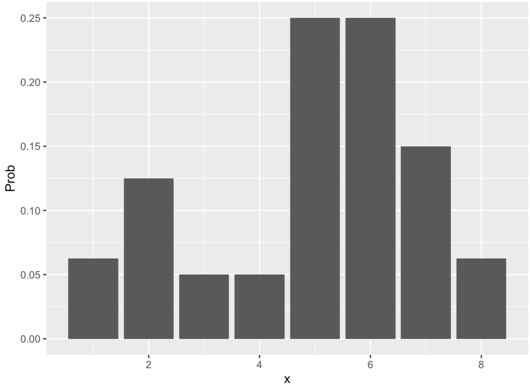
STA 365: Applied Bayesian Statistics

Boris Babic
Assistant Professor, University of Toronto

Week 8A: Markov Chain Monte Carlo

The Metropolis Algorithm

- Suppose we want to sample X which follows a discrete distribution on the integers 1-8.
- Suppose that the probabilities that $X = x$, for 1-8, are given by $p = (0.0625, 0.1250, 0.0500, 0.0500, 0.2500, 0.2500, 0.1500, 0.0625)$.



The Metropolis Algorithm

- To simulate from this probability distribution, we will take a simple random walk described as follows.
  1. We start at any possible location of our random variable X.
  2. To decide where to visit next, a fair coin is flipped. If the coin lands heads, we consider visiting the location one value to the left. If the coin lands tails, we consider visiting the location one value to the right. We call this location the candidate location.
  3. We compute
     $$R = \frac{p(candidate)}{p(current)}$$
     the ratio of the probabilities at the candidate and current locations. Notice that we only need to define the target distribution p up to proportionality.
  4. We spin a continuous spinner that lands anywhere from 0 to 1. Call the random spin Y. If $R > Y$, we indeed move to the candidate location. Otherwise, we remain at the current location.
- Steps 1 through 4 define an irreducible, aperiodic, positive recurrent Markov chain on the state values $(1, 2, ..., 8)$ where Step 1 gives the starting location and Steps 2-4 define the transition matrix P.
- We can "discover" the discrete probability distribution p by starting at any location and walking through the distribution many times repeating Steps 2, 3, and 4 (propose a candidate location, compute the ratio, and decide whether to visit the candidate location).

The Metropolis Algorithm

- The general algorithm is a generalization of the random walk example above.
- The MCMC sampling strategy sets up an irreducible, aperiodic, positive recurrent Markov chain for which the stationary distribution equals the posterior distribution of interest.

The Metropolis Algorithm

- Let $\pi(\theta|y) \propto \pi(\theta) f(y|\theta)$
- The general algorithm proceeds as follows:
  1. Start: Select a θ value for which there is positive posterior density. Call it $\theta^{(0)}$. This is the starting value.
  2. Propose: Given a current simulated value $\theta^{(j)}$, propose a new value $\theta^{(proposed)}$, which is selected at random in the interval $(\theta^{(j)} - C, \theta^{(j)} + C)$, where C is a pre-selected constant.
  3. Acceptance probability: Compute the ratio R of the posterior density at the proposed value and the current value:
  $$R = \frac{\pi(\theta^{(proposed)}|y)}{\pi(\theta^{(j)}|y)}$$
  The acceptance probability is the minimum of R and 1: $\text{Pr} = \min(R, 1)$.
  4. Move or Stay: Simulate a uniform random variable U. If U is smaller than the acceptance probability Pr, move to the proposed value $\theta^{(proposed)}$, otherwise stay at the current value $\theta^{(j)}$. Hence
  $$\theta^{(j+1)} = \begin{cases} \theta^{(proposed)}, U < \text{Pr} \\ \theta^{(j)} \text{ otherwise.} \end{cases}$$
- That is one step. One continues by returning to Step 2, proposing a new simulated value, computing an acceptance probability, deciding to move to the proposed value or stay, and so on.

