LaplacesDemon Examples

Statisticat, LLC

Abstract

The **LaplacesDemon** package is a complete environment for Bayesian inference within R. Virtually any probability model may be specified. This vignette is a compendium of examples of how to specify different model forms.

Keywords: Bayesian, LaplacesDemon, LaplacesDemonCpp, R.

LaplacesDemon (Statisticat LLC. 2015), often referred to as LD, is an R package that is available at `https://web.archive.org/web/20150430054143/http://www.bayesian-inference.com/software`. **LaplacesDemonCpp** is an extension package that uses C++. A formal introduction to **LaplacesDemon** is provided in an accompanying vignette entitled "**LaplacesDemon** Tutorial", and an introduction to Bayesian inference is provided in the "Bayesian Inference" vignette.

The purpose of this document is to provide users of the **LaplacesDemon** package with examples of a variety of Bayesian methods. It is also a testament to the diverse applicability of **LaplacesDemon** to Bayesian inference.

To conserve space, the examples are not worked out in detail, and only the minimum of necessary materials is provided for using the various methodologies. Necessary materials include the form expressed in notation, data (which is often simulated), the `Model` function, and initial values. The provided data, model specification, and initial values may be copy/pasted into an R file and updated with the `LaplacesDemon` or (usually) `LaplaceApproximation` functions. Although many of these examples update quickly, some examples are computationally intensive.

All examples are provided in R code, but the model specification function can be in another language. A goal is to provide these example model functions in C++ as well, and some are now available at `https://web.archive.org/web/20140513065103/http://www.bayesian-inference.com/cpp/LaplacesDemonExamples.txt`.

Initial values are usually hard-coded in the examples, though the Parameter-Generating Function (PGF) is also specified. It is recommended to generate initial values with the `GIV` function according to the user-specified `PGF`.

Notation in this vignette follows these standards: Greek letters represent parameters, lower case letters represent indices, lower case bold face letters represent scalars or vectors, probabil-

ity distributions are represented with calligraphic font, upper case letters represent index limits, and upper case bold face letters represent matrices. More information on notation is available at https://web.archive.org/web/20150501205317/http://www.bayesian-inference.com/notation.

This vignette may grow over time as examples of more methods become included. Contributed examples are welcome via https://github.com/LaplacesDemonR/LaplacesDemon/issues. All accepted contributions are, of course, credited.

Contents

1. Adaptive Logistic Basis (ALB) Regression

Adaptive Logistic Basis (ALB) regression is an essentially automatic non-parametric approach to estimating the nonlinear relationship between each of multiple independent variables (IVs) and the dependent variable (DV). It is automatic because when using the suggested $K = 2J+1$ components (see below) given J IVs, the data determines the nonlinear relationships, whereas in other methods, such as with splines, the user must specify the number of knots and possibly consider placement of the knots. Knots do not exist in ALB. Both the DV and IVs should be centered and scaled.

1.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \mathbf{S}\delta$$
$$\mathbf{S}_{i,m} = \frac{\phi_{i,m}}{\sum_{m=1}^{M} \phi_{i,m}}$$
$$\phi_{i,m} = \exp(\alpha_m + \mathbf{X}_{i,1:J}\beta_{1:J,m}), \quad i = 1, \ldots, N, \quad m = 1, \ldots, M$$
$$\alpha_m \sim \mathcal{N}(0, 10), \quad m = 1, \ldots, (M-1)$$
$$\alpha_M = 0$$
$$\beta_{j,m} \sim \mathcal{N}(0, 100), \quad j = 1, \ldots, J, \quad m = 1, \ldots, (M-1)$$
$$\beta_{j,M} = 0$$
$$\delta_m \sim \mathcal{N}(\zeta, \tau^2), \quad m = 1, \ldots, M$$
$$\sigma \sim \mathcal{HC}(25)$$
$$\zeta \sim \mathcal{N}(0, 10)$$
$$\tau \sim \mathcal{HC}(25)$$

1.2. Data

```
data(demonsnacks)
N <- nrow(demonsnacks)
y <- log(demonsnacks$Calories)
X <- as.matrix(log(demonsnacks[,c(1,4,10)]+1))
J <- ncol(X)
y <- CenterScale(y)
for (j in 1:J) X[,j] <- CenterScale(X[,j])
K <- 2*J+1
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,K-1), beta=matrix(0,J,K-1),
    delta=rep(0,K), zeta=0, sigma=0, tau=0))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.delta <- grep("delta", parm.names)
pos.zeta <- grep("zeta", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.tau <- grep("tau", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(Data$K-1)
    beta <- rnorm(Data$J*(Data$K-1))
    delta <- rnorm(Data$K)
    zeta <- rnorm(1)
    sigma <- rhalfcauchy(1,5)
```

```
    tau <- rhalfcauchy(1,5)
    return(c(alpha, beta, delta, zeta, sigma, tau))
    }
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, y=y, pos.alpha=pos.alpha, pos.beta=pos.beta,
    pos.delta=pos.delta, pos.zeta=pos.zeta, pos.sigma=pos.sigma,
    pos.tau=pos.tau)
```

1.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperparameters
    zeta <- parm[Data$pos.zeta]
    parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    beta <- matrix(parm[Data$pos.beta], Data$J, Data$K-1)
    delta <- parm[Data$pos.delta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Hyperprior
    zeta.prior <- dnormv(zeta, 0, 10, log=TRUE)
    tau.prior <- dhalfcauchy(tau, 25, log=TRUE)
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 10, log=TRUE))
    beta.prior <- sum(dnormv(beta, 0, 100, log=TRUE))
    delta.prior <- sum(dnorm(delta, zeta, tau, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    phi <- cbind(exp(matrix(alpha, Data$N, Data$K-1, byrow=TRUE) +
        tcrossprod(Data$X, t(beta))),1)
    mu <- tcrossprod(phi / rowSums(phi), t(delta))
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior + delta.prior + zeta.prior
        sigma.prior + tau.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

1.4. Initial Values

```
Initial.Values <- c(rep(0,K), rep(0,J*(K-1)), rep(0,K-1), 0, 1, 1)
```

2. ANCOVA

This example is essentially the same as the two-way ANOVA (see section 4), except that a covariate $\mathbf{X}_{,3}$ has been added, and its parameter is δ.

2.1. Form

$$\mathbf{y}_i \sim \mathcal{N}(\mu_i, \sigma_1^2)$$

$$\mu_i = \alpha + \beta[\mathbf{X}_{i,1}] + \gamma[\mathbf{X}_{i,2}] + \delta\mathbf{X}_{i,2}, \quad i = 1, \ldots, N$$

$$\epsilon_i = \mathbf{y}_i - \mu_i$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\beta_j \sim \mathcal{N}(0, \sigma_2^2), \quad j = 1, \ldots, J$$

$$\beta_J = -\sum_{j=1}^{J-1} \beta_j$$

$$\gamma_k \sim \mathcal{N}(0, \sigma_3^2), \quad k = 1, \ldots, K$$

$$\gamma_K = -\sum_{k=1}^{K-1} \gamma_k$$

$$\delta \sim \mathcal{N}(0, 1000)$$

$$\sigma_m \sim \mathcal{HC}(25), \quad m = 1, \ldots, 3$$

2.2. Data

```
N <- 100
J <- 5 #Number of levels in factor (treatment) 1
K <- 3 #Number of levels in factor (treatment) 2
X <- cbind(rcat(N,rep(1/J,J)), rcat(N,rep(1/K,K)), runif(N,-2,2))
alpha <- runif(1,-1,1)
beta <- runif(J-1,-2,2)
beta <- c(beta, -sum(beta))
gamma <- runif(K-1,-2,2)
gamma <- c(gamma, -sum(gamma))
delta <- runif(1,-2,2)
y <- alpha + beta[X[,1]] + gamma[X[,2]] + delta*X[,3] + rnorm(N,0,0.1)
mon.names <- c("LP", paste("beta[",J,"]",sep=""),
    paste("gamma[",K,"]",sep=""),"s.beta","s.gamma","s.epsilon")
parm.names <- as.parm.names(list(alpha=0, beta=rep(0,J-1), gamma=rep(0,K-1),
    delta=0, sigma=rep(0,3)))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.delta <- grep("delta", parm.names)
```

```
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     alpha <- rnorm(1)
     beta <- rnorm(Data$J-1)
     gamma <- rnorm(Data$K-1)
     delta <- rnorm(1)
     sigma <- runif(3)
     return(c(alpha, beta, gamma, delta, sigma))
     }
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.alpha=pos.alpha,
     pos.beta=pos.beta, pos.gamma=pos.gamma, pos.delta=pos.delta,
     pos.sigma=pos.sigma, y=y)
```

2.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     alpha <- parm[Data$pos.alpha]
     beta <- parm[Data$pos.beta]
     beta <- c(beta, -sum(beta)) #Sum-to-zero constraint
     gamma <- parm[Data$pos.gamma]
     gamma <- c(gamma, -sum(gamma)) #Sum-to-zero constraint
     delta <- parm[Data$pos.delta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
     beta.prior <- sum(dnorm(beta, 0, sigma[2], log=TRUE))
     gamma.prior <- sum(dnorm(gamma, 0, sigma[3], log=TRUE))
     delta.prior <- dnormv(delta, 0, 1000, log=TRUE)
     sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
     ### Log-Likelihood
     mu <- alpha + beta[Data$X[,1]] + gamma[Data$X[,2]] +
          delta*Data$X[,3]
     LL <- sum(dnorm(Data$y, mu, sigma[1], log=TRUE))
     ### Variance Components
     s.beta <- sd(beta)
     s.gamma <- sd(gamma)
     s.epsilon <- sd(Data$y - mu)
     ### Log-Posterior
     LP <- LL + alpha.prior + beta.prior + gamma.prior + delta.prior +
          sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP, beta[Data$J],
          gamma[Data$K], s.beta, s.gamma, s.epsilon),
```

```
        yhat=rnorm(length(mu), mu, sigma[1]), parm=parm)
    return(Modelout)
    }
```

2.4. Initial Values

```
Initial.Values <- c(0, rep(0,(J-1)), rep(0,(K-1)), 0, rep(1,3))
```

3. ANOVA, One-Way

When $J = 2$, this is a Bayesian form of a t-test.

3.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma_1^2)$$
$$\mu_i = \alpha + \beta[\mathbf{x}_i], \quad i = 1, \ldots, N$$
$$\alpha \sim \mathcal{N}(0, 1000)$$
$$\beta_j \sim \mathcal{N}(0, \sigma_2^2), \quad j = 1, \ldots, J$$
$$\beta_J = -\sum_{j=1}^{J-1} \beta_j$$
$$\sigma_{1:2} \sim \mathcal{HC}(25)$$

3.2. Data

```
N <- 1000
J <- 3
x <- rcat(N, rep(1/J, J))
alpha <- runif(1,-1,1)
beta <- runif(J-1,-2,2)
beta <- c(beta, -sum(beta))
y <- alpha + beta[x] + rnorm(N,0,0.2)
mon.names <- c("LP",paste("beta[",J,"]",sep=""))
parm.names <- as.parm.names(list(alpha=0, beta=rep(0,J-1), sigma=rep(0,2)))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    beta <- rnorm(Data$J-1)
    sigma <- runif(2)
    return(c(alpha, beta, sigma))
    }
```

```
MyData <- list(J=J, N=N, PGF=PGF, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta,
    pos.sigma=pos.sigma, x=x, y=y)
```

3.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    beta <- parm[Data$pos.beta]
    beta <- c(beta, -sum(beta)) #Sum-to-zero constraint
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
    beta.prior <- sum(dnorm(beta, 0, sigma[2], log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    mu <- alpha + beta[Data$x]
    LL <- sum(dnorm(Data$y, mu, sigma[1], log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,beta[Data$J]),
        yhat=rnorm(length(mu), mu, sigma[1]), parm=parm)
    return(Modelout)
    }
```

3.4. Initial Values

```
Initial.Values <- c(0, rep(0,(J-1)), rep(1,2))
```

4. ANOVA, Two-Way

In this representation, σ^m are the superpopulation variance components, `s.beta` and `s.gamma` are the finite-population within-variance components of the factors or treatments, and `s.epsilon` is the finite-population between-variance component.

4.1. Form

$$\mathbf{y}_i \sim \mathcal{N}(\mu_i, \sigma_1^2)$$
$$\mu_i = \alpha + \beta[\mathbf{X}_{i,1}] + \gamma[\mathbf{X}_{i,2}], \quad i = 1, \ldots, N$$
$$\epsilon_i = \mathbf{y}_i - \mu_i$$

$$\alpha \sim \mathcal{N}(0, 1000)$$
$$\beta_j \sim \mathcal{N}(0, \sigma_2^2), \quad j = 1, \ldots, J$$
$$\beta_J = -\sum_{j=1}^{J-1} \beta_j$$
$$\gamma_k \sim \mathcal{N}(0, \sigma_3^2), \quad k = 1, \ldots, K$$
$$\gamma_K = -\sum_{k=1}^{K-1} \gamma_k$$
$$\sigma_m \sim \mathcal{HC}(25), \quad m = 1, \ldots, 3$$

4.2. Data

```
N <- 1000
J <- 5 #Number of levels in factor (treatment) 1
K <- 3 #Number of levels in factor (treatment) 2
X <- cbind(rcat(N,rep(1/J,J)), rcat(N,rep(1/K,K)))
alpha <- runif(1,-1,1)
beta <- runif(J-1,-2,2)
beta <- -sum(beta)
gamma <- runif(K-1,-2,2)
gamma <- -sum(gamma)
y <- alpha + beta[X[,1]] + gamma[X[,2]] + rnorm(N,0,0.1)
mon.names <- c("LP", paste("beta[",J,"]",sep=""),
    paste("gamma[",K,"]",sep=""), "s.beta", "s.gamma", "s.epsilon")
parm.names <- as.parm.names(list(alpha=0, beta=rep(0,J-1), gamma=rep(0,K-1),
    sigma=rep(0,3)))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    beta <- rnorm(Data$J-1)
    gamma <- rnorm(Data$K-1)
    sigma <- runif(3)
    return(c(alpha, beta, gamma, sigma))
    }
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta,
    pos.gamma=pos.gamma, pos.sigma=pos.sigma, y=y)
```

4.3. Model

```
Model <- function(parm, Data)
    {
```

```
### Parameters
alpha <- parm[Data$pos.alpha]
beta <- parm[Data$pos.beta]
beta <- c(beta, -sum(beta)) #Sum-to-zero constraint
gamma <- parm[Data$pos.gamma]
gamma <- c(gamma, -sum(gamma)) #Sum-to-zero constraint
sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
parm[Data$pos.sigma] <- sigma
### Log-Prior
alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
beta.prior <- sum(dnorm(beta, 0, sigma[2], log=TRUE))
gamma.prior <- sum(dnorm(gamma, 0, sigma[3], log=TRUE))
sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
### Log-Likelihood
mu <- alpha + beta[Data$X[,1]] + gamma[Data$X[,2]]
LL <- sum(dnorm(Data$y, mu, sigma[1], log=TRUE))
### Variance Components
s.beta <- sd(beta)
s.gamma <- sd(gamma)
s.epsilon <- sd(Data$y - mu)
### Log-Posterior
LP <- LL + alpha.prior + beta.prior + gamma.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP, beta[Data$J],
     gamma[Data$K], s.beta, s.gamma, s.epsilon),
     yhat=rnorm(length(mu), mu, sigma[1]), parm=parm)
return(Modelout)
}
```

4.4. Initial Values

```
Initial.Values <- c(0, rep(0,(J-1)), rep(0,(K-1)), rep(1,3))
```

5. Approximate Bayesian Computation (ABC)

Approximate Bayesian Computation (ABC), also called likelihood-free estimation, is not a statistical method, but a family of numerical approximation techniques in Bayesian inference. ABC is especially useful when evaluation of the likelihood, $p(\mathbf{y}|\Theta)$ is computationally prohibitive, or when suitable likelihoods are unavailable. The current example is the application of ABC in the context of linear regression. The log-likelihood is replaced with the negative sum of the distance between $\mathbf{y}$ and $\mathbf{y}^{rep}$ as the approximation of the log-likelihood. Distance reduces to the absolute difference. Although linear regression has an easily calculated likelihood, it is used as an example due to its generality. This example demonstrates how ABC may be estimated either with MCMC via the `LaplacesDemon` function or with Laplace Approximation via the `LaplaceApproximation` function. In this method, a tolerance (which is found often in ABC) does not need to be specified, and the logarithm of the unnormalized

joint posterior density is maximized, as usual. The negative and summed distance, above, may be replaced with the negative and summed distance between summaries of the data, rather than the data itself, but this has not been desirable in testing.

5.1. Form

$$\mathbf{y} = \mu + \epsilon$$

$$\mu = \mathbf{X}\beta$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

5.2. Data

```
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- c("LP","sigma")
parm.names <- as.parm.names(list(beta=rep(0,J)))
pos.beta <- grep("beta", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    return(beta)
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, y=y)
```

5.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    ### Log-Likelihood Approximation
    mu <- as.vector(tcrossprod(Data$X, t(beta)))
    epsilon <- Data$y - mu
    sigma <- sd(epsilon)
    LL <- -sum(abs(epsilon))
    ### Log-Posterior Approximation
    LP <- LL + beta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,sigma),
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
```

```
    return(Modelout)
    }
```

5.4. Initial Values

```
Initial.Values <- c(rep(0,J))
```

6. AR(p)

6.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma^2), \quad t = 1, \ldots, T$$

$$\mu_t = \alpha + \sum_{p=1}^{P} \phi_p \mathbf{y}_{t-p}, \quad t = 1, \ldots, T$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\phi_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$

$$\sigma \sim \mathcal{HC}(25)$$

6.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
L <- c(1,5,20) #Autoregressive lags
P <- length(L) #Autoregressive order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, phi=rep(0,P), sigma=0))
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    phi <- runif(Data$P,-1,1)
    sigma <- rhalfcauchy(1,5)
    return(c(alpha, phi, sigma))
    }
MyData <- list(L=L, PGF=PGF, P=P, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.phi=pos.phi,
    pos.sigma=pos.sigma, y=y)
```

6.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     alpha <- parm[Data$pos.alpha]
     phi <- parm[Data$pos.phi]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
     phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- rep(alpha, Data$T)
     for (p in 1:Data$P)
          mu[-c(1:Data$L[p])] <- mu[-c(1:Data$L[p])] +
               phi[p]*Data$y[1:(Data$T-Data$L[p])]
     LL <- sum(dnorm(Data$y[-c(1:Data$L[Data$P])], mu[-c(1:Data$L[Data$P])],
          sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + alpha.prior + phi.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }
```

6.4. Initial Values

```
Initial.Values <- c(rep(0,P+1), 1)
```

7. AR(p)-ARCH(q)

7.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma_t^2), \quad t = 1, \ldots, T$$

$$\mu_t = \alpha + \sum_{p=1}^{P} \phi_p \mathbf{y}_{t-p}, \quad t = 1, \ldots, T$$

$$\epsilon_t = \mathbf{y}_t - \mu_t$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\phi_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$

$$\sigma_t^2 = \omega + \sum_{q=1}^{Q} \theta_q \epsilon_{t-q}^2, \quad t = 2, \ldots, T$$

$$\omega \sim \mathcal{HC}(25)$$

$$\theta_q \sim \mathcal{U}(0,1), \quad q = 1, \ldots, Q$$

7.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
L.P <- c(1,5,20) #Autoregressive lags
L.Q <- c(1,2) #Volatility lags
P <- length(L.P) #Autoregressive order
Q <- length(L.Q) #Volatility order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, phi=rep(0,P), omega=0,
    theta=rep(0,Q)))
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.omega <- grep("omega", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    phi <- runif(Data$P,-1,1)
    omega <- rhalfcauchy(1,5)
    theta <- runif(Data$Q, 1e-10, 1-1e-5)
    return(c(alpha, phi, omega, theta))
    }
MyData <- list(L.P=L.P, L.Q=L.Q, PGF=PGF, P=P, Q=Q, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.phi=pos.phi,
    pos.omega=pos.omega, pos.theta=pos.theta, y=y)
```

7.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    phi <- parm[Data$pos.phi]
    omega <- interval(parm[Data$pos.omega], 1e-100, Inf)
    parm[Data$pos.omega] <- omega
    theta <- interval(parm[Data$pos.theta], 1e-10, 1-1e-5)
    parm[Data$pos.theta] <- theta
    ### Log-Prior
```

```
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
    phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
    omega.prior <- dhalfcauchy(omega, 25, log=TRUE)
    theta.prior <- sum(dunif(theta, 1e-10, 1-1e-5, log=TRUE))
    ### Log-Likelihood
    mu <- rep(alpha, Data$T)
    for (p in 1:Data$P)
        mu[-c(1:Data$L.P[p])] <- mu[-c(1:Data$L.P[p])] +
            phi[p]*Data$y[1:(Data$T-Data$L.P[p])]
    epsilon <- Data$y - mu
    sigma2 <- rep(omega, Data$T)
    for (q in 1:Data$Q)
        sigma2[-c(1:Data$L.Q[q])] <- sigma2[-c(1:Data$L.Q[q])] +
            theta[q]*epsilon[1:(Data$T-Data$L.Q[q])]^2
    LL <- sum(dnormv(Data$y[-c(1:Data$L.P[Data$P])],
        mu[-c(1:Data$L.P[Data$P])], sigma2[-c(1:Data$L.P[Data$P])],
        log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + phi.prior + omega.prior + theta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnormv(length(mu), mu, sigma2), parm=parm)
    return(Modelout)
    }
```

7.4. Initial Values

```
Initial.Values <- c(rep(0,P+1), 1, rep(0.5,Q))
```

8. AR(p)-ARCH(q)-M

8.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma_t^2), \quad t = 1, \ldots, T$$

$$\mu_t = \alpha + \sum_{p=1}^{P} \phi_p \mathbf{y}_{t-p} + \delta \sigma_{t-1}^2, \quad t = 1, \ldots, T$$

$$\epsilon_t = \mathbf{y}_t - \mu_t$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\phi_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$

$$\delta \sim \mathcal{N}(0, 1000)$$

$$\sigma_t^2 = \omega + \sum_{q=1}^{Q} \theta_q \epsilon_{t-q}^2, \quad t = 2, \ldots, T$$

$$\omega \sim \mathcal{HC}(25)$$

$$\theta_q \sim \mathcal{U}(0,1), \quad q = 1, \ldots, Q$$

8.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
L.P <- c(1,5,20) #Autoregressive lags
L.Q <- c(1,2) #Volatility lags
P <- length(L.P) #Autoregressive order
Q <- length(L.Q) #Volatility order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, phi=rep(0,P), delta=0, omega=0,
    theta=rep(0,Q)))
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.delta <- grep("delta", parm.names)
pos.omega <- grep("omega", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    phi <- runif(Data$P,-1,1)
    delta <- rnorm(1)
    omega <- rhalfcauchy(1,5)
    theta <- runif(Data$Q, 1e-10, 1-1e-5)
    return(c(alpha, phi, delta, omega, theta))
    }
MyData <- list(L.P=L.P, L.Q=L.Q, PGF=PGF, P=P, Q=Q, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.phi=pos.phi,
    pos.delta=pos.delta, pos.omega=pos.omega, pos.theta=pos.theta, y=y)
```

8.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    phi <- parm[Data$pos.phi]
    delta <- parm[Data$pos.delta]
    omega <- interval(parm[Data$pos.omega], 1e-100, Inf)
    parm[Data$pos.omega] <- omega
    theta <- interval(parm[Data$pos.theta], 1e-10, 1-1e-5)
    parm[Data$pos.theta] <- theta
    ### Log-Prior
```

```
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
    phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
    delta.prior <- dnormv(delta, 0, 1000, log=TRUE)
    omega.prior <- dhalfcauchy(omega, 25, log=TRUE)
    theta.prior <- sum(dunif(theta, 1e-10, 1-1e-5, log=TRUE))
    ### Log-Likelihood
    mu <- rep(alpha, Data$T)
    for (p in 1:Data$P)
        mu[-c(1:Data$L.P[p])] <- mu[-c(1:Data$L.P[p])] +
            phi[p]*Data$y[1:(Data$T-Data$L.P[p])]
    epsilon <- Data$y - mu
    sigma2 <- rep(omega, Data$T)
    for (q in 1:Data$Q)
        sigma2[-c(1:Data$L.Q[q])] <- sigma2[-c(1:Data$L.Q[q])] +
            theta[q]*epsilon[1:(Data$T-Data$L.Q[q])]^2
    mu <- mu + delta*sigma2
    LL <- sum(dnormv(Data$y[-c(1:Data$L.P[Data$P])],
        mu[-c(1:Data$L.P[Data$P])], sigma2[-c(1:Data$L.P[Data$P])],
        log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + phi.prior + delta.prior + omega.prior +
        theta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnormv(length(mu), mu, sigma2), parm=parm)
        return(Modelout)
    }
```

8.4. Initial Values

```
Initial.Values <- c(rep(0,P+2), 1, rep(0.5,Q))
```

9. AR(p)-GARCH(1,1)

9.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma_t^2), \quad t = 1, \ldots, T$$

$$\mu_t = \alpha + \sum_{p=1}^{P} \phi_p \mathbf{y}_{t-p}, \quad t = 1, \ldots, T$$

$$\epsilon_t = \mathbf{y}_t - \mu_t$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\phi_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$

$$\sigma_t^2 = \theta_1 + \theta_2 \epsilon_{t-1}^2 + \theta_3 \sigma_{t-1}^2$$
$$\omega \sim \mathcal{HC}(25)$$
$$\theta_k = \frac{1}{1+\exp(-\theta_k)}, \quad k = 1, \ldots, 3$$
$$\theta_k \sim \mathcal{N}(0, 1000) \in [-10, 10], \quad k = 1, \ldots, 3$$

9.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
L <- c(1,5,20) #Autoregressive lags
P <- length(L) #Autoregressive order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, phi=rep(0,P), omega=0,
    theta=rep(0,2)))
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.omega <- grep("omega", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    phi <- runif(Data$P,-1,1)
    omega <- rhalfcauchy(1,5)
    theta <- runif(2, 1e-10, 1-1e-5)
    return(c(alpha, phi, omega, theta))
    }
MyData <- list(L=L, P=P, PGF=PGF, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.phi=pos.phi,
    pos.omega=pos.omega, pos.theta=pos.theta, y=y)
```

9.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    phi <- parm[Data$pos.phi]
    omega <- interval(parm[Data$pos.omega], 1e-100, Inf)
    parm[Data$pos.omega] <- omega
    theta <- interval(parm[Data$pos.theta], 1e-10, 1-1e-5)
    if(sum(theta) >= 1) theta[2] <- 1 - 1e-5 - theta[1]
    parm[Data$pos.theta] <- theta
    ### Log-Prior
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
```

```
phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
omega.prior <- dhalfcauchy(omega, 25, log=TRUE)
theta.prior <- sum(dunif(theta, 0, 1, log=TRUE))
### Log-Likelihood
mu <- rep(alpha, Data$T)
for (p in 1:Data$P)
    mu[-c(1:Data$L[p])] <- mu[-c(1:Data$L[p])] +
        phi[p]*Data$y[1:(Data$T-Data$L[p])]
epsilon <- Data$y - mu
sigma2 <- c(omega, omega + theta[1]*epsilon[-Data$T]^2)
sigma2[-1] <- sigma2[-1] + theta[2]*sigma2[-Data$T]
LL <- sum(dnormv(Data$y[-c(1:Data$L[Data$P])],
    mu[-c(1:Data$L[Data$P])], sigma2[-c(1:Data$L[Data$P])],
        log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + phi.prior + omega.prior + theta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnormv(length(mu), mu, sigma2), parm=parm)
return(Modelout)
}
```

9.4. Initial Values

```
Initial.Values <- c(0, rep(0,P), rep(0.4,3))
```

10. AR(p)-GARCH(1,1)-M

10.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma_t^2), \quad t = 1, \ldots, T$$

$$\mu_t = \alpha + \sum_{p=1}^{P} \phi_p \mathbf{y}_{t-p} + \delta\sigma_{t-1}^2, \quad t = 1, \ldots, (T+1)$$

$$\epsilon_t = \mathbf{y}_t - \mu_t$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\phi_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$

$$\sigma_t^2 = \omega + \theta_1\epsilon_{t-1}^2 + \theta_2\sigma_{t-1}^2$$

$$\omega \sim \mathcal{HC}(25)$$

$$\theta_k \sim \mathcal{U}(0, 1), \quad k = 1, \ldots, 2$$

10.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
L <- c(1,5,20) #Autoregressive lags
P <- length(L) #Autoregressive order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, delta=0, phi=rep(0,P), omega=0,
    theta=rep(0,2)))
pos.alpha <- grep("alpha", parm.names)
pos.delta <- grep("delta", parm.names)
pos.phi <- grep("phi", parm.names)
pos.omega <- grep("omega", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    delta <- rnorm(1)
    phi <- runif(Data$P,-1,1)
    omega <- rhalfcauchy(1,5)
    theta <- runif(2, 1e-10, 1-1e-5)
    return(c(alpha, delta, phi, omega, theta))
    }
MyData <- list(L=L, P=P, PGF=PGF, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.delta=pos.delta,
    pos.phi=pos.phi, pos.omega=pos.omega, pos.theta=pos.theta, y=y)
```

