P)\end{aligned}$$

Two possible strategies:

1. Marginal sampler, we resort to

$$\mathrm{E}_P[\mathscr{L}(\boldsymbol{Y}, \boldsymbol{\theta})] = \mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{\theta})\mathrm{E}_P[\mathscr{L}(\boldsymbol{\theta} \mid P)]$$

   i.e. we marginalize out the distribution of P.
2. Conditional samplers, we update $P \sim \mathscr{L}(P)$ and then, conditionally on the current value of P we sample from

$$\mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{\theta})\mathscr{L}(\boldsymbol{\theta} \mid P)$$

Different methods – different estimation of the uncertainty associated to the problem.

Marginal sampler

We are interested into update the cluster allocation of a single observation, conditionally on the rest of the sample. Let $\boldsymbol{Y}_{(i)} = \{Y_1, \ldots, Y_n\} \setminus Y_i$ and $\boldsymbol{\theta}_{(i)} = \{\theta_1, \ldots, \theta_n\} \setminus \theta_i$. Let $\theta_{1(i)}, \ldots, \theta_{k_{(i)}(i)}$ be the $k_{(i)}$ distinct values in $\boldsymbol{\theta}_{(i)}$, with frequencies $n_{1(i)}, \ldots, n_{k_{(i)}(i)}$ Then we have

$$\begin{aligned}\mathscr{L}(Y_i, \theta_i \mid \boldsymbol{Y}_{(i)}, \boldsymbol{\theta}_{(i)}) &= \int_{\mathbb{P}_\Theta} \mathscr{L}(Y_i, \theta_i, P \mid \boldsymbol{Y}_{(i)}, \boldsymbol{\theta}_{(i)})\mathrm{d}P \\ &= \int_{\mathbb{P}_\Theta} \underbrace{\mathscr{L}(Y_i, \theta_i, \mid P)}_{Y_i \mid \theta_i \overset{\text{ind}}{\sim} \mathcal{K}(Y_i; \theta_i),\ \theta_i \mid P \sim P} \underbrace{\mathscr{L}(P \mid \boldsymbol{\theta}_{(i)})}_{P \mid \boldsymbol{\theta}_{(i)} \sim DP(\alpha(\cdot) + \sum_{j=1, j\neq i}^{n} \delta_{\theta_j}(\cdot))} \mathrm{d}P \\ &= \int_{\mathbb{P}_\Theta} \int_\Theta \mathcal{K}(Y_i; \theta_i) P(\mathrm{d}\theta_i)\mathscr{L}(P \mid \boldsymbol{\theta}_{(i)})\mathrm{d}P \\ &= \int_\Theta \mathcal{K}(Y_i; \theta_i) \int_{\mathbb{P}_\Theta} P(\mathrm{d}\theta_i)\mathscr{L}(\mathrm{d}P \mid \boldsymbol{\theta}_{(i)}) \\ &= \int_\Theta \mathcal{K}(Y_i; \theta_i)\mathrm{E}[P(\mathrm{d}\theta_i) \mid \boldsymbol{\theta}_{(i)}] \\ &= \frac{a}{a+n}\int_\Theta \mathcal{K}(Y_i; \theta_i)P_0(\theta_i) + \frac{n}{a+n}\sum_{j=1}^{k_{(i)}} \frac{n_{j(i)}}{a+n}\mathcal{K}(Y_i; \theta^*_{j(i)})\end{aligned}$$

probability of the i-th observation of being assigned to a new group

Marginal sampler

The marginal sampler is based on

$$\mathscr{L}(Y_i, \theta_i \mid \boldsymbol{Y}_{(i)}, \boldsymbol{\theta}_{(i)}) = \frac{a}{a+n}\int_\Theta \mathcal{K}(Y_i; \theta_i)P_0(\theta_i) + \frac{n}{a+n}\sum_{j=1}^{k_{(i)}} \frac{n_{j(i)}}{a+n}\mathcal{K}(Y_i; \theta^*_{j(i)})$$

probability of the i-th observation of being assigned to an already existing group

Remarks:

- Choose $P_0(\cdot)$ in a smart way! If $P_0(\cdot)$ is conjugate to $\mathcal{K}(\cdot;\cdot)$, then the integral $\int_\Theta \mathcal{K}(Y_i; \theta_i)P_0(\theta_i)$ is the prior predictive distribution, often available in closed form. Otherwise there are other strategies, for example you can evaluate it via Monte Carlo integration.
- When you re-allocate the i-th observation, and it is a singleton, its cluster must be destroyed (this is the reason for the nefarious notation in the previous slide).
- The distribution, which correspond to the predictive distribution of a DPM, is still a convex combination between the prior part and an empirical part, weighted by the kernel.
- We can trace the group allocation using the variables $S_1, \ldots, S_n$ with

$$\begin{aligned}P(S_i = j \mid \textit{rest}) &\propto n_j K(Y_i; \theta - j(i)) \\ P(S_i = k_{(i)} + 1 \mid \textit{rest}) &\propto a\int_\Theta K(Y_i; \theta)P_0(\mathrm{d}\theta)\end{aligned}$$

10/12/2020
9/6

Marginal sampler

Algorithm 1: Pseudocode for the marginal sampler

set *admissible initial values* $\boldsymbol{\theta}^{(0)}$;
for *each iteration* $r = 1, \ldots, R$ **do**
  for *each* $i = 1, \ldots, n$ **do**
    let $k_{(i)}$ *be the number of distinct values* $\theta^{*(r)}_{j(i)}$*'s in* $\boldsymbol{\theta}^{(r)}_{(i)} = (\theta^{(r)}_1, \ldots, \theta^{(r)}_{i-1}, \theta^{(r)}_{i+1}, \ldots \theta^{(r)}_n)$ *and* $n_{j(i)}$*, for* $j = 1, \ldots k_{(i)}$*, the corresponding frequencies;*
    sample $S^{(r)}_i$ *from*

$$\mathrm{P}(S^{(r)}_i = t \mid \ ) \propto \begin{cases} n_j \mathcal{K}(X_i; \theta^{*(r)}_{j(i)}) & \text{if } j \in \{1, \ldots, k_{(i)}\} \\ a \int \mathcal{K}(X_i, \theta) P_0(\mathrm{d}\theta) & \text{if } j = k_{(i)} + 1 \end{cases}$$

  for *each unique value* $\theta^{*(r)}_j$ *in* $\boldsymbol{\theta}^{(r)}$ **do**
    update $\theta^{*(r)}_j$ *from* $\mathrm{P}(\theta^{*(r)}_j \in \mathrm{d}t \mid -) \propto P_0(\mathrm{d}t) \prod_{i:S_i=j} \mathcal{K}(X_i; t)$
end

21/26

Another approach:

Truncated stick-breaking sampler

We are interested in both update the cluster allocation all single observation, and estimate the random density. Then in a Gibbs style algorithm, we have

$$\begin{aligned} \mathscr{L}(\boldsymbol{Y}, \boldsymbol{S}, P) &= \mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{S}, P) \mathscr{L}(\boldsymbol{S} \mid P) \mathscr{L}(P) \\ &= \mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{S}) \mathscr{L}(\boldsymbol{S}) \mathscr{L}(P) \end{aligned}$$

where

$$\mathscr{L}(\boldsymbol{Y} \mid \boldsymbol{S}, P) = \prod_{i=1}^{n} \mathcal{K}(Y_i; \theta_i) = \prod_{j=1}^{k} \prod_{i:S_i=j} \mathcal{K}(Y_i; \theta^*_j),$$

$$\mathscr{L}(\boldsymbol{S}) \propto \prod_{i=1}^{n} \sum_{j=1}^{M} \mathbf{1}_{[S_i=j]}$$

and P is distributed as a truncated DP

$$P = \sum_{j=1}^{M} W_j \delta_{\theta^*_j}(\cdot)$$

with $W_j = V_j \prod_{\ell<j}(1 - V_\ell)$, $V_j \sim Beta(1, a)$ if $j < M$, $V_M = 1$ and $\{\theta^*_1, \ldots, \theta^*_M\} \overset{\text{iid}}{\sim} P_0(\cdot)$.