Gibbs Sampling

- The Metropolis algorithm above simulates values from a posterior distribution of a single unknown parameter.
- The Gibbs algorithm is an MCMC algorithm for simulating from a probability distribution of several variables based on their individual conditional distributions.

Gibbs Sampling

Suppose we have a joint distribution $p(\theta_1, \dots, \theta_k)$ as our target density distribution, where $\boldsymbol{\theta} = (\theta_1, \dots, \theta_k)$.

We can use the Gibbs sampler to sample from the joint distribution if we knew the **full conditional** distribution for each parameter

For each parameter, the **full conditional** distribution is the distribution of the parameter conditional on the known information and all the other parameters $p(\theta_j \mid \boldsymbol{\theta}_{-j}, y)$, where $\boldsymbol{\theta}_{-j} = (\theta_1, \dots, \theta_{j-1}, \theta_{j+1}, \dots, \theta_k)$

How can we know the joint distribution simply using the full conditional distributions?

Hammersley-Clifford Theorem

Suppose we have a joint density $f(x, y)$. The theorem shows that the joint density can be represented in terms of the conditional densities $f(x \mid y)$ and $f(y \mid x)$:

$$f(x, y) = \frac{f(y \mid x)}{\int \frac{f(y \mid x)}{f(x \mid y)} \mathrm{d}y}$$

We can write the denominator as

$$\int \frac{f(y \mid x)}{f(x \mid y)} \mathrm{d}y = \int \frac{f(x, y)/f(x)}{f(x, y)/f(y)} \mathrm{d}y = \int \frac{f(y)}{f(x)} \mathrm{d}y = \frac{1}{f(x)}.$$

Thus, the right-hand side is

$$\frac{f(y \mid x)}{1/f(x)} = f(y \mid x) f(x) = f(x, y)$$

The theorem shows that knowledge of the conditional densities allows us to get the joint density

This works for more than two blocks of parameters

But how do we figure out the full conditionals?

Steps

Suppose we have a posterior $p(\boldsymbol{\theta} \mid \mathbf{y})$. To calculate the full conditionals for each θ_i in $\boldsymbol{\theta}$, do the following steps:

1. Write out the full posterior ignoring constants of proportionality
2. Pick a block of parameters (for example, θ_1) and drop everything that does not depend on θ_1
3. Use your knowledge of distributions to figure out what the normalizing constant is (and thus what the full conditional distribution $p(\theta_1 \mid \boldsymbol{\theta}_{-1}, \mathbf{y})$ is).
4. Repeat steps 2 and 3 for all parameter blocks.

Gibbs Algorithm

- Suppose that for some $p > 1$, the random variable $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$, where θ_i's are either uni-or-multidimensional.
- Suppose that we can sample from the full conditional densities θ_i

$$f_i(\theta_i \mid \boldsymbol{\theta}_{-i})$$

where $\boldsymbol{\theta}_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_p)$ for $i = 1, 2, \ldots, p$. The associated Gibbs sampling algorithm is given by the following transition from $\theta^{(t)}$ to $\theta^{(t+1)}$

The Gibbs Sampler
For $t = 1, 2, \ldots, T$, repeat the following steps:
Given $\boldsymbol{\theta}^{(t)} = (\theta_1^{(t)}, \ldots, \theta_p^{(t)})$, generate

1. $\theta_1^{(t+1)} \sim f_1(\theta_1 \mid \theta_2^{(t)}, \ldots, \theta_p^{(t)})$
2. $\theta_2^{(t+1)} \sim f_2(\theta_2 \mid \theta_1^{(t+1)}, \theta_3^{(t)}, \ldots, \theta_p^{(t)})$

   $\ldots\ldots$

p. $\theta_p^{(t+1)} \sim f_p(\theta_p \mid \theta_1^{(t+1)}, \ldots, \theta_{p-1}^{(t+1)})$