10.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    delta <- parm[Data$pos.delta]
    phi <- parm[Data$pos.phi]
    omega <- interval(parm[Data$pos.omega], 1e-100, Inf)
    parm[Data$pos.omega] <- omega
    theta <- interval(parm[Data$pos.theta], 1e-10, 1-1e-5)
    if(sum(theta) >= 1) theta[2] <- 1 - 1e-5 - theta[1]
    parm[Data$pos.theta] <- theta
    ### Log-Prior
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
    delta.prior <- dnormv(delta, 0, 1000, log=TRUE)
    phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
    omega.prior <- dhalfcauchy(omega, 25, log=TRUE)
    theta.prior <- sum(dunif(theta, 0, 1, log=TRUE))
```

```
### Log-Likelihood
mu <- rep(alpha, Data$T)
for (p in 1:Data$P)
    mu[-c(1:Data$L[p])] <- mu[-c(1:Data$L[p])] +
        phi[p]*Data$y[1:(Data$T-Data$L[p])]
epsilon <- Data$y - mu
sigma2 <- c(omega, omega + theta[1]*epsilon[-Data$T]^2)
sigma2[-1] <- sigma2[-1] + theta[2]*sigma2[-Data$T]
mu <- mu + delta*sigma2
LL <- sum(dnormv(Data$y[-c(1:Data$L[Data$P])],
    mu[-c(1:Data$L[Data$P])], sigma2[-c(1:Data$L[Data$P])],
    log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + delta.prior + phi.prior + omega.prior +
    theta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnormv(length(mu), mu, sigma2), parm=parm)
return(Modelout)
}
```

10.4. Initial Values

```
Initial.Values <- c(rep(0,2), rep(0,P), rep(0.4,3))
```

11. AR(p)-TARCH(q)

11.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma_t^2), \quad t = 2, \ldots, T$$

$$\mu_t = \alpha + \phi_{p=1}^{P} \mathbf{y}_{t-p}, \quad t = (p+1), \ldots, T$$

$$\epsilon = \mathbf{y} - \mu$$

$$\delta = (\epsilon > 0) \times 1$$

$$\sigma_t^2 = \omega + \sum_{q=1}^{Q} \theta_{q,1} \delta_{t-1} \epsilon_{t-1}^2 + \theta_{q,2}(1 - \delta_{t-1}) \epsilon_{t-1}^2$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\phi_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$

$$\omega \sim \mathcal{HC}(25)$$

$$\theta_{q,j} \sim \mathcal{U}(0, 1), \quad q = 1 \ldots, Q, \quad j = 1, \ldots, 2$$

11.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
L.P <- c(1,5,20) #Autoregressive lags
L.Q <- c(1,2) #Volatility lags
P <- length(L.P) #Autoregressive order
Q <- length(L.Q) #Volatility order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, phi=rep(0,P), omega=0,
    theta=matrix(0,Q,2)))
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.omega <- grep("omega", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    phi <- runif(Data$P,-1,1)
    omega <- rhalfcauchy(1,5)
    theta <- runif(Data$Q*2, 1e-10, 1-1e-5)
    return(c(alpha, phi, omega, theta))
    }
MyData <- list(L.P=L.P, L.Q=L.Q, PGF=PGF, P=P, Q=Q, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.phi=pos.phi,
    pos.omega=pos.omega, pos.theta=pos.theta, y=y)
```

11.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    phi <- parm[Data$pos.phi]
    omega <- interval(parm[Data$pos.omega], 1e-100, Inf)
    parm[Data$pos.omega] <- omega
    theta <- matrix(interval(parm[Data$pos.theta], 1e-10, 1-1e-5), Data$Q,
        2)
    parm[Data$pos.theta] <- as.vector(theta)
    ### Log-Prior
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
    phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
    omega.prior <- dhalfcauchy(omega, 25, log=TRUE)
    theta.prior <- sum(dunif(theta, 1e-10, 1-1e-5, log=TRUE))
    ### Log-Likelihood
    mu <- rep(alpha, Data$T)
```

```
for (p in 1:Data$P)
    mu[-c(1:Data$L.P[p])] <- mu[-c(1:Data$L.P[p])] +
        phi[p]*Data$y[1:(Data$T-Data$L.P[p])]
epsilon <- Data$y - mu
delta <- (epsilon > 0) * 1
sigma2 <- rep(omega, Data$T)
for (q in 1:Data$Q)
    sigma2[-c(1:Data$L.Q[q])] <- sigma2[-c(1:Data$L.Q[q])] +
        delta[1:(Data$T-Data$L.Q[q])] * theta[q,1] *
        epsilon[1:(Data$T-Data$L.Q[q])]^2 +
        (1 - delta[1:(Data$T-Data$L.Q[q])]) * theta[q,2] *
        epsilon[1:(Data$T-Data$L.Q[q])]^2
LL <- sum(dnormv(Data$y[-c(1:Data$L.P[Data$P])],
    mu[-c(1:Data$L.P[Data$P])], sigma2[-c(1:Data$L.P[Data$P])],
    log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + phi.prior + omega.prior + theta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnormv(length(mu), mu, sigma2), parm=parm)
return(Modelout)
}
```

11.4. Initial Values

```
Initial.Values <- c(0, rep(0,P), 1, rep(0.5,Q*2))
```

12. AR(p)-TARCH(q)-M

12.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma_t^2), \quad t = 2, \ldots, T$$

$$\mu_t = \alpha + \phi_{p=1}^{P} \mathbf{y}_{t-p} + \delta_{t-1}\gamma_1\sigma_{t-1}^2 + (1 - \delta_{t-1})\gamma_2\sigma_{t-1}^2, \quad t = (p+1), \ldots, T$$

$$\epsilon = \mathbf{y} - \mu$$

$$\delta = (\epsilon > 0) \times 1$$

$$\sigma_t^2 = \omega + \sum_{q=1}^{Q} \theta_{q,1}\delta_{t-1}\epsilon_{t-1}^2 + \theta_{q,2}(1 - \delta_{t-1})\epsilon_{t-1}^2$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\gamma_k \sim \mathcal{N}(0, 1000), \quad k = 1, \ldots, 2$$

$$\phi_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$

$$\omega \sim \mathcal{HC}(25)$$

$$\theta_{q,j} \sim \mathcal{U}(0,1), \quad q = 1\ldots,Q, \quad j = 1,\ldots,2$$

12.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
L.P <- c(1,5,20) #Autoregressive lags
L.Q <- c(1,2) #Volatility lags
P <- length(L.P) #Autoregressive order
Q <- length(L.Q) #Volatility order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, gamma=rep(0,2), phi=rep(0,P),
    omega=0, theta=matrix(0,Q,2)))
pos.alpha <- grep("alpha", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.phi <- grep("phi", parm.names)
pos.omega <- grep("omega", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    gamma <- rnorm(2)
    phi <- runif(Data$P,-1,1)
    omega <- rhalfcauchy(1,5)
    theta <- runif(Data$Q*2, 1e-10, 1-1e-5)
    return(c(alpha, gamma, phi, omega, theta))
    }
MyData <- list(L.P=L.P, L.Q=L.Q, PGF=PGF, P=P, Q=Q, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.gamma=pos.gamma,
    pos.phi=pos.phi, pos.omega=pos.omega, pos.theta=pos.theta, y=y)
```

12.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    gamma <- parm[Data$pos.gamma]
    phi <- parm[Data$pos.phi]
    omega <- interval(parm[Data$pos.omega], 1e-100, Inf)
    parm[Data$pos.omega] <- omega
    theta <- matrix(interval(parm[Data$pos.theta], 1e-10, 1-1e-5), Data$Q,
        2)
    parm[Data$pos.theta] <- as.vector(theta)
    ### Log-Prior
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
```

```
    gamma.prior <- sum(dnormv(gamma, 0, 1000, log=TRUE))
    phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
    omega.prior <- dhalfcauchy(omega, 25, log=TRUE)
    theta.prior <- sum(dunif(theta, 1e-10, 1-1e-5, log=TRUE))
    ### Log-Likelihood
    mu <- rep(alpha, Data$T)
    for (p in 1:Data$P)
        mu[-c(1:Data$L.P[p])] <- mu[-c(1:Data$L.P[p])] +
            phi[p]*Data$y[1:(Data$T-Data$L.P[p])]
    epsilon <- Data$y - mu
    delta <- (epsilon > 0) * 1
    sigma2 <- rep(omega, Data$T)
    for (q in 1:Data$Q)
        sigma2[-c(1:Data$L.Q[q])] <- sigma2[-c(1:Data$L.Q[q])] +
            delta[1:(Data$T-Data$L.Q[q])] * theta[q,1] *
            epsilon[1:(Data$T-Data$L.Q[q])]^2 +
            (1 - delta[1:(Data$T-Data$L.Q[q])]) * theta[q,2] *
            epsilon[1:(Data$T-Data$L.Q[q])]^2
    mu <- mu + delta*gamma[1]*sigma2 + (1 - delta)*gamma[2]*sigma2
    LL <- sum(dnormv(Data$y[-c(1:Data$L.P[Data$P])],
        mu[-c(1:Data$L.P[Data$P])], sigma2[-c(1:Data$L.P[Data$P])],
        log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + gamma.prior + phi.prior + omega.prior +
        theta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnormv(length(mu), mu, sigma2), parm=parm)
    return(Modelout)
    }
```

12.4. Initial Values

```
Initial.Values <- c(rep(0,3), rep(0,P), 1, rep(0.5,Q*2))
```

13. Autoregressive Moving Average, ARMA(p,q)

13.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma^2), \quad t = 1, \ldots, T$$

$$\mu_t = \alpha + \sum_{p=1}^{P} \phi_p \mathbf{y}_{t-p} + \sum_{q=1}^{Q} \theta_q \epsilon_{t-q}$$

$$\epsilon_t = \mathbf{y}_t - \mu_t$$

$$\alpha \sim \mathcal{N}(0, 1000)$$
$$\phi_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$
$$\sigma \sim \mathcal{HC}(25)$$
$$\theta_q \sim \mathcal{N}(0, 1000), \quad q = 1, \ldots, Q$$

13.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
L.P <- c(1,5,20) #Autoregressive lags
L.Q <- c(1,2) #Moving average lags
P <- length(L.P) #Autoregressive order
Q <- length(L.Q) #Moving average order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, phi=rep(0,P), sigma=0,
    theta=rep(0,Q)))
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    phi <- runif(Data$P,-1,1)
    sigma <- rhalfcauchy(1,5)
    theta <- rnorm(Data$Q)
    return(c(alpha, phi, sigma, theta))
    }
MyData <- list(L.P=L.P, L.Q=L.Q, PGF=PGF, P=P, Q=Q, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.phi=pos.phi,
    pos.sigma=pos.sigma, pos.theta=pos.theta, y=y)
```

13.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    phi <- parm[Data$pos.phi]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    theta <- parm[Data$pos.theta]
    ### Log-Prior
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
```

```
phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
theta.prior <- sum(dnormv(theta, 0, 1000, log=TRUE))
### Log-Likelihood
mu <- rep(alpha, Data$T)
for (p in 1:Data$P)
    mu[-c(1:Data$L.P[p])] <- mu[-c(1:Data$L.P[p])] +
        phi[p]*Data$y[1:(Data$T-Data$L.P[p])]
epsilon <- Data$y - mu
for (q in 1:Data$Q)
    mu[-c(1:Data$L.Q[q])] <- mu[-c(1:Data$L.Q[q])] +
        theta[q]*epsilon[1:(Data$T-Data$L.Q[q])]
LL <- sum(dnorm(Data$y[-c(1:Data$L.P[Data$P])],
    mu[-c(1:Data$L.P[Data$P])], sigma, log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + phi.prior + sigma.prior + theta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnorm(length(mu), mu, sigma), parm=parm)
return(Modelout)
}
```

13.4. Initial Values

```
Initial.Values <- c(0, rep(0,P), 1, rep(0,Q))
```

14. Beta Regression

14.1. Form

$$\mathbf{y} \sim \mathcal{BETA}(a, b)$$

$$a = \mu\phi$$

$$b = (1-\mu)\phi$$

$$\mu = \Phi(\beta_1 + \beta_2\mathbf{x}), \quad \mu \in (0,1)$$

$$\beta_j \sim \mathcal{N}(0, 10), \quad j = 1, \ldots, J$$

$$\phi \sim \mathcal{HC}(25)$$

where Φ is the normal CDF.

14.2. Data

```
N <- 100
x <- runif(N)
```

```
y <- rbeta(N, (0.5-0.2*x)*3, (1-(0.5-0.2*x))*3) mon.names <- "LP"
parm.names <- c("beta[1]","beta[2]","phi")
pos.beta <- grep("beta", parm.names)
pos.phi <- grep("phi", parm.names)
PGF <- function(Data) return(c(rnormv(2,0,10), rhalfcauchy(1,5)))
MyData <- list(PGF=PGF, mon.names=mon.names, parm.names=parm.names,
    pos.beta=pos.beta, pos.phi=pos.phi, x=x, y=y)
```

14.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], 1e-100, Inf)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 10, log=TRUE))
    phi.prior <- dhalfcauchy(phi, 25, log=TRUE)
    ### Log-Likelihood
    mu <- interval(pnorm(beta[1] + beta[2]*Data$x), 0.001, 0.999,
        reflect=FALSE)
    a <- mu * phi
    b <- (1 - mu) * phi
    LL <- sum(dbeta(Data$y, a, b, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + phi.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rbeta(length(mu), a, b), parm=parm)
    return(Modelout)
    }
```

14.4. Initial Values

```
Initial.Values <- c(rep(0,2), 0.01)
```

15. Beta-Binomial

15.1. Form

$$\mathbf{y}_i \sim \mathcal{BIN}(\mathbf{n}_i, \pi_i), \quad i = 1, \ldots, N$$

$$\pi_i \sim \mathcal{BETA}(\alpha, \beta) \in [0.001, 0.999]$$

15.2. Data

```
N <- 20
n <- round(runif(N, 50, 100))
y <- round(runif(N, 1, 10))
mon.names <- "LP"
parm.names <- as.parm.names(list(pi=rep(0,N)))
PGF <- function(Data) {
    pi <- rbeta(Data$N,1,1)
    return(pi)
    }
MyData <- list(N=N, PGF=PGF, mon.names=mon.names, n=n,
    parm.names=parm.names, y=y)
```

15.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    parm[1:Data$N] <- pi <- interval(parm[1:Data$N], 0.001, 0.999)
    ### Log-Prior
    pi.prior <- sum(dbeta(pi, 1, 1, log=TRUE))
    ### Log-Likelihood
    LL <- sum(dbinom(Data$y, Data$n, pi, log=TRUE))
    ### Log-Posterior
    LP <- LL + pi.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rbinom(Data$N, Data$n, pi), parm=parm)
    return(Modelout)
    }
```

15.4. Initial Values

```
Initial.Values <- c(rep(0.5,N))
```

16. Binary Logit

16.1. Form

$$\mathbf{y} \sim \mathcal{BERN}(\eta)$$
$$\eta = \frac{1}{1 + \exp(-\mu)}$$
$$\mu = \mathbf{X}\beta$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

16.2. Data

```
data(demonsnacks)
J <- 3
y <- ifelse(demonsnacks$Calories <= 137, 0, 1)
X <- cbind(1, as.matrix(demonsnacks[,c(7,8)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J)))
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    return(beta)
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, y=y)
```

16.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[1:Data$J]
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    eta <- invlogit(mu)
    LL <- sum(dbern(Data$y, eta, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rbern(length(eta), eta), parm=parm)
    return(Modelout)
    }
```

16.4. Initial Values

```
Initial.Values <- rep(0,J)
```

17. Binary Log-Log Link Mixture

A weighted mixture of the log-log and complementary log-log link functions is used, where α is the weight. Since the log-log and complementary log-log link functions are asymmetric

(as opposed to the symmetric logit and probit link functions), it may be unknown *a priori* whether the log-log or complementary log-log will perform better.

17.1. Form

$$\mathbf{y} \sim \mathcal{BERN}(\eta)$$
$$\eta = \alpha \exp(-\exp(\mu)) + (1-\alpha)(1-\exp(-\exp(\mu)))$$
$$\mu = \mathbf{X}\beta$$
$$\alpha \sim \mathcal{U}(0,1)$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

17.2. Data

```
N <- 100
J <- 3
X <- cbind(1, matrix(rnorm(N*(J-1),N,J-1)))
alpha <- runif(1)
beta <- rnorm(J)
mu <- tcrossprod(X, t(beta))
eta <- alpha*invloglog(mu) + (1-alpha)*invcloglog(mu)
y <- rbern(N, eta)
mon.names <- c("LP","alpha")
parm.names <- as.parm.names(list(beta=rep(0,J), logit.alpha=0))
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     logit.alpha <- rnorm(1)
     return(c(beta, logit.alpha))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, y=y)
```

17.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     parm[Data$J+1] <- alpha <- interval(parm[Data$J+1], -700, 700)
     beta <- parm[1:Data$J]
     ### Log-Prior
     alpha.prior <- dunif(alpha, 0, 1, log=TRUE)
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
```

```
    eta <- alpha*invloglog(mu) + (1-alpha)*invcloglog(mu)
    LL <- sum(dbern(Data$y, eta, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,alpha),
        yhat=rbern(length(eta), eta), parm=parm)
    return(Modelout)
    }
```

17.4. Initial Values

```
Initial.Values <- c(rep(0,J), 0)
```

18. Binary Probit

18.1. Form

$$\begin{aligned}
\mathbf{y} &\sim \mathcal{BERN}(\mathbf{p}) \\
\mathbf{p} &= \phi(\mu) \\
\mu &= \mathbf{X}\beta \in [-10, 10] \\
\beta_j &\sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J
\end{aligned}$$

where ϕ is the CDF of the standard normal distribution, and J=3.

18.2. Data

```
data(demonsnacks)
J <- 3
y <- ifelse(demonsnacks$Calories <= 137, 0, 1)
X <- cbind(1, as.matrix(demonsnacks[,c(7,8)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J)))
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    return(beta)
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, y=y)
```

18.3. Model

```
Model <- function(parm, Data)
```

```
{
### Parameters
beta <- parm[1:Data$J]
### Log-Prior
beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
### Log-Likelihood
mu <- tcrossprod(Data$X, t(beta))
mu <- interval(mu, -10, 10, reflect=FALSE)
p <- pnorm(mu)
LL <- sum(dbern(Data$y, p, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
     yhat=rbern(length(p), p), parm=parm)
return(Modelout)
}
```

18.4. Initial Values

```
Initial.Values <- rep(0,J)
```

19. Binary Robit

19.1. Form

$$\mathbf{y} \sim \mathcal{BERN}(\mathbf{p})$$
$$\mathbf{p} = \mathbf{T}_\nu(\mu)$$
$$\mu = \mathbf{X}\beta \in [-10, 10]$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\nu \sim \mathcal{U}(5, 10)$$

where $\mathbf{T}_\nu$ is the CDF of the standard t-distribution with ν degrees of freedom.

19.2. Data

```
data(demonsnacks)
y <- ifelse(demonsnacks$Calories <= 137, 0, 1)
X <- cbind(1, as.matrix(demonsnacks[,c(7,8)]))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), nu=0))
pos.beta <- grep("beta", parm.names)
```

```
pos.nu <- grep("nu", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     nu <- runif(1,5,10)
     return(c(beta, nu))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.nu=pos.nu, y=y)
```

19.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     parm[Data$pos.nu] <- nu <- interval(parm[Data$pos.nu], 1e-100, 1000)
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     nu.prior <- dunif(nu, 1e-100, 1000, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     mu <- interval(mu, -10, 10, reflect=FALSE)
     p <- pst(mu, nu=nu)
     LL <- sum(dbern(Data$y, p, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + nu.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rbern(length(p), p), parm=parm)
     return(Modelout)
     }
```

19.4. Initial Values

```
Initial.Values <- c(rep(0,J), 5)
```

20. Binomial Logit

20.1. Form

$$\mathbf{y} \sim \mathcal{BIN}(\mathbf{p}, \mathbf{n})$$

$$\mathbf{p} = \frac{1}{1 + \exp(-\mu)}$$

$$\mu = \beta_1 + \beta_2 \mathbf{x}$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

20.2. Data

```
#10 Trials
exposed <- c(100,100,100,100,100,100,100,100,100,100)
deaths <- c(10,20,30,40,50,60,70,80,90,100)
dose <- c(1,2,3,4,5,6,7,8,9,10)
J <- 2 #Number of parameters
mon.names <- "LP"
parm.names <- c("beta[1]","beta[2]")
PGF <- function(Data) return(rnormv(Data$J,0,1000))
MyData <- list(J=J, PGF=PGF, n=exposed, mon.names=mon.names,
    parm.names=parm.names, x=dose, y=deaths)
```

20.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[1:Data$J]
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    ### Log-Likelihood
    mu <- beta[1] + beta[2]*Data$x
    p <- invlogit(mu)
    LL <- sum(dbinom(Data$y, Data$n, p, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rbinom(length(p), Data$n, p), parm=parm)
    return(Modelout)
    }
```

20.4. Initial Values

```
Initial.Values <- rep(0,J)
```

21. Binomial Probit

21.1. Form

$$\mathbf{y} \sim \mathcal{BIN}(\mathbf{p}, \mathbf{n})$$
$$\mathbf{p} = \phi(\mu)$$

$$\mu = \beta_1 + \beta_2 \mathbf{x} \in [-10, 10]$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

where ϕ is the CDF of the standard normal distribution, and J=2.

21.2. Data

```
#10 Trials
exposed <- c(100,100,100,100,100,100,100,100,100,100)
deaths <- c(10,20,30,40,50,60,70,80,90,100)
dose <- c(1,2,3,4,5,6,7,8,9,10)
J <- 2 #Number of parameters
mon.names <- "LP"
parm.names <- c("beta[1]","beta[2]")
PGF <- function(Data) return(rnormv(Data$J,0,1000))
MyData <- list(J=J, PGF=PGF, n=exposed, mon.names=mon.names,
    parm.names=parm.names, x=dose, y=deaths)
```

21.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[1:Data$J]
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    ### Log-Likelihood
    mu <- beta[1] + beta[2]*Data$x
    mu <- interval(mu, -10, 10, reflect=FALSE)
    p <- pnorm(mu)
    LL <- sum(dbinom(Data$y, Data$n, p, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rbinom(length(p), Data$n, p), parm=parm)
    return(Modelout)
    }
```

21.4. Initial Values

```
Initial.Values <- rep(0,J)
```

22. Binomial Robit

22.1. Form

$$\mathbf{y} \sim \mathcal{BIN}(\mathbf{p}, \mathbf{n})$$
$$\mathbf{p} = \mathbf{T}_\nu(\mu)$$
$$\mu = \beta_1 + \beta_2 \mathbf{x} \in [-10, 10]$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\nu \sim \mathcal{U}(5, 10)$$

where $\mathbf{T}_\nu$ is the CDF of the standard t-distribution with ν degrees of freedom.

22.2. Data

```
#10 Trials
exposed <- c(100,100,100,100,100,100,100,100,100,100)
deaths <- c(10,20,30,40,50,60,70,80,90,100)
dose <- c(1,2,3,4,5,6,7,8,9,10)
J <- 2 #Number of parameters
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,2), nu=0))
PGF <- function(Data) return(c(rnormv(Data$J,0,1000), runif(1,5,10)))
MyData <- list(J=J, PGF=PGF, n=exposed, mon.names=mon.names,
    parm.names=parm.names, x=dose, y=deaths)
```

22.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[1:Data$J]
    parm[Data$J+1] <- nu <- interval(parm[Data$J+1], 5, 10)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    nu.prior <- dunif(nu, 5, 10, log=TRUE)
    ### Log-Likelihood
    mu <- beta[1] + beta[2]*Data$x
    mu <- interval(mu, -10, 10, reflect=FALSE)
    p <- pst(mu, nu=nu)
    LL <- sum(dbinom(Data$y, Data$n, p, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + nu.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rbinom(length(p), Data$n, p), parm=parm)
    return(Modelout)
    }
```

22.4. Initial Values

```
Initial.Values <- c(rep(0,J), 5)
```

23. Change Point Regression

This example uses a popular variant of the stagnant water data set.

23.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \alpha + \beta_1 \mathbf{x} + \beta_2 (\mathbf{x} - \theta)[(\mathbf{x} - \theta) > 0]$$
$$\alpha \sim \mathcal{N}(0, 1000)$$
$$\beta \sim \mathcal{N}(0, 1000)$$
$$\sigma \sim \mathcal{HC}(25)$$
$$\theta \sim \mathcal{U}(-1.3, 1.1)$$

23.2. Data

```
N <- 29
y <- c(1.12, 1.12, 0.99, 1.03, 0.92, 0.90, 0.81, 0.83, 0.65, 0.67, 0.60,
    0.59, 0.51, 0.44, 0.43, 0.43, 0.33, 0.30, 0.25, 0.24, 0.13, -0.01,
    -0.13, -0.14, -0.30, -0.33, -0.46, -0.43, -0.65)
x <- c(-1.39, -1.39, -1.08, -1.08, -0.94, -0.80, -0.63, -0.63, -0.25, -0.25,
    -0.12, -0.12, 0.01, 0.11, 0.11, 0.11, 0.25, 0.25, 0.34, 0.34, 0.44,
    0.59, 0.70, 0.70, 0.85, 0.85, 0.99, 0.99, 1.19)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, beta=rep(0,2), sigma=0, theta=0))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    beta <- rnorm(2)
    sigma <- runif(1)
    theta <- runif(1)
    return(c(alpha, beta, sigma, theta))
    }
MyData <- list(N=N, PGF=PGF, mon.names=mon.names, parm.names=parm.names,
    pos.alpha=pos.alpha, pos.beta=pos.beta, pos.sigma=pos.sigma,
    pos.theta=pos.theta, x=x, y=y)
```

23.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    theta <- interval(parm[Data$pos.theta], -1.3, 1.1)
    parm[Data$pos.theta] <- theta
    ### Log-Prior
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    theta.prior <- dunif(theta, -1.3, 1.1, log=TRUE)
    ### Log-Likelihood
    mu <- alpha + beta[1]*x + beta[2]*(x - theta)*(x - theta) > 0
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior + sigma.prior + theta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

23.4. Initial Values

```
Initial.Values <- c(0.2, -0.45, 0, 0.2, 0)
```

24. Cluster Analysis, Confirmatory (CCA)

This is a parametric, model-based, cluster analysis, also called a finite mixture model or latent class cluster analysis, where the number of clusters C is fixed. When the number of clusters is unknown, exploratory cluster analysis should be used. The record-level cluster membership parameter vector, θ, is a vector of discrete parameters. Discrete parameters are not supported in all algorithms. The example below is updated with the Slice sampler.

24.1. Form

$$\mathbf{Y}_{i,j} \sim \mathcal{N}(\mu_{\theta[i],j}, \sigma^2_{\theta[i]}), \quad i = 1, \ldots, N, \quad j = 1, \ldots, J$$

$$\theta_i \sim \mathcal{CAT}(\pi_{1:C}), \quad i = 1, \ldots, N$$

$$\pi_{1:C} \sim \mathcal{D}(\alpha_{1:C})$$

$$\alpha_c = 1$$

$$\mu_{c,j} \sim \mathcal{N}(0, \nu_c^2), \quad c = 1, \ldots, C, \quad j = 1, \ldots, J$$
$$\sigma_c \sim \mathcal{HC}(25), \quad c = 1, \ldots, C$$
$$\nu_c \sim \mathcal{HC}(25), \quad c = 1, \ldots, C$$

24.2. Data

```
data(demonsnacks)
Y <- as.matrix(log(demonsnacks + 1))
N <- nrow(Y)
J <- ncol(Y)
for (j in 1:J) Y[,j] <- CenterScale(Y[,j])
C <- 3 #Number of clusters
alpha <- rep(1,C) #Prior probability of cluster proportion
mon.names <- c("LP", paste("pi[", 1:C, "]", sep=""))
parm.names <- as.parm.names(list(theta=rep(0,N), mu=matrix(0,C,J),
    nu=rep(0,C), sigma=rep(0,C)))
pos.theta <- grep("theta", parm.names)
pos.mu <- grep("mu", parm.names)
pos.nu <- grep("nu", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    theta <- rcat(Data$N, p=rep(1/Data$C, Data$C))
    mu <- rnorm(Data$J*Data$J)
    nu <- runif(Data$C)
    sigma <- runif(Data$C)
    return(c(theta, mu, nu, sigma))
    }
MyData <- list(C=C, J=J, N=N, PGF=PGF, Y=Y, alpha=alpha,
    mon.names=mon.names, parm.names=parm.names, pos.theta=pos.theta,
    pos.mu=pos.mu, pos.nu=pos.nu, pos.sigma=pos.sigma)
```

24.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    theta <- parm[Data$pos.theta]
    mu <- matrix(parm[Data$pos.mu], Data$C, Data$J)
    parm[Data$pos.nu] <- nu <- interval(parm[Data$pos.nu], 1e-100, Inf)
    pi <- rep(0, Data$C)
    tab <- table(theta)
    pi[as.numeric(names(tab))] <- as.vector(tab)
    pi <- pi / sum(pi)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
```

```
### Log-Prior
theta.prior <- sum(dcat(theta, pi, log=TRUE))
mu.prior <- sum(dnorm(mu, 0, matrix(nu, Data$C, Data$J), log=TRUE))
nu.prior <- sum(dhalfcauchy(nu, 25, log=TRUE))
pi.prior <- ddirichlet(pi, Data$alpha, log=TRUE)
sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
### Log-Likelihood
LL <- sum(dnorm(Data$Y, mu[theta,], sigma[theta], log=TRUE))
### Log-Posterior
LP <- LL + theta.prior + mu.prior + nu.prior + pi.prior +
     sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,pi),
     yhat=rnorm(prod(dim(mu[theta,])), mu[theta,], sigma[theta]),
     parm=parm)
return(Modelout)
}
```

24.4. Initial Values

```
Initial.Values <- c(rcat(N,rep(1/C,C)), rep(0,C*J), rep(1,C), rep(1,C))
```

25. Cluster Analysis, Exploratory (ECA)

This is a nonparametric, model-based, cluster analysis, also called an infinite mixture model or latent class cluster analysis, where the number of clusters C is unknown, and a Dirichlet process via truncated stick-breaking is used to estimated the number of clusters. The record-level cluster membership parameter vector, θ, is a vector of discrete parameters. Discrete parameters are not supported in all algorithms. The example below is updated with the Slice sampler.