22/26

Truncated stick-breaking sampler

Conditionally on $\boldsymbol{Y}, \boldsymbol{\theta}, P$ we can reallocate the observations, where

$$P(S_i = j \mid -) \propto W_j \mathcal{K}(Y_i; \theta^*_j),$$

we can then update P as

$$\mathscr{L}(P \mid \boldsymbol{Y}, \boldsymbol{S}) = \sum_{j=1}^{M} W_j \delta_{\theta^*_j}(\cdot)$$

with

$$W_j = V_j \prod_{\ell<j}(1 - V_\ell),$$

and

$$V_j \sim Beta(1 + n_j, a + m_j) \text{ if } j < M, \quad V_M = 1$$

where $n_j = \sum_{i=1}^{n} \mathbf{1}_{[S_i=j]}$ and $m_j = \sum_{i=1}^{n} \mathbf{1}_{[S_i>j]}$. We can update the θ^*_j's from

$$\mathscr{L}(\theta^*_j \mid \boldsymbol{Y}, \boldsymbol{S}) \propto P_0(\mathrm{d}\theta^*_j) \prod_{i:S_i=J} \mathcal{K}(Y_i; \theta^*_j)$$

23/26

Truncated stick-breaking sampler

Algorithm 2: Pseudocode for the truncated stick-breaking sampler

set *admissible initial values* $\boldsymbol{S}^{(0)}, P_M$;
for *each iteration* $r = 1, \ldots, R$ **do**
  for *each* $i = 1, \ldots, n$ **do**
    sample $S^{(r)}_i$ *from*

$$\mathrm{P}(S^{(r)}_i = j \mid -) \propto W^{(r)}_j \mathcal{K}(X_i; \theta^{*(r)}_{j(i)}), \quad j = 1, \ldots, M$$

  for $j = 1, \ldots, M$ **do**
    update V_j *from* $\mathscr{L}(V^{(r)}_j \mid -) \sim Beta(1 + n_j, a + m_j)$ if $j < M$, $V_M = 1$
    with $n_j = \sum_{i=1}^{n} \mathbf{1}_{[S_i=j]}$ and $m_j = \sum_{i=1}^{n} \mathbf{1}_{[S_i>j]}$
    set $W^{(r)}_j = V^{(r)}_j \prod_{\ell<j}(1 - V^{(r)}_j)$ *from*
    update $\theta^{*(r)}_j$ *from*

$$\mathscr{L}(\theta^{*(r)}_j \mid \boldsymbol{Y}, \boldsymbol{S}) \propto P_0(\mathrm{d}\theta^{*(r)}_j) \prod_{i:S_i=J} \mathcal{K}(Y_i; \theta^{*(r)}_j)$$

end

24/26

To synthesize

Different sampling strategies to deal with P

- **Marginal sampler**: we integrate out the distribution of P, resorting to the distribution of $\theta_i \mid \theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_n$.
- **Truncated stick-breaking sampler**: we work with finite summaries of P.

Note that:

- With the marginal sampler we focus on $\mathscr{L}(\theta_1, \ldots, \theta_n) = \mathrm{E}[\mathscr{L}(\theta_1, \ldots, \theta_n \mid P)]$, the update must be done sequentially.
- With the conditional samplers we focus on $\mathscr{L}(\theta_1, \ldots, \theta_n \mid P)$, the update can be done in parallel.

Both the algorithms produce a realization for the latent partition of the data at each iteration $\rightarrow$ we need a point estimate.

Example

Mixture of Gaussian distribution.

We consider $\mathcal{K}(Y; \theta) \sim N(\mu, \sigma^2)$, with $\theta = \mu$, i.e. location mixtures.

We choose as base measure a Gaussian distribution, $P_0(\mu) \sim N(m_0, \sigma_0^2)$, to preserve conjugacy, i.e.

$$\int \mathcal{K}(y; \theta) P_0(\mathrm{d}\theta) \sim N(m_0, \sigma_0^2 + \sigma^2)$$

We consider a model

$$\tilde{f}(y) = \sum_{j=1}^{\infty} W_j \mathcal{K}(y; \theta_j^*)$$

We compare the marginal sampling strategy and the truncated stick-breaking sampling strategy.

See the code!

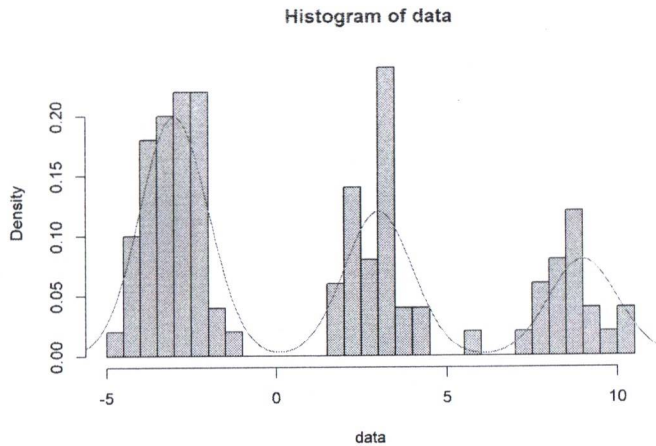
```r
# R & STAN are friends!
library(coda)
library(BNPmix)

# for plots
library(ggplot2)
library(tidyr)
library(dplyr)
```

```r
library(purrr)
library(ggsci)
require(gplots)
```

```r
require(ggpubr)
```

```r
# MARGINAL SAMPLER
MS_univ <- function(data, niter, burnin, m0, sigma2_0, sigma2, mass){

  n <- length(data)

  res_part <- matrix(0, ncol = n, nrow = niter - burnin)
  mu <- m0

  # initialize the partition - all in ome cluster
  S <- rep(1, n)

  mu_out <- list()
  probs <- c()

  pb <- txtProgressBar(0, niter, style = 3)
  # for all the iterations
  for(iter in 1:niter){

    #---------------------------------
    # allocation step
    #---------------------------------

    # for all the observations
    for(i in 1:n){

      # J = number of current clusters
      J <- length(mu)
      probs <- rep(0, J + 1)

      # for all the active clusters
      for(j in 1:J){
        probs[j] <- (sum(S[-i] == j) / (n + mass - 1)) * dnorm(data[i], mean = mu[j], sd = sqrt(sigma2))
      }

      # probability of new cluster
      probs[J + 1] <- (mass / (n + mass - 1)) * dnorm(data[i], mean = m0, sd = sqrt(sigma2 + sigma2_0))

      # sample from 1:J+1 with probability probs
      S[i] <- sample(1:(J + 1), size = 1, replace = F, prob = probs)

      # if the cluster is new, generate its mean value
      if(S[i] > J){
        mu[S[i]] <- rnorm(1, mean = (data[i] / sigma2 + m0 / sigma2_0) * (1 / sigma2 + 1 / sigma2_0),
                          sd = sqrt(1 / (1 / sigma2 + 1 / sigma2_0)))
      }
    }

    # resize the objects
    mu <- mu[unique(sort(S))]
    S <- as.numeric(as.factor(S))

    if(iter > burnin){
      res_part[iter - burnin, ] <- S
      mu_out[[iter - burnin]] <- mu
    }

    #---------------------------------
    # acceleration step
    #---------------------------------

    for(j in 1:length(mu)){
      ntemp <- length(data[S == j])
      mu[j] <- rnorm(1, mean = (sum(data[S == j]) / sigma2 + m0 / sigma2_0) * (length(data[S == j]) / sig
ma2 + 1 / sigma2_0),
                     sd = sqrt(length(data[S == j]) / (1 / sigma2 + 1 / sigma2_0)))
    }
    setTxtProgressBar(pb, iter)
  }
  close(pb)
  return(list(res_part, mu_out))
}

# TRUNCATED SB SAMPLER
TRSB_univ <- function(data, niter, burnin, m0, sigma2_0, sigma2, mass, M){

  n <- length(data)
```