Example

- Suppose we have data of the number of failures y_i for each of 10 pumps in a nuclear plan, denoted $\mathbf{y} = (y_1, \dots, y_{10})$.
- We also have the times t_i at which each pump was observed, denoted $\mathbf{t} = (t_1, \dots, t_{10})$.
- We want to model the number of failures with a Poisson likelihood, where the expected number of failure λ_i differs for each pump. Since the time which we observed each pump is different, we need to scale each λ_i by its observed time t_i.
- The likelihood is $\prod_{i=1}^{10} \mathrm{Poisson}(\lambda_i t_i)$
- Prior specifications:

  $$\lambda_i \sim \mathrm{G}(\alpha, \beta)$$

  $$\beta \sim \mathrm{G}(\gamma, \delta)$$

  where we assume $\alpha = 1.8$, $\gamma = 0.01$ and $\delta = 1$.
- The total number of unknown parameters in the model is 11. (10 λ_i's s and β).

Example

Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)$. The posterior is

$$
\begin{aligned}
p(\boldsymbol{\lambda}, \beta \mid \mathbf{y}, \mathbf{t}) &\propto \left(\prod_{i=1}^{10} \frac{e^{-\lambda_i t_i}(\lambda_i t_i)^{y_i}}{y_i!} \times \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda_i^{\alpha-1} e^{-\beta\lambda_i}\right) \frac{\delta^\gamma}{\Gamma(\gamma)} \beta^{\gamma-1} e^{-\delta\beta} \\
&\propto \left(\prod_{i=1}^{10} e^{-\lambda_i t_i}(\lambda_i t_i)^{y_i} \times \beta^\alpha \lambda_i^{\alpha-1} e^{-\beta\lambda_i}\right) \times \beta^{\gamma-1} e^{-\delta\beta} \\
&= \left(\prod_{i=1}^{10} \lambda_i^{y_i+\alpha-1} e^{-(t_i+\beta)\lambda_i}\right) \beta^{10\alpha+\gamma-1} e^{-\delta\beta}
\end{aligned}
$$

Finding the full conditionals:

$$
p(\lambda_i \mid \boldsymbol{\lambda}_{-i}, \beta, \mathbf{y}, \mathbf{t}) \propto \lambda_i^{y_i+\alpha+1} e^{-(t_i+\beta)\lambda_i}
$$

$$
p(\beta \mid \boldsymbol{\lambda}, \mathbf{y}, \mathbf{t}) \propto e^{-\beta(\delta+\sum_{i=1}^{10} \lambda_i)} \beta^{10\alpha+\gamma-1}
$$

This implies that

$$
\lambda_i \mid \boldsymbol{\lambda}_{-i}, \beta, \mathbf{y}, \mathbf{t} \sim \text{Gamma}(y_i + \alpha, t_i + \beta)
$$

Beta Binomial Sampling

- Suppose we flip a coin n times and observe y heads where the probability of heads is p and the prior for p is beta(a, b).
- Hence, we know that $Y|p \sim \mathrm{Bin}(n, p)$ and $p \sim \mathrm{Beta}(\alpha, \beta)$.
- To implement Gibbs sampling for this situation, we need to identify the two conditional distributions $Y|p$ and $p|Y$.
- The joint density $\pi(y, p) \propto \mathrm{Beta}(\alpha, \beta) \times \mathrm{Bin}(n, p)$.
- The conditional $\pi(y|p)$ would be the density of y when p is fixed at some p^*. This is $\mathrm{Bin}(n, p^*)$.
- The conditional density $\pi(p|y)$ when y is fixed is the beta posterior for p, given by $\mathrm{Beta}(\alpha + y, \beta + n - y)$.
- Once we identify these distributions, sampling from them is straightforward.