25.1. Form

$$\mathbf{Y}_{i,j} \sim \mathcal{N}(\mu_{\theta[i],j}, \sigma^2_{\theta[i]}), \quad i = 1, \ldots, N, \quad j = 1, \ldots, J$$

$$\theta_i \sim \mathcal{CAT}(\pi_{1:C}), \quad i = 1, \ldots, N$$

$$\mu_{c,j} \sim \mathcal{N}(0, \nu_c^2), \quad c = 1, \ldots, C, \quad j = 1, \ldots, J$$

$$\sigma_c \sim \mathcal{HC}(25), \quad c = 1, \ldots, C$$

$$\pi = \text{Stick}(\delta)$$

$$\delta_c \sim \mathcal{BETA}(1, \gamma), c = 1, \ldots, (C-1)$$

$$\gamma \sim \mathcal{G}(\alpha, \beta)$$

$$\alpha \sim \mathcal{HC}(25)$$

$$\beta \sim \mathcal{HC}(25)$$

$$\nu_c \sim \mathcal{HC}(25), \quad c = 1, \ldots, C$$

25.2. Data

```
data(demonsnacks)
Y <- as.matrix(log(demonsnacks + 1))
N <- nrow(Y)
J <- ncol(Y)
for (j in 1:J) Y[,j] <- CenterScale(Y[,j])
C <- 5 #Maximum number of clusters to explore
mon.names <- c("LP", paste("pi[", 1:C, "]", sep=""))
parm.names <- as.parm.names(list(theta=rep(0,N), delta=rep(0,C-1),
    mu=matrix(0,C,J), nu=rep(0,C), sigma=rep(0,C), alpha=0, beta=0,
    gamma=0))
pos.theta <- grep("theta", parm.names)
pos.delta <- grep("delta", parm.names)
pos.mu <- grep("mu", parm.names)
pos.nu <- grep("nu", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
PGF <- function(Data) {
    mu <- rnorm(Data$C*Data$J)
    nu <- runif(Data$C)
    sigma <- runif(Data$C)
    alpha <- runif(1)
    beta <- runif(1)
    gamma <- rgamma(1, alpha, beta)
    delta <- rev(sort(rbeta(Data$C-1, 1, gamma)))
    theta <- rcat(Data$N, Stick(delta))
    return(c(theta, delta, mu, nu, sigma, alpha, beta, gamma))
    }
MyData <- list(C=C, J=J, N=N, PGF=PGF, Y=Y, mon.names=mon.names,
    parm.names=parm.names, pos.theta=pos.theta, pos.delta=pos.delta,
    pos.mu=pos.mu, pos.nu=pos.nu, pos.sigma=pos.sigma,
    pos.alpha=pos.alpha, pos.beta=pos.beta, pos.gamma=pos.gamma)
```

25.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperhyperparameters
    alpha <- interval(parm[Data$pos.alpha], 1e-100, Inf)
    parm[Data$pos.alpha] <- alpha
    beta <- interval(parm[Data$pos.beta], 1e-100, Inf)
```

```
    parm[Data$pos.beta] <- beta
    ### Hyperparameters
    delta <- interval(parm[Data$pos.delta], 1e-10, 1-1e-10)
    parm[Data$pos.delta] <- delta
    gamma <- interval(parm[Data$pos.gamma], 1e-100, Inf)
    parm[Data$pos.nu] <- nu <- interval(parm[Data$pos.nu], 1e-100, Inf)
    ### Parameters
    theta <- parm[Data$pos.theta]
    mu <- matrix(parm[Data$pos.mu], Data$C, Data$J)
    pi <- Stick(delta)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Hyperhyperprior
    alpha.prior <- dhalfcauchy(alpha, 25, log=TRUE)
    beta.prior <- dhalfcauchy(beta, 25, log=TRUE)
    ### Log-Hyperprior
    delta.prior <- dStick(delta, gamma, log=TRUE)
    gamma.prior <- dgamma(gamma, alpha, beta, log=TRUE)
    nu.prior <- sum(dhalfcauchy(nu, 25, log=TRUE))
    ### Log-Prior
    theta.prior <- sum(dcat(theta, pi, log=TRUE))
    mu.prior <- sum(dnorm(mu, 0, matrix(nu, Data$C, Data$J), log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    LL <- sum(dnorm(Data$Y, mu[theta,], sigma[theta], log=TRUE))
    ### Log-Posterior
    LP <- LL + theta.prior + delta.prior + mu.prior + nu.prior +
         alpha.prior + beta.prior + gamma.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,pi),
         yhat=rnorm(prod(dim(mu[theta,])), mu[theta,], sigma[theta]),
         parm=parm)
    return(Modelout)
    }
```

25.4. Initial Values

```
Initial.Values <- c(rcat(N, rev(sort(rStick(C-1,1)))), rep(0.5,C-1),
    rep(0,C*J), rep(1,C), rep(1,C), rep(1,3))
```

26. Conditional Autoregression (CAR), Poisson

This CAR example is a slightly modified form of example 7.3 (Model A) in Congdon (2003). The Scottish lip cancer data also appears in the WinBUGS (Spiegelhalter, Thomas, Best, and Lunn 2003) examples and is a widely analyzed example. The data $\mathbf{y}$ consists of counts for $i = 1, \ldots, 56$ counties in Scotland. A single predictor $\mathbf{x}$ is provided. The errors, ϵ, are

allowed to include spatial effects as smoothing by spatial effects from areal neighbors. The vector ϵ_μ is the mean of each area's error, and is a weighted average of errors in contiguous areas. Areal neighbors are indicated in adjacency matrix $\mathbf{A}$.

26.1. Form

$$\mathbf{y} \sim \mathcal{P}(\lambda)$$

$$\lambda = \exp(\log(\mathbf{E}) + \beta_1 + \beta_2 \mathbf{x} + \epsilon)$$

$$\epsilon \sim \mathcal{N}(\epsilon_\mu, \sigma^2)$$

$$\epsilon_{\mu[i]} = \rho \sum_{j=1}^{J} \mathbf{A}_{i,j} \epsilon_i, \quad i = 1, \ldots, N$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\rho \sim \mathcal{U}(-1, 1)$$

$$\sigma \sim \mathcal{HC}(25)$$

26.2. Data

```
N <- 56 #Number of areas
NN <- 264 #Number of adjacent areas
y <- c(9,39,11,9,15,8,26,7,6,20,13,5,3,8,17,9,2,7,9,7,16,31,11,7,19,15,7,
    10,16,11,5,3,7,8,11,9,11,8,6,4,10,8,2,6,19,3,2,3,28,6,1,1,1,1,0,0)
E <- c( 1.4,8.7,3.0,2.5,4.3,2.4,8.1,2.3,2.0,6.6,4.4,1.8,1.1,3.3,7.8,4.6,
    1.1,4.2,5.5,4.4,10.5,22.7,8.8,5.6,15.5,12.5,6.0,9.0,14.4,10.2,4.8,
    2.9,7.0,8.5,12.3,10.1,12.7,9.4,7.2,5.3,18.8,15.8,4.3,14.6,50.7,8.2,
    5.6,9.3,88.7,19.6,3.4,3.6,5.7,7.0,4.2,1.8) #Expected
x <- c(16,16,10,24,10,24,10,7,7,16,7,16,10,24,7,16,10,7,7,10,7,16,10,7,1,1,
    7,7,10,10,7,24,10,7,7,0,10,1,16,0,1,16,16,0,1,7,1,1,0,1,1,0,1,1,16,10)
A <- matrix(0, N, N)
A[1,c(5,9,11,19)] <- 1 #Area 1 is adjacent to areas 5, 9, 11, and 19
A[2,c(7,10)] <- 1 #Area 2 is adjacent to areas 7 and 10
A[3,c(6,12)] <- 1; A[4,c(18,20,28)] <- 1; A[5,c(1,11,12,13,19)] <- 1
A[6,c(3,8)] <- 1; A[7,c(2,10,13,16,17)] <- 1; A[8,6] <- 1
A[9,c(1,11,17,19,23,29)] <- 1; A[10,c(2,7,16,22)] <- 1
A[11,c(1,5,9,12)] <- 1; A[12,c(3,5,11)] <- 1; A[13,c(5,7,17,19)] <- 1
A[14,c(31,32,35)] <- 1; A[15,c(25,29,50)] <- 1
A[16,c(7,10,17,21,22,29)] <- 1; A[17,c(7,9,13,16,19,29)] <- 1
A[18,c(4,20,28,33,55,56)] <- 1; A[19,c(1,5,9,13,17)] <- 1
A[20,c(4,18,55)] <- 1; A[21,c(16,29,50)] <- 1; A[22,c(10,16)] <- 1
A[23,c(9,29,34,36,37,39)] <- 1; A[24,c(27,30,31,44,47,48,55,56)] <- 1
A[25,c(15,26,29)] <- 1; A[26,c(25,29,42,43)] <- 1
A[27,c(24,31,32,55)] <- 1; A[28,c(4,18,33,45)] <- 1
A[29,c(9,15,16,17,21,23,25,26,34,43,50)] <- 1
```

```
A[30,c(24,38,42,44,45,56)] <- 1; A[31,c(14,24,27,32,35,46,47)] <- 1
A[32,c(14,27,31,35)] <- 1; A[33,c(18,28,45,56)] <- 1
A[34,c(23,29,39,40,42,43,51,52,54)] <- 1; A[35,c(14,31,32,37,46)] <- 1
A[36,c(23,37,39,41)] <- 1; A[37,c(23,35,36,41,46)] <- 1
A[38,c(30,42,44,49,51,54)] <- 1; A[39,c(23,34,36,40,41)] <- 1
A[40,c(34,39,41,49,52)] <- 1; A[41,c(36,37,39,40,46,49,53)] <- 1
A[42,c(26,30,34,38,43,51)] <- 1; A[43,c(26,29,34,42)] <- 1
A[44,c(24,30,38,48,49)] <- 1; A[45,c(28,30,33,56)] <- 1
A[46,c(31,35,37,41,47,53)] <- 1; A[47,c(24,31,46,48,49,53)] <- 1
A[48,c(24,44,47,49)] <- 1; A[49,c(38,40,41,44,47,48,52,53,54)] <- 1
A[50,c(15,21,29)] <- 1; A[51,c(34,38,42,54)] <- 1
A[52,c(34,40,49,54)] <- 1; A[53,c(41,46,47,49)] <- 1
A[54,c(34,38,49,51,52)] <- 1; A[55,c(18,20,24,27,56)] <- 1
A[56,c(18,24,30,33,45,55)] <- 1
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,2), epsilon=rep(0,N), rho=0,
    sigma=0))
pos.beta <- grep("beta", parm.names)
pos.epsilon <- grep("epsilon", parm.names)
pos.rho <- grep("rho", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(2)
    epsilon <- rnorm(Data$N)
    rho <- runif(1,-1,1)
    sigma <- runif(1)
    return(c(beta, epsilon, rho, sigma))
    }
MyData <- list(A=A, E=E, N=N, PGF=PGF, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.epsilon=pos.epsilon,
    pos.rho=pos.rho, pos.sigma=pos.sigma, x=x, y=y)
```

26.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    epsilon <- parm[Data$pos.epsilon]
    parm[Data$pos.rho] <- rho <- interval(parm[Data$pos.rho], -1, 1)
    epsilon.mu <- rho * rowSums(epsilon * Data$A)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    epsilon.prior <- sum(dnorm(epsilon, epsilon.mu, sigma, log=TRUE))
    rho.prior <- dunif(rho, -1, 1, log=TRUE)
```

```
sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
### Log-Likelihood
lambda <- exp(log(Data$E) + beta[1] + beta[2]*Data$x/10 + epsilon)
LL <- sum(dpois(Data$y, lambda, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior + epsilon.prior + rho.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rpois(length(lambda), lambda), parm=parm)
return(Modelout)
}
```

26.4. Initial Values

```
Initial.Values <- c(rep(0,2), rep(0,N), 0, 1)
```

27. Conditional Predictive Ordinate

For a more complete introduction to the conditional predictive ordinate (CPO), see the vignette entitled "Bayesian Inference". Following is a brief guide to the applied use of CPO.

To include CPO in any model that is to be updated with MCMC, calculate and monitor the record-level inverse of the likelihood, InvL_i for records $i = 1, \ldots, N$. CPO_i is the inverse of the posterior mean of InvL_i. The inverse CPO_i, or ICPO_i, is the posterior mean of InvL_i. ICPOs larger than 40 can be considered as possible outliers, and higher than 70 as extreme values.

Here, CPO is added to the linear regression example in section 48. In this data, record 6 is a possible outlier, and record 8 is an extreme value.

27.1. Form

$$\begin{aligned}
\mathbf{y} &\sim \mathcal{N}(\mu, \sigma^2) \\
\mu &= \mathbf{X}\beta \\
\beta_j &\sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J \\
\sigma &\sim \mathcal{HC}(25)
\end{aligned}$$

27.2. Data

```
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- c("LP",as.parm.names(list(InvL=rep(0,N))))
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
```

```
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    sigma <- runif(1)
    return(c(beta, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)
```

27.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- dnorm(Data$y, mu, sigma, log=TRUE)
    InvL <- 1 / exp(LL)
    LL <- sum(LL)
    ### Log-Posterior
    LP <- LL + beta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,InvL),
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

27.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

28. Contingency Table

The two-way contingency table, matrix $\mathbf{Y}$, can easily be extended to more dimensions. Contingency table $\mathbf{Y}$ has J rows and K columns. The cell counts are fit with Poisson regression, according to intercept α, main effects β_j for each row, main effects γ_k for each column, and interaction effects $\delta_{j,k}$ for dependence effects. An omnibus (all cells) test of independence is done by estimating two models (one with δ, and one without), and a large enough Bayes Factor indicates a violation of independence when the model with δ fits better than the model

without δ. In an ANOVA-like style, main effects contrasts can be used to distinguish rows or groups of rows from each other, as well as with columns. Likewise, interaction effects contrasts can be used to test independence in groups of $\delta_{j,k}$ elements. Finally, single-cell interactions can be used to indicate violations of independence for a given cell, such as when zero is not within its 95% probability interval.

28.1. Form

$$\mathbf{Y}_{j,k} \sim \mathcal{P}(\lambda_{j,k}), \quad j = 1, \ldots, J, \quad k = 1, \ldots, K$$
$$\lambda_{j,k} = \exp(\alpha + \beta_j + \gamma_k + \delta_{j,k}), \quad j = 1, \ldots, J, \quad k = 1, \ldots, K$$
$$\alpha \sim \mathcal{N}(0, 1000)$$
$$\beta_j \sim \mathcal{N}(0, \beta_\sigma^2), \quad j = 1, \ldots, J$$
$$\beta_J = -\sum_{j=1}^{J-1} \beta_j$$
$$\beta_\sigma \sim \mathcal{HC}(25)$$
$$\gamma_k \sim \mathcal{N}(0, \gamma_\sigma^2), \quad k = 1, \ldots, K$$
$$\gamma_K = -\sum_{k=1}^{K-1} \gamma_k$$
$$\gamma_\sigma \sim \mathcal{HC}(25)$$
$$\delta_{j,k} \sim \mathcal{N}(0, \delta_\sigma^2)$$
$$\delta_{J,K} = -\sum \delta_{-J,-K}$$
$$\delta_\sigma \sim \mathcal{HC}(25)$$

28.2. Data

```
J <- 4 #Rows
K <- 4 #Columns
Y <- matrix(c(20,94,84,17,68,7,119,26,5,16,29,14,15,10,54,14), 4, 4)
rownames(Y) <- c("Black", "Blond", "Brunette", "Red")
colnames(Y) <- c("Blue", "Brown", "Green", "Hazel")
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, beta=rep(0,J-1),
    gamma=rep(0,K-1), delta=rep(0,J*K-1), b.sigma=0, g.sigma=0,
    d.sigma=0))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.delta <- grep("delta", parm.names)
pos.b.sigma <- grep("b.sigma", parm.names)
pos.g.sigma <- grep("g.sigma", parm.names)
```

```
pos.d.sigma <- grep("d.sigma", parm.names)
PGF <- function(Data) {
     alpha <- rnorm(1,log(mean(Y)),1)
     beta <- rnorm(Data$J-1)
     gamma <- rnorm(Data$K-1)
     delta <- rnorm(Data$J*Data$K-1)
     sigma <- runif(3)
     return(c(alpha, beta, gamma, delta, sigma))
     }
MyData <- list(J=J, K=K, PGF=PGF, Y=Y, mon.names=mon.names,
     parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta,
     pos.gamma=pos.gamma, pos.delta=pos.delta, pos.b.sigma=pos.b.sigma,
     pos.g.sigma=pos.g.sigma, pos.d.sigma=pos.d.sigma)
```

28.3. Model

```
Model <- function(parm, Data)
     {
     ### Hyperparameters
     beta.sigma <- interval(parm[Data$pos.b.sigma], 1e-100, Inf)
     parm[Data$pos.b.sigma] <- beta.sigma
     gamma.sigma <- interval(parm[Data$pos.g.sigma], 1e-100, Inf)
     parm[Data$pos.g.sigma] <- gamma.sigma
     delta.sigma <- interval(parm[Data$pos.d.sigma], 1e-100, Inf)
     parm[Data$pos.d.sigma] <- delta.sigma
     ### Parameters
     alpha <- parm[Data$pos.alpha]
     beta <- parm[Data$pos.beta]
     beta <- c(beta, -sum(beta))
     gamma <- parm[Data$pos.gamma]
     gamma <- c(gamma, -sum(gamma))
     delta <- parm[Data$pos.delta]
     delta <- c(delta, -sum(delta))
     delta <- matrix(delta, Data$J, Data$K)
     ### Log-Hyperprior
     beta.sigma.prior <- dhalfcauchy(beta.sigma, 25, log=TRUE)
     gamma.sigma.prior <- dhalfcauchy(gamma.sigma, 25, log=TRUE)
     delta.sigma.prior <- dhalfcauchy(delta.sigma, 25, log=TRUE)
     ### Log-Prior
     alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
     beta.prior <- sum(dnorm(beta, 0, beta.sigma, log=TRUE))
     gamma.prior <- sum(dnorm(gamma, 0, gamma.sigma, log=TRUE))
     delta.prior <- sum(dnorm(delta, 0, delta.sigma, log=TRUE))
     ### Log-Likelihood
     beta <- matrix(beta, Data$J, Data$K)
     gamma <- matrix(gamma, Data$J, Data$K, byrow=TRUE)
```

```
    lambda <- exp(alpha + beta + gamma + delta)
    LL <- sum(dpois(Data$Y, lambda, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior + beta.sigma.prior +
         gamma.prior + gamma.sigma.prior + delta.prior +
         delta.sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
         yhat=rpois(length(lambda), lambda),
         parm=parm)
    return(Modelout)
    }
```

28.4. Initial Values

```
Initial.Values <- c(log(mean(Y)), rep(0,J-1), rep(0,K-1), rep(0,J*K-1),
    rep(1,3))
```

29. Discrete Choice, Conditional Logit

29.1. Form

$$\mathbf{y}_i \sim \mathcal{CAT}(\mathbf{p}_{i,1:J}), \quad i = 1, \ldots, N, \quad j = 1, \ldots, J$$

$$\mathbf{p}_{i,j} = \frac{\phi_{i,j}}{\sum_{j=1}^{J} \phi_{i,j}}$$

$$\phi = \exp(\mu)$$

$$\mu_{i,j} = \beta_{j,1:K}\mathbf{X}_{i,1:K} + \gamma\mathbf{Z}_{i,1:C} \in [-700, 700], \quad j = 1, \ldots, (J-1)$$

$$\mu_{i,J} = \gamma\mathbf{Z}_{i,1:C}$$

$$\beta_{j,k} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, (J-1)$$

$$\gamma_c \sim \mathcal{N}(0, 1000)$$

29.2. Data

```
data(demonchoice)
y <- as.numeric(demonchoice[,1])
X <- cbind(1, as.matrix(demonchoice[,2:3]))
Z <- as.matrix(demonchoice[,4:9])
for (j in 2:ncol(X)) X[,j] <- CenterScale(X[,j])
for (j in 1:ncol(Z)) Z[,j] <- CenterScale(Z[,j])
N <- length(y) #Number of records
J <- length(unique(y)) #Number of categories in y
```

```
K <- ncol(X) #Number of individual attributes (including the intercept)
C <- ncol(Z) #Number of choice-based attributes (intercept is not included)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=matrix(0,J-1,K), gamma=rep(0,C)))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
PGF <- function(Data) {
     beta <- rnorm((Data$J-1)*Data$K)
     gamma <- rnorm(Data$C)
     return(c(beta, gamma))
     }
MyData <- list(C=C, J=J, K=K, N=N, PGF=PGF, X=X, Z=Z, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma, y=y)
```

29.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- matrix(parm[Data$pos.beta], Data$J-1, Data$K)
     gamma <- parm[Data$pos.gamma]
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     gamma.prior <- sum(dnormv(gamma, 0, 1000, log=TRUE))
     ### Log-Likelihood
     mu <- matrix(tcrossprod(gamma, Data$Z), Data$N, Data$J)
     mu[,-Data$J] <- mu[,-Data$J] + tcrossprod(Data$X, beta)
     mu <- interval(mu, -700, 700, reflect=FALSE)
     phi <- exp(mu)
     p <- phi / rowSums(phi)
     LL <- sum(dcat(Data$y, p, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + gamma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=rcat(nrow(p), p),
          parm=parm)
     return(Modelout)
     }
```

29.4. Initial Values

```
Initial.Values <- c(rep(0,(J-1)*K), rep(0,C))
```

30. Discrete Choice, Mixed Logit

30.1. Form

$$\mathbf{y}_i \sim \mathcal{CAT}(\mathbf{p}_{i,1:J}), \quad i = 1, \ldots, N$$

$$\mathbf{p}_{i,j} = \frac{\phi_{i,j}}{\sum_{j=1}^{J} \phi_{i,j}}$$

$$\phi = \exp(\mu)$$

$$\mu_{i,j} = \beta_{j,1:K,i}\mathbf{X}_{i,1:K} + \gamma\mathbf{Z}_{i,1:C} \in [-700, 700], \quad i = 1, \ldots, N, \quad j = 1, \ldots, (J-1)$$

$$\mu_{i,J} = \gamma\mathbf{Z}_{i,1:C}$$

$$\beta_{j,k,i} \sim \mathcal{N}(\zeta^{\mu}_{j,k}, \zeta^{\sigma}2_{j,k}), \quad i = 1, \ldots, N, \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

$$\gamma_c \sim \mathcal{N}(0, 1000), \quad c = 1, \ldots, C$$

$$\zeta^{\mu}_{j,k} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

$$\zeta^{\sigma}_{j,k} \sim \mathcal{HC}25), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

30.2. Data

```
data(demonchoice)
y <- as.numeric(demonchoice[,1])
X <- cbind(1, as.matrix(demonchoice[,2:3]))
Z <- as.matrix(demonchoice[,4:9])
for (j in 2:ncol(X)) X[,j] <- CenterScale(X[,j])
for (j in 1:ncol(Z)) Z[,j] <- CenterScale(Z[,j])
N <- length(y)
J <- length(unique(y)) #Number of categories in y
K <- ncol(X) #Number of predictors (including the intercept)
C <- ncol(Z) #Number of choice-based attributes (intercept is not included)
S <- diag(J-1)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=array(0, dim=c(J-1,K,N)),
    gamma=rep(0,C), zeta.mu=matrix(0,J-1,K), zeta.sigma=matrix(0,J-1,K)))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.zeta.mu <- grep("zeta.mu", parm.names)
pos.zeta.sigma <- grep("zeta.sigma", parm.names)
PGF <- function(Data) {
    zeta.mu <- matrix(rnorm((Data$J-1)*Data$K), Data$J-1, Data$K)
    zeta.sigma <- matrix(runif((Data$J-1)*Data$K), Data$J-1, Data$K)
    beta <- array(rnorm((Data$J-1)*Data$K*Data$N),
        dim=c( Data$J-1, Data$K, Data$N))
    gamma <- rnorm(Data$C)
    return(c(beta, gamma, as.vector(zeta.mu), as.vector(zeta.sigma)))
    }
MyData <- list(C=C, J=J, K=K, N=N, PGF=PGF, S=S, X=X, Z=Z,
```

```
    mon.names=mon.names, parm.names=parm.names, pos.beta=pos.beta,
    pos.gamma=pos.gamma, pos.zeta.mu=pos.zeta.mu,
    pos.zeta.sigma=pos.zeta.sigma, y=y)
```

30.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- array(parm[Data$pos.beta], dim=c(Data$J-1, Data$K, Data$N))
    gamma <- parm[Data$pos.gamma]
    zeta.mu <- matrix(parm[Data$pos.zeta.mu], Data$J-1, Data$K)
    zeta.sigma <- matrix(interval(parm[Data$pos.zeta.sigma], 1e-100, Inf),
        Data$J-1, Data$K)
    parm[Data$pos.zeta.sigma] <- as.vector(zeta.sigma)
    ### Log-Hyperprior
    zeta.mu.prior <- sum(dnormv(zeta.mu, 0, 1000, log=TRUE))
    zeta.sigma.prior <- sum(dhalfcauchy(zeta.sigma, 25, log=TRUE))
    ### Log-Prior
    beta.prior <- sum(dnorm(beta, zeta.mu, zeta.sigma, log=TRUE))
    gamma.prior <- sum(dnormv(gamma, 0, 1000, log=TRUE))
    ### Log-Likelihood
    mu <- matrix(tcrossprod(Data$Z, t(gamma)), Data$N, Data$J)
    for (j in 1:(Data$J-1)) mu[,j] <- rowSums(Data$X * t(beta[j, , ]))
    mu <- interval(mu, -700, 700, reflect=FALSE)
    phi <- exp(mu)
    p <- phi / rowSums(phi)
    LL <- sum(dcat(Data$y, p, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + gamma.prior + zeta.mu.prior + zeta.sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=rcat(nrow(p), p),
        parm=parm)
    return(Modelout)
    }
```

30.4. Initial Values

```
Initial.Values <- c(rep(0,(J-1)*K*N), rep(0,C), rep(0,(J-1)*K),
    rep(1,(J-1)*K))
```

31. Discrete Choice, Multinomial Probit

31.1. Form

$$\mathbf{W}_{i,1:(J-1)} \sim \mathcal{N}_{J-1}(\mu_{i,1:(J-1)}, \Sigma), \quad i = 1, \ldots, N$$

$$\mathbf{W}_{i,j} \in \begin{cases} [0,10] & \text{if } \mathbf{y}_i = j \\ [-10,0] \end{cases}$$

$$\mu_{1:N,j} = \mathbf{X}\beta_{j,1:K} + \mathbf{Z}\gamma$$

$$\Sigma = \mathbf{U}^T\mathbf{U}$$

$$\beta_{j,k} \sim \mathcal{N}(0, 10), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

$$\gamma_c \sim \mathcal{N}(0, 10), \quad c = 1, \ldots, C$$

$$\mathbf{U}_{j,k} \sim \mathcal{N}(0, 1), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, (J-1), \quad j \geq k, \quad j \neq k = 1$$