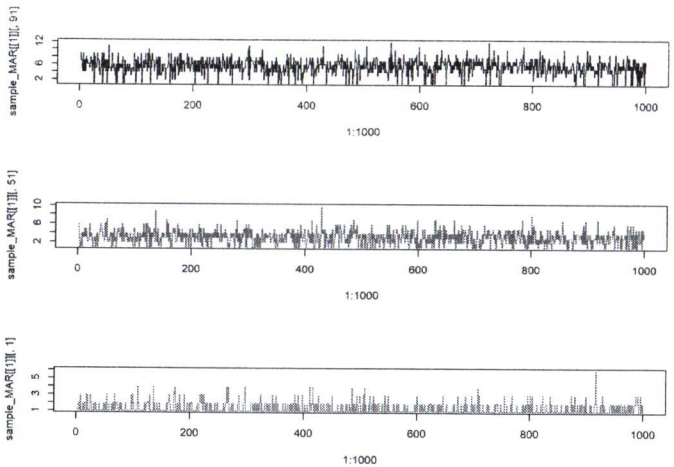
```
  res_part <- matrix(0, ncol = n, nrow = niter - burnin)
  mu <- m0
  W <- V <- c()

  # initialize the partition - all in ome cluster
  S <- rep(1, n)

  mu_out <- list()
  probs_out <- list()
  probs <- c()

  pb <- txtProgressBar(0, niter, style = 3)
  # for all the iterations
  for(iter in 1:niter){
    #---------------------------------
    # UPDATE THE WEIGHTS and THE ATOMS
    #---------------------------------

    for(j in 1:M){
      ntemp <- length(data[S == j])
      mu[j] <- rnorm(1, mean = (sum(data[S == j]) / sigma2 + m0 / sigma2_0) * (length(data[S == j]) / sig
ma2 + 1 / sigma2_0),
                     sd = sqrt(1 / (length(data[S == j]) / sigma2 + 1 / sigma2_0)))
    }

    V[1] <- rbeta(1, 1 + sum(S == 1), mass + sum(S > 1))
    W[1] <- V[1]

    for(l in 2:(M - 1)){
      V[l] <- rbeta(1, 1 + sum(S == l), mass + sum(S > j))
      W[l] <- V[l] * prod(1 - V[1:(l - 1)])
    }

    V[M] <- 1
    W[M] <- V[M] * prod(1 - V[1:(M - 1)])

    #---------------------------------
    # allocation step
    #---------------------------------

    # for all the observations
    for(i in 1:n){

      # for all the active clusters
      for(j in 1:M){
        probs[j] <- W[j] * dnorm(data[i], mean = mu[j], sd = sqrt(sigma2))
      }

      # sample from 1:J+1 with probability probs
      S[i] <- sample(1:M, size = 1, replace = F, prob = probs)
    }

    if(iter > burnin){
      res_part[iter - burnin, ] <- S
      mu_out[[iter - burnin]] <- mu
      probs_out[[iter - burnin]] <- W
    }

    setTxtProgressBar(pb, iter)
  }
  close(pb)
  return(list(res_part, mu_out))
}

#-------------------------------------------------------------
#-------------------------------------------------------------
#-------------------------------------------------------------
# with simulated data

data <- c(rnorm(50, -3, 1), rnorm(30, 3, 1), rnorm(20, 9, 1))
hist(data, breaks = 25, freq = F)
sq <- seq(-6, 13, by = 0.1)
fsq <- .5 * dnorm(sq, -3, 1) + .3 * dnorm(sq, 3, 1) + .2 * dnorm(sq, 9, 1)
lines(sq, fsq, col = 2)
```




```
# SAMPLE WITH BOTH STRATEGIES
sample_MAR <- MS_univ(data = data,
                      niter = 2000,
                      burnin = 1000,
                      m0 = 0,
                      sigma2_0 = 25,
                      sigma2 = 1,
                      mass = 1)
```

```
sample_TRSB <- TRSB_univ(data = data,
                         niter = 2000,
                         burnin = 1000,
                         m0 = 0,
                         sigma2_0 = 25,
                         sigma2 = 1,
                         mass = 1,
                         M = 50)
```

```
#-------------------------------------------------------------
#-------------------------------------------------------------
#-------------------------------------------------------------
# SOME DIAGNOSTIC
par(mfrow = c(3,1))
plot(1:1000, sample_MAR[[1]][,91], type = "l")
plot(1:1000, sample_MAR[[1]][,51], type = "l", col = 2)
plot(1:1000, sample_MAR[[1]][,1], type = "l", col = 3)
```




```
dev.off()
```

```
## null device
##           1
```

```
par(mfrow = c(3,1))
plot(1:1000, sample_TRSB[[1]][,91], type = "l")
plot(1:1000, sample_TRSB[[1]][,51], type = "l", col = 2)
plot(1:1000, sample_TRSB[[1]][,1], type = "l", col = 3)
dev.off()
```

```
## null device
##           1
```

```
# diagnostic on the number of clusters
CL_MAR <- apply(sample_MAR[[1]], 1, function(x) length(unique(x)))
CL_TRSB <- apply(sample_TRSB[[1]], 1, function(x) length(unique(x)))

coda_import <- as.mcmc(cbind(CL_MAR, CL_TRSB))
summary(coda_import)
```

```
##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean    SD Naive SE Time-series SE
## CL_MAR  6.395 1.667  0.05270        0.06342
## CL_TRSB 4.783 1.639  0.05183        0.04134
##
## 2. Quantiles for each variable:
##
##         2.5% 25% 50% 75% 97.5%
## CL_MAR     4   5   6   7    10
## CL_TRSB    2   4   5   6     8
```

```
geweke.diag(coda_import)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  CL_MAR CL_TRSB
##  0.1344 -1.7143
```

```
acfplot(coda_import)

# WHY NOT WITH THE PARAMETERS?

#-------------------------------------------------------------
#-------------------------------------------------------------
#-------------------------------------------------------------
# a first visualization of the clustering
# compute the posterior similarity matrix (PSM)
# then plot the heatmap
PSM <- function(cls){
  n <- ncol(cls)
  mat_out <- matrix(0, ncol = n, nrow = n)
  for(i in 1:n){
    for(j in 1:i){
      mat_out[i,j] <- mat_out[j,i] <- mean(cls[,i] == cls[,j])
    }
  }
  return(mat_out)
}

heatmap(PSM(sample_MAR[[1]]), Rowv = NA, Colv = NA)
heatmap(PSM(sample_TRSB[[1]]), Rowv = NA, Colv = NA)
```

BNP3

Riccardo Corradin
December 11, 2020

Recap

Let $\mathbb{N}_n = \{1, \dots, n\}$ space of first n integer.

Definition

We say that $\psi_n = \{A_1, \dots, A_k\}$ is a *partition* of $\mathbb{N}_n$ if the sets $\{A_1, \dots, A_k\}$ are nonempty and such that $\mathbb{N}_n = \cup_i A_i$ and $A_i \cap A_j = \emptyset$ for all $i \neq j$.

Let $\mathcal{B}$ be the space of all partitions of $\mathbb{N}_n$ and $\mathscr{B}$ its discrete σ-field.

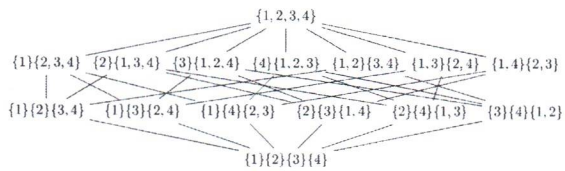
Definition

Let $(\Omega, \mathscr{F}, \mathrm{P})$ be a generic probability space. A random partition is any measurable function $\Psi_n : (\Omega, \mathscr{F}, \mathrm{P}) \to (\mathcal{B}, \mathscr{B})$.

The definition of Ψ_n induces a probability measure on the space of partitions $\mathcal{B}$.

Recap

Problem: the space of the possible partition of n elements is large, and is growing fast as far as n grows.



We have

$$S_{n,k} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (k-j)^n$$

(Stirling number of the second kind) ways to partition n elements in k groups, and

$$B_n = \sum_{k=1}^{n} S_{n,k}$$

(Bell number) ways to partition n elements. B_n is exploding as far as n grows.