Beta Binomial Sampling

- Suppose $n = 20$ and the prior for p is $\mathrm{Beta}(5, 5)$.
- Suppose that the current simulated value of p is $p^{(j)}$.
- Step 1: Simulate $Y^{(j)}$ from a $\mathrm{Bin}(20, p^{(j)})$ distribution:

```
y <- rbinom(1, size = 20, prob = p)
```

- Given the current simulated value $Y^{(j)}$, simulate $p^{(j+1)}$ from $\mathrm{Beta}(Y^{(j)} + 5, 20 - Y^{(j)} + 5)$:

```
p <- rbeta(1, y + a, n - y + b)
```

- This is one step.

Beta Binomial Sampling

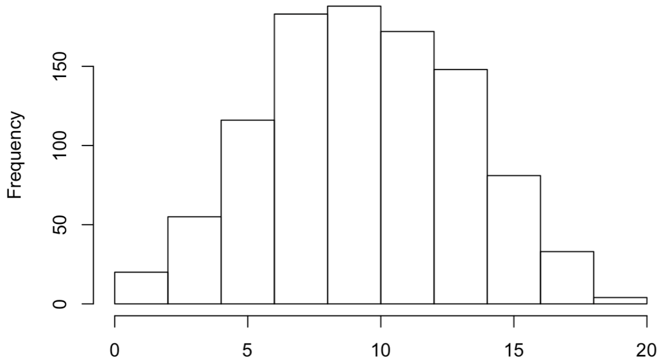
- One could use the following simple function (Albert and Hu, 2020), which starts the algorithm at $p = 0.5$, and takes 1000 iterations from conditional beta binomial distributions.

```
gibbs_betabin <- function(n, a, b, p = 0.5, iter = 1000){
  x <- matrix(0, iter, 2)
  for(k in 1:iter){
    y <- rbinom(1, size = n, prob = p)
    p <- rbeta(1, y + a, n - y + b )
    x[k, ] <- c(y, p)
  }
  x
}
```

To run the algorithm with $n = 20, \alpha = 5, \beta = 5$, and then plot a histogram of the simulated draws of Y, we run the following:

```
sp <- data.frame(gibbs_betabin(20, 5, 5))
hist(sp$X1)
```

Beta Binomial Sampling



- This empirical frequency closely resembles a beta(5, 5) density.

Background

- BUGS: Bayesian Inference Using Gibbs Sampling
- Examples used are WinBugs, OpenBUGS, and JAGS
- WinBugs
  - Developed in 1997 and only ran on Windows machines.
  - A point and click type program and your results will dump into coda files (G.S. output) which you can then read into R
  - Can be run on Unix or Mac using a windows simulator
- OpenBUGS
  - An open source version of WinBUGS; Runs on all platforms
  - Can be either menu or command line driven. Running from the command line requires BRugs package in R (NOT compatible with Unix or Linux).
  - Since BRugs currently does not work with all platforms, I prefer JAGS.

JAGS

- JAGS: Just Another Gibbs Sampler by Martyn Plummer in 2003

- It is a program for analysis of Bayesian hierarchical models using Markov Chain Monte Carlo (MCMC) simulation
- Created to make more similar to Classic Bugs as well as make improvements.
- Runs on all platforms.
- Can run either in JAGS and read coda into R or run JAGS directly from R.
- Utilizes the Adaptive Rejection Metropolis sampler.

Installation

- Download: `https://sourceforge.net/projects/mcmc-jags/`. Latest version: JAGS-4.3.0
- Mac users: Install the GNU Fortran library from the CRAN tools directory: `https://cran.r-project.org/bin/macosx/tools/`
- After the installation, start the Terminal (Mac) or Console (Windows) and type: jags. The following message indicates your installation is successful!

```
Welcome to JAGS 4.3.0 on Mon Nov 13 23:40:47 2017
JAGS is free software and comes with ABSOLUTELY NO WARRANTY
Loading module: basemod: ok
Loading module: bugs: ok
.
```

- In `R`, you need to install packages: "R2jags"

How to Run JAGS?