31.2. Data

```
data(demonchoice)
y <- as.numeric(demonchoice[,1])
X <- cbind(1, as.matrix(demonchoice[,2:3]))
Z <- as.matrix(demonchoice[,4:9])
for (j in 2:ncol(X)) X[,j] <- CenterScale(X[,j])
for (j in 1:ncol(Z)) Z[,j] <- CenterScale(Z[,j])
J <- length(unique(y)) #Number of categories in y
K <- ncol(X) #Number of predictors (including the intercept)
C <- ncol(Z) #Number of choice-based attributes (intercept is not included)
S <- diag(J-1)
U <- matrix(NA,J-1,J-1)
U[upper.tri(U, diag=TRUE)] <- 0
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=matrix(0,(J-1),K), gamma=rep(0,C),
    U=U, W=matrix(0,N,J-1)))
parm.names <- parm.names[-which(parm.names == "U[1,1]")]
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.U <- grep("U", parm.names)
pos.W <- grep("W", parm.names)
PGF <- function(Data) {
    beta <- rnorm((Data$J-1)*Data$K)
    gamma <- rnorm(Data$C)
    U <- rnorm((Data$J-2) + (factorial(Data$J-1) /
        (factorial(Data$J-1-2)*factorial(2))),0,1)
    W <- matrix(runif(Data$N*(Data$J-1),-10,0), Data$N, Data$J-1)
    Y <- as.indicator.matrix(Data$y)
    W <- ifelse(Y[,-Data$J] == 1, abs(W), W)
    return(c(beta, gamma, U, as.vector(W)))}
MyData <- list(C=C, J=J, K=K, N=N, PGF=PGF, S=S, X=X, Z=Z,
```

```
    mon.names=mon.names, parm.names=parm.names, pos.beta=pos.beta,
    pos.gamma=pos.gamma, pos.U=pos.U, pos.W=pos.W, y=y)
```

31.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- matrix(parm[Data$pos.beta], Data$J-1, Data$K)
    gamma <- parm[Data$pos.gamma]
    u <- c(0, parm[Data$pos.U])
    U <- diag(Data$J-1)
    U[upper.tri(U, diag=TRUE)] <- u
    diag(U) <- exp(diag(U))
    Sigma <- t(U) %*% U
    Sigma[1,] <- Sigma[,1] <- U[1,]
    W <- matrix(parm[Data$pos.W], Data$N, Data$J-1)
    Y <- as.indicator.matrix(Data$y)
    temp <- which(Y[,-c(Data$J)] == 1)
    W[temp] <- interval(W[temp], 0, 10)
    temp <- which(Y[,-c(Data$J)] == 0)
    W[temp] <- interval(W[temp], -10, 0)
    parm[Data$pos.W] <- as.vector(W)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 10, log=TRUE))
    gamma.prior <- sum(dnormv(gamma, 0, 10, log=TRUE))
    U.prior <- sum(dnorm(u[-1], 0, 1, log=TRUE))
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, beta) +
        as.vector(tcrossprod(Data$Z, t(gamma)))
    #eta <- exp(cbind(mu,0))
    #p <- eta / rowSums(eta)
    LL <- sum(dmvn(W, mu, Sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + gamma.prior + U.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=max.col(cbind(rmvn(nrow(mu), mu, Sigma),0)), parm=parm)
    return(Modelout)
    }
```

31.4. Initial Values

```
Initial.Values <- GIV(Model, MyData, PGF=TRUE)
```

32. Distributed Lag, Koyck

This example applies Koyck or geometric distributed lags to $k = 1, \ldots, K$ discrete events in covariate $\mathbf{x}$, transforming the covariate into a N x K matrix $\mathbf{X}$ and creates a N x K lag matrix $\mathbf{L}$.

32.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mu_t = \alpha + \phi \mathbf{y}_{t-1} + \sum_{k=1}^{K} \mathbf{X}_{t,k} \beta \lambda^{\mathbf{L}[t,k]}, \quad k = 1, \ldots, K, \quad t = 2, \ldots, T$$

$$\mu_1 = \alpha + \sum_{k=1}^{K} \mathbf{X}_{1,k} \beta \lambda^{\mathbf{L}[1,k]}, \quad k = 1, \ldots, K$$

$$\alpha \sim \mathcal{N}(0, 1000)$$

$$\beta \sim \mathcal{N}(0, 1000)$$

$$\lambda \sim \mathcal{U}(0, 1)$$

$$\phi \sim \mathcal{N}(0, 1000)$$

$$\sigma \sim \mathcal{HC}(25)$$

32.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
x <- (y > 0.01)*1 #Made-up distributed lag IV
T <- length(y)
K <- length(which(x != 0))
L <- X <- matrix(0, T, K)
for (i in 1:K) {
     X[which(x != 0)[i]:T,i] <- x[which(x != 0)[i]]
     L[(which(x != 0)[i]):T,i] <- 0:(T - which(x != 0)[i])}
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, beta=0, lambda=0, phi=0, sigma=0))
PGF <- function(Data) {
     alpha <- rnorm(1)
     beta <- rnorm(1)
     lambda <- runif(1)
     phi <- rnorm(1)
     sigma <- runif(1)
     return(c(alpha, beta, lambda, phi, sigma))
     }
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
```

```
pos.lambda <- grep("lambda", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
MyData <- list(L=L, PGF=PGF, T=T, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta,
    pos.lambda=pos.lambda, pos.phi=pos.phi, pos.sigma=pos.sigma, y=y)
```

32.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    beta <- parm[Data$pos.beta]
    lambda <- interval(parm[Data$pos.lambda], 0, 1)
    parm[Data$pos.lambda] <- lambda
    phi <- parm[Data$pos.phi]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
    beta.prior <- dnormv(beta, 0, 1000, log=TRUE)
    lambda.prior <- dunif(lambda, 0, 1, log=TRUE)
    phi.prior <- dnormv(phi, 0, 1000, log=TRUE)
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- c(alpha, alpha + phi*Data$y[-Data$T]) +
        rowSums(Data$X * beta * lambda^Data$L)
    LL <- sum(dnorm(Data$y[-1], mu[-1], sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior + lambda.prior + phi.prior +
        sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

32.4. Initial Values

```
Initial.Values <- c(rep(0,2), 0.5, 0, 1)
```

33. Dynamic Sparse Factor Model (DSFM)

33.1. Form

$$\mathbf{Y}_{t,j} \sim \mathcal{N}(\alpha t, j + \mathbf{F}_{t,1:P}\Lambda_{1:P,1:j,t}, \Sigma^2_{t,j}), \quad t = 1, \dots, T, \quad j = 1, \dots, J$$

$$\alpha_{t,j} \sim \mathcal{N}(\alpha^{\mu}_j + \alpha^{\phi}_j(\alpha_{t-1,j} - \alpha^m u_j), \alpha^{\sigma} 2_j)$$

$$\mathbf{F}_{t,1:P} \sim \mathcal{N}_P(\mathbf{F}^{\phi}\mathbf{F}_{t-1,1:P}, \mathbf{f}^{\Sigma}_{t,1:P})$$

$$\mathbf{f}^{\Sigma}_{t,1:P} = t(\mathbf{f}^{\mathbf{U}}_{1:P,1:P,t})\mathbf{f}^{\mathbf{U}}_{1:P,1:P,t}$$

$$\mathbf{f}^{\mathbf{U}}_{p,q,t} \sim \mathcal{N}(\mathbf{f}^{\mathbf{u}_{\mu}}_{p,q} + \mathbf{f}^{\mathbf{u}_{\phi}}_{p,q}(\mathbf{f}^{\mathbf{U}}_{p,q,t-1} - \mathbf{f}^{\mathbf{u}_{\mu}}_{p,q}), \mathbf{f}^{\mathbf{u}^2}_{\sigma,q})$$

$$\Lambda_{p,j,t} \sim \mathcal{N}(\lambda^{\mu}_{p,j} + \lambda^{\phi}_{p,j}(\Lambda_{p,j,t-1} - \lambda^m u_{p,j}), \lambda^{\sigma} 2_{p,j})$$

$$\Sigma_{t,j} = \exp(\log(\Sigma_{t,j}))$$

$$log(\Sigma_{t,j}) \sim \mathcal{N}(\sigma^{\mu}_j + \sigma^{\phi}_j(log(\Sigma_{t-1,j}) - \sigma^m u_j), \sigma^{\sigma} 2_j)$$

$$\alpha^0_j \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\alpha^{\mu}_j \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\frac{\alpha^{\phi}_j + 1}{2} \sim \mathcal{BETA}(20, 1.5), \quad j = 1, \dots, J$$

$$\alpha^{\sigma}_j \sim \mathcal{HC}(5), \quad j = 1, \dots, J$$

$$\mathbf{f}^0_j \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\frac{\mathbf{f}^{\phi}_j + 1}{2} \sim \mathcal{BETA}(1, 1), \quad j = 1, \dots, J$$

$$\mathbf{f}^{\mathbf{u}0}_j \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\mathbf{f}^{\mathbf{u}\mu}_j \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\frac{\mathbf{f}^{\mathbf{u}\phi}_j + 1}{2} \sim \mathcal{BETA}(20, 1.5), \quad j = 1, \dots, J$$

$$\mathbf{f}^{\mathbf{u}\sigma}_j \sim \mathcal{HC}(1), \quad j = 1, \dots, J$$

$$\lambda^0_j \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\lambda^d_j \sim \mathcal{U}(0, |\lambda^{\mu}_j| + 3\sqrt{\frac{\lambda^{\sigma}_j}{1 - \lambda^{\phi}_j \lambda^{\phi}_j}}), \quad j = 1, \dots, J$$

$$\lambda^{\mu}_j \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\frac{\lambda^{\phi}_j + 1}{2} \sim \mathcal{BETA}(20, 1.5), \quad j = 1, \dots, J$$

$$\lambda^{\sigma}_j \sim \mathcal{HC}(1), \quad j = 1, \dots, J$$

$$\log(\sigma^0_j) \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\log(\sigma^{\mu}_j) \sim \mathcal{N}(0, 1), \quad j = 1, \dots, J$$

$$\frac{\log(\sigma_j^{\phi})+1}{2} \sim \mathcal{BETA}(20, 1.5), \quad j = 1, \ldots, J$$

$$\log(\sigma_j^{\sigma}) \sim \mathcal{HC}(1), \quad j = 1, \ldots, J$$

33.2. Data

```
data(demonfx)
Y.orig <- demonfx
Y <- log(as.matrix(Y.orig[1:20,1:3]))
Y.means <- colMeans(Y)
Y <- Y - matrix(Y.means, nrow(Y), ncol(Y), byrow=TRUE) #Center
Y.scales <- sqrt(.colVars(Y))
Y <- Y / matrix(Y.scales, nrow(Y), ncol(Y), byrow=TRUE) #Scale
T <- nrow(Y) #Number of time-periods
J <- ncol(Y) #Number of time-series
P <- 2 #Number of dynamic factors
mon.names <- "LP"
U1 <- matrix(NA,P,P); U2 <- matrix(NA,P,J)
U1[upper.tri(U1, diag=TRUE)] <- 0; U2[upper.tri(U2)] <- 0
Lambda <- array(NA, dim=c(P,J,T))
U <- array(NA, dim=c(P,P,T))
for (t in 1:T) {
    U[ , , t] <- U1
    Lambda[ , , t] <- U2}
parm.names <- as.parm.names(list(alpha0=rep(0,J), Alpha=matrix(0,T,J),
    alpha.mu=rep(0,J), alpha.phi=rep(0,J), alpha.sigma=rep(0,J),
    f0=rep(0,P), F=matrix(0,T,P), f.phi=rep(0,P), f.u0=U1, f.U=U,
    f.u.mu=U1, f.u.phi=U1, f.u.sigma=U1, lambda0=U2, Lambda=Lambda,
    lambda.d=U2, lambda.mu=U2, lambda.phi=U2, lambda.sigma=U2,
    lsigma0=rep(0,J), lSigma=matrix(0,T,J),
    lsigma.mu=rep(0,J), lsigma.phi=rep(0,J), lsigma.sigma=rep(0,J)))
pos.alpha0 <- grep("alpha0", parm.names)
pos.Alpha <- grep("Alpha", parm.names)
pos.alpha.mu <- grep("alpha.mu", parm.names)
pos.alpha.phi <- grep("alpha.phi", parm.names)
pos.alpha.sigma <- grep("alpha.sigma", parm.names)
pos.f0 <- grep("f0", parm.names)
pos.F <- grep("F", parm.names)
pos.f.phi <- grep("f.phi", parm.names)
pos.f.u0 <- grep("f.u0", parm.names)
pos.f.U <- grep("f.U", parm.names)
pos.f.u.mu <- grep("f.u.mu", parm.names)
pos.f.u.phi <- grep("f.u.phi", parm.names)
pos.f.u.sigma <- grep("f.u.sigma", parm.names)
pos.lambda0 <- grep("lambda0", parm.names)
pos.Lambda <- grep("Lambda", parm.names)
```

```
pos.lambda.d <- grep("lambda.d", parm.names)
pos.lambda.mu <- grep("lambda.mu", parm.names)
pos.lambda.phi <- grep("lambda.phi", parm.names)
pos.lambda.sigma <- grep("lambda.sigma", parm.names)
pos.lsigma0 <- grep("lsigma0", parm.names)
pos.lSigma <- grep("lSigma", parm.names)
pos.lsigma.mu <- grep("lsigma.mu", parm.names)
pos.lsigma.phi <- grep("lsigma.phi", parm.names)
pos.lsigma.sigma <- grep("lsigma.sigma", parm.names)
PGF <- function(Data) {
    alpha0 <- rnorm(Data$J)
    Alpha <- rnorm(Data$T*Data$J)
    alpha.mu <- rnorm(Data$J)
    alpha.phi <- rbeta(Data$J, 20, 1.5) * 2 - 1
    alpha.sigma <- runif(Data$J)
    f0 <- rnorm(Data$P)
    F <- rnorm(Data$T*Data$P)
    f.phi <- rbeta(Data$P, 1, 1) * 2 - 1
    f.u0 <- rnorm(length(Data$pos.f.u0))
    f.U <- rnorm(length(Data$pos.f.U))
    f.u.mu <- rnorm(length(Data$pos.f.u.mu))
    f.u.phi <- runif(length(Data$pos.f.u.phi))
    f.u.sigma <- runif(length(Data$pos.f.u.sigma))
    lambda0 <- rnorm(length(Data$pos.lambda0))
    Lambda <- rnorm(length(Data$pos.Lambda))
    lambda.mu <- rnorm(length(Data$pos.lambda.mu))
    lambda.phi <- rbeta(length(Data$pos.lambda.phi), 20, 1.5)
    lambda.sigma <- runif(length(Data$pos.lambda.sigma))
    lambda.d <- runif(length(Data$pos.lambda.d), 0, abs(lambda.mu) +
        3*sqrt(lambda.sigma/(1-lambda.phi^2)))
    lsigma0 <- rnorm(Data$J)
    lSigma <- rnorm(Data$T*Data$J)
    lsigma.mu <- rnorm(Data$J)
    lsigma.phi <- rbeta(Data$J, 20, 1.5) * 2 - 1
    lsigma.sigma <- runif(Data$J)
    return(c(alpha0, Alpha, alpha.mu, alpha.phi, alpha.sigma, f0, F,
        f.phi, f.u0, f.U, f.u.mu, f.u.phi, f.u.sigma, lambda0, Lambda,
        lambda.d, lambda.mu, lambda.phi, lambda.sigma, lsigma0, lSigma,
        lsigma.mu, lsigma.phi, lsigma.sigma))
    }
MyData <- list(J=J, P=P, PGF=PGF, T=T, Y=Y, mon.names=mon.names,
    parm.names=parm.names, pos.alpha0=pos.alpha0, pos.Alpha=pos.Alpha,
    pos.alpha.mu=pos.alpha.mu, pos.alpha.phi=pos.alpha.phi,
    pos.alpha.sigma=pos.alpha.sigma, pos.f0=pos.f0, pos.F=pos.F,
    pos.f.phi=pos.f.phi, pos.f.u0=pos.f.u0, pos.f.U=pos.f.U,
    pos.f.u.mu=pos.f.u.mu, pos.f.u.phi=pos.f.u.phi,
    pos.f.u.sigma=pos.f.u.sigma, pos.lambda0=pos.lambda0,
```

```
    pos.Lambda=pos.Lambda, pos.lambda.d=pos.lambda.d,
    pos.lambda.mu=pos.lambda.mu, pos.lambda.phi=pos.lambda.phi,
    pos.lambda.sigma=pos.lambda.sigma, pos.lsigma0=pos.lsigma0,
    pos.lSigma=pos.lSigma, pos.lsigma.mu=pos.lsigma.mu,
    pos.lsigma.phi=pos.lsigma.phi, pos.lsigma.sigma=pos.lsigma.sigma)
```

33.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha0 <- parm[Data$pos.alpha0]
    Alpha <- matrix(parm[Data$pos.Alpha], Data$T, Data$J)
    alpha.mu <- parm[Data$pos.alpha.mu]
    alpha.phi <- interval(parm[Data$pos.alpha.phi], -1, 1)
    parm[Data$pos.alpha.phi] <- alpha.phi
    alpha.sigma <- interval(parm[Data$pos.alpha.sigma], 1e-100, Inf)
    parm[Data$pos.alpha.sigma] <- alpha.sigma
    f0 <- parm[Data$pos.f0]
    F <- matrix(parm[Data$pos.F], Data$T, Data$P)
    f.phi <- interval(parm[Data$pos.f.phi], -1, 1)
    parm[Data$pos.f.phi] <- f.phi
    f.u0 <- parm[Data$pos.f.u0]
    f.U <- parm[Data$pos.f.U]
    f.u.mu <- parm[Data$pos.f.u.mu]
    f.u.phi <- interval(parm[Data$pos.f.u.phi], -1, 1)
    parm[Data$pos.f.u.phi] <- f.u.phi
    f.u.sigma <- interval(parm[Data$pos.f.u.sigma], 1e-100, Inf)
    parm[Data$pos.f.u.sigma] <- f.u.sigma
    lambda0 <- parm[Data$pos.lambda0]
    Lambda <- parm[Data$pos.Lambda]
    lambda.mu <- parm[Data$pos.lambda.mu]
    lambda.phi <- interval(parm[Data$pos.lambda.phi], -1, 1)
    parm[Data$pos.lambda.phi] <- lambda.phi
    lambda.sigma <- interval(parm[Data$pos.lambda.sigma], 1e-100, Inf)
    parm[Data$pos.lambda.sigma] <- lambda.sigma
    lambda.d <- parm[Data$pos.lambda.d]
    for (i in 1:length(lambda.d))
        lambda.d[i] <- interval(lambda.d[i], 0, abs(lambda.mu[i]) +
        3*sqrt(lambda.sigma[i]/(1-lambda.phi[i]^2)))
    parm[Data$pos.lambda.d] <- lambda.d
    lsigma0 <- parm[Data$pos.lsigma0]
    lSigma <- matrix(parm[Data$pos.lSigma], Data$T, Data$J)
    lsigma.mu <- parm[Data$pos.lsigma.mu]
    lsigma.phi <- interval(parm[Data$pos.lsigma.phi], -1, 1)
    parm[Data$pos.lsigma.phi] <- lsigma.phi
```

```
    lsigma.sigma <- interval(parm[Data$pos.lsigma.sigma], 1e-100, Inf)
    parm[Data$pos.lsigma.sigma] <- lsigma.sigma
    ### Log-Prior
    alpha0.prior <- sum(dnorm(alpha0, 0, 1, log=TRUE))
    Alpha.prior <- sum(dnorm(Alpha,
        matrix(alpha.mu, Data$T, Data$J, byrow=TRUE) +
        matrix(alpha.phi, Data$T, Data$J, byrow=TRUE) *
        (rbind(alpha0, Alpha[-Data$T,]) -
        matrix(alpha.mu, Data$T, Data$J, byrow=TRUE)),
        matrix(alpha.sigma, Data$T, Data$J, byrow=TRUE), log=TRUE))
    alpha.mu.prior <- sum(dnorm(alpha.mu, 0, 1, log=TRUE))
    alpha.phi.prior <- sum(dbeta((alpha.phi + 1) / 2, 20, 1.5, log=TRUE))
    alpha.sigma.prior <- sum(dhalfcauchy(alpha.sigma, 5, log=TRUE))
    f0.prior <- sum(dnorm(f0, 0, 1, log=TRUE))
    f.phi.prior <- sum(dbeta((f.phi + 1) / 2, 1, 1, log=TRUE))
    f.u0.prior <- sum(dnorm(f.u0, 0, 1, log=TRUE))
    f.U.prior <- sum(dnorm(matrix(f.U, nrow=Data$T, byrow=TRUE),
        matrix(f.u.mu, Data$T, Data$P*(Data$P-1)/2+Data$P, byrow=TRUE) +
        matrix(f.u.phi, Data$T, Data$P*(Data$P-1)/2+Data$P, byrow=TRUE) *
        (rbind(f.u0, matrix(f.U, nrow=Data$T, byrow=TRUE)[-Data$T,]) -
        matrix(f.u.mu, Data$T, Data$P*(Data$P-1)/2+Data$P, byrow=TRUE)),
        matrix(f.u.sigma, Data$T, Data$P*(Data$P-1)/2+Data$P, byrow=TRUE),
        log=TRUE))
    f.u.mu.prior <- sum(dnorm(f.u.mu, 0, 1, log=TRUE))
    f.u.phi.prior <- sum(dbeta((f.u.phi + 1) / 2, 20, 1.5, log=TRUE))
    f.u.sigma.prior <- sum(dhalfcauchy(f.u.sigma, 1, log=TRUE))
    lambda0.prior <- sum(dnorm(lambda0, 0, 1, log=TRUE))
    Lambda.prior <- sum(dnorm(matrix(Lambda, nrow=Data$T, byrow=TRUE),
        matrix(lambda.mu, Data$T, length(lambda.mu), byrow=TRUE) +
        (rbind(lambda0, matrix(Lambda, nrow=Data$T, byrow=TRUE))[-(Data$T+1),]
-
        matrix(lambda.mu, Data$T, length(lambda.mu), byrow=TRUE)),
        matrix(lambda.sigma, Data$T, length(lambda.sigma), byrow=TRUE),
        log=TRUE))
    lambda.d.prior <- sum(dunif(lambda.d, 0, abs(lambda.mu) +
        3*sqrt(lambda.sigma/(1-lambda.phi^2)), log=TRUE))
    lambda.mu.prior <- sum(dnorm(lambda.mu, 0, 1, log=TRUE))
    lambda.phi.prior <- sum(dbeta((lambda.phi + 1) / 2, 20, 1.5, log=TRUE))
    lambda.sigma.prior <- sum(dhalfcauchy(lambda.sigma, 1, log=TRUE))
    lsigma0.prior <- sum(dnorm(lsigma0, 0, 1, log=TRUE))
    lSigma.prior <- sum(dnorm(lSigma,
        matrix(lsigma.mu, Data$T, Data$J, byrow=TRUE) +
        matrix(lsigma.phi, Data$T, Data$J, byrow=TRUE) *
        (rbind(lsigma0, lSigma[-Data$T,]) -
        matrix(lsigma.mu, Data$T, Data$J, byrow=TRUE)),
        matrix(lsigma.sigma, Data$T, Data$J, byrow=TRUE), log=TRUE))
    lsigma.mu.prior <- sum(dnorm(lsigma.mu, 0, 1, log=TRUE))
```

```
    lsigma.phi.prior <- sum(dbeta((lsigma.phi + 1) / 2, 20, 1.5, log=TRUE))
    lsigma.sigma.prior <- sum(dhalfcauchy(lsigma.sigma, 1, log=TRUE))
    ### Log-Likelihood
    LL <- 0; Yhat <- Data$Y; F.prior <- 0
    for (t in 1:Data$T) {
        f.U.temp <- matrix(0, Data$P, Data$P)
        f.U.temp[upper.tri(f.U.temp, diag=TRUE)] <- matrix(f.U, nrow=Data$T,
            byrow=TRUE)[t,]
        diag(f.U.temp) <- exp(diag(f.U.temp))
        f.Sigma <- as.symmetric.matrix(t(f.U.temp) %*% f.U.temp)
        F.prior <- F.prior + dmvn(F[t,], rbind(f0, F)[t,] %*% diag(f.phi),
            f.Sigma, log=TRUE)
        Lambda.temp <- matrix(1, Data$P, Data$J)
        Lambda.temp[lower.tri(Lambda.temp)] <- 0
        Lambda.temp[upper.tri(Lambda.temp)] <- matrix(Lambda,
            nrow=Data$T, byrow=TRUE)[t,]*(abs(matrix(Lambda,
            nrow=Data$T, byrow=TRUE)[t,]) > lambda.d)
        mu <- Alpha[t,] + F[t,] %*% Lambda.temp
        LL <- LL + sum(dnorm(Data$Y[t,], mu, exp(lSigma[t,]), log=TRUE))
        Yhat[t,] <- rnorm(Data$J, mu, exp(lSigma[t,])) #Fitted
        }
    ### Log-Posterior
    LP <- LL + alpha0.prior + Alpha.prior + alpha.mu.prior +
        alpha.phi.prior + alpha.sigma.prior + f0.prior + F.prior +
        f.phi.prior + f.u0.prior + f.U.prior + f.u.mu.prior +
        f.u.phi.prior + f.u.sigma.prior + lambda0.prior +
        Lambda.prior + lambda.d.prior + lambda.mu.prior +
        lambda.phi.prior + lambda.sigma.prior + lsigma0.prior +
        lSigma.prior + lsigma.mu.prior + lsigma.phi.prior +
        lsigma.sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=Yhat, parm=parm)
    return(Modelout)
    }
```

33.4. Initial Values

```
Initial.Values <- c(rnorm(J), rnorm(T*J), rnorm(J), runif(J), runif(J),
    rnorm(P), rnorm(T*P), rbeta(P,1,1)*2-1, rnorm(P*(P-1)/2+P),
    rnorm((P*(P-1)/2+P)*T), rnorm(P*(P-1)/2+P),
    rbeta(P*(P-1)/2+P,1,1)*2-1, runif(P*(P-1)/2+P),
    rnorm(P*J-P-P*(P-1)/2), rnorm((P*J-P-P*(P-1)/2)*T),
    runif(P*J-P-P*(P-1)/2,0,1e-3), rnorm(P*J-P-P*(P-1)/2),
    rbeta(P*J-P-P*(P-1)/2,20,1.5)*2-1, runif(P*J-P-P*(P-1)/2),
    rnorm(J), rnorm(T*J), rnorm(J), rbeta(J,20,1.5)*2-1, runif(J))
```

34. Exponential Smoothing

34.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu_t = \alpha \mathbf{y}_{t-1} + (1-\alpha)\mu_{t-1}, \quad t = 2, \ldots, T$$
$$\alpha \sim \mathcal{U}(0,1)$$
$$\sigma \sim \mathcal{HC}$$

34.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
mon.names <- "LP"
parm.names <- c("alpha","sigma")
PGF <- function(Data) {
    alpha <- runif(1)
    sigma <- runif(1)
    return(c(alpha, sigma))
    }
MyData <- list(PGF=PGF, T=T, mon.names=mon.names, parm.names=parm.names,
    y=y)
```

34.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    parm[1] <- alpha <- interval(parm[1], 0, 1)
    parm[2] <- sigma <- interval(parm[2], 1e-100, Inf)
    ### Log-Prior
    alpha.prior <- dunif(alpha, 0, 1, log=TRUE)
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- y
    mu[-1] <- alpha*Data$y[-1]
    mu[-1] <- mu[-1] + (1 - alpha) * mu[-Data$T]
    LL <- sum(dnorm(Data$y[-1], mu[-Data$T], sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
```

```
}
```

34.4. Initial Values

```
Initial.Values <- c(0.5, 1)
```

35. Factor Analysis, Approximate Dynamic

The Approximate Dynamic Factor Analysis (ADFA) model has many names, including the approximate factor model and approximate dynamic factor model. An ADFA is a Dynamic Factor Analysis (DFA) in which the factor scores of the dynamic factors are approximated with principal components. This is a combination of principal components and common factor analysis, in which the factor loadings of common factors are estimated from the data and factor scores are estimated from principal components. This is a two-stage model: principal components are estimated in the first stage and a decision is made regarding how many principal components to retain, and ADFA is estimated in the second stage. For more information on DFA, see section 33.

35.1. Form

$$\mathbf{Y}_{t,j} \sim \mathcal{N}(\mu_{t,j}, \sigma_j^2), \quad t = 2, \ldots, T, \quad j = 1, \ldots, J$$
$$\mu_{t,} = \mathbf{F}_{t-1,}\Lambda$$
$$\Lambda_{p,j} \sim \mathcal{N}(0, 1), \quad p = 1, \ldots, P, \quad j = 1, \ldots, J$$
$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, J$$

35.2. Data

```
data(demonfx)
Y.orig <- as.matrix(demonfx)
Y <- diff(log(Y.orig[1:100,]))
Y.scales <- sqrt(.colVars(Y))
Y <- Y / matrix(Y.scales, nrow(Y), ncol(Y), byrow=TRUE)
T <- nrow(Y) #Number of time-periods
J <- ncol(Y) #Number of time-series
P <- 7 #Number of approximate factors
PCA <- prcomp(Y, scale=TRUE)
F <- PCA$x[,1:P]
mon.names <- "LP"
parm.names <- as.parm.names(list(Lambda=matrix(0,P,J), sigma=rep(0,J)))
pos.Lambda <- grep("Lambda", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    Lambda <- rnorm(Data$P*Data$J)
```

```
    sigma <- runif(Data$J)
    return(c(Lambda, sigma))
    }
MyData <- list(F=F, J=J, P=P, PGF=PGF, T=T, Y=Y, mon.names=mon.names,
    parm.names=parm.names, pos.Lambda=pos.Lambda, pos.sigma=pos.sigma)
```

35.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    Lambda <- matrix(parm[Data$pos.Lambda], Data$P, Data$J)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    Lambda.prior <- sum(dnorm(Lambda, 0, 1, log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    mu <- tcrossprod(rbind(rep(0,Data$P), F[-Data$T,]), t(Lambda))
    Sigma <- matrix(sigma, Data$T, Data$J, byrow=TRUE)
    LL <- sum(dnorm(Data$Y[-1,], mu[-1,], Sigma[-1,], log=TRUE))
    ### Log-Posterior
    LP <- LL + Lambda.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(prod(dim(mu)), mu, Sigma), parm=parm)
    return(Modelout)
    }
```

35.4. Initial Values

```
Initial.Values <- c(rep(0,P*J), rep(1,J))
```

36. Factor Analysis, Confirmatory

Factor scores are in matrix $\mathbf{F}$, factor loadings for each variable are in vector λ, and $\mathbf{f}$ is a vector that indicates which variable loads on which factor.