Recap

We are in a mixture model framework. Suppose we have a set $\boldsymbol{Y} = \{Y_1, \dots, Y_n\}$ of observations, and we assume a mixture model

$$\tilde{f}(y) = \int_{\Theta} \mathcal{K}(y; \theta) P(\mathrm{d}\theta)$$

with $P \sim DP(a, P_0(\cdot))$. Equivalently

$$\begin{aligned} Y_i \mid \theta_i &\overset{\text{ind}}{\sim} \mathcal{K}(Y_i; \theta_i) \\ \theta_1, \dots, \theta_n \mid P &\overset{\text{iid}}{\sim} P \\ P &\sim DP(a, P_0(\cdot)) \end{aligned}$$

P a.s. discrete $\to$ the sequence $\theta_1, \dots, \theta_n$ may show ties
$\to$ latent partition Ψ_n in $\theta_1, \dots, \theta_n$

$\pi(\Psi_n) = p_k^{(n)}(n_1, \dots, n_k)$ EPPF of a Dirichlet process.

We saw how to sample from the posterior distribution of a DP mixture.

$$\tilde{f}(Y \mid Y_1, \dots, Y_n) = \int_{\Theta} \mathcal{K}(Y; \theta) P(\mathrm{d}\theta \mid \theta_1, \dots, \theta_n)$$

At each step we have a partition of the data, i.e. a realization from $\pi(\Psi_n \mid \boldsymbol{Y})$.

Recap

Scan the entire partitions space is unfeasible in a reasonable time, as far as $n \nearrow$.

n	1	2	3	4	5	6	7	8	9	10
B_n	1	2	5	15	52	203	877	4140	21147	115975

Given a set of observations, we aim to estimate the best partition latent in the data among the B_n possible partitions.

We set a prior $\pi(\Psi_n)$ on the partitions' space.

We are interested in area where the posterior concentrates its mass.

We need a strategy to reduce the number of partitions considered in the estimation, starting from our MCMC realizations.

Point estimate

We approach the problem from a decision theory point of view. Let $L(\cdot,\cdot)$ denotes a loss function, two partitions of n elements as arguments. We have that the optimal partition $\psi_n^* \in \mathbb{B}$ (or equivalently $(S_1 \dots, S_n) \in \mathbb{N}^n$) is the solution of

$$\psi_n^* = \arg\min_{\hat{\psi}_n \in \mathbb{B}} \left\{ \mathrm{E}_{\Psi_n}[L(\psi_n, \hat{\psi}_n) \mid \boldsymbol{Y}] \right\}$$

$$= \arg\min_{\hat{\psi}_n \in \mathbb{B}} \left\{ \sum_{\psi_n \in \mathbb{B}} L(\psi_n, \hat{\psi}_n) \mathscr{L}(\Psi_n = \psi_n \mid \boldsymbol{Y}) \right\}$$

where ψ_n^* denotes the optimal partition, $\hat{\psi}_n$ the partition we are considering, ψ_n the partitions wrt we take the expectation, and $\mathscr{L}(\psi_n \mid \boldsymbol{Y})$ is the posterior probability of ψ_n, given a set of observations $\boldsymbol{Y}$.

$L(\cdot,\cdot)$ = how much we're losing describing one partition in terms of the other (it's in terms of the distance between the two partitions)

- The space $\mathbb{B}$ is too large
- We can choose $L(\cdot,\cdot)$ in several ways
- We can set different priors on Ψ_n, for example the EPPF of a DP

1. Point estimate - $0-1$

= dummy loss function

A first loss function that we can consider is the $0-1$ loss function, i.e.

$$L_{0-1}(\psi_n, \hat{\psi}_n) = \mathbf{1}_{[\psi_n = \hat{\psi}_n]}$$

Such loss function is taking value 1 when all the blocks of ψ_n and $\hat{\psi}_n$ coincide, and 0 otherwise.

- Is not accommodating possible similarities in the partitions, all the partitions different from $\hat{\psi}_n$ are penalized in the same way
- the point estimate coincides with

$$\arg\min_{\hat{\psi}_n \in \mathbb{B}} \left\{ \sum_{\psi_n \in \mathbb{B}} \mathbf{1}_{[\psi_n = \hat{\psi}_n]} \mathscr{L}(\Psi_n = \psi_n \mid \boldsymbol{Y}) \right\} = \arg\max_{\hat{\psi}_n \in \mathbb{B}} \left\{ \mathscr{L}(\Psi_n = \hat{\psi}_n \mid \boldsymbol{Y}) \right\}$$

the maximum a posteriori of the distribution of Ψ_n.

2. Point estimate - Binder

A more relaxed loss function: the Binder loss function.

$$L_B(\psi_n, \hat{\psi}_n) = \sum_{j<i} \left[C_1 \mathbf{1}_{[S_i = S_j]} \mathbf{1}_{[\hat{S}_i \neq \hat{S}_j]} + C_2 \mathbf{1}_{[S_i \neq S_j]} \mathbf{1}_{[\hat{S}_i = \hat{S}_j]} \right]$$

here we're penalizing with C_1 if two elements agree in ψ_n and disagree in $\hat{\psi}_n$ and we penalize with C_2 if two elements agree in $\hat{\psi}_n$ but disagree in ψ_n.

and by setting $C_1 = C_2$ it can be written as

$$L_B(\psi_n, \hat{\psi}_n) = \frac{1}{2} \left(\sum_{i=1}^{k_n} n_{i\bullet}^2 + \sum_{j=1}^{\hat{k}_n} n_{\bullet j}^2 - 2 \sum_{j=1}^{\hat{k}_n} \sum_{i=1}^{k_n} n_{ij}^2 \right)$$

it is based on the cluster frequencies, where

- k_n is the number of blocks in ψ_n
- $\hat{k}_n$ is the number of blocks in $\hat{\psi}_n$
- $n_{ij} = \sum_{\ell=1}^{n} \mathbf{1}_{[S_\ell = i]} \mathbf{1}_{[\hat{S}_\ell = j]}$ is the number of observations in the i-th block of ψ_n and in the j-th block of $\hat{\psi}_n$
- $n_{i\bullet} = \sum_{j=1}^{\hat{k}_n} n_{ij} = \sum_{t=1}^{n} \mathbf{1}_{[S_\ell = i]}$ is the number of observations in the i-th block of ψ_n
- $n_{\bullet j} = \sum_{i=1}^{k_n} n_{ij} = \sum_{t=1}^{n} \mathbf{1}_{[\hat{S}_\ell = j]}$ is the number of observations in the j-th block of $\hat{\psi}_n$

11/12/2020
2/6

Point estimate - Binder

You can show that the solution of

$$\psi_n^* = \arg\min_{\hat{\psi}_n \in \mathbb{B}} \left\{ \sum_{\psi_n \in \mathbb{B}} L_B(\psi_n, \hat{\psi}_n) \mathscr{L}(\Psi_n = \psi_n \mid \boldsymbol{Y}) \right\}$$

is given by the partition which, in binary representation, minimize the distance from the posterior similarity matrix $\mathcal{M}$, where

$$\mathcal{M}_{ij} = P(S_i = S_j \mid \boldsymbol{Y})$$

and

$$\boldsymbol{S}^* = \arg\min_{\hat{\boldsymbol{S}} \in \mathbb{N}^n} \left\{ \sum_{i<j} |\mathbf{1}_{[\hat{S}_i = \hat{S}_j]} - \mathcal{M}_{ij}| \right\}$$

The Binder loss function, differently from the $0-1$ loss function, is not penalizing in the same way the partitions different from the optimal one.

9/15

Point estimate - Binder

A practical example: consider $n = 3$, and all the possible partitions are

$$\mathbb{B} = \{\{1,2,3\}, \{12,3\}, \{1,23\}, \{13,2\}, \{123\}\}$$

Equivalently in a binary representation

$$A_1 = \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 0 & 0 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 1 & & \\ 1 & 1 & \\ 0 & 0 & 1 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 0 & 1 & 1 \end{bmatrix},$$
$$A_4 = \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 1 & 0 & 1 \end{bmatrix}, \quad A_5 = \begin{bmatrix} 1 & & \\ 1 & 1 & \\ 1 & 1 & 1 \end{bmatrix}$$

posterior prob. that the observation **1** and **2** are grouped together

and $\mathcal{M} = \begin{bmatrix} 1 & & \\ 0.8 & 1 & \\ 0.2 & 0.2 & 1 \end{bmatrix}$, then we have

$$d(A_1, \mathcal{M}) = 0.8 + 0.2 + 0.2 = 1.2; \quad d(A_2, \mathcal{M}) = 0.6; \quad d(A_3, \mathcal{M}) = 1.8;$$
$$d(A_4, \mathcal{M}) = 1.8; \quad d(A_5, \mathcal{M}) = 2.4$$

and the point estimate is given by A_2.