Running a model refers to generating samples from the posterior distribution of the model parameters. This takes place in five steps:

- Description of the model
- Definition of the data
- Set initial values and parameters to simulate
- Run model fitting
- Diagnostics

Description of the Model in JAGS

Consider a simple linear regression model

$$\begin{aligned} y_i &\sim \mathrm{N}\left\{\alpha + \beta(x_i - \bar{x}), \sigma^2\right\}, \qquad i = 1, \ldots, n \\ \alpha &\sim \mathrm{N}(0, 10^4), \\ \beta &\sim \mathrm{N}(0, 10^4), \\ \sigma^{-2} &\sim \mathrm{G}(0.1, 0, 1). \end{aligned}$$

- The model in JAGS is defined using a dialect of the BUGS language.
- It consists of a series of stochastic relations "$\sim$" and deterministic relation (arrows) "$< -$"

Define Model Using R2jags

```
linear.model.JAGS = function(){
  for(i in 1:n){
    y[i] ~ dnorm(mu[i],tau2)
    mu[i]<- alpha + beta*(x[i]-x.bar)
  }
  x.bar <- mean(x)
  alpha ~ dnorm(0.0, 1.0E-4)
  beta ~ dnorm(0.0, 1.0E-4)
  sigma2 <- 1.0/tau2
  tau2 ~ dgamma(0.1,0.1)
}
```

anything you do not define here you have
to define in the data
right now that's y_i, x_i and n

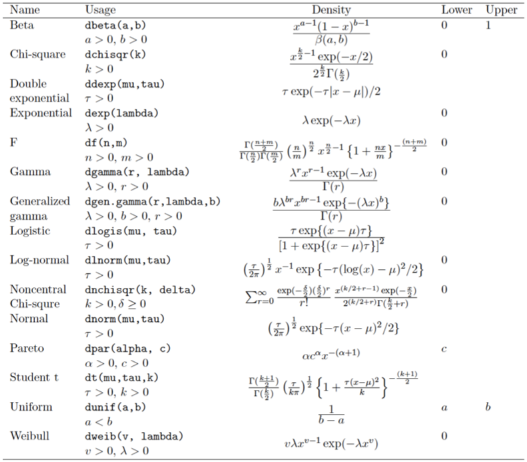
Relations

- Each relation defines a node in the model in terms of other nodes that appear on the right hand side.
- These are referred to as the parent nodes.
- Taken together, the nodes in the model (together with the parent/child relationships represented as directed edges) form a directed acyclic graph.
- The very top-level nodes in the graph, with no parents, are constant nodes, which are defined either in the model definition (e.g. 1.0E-3), or in the data file (e.g. x[1]).
- Relations can be of two types.
  - A stochastic relation ($\sim$) defines a stochastic node, representing a random variable in the model.
  - A deterministic relation ($< -$) defines a deterministic node, the value of which is determined exactly by the values of its parents.

Distributions

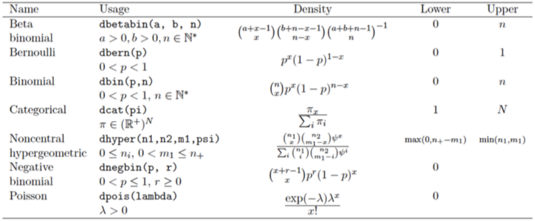
- Distributions are used to define stochastic nodes using the "$\sim$" operator.
- Some distributions have restrictions on the valid parameter values,
- If a Distribution is given invalid parameter values when evaluating the loglikelihood, it returns $-\infty$.
- When a model is initialized, all stochastic nodes are checked to ensure that the initial parameter values are valid for their distribution