36.1. Form

$$\mathbf{Y}_{i,m} \sim \mathcal{N}(\mu_{i,m}, \sigma_m^2), \quad i = 1, \ldots, N, \quad m = 1, \ldots, M$$
$$\mu = \mathbf{F}_{1:N,\mathbf{f}} \lambda^T$$
$$\mathbf{F}_{i,1:P} \sim \mathcal{N}_P(0, \Omega^{-1}), \quad i = 1, \ldots, N$$
$$\lambda_m \sim \mathcal{N}(0, 1), \quad m = 1, \ldots, M$$

$$\sigma_m \sim \mathcal{HC}(25), \quad m = 1, \ldots, M$$

$$\Omega \sim \mathcal{W}_N(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_P$$

36.2. Data

```
data(swiss)
Y <- cbind(swiss$Agriculture, swiss$Examination, swiss$Education,
    swiss$Catholic, swiss$Infant.Mortality)
M <- ncol(Y) #Number of variables
N <- nrow(Y) #Number of records
P <- 3 #Number of factors
f <- c(1,3,2,2,1) #Indicator f for the factor for each variable m
for (m in 1:M) Y[,m] <- CenterScale(Y[,m])
S <- diag(P)
mon.names <- "LP"
parm.names <- as.parm.names(list(F=matrix(0,N,P), lambda=rep(0,M),
    U=diag(P), sigma=rep(0,M)), uppertri=c(0,0,1,0))
pos.F <- grep("F", parm.names)
pos.lambda <- grep("lambda", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    U <- rwishartc(Data$N, Data$S)
    F <- as.vector(rmvnpc(Data$N, rep(0,Data$P), U))
    U <- U[upper.tri(U, diag=TRUE)]
    lambda <- rnorm(Data$M)
    sigma <- runif(Data$M)
    return(c(F, lambda, U, sigma))
    }
MyData <- list(M=M, N=N, P=P, PGF=PGF, S=S, Y=Y, f=f, mon.names=mon.names,
    parm.names=parm.names, pos.F=pos.F, pos.lambda=pos.lambda,
    pos.sigma=pos.sigma)
```

36.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    lambda <- parm[Data$pos.lambda]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    F <- matrix(parm[Data$pos.F], Data$N, Data$P)
    U <- as.parm.matrix(U, Data$P, parm, Data, chol=TRUE)
    diag(U) <- exp(diag(U))
    ### Log-Prior
    lambda.prior <- sum(dnorm(lambda, 0, 1, log=TRUE))
```

```
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    U.prior <- dwishartc(U, Data$N, Data$S, log=TRUE)
    F.prior <- sum(dmvnpc(F, rep(0,Data$P), U, log=TRUE))
    ### Log-Likelihood
    mu <- F[,Data$f] * matrix(lambda, Data$N, Data$M, byrow=TRUE)
    Sigma <- matrix(sigma, Data$N, Data$M, byrow=TRUE)
    LL <- sum(dnorm(Data$Y, mu, Sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + lambda.prior + sigma.prior + F.prior + U.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(prod(dim(mu)), mu, Sigma), parm=parm)
    return(Modelout)
    }
```

36.4. Initial Values

```
Initial.Values <- c(rep(0,N*P), rep(0,M), upper.triangle(S, diag=TRUE),
    rep(1,M))
```

37. Factor Analysis, Exploratory

Factor scores are in matrix $\mathbf{F}$ and factor loadings are in matrix Λ.

37.1. Form

$$\mathbf{Y}_{i,m} \sim \mathcal{N}(\mu_{i,m}, \sigma_m^2), \quad i = 1, \ldots, N, \quad m = 1, \ldots, M$$

$$\mu = \mathbf{F}\Lambda$$

$$\mathbf{F}_{i,1:P} \sim \mathcal{N}_P(0, \Omega^{-1}), \quad i = 1, \ldots, N$$

$$\Lambda_{p,m} \sim \mathcal{N}(0, 1), \quad p = 1, \ldots, P, \quad m = (p+1), \ldots, M$$

$$\Omega \sim \mathcal{W}_N(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_P$$

$$\sigma_m \sim \mathcal{HC}(25), \quad m = 1, \ldots, M$$

37.2. Data

```
data(USJudgeRatings)
Y <- as.matrix(USJudgeRatings)
for (m in 1:M) Y[,m] <- CenterScale(Y[,m])
M <- ncol(Y) #Number of variables
N <- nrow(Y) #Number of records
P <- 3 #Number of factors
Lambda <- matrix(NA, P, M)
Lambda[upper.tri(Lambda)] <- 0
```

```
S <- diag(P)
mon.names <- "LP"
parm.names <- as.parm.names(list(F=matrix(0,N,P), Lambda=Lambda, U=S,
    sigma=rep(0,M)), uppertri=c(0,0,1,0))
pos.F <- grep("F", parm.names)
pos.Lambda <- grep("Lambda", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    U <- rwishartc(Data$N, Data$S)
    F <- as.vector(rmvnpc(Data$N, rep(0,Data$P), U))
    Lambda <- rnorm(Data$P*Data$M-Data$P-Data$P*(Data$P-1)/2,0,1)
    sigma <- runif(Data$M)
    return(c(F, Lambda, U[upper.tri(U, diag=TRUE)], sigma))
    }
MyData <- list(M=M, N=N, P=P, PGF=PGF, S=S, Y=Y, mon.names=mon.names,
    parm.names=parm.names, pos.F=pos.F, pos.Lambda=pos.Lambda,
    pos.sigma=pos.sigma)
```

37.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    F <- matrix(parm[Data$pos.F], Data$N, Data$P)
    lambda <- parm[Data$pos.Lambda]
    U <- as.parm.matrix(U, Data$P, parm, Data, chol=TRUE)
    diag(U) <- exp(diag(U))
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    F.prior <- sum(dmvnpc(F, rep(0,Data$P), U, log=TRUE))
    Lambda.prior <- sum(dnorm(lambda, 0, 1, log=TRUE))
    U.prior <- dwishartc(U, Data$N, Data$S, log=TRUE)
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    Lambda <- matrix(1, Data$P, Data$M)
    Lambda[lower.tri(Lambda)] <- 0
    Lambda[upper.tri(Lambda)] <- lambda
        mu <- tcrossprod(F, t(Lambda))
    Sigma <- matrix(sigma, Data$N, Data$M, byrow=TRUE)
    LL <- sum(dnorm(Data$Y, mu, Sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + F.prior + Lambda.prior + U.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnorm(prod(dim(mu)), mu, Sigma), parm=parm)
    return(Modelout)
```

```
}
```

37.4. Initial Values

```
Initial.Values <- c(rep(0,N*P), rep(0,P*M-P-P*(P-1)/2), rep(0,P*(P-1)/2+P),
    rep(1,M))
```

38. Factor Analysis, Exploratory Ordinal

This exploratory ordinal factor analysis (EOFA) model form is also suitable for collaborative filtering.

38.1. Form

$$\mathbf{Y}_{i,m} \sim \mathcal{CAT}(\mathbf{P}_{i,m,1:K}), \quad i = 1, \ldots, N, \quad m = 1, \ldots, M$$
$$\mathbf{P}_{,,K} = 1 - Q_{,,(K-1)}$$
$$\mathbf{P}_{,,k} = |Q_{,,k} - Q_{,,(k-1)}|, \quad k = 2, \ldots, (K-1)$$
$$\mathbf{P}_{,,1} = Q_{,,1}$$
$$Q = \phi(\mu)$$
$$\mu_{,,k} = \alpha_k - \mathbf{F}\Lambda, \quad k = 1, \ldots, (K-1)$$
$$\mathbf{F}_{i,1:P} \sim \mathcal{N}_P(0, \Omega^{-1}), \quad i = 1, \ldots, N$$
$$\gamma_p = 0, \quad p = 1, \ldots, P$$
$$\Lambda_{p,m} \sim \mathcal{N}(0, 1), \quad p = 1, \ldots, P, \quad m = (p+1), \ldots, M$$
$$\Omega \sim \mathcal{W}_N(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_P$$
$$\alpha_k \sim \mathcal{N}(0, 1) \in [(k-1), k] \in [-5, 5], \quad k = 2, \ldots, (K-1)$$

38.2. Data

```
M <- 10 #Number of variables
N <- 20 #Number of records
K <- 3 #Number of discrete values
P <- 3 #Number of factors
alpha <- sort(rnorm(K-1))
Lambda <- matrix(1, P, M)
Lambda[lower.tri(Lambda)] <- 0
Lambda[upper.tri(Lambda)] <- rnorm(P*M-P-P*(P-1)/2)
Omega <- runif(P)
F <- rmvnp(N, rep(0,P), Omega)
mu <- aperm(array(alpha, dim=c(K-1, M, N)), perm=c(3,2,1))
mu <- mu - array(tcrossprod(F, t(Lambda)), dim=c(N, M, K-1))
```

```
Pr <- Q <- pnorm(mu)
Pr[ , , -1] <- abs(Q[ , , -1] - Q[ , , -(K-1)])
Pr <- array(Pr, dim=c(N, M, K))
Pr[ , , K] <- 1 - Q[ , , (K-1)]
dim(Pr) <- c(N*M, K)
Y <- matrix(rcat(nrow(Pr), Pr), N, M) #Make sure Y has all values
S <- diag(P)
Lambda <- matrix(0, P, M)
Lambda[lower.tri(Lambda, diag=TRUE)] <- NA
mon.names <- "LP"
parm.names <- as.parm.names(list(F=matrix(0,N,P), Omega=rep(0,P),
    Lambda=Lambda, alpha=rep(0,K-1)))
pos.F <- grep("F", parm.names)
pos.Omega <- grep("Omega", parm.names)
pos.Lambda <- grep("Lambda", parm.names)
pos.alpha <- grep("alpha", parm.names)
PGF <- function(Data) {
    Omega <- runif(Data$P)
    F <- as.vector(rmvnp(Data$N, rep(0,Data$P), diag(Omega)))
    Lambda <- rnorm(Data$P*Data$M-Data$P-Data$P*(Data$P-1)/2)
    alpha <- sort(rnorm(Data$K-1))
    return(c(F, Omega, Lambda, alpha))
    }
MyData <- list(K=K, M=M, N=N, P=P, PGF=PGF, S=S, Y=Y,
    mon.names=mon.names, parm.names=parm.names, pos.F=pos.F,
    pos.Omega=pos.Omega, pos.Lambda=pos.Lambda, pos.alpha=pos.alpha)
```

38.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    F <- matrix(parm[Data$pos.F], Data$N, Data$P)
    Omega <- interval(parm[Data$pos.Omega], 1e-100, Inf)
    parm[Data$pos.Omega] <- Omega
    lambda <- parm[Data$pos.Lambda]
    alpha <- sort(interval(parm[Data$pos.alpha], -5, 5))
    parm[Data$pos.alpha] <- alpha
    ### Log-Prior
    F.prior <- sum(dmvnp(F, rep(0,Data$P), diag(Omega), log=TRUE))
    Omega.prior <- dwishart(diag(Omega), Data$N, Data$S, log=TRUE)
    Lambda.prior <- sum(dnorm(lambda, 0, 1, log=TRUE))
    alpha.prior <- sum(dnormv(alpha, 0, 10, log=TRUE))
    ### Log-Likelihood
    Lambda <- matrix(1, Data$P, Data$M)
    Lambda[lower.tri(Lambda)] <- 0
```

```
    Lambda[upper.tri(Lambda)] <- lambda
    mu <- aperm(array(alpha, dim=c(Data$K-1, Data$M, Data$N)),
         perm=c(3,2,1))
    mu <- mu - array(tcrossprod(F, t(Lambda)),
         dim=c(Data$N, Data$M, Data$K-1))
    P <- Q <- pnorm(mu)
    P[ , , -1] <- abs(Q[ , , -1] - Q[ , , -(Data$K-1)])
    P <- array(P, dim=c(Data$N, Data$M, Data$K))
    P[ , , Data$K] <- 1 - Q[ , , (Data$K-1)]
    y <- as.vector(Data$Y)
    dim(P) <- c(Data$N*Data$M, Data$K)
    LL <- sum(dcat(y, P, log=TRUE))
    ### Log-Posterior
    LP <- LL + F.prior + Omega.prior + Lambda.prior + alpha.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
         yhat=matrix(rcat(nrow(P), P), Data$N, Data$M), parm=parm)
    return(Modelout)
    }
```

38.4. Initial Values

```
Initial.Values <- c(rep(0,N*P), rep(0,P), rep(0,P*M-P-P*(P-1)/2),
    seq(from=-1, to=1, len=K-1))
```

39. Factor Regression

This example of factor regression is constrained to the case where the number of factors is equal to the number of independent variables (IVs) less the intercept. The purpose of this form of factor regression is to orthogonalize the IVs with respect to $\mathbf{y}$, rather than variable reduction. This method is the combination of confirmatory factor analysis in section 36 and linear regression in section 48.

39.1. Form

$$\mathbf{y} \sim \mathcal{N}(\nu, \sigma^2_{J+1})$$

$$\nu = \mathbf{F}\beta$$

$$\mu_{i,1} = 1$$

$$\mu_{i,j+1} = \mu_{i,j}, \quad j = 1, \ldots, J$$

$$\mathbf{X}_{i,j} \sim \mathcal{N}(\mu_{i,j}, \sigma^2_j), \quad i = 1, \ldots, N, \quad j = 2, \ldots, J$$

$$\mu_{i,j} = \lambda_j \mathbf{F}_{i,j}, \quad i = 1, \ldots, N, \quad j = 2, \ldots, J$$

$$\mathbf{F}_{i,1:J} \sim \mathcal{N}_{J-1}(0, \Omega^{-1}), \quad i = 1, \ldots, N$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\lambda_j \sim \mathcal{N}(0,1), \quad j = 1,\ldots,(J-1)$$
$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1,\ldots,(J+1)$$
$$\Omega \sim \mathcal{W}_N(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_J$$

39.2. Data

```
data(demonsnacks)
N <- nrow(demonsnacks)
y <- log(demonsnacks$Calories)
X <- as.matrix(log(demonsnacks[,c(1,4,10)]+1))
J <- ncol(X)
for (j in 1:J) X[,j] <- CenterScale(X[,j])
S <- diag(J)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J+1), lambda=rep(0,J),
    sigma=rep(0,J+1), F=matrix(0,N,J), Omega=rep(0,J)))
pos.beta <- grep("beta", parm.names)
pos.lambda <- grep("lambda", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.F <- grep("F", parm.names)
pos.Omega <- grep("Omega", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J+1)
    lambda <- rnorm(Data$J)
    sigma <- runif(Data$J+1)
    Omega <- runif(Data$J)
    F <- as.vector(rmvnp(Data$N, rep(0,Data$J), diag(Omega)))
    return(c(beta, lambda, sigma, F, Omega))
    }
MyData <- list(J=J, N=N, PGF=PGF, S=S, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.lambda=pos.lambda,
    pos.sigma=pos.sigma, pos.F=pos.F, pos.Omega=pos.Omega, y=y)
```

39.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    lambda <- parm[Data$pos.lambda]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    F <- matrix(Data$pos.F], Data$N, Data$J)
    Omega <- interval(parm[Data$pos.Omega], 1e-100, Inf)
    parm[Data$pos.Omega] <- Omega
```

```
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    lambda.prior <- sum(dnorm(lambda, 0, 1, log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    F.prior <- sum(dmvnp(F, rep(0,Data$J), diag(Omega), log=TRUE))
    Omega.prior <- dwishart(diag(Omega), Data$N, Data$S, log=TRUE)
    ### Log-Likelihood
    mu <- F * matrix(lambda, Data$N, Data$J, byrow=TRUE)
    nu <- tcrossprod(cbind(1,F), t(beta))
    LL <- sum(dnorm(Data$X, mu, matrix(sigma[1:Data$J], Data$N, Data$J,
        byrow=TRUE), log=TRUE))
    LL <- LL + dnorm(Data$y, nu, sigma[Data$J+1], log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + lambda.prior + sigma.prior + F.prior
        Omega.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(Data$N, nu, sigma[Data$J+1]), parm=parm)
    return(Modelout)
    }
```

39.4. Initial Values

```
Initial.Values <- c(rep(0,J+1), rep(0,J), rep(0,J+1), rep(0,N*J), rep(1,J))
```

40. Gamma Regression

40.1. Form

$$\mathbf{y} \sim \mathcal{G}(\lambda\tau, \tau)$$
$$\lambda = \exp(\mathbf{X}\beta)$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\tau \sim \mathcal{HC}(25)$$

40.2. Data

```
N <- 20
J <- 3
X <- matrix(runif(N*J,-2,2),N,J); X[,1] <- 1
beta <- runif(J,-2,2)
y <- round(exp(tcrossprod(X, t(beta)))) + 0.1 #Must be > 0
mon.names <- c("LP","sigma2")
parm.names <- as.parm.names(list(beta=rep(0,J), tau=0))
```

```
pos.beta <- grep("beta", parm.names)
pos.tau <- grep("tau", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    tau <- runif(1)
    return(c(beta, tau))
    }
MyData <- list(J=J, N=N, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.tau=pos.tau, y=y)
```

40.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
    parm[Data$pos.tau] <- tau
    sigma2 <- 1/tau
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    tau.prior <- dhalfcauchy(tau, 25, log=TRUE)
    ### Log-Likelihood
    lambda <- exp(tcrossprod(Data$X, t(beta)))
    LL <- sum(dgamma(Data$y, tau*lambda, tau, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + tau.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,sigma2),
        yhat=rgamma(nrow(lambda), tau*lambda, tau), parm=parm)
    return(Modelout)
    }
```

40.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

41. Geographically Weighted Regression

41.1. Form

$$\mathbf{y}_{i,k} \sim \mathcal{N}(\mu_{i,k}, \tau_{i,k}^{-1}), \quad i = 1, \ldots, N, \quad k = 1, \ldots, N$$

$$\mu_{i,1:N} = \mathbf{X}\beta_{i,1:J}$$

$$\tau = \frac{1}{\sigma^2}\mathbf{w}\nu$$

$$\mathbf{w} = \frac{\exp(-0.5\mathbf{Z}^2)}{\mathbf{h}}$$

$$\alpha \sim \mathcal{U}(1.5, 100)$$

$$\beta_{i,j} \sim \mathcal{N}(0, 1000), \quad i = 1, \ldots, N, \quad j = 1, \ldots, J$$

$$\mathbf{h} \sim \mathcal{N}(0.1, 1000) \in [0.1, \infty]$$

$$\nu_{i,k} \sim \mathcal{G}(\alpha, 2), \quad i = 1, \ldots, N, \quad k = 1, \ldots, N$$

$$\sigma_i \sim \mathcal{HC}(25), \quad i = 1, \ldots, N$$

41.2. Data

```
crime <- c(18.802, 32.388, 38.426, 0.178, 15.726, 30.627, 50.732,
    26.067, 48.585, 34.001, 36.869, 20.049, 19.146, 18.905, 27.823,
    16.241, 0.224, 30.516, 33.705, 40.970, 52.794, 41.968, 39.175,
    53.711, 25.962, 22.541, 26.645, 29.028, 36.664, 42.445, 56.920,
    61.299, 60.750, 68.892, 38.298, 54.839, 56.706, 62.275, 46.716,
    57.066, 54.522, 43.962, 40.074, 23.974, 17.677, 14.306, 19.101,
    16.531, 16.492)
income <- c(21.232, 4.477, 11.337, 8.438, 19.531, 15.956, 11.252,
    16.029, 9.873, 13.598, 9.798, 21.155, 18.942, 22.207, 18.950,
    29.833, 31.070, 17.586, 11.709, 8.085, 10.822, 9.918, 12.814,
    11.107, 16.961, 18.796, 11.813, 14.135, 13.380, 17.017, 7.856,
    8.461, 8.681, 13.906, 14.236, 7.625, 10.048, 7.467, 9.549,
    9.963, 11.618, 13.185, 10.655, 14.948, 16.940, 18.739, 18.477,
    18.324, 25.873)
housing <- c(44.567, 33.200, 37.125, 75.000, 80.467, 26.350, 23.225,
    28.750, 18.000, 96.400, 41.750, 47.733, 40.300, 42.100, 42.500,
    61.950, 81.267, 52.600, 30.450, 20.300, 34.100, 23.600, 27.000,
    22.700, 33.500, 35.800, 26.800, 27.733, 25.700, 43.300, 22.850,
    17.900, 32.500, 22.500, 53.200, 18.800, 19.900, 19.700, 41.700,
    42.900, 30.600, 60.000, 19.975, 28.450, 31.800, 36.300, 39.600,
    76.100, 44.333)
easting <- c(35.62, 36.50, 36.71, 33.36, 38.80, 39.82, 40.01, 43.75,
    39.61, 47.61, 48.58, 49.61, 50.11, 51.24, 50.89, 48.44, 46.73,
    43.44, 43.37, 41.13, 43.95, 44.10, 43.70, 41.04, 43.23, 42.67,
    41.21, 39.32, 41.09, 38.3, 41.31, 39.36, 39.72, 38.29, 36.60,
    37.60, 37.13, 37.85, 35.95, 35.72, 35.76, 36.15, 34.08, 30.32,
    27.94, 27.27, 24.25, 25.47, 29.02)
northing <- c(42.38, 40.52, 38.71, 38.41, 44.07, 41.18, 38.00, 39.28,
    34.91, 36.42, 34.46, 32.65, 29.91, 27.80, 25.24, 27.93, 31.91,
    35.92, 33.46, 33.14, 31.61, 30.40, 29.18, 28.78, 27.31, 24.96,
    25.90, 25.85, 27.49, 28.82, 30.90, 32.88, 30.64, 30.35, 32.09,
    34.08, 36.12, 36.30, 36.40, 35.60, 34.66, 33.92, 30.42, 28.26,
```

```
    29.85, 28.21, 26.69, 25.71, 26.58)
N <- length(crime)
J <- 3 #Number of predictors, including the intercept
X <- matrix(c(rep(1,N), income, housing),N,J)
D <- as.matrix(dist(cbind(northing,easting), diag=TRUE, upper=TRUE))
Z <- D / sd(as.vector(D))
y <- matrix(0,N,N); for (i in 1:N) {for (k in 1:N) {y[i,k] <- crime[k]}}
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=0, beta=matrix(0,N,J), H=0,
    nu=matrix(0,N,N), sigma=rep(0,N)))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.H <- grep("H", parm.names)
pos.nu <- grep("nu", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- runif(1,1.5,100)
    beta <- rnorm(Data$N*Data$J)
    H <- runif(1,0.1,1000)
    nu <- rgamma(Data$N*Data$N,alpha,2)
    sigma <- runif(Data$N)
    return(c(alpha, beta, H, nu, sigma))
    }
MyData <- list(J=J, N=N, PGF=PGF, X=X, Z=Z, latitude=northing,
    longitude=easting, mon.names=mon.names, parm.names=parm.names,
    pos.alpha=pos.alpha, pos.beta=pos.beta, pos.H=pos.H, pos.nu=pos.nu,
    pos.sigma=pos.sigma, y=y)
```

41.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- interval(parm[Data$pos.alpha], 1.5, 100)
    parm[Data$pos.alpha] <- alpha
    beta <- matrix(parm[Data$pos.beta], Data$N, Data$J)
    parm[Data$pos.H] <- H <- interval(parm[Data$pos.H], 0.1, Inf)
    parm[Data$pos.nu] <- nu <- interval(parm[Data$pos.nu], 1e-100, Inf)
    nu <- matrix(nu, Data$N, Data$N)     sigma <- interval(parm[Data$pos.sigma],
1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- dunif(alpha, 1.5, 100, log=TRUE)
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    h.prior <- dhalfnorm(H-0.1, 1000, log=TRUE)
    nu.prior <- sum(dgamma(nu, alpha, 2, log=TRUE))
```

```
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    w <- exp(-0.5 * Data$Z^2) / H
    tau <- (1/sigma^2) * w * nu
    mu <- tcrossprod(Data$X, beta)
    LL <- sum(dnormp(Data$y, mu, tau, log=TRUE))
    #WSE <- w * nu * (Data$y - mu)^2; w.y <- w * nu * Data$y
    #WMSE <- rowMeans(WSE); y.w <- rowSums(w.y) / rowSums(w)
    #LAR2 <- 1 - WMSE / sd(y.w)^2
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior + h.prior + nu.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
         yhat=rnormp(prod(dim(mu)), mu, tau), parm=parm)
    return(Modelout)
    }
```

41.4. Initial Values

```
Initial.Values <- c(runif(1,1.5,100), rep(0,N*J), 1, rep(1,N*N), rep(1,N))
```

42. Hidden Markov Model

42.1. Form

This introductory hidden Markov model (HMM) includes N discrete states.

$$\mathbf{y}_t \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2), \quad t = 1, \ldots, T$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma^2)$$

$$\sigma^2 \sim \mathcal{HC}(25)$$

$$\theta_t \sim \mathcal{CAT}(\phi_{\theta_{t-1},1:N}), \quad t = 1, \ldots, T$$

$$\phi_{i,1:N} \sim \mathcal{D}(\alpha_{1:N}), \quad i = 1, \ldots, N$$

$$\mu_0 \sim \mathcal{N}(0, 1000)$$

$$\sigma_0^2 \sim \mathcal{HC}(25)$$

42.2. Data

```
data(demonfx)
y <- as.vector(log(as.matrix(demonfx[1:50,1])))
T <- length(y) #Number of time-periods
N <- 2 #Number of discrete (hidden) states
alpha <- matrix(1,N,N) #Concentration hyperparameter
mon.names <- "LP"
```

```
parm.names <- as.parm.names(list(mu0=rep(0,N), mu1=rep(0,N),
    phi=matrix(0,N,N), sigma2=rep(0,N), theta=rep(0,T)))
pos.mu0 <- grep("mu0", parm.names)
pos.mu1 <- grep("mu1", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma2 <- grep("sigma2", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    mu0 <- sort(runif(Data$N, min(Data$y), max(Data$y)))
    mu1 <- sort(runif(Data$N, min(Data$y), max(Data$y)))
    phi <- matrix(runif(Data$N*Data$N), Data$N, Data$N)
    phi <- as.vector(phi / rowSums(phi))
    sigma2 <- runif(Data$N)
    theta <- rcat(Data$T, rep(1/Data$N,Data$N))
    return(c(mu0, mu1, phi, sigma2, theta))
    }
MyData <- list(N=N, PGF=PGF, T=T, alpha=alpha, mon.names=mon.names,
    parm.names=parm.names, pos.mu0=pos.mu0, pos.mu1=pos.mu1,
    pos.phi=pos.phi, pos.sigma2=pos.sigma2, pos.theta=pos.theta, y=y)
```

42.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    mu0 <- interval(parm[Data$pos.mu0], min(Data$y), max(Data$y))
    parm[Data$pos.mu0] <- mu0
    mu <- interval(parm[Data$pos.mu1], min(Data$y), max(Data$y))
    parm[Data$pos.mu1] <- mu <- sort(mu)
    phi <- matrix(abs(parm[Data$pos.phi]), Data$N, Data$N)
    parm[Data$pos.phi] <- phi <- phi / rowSums(phi)
    sigma2 <- interval(parm[Data$pos.sigma2], 1e-100, Inf)
    parm[Data$pos.sigma2] <- sigma2
    theta <- parm[Data$pos.theta]
    ### Log-Hyperprior
    mu0.prior <- sum(dnormv(mu0, 0, 1000, log=TRUE))
    ### Log-Prior
    mu.prior <- sum(dnormv(mu, mu0, sigma2, log=TRUE))
    phi.prior <- 0
    for (i in 1:Data$N)
        phi.prior <- phi.prior + sum(ddirichlet(phi[i,], Data$alpha[i,],
            log=TRUE))
    sigma2.prior <- sum(dhalfcauchy(sigma2, 25, log=TRUE))
    theta.prior <- sum(dcat(theta, rbind(rep(1/Data$N,Data$N),
        phi[theta[-Data$T],]), log=TRUE))
    ### Log-Likelihood
```

```
    LL <- sum(dnormv(Data$y, mu[theta], sigma2[theta], log=TRUE))
    ### Log-Posterior
    LP <- LL + mu0.prior + mu.prior + phi.prior + sigma2.prior + theta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnormv(length(theta), mu[theta], sigma2[theta]), parm=parm)
    return(Modelout)
    }
```

42.4. Initial Values

```
Initial.Values <- c(sort(runif(N, min(y), max(y))),
    sort(runif(N, min(y), max(y))), runif(N*N), runif(N),
    rcat(T, rep(1/N,N)))
```

43. Inverse Gaussian Regression

43.1. Form

$$\mathbf{y} \sim \mathcal{N}^{-1}(\mu, \lambda)$$

$$\mu = \exp(\mathbf{X}\beta) + C$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\lambda \sim \mathcal{HC}(25)$$

where C is a small constant, such as 1.0E-10.

43.2. Data

```
N <- 100
J <- 3 #Number of predictors, including the intercept
X <- matrix(1,N,J)
for (j in 2:J) {X[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))}
beta.orig <- runif(J,-3,3)
e <- rnorm(N,0,0.1)
y <- exp(tcrossprod(X, t(beta.orig)) + e)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), lambda=0))
pos.beta <- grep("beta", parm.names)
pos.lambda <- grep("lambda", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    lambda <- runif(1)
    return(c(beta, lambda))
    }
```

```
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.lambda=pos.lambda, y=y)
```

43.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    lambda <- interval(parm[Data$pos.lambda], 1e-100, Inf)
    parm[Data$pos.lambda] <- lambda
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    lambda.prior <- dhalfcauchy(lambda, 25, log=TRUE)
    ### Log-Likelihood
    mu <- exp(tcrossprod(Data$X, t(beta))) + 1.0E-10
    LL <- sum(dinvgaussian(Data$y, mu, lambda, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + lambda.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rinvgaussian(length(mu), mu, lambda), parm=parm)
    return(Modelout)
    }
```

43.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

44. Kriging

This is an example of universal kriging of $\mathbf{y}$ given $\mathbf{X}$, regression effects β, and spatial effects ζ. Euclidean distance between spatial coordinates (longitude and latitude) is used for each of $i = 1, \ldots, N$ records of $\mathbf{y}$. An additional record is created from the same data-generating process to compare the accuracy of interpolation. For the spatial component, ϕ is the rate of spatial decay and κ is the scale. κ is often difficult to identify, so it is set to 1 (Gaussian), but may be allowed to vary up to 2 (Exponential). In practice, ϕ is also often difficult to identify. While Σ is spatial covariance, spatial correlation is $\rho = \exp(-\phi\mathbf{D})$. To extend this to a large data set, consider the predictive process kriging example in section 45.