10/15

3. Point estimate - Variation of information

Another loss function: variation of information.

$$L_{VI}(\psi_n, \hat{\psi}_n) = H(\psi_n) + H(\hat{\psi}_n) - 2I(\psi_n, \hat{\psi}_n)$$

where $H(\cdot)$ denotes the entropy and $I(\cdot,\cdot)$ the relative entropy, and it can be written as

$$L_{VI}(\psi_n, \hat{\psi}_n) = \sum_{i=1}^{k_n} \frac{n_{i\cdot}}{n} \log\left(\frac{n_{i\cdot}}{n}\right) + \sum_{j=1}^{\hat{k}_n} \frac{n_{\cdot j}}{n} \log\left(\frac{n_{\cdot j}}{n}\right) - 2\sum_{j=1}^{\hat{k}_n}\sum_{i=1}^{k_n} \frac{n_{ij}}{n} \log\left(\frac{n_{ij}}{n}\right)$$

it is based on the cluster frequencies, where

- k_n is the number of blocks in ψ_n
- $\hat{k}_n$ is the number of blocks in $\hat{\psi}_n$
- $n_{ij} = \sum_{\ell=1}^n \mathbf{1}_{[S_\ell = i]} \mathbf{1}_{[\hat{S}_\ell = j]}$ is the number of observations in the i-th block of ψ_n and in the j-th block of $\hat{\psi}_n$
- $n_{i\cdot} = \sum_{j=1}^{\hat{k}_n} n_{ij} = \sum_{\ell=1}^n \mathbf{1}_{[S_\ell = i]}$ is the number of observations in the i-th block of ψ_n
- $n_{\cdot j} = \sum_{i=1}^{k_n} n_{ij} = \sum_{\ell=1}^n \mathbf{1}_{[\hat{S}_\ell = j]}$ is the number of observations in the j-th block of $\hat{\psi}_n$

11/15

Point estimate

We still have a problem: the partitions' space is too large.

We restrict our attention to a subset of the entire partition space $\mathbb{S}_R \subset \mathbb{B}$, the space of the partition visited during R steps of an MCMC algorithm.

Our solution is sub-optimal, but feasible

$$\psi_n^* = \arg\min_{\psi_n \in \mathbb{B}} \left\{ \mathrm{E}_{\psi_n}[L(\psi_n, \hat{\psi}_n) \mid \boldsymbol{Y}] \right\} \simeq \arg\min_{\hat{\psi}_n \in \mathbb{S}_R} \left\{ \mathrm{E}_{\psi_n}[L(\psi_n, \hat{\psi}_n) \mid \boldsymbol{Y}] \right\}$$

Instead of considering B_n elements, we consider only R partitions. You can also reduce this number due to the fact that some partitions are visited more than one time.

- This strategy can be relaxed by scanning the neighborhood of the partitions visited by the sampler, with a *greedy* strategy;
- You can also provide an estimation of the uncertainty associated to the point estimate by defining a credible ball on the partition space.

Instead of considering the entire partition space we restrict our attention just to the subset of the partitions that we visit during the MCMC estimation. We saw that at each iteration we produce a partition: instead of considering the whole partition space we consider the R partitions that we visited during the estimation of the model

12/15

Examples

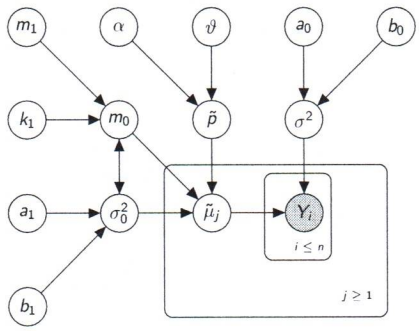
Location Gaussian DP mixture:

$$\begin{aligned} K(Y;\mu,\sigma^2) &\overset{\text{ind}}{\sim} N(\mu,\sigma^2) \\ \mu \mid P &\overset{\text{iid}}{\sim} P \\ P &\sim DP(a, P_0) \\ P_0 &\sim N(m_0, \sigma_0^2) \\ \sigma^2 &\sim IG(a_0, b_0) \\ (m_0, \sigma_0^2) &\sim NIG(m_1, k_1, a_1, b_1) \end{aligned}$$

Location-scale Gaussian DP mixture:

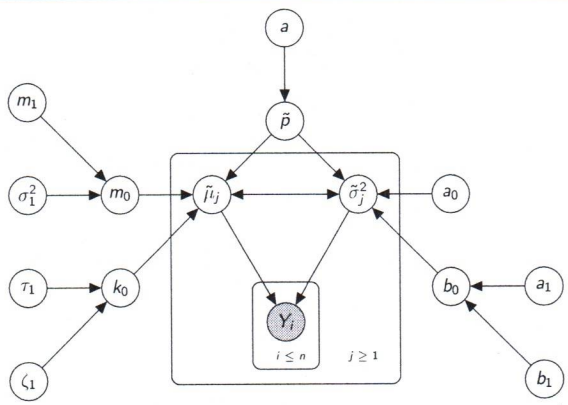
$$\begin{aligned} K(Y;\mu,\sigma^2) &\overset{\text{ind}}{\sim} N(\mu,\sigma^2) \\ (\mu,\sigma^2) \mid P &\overset{\text{iid}}{\sim} P \\ P &\sim DP(a, P_0) \\ P_0 &\sim NIG(m_0, k_0, a_0, b_0) \\ m_0 &\sim N(m_1, \sigma_1^2) \\ k_0 &\sim G(\tau_1, \zeta_1) \\ b_0 &\sim G(a_1, b_1) \end{aligned}$$

Examples



Hierarchical representation of the univariate location DP mixture model with Gaussian kernel and Gaussian base measure.

Examples



Hierarchical representation of the univariate location-scale DP mixture model with Gaussian kernel and Normal-Inverse-Gamma

See the code!

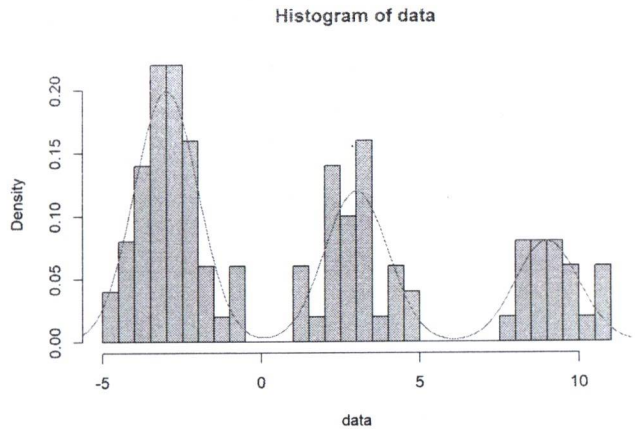
```r
# R & STAN are friends!
library(coda)
library(BNPmix)

# for plots
library(ggplot2)
library(tidyr)
library(dplyr)
```

```r
library(purrr)
library(ggsci)
require(gplots)
```

```r
require(ggpubr)
```

```r
# MARGINAL SAMPLER
MS_univ <- function(data, niter, burnin, m0, sigma2_0, sigma2, mass){

  n <- length(data)

  res_part <- matrix(0, ncol = n, nrow = niter - burnin)
  mu <- m0

  # initialize the partition - all in ome cluster
  S <- rep(1, n)

  mu_out <- list()
  probs <- c()

  pb <- txtProgressBar(0, niter, style = 3)
  # for all the iterations
  for(iter in 1:niter){

    #---------------------------------
    # allocation step
    #---------------------------------

    # for all the observations
    for(i in 1:n){

      # J = number of current clusters
      J <- length(mu)
      probs <- rep(0, J + 1)

      # for all the active clusters
      for(j in 1:J){
        probs[j] <- (sum(S[-i] == j) / (n + mass - 1)) * dnorm(data[i], mean = mu[j], sd = sqrt(sigma2))
      }

      # probability of new cluster
      probs[J + 1] <- (mass / (n + mass - 1)) * dnorm(data[i], mean = m0, sd = sqrt(sigma2 + sigma2_0))

      # sample from 1:J+1 with probability probs
      S[i] <- sample(1:(J + 1), size = 1, replace = F, prob = probs)

      # if the cluster is new, generate its mean value
      if(S[i] > J){
        mu[S[i]] <- rnorm(1, mean = (data[i] / sigma2 + m0 / sigma2_0) * (1 / sigma2 + 1 / sigma2_0),
                          sd = sqrt(1 / (1 / sigma2 + 1 / sigma2_0)))
      }
    }

    # resize the objects
    mu <- mu[unique(sort(S))]
    S <- as.numeric(as.factor(S))

    if(iter > burnin){
      res_part[iter - burnin, ] <- S
      mu_out[[iter - burnin]] <- mu
    }

    #---------------------------------
    # acceleration step
    #---------------------------------

    for(j in 1:length(mu)){
      ntemp <- length(data[S == j])
      mu[j] <- rnorm(1, mean = (sum(data[S == j]) / sigma2 + m0 / sigma2_0) * (length(data[S == j]) / sigma2
+ 1 / sigma2_0),
                     sd = sqrt(length(data[S == j]) / (1 / sigma2 + 1 / sigma2_0)))
    }
    setTxtProgressBar(pb, iter)
  }
  close(pb)
  return(list(res_part, mu_out))
}

# TRUNCATED SB SAMPLER
TRSB_univ <- function(data, niter, burnin, m0, sigma2_0, sigma2, mass, M){

  n <- length(data)
  res_part <- matrix(0, ncol = n, nrow = niter - burnin)
  mu <- m0
  W <- V <- c()
```