Univariate Continuous Distributions



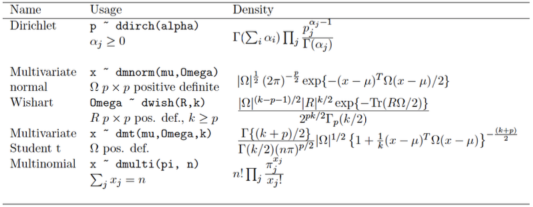
Name	Usage	Density	Lower	Upper
Beta	`dbeta(a,b)` $a > 0, b > 0$	$\frac{x^{a-1}(1-x)^{b-1}}{\beta(a,b)}$	0	1
Chi-square	`dchisqr(k)` $k > 0$	$\frac{x^{\frac{k}{2}-1}\exp(-x/2)}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})}$	0	
Double exponential	`ddexp(mu,tau)` $\tau > 0$	$\tau\exp(-\tau\|x-\mu\|)/2$		
Exponential	`dexp(lambda)` $\lambda > 0$	$\lambda\exp(-\lambda x)$	0	
F	`df(n,m)` $n > 0, m > 0$	$\frac{\Gamma(\frac{n+m}{2})}{\Gamma(\frac{n}{2})\Gamma(\frac{m}{2})}\left(\frac{n}{m}\right)^{\frac{n}{2}}x^{\frac{n}{2}-1}\left\{1+\frac{nx}{m}\right\}^{-\frac{(n+m)}{2}}$	0	
Gamma	`dgamma(r, lambda)` $\lambda > 0, r > 0$	$\frac{\lambda^r x^{r-1}\exp(-\lambda x)}{\Gamma(r)}$	0	
Generalized gamma	`dgen.gamma(r,lambda,b)` $\lambda > 0, b > 0, r > 0$	$\frac{b\lambda^{br}x^{br-1}\exp\{-(\lambda x)^b\}}{\Gamma(r)}$	0	
Logistic	`dlogis(mu, tau)` $\tau > 0$	$\frac{\tau\exp\{(x-\mu)\tau\}}{[1+\exp\{(x-\mu)\tau\}]^2}$		
Log-normal	`dlnorm(mu,tau)` $\tau > 0$	$\left(\frac{\tau}{2\pi}\right)^{\frac{1}{2}}x^{-1}\exp\left\{-\tau(\log(x)-\mu)^2/2\right\}$	0	
Noncentral Chi-squre	`dnchisqr(k, delta)` $k > 0, \delta \geq 0$	$\sum_{r=0}^{\infty}\frac{\exp(-\frac{\delta}{2})(\frac{\delta}{2})^r}{r!}\frac{x^{(k/2+r-1)}\exp(-\frac{x}{2})}{2^{(k/2+r)}\Gamma(\frac{k}{2}+r)}$	0	
Normal	`dnorm(mu,tau)` $\tau > 0$	$\left(\frac{\tau}{2\pi}\right)^{\frac{1}{2}}\exp\{-\tau(x-\mu)^2/2\}$		
Pareto	`dpar(alpha, c)` $\alpha > 0, c > 0$	$\alpha c^{\alpha}x^{-(\alpha+1)}$	c	
Student t	`dt(mu,tau,k)` $\tau > 0, k > 0$	$\frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})}\left(\frac{\tau}{k\pi}\right)^{\frac{1}{2}}\left\{1+\frac{\tau(x-\mu)^2}{k}\right\}^{-\frac{(k+1)}{2}}$		
Uniform	`dunif(a,b)` $a < b$	$\frac{1}{b-a}$	a	b
Weibull	`dweib(v, lambda)` $v > 0, \lambda > 0$	$v\lambda x^{v-1}\exp(-\lambda x^v)$	0	