44.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma_1^2)$$

$$\mu = \mathbf{X}\beta + \zeta$$

$$\mathbf{y}^{new} = \mathbf{X}\beta + \sum_{i=1}^{N}\left(\frac{\rho_i}{\sum \rho}\zeta_i\right)$$
$$\rho = \exp(-\phi \mathbf{D}^{new})^{\kappa}$$
$$\zeta \sim \mathcal{N}_N(\zeta_\mu, \Sigma)$$
$$\Sigma = \sigma_2^2 \exp(-\phi \mathbf{D})^{\kappa}$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, 2$$
$$\sigma_j \sim \mathcal{HC}(25) \in [0.1, 10], \quad j = 1, \ldots, 2$$
$$\phi \sim \mathcal{U}(1, 5)$$
$$\zeta_\mu = 0$$
$$\kappa = 1$$

44.2. Data

```
N <- 20
longitude <- runif(N+1,0,100)
latitude <- runif(N+1,0,100)
D <- as.matrix(dist(cbind(longitude,latitude), diag=TRUE, upper=TRUE))
Sigma <- 10000 * exp(-1.5 * D)
zeta <- colMeans(rmvn(1000, rep(0,N+1), Sigma))
beta <- c(50,2)
X <- matrix(runif((N+1)*2,-2,2),(N+1),2); X[,1] <- 1
mu <- as.vector(tcrossprod(X, t(beta)))
y <- mu + zeta
longitude.new <- longitude[N+1]; latitude.new <- latitude[N+1]
Xnew <- X[N+1,]; ynew <- y[N+1]
longitude <- longitude[1:N]; latitude <- latitude[1:N]
X <- X[1:N,]; y <- y[1:N]
D <- as.matrix(dist(cbind(longitude,latitude), diag=TRUE, upper=TRUE))
D.new <- sqrt((longitude - longitude.new)^2 + (latitude - latitude.new)^2)
mon.names <- c("LP","ynew")
parm.names <- as.parm.names(list(zeta=rep(0,N), beta=rep(0,2),
    sigma=rep(0,2), phi=0))
pos.zeta <- grep("zeta", parm.names)
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.phi <- grep("phi", parm.names)
PGF <- function(Data) {
    beta <- rnorm(2)
    sigma <- runif(2,0.1,10)
    phi <- runif(1,1,5)
    kappa <- 1
    zeta <- rmvn(1, rep(0,Data$N),
        sigma[2]*sigma[2]*exp(-phi*Data$D)^kappa)
```

```
    return(c(zeta, beta, sigma, phi))
    }
MyData <- list(D=D, D.new=D.new, latitude=latitude, longitude=longitude,
    N=N, PGF=PGF, X=X, Xnew=Xnew, mon.names=mon.names,
    parm.names=parm.names, pos.zeta=pos.zeta, pos.beta=pos.beta,
    pos.sigma=pos.sigma, pos.phi=pos.phi, y=y)
```

44.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    zeta <- parm[Data$pos.zeta]
    kappa <- 1
    sigma <- interval(parm[Data$pos.sigma], 0.1, 10)
    parm[Data$pos.sigma] <- sigma
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], 1, 5)
    Sigma <- sigma[2]*sigma[2] * exp(-phi * Data$D)^kappa
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    zeta.prior <- dmvn(zeta, rep(0, Data$N), Sigma, log=TRUE)
    sigma.prior <- sum(dhalfcauchy(sigma - 1, 25, log=TRUE))
    phi.prior <- dunif(phi, 1, 5, log=TRUE)
    ### Interpolation
    rho <- exp(-phi * Data$D.new)^kappa
    ynew <- rnorm(1, sum(beta * Data$Xnew) + sum(rho / sum(rho) * zeta),
        sigma[1])
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta)) + zeta
    LL <- sum(dnorm(Data$y, mu, sigma[1], log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + zeta.prior + sigma.prior + phi.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,ynew),
        yhat=rnorm(length(mu), mu, sigma[1]), parm=parm)
    return(Modelout)
    }
```

44.4. Initial Values

```
Initial.Values <- c(rep(0,N), rep(0,2), rep(1,2), 1)
```

45. Kriging, Predictive Process

The first K of N records in $\mathbf{y}$ are used as knots for the parent process, and the predictive

process involves records $(K+1), \ldots, N$. For more information on kriging, see section 44.

45.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma_1^2)$$

$$\mu_{1:K} = \mathbf{X}_{1:K,1:J}\beta + \zeta$$

$$\mu_{(K+1):N} = \mathbf{X}_{(K+1):N,1:J}\beta + \sum_{p=1}^{N-K} \frac{\lambda_{p,1:K}}{\sum_{q=1}^{N-K} \lambda_{q,1:K}} \zeta^T$$

$$\lambda = \exp(-\phi \mathbf{D}_P)^\kappa$$

$$\mathbf{y}^{new} = \mathbf{X}\beta + \sum_{k=1}^{K} (\frac{\rho_k}{\sum \rho} \zeta_k)$$

$$\rho = \exp(-\phi \mathbf{D}^{new})^\kappa$$

$$\zeta \sim \mathcal{N}_K(0, \Sigma)$$

$$\Sigma = \sigma_2^2 \exp(-\phi \mathbf{D})^\kappa$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, 2$$

$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, 2$$

$$\phi \sim \mathrm{N}(0, 1000) \in [1, 5]$$

$$\kappa = 1$$

45.2. Data

```
N <- 100
K <- 30 #Number of knots
longitude <- runif(N+1,0,100)
latitude <- runif(N+1,0,100)
D <- as.matrix(dist(cbind(longitude,latitude), diag=TRUE, upper=TRUE))
Sigma <- 10000 * exp(-1.5 * D)
zeta <- colMeans(rmvn(1000, rep(0,N+1), Sigma))
beta <- c(50,2)
X <- matrix(runif((N+1)*2,-2,2),(N+1),2); X[,1] <- 1
mu <- as.vector(tcrossprod(X, t(beta)))
y <- mu + zeta
longitude.new <- longitude[N+1]; latitude.new <- latitude[N+1]
Xnew <- X[N+1,]; ynew <- y[N+1]
longitude <- longitude[1:N]; latitude <- latitude[1:N]
X <- X[1:N,]; y <- y[1:N]
D <- as.matrix(dist(cbind(longitude[1:K],latitude[1:K]), diag=TRUE,
    upper=TRUE))
D.P <- matrix(0, N-K, K)
for (i in (K+1):N) {
```

```
    D.P[K+1-i,] <- sqrt((longitude[1:K] - longitude[i])^2 +
        (latitude[1:K] - latitude[i])^2)}
D.new <- sqrt((longitude[1:K] - longitude.new)^2 +
    (latitude[1:K] - latitude.new)^2)
mon.names <- c("LP","ynew")
parm.names <- as.parm.names(list(zeta=rep(0,K), beta=rep(0,2),
    sigma=rep(0,2), phi=0))
pos.zeta <- grep("zeta", parm.names)
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.phi <- grep("phi", parm.names)
PGF <- function(Data) {
    beta <- rnorm(2)
    sigma <- runif(2,0.1,10)
    phi <- runif(1,1,5)
    kappa <- 1
    zeta <- rmvn(1, rep(0,Data$K),
        sigma[2]*sigma[2]*exp(-phi*Data$D)^kappa)
    return(c(zeta, beta, sigma, phi))
    }
MyData <- list(D=D, D.new=D.new, D.P=D.P, K=K, N=N, PGF=PGF, X=X,
    Xnew=Xnew, latitude=latitude, longitude=longitude,
    mon.names=mon.names, parm.names=parm.names, pos.zeta=pos.zeta,
    pos.beta=pos.beta, pos.sigma=pos.sigma, pos.phi=pos.phi, y=y)
```

45.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    zeta <- parm[Data$pos.zeta]
    kappa <- 1
    sigma <- interval(parm[Data$pos.sigma], 1, Inf)
    parm[Data$pos.sigma] <- sigma
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], 1, 5)
    Sigma <- sigma[2]*sigma[2] * exp(-phi * Data$D)^kappa
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    zeta.prior <- dmvn(zeta, rep(0, Data$K), Sigma, log=TRUE)
    sigma.prior <- sum(dhalfcauchy(sigma - 1, 25, log=TRUE))
    phi.prior <- dunif(phi, 1, 5, log=TRUE)
    ### Interpolation
    rho <- exp(-phi * Data$D.new)^kappa
    ynew <- rnorm(1, sum(beta * Data$Xnew) + sum(rho / sum(rho) * zeta),
        sigma)
```

```
### Log-Likelihood
mu <- tcrossprod(Data$X, t(beta))
mu[1:Data$K] <- mu[1:Data$K] + zeta
lambda <- exp(-phi * Data$D.P)^kappa
mu[(Data$K+1):Data$N] <- mu[(Data$K+1):Data$N] +
     rowSums(lambda / rowSums(lambda) *
     matrix(zeta, Data$N - Data$K, Data$K, byrow=TRUE))
LL <- sum(dnorm(Data$y, mu, sigma[1], log=TRUE))
### Log-Posterior
LP <- LL + beta.prior + zeta.prior + sigma.prior + phi.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,ynew),
     yhat=rnorm(length(mu), mu, sigma[1]), parm=parm)
return(Modelout)
}
```

45.4. Initial Values

```
Initial.Values <- c(rep(0,K), c(mean(y), 0), rep(1,2), 3)
```

46. Laplace Regression

This linear regression specifies that $\mathbf{y}$ is Laplace-distributed, where it is usually Gaussian or normally-distributed. It has been claimed that it should be surprising that the normal distribution became the standard, when the Laplace distribution usually fits better and has wider tails (Kotz, Kozubowski, and Podgorski 2001). Another popular alternative is to use the t-distribution (see Robust Regression in section 84), though it is more computationally expensive to estimate, because it has three parameters. The Laplace distribution has only two parameters, location and scale like the normal distribution, and is computationally easier to fit. This example could be taken one step further, and the parameter vector β could be Laplace-distributed. Laplace's Demon recommends that users experiment with replacing the normal distribution with the Laplace distribution.

46.1. Form

$$\begin{aligned}
\mathbf{y} &\sim \mathcal{L}(\mu, \sigma^2) \\
\mu &= \mathbf{X}\beta \\
\beta_j &\sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J \\
\sigma &\sim \mathcal{HC}(25)
\end{aligned}$$

46.2. Data

```
N <- 10000
J <- 5
```

```
X <- matrix(1,N,J)
for (j in 2:J) {X[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))}
beta <- runif(J,-3,3)
e <- rlaplace(N,0,0.1)
y <- tcrossprod(X, t(beta)) + e
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    sigma <- runif(1)
    return(c(beta, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)
```

46.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- sum(dlaplace(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rlaplace(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

46.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

47. Latent Dirichlet Allocation

47.1. Form

$$\mathbf{Y}_{m,n} \sim \mathcal{CAT}(\phi[\mathbf{Z}_{m,n},]), \quad m = 1,\ldots,M, \quad n = 1,\ldots,N$$
$$\mathbf{Z}_{m,n} \sim \mathcal{CAT}(\theta_{m,1:K})$$
$$\phi_{k,v} \sim \mathcal{D}(\beta)$$
$$\theta_{m,k} \sim \mathcal{D}(\alpha)$$
$$\alpha_k = 1, \quad k = 1,\ldots,K$$
$$\beta_v = 1, \quad v = 1,\ldots,V$$

47.2. Data

```
K <- 2 #Number of (latent) topics
M <- 4 #Number of documents in corpus
N <- 15 #Maximum number of (used) words per document
V <- 5 #Maximum number of occurrences of any word (Vocabulary size)
Y <- matrix(rcat(M*N,rep(1/V,V)), M, N)
rownames(Y) <- paste("doc", 1:nrow(Y), sep="")
colnames(Y) <- paste("word", 1:ncol(Y), sep="")
#Note:  Y is usually represented as w, a matrix of word counts.
if(min(Y) == 0) Y <- Y + 1 #A zero cannot occur, Y must be 1,2,...,V.
V <- max(Y) #Maximum number of occurrences of any word (Vocabulary size)
alpha <- rep(1,K) # hyperparameters (constant)
beta <- rep(1,V)
mon.names <- "LP"
parm.names <- as.parm.names(list(phi=matrix(0,K,V), theta=matrix(0,M,K),
    Z=matrix(0,M,N)))
pos.phi <- grep("phi", parm.names)
pos.theta <- grep("theta", parm.names)
pos.Z <- grep("Z", parm.names)
PGF <- function(Data) {
    phi <- matrix(runif(Data$J*Data$V), Data$K, Data$V)
    phi <- phi / rowSums(phi)
    theta <- matrix(runif(Data$M*Data$K), Data$M, Data$K)
    theta <- theta / rowSums(theta)
    z <- rcat(Data$M*Data$N, rep(1/Data$K,Data$K))
    return(c(as.vector(phi), as.vector(theta), z))}
MyData <- list(K=K, M=M, N=N, PGF=PGF, V=V, Y=Y, alpha=alpha, beta=beta,
    mon.names=mon.names, parm.names=parm.names, pos.phi=pos.phi,
    pos.theta=pos.theta, pos.Z=pos.Z)
```

47.3. Model

```
Model <- function(parm, Data)
    {
```

```
### Parameters
phi <- matrix(interval(parm[Data$pos.phi], 0, 1), Data$K, Data$V)
phi <- phi / rowSums(phi)
parm[Data$pos.phi] <- as.vector(phi)
theta <- matrix(interval(parm[Data$pos.theta], 0, 1), Data$M, Data$K)
theta <- theta / rowSums(theta)
parm[Data$pos.theta] <- as.vector(theta)
Z <- matrix(parm[Data$pos.Z], Data$M, Data$N)
### Log-Prior
phi.prior <- sum(ddirichlet(phi, beta, log=TRUE))
theta.prior <- sum(ddirichlet(theta, alpha, log=TRUE))
### Log-Likelihood
LL <- Z.prior <- 0
Yhat <- Data$Y
for (m in 1:Data$M) {for (n in 1:Data$N) {
    Z.prior + Z.prior + dcat(Z[m,n], theta[m,], log=TRUE)
    LL <- LL + dcat(Data$Y[m,n], as.vector(phi[Z[m,n],]), log=TRUE)
    Yhat[m,n] <- rcat(1, as.vector(phi[Z[m,n],]))}}
### Log-Posterior
LP <- LL + phi.prior + theta.prior + Z.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=Yhat, parm=parm)
return(Modelout)
}
```

47.4. Initial Values

```
Initial.Values <- c(rep(1/V,K*V), rep(1/K,M*K), rcat(M*N,rep(1/K,K)))
```

48. Linear Regression

48.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mu = \mathbf{X}\beta$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\sigma \sim \mathcal{HC}(25)$$

48.2. Data

```
N <- 10000
J <- 5
X <- matrix(1,N,J)
for (j in 2:J) {X[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))}
```

```
beta <- runif(J,-3,3)
e <- rnorm(N,0,0.1)
y <- tcrossprod(X, t(beta)) + e
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    sigma <- runif(1)
    return(c(beta, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)
```

48.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    sigma.prior <- dgamma(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

48.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

49. Linear Regression, Frequentist

By eliminating prior probabilities, a frequentist linear regression example is presented. Although frequentism is not endorsed here, the purpose of this example is to illustrate how the **LaplacesDemon** package can be used for Bayesian or frequentist inference.

49.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mu = \mathbf{X}\beta$$

49.2. Data

```
N <- 10000
J <- 5
X <- matrix(1,N,J)
for (j in 2:J) {X[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))}
beta <- runif(J,-3,3)
e <- rnorm(N,0,0.1)
y <- tcrossprod(X, t(beta)) + e
mon.names <- "LL"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)
```

49.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma, 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     Modelout <- list(LP=LL, Dev=-2*LL, Monitor=LL,
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }
```

49.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

50. Linear Regression, Hierarchical Bayesian

50.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \mathbf{X}\beta$$
$$\beta_j \sim \mathcal{N}(\gamma, \delta), \quad j = 1, \ldots, J$$
$$\gamma \sim \mathcal{N}(0, 1000)$$
$$\delta \sim \mathcal{HC}(25)$$
$$\sigma \sim \mathcal{HC}(\tau)$$
$$\tau \sim \mathcal{HC}(25)$$

50.2. Data

```
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), gamma=0, delta=0, sigma=0,
    tau=0))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.delta <- grep("delta", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.tau <- grep("tau", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    gamma <- rnorm(1)
    delta <- runif(1)
    sigma <- runif(1)
    tau <- runif(1)
    return(c(beta, gamma, delta, sigma, tau))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma,
```

```
pos.delta=pos.delta, pos.sigma=pos.sigma, pos.tau=pos.tau, y)
```

50.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperparameters
    gamma <- parm[Data$pos.gamma]
    delta <- interval(parm[Data$pos.delta], 1e-100, Inf)
    parm[Data$pos.delta] <- delta
    parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Hyperprior
    gamma.prior <- dnormv(gamma, 0, 1000, log=TRUE)
    delta.prior <- dhalfcauchy(delta, 25, log=TRUE)
    tau.prior <- dhalfcauchy(tau, 25, log=TRUE)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, gamma, delta, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, tau, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + gamma.prior + delta.prior + sigma.prior +
        tau.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

50.4. Initial Values

```
Initial.Values <- c(rep(0,J), 0, rep(1,3))
```

51. Linear Regression, Multilevel

51.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu_i = \mathbf{X}\beta_{\mathbf{m}[i],1:J}$$

$$\beta_{g,1:J} \sim \mathcal{N}_J(\gamma, \Omega^{-1}), \quad g = 1, \ldots, G$$
$$\Omega \sim \mathcal{W}_{J+1}(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_J$$
$$\gamma_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\sigma \sim \mathcal{HC}(25)$$

where $\mathbf{m}$ is a vector of length N, and each element indicates the multilevel group ($g = 1, \ldots, G$) for the associated record.

51.2. Data

```
N <- 30
J <- 2 ### Number of predictors (including intercept)
G <- 2 ### Number of Multilevel Groups
X <- cbind(1, matrix(rnorm(N*(J-1),0,1),N,J-1))
Sigma <- matrix(runif(J*J,-1,1),J,J)
diag(Sigma) <- runif(J,1,5)
Sigma <- as.positive.definite(Sigma)
gamma <- runif(J,-1,1)
beta <- matrix(NA,G,J)
for (g in 1:G) {beta[g,] <- rmvn(1, gamma, Sigma)}
m <- rcat(N, rep(1/G,G)) ### Multilevel group indicator
y <- rowSums(beta[m,] * X) + rnorm(N,0,0.1)
S <- diag(J)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=matrix(0,G,J), gamma=rep(0,J),
    sigma=0, U=S), uppertri=c(0,0,0,1))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    U <- rwishartc(Data$J+1, Data$S)
    gamma <- rnorm(Data$J)
    beta <- as.vector(rmvnpc(Data$G, gamma, U))
    sigma <- runif(1)
    return(c(beta, gamma, sigma, U[upper.tri(U, diag=TRUE)]))
    }
MyData <- list(G=G, J=J, N=N, PGF=PGF, S=S, X=X, m=m, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma,
    pos.sigma=pos.sigma, y=y)
```

51.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
```

```
beta <- matrix(parm[Data$pos.beta], Data$G, Data$J)
gamma <- parm[Data$pos.gamma]
sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
parm[Data$pos.sigma] <- sigma
U <- as.parm.matrix(U, Data$J, parm, Data, chol=TRUE)
diag(U) <- exp(diag(U))
### Log-Prior
U.prior <- dwishartc(U, Data$J+1, Data$S, log=TRUE)
beta.prior <- sum(dmvnpc(beta, gamma, U, log=TRUE))
gamma.prior <- sum(dnormv(gamma, 0, 100, log=TRUE))
sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
### Log-Likelihood
mu <- rowSums(beta[Data$m,] * Data$X)
LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
### Log-Posterior
LP <- LL + U.prior + beta.prior + gamma.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
     yhat=rnorm(length(mu), mu, sigma), parm=parm)
return(Modelout)
}
```

51.4. Initial.Values

```
Initial.Values <- c(rep(0,G*J), rep(0,J), 1,
     upper.triangle(S, diag=TRUE))
```

52. Linear Regression with Full Missingness

With 'full missingness', there are missing values for both the dependent variable (DV) and at least one independent variable (IV). The 'full likelihood' approach to full missingness is excellent as long as the model is identifiable. When it is not identifiable, imputation may be done in a previous stage, such as with the `MISS` function. In this example, matrix α is for regression effects for IVs, vector β is for regression effects for the DV, vector γ is for missing values for IVs, and δ is for missing values for the DV.

52.1. Form

$$\mathbf{y}^{imp} \sim \mathcal{N}(\nu, \sigma_J^2)$$
$$\mathbf{X}^{imp} \sim \mathcal{N}(\mu, \sigma_{-J}^2$$
$$\nu = \mathbf{X}^{imp}\beta$$
$$\mu = \mathbf{X}^{imp}\alpha$$
$$\mathbf{y}^{imp} = \begin{cases} \delta & \text{if } \mathbf{y}^{mis} \\ \mathbf{y}^{obs} & \end{cases}$$

$$\mathbf{X}^{imp} = \begin{cases} \gamma & \text{if } \mathbf{X}^{mis} \\ \mathbf{X}^{obs} \end{cases}$$

$$\alpha_{j,l} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, (J-1), \quad l = 1, \ldots, (J-1)$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\gamma_m \sim \mathcal{N}(0, 1000), \quad m = 1, \ldots, M$$

$$\delta_p \sim \mathcal{N}(0, 1000), \quad p = 1, \ldots, P$$

$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, J$$

52.2. Data

```
N <- 100
J <- 5
X <- matrix(runif(N*J,-2,2),N,J); X[,1] <- 1 #Design matrix X
M <- matrix(round(runif(N*J)-0.45),N,J); M[,1] <- 0 #Missing indicators
X <- ifelse(M == 1, NA, X) #Simulated X gets missings according to M
beta.orig <- runif(J,-2,2)
y <- as.vector(tcrossprod(X, t(beta.orig)) + rnorm(N,0,0.1))
y[sample(1:N, round(N*.05))] <- NA
m <- ifelse(is.na(y), 1, 0) #Missing indicator for vector y
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=matrix(0,J-1,J-1),
     beta=rep(0,J),
     gamma=rep(0,sum(is.na(X))),
     delta=rep(0,sum(is.na(y))),
     sigma=rep(0,J)))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.delta <- grep("delta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     alpha <- rnorm((Data$J-1)*(Data$J-1))
     beta <- rnorm(Data$J)
     gamma <- rnorm(sum(is.na(Data$X)))
     delta <- rnorm(sum(is.na(Data$y)), mean(Data$y, na.rm=TRUE), 1)
     sigma <- runif(Data$J)
     return(c(alpha, beta, gamma, delta, sigma))
     }
MyData <- list(J=J, N=N, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta,
     pos.gamma=pos.gamma, pos.delta=pos.delta, pos.sigma=pos.sigma, y=y)
```

52.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     alpha <- matrix(parm[Data$pos.alpha], Data$J-1, Data$J-1)
     beta <- parm[Data$pos.beta]
     gamma <- parm[Data$pos.gamma]
     delta <- parm[Data$pos.delta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     gamma.prior <- sum(dnormv(gamma, 0, 1000, log=TRUE))
     delta.prior <- sum(dnormv(delta, 0, 1000, log=TRUE))
     sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
     ### Log-Likelihood
     mu <- X.imputed <- Data$X
     X.imputed[which(is.na(X.imputed))] <- gamma
     y.imputed <- Data$y
     y.imputed[which(is.na(y.imputed))] <- delta
     for (j in 2:Data$J) {mu[,j] <- tcrossprod(X.imputed[,-j],
          t(alpha[,(j-1)]))}
     nu <- tcrossprod(X.imputed, t(beta))
     LL <- sum(dnorm(X.imputed[,-1], mu[,-1],
          matrix(sigma[1:(Data$J-1)], Data$N, Data$J-1), log=TRUE),
          dnorm(y.imputed, nu, sigma[Data$J], log=TRUE))
     ### Log-Posterior
     LP <- LL + alpha.prior + beta.prior + gamma.prior + delta.prior +
          sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(nu), nu, sigma[Data$J]), parm=parm)
     return(Modelout)
     }
```

52.4. Initial Values

```
Initial.Values <- c(rep(0, (J-1)^2), rep(0,J), rep(0, sum(is.na(X))),
     rep(0, sum(is.na(y))), rep(1,J))
```

53. Linear Regression with Missing Response

This is an introductory example to missing values using data augmentation with auxiliary variables. The dependent variable, or response, has both observed values, $\mathbf{y}^{obs}$, and missing values, $\mathbf{y}^{mis}$. The α vector is for missing value imputation, and enables the use of the full-likelihood by augmenting te state with these auxiliary variables. In the model form, M is

used to denote the number of missing values, though it is used as an indicator in the data.

53.1. Form

$$\mathbf{y}^{imp} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mathbf{y}^{imp} = \begin{cases} \alpha & \text{if } \mathbf{y}^{mis} \\ \mathbf{y}^{obs} & \end{cases}$$

$$\mu = \mathbf{X}\beta$$

$$\alpha_m \sim \mathcal{N}(0, 1000), \quad m = 1, \ldots, M$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\sigma \sim \mathcal{HC}(25)$$

53.2. Data

```
data(demonsnacks)
N <- nrow(demonsnacks)
J <- ncol(demonsnacks)
y <- log(demonsnacks$Calories)
y[sample(1:N, round(N*0.05))] <- NA
M <- ifelse(is.na(y), 1, 0)
X <- cbind(1, as.matrix(demonsnacks[,c(1,3:10)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,sum(M)), beta=rep(0,J),
    sigma=0))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(sum(Data$M), mean(y, na.rm=TRUE), 1)
    beta <- rnorm(Data$J)
    sigma <- runif(1)
    return(c(alpha, beta, sigma))
    }
MyData <- list(J=J, M=M, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta,
    pos.sigma=pos.sigma, y=y)
```

53.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
```

```
alpha <- parm[Data$pos.alpha]
beta <- parm[Data$pos.beta]
sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
parm[Data$pos.sigma] <- sigma
### Log-Prior
alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE)
beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
sigma.prior <- dgamma(sigma, 25, log=TRUE)
### Log-Likelihood
mu <- tcrossprod(Data$X, t(beta))
y.imputed <- Data$y
y.imputed[which(is.na(Data$y))] <- alpha
LL <- sum(dnorm(y.imputed, mu, sigma, log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + beta.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnorm(length(mu), mu, sigma), parm=parm)
return(Modelout)
}
```

53.4. Initial Values

```
Initial.Values <- c(rep(0,sum(M)), rep(0,J), 1)
```

54. Linear Regression with Missing Response via ABB

The Approximate Bayesian Bootstrap (ABB), using the `ABB` function, is used to impute missing values in the dependent variable (DV), or response, given a propensity score. In this example, vector α is used to estimate propensity score η, while vector β is for regression effects, and vector γ has the monitored missing values. For more information on ABB, see the `ABB` function.