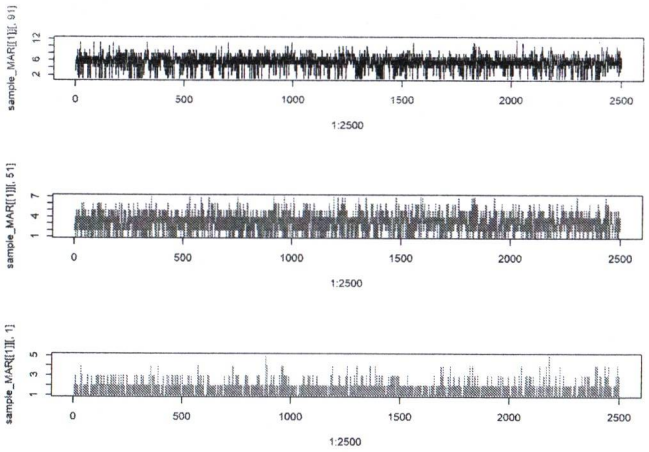
```
  # initialize the partition - all in one cluster
  S <- rep(1, n)

  mu_out <- list()
  probs_out <- list()
  probs <- c()

  pb <- txtProgressBar(0, niter, style = 3)
  # for all the iterations
  for(iter in 1:niter){
    #---------------------------------
    # UPDATE THE WEIGHTS and THE ATOMS
    #---------------------------------

    for(j in 1:M){
      ntemp <- length(data[S == j])
      mu[j] <- rnorm(1, mean = (sum(data[S == j]) / sigma2 + m0 / sigma2_0) * (length(data[S == j]) / sigma2
+ 1 / sigma2_0),
                     sd = sqrt(1 / (length(data[S == j]) / sigma2 + 1 / sigma2_0)))
    }

    V[1] <- rbeta(1, 1 + sum(S == 1), mass + sum(S > 1))
    W[1] <- V[1]

    for(l in 2:(M - 1)){
      V[l] <- rbeta(1, 1 + sum(S == l), mass + sum(S > j))
      W[l] <- V[l] * prod(1 - V[1:(l - 1)])
    }

    V[M] <- 1
    W[M] <- V[M] * prod(1 - V[1:(M - 1)])

    #---------------------------------
    # allocation step
    #---------------------------------

    # for all the observations
    for(i in 1:n){

      # for all the active clusters
      for(j in 1:M){
        probs[j] <- W[j] * dnorm(data[i], mean = mu[j], sd = sqrt(sigma2))
      }

      # sample from 1:J+1 with probability probs
      S[i] <- sample(1:M, size = 1, replace = F, prob = probs)
    }

    if(iter > burnin){
      res_part[iter - burnin, ] <- S
      mu_out[[iter - burnin]] <- mu
      probs_out[[iter - burnin]] <- W
    }

    setTxtProgressBar(pb, iter)
  }
  close(pb)
  return(list(res_part, mu_out))
}

#-------------------------------------------------------------
#-------------------------------------------------------------
#-------------------------------------------------------------
# with simulated data

data <- c(rnorm(50, -3, 1), rnorm(30, 3, 1), rnorm(20, 9, 1))
hist(data, breaks = 25, freq = F)
sq <- seq(-6, 13, by = 0.1)
fsq <- .5 * dnorm(sq, -3, 1) + .3 * dnorm(sq, 3, 1) + .2 * dnorm(sq, 9, 1)
lines(sq, fsq, col = 2)
```




```
# SAMPLE WITH BOTH STRATEGIES
sample_MAR <- MS_univ(data = data,
                      niter = 5000,
                      burnin = 2500,
                      m0 = 0,
                      sigma2_0 = 100,
                      sigma2 = 1,
                      mass = 1)
```

```
sample_TRSB <- TRSB_univ(data = data,
                         niter = 5000,
                         burnin = 2500,
                         m0 = 0,
                         sigma2_0 = 100,
                         sigma2 = 1,
                         mass = 1,
                         M = 50)
```

```
#-------------------------------------------------------------
#-------------------------------------------------------------
#-------------------------------------------------------------
# SOME DIAGNOSTIC
par(mfrow = c(3,1))
plot(1:2500, sample_MAR[[1]][,91], type = "l")
plot(1:2500, sample_MAR[[1]][,51], type = "l", col = 2)
plot(1:2500, sample_MAR[[1]][,1], type = "l", col = 3)
```




```
dev.off()
```

```
## null device
##           1
```

```
par(mfrow = c(3,1))
plot(1:2500, sample_TRSB[[1]][,91], type = "l")
plot(1:2500, sample_TRSB[[1]][,51], type = "l", col = 2)
plot(1:2500, sample_TRSB[[1]][,1], type = "l", col = 3)
dev.off()
```

```
## null device
##           1
```

```
# diagnostic on the number of clusters
CL_MAR <- apply(sample_MAR[[1]], 1, function(x) length(unique(x)))
CL_TRSB <- apply(sample_TRSB[[1]], 1, function(x) length(unique(x)))

coda_import <- as.mcmc(cbind(CL_MAR, CL_TRSB))
summary(coda_import)
```

```
##
## Iterations = 1:2500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean     SD Naive SE Time-series SE
## CL_MAR  6.417 1.555  0.03109        0.03310
## CL_TRSB 4.793 1.485  0.02970        0.02436
##
## 2. Quantiles for each variable:
##
##         2.5% 25% 50% 75% 97.5%
## CL_MAR     4   5   6   7    10
## CL_TRSB    2   4   5   6     8
```

```
geweke.diag(coda_import)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  CL_MAR CL_TRSB
##  2.2153 -0.1163
```

```
acfplot(coda_import)

# WHY NOT WITH THE PARAMETERS?

#------------------------------------------------------------
#------------------------------------------------------------
#------------------------------------------------------------
# a first visualization of the clustering
# compute the posterior similarity matrix (PSM)
# then plot the heatmap

PSM <- function(cls){
  n <- ncol(cls)
  mat_out <- matrix(0, ncol = n, nrow = n)
  for(i in 1:n){
    for(j in 1:i){
      mat_out[i,j] <- mat_out[j,i] <- mean(cls[,i] == cls[,j])
    }
  }
  return(mat_out)
}

heatmap(PSM(sample_MAR[[1]]), Rowv = NA, Colv = NA)
heatmap(PSM(sample_TRSB[[1]]), Rowv = NA, Colv = NA)

#------------------------------------------------------------
#------------------------------------------------------------
#------------------------------------------------------------

est_part_BINDER <- function(clust, PSM){
  res_loss <- c()
  for(i in 1:nrow(clust)){
    binary_part <- sapply(clust[i,], function(x) as.numeric(x == clust[i,]))
    res_loss[i] <- sum(abs(binary_part - PSM))
  }
  return(clust[which.min(res_loss),])
}

#------------------------------------------------------------
#------------------------------------------------------------
#------------------------------------------------------------
# DUMMY EXAMPLE

part <- matrix(c(
  1,2,1,1,2,3,
  1,1,1,2,2,2,
  1,2,1,2,2,3,
  3,1,3,3,1,1,
  3,1,3,3,2,2
), ncol = 6, byrow = T)

PSM(part)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  1.0  0.2  1.0  0.6  0.0  0.0
## [2,]  0.2  1.0  0.2  0.2  0.6  0.2
## [3,]  1.0  0.2  1.0  0.6  0.0  0.0
## [4,]  0.6  0.2  0.6  1.0  0.4  0.2
## [5,]  0.0  0.6  0.0  0.4  1.0  0.6
## [6,]  0.0  0.2  0.0  0.2  0.6  1.0
```

```
est_part_BINDER(part, PSM(part))
```

```
## [1] 1 2 1 1 2 3
```

```
#-------------------------------------------------------------
#-------------------------------------------------------------
#-------------------------------------------------------------

part_MAR <- est_part_BINDER(sample_MAR[[1]], PSM(sample_MAR[[1]]))
part_TRSB <- est_part_BINDER(sample_TRSB[[1]], PSM(sample_TRSB[[1]]))
table(part_MAR, part_TRSB)
```

```
##         part_TRSB
## part_MAR  1  2  5
##        1 20  0  0
##        2  0  0 47
##        3  0  0  3
##        4  0 30  0
```

```
#-------------------------------------------------------------
#-------------------------------------------------------------
#-------------------------------------------------------------
# USING AN EXTERNAL PACKAGE

library(BNPmix)
?PYdensity
```

```
## starting httpd help server ...
```

```
##  done
```

```
model_L <- PYdensity(data,