Univariate Discrete Distributions



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

Multivariate Distributions

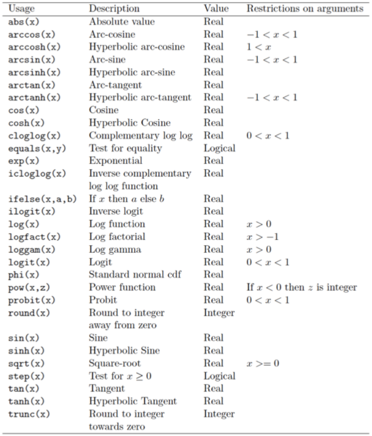


Name	Usage	Density
Dirichlet	`p ~ ddirch(alpha)` $\alpha_j \geq 0$	$\Gamma(\sum_i \alpha_i) \prod_j \frac{p_j^{\alpha_j - 1}}{\Gamma(\alpha_j)}$
Multivariate normal	`x ~ dmnorm(mu,Omega)` Ω $p \times p$ positive definite	$\lvert\Omega\rvert^{\frac{1}{2}} (2\pi)^{-\frac{p}{2}} \exp\{-(x-\mu)^T \Omega (x-\mu)/2\}$
Wishart	`Omega ~ dwish(R,k)` R $p \times p$ pos. def., $k \geq p$	$\frac{\lvert\Omega\rvert^{(k-p-1)/2} \lvert R\rvert^{k/2} \exp\{-\mathrm{Tr}(R\Omega/2)\}}{2^{pk/2}\Gamma_p(k/2)}$
Multivariate Student t	`x ~ dmt(mu,Omega,k)` Ω pos. def.	$\frac{\Gamma\{(k+p)/2\}}{\Gamma(k/2)(n\pi)^{p/2}} \lvert\Omega\rvert^{1/2} \{1 + \frac{1}{k}(x-\mu)^T \Omega (x-\mu)\}^{-\frac{(k+p)}{2}}$
Multinomial	`x ~ dmulti(pi, n)` $\sum_j x_j = n$	$n! \prod_j \frac{\pi_j^{x_j}}{x_j!}$

Functions

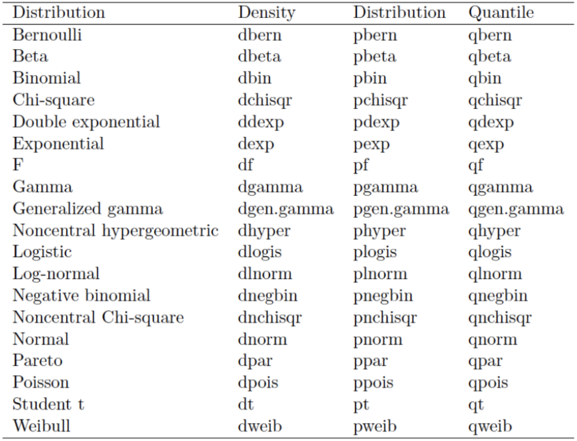
- Functions allow deterministic nodes to be defined using the "$< -$" operator.
- Most of the functions in JAGS are scalar functions taking scalar arguments.
- JAGS allows arbitrary vector- and array-valued functions, such as the matrix multiplication operator $\% * \%$ and the transpose function $t()$.