54.1. Form

$$\mathbf{y}^{imp} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mathbf{y}^{imp} = \begin{cases} \gamma & \text{if } \mathbf{y}^{mis} \\ \mathbf{y}^{obs} \end{cases}$$

$$\mu = \mathbf{X}\beta$$

$$\gamma \sim p(\mathbf{y}^{obs}|\mathbf{y}^{obs}, \mathbf{y}^{mis}, \eta)$$

$$\eta = \frac{1}{1 + \exp(-\nu)}$$

$$\nu = \mathbf{X}\alpha$$

$$\alpha_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\sigma \sim \mathcal{HC}(25)$$

54.2. Data

```
data(demonsnacks)
N <- nrow(demonsnacks)
J <- ncol(demonsnacks)
y <- log(demonsnacks$Calories)
y[sample(1:N, round(N*0.05))] <- NA
M <- ifelse(is.na(y), 1, 0)
X <- cbind(1, as.matrix(demonsnacks[,c(1,3:10)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- c("LP",paste("gamma[",1:sum(is.na(y)),"]",sep=""))
parm.names <- as.parm.names(list(alpha=rep(0,J), beta=rep(0,J), sigma=0))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(Data$J)
    beta <- rnorm(Data$J)
    sigma <- runif(1)
    return(c(alpha, beta, sigma))
    }
MyData <- list(J=J, M=M, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta,
    pos.sigma=pos.sigma, y=y)
```

54.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    sigma.prior <- dgamma(sigma, 25, log=TRUE)
    ### Log-Likelihood
    y.imputed <- Data$y
    mu <- tcrossprod(Data$X, t(beta))
```

```
nu <- as.vector(tcrossprod(Data$X, t(alpha)))
eta <- invlogit(nu)
breaks <- as.vector(quantile(eta, probs=c(0,0.2,0.4,0.6,0.8,1)))
B <- matrix(breaks[-length(breaks)], length(Data$y), 5, byrow=TRUE)
z <- rowSums(eta >= B)
for (i in 1:5) {
     if(any(is.na(Data$y[which(z == i)]))) {
          imp <- unlist(ABB(Data$y[which(z == i)]))
          y.imputed[which({z == i} & is.na(Data$y))] <- imp}}
gamma <- y.imputed[which(is.na(Data$y))]
LL <- sum(dbern(Data$M, eta, log=TRUE),
     dnorm(y.imputed, mu, sigma, log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + beta.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,gamma),
     yhat=rnorm(length(mu), mu, sigma), parm=parm)
return(Modelout)
}
```

54.4. Initial Values

```
Initial.Values <- c(rep(0,J), rep(0,J), 1)
```

55. Linear Regression with Power Priors

Power priors (Ibrahim and Chen 2000) are a class of informative priors when relevant historical data is available. Power priors may be used when it is desirable to take historical data into account while analyzing similar, current data. Both the current data, $\mathbf{y}$ and $\mathbf{X}$, and historical data, $\mathbf{y}_h$ and $\mathbf{X}_h$, are included in the power prior analysis, where h indicates historical data. Each data set receives its own likelihood function, though the likelihood of the historical data is raised to an exponential power, $\alpha \in [0, 1]$. In this example, α is a constant.

55.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mathbf{y}_h \sim \mathcal{N}(\mu_h, \sigma^2)^\alpha$$

$$\mu = \mathbf{X}\beta$$

$$\mu_h = \mathbf{X}_h\beta$$

$$\alpha = 0.5$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\sigma \sim \mathcal{HC}(25)$$

55.2. Data

```
N <- 100
J <- 5 #Number of predictors, including the intercept
X <- Xh <- matrix(1,N,J)
for (j in 2:J) {
     X[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))
     Xh[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))}
beta.orig <- runif(J,-3,3)
e <- rnorm(N,0,0.1)
yh <- as.vector(tcrossprod(beta.orig, Xh) + e)
y <- as.vector(tcrossprod(beta.orig, X) + e)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(alpha=0.5, J=J, PGF=PGF, X=X, Xh=Xh, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y,
     yh=yh)
```

55.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     muh <- tcrossprod(Data$Xh, t(beta))
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(Data$alpha*dnorm(Data$yh, muh, sigma, log=TRUE) +
          dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
```

```
    return(Modelout)
    }
```

55.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

56. Linear Regression with Zellner's g-Prior

For more information on Zellner's g-prior, see the documentation for the `dzellner` function.

56.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mu = \mathbf{X}\beta$$

$$\beta \sim \mathcal{N}_J(0, g\sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$$

$$g \sim \mathcal{HG}(\alpha), \quad \alpha = 3$$

$$\sigma \sim \mathcal{HC}(25)$$

56.2. Data

```
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(demonsnacks[,c(1,3:10)]))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), g0=0, sigma=0))
pos.beta <- grep("beta", parm.names)
pos.g <- grep("g0", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    g0 <- runif(1)
    sigma <- runif(1)
    return(c(beta, g0, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.g=pos.g,     pos.sigma=pos.sigma,
y=y)
```

56.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    parm[Data$pos.g] <- g <- interval(parm[Data$pos.g], 1e-100, Inf)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Hyperprior
    g.prior <- dhyperg(g, alpha=3, log=TRUE)
    ### Log-Prior
    beta.prior <- dzellner(beta, g, sigma, Data$X, log=TRUE)
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + g.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

56.4. Initial Values

```
Initial.Values <- c(rep(1,J), rep(1,2))
```

57. LSTAR

This is a Logistic Smooth-Threshold Autoregression (LSTAR), and is specified with a transition function that includes γ as the shape parameter, $\mathbf{y}$ as the transition variable, θ as the location parameter, and d as the delay parameter.

57.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma^2), \quad t = 1, \dots, T$$

$$\mu_t = \pi_t(\alpha_1 + \phi_1 \mathbf{y}_{t-1}) + (1 - \pi_t)(\alpha_2 + \phi_2 \mathbf{y}_{t-1}), \quad t = 2, \dots, T$$

$$\pi_t = \frac{1}{1 + \exp(-(\gamma(\mathbf{y}_{t-d} - \theta)))}$$

$$\alpha_j \sim \mathcal{N}(0, 1000) \in [\mathbf{y}_{min}, \mathbf{y}_{max}], \quad j = 1, \dots, 2$$

$$\frac{\phi_j + 1}{2} \sim \mathcal{BETA}(1, 1), \quad j = 1, \dots, 2$$

$$\gamma \sim \mathcal{HC}(25)$$

$$\theta \sim \mathcal{U}(\mathbf{y}_{min}, \mathbf{y}_{max})$$
$$\pi_1 \sim \mathcal{U}(0.001, 0.999)$$
$$\sigma \sim \mathcal{HC}(25)$$

57.2. Data

```
data(demonfx)
y <- as.vector((log(as.matrix(demonfx[,1]))))
T <- length(y)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,2), phi=rep(0,2), gamma=0,
    theta=0, pi=0, sigma=0))
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.theta <- grep("theta", parm.names)
pos.pi <- grep("pi", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- runif(2,min(Data$y),max(Data$y))
    phi <- runif(2, -1, 1)
    gamma <- runif(1)
    theta <- runif(1,min(Data$y),max(Data$y))
    pi <- runif(1, 0.001, 0.999)
    sigma <- runif(1)
    return(c(alpha, phi, gamma, theta, pi, sigma))
    }
MyData <- list(PGF=PGF, T=T, mon.names=mon.names, parm.names=parm.names,
    pos.alpha=pos.alpha, pos.phi=pos.phi, pos.gamma=pos.gamma,
    pos.theta=pos.theta, pos.pi=pos.pi, pos.sigma=pos.sigma, y=y)
```

57.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- interval(parm[Data$pos.alpha], min(Data$y), max(Data$y))
    parm[Data$pos.alpha] <- alpha
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], -1, 1)
    gamma <- interval(parm[Data$pos.gamma], 1e-100, Inf)
    parm[Data$pos.gamma] <- gamma
    theta <- interval(parm[Data$pos.theta], min(Data$y), max(Data$y))
    parm[Data$pos.theta] <- theta
    parm[Data$pos.pi] <- pi <- interval(parm[Data$pos.pi], 0.001, 0.999)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
```

```
parm[Data$pos.sigma] <- sigma
### Log-Prior
alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
phi.prior <- sum(dbeta((phi+1)/2, 1, 1, log=TRUE))
gamma.prior <- dhalfcauchy(gamma, 25, log=TRUE)
theta.prior <- dunif(theta, min(Data$y), max(Data$y), log=TRUE)
pi.prior <- dunif(pi, 0.001, 0.999, log=TRUE)
sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
### Log-Likelihood
pi <- c(pi, 1 / (1 + exp(-(gamma*(Data$y[-Data$T]-theta)))))
mu <- pi * c(alpha[1], alpha[1] + phi[1]*Data$y[-Data$T]) +
    (1-pi) * c(alpha[2], alpha[2] + phi[2]*Data$y[-Data$T])
LL <- sum(dnorm(Data$y[-1], mu[-1], sigma, log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + phi.prior + gamma.prior + theta.prior +
    pi.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnorm(length(mu), mu, sigma), parm=parm)
return(Modelout)
}
```

57.4. Initial Values

```
Initial.Values <- c(rep(mean(y),2), rep(0.5,2), 1, mean(y), 0.5, 1)
```

58. MANCOVA

Since this is a multivariate extension of ANCOVA, please see the ANCOVA example in section 2 for a univariate introduction.

58.1. Form

$$\mathbf{Y}_{i,1:J} \sim \mathcal{N}_K(\mu_{i,1:J}, \Sigma), \quad i = 1, \ldots, N$$

$$\mu_{i,k} = \alpha_k + \beta_{k,\mathbf{X}[i,1]} + \gamma_{k,\mathbf{X}[i,1]} + \mathbf{X}_{1:N,3:(C+J)}\delta_{k,1:C}$$

$$\epsilon_{i,k} = \mathbf{Y}_{i,k} - \mu_{i,k}$$

$$\alpha_k \sim \mathcal{N}(0, 1000), \quad k = 1, \ldots, K$$

$$\beta_{k,l} \sim \mathcal{N}(0, \sigma_1^2), \quad l = 1, \ldots, (L-1)$$

$$\beta_{1:K,L} = -\sum_{l=1}^{L-1} \beta_{1:K,l}$$

$$\gamma_{k,m} \sim \mathcal{N}(0, \sigma_2^2), \quad m = 1, \ldots, (M-1)$$

$$\gamma_{1:K,M} = -\sum_{m=1}^{M-1} \beta_{1:K,m}$$

$$\delta_{k,c} \sim \mathcal{N}(0, 1000)$$

$$\Omega \sim \mathcal{W}_{K+1}(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_K$$

$$\Sigma = \Omega^{-1}$$

$$\sigma_{1:J} \sim \mathcal{HC}(25)$$

58.2. Data

```
C <- 2 #Number of covariates
J <- 2 #Number of factors (treatments)
K <- 3 #Number of endogenous (dependent) variables
L <- 4 #Number of levels in factor (treatment) 1
M <- 5 #Number of levels in factor (treatment) 2
N <- 100
X <- matrix(c(rcat(N, rep(1/L,L)), rcat(N, rep(1/M,M)),
    runif(N*C,0,1)), N, J + C)
alpha <- runif(K,-1,1)
beta <- matrix(runif(K*L,-2,2), K, L)
beta[,L] <- -rowSums(beta[,-L])
gamma <- matrix(runif(K*M,-2,2), K, M)
gamma[,M] <- -rowSums(gamma[,-M])
delta <- matrix(runif(K*C), K, C)
Y <- matrix(NA,N,K)
for (k in 1:K) {
    Y[,k] <- alpha[k] + beta[k,X[,1]] + gamma[k,X[,2]] +
    tcrossprod(delta[k,], X[,-c(1,2)]) + rnorm(1,0,0.1)}
S <- diag(K)
mon.names <- c("LP", "s.o.beta", "s.o.gamma", "s.o.epsilon",
    as.parm.names(list(s.beta=rep(0,K), s.gamma=rep(0,K),
    s.epsilon=rep(0,K))))
parm.names <- as.parm.names(list(alpha=rep(0,K), beta=matrix(0,K,(L-1)),
    gamma=matrix(0,K,(M-1)), delta=matrix(0,K,C), U=diag(K),
    sigma=rep(0,2)), uppertri=c(0,0,0,0,1,0))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.delta <- grep("delta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(Data$K)
    sigma <- runif(2)
    beta <- rnorm(Data$K*(Data$L-1), 0, sigma[1])
    gamma <- rnorm(Data$K*(Data$M-1), 0, sigma[2])
```

```
    delta <- rnorm(Data$K*Data$C)
    U <- rwishartc(Data$K+1, Data$S)
    return(c(alpha, beta, gamma, delta, U[upper.tri(U, diag=TRUE)],
        sigma))
    }
MyData <- list(C=C, J=J, K=K, L=L, M=M, N=N, PGF=PGF, S=S, X=X, Y=Y,
    mon.names=mon.names, parm.names=parm.names, pos.alpha=pos.alpha,
    pos.beta=pos.beta, pos.gamma=pos.gamma, pos.delta=pos.delta,
    pos.sigma=pos.sigma)
```

58.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    beta <- matrix(c(parm[Data$pos.beta], rep(0,Data$K)), Data$K, Data$L)
    beta[,Data$L] <- -rowSums(beta[,-Data$L])
    gamma <- matrix(c(parm[Data$[pos.gamma],
        rep(0,Data$K)), Data$K, Data$M)
    gamma[,Data$M] <- -rowSums(gamma[,-Data$M])
    delta <- matrix(parm[Data$pos.delta], Data$K, Data$C)
    U <- as.parm.matrix(U, Data$K, parm, Data, chol=TRUE)
    diag(U) <- exp(diag(U))
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    beta.prior <- sum(dnorm(beta, 0, sigma[1], log=TRUE))
    gamma.prior <- sum(dnorm(gamma, 0, sigma[2], log=TRUE))
    delta.prior <- sum(dnormv(delta, 0, 1000, log=TRUE))
    U.prior <- dwishartc(U, Data$K+1, Data$S, log=TRUE)
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    mu <- matrix(0,Data$N,Data$K)
    for (k in 1:Data$K) {
        mu[,k] <- alpha[k] + beta[k,Data$X[,1]] + gamma[k,Data$X[,2]] +
        tcrossprod(Data$X[,-c(1,2)], t(delta[k,]))}
    LL <- sum(dmvnpc(Data$Y, mu, U, log=TRUE))
    ### Variance Components, Omnibus
    s.o.beta <- sd(as.vector(beta))
    s.o.gamma <- sd(as.vector(gamma))
    s.o.epsilon <- sd(as.vector(Data$Y - mu))
    ### Variance Components, Univariate
    s.beta <- sqrt(.rowVars(beta))
    s.gamma <- sqrt(.rowVars(gamma))
```

```
s.epsilon <- sqrt(.colVars(Data$Y - mu))
### Log-Posterior
LP <- LL + alpha.prior + beta.prior + gamma.prior + delta.prior +
    U.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP, s.o.beta, s.o.gamma,
    s.o.epsilon, s.beta, s.gamma, s.epsilon),
    yhat=rmvnpc(nrow(mu), mu, U), parm=parm)
return(Modelout)
}
```

58.4. Initial Values

```
Initial.Values <- c(rep(0,K), rep(0,K*(L-1)), rep(0,K*(M-1)),
    rep(0,C*K), upper.triangle(S, diag=TRUE), rep(1,2))
```

59. MANOVA

Since this is a multivariate extension of ANOVA, please see the two-way ANOVA example in section 4 for a univariate introduction.

59.1. Form

$$\mathbf{Y}_{i,1:J} \sim \mathcal{N}_K(\mu_{i,1:J}, \Omega^{-1}), \quad i = 1, \ldots, N$$

$$\mu_{i,k} = \alpha_k + \beta_{k,\mathbf{X}[i,1]} + \gamma_{k,\mathbf{X}[i,1]}$$

$$\epsilon_{i,k} = \mathbf{Y}_{i,k} - \mu_{i,k}$$

$$\alpha_k \sim \mathcal{N}(0, 1000), \quad k = 1, \ldots, K$$

$$\beta_{k,l} \sim \mathcal{N}(0, \sigma_1^2), \quad l = 1, \ldots, (L-1)$$

$$\beta_{1:K,L} = -\sum_{l=1}^{L-1} \beta_{1:K,l}$$

$$\gamma_{k,m} \sim \mathcal{N}(0, \sigma_2^2), \quad m = 1, \ldots, (M-1)$$

$$\gamma_{1:K,M} = -\sum_{m=1}^{M-1} \beta_{1:K,m}$$

$$\Omega \sim \mathcal{W}_{K+1}(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_K$$

$$\sigma_{1:J} \sim \mathcal{HC}(25)$$

59.2. Data

```
J <- 2 #Number of factors (treatments)
K <- 3 #Number of endogenous (dependent) variables
```

```
L <- 4 #Number of levels in factor (treatment) 1
M <- 5 #Number of levels in factor (treatment) 2
N <- 100
X <- cbind(rcat(N, rep(1/L,L)), rcat(N, rep(1/M,M)))
alpha <- runif(K,-1,1)
beta <- matrix(runif(K*L,-2,2), K, L)
beta[,L] <- -rowSums(beta[,-L])
gamma <- matrix(runif(K*M,-2,2), K, M)
gamma[,M] <- -rowSums(gamma[,-M])
Y <- matrix(NA,N,K)
for (k in 1:K) {
    Y[,k] <- alpha[k] + beta[k,X[,1]] + gamma[k,X[,2]] + rnorm(1,0,0.1)}
S <- diag(K)
mon.names <- c("LP", "s.o.beta", "s.o.gamma", "s.o.epsilon",
    as.parm.names(list(s.beta=rep(0,K), s.gamma=rep(0,K),
    s.epsilon=rep(0,K))))
parm.names <- as.parm.names(list(alpha=rep(0,K), beta=matrix(0,K,(L-1)),
    gamma=matrix(0,K,(M-1)), U=diag(K), sigma=rep(0,2)),
    uppertri=c(0,0,0,1,0))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(Data$K)
    sigma <- runif(2)
    beta <- rnorm(Data$K*(Data$L-1), 0, sigma[1])
    gamma <- rnorm(Data$K*(Data$M-1), 0, sigma[2])
    U <- rwishartc(Data$K+1, Data$S)
    return(c(alpha, beta, gamma, U[upper.tri(U, diag=TRUE)], sigma))
    }
MyData <- list(J=J, K=K, L=L, M=M, N=N, PGF=PGF, S=S, X=X, Y=Y,
    mon.names=mon.names, parm.names=parm.names, pos.alpha=pos.alpha,
    pos.beta=pos.beta, pos.gamma=pos.gamma, pos.sigma=pos.sigma)
```

59.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    beta <- matrix(c(parm[Data$pos.beta], rep(0,Data$K)),
    beta[,Data$L] <- -rowSums(beta[,-Data$L])
    gamma <- matrix(c(parm[Data$pos.gamma],
        rep(0,Data$K)), Data$K, Data$M)
    gamma[,Data$M] <- -rowSums(gamma[,-Data$M])
```

```
    U <- as.parm.matrix(U, Data$K, parm, Data, chol=TRUE)
    diag(U) <- exp(diag(U))
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    beta.prior <- sum(dnorm(beta, 0, sigma[1], log=TRUE))
    gamma.prior <- sum(dnorm(gamma, 0, sigma[2], log=TRUE))
    U.prior <- dwishartc(U, Data$K+1, Data$S, log=TRUE)
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    mu <- matrix(0,Data$N,Data$K)
    for (k in 1:Data$K) {
        mu[,k] <- alpha[k] + beta[k,Data$X[,1]] + gamma[k,Data$X[,2]]}
    LL <- sum(dmvnpc(Data$Y, mu, U, log=TRUE))
    ### Variance Components, Omnibus
    s.o.beta <- sd(as.vector(beta))
    s.o.gamma <- sd(as.vector(gamma))
    s.o.epsilon <- sd(as.vector(Data$Y - mu))
    ### Variance Components, Univariate
    s.beta <- sqrt(.rowVars(beta))
    s.gamma <- sqrt(.rowVars(gamma))
    s.epsilon <- sqrt(.colVars(Data$Y - mu))
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior + gamma.prior + U.prior +
        sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP, s.o.beta, s.o.gamma,
        s.o.epsilon, s.beta, s.gamma, s.epsilon),
        yhat=rmvnpc(nrow(mu), mu, U), parm=parm)
    return(Modelout)
    }
```

59.4. Initial Values

```
Initial.Values <- c(rep(0,K), rep(0,K*(L-1)), rep(0,K*(M-1)),
    upper.triangle(S, diag=TRUE), rep(1,2))
```

60. Mixed Logit

60.1. Form

$$\mathbf{y}_i \sim \mathcal{CAT}(\mathbf{p}_{i,1:J}), \quad i = 1, \ldots, N$$

$$\mathbf{p}_{i,j} = \frac{\phi_{i,j}}{\sum_{j=1}^{J} \phi_{i,j}}$$

$$\phi = \exp(\mu)$$

$$\mu_{i,j} = \beta_{j,1:K,i}\mathbf{X}_{i,1:K} \in [-700, 700], \quad i = 1, \ldots, N, \quad j = 1, \ldots, (J-1)$$

$$\mu_{i,J} = 0$$

$$\beta_{j,k,i} \sim \mathcal{N}(\zeta^{\mu}_{j,k}, \zeta^{\sigma}2_{j,k}), \quad i = 1, \ldots, N, \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

$$\zeta^{\mu}_{j,k} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

$$\zeta^{\sigma}_{j,k} \sim \mathcal{HC}25), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

60.2. Data

```
data(demonchoice)
y <- as.numeric(demonchoice[,1])
X <- cbind(1, as.matrix(demonchoice[,2:3]))
for (j in 2:ncol(X)) X[,j] <- CenterScale(X[,j])
N <- length(y)
J <- length(unique(y)) #Number of categories in y
K <- ncol(X) #Number of predictors (including the intercept)
S <- diag(J-1)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=array(0, dim=c(J-1,K,N)),
    zeta.mu=matrix(0,J-1,K), zeta.sigma=matrix(0,J-1,K)))
pos.beta <- grep("beta", parm.names)
pos.zeta.mu <- grep("zeta.mu", parm.names)
pos.zeta.sigma <- grep("zeta.sigma", parm.names)
PGF <- function(Data) {
    zeta.mu <- matrix(rnorm((Data$J-1)*Data$K), Data$J-1, Data$K)
    zeta.sigma <- matrix(runif((Data$J-1)*Data$K), Data$J-1, Data$K)
    beta <- array(rnorm((Data$J-1)*Data$K*Data$N),
        dim=c( Data$J-1, Data$K, Data$N))
    return(c(beta, as.vector(zeta.mu), as.vector(zeta.sigma)))
    }
MyData <- list(J=J, K=K, N=N, PGF=PGF, S=S, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.zeta.mu=pos.zeta.mu,
    pos.zeta.sigma=pos.zeta.sigma, y=y)
```

60.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- array(parm[Data$pos.beta], dim=c(Data$J-1, Data$K, Data$N))
    zeta.mu <- matrix(parm[Data$pos.zeta.mu], Data$J-1, Data$K)
    zeta.sigma <- matrix(interval(parm[Data$pos.zeta.sigma], 1e-100, Inf),
        Data$J-1, Data$K)
```

```
parm[Data$pos.zeta.sigma] <- as.vector(zeta.sigma)
### Log-Hyperprior
zeta.mu.prior <- sum(dnormv(zeta.mu, 0, 1000, log=TRUE))
zeta.sigma.prior <- sum(dhalfcauchy(zeta.sigma, 25, log=TRUE))
### Log-Prior
beta.prior <- sum(dnorm(beta, zeta.mu, zeta.sigma, log=TRUE))
### Log-Likelihood
mu <- matrix(0, Data$N, Data$J)
for (j in 1:(Data$J-1)) mu[,j] <- rowSums(Data$X * t(beta[j, , ]))
mu <- interval(mu, -700, 700, reflect=FALSE)
phi <- exp(mu)
p <- phi / rowSums(phi)
LL <- sum(dcat(Data$y, p, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior + zeta.mu.prior + zeta.sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=rcat(nrow(p), p),
    parm=parm)
return(Modelout)
}
```

60.4. Initial Values

```
Initial.Values <- c(rep(0,(J-1)*K*N), rep(0,(J-1)*K), rep(1,(J-1)*K))
```

61. Mixture Model, Finite

This finite mixture model (FMM) imposes a multilevel structure on each of the J regression effects in β, so that mixture components share a common residual standard deviation, ν_m. Identifiability is gained at the expense of some shrinkage. The record-level mixture membership parameter vector, θ, is a vector of discrete parameters. Discrete parameters are not supported in all algorithms. The example below is updated with the Slice sampler.