                     mcmc = list(niter = 5000, nburn = 2500, method = "MAR", model = "L", hyper = F),
                     prior = list(m0 = 0, s20 = 25, a0 = 2, b0 = 1))
```

```
## Completed:   500/5000 - in 1.804 sec
## Completed:   1000/5000 - in 2.877 sec
## Completed:   1500/5000 - in 4.02 sec
## Completed:   2000/5000 - in 5.095 sec
## Completed:   2500/5000 - in 6.446 sec
## Completed:   3000/5000 - in 7.552 sec
## Completed:   3500/5000 - in 8.727 sec
## Completed:   4000/5000 - in 9.805 sec
## Completed:   4500/5000 - in 10.935 sec
## Completed:   5000/5000 - in 11.991 sec
##
## Estimation done in 11.991 seconds
```

```
print(model_L)
```

```
## BNPdens object
## Type: univariate density
```

```
summary(model_L)
```

```
## PYdensity function call:
##  2500    burn-in iterations
##  5000    iterations
##  Global estimation time: 11.99 seconds
##  Average number of groups:  15.908
##  Min number of groups:  5 ; Max number of groups:  27
```

```
plot(model_L, show_hist = T, show_clust = T)

# diagnostic
coda_L <- BNPdens2coda(model_L)
summary(coda_L)
```

```
##
## Iterations = 1:2500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       15.90800        3.18302        0.06366        0.12707
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
##    10    14    16    18    22
```

```
geweke.diag(coda_L)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   var1
## -1.017
```

```
acfplot(coda_L)
traceplot(coda_L)

# point estimate for the latent partition
opt_part_L <- partition(model_L, dist = "Binder")
best_L <- opt_part_L$partitions[1,]

#-------------------------------------------------------------
# LOCATION-SCALE MODEL
model_LS <- PYdensity(data,
                      mcmc = list(niter = 5000, nburn = 2500, method = "MAR", model = "LS", hyper = F),
                      prior = list(m0 = 0, k0 = 0.1, a0 = 2, b0 = 1))
```

```
## Completed:   500/5000 - in 0.219 sec
## Completed:   1000/5000 - in 0.458 sec
## Completed:   1500/5000 - in 0.692 sec
## Completed:   2000/5000 - in 0.917 sec
## Completed:   2500/5000 - in 1.137 sec
## Completed:   3000/5000 - in 1.38 sec
## Completed:   3500/5000 - in 1.642 sec
## Completed:   4000/5000 - in 1.893 sec
## Completed:   4500/5000 - in 2.124 sec
## Completed:   5000/5000 - in 2.378 sec
##
## Estimation done in 2.378 seconds
```

```
print(model_LS)
```

```
## BNPdens object
## Type: univariate density
```

```
summary(model_LS)
```

```
## PYdensity function call:
##  2500    burn-in iterations
##  5000    iterations
##  Global estimation time: 2.38 seconds
##  Average number of groups:  5.9648
##  Min number of groups:  3 ; Max number of groups:  13
```

```
plot(model_LS, show_hist = T, show_clust = T)

# diagnostic
coda_LS <- BNPdens2coda(model_LS)
summary(coda_LS)
```

```
##
## Iterations = 1:2500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##         5.9648         1.5648         0.0313         0.0869
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
##     3     5     6     7     9
```

```
geweke.diag(coda_LS)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    var1
## -0.8264
```

```
acfplot(coda_LS)
traceplot(coda_LS)

# point estimate for the latent partition
opt_part_LS <- partition(model_LS, dist = "Binder")
best_LS <- opt_part_LS$partitions[1,]

table(best_L, best_LS)
```

```
##       best_LS
## best_L  1  2  3  4
##      1  15  0  0  0
##      2  29  0  0  0
##      3   0  1  0  0
##      4   3  0  0  0
##      5   0  1  0  0
##      6   0  1  0  0
##      7   0  0 23  0
##      8   0  0  4  0
##      9   0  0  3  0
##      10  0  0  0 11
##      11  0  0  0  5
##      12  0  0  0  3
##      13  0  0  0  1
```

```
#-------------------------------------------------------------
#-------------------------------------------------------------
#-------------------------------------------------------------
# UNIVARIATE CASE - a real case study

load(file = "gal_data.dat")
data_temp <- gal_data[gal_data[,2] == "luminosity = H/M" & gal_data[,3] == "environment = M/H",1]

model_MAR <- PYdensity(data_temp,
                       mcmc = list(niter = 5000, nburn = 2500,
                                   method = "MAR", model = "LS", hyper = F, print_message = T),
                       prior = list(m0 = mean(data_temp), k0 = .1, a0 = 4, b0 = 1),
                       output = list(grid = seq(0, 4, by = 0.05)))
```

```
## Completed:   500/5000 - in 5.867 sec
## Completed:   1000/5000 - in 12.062 sec
## Completed:   1500/5000 - in 18.361 sec
## Completed:   2000/5000 - in 23.914 sec
## Completed:   2500/5000 - in 29.983 sec
## Completed:   3000/5000 - in 35.807 sec
## Completed:   3500/5000 - in 41.659 sec
## Completed:   4000/5000 - in 47.602 sec
## Completed:   4500/5000 - in 53.972 sec
## Completed:   5000/5000 - in 59.373 sec
##
## Estimation done in 59.373 seconds
```

```
coda_model_MAR <- BNPdens2coda(model_MAR)
summary(coda_model_MAR)
```

```
##
## Iterations = 1:2500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##         6.6412         1.6402         0.0328         0.1000
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
##     4     5     6     8    10
```

```
plot(coda_model_MAR)
effectiveSize(coda_model_MAR)
```

```
##     var1
## 269.0205
```

```
geweke.diag(coda_model_MAR)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   var1
## -1.023
```

```
summary(model_MAR)
```

```
## PYdensity function call:
##  2500    burn-in iterations
##  5000    iterations
##  Global estimation time: 59.37 seconds
##  Average number of groups:  6.6412
##  Min number of groups:  3 ; Max number of groups:  14
```

```
plot(model_MAR, show_points = T, show_hist = T, show_clust = T)

#-------------------------------------------------------------
# expected number of cluster

ENOC <- function(alpha, n){
  sum(alpha / (alpha + 0:(n - 1)))
}
ENOC(1, length(data_temp))
```

```
## [1] 7.471379
```

```
part_MAR <- partition(model_MAR, dist = "VI")
table(part_MAR$partitions[1,])
```

```
##
##   1   2   3   4
## 287 684   9   6
```

```
part_MARB <- partition(model_MAR, dist = "Binder")
table(part_MAR$partitions[1,])
```

```
##
##   1   2   3   4
## 287 684   9   6
```

```
#-----------------------------------------------------------
# WITH HYPERPRIORS

load(file = "gal_data.dat")
data_temp <- gal_data[gal_data[,2] == "luminosity = H/M" & gal_data[,3] == "environment = M/H",1]

model_MARH <- PYdensity(data_temp,
                        mcmc = list(niter = 5000, nburn = 2500,
                                    method = "MAR", model = "LS", hyper = T, print_message = T),
                        output = list(grid = seq(0, 4, by = 0.05)))
```

```