Scalar Functions



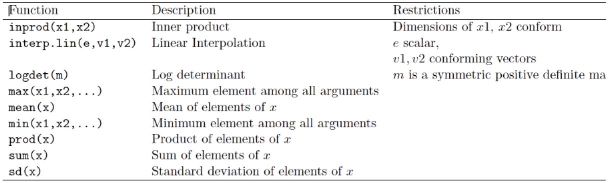
Usage	Description	Value	Restrictions on arguments
`abs(x)`	Absolute value	Real	
`arccos(x)`	Arc-cosine	Real	$-1 < x < 1$
`arccosh(x)`	Hyperbolic arc-cosine	Real	$1 < x$
`arcsin(x)`	Arc-sine	Real	$-1 < x < 1$
`arcsinh(x)`	Hyperbolic arc-sine	Real	
`arctan(x)`	Arc-tangent	Real	
`arctanh(x)`	Hyperbolic arc-tangent	Real	$-1 < x < 1$
`cos(x)`	Cosine	Real	
`cosh(x)`	Hyperbolic Cosine	Real	
`cloglog(x)`	Complementary log log	Real	$0 < x < 1$
`equals(x,y)`	Test for equality	Logical	
`exp(x)`	Exponential	Real	
`icloglog(x)`	Inverse complementary log log function	Real	
`ifelse(x,a,b)`	If x then a else b	Real	
`ilogit(x)`	Inverse logit	Real	
`log(x)`	Log function	Real	$x > 0$
`logfact(x)`	Log factorial	Real	$x > -1$
`loggam(x)`	Log gamma	Real	$x > 0$
`logit(x)`	Logit	Real	$0 < x < 1$
`phi(x)`	Standard normal cdf	Real	
`pow(x,z)`	Power function	Real	If $x < 0$ then z is integer
`probit(x)`	Probit	Real	$0 < x < 1$
`round(x)`	Round to integer away from zero	Integer	
`sin(x)`	Sine	Real	
`sinh(x)`	Hyperbolic Sine	Real	
`sqrt(x)`	Square-root	Real	$x >= 0$
`step(x)`	Test for $x \geq 0$	Logical	
`tan(x)`	Tangent	Real	
`tanh(x)`	Hyperbolic Tangent	Real	
`trunc(x)`	Round to integer towards zero	Integer	

Distribution, Density, Quantile Functions



Distribution	Density	Distribution	Quantile
Bernoulli	dbern	pbern	qbern
Beta	dbeta	pbeta	qbeta
Binomial	dbin	pbin	qbin
Chi-square	dchisqr	pchisqr	qchisqr
Double exponential	ddexp	pdexp	qdexp
Exponential	dexp	pexp	qexp
F	df	pf	qf
Gamma	dgamma	pgamma	qgamma
Generalized gamma	dgen.gamma	pgen.gamma	qgen.gamma
Noncentral hypergeometric	dhyper	phyper	qhyper
Logistic	dlogis	plogis	qlogis
Log-normal	dlnorm	plnorm	qlnorm
Negative binomial	dnegbin	pnegbin	qnegbin
Noncentral Chi-square	dnchisqr	pnchisqr	qnchisqr
Normal	dnorm	pnorm	qnorm
Pareto	dpar	ppar	qpar
Poisson	dpois	ppois	qpois
Student t	dt	pt	qt
Weibull	dweib	pweib	qweib

Scalar Value with General Input Functions



Function	Description	Restrictions
`inprod(x1,x2)`	Inner product	Dimensions of $x1$, $x2$ conform
`interp.lin(e,v1,v2)`	Linear Interpolation	e scalar, $v1, v2$ conforming vectors
`logdet(m)`	Log determinant	m is a symmetric positive definite ma
`max(x1,x2,...)`	Maximum element among all arguments	
`mean(x)`	Mean of elements of x	
`min(x1,x2,...)`	Minimum element among all arguments	
`prod(x)`	Product of elements of x	
`sum(x)`	Sum of elements of x	
`sd(x)`	Standard deviation of elements of x	

Vector of Matrix Value Functions

Usage	Description	Restrictions
`inverse(a)`	Matrix inverse	a is a symmetric positive definite matrix
`rank(v)`	Ranks of elements of v	v is a vector
`order(v)`	Ordering permutation of v	v is a vector
`sort(v)`	Elements of v in order	v is a vector
`t(a)`	Transpose	a is a matrix
`a %*% b`	Matrix multiplication	a, b conforming vector or matrices

Arrays

- Nodes defined by a relation are embedded in named arrays.
- Array names may contain letters, numbers, decimal points and underscores, but they must start with a letter.
- The node array "mu" is a vector of length "n" containing n nodes (mu[1], $\cdots$, mu[n]).
- The node array "alpha" is a scalar. Hence the array "alpha" contains a single node "alpha[1]"(The same as R). The same for "tau2", "beta" and "sigma2".
- Node arrays can be traveled with for loops.