## Completed:   500/5000 - in 7.553 sec
## Completed:   1000/5000 - in 15.464 sec
## Completed:   1500/5000 - in 23.621 sec
## Completed:   2000/5000 - in 32.307 sec
## Completed:   2500/5000 - in 39.667 sec
## Completed:   3000/5000 - in 47.282 sec
## Completed:   3500/5000 - in 56.181 sec
## Completed:   4000/5000 - in 64.887 sec
## Completed:   4500/5000 - in 73.507 sec
## Completed:   5000/5000 - in 81.874 sec
##
## Estimation done in 81.874 seconds
```

```
coda_model_MARH <- BNPdens2coda(model_MARH)
summary(coda_model_MARH)
```

```
##
## Iterations = 1:2500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##        9.11560        2.12830        0.04257        0.16516
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
##     5     8     9    10    14
```

```
plot(coda_model_MARH)
effectiveSize(coda_model_MARH)
```

```
##    var1
## 166.055
```

```
geweke.diag(coda_model_MARH)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   var1
## -2.408
```

```
summary(model_MARH)
```

```
## PYdensity function call:
##  2500    burn-in iterations
##  5000    iterations
##  Global estimation time: 81.87 seconds
##  Average number of groups:  9.1156
##  Min number of groups:  3 ; Max number of groups:  17
```

```
plot(model_MARH, show_points = T, show_hist = T, show_clust = T)

part_MARH <- partition(model_MARH, dist = "VI")
table(part_MARH$partitions[1,])
```

```
##
##   1   2   3
## 296 681   9
```

```
part_MARBH <- partition(model_MARH, dist = "Binder")
table(part_MARH$partitions[1,])
```

```
##
##   1   2   3
## 296 681   9
```

```
#-----------------------------------------------------------
#-----------------------------------------------------------
#-----------------------------------------------------------
# MULTIVARIATE CASE - a real case study
load(file = "clust_data.dat")
clust_data <- as.data.frame(clust_data)
colnames(clust_data) <- c("[Fe/H]", "X", "Y")


model_MAR_mv <- PYdensity(clust_data,
                          mcmc = list(niter = 5000, nburn = 2500,
                                      method = "MAR", model = "LS", hyper = F, print_message = T),
                          prior = list(m0 = colMeans(clust_data), k0 = .1, n0 = 6, Sigma0 = diag(4, 3)),
                          output = list(grid = expand.grid(seq(-4, 4, by = 0.2),
                                                           seq(-4, 4, by = 0.2),
                                                           seq(-4, 4, by = 0.2)), out_type = "MEAN"))
```

```
## Completed:   500/5000 - in 0.15 sec
## Completed:   1000/5000 - in 0.284 sec
## Completed:   1500/5000 - in 0.416 sec
## Completed:   2000/5000 - in 0.551 sec
## Completed:   2500/5000 - in 0.687 sec
## Completed:   3000/5000 - in 4.203 sec
## Completed:   3500/5000 - in 7.68 sec
## Completed:   4000/5000 - in 11.198 sec
## Completed:   4500/5000 - in 14.759 sec
## Completed:   5000/5000 - in 18.234 sec
##
## Estimation done in 18.234 seconds
```

```
#-----------------------------------------------------------
# diagnostic

coda_MAR_mv <- BNPdens2coda(model_MAR_mv)
summary(coda_MAR_mv)
```

```
##
## Iterations = 1:2500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##              1              0              0              0
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
##     1     1     1     1     1
```

```
plot(coda_MAR_mv)
effectiveSize(coda_MAR_mv)
```

```
## var1
##    0
```

```
summary(model_MAR_mv)
```

```
## PYdensity function call:
##  2500    burn-in iterations
##  5000    iterations
##  Global estimation time: 18.23 seconds
##  Average number of groups:  1
##  Min number of groups:  1 ; Max number of groups:  1
```

```
#-----------------------------------------------------------
# plots

p1 <- plot(model_MAR_mv, show_points = T, show_clust = T, dimension = c(1,2))
p2 <- plot(model_MAR_mv, show_points = T, show_clust = T, dimension = c(1,3))
p3 <- plot(model_MAR_mv, show_points = T, show_clust = T, dimension = c(2,3))
ggarrange(p1, p2, NULL, p3, ncol = 2, nrow = 2)

part_MAR_mv <- partition(model_MAR_mv, dist = "VI")
table(part_MAR_mv$partitions[1,])
```

```
##
##   1
## 139
```

```
part_MAR_mvB <- partition(model_MAR_mv, dist = "Binder")
table(part_MAR_mvB$partitions[1,])
```

```
##
##   1
## 139
```

```
#-----------------------------------------------------------
#-----------------------------------------------------------
model_MAR_mvH <- PYdensity(clust_data,
                           mcmc = list(niter = 5000, nburn = 2500,
                                       method = "MAR", model = "LS", hyper = T, print_message = T),
                           output = list(grid = expand.grid(seq(-4, 4, by = 0.2),
                                                            seq(-4, 4, by = 0.2),
                                                            seq(-4, 4, by = 0.2)), out_type = "MEAN"))
```

```
## Completed:   500/5000 - in 0.341 sec
## Completed:   1000/5000 - in 0.697 sec
## Completed:   1500/5000 - in 1.064 sec
## Completed:   2000/5000 - in 1.423 sec
## Completed:   2500/5000 - in 1.784 sec
## Completed:   3000/5000 - in 13.3 sec
## Completed:   3500/5000 - in 24.304 sec
## Completed:   4000/5000 - in 34.29 sec
## Completed:   4500/5000 - in 46.047 sec
## Completed:   5000/5000 - in 57.668 sec
##
## Estimation done in 57.668 seconds
```

```
#-----------------------------------------------------------
# diagnostic

coda_MAR_mv <- BNPdens2coda(model_MAR_mvH)
summary(coda_MAR_mv)
```

```
##
## Iterations = 1:2500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##        3.62240        0.68294        0.01366        0.05742
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
##     3     3     4     4     5
```

```
plot(coda_MAR_mv)
effectiveSize(coda_MAR_mv)
```

```
##     var1
## 141.4634
```

```
summary(model_MAR_mvH)
```

```
## PYdensity function call:
##  2500    burn-in iterations
##  5000    iterations
##  Global estimation time: 57.67 seconds
##  Average number of groups:  3.6224
##  Min number of groups:  3 ; Max number of groups:  7
```

```
#-----------------------------------------------------------
# plots

p1 <- plot(model_MAR_mvH, show_points = T, show_clust = T, dimension = c(1,2))
p2 <- plot(model_MAR_mvH, show_points = T, show_clust = T, dimension = c(1,3))
p3 <- plot(model_MAR_mvH, show_points = T, show_clust = T, dimension = c(2,3))
ggarrange(p1, p2, NULL, p3, ncol = 2, nrow = 2)

part_MAR_mvH <- partition(model_MAR_mvH, dist = "VI")
table(part_MAR_mvH$partitions[1,])
```

```
##
##   1   2
## 122  17
```

```
part_MAR_mvBH <- partition(model_MAR_mvH, dist = "Binder")
table(part_MAR_mvBH$partitions[1,])
```

```
##
##   1   2   3   4
## 105  17   7  10
```