61.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu_i = \mathbf{X}_{i,1:J}\beta_{\theta[i],1:J}, \quad i = 1, \ldots, N$$
$$\theta_i \sim \mathcal{CAT}(\pi_{1:M}), \quad i = 1, \ldots, N$$
$$\beta_{m,j} \sim \mathcal{N}(0, \nu_m^2), \quad j = 1, \ldots, J, \quad m = 2, \ldots, M$$
$$\beta_{1,j} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\nu_m \sim \mathcal{HC}(25), \quad m = 1, \ldots, M$$
$$\sigma \sim \mathcal{HC}(25)$$
$$\pi_{1:M} \sim \mathcal{D}(\alpha_{1:M})$$

$$\alpha_m = 1$$

61.2. Data

```
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
M <- 2 #Number of mixtures
N <- length(y) #Number of records
J <- ncol(X) #Number of predictors, including the intercept
for (j in 2:J) X[,j] <- CenterScale(X[,j])
alpha <- rep(1,M) #Prior probability of mixing probabilities
mon.names <- "LP"
parm.names <- as.parm.names(list(theta=rep(0,N), beta=matrix(0,M,J),
    nu=rep(0,M), sigma=0))
pos.theta <- grep("theta", parm.names)
pos.beta <- grep("beta", parm.names)
pos.nu <- grep("nu", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    theta <- rcat(Data$N, rep(1/Data$M, Data$M))
    nu <- runif(Data$M)
    beta <- rnormv(Data$M*Data$J, 0,
        cbind(1000, matrix(nu, Data$M, Data$J-1)))
    sigma <- runif(1)
    return(c(theta, beta, nu, sigma))
    }
MyData <- list(J=J, M=M, N=N, PGF=PGF, X=X, alpha=alpha,
    mon.names=mon.names, parm.names=parm.names, pos.theta=pos.theta,
    pos.beta=pos.beta, pos.nu=pos.nu, pos.sigma=pos.sigma, y=y)
```

61.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- matrix(parm[Data$pos.beta], Data$M, Data$J)
    theta <- parm[Data$pos.theta]
    parm[Data$pos.nu] <- nu <- interval(parm[Data$pos.nu], 1e-100, Inf)
    pi <- rep(0, Data$M)
    tab <- table(theta)
    pi[as.numeric(names(tab))] <- as.vector(tab)
    pi <- pi / sum(pi)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
```

```
beta.prior <- sum(dnormv(beta, 0,
    cbind(1000, matrix(nu, Data$M, Data$J-1)), log=TRUE))
theta.prior <- sum(dcat(theta, p=pi, log=TRUE))
pi.prior <- ddirichlet(pi, Data$alpha, log=TRUE)
nu.prior <- sum(dhalfcauchy(nu, 25, log=TRUE))
sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
### Log-Likelihood
mu <- rowSums(beta[theta,] * Data$X)
LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior + theta.prior + pi.prior + nu.prior +
    sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnorm(length(mu), mu, sigma), parm=parm)
return(Modelout)
}
```

61.4. Initial Values

```
Initial.Values <- c(rcat(N,rep(1/M,M)), rep(0,M*J), rep(1,M), 1)
```

62. Mixture Model, Infinite

This infinite mixture model (IMM) uses a Dirichlet process via truncated stick-breaking. The record-level mixture membership parameter vector, θ, is a vector of discrete parameters. Discrete parameters are not supported in all algorithms. The example below is updated with the Slice sampler.

62.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mu_i = \mathbf{X}_{i,1:J} \beta_{\theta[i],1:J}, \quad i = 1, \ldots, N$$

$$\theta_i \sim \mathcal{CAT}(\pi_{1:M}), \quad i = 1, \ldots, N$$

$$\beta_{m,j} \sim \mathcal{N}(0, \nu_m^2), \quad j = 1, \ldots, J, \quad m = 2, \ldots, M$$

$$\beta_{1,j} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\nu_m \sim \mathcal{HC}(25), \quad m = 1, \ldots, M$$

$$\sigma \sim \mathcal{HC}(25)$$

$$\pi = \text{Stick}(\delta)$$

$$\delta_m \sim \mathcal{BETA}(1, \gamma), m = 1, \ldots, (M-1)$$

$$\gamma \sim \mathcal{G}(\alpha, \iota)$$

$$\alpha \sim \mathcal{HC}(25)$$

$$\iota \sim \mathcal{HC}(25)$$

62.2. Data

```
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
M <- 3 #Maximum number of mixtures to explore
N <- length(y) #Number of records
J <- ncol(X) #Number of predictors, including the intercept
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- c("LP", as.parm.names(list(pi=rep(0,M))))
parm.names <- as.parm.names(list(theta=rep(0,N), delta=rep(0,M-1),
    beta=matrix(0,M,J), nu=rep(0,M), sigma=0, alpha=0, iota=0, gamma=0))
pos.theta <- grep("theta", parm.names)
pos.delta <- grep("delta", parm.names)
pos.beta <- grep("beta", parm.names)
pos.nu <- grep("nu", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.alpha <- grep("alpha", parm.names)
pos.iota <- grep("iota", parm.names)
pos.gamma <- grep("gamma", parm.names)
PGF <- function(Data) {
    nu <- runif(Data$M)
    beta <- rnormv(Data$M*Data$J, 0,
        cbind(1000, matrix(nu, Data$M, Data$J-1)))
    sigma <- runif(1)
    alpha <- runif(1)
    iota <- runif(1)
    gamma <- rgamma(1, alpha, iota)
    delta <- rev(sort(rbeta(Data$M-1, 1, gamma)))
    theta <- rcat(Data$N, Stick(delta))
    return(c(theta, delta, beta, nu, sigma, alpha, iota, gamma))
    }
MyData <- list(J=J, M=M, N=N, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.theta=pos.theta, pos.delta=pos.delta,
    pos.beta=pos.beta, pos.nu=pos.nu, pos.sigma=pos.sigma,
    pos.alpha=pos.alpha, pos.iota=pos.iota, pos.gamma=pos.gamma, y=y)
```

62.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperhyperparameters
```

```
    alpha <- interval(parm[Data$pos.alpha], 1e-100, Inf)
    parm[Data$pos.alpha] <- alpha
    iota <- interval(parm[Data$pos.iota], 1e-100, Inf)
    parm[Data$pos.iota] <- iota
    ### Hyperparameters
    delta <- interval(parm[Data$pos.delta], 1e-10, 1-1e-10)
    parm[Data$pos.delta] <- delta
    gamma <- interval(parm[Data$pos.gamma], 1e-100, Inf)
    parm[Data$pos.gamma] <- gamma
    parm[Data$pos.nu] <- nu <- interval(parm[Data$pos.nu], 1e-100, Inf)
    ### Parameters
    beta <- matrix(parm[Data$pos.beta], Data$M, Data$J)
    theta <- parm[Data$pos.theta]
    pi <- Stick(delta)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Hyperhyperprior
    alpha.prior <- dhalfcauchy(alpha, 25, log=TRUE)
    iota.prior <- dhalfcauchy(iota, 25, log=TRUE)
    ### Log-Hyperprior
    delta.prior <- dStick(delta, gamma, log=TRUE)
    gamma.prior <- dgamma(gamma, alpha, iota, log=TRUE)
    nu.prior <- sum(dhalfcauchy(nu, 25, log=TRUE))
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0,
        cbind(1000, matrix(nu, Data$M, Data$J-1)), log=TRUE))
    theta.prior <- sum(dcat(theta, pi, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- rowSums(beta[theta,]*Data$X)
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + delta.prior + theta.prior + nu.prior +
        sigma.prior + alpha.prior + iota.prior + gamma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,pi),
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

62.4. Initial Values

```
Initial.Values <- c(rcat(N, rev(sort(rStick(M-1,1)))), rep(0.5,M-1),
    rep(0,M*J), rep(1,M), rep(1,4))
```

63. Multinomial Logit

63.1. Form

$$\mathbf{y}_i \sim \mathcal{CAT}(\mathbf{p}_{i,1:J}), \quad i = 1, \ldots, N$$

$$\mathbf{p}_{i,j} = \frac{\phi_{i,j}}{\sum_{j=1}^{J} \phi_{i,j}}, \quad \sum_{j=1}^{J} \mathbf{p}_{i,j} = 1$$

$$\phi = \exp(\mu)$$

$$\mu_{i,J} = 0, \quad i = 1, \ldots, N$$

$$\mu_{i,j} = \mathbf{X}_{i,1:K} \beta_{j,1:K} \in [-700, 700], \quad j = 1, \ldots, (J-1)$$

$$\beta_{j,k} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

63.2. Data

```
data(demonchoice)
y <- as.numeric(demonchoice[,1])
X <- cbind(1, as.matrix(demonchoice[,2:3]))
for (j in 2:ncol(X)) X[,j] <- CenterScale(X[,j])
N <- length(y)
J <- length(unique(y)) #Number of categories in y
K <- ncol(X) #Number of predictors (including the intercept)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=matrix(0,J-1,K)))
PGF <- function(Data) {
    beta <- rnorm((Data$J-1)*Data$K)
    return(beta)
    }
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, y=y)
```

63.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- matrix(parm, Data$J-1, Data$K)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    ### Log-Likelihood
    mu <- matrix(0, Data$N, Data$J)
    mu[,-Data$J] <- tcrossprod(Data$X, beta)
    mu <- interval(mu, -700, 700, reflect=FALSE)
```

```
phi <- exp(mu)
p <- phi / rowSums(phi)
LL <- sum(dcat(Data$y, p, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=rcat(nrow(p), p),
    parm=parm)
return(Modelout)
}
```

63.4. Initial Values

```
Initial.Values <- c(rep(0,(J-1)*K))
```

64. Multinomial Logit, Nested

64.1. Form

$$\mathbf{y}_i \sim \mathcal{CAT}(\mathbf{P}_{i,1:J}), \quad i = 1, \ldots, N$$

$$\mathbf{P}_{1:N,1} = \frac{\mathbf{R}}{\mathbf{R} + \exp(\alpha \mathbf{I})}$$

$$\mathbf{P}_{1:N,2} = \frac{(1 - \mathbf{P}_{1:N,1})\mathbf{S}_{1:N,1}}{\mathbf{V}}$$

$$\mathbf{P}_{1:N,3} = \frac{(1 - \mathbf{P}_{1:N,1})\mathbf{S}_{1:N,2}}{\mathbf{V}}$$

$$\mathbf{R}_{1:N} = \exp(\mu_{1:N,1})$$

$$\mathbf{S}_{1:N,1:2} = \exp(\mu_{1:N,2:3})$$

$$\mathbf{I} = \log(\mathbf{V})$$

$$\mathbf{V}_i = \sum_{k=1}^{K} \mathbf{S}_{i,k}, \quad i = 1, \ldots, N$$

$$\mu_{1:N,1} = \mathbf{X}\iota \in [-700, 700]$$

$$\mu_{1:N,2} = \mathbf{X}\beta_{2,1:K} \in [-700, 700]$$

$$\iota = \alpha\beta_{1,1:K}$$

$$\alpha \sim \mathcal{EXP}(1) \in [0, 2]$$

$$\beta_{j,k} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, (J-1) \quad k = 1, \ldots, K$$

where there are $J = 3$ categories of $\mathbf{y}$, $K = 3$ predictors, $\mathbf{R}$ is the non-nested alternative, $\mathbf{S}$ is the nested alternative, $\mathbf{V}$ is the observed utility in the nest, α is effectively 1 - correlation and

has a truncated exponential distribution, and ι is a vector of regression effects for the isolated alternative after α is taken into account. The third alternative is the reference category.

64.2. Data

```
data(demonchoice)
y <- as.numeric(demonchoice[,1])
X <- cbind(1, as.matrix(demonchoice[,2:3]))
for (j in 2:ncol(X)) X[,j] <- CenterScale(X[,j])
N <- length(y)
J <- length(unique(y)) #Number of categories in y
K <- ncol(X) #Number of predictors (including the intercept)
mon.names <- c("LP", as.parm.names(list(iota=rep(0,K))))
parm.names <- as.parm.names(list(alpha=0, beta=matrix(0,J-1,K)))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
PGF <- function(Data) {
     alpha <- rtrunc(1, "exp", a=0, b=2, rate=1)
     beta <- rnorm((Data$J-1)*Data$K)
     return(c(alpha, beta))
     }
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta, y=y)
```

64.3. Model

```
Model <- function(parm, Data)
     {
     ### Hyperparameters
     alpha.rate <- 1
     ### Parameters
     parm[Data$pos.alpha] <- alpha <- interval(parm[Data$pos.alpha],0,2)
     beta <- matrix(parm[Data$pos.beta], Data$J-1, Data$K)
     ### Log-Prior
     alpha.prior <- dtrunc(alpha, "exp", a=0, b=2, rate=alpha.rate,
          log=TRUE)
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     ### Log-Likelihood
     mu <- P <- matrix(0, Data$N, Data$J)
     iota <- alpha * beta[1,]
     mu[,1] <- tcrossprod(Data$X, t(iota))
     mu[,2] <- tcrossprod(Data$X, t(beta[2,]))
     mu <- interval(mu, -700, 700, reflect=FALSE)
     R <- exp(mu[,1])
     S <- exp(mu[,-1])
     V <- rowSums(S)
     I <- log(V)
```

```
P[,1] <- R / (R + exp(alpha*I))
P[,2] <- (1 - P[,1]) * S[,1] / V
P[,3] <- (1 - P[,1]) * S[,2] / V
LL <- sum(dcat(Data$y, P, log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + beta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,iota),
    yhat=rcat(nrow(P), P), parm=parm)
return(Modelout)
}
```

64.4. Initial Values

```
Initial.Values <- c(0.5, rep(0.1,(J-1)*K))
```

65. Multinomial Probit

65.1. Form

$$\mathbf{W}_{i,1:(J-1)} \sim \mathcal{N}_{J-1}(\mu_{i,1:(J-1)}, \Sigma), \quad i = 1, \ldots, N$$

$$\mathbf{W}_{i,j} \in \begin{cases} [0,10] & \text{if } \mathbf{y}_i = j \\ [-10,0] \end{cases}$$

$$\mu_{1:N,j} = \mathbf{X}\beta_{j,1:K}$$

$$\Sigma = \mathbf{U}^T\mathbf{U}$$

$$\beta_{j,k} \sim \mathcal{N}(0, 10), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

$$\mathbf{U}_{j,k} \sim \mathcal{N}(0, 1), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, (J-1), \quad j \geq k, \quad j \neq k = 1$$

65.2. Data

```
data(demonchoice)
y <- as.numeric(demonchoice[,1])
X <- cbind(1, as.matrix(demonchoice[,2:3]))
for (j in 2:ncol(X)) X[,j] <- CenterScale(X[,j])
N <- length(y)
J <- length(unique(y)) #Number of categories in y
K <- ncol(X) #Number of predictors (including the intercept)
S <- diag(J-1)
U <- matrix(NA,J-1,J-1)
U[upper.tri(U, diag=TRUE)] <- 0
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=matrix(0,(J-1),K),
```

```
    U=U, W=matrix(0,N,J-1)))
parm.names <- parm.names[-which(parm.names == "U[1,1]")]
pos.beta <- grep("beta", parm.names)
pos.U <- grep("U", parm.names)
pos.W <- grep("W", parm.names)
PGF <- function(Data) {
    beta <- rnorm((Data$J-1)*Data$K)
    U <- rnorm((Data$J-2) + (factorial(Data$J-1) /
        (factorial(Data$J-1-2)*factorial(2))))
    W <- matrix(runif(Data$N*(Data$J-1),-10,0), Data$N, Data$J-1)
    Y <- as.indicator.matrix(Data$y)
    W <- ifelse(Y[,-Data$J] == 1, abs(W), W)
    return(c(beta, U, as.vector(W)))}
MyData <- list(J=J, K=K, N=N, PGF=PGF, S=S, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.U=pos.U, pos.W=pos.W,
    y=y)
```

65.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- matrix(parm[Data$pos.beta], Data$J-1, Data$K)
    u <- c(0, parm[Data$pos.U])
    U <- diag(Data$J-1)
    U[upper.tri(U, diag=TRUE)] <- u
    diag(U) <- exp(diag(U))
    Sigma <- t(U) %*% U
    Sigma[1,] <- Sigma[,1] <- U[1,]
    W <- matrix(parm[Data$pos.W], Data$N, Data$J-1)
    Y <- as.indicator.matrix(Data$y)
    temp <- which(Y[,-c(Data$J)] == 1)
    W[temp] <- interval(W[temp], 0, 10)
    temp <- which(Y[,-c(Data$J)] == 0)
    W[temp] <- interval(W[temp], -10, 0)
    parm[Data$pos.W] <- as.vector(W)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 10, log=TRUE))
    U.prior <- sum(dnorm(u[-1], 0, 1, log=TRUE))
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, beta)
    #eta <- exp(cbind(mu,0))
    #p <- eta / rowSums(eta)
    LL <- sum(dmvn(W, mu, Sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + U.prior
```

```
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
     yhat=max.col(cbind(rmvn(nrow(mu), mu, Sigma),0)), parm=parm)
return(Modelout)
}
```

65.4. Initial Values

```
Initial.Values <- GIV(Model, MyData, PGF=TRUE)
```

66. Multiple Discrete-Continuous Choice

This form of a multivariate discrete-continuous choice model was introduced in Kim, Allenby, and Rossi (2002) and referred to as a variety model. The original version is presented with log-normally distributed errors, but a gamma regression form is used here instead, which has always mixed better in testing. Note that the γ parameters are fixed here, as recommended for identifiability in future articles by these authors.

66.1. Form

$$\mathbf{Y} \sim \mathcal{G}(\lambda\tau, \tau)$$

$$\lambda_{i,j} = \exp(\mathbf{Z}_{i,j} \log(\psi 1_{m[i],j}) + \mathbf{X1}_{i,1:K} \log(\beta) + \mathbf{X2}_{i,1:L} \log(\delta))(\mathbf{Y}_{i,j} + \gamma_j)^{\alpha}_j), \quad i = 1, \ldots, N, j = 1, \ldots, J$$

$$\alpha_j \sim \mathcal{U}(0, 1), \quad j = 1, \ldots, J$$

$$\log(\beta_k) \sim \mathcal{N}(0, 1000), \quad k = 1, \ldots, K$$

$$\gamma_j = 1, \quad j = 1, \ldots, J$$

$$\log(\delta_{j,l}) \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, (J-1), \quad l = 1, \ldots, L$$

$$\log(\psi 0_j) \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\log(\psi 1_{g,j}) \sim \mathcal{N}_J(\log(\psi 0), \Omega^{-1}), \quad g = 1, \ldots, G, \quad j1 =, \ldots, J$$

$$\Omega \sim \mathcal{W}_{J+1}(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_J$$

$$\tau_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, J$$

66.2. Data

```
G <- 6 #Number of Multilevel Groups (decision-makers, households, etc.)
J <- 3 #Number of products
K <- 4 #Number of product attributes
L <- 5 #Number of decision-maker attributes
N <- 30 #Number of records
X1 <- matrix(rnorm(N*K), N, K) #Product attributes
X2 <- matrix(rnorm(N*L), N, L) #Decision-maker attributes
Sigma <- matrix(runif((J-1)*(J-1),-1,1),J-1,J-1)
```

```
diag(Sigma) <- runif(J-1,1,5)
Sigma <- as.positive.definite(Sigma) / 100
alpha <- runif(J)
log.beta <- rnorm(K,0,0.1)
log.delta <- matrix(rnorm((J-1)*L,0,0.1), J-1, L)
log.psi0 <- rnorm(J)
log.psi1 <- rmvn(G, log.psi0, Sigma)
m <- rcat(N, rep(1/G,G)) # Multilevel group indicator
Z <- as.indicator.matrix(m)
Y <- matrix(0, N, J)
Y <- round(exp(tcrossprod(Z, t(cbind(log.psi1,0))) +
    matrix(tcrossprod(X1, t(log.beta)), N, J) +
    tcrossprod(X2, rbind(log.delta, colSums(log.delta)*-1))) *
    (Y + 1)^matrix(alpha,N,J,byrow=TRUE) +
    matrix(rnorm(N*J,0,0.1),N,J))
S <- diag(J)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,J), log.beta=rep(0,K),
    log.delta=matrix(0,J-1,L), log.psi0=rep(0,J),
    log.psi1=matrix(0,G,J), tau=rep(0,J), U=S),
    uppertri=c(0,0,0,0,0,0,1))
pos.alpha <- grep("alpha", parm.names)
pos.log.beta <- grep("log.beta", parm.names)
pos.log.delta <- grep("delta", parm.names)
pos.log.psi0 <- grep("log.psi0", parm.names)
pos.log.psi1 <- grep("log.psi1", parm.names)
pos.tau <- grep("tau", parm.names)
PGF <- function(Data) {
    alpha <- runif(Data$J,0.9,1)
    log.beta <- rnorm(Data$K,0,0.1)
    log.delta <- rnorm((Data$J-1)*Data$L,0,0.1)
    log.psi0 <- rnorm(Data$J)
    U <- rwishartc(Data$J+1, Data$S)
    log.psi1 <- as.vector(rmvnpc(Data$G, log.psi0, U))
    tau <- runif(Data$J)
    return(c(alpha, log.beta, log.delta, log.psi0, log.psi1, tau,
        U[upper.tri(U, diag=TRUE)]))
    }
MyData <- list(G=G, J=J, K=K, L=L, N=N, PGF=PGF, S=S, X1=X1, X2=X2, Y=Y,
    Z=Z, m=m, mon.names=mon.names, parm.names=parm.names,
    pos.alpha=pos.alpha, pos.log.beta=pos.log.beta,
    pos.log.delta=pos.log.delta, pos.log.psi0=pos.log.psi0,
    pos.log.psi1=pos.log.psi1, pos.tau=pos.tau)
```

66.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     parm[Data$pos.alpha] <- alpha <- interval(parm[Data$pos.alpha], 0, 1)
     log.beta <- parm[Data$pos.log.beta]
     log.delta <- matrix(parm[Data$pos.log.delta], Data$J-1, Data$L)
     log.psi0 <- parm[Data$pos.log.psi0]
     log.psi1 <- matrix(parm[Data$pos.log.psi1], Data$G, Data$J)
     parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
     U <- as.parm.matrix(U, Data$J, parm, Data, chol=TRUE)
     diag(U) <- exp(diag(U))
     lambda <- tcrossprod(Data$Z, t(log.psi1)) +
          matrix(tcrossprod(Data$X1, t(log.beta)), Data$N, Data$J) +
          tcrossprod(Data$X2, rbind(log.delta, colSums(log.delta)*-1))
     ### Log-Prior
     U.prior <- dwishartc(U, Data$J+1, Data$S, log=TRUE)
     alpha.prior <- sum(dunif(alpha, 0, 1, log=TRUE))
     log.beta.prior <- sum(dnormv(log.beta, 0, 1000, log=TRUE))
     log.delta.prior <- sum(dnormv(log.delta, 0, 1000, log=TRUE))
     log.psi0.prior <- sum(dnormv(log.psi0, 0, 1000, log=TRUE))
     log.psi1.prior <- sum(dmvnpc(lambda,
          matrix(log.psi0, Data$N, Data$J, byrow=TRUE), U, log=TRUE))
     tau.prior <- sum(dhalfcauchy(tau, 25, log=TRUE))
     ### Log-Likelihood
     alpha <- matrix(alpha, Data$N, Data$J, byrow=TRUE)
     lambda <- exp(lambda)*(Data$Y + 1)^alpha
     tau <- matrix(tau, Data$N, Data$J, byrow=TRUE)
     LL <- sum(dgamma(Data$Y+1, lambda*tau, tau, log=TRUE))
     ### Log-Posterior
     LP <- LL + U.prior + alpha.prior + log.beta.prior + log.delta.prior +
          log.psi0.prior + log.psi1.prior + tau.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rgamma(prod(dim(lambda)), lambda*tau, tau)-1,
          parm=parm)
     return(Modelout)
     }
```

66.4. Initial Values

```
Initial.Values <- c(runif(J,0.9,1), rnorm(K,0,0.1),
     rnorm((J-1)*L,0,0.1), rnorm(J,0,0.1),
     rmvnpc(G, rnorm(J,0,0.1), rwishartc(J+1,S)), runif(J),
     upper.triangle(rwishartc(J+1,S), diag=TRUE))
```

67. Multivariate Binary Probit

67.1. Form

$$\mathbf{W}_{i,1:J} \sim \mathcal{N}_J(\mu_{i,1:J}, \Omega^{-1}), \quad i = 1, \ldots, N$$

$$\mathbf{W}_{i,j} \in \begin{cases} [0,10] & \text{if } \mathbf{y}_i = j \\ [-10,0] \end{cases}$$

$$\mu_{1:N,j} = \mathbf{X}\beta_{j,1:K}$$

$$\Omega = \rho^{-1}$$

$$\beta_{j,k} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, (J-1), \quad k = 1, \ldots, K$$

$$\beta_{J,k} = -\sum_{j=1}^{J-1} \beta_{j,k}$$

$$\rho \sim \mathcal{U}(-1, 1)$$

67.2. Data

```
N <- 30
J <- 2 #Number of binary dependent variables
K <- 3 #Number of columns to be in design matrix X
X <- cbind(1, matrix(rnorm(N*(K-1),0,1), N, K-1))
beta <- matrix(rnorm(J*K), J, K)
mu <- tcrossprod(X, beta)
u <- runif(length(which(upper.tri(diag(J)) == TRUE)), -1, 1)
rho <- diag(J)
rho[upper.tri(rho)] <- u
rho[lower.tri(rho)] <- t(rho)[lower.tri(rho)]
rho <- as.positive.semidefinite(rho)
Omega <- as.inverse(rho)
U <- chol(Omega)
W <- interval(rmvnpc(N, mu, U) + matrix(rnorm(N*J,0,0.1), N, J),
    -10, 10)
Y <- 1 * (W >= 0)
apply(Y, 2, table)
mon.names <- "LP"
rho <- matrix(NA, J, J)
rho[upper.tri(rho)] <- 0
parm.names <- as.parm.names(list(beta=matrix(0,J,K), rho=rho,
    W=matrix(0,N,J)))
pos.beta <- grep("beta", parm.names)
pos.rho <- grep("rho", parm.names)
pos.W <- grep("W", parm.names)
PGF <- function(Data) {
```

```
    beta <- rnorm(Data$J*Data$K)
    rho <- rep(0, length(which(upper.tri(diag(Data$J)))))
    W <- matrix(runif(Data$N*Data$J,-10,0), Data$N, Data$J)
    W <- ifelse(Y == 1, abs(W), W)
    return(c(beta, rho, as.vector(W)))}
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, Y=Y,
    mon.names=mon.names, parm.names=parm.names, pos.beta=pos.beta,
    pos.rho=pos.rho, pos.W=pos.W)
```

67.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- matrix(parm[Data$pos.beta], Data$J, Data$K)
    u <- interval(parm[Data$pos.rho], -1, 1)
    rho <- diag(MyData$J)
    rho[upper.tri(rho)] <- u
    rho[lower.tri(rho)] <- t(rho)[lower.tri(rho)]
    if(is.positive.semidefinite(rho) == FALSE)
        rho <- as.positive.semidefinite(rho)
    parm[Data$pos.rho] <- upper.triangle(rho)
    Omega <- as.inverse(rho)
    U <- chol(Omega)
    W <- matrix(parm[Data$pos.W], Data$N, Data$J)
    W[Data$Y == 0] <- interval(W[Data$Y == 0], -10, 0)
    W[Data$Y == 1] <- interval(W[Data$Y == 1], 0, 10)
    parm[Data$pos.W] <- as.vector(W)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    rho.prior <- sum(dunif(u, -1, 1, log=TRUE))
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, beta)
    LL <- sum(dmvnpc(W, mu, U, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + rho.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=1*(rmvnpc(nrow(mu), mu, U) >= 0), parm=parm)
    return(Modelout)
    }
```

67.4. Initial Values

```
Initial.Values <- GIV(Model, MyData, PGF=TRUE)
```

68. Multivariate Laplace Regression

68.1. Form

$$\mathbf{Y}_{i,k} \sim \mathcal{L}_K(\mu_{i,k}, \Sigma), \quad i = 1, \ldots, N; \quad k = 1, \ldots, K$$
$$\mu_{i,k} = \mathbf{X}_{1:N,k}\beta_{k,1:J}$$
$$\Sigma = \Omega^{-1}$$
$$\Omega \sim \mathcal{W}_{K+1}(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_K$$
$$\beta_{k,j} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

68.2. Data

```
data(mtcars)
Y <- as.matrix(mtcars[,c(1,7)])
X <- cbind(1, as.matrix(mtcars[,c(3,4,6)]))
N <- nrow(Y) #Number of records
J <- ncol(X) #Number of columns in design matrix
K <- ncol(Y) #Number of DVs
S <- diag(K)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=matrix(0,K,J), U=diag(K)),
    uppertri=c(0,1))
pos.beta <- grep("beta", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$K*Data$J)
    U <- rwishartc(Data$K+1, Data$S)
    return(c(beta, U[upper.tri(U, diag=TRUE)]))
    }
MyData <- list(J=J, K=K, N=N, PGF=PGF, S=S, X=X, Y=Y, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta)
```

68.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- matrix(parm[Data$pos.beta], Data$K, Data$J)
    U <- as.parm.matrix(U, Data$K, parm, Data, chol=TRUE)
    diag(U) <- exp(diag(U))
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    U.prior <- dwishart(U, Data$K+1, Data$S, log=TRUE)
    ### Log-Likelihood
```

```
    mu <- tcrossprod(Data$X, beta)
    LL <- sum(dmvlc(Data$Y, mu, U, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + U.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rmvlc(nrow(mu), mu, U), parm=parm)
    return(Modelout)
    }
```

68.4. Initial Values

```
Initial.Values <- c(rep(0,J*K), upper.triangle(S, diag=TRUE))
```

69. Multivariate Poisson Regression

69.1. Form

$$\mathbf{Y}_{i,k} \sim \mathcal{P}(\lambda_{i,k}), \quad i = 1, \ldots, N \quad k = 1, \ldots, K$$
$$\lambda_{i,k} = \exp(\mathbf{X}_{i,k}\beta_{k,1:J} + \gamma_{i,k}), \quad i = 1, \ldots, N, \quad k = 1, \ldots, K$$
$$\beta_{k,j} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J, \quad k = 1, \ldots, K$$
$$\gamma_{i,1:K} \sim \mathcal{N}_K(0, \Omega^{-1}), \quad i = 1, \ldots, N$$
$$\Omega \sim \mathcal{W}_{K+1}(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_K$$

69.2. Data

```
N <- 20 #Number of records
J <- 4 #Number of columns in design matrix
K <- 3 #Number of DVs
X <- matrix(runif(N*J),N,J); X[,1] <- 1
beta <- matrix(rnorm(K*J),K,J)
Omega <- matrix(runif(K*K),K,K); diag(Omega) <- runif(K,1,K)
Omega <- as.symmetric.matrix(Omega)
gamma <- rmvnp(N, 0, Omega)
Y <- round(exp(tcrossprod(X, beta) + gamma))
S <- diag(K)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=matrix(0,K,J), gamma=matrix(0,N,K),
    U=S), uppertri=c(0,0,1))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$K*Data$J)
```

```
    gamma <- rnorm(Data$N*Data$K)
    U <- rwishartc(Data$K+1, Data$S)
    return(c(beta, gamma, U[upper.tri(U, diag=TRUE)]))
    }
MyData <- list(J=J, K=K, N=N, PGF=PGF, S=S, X=X, Y=Y, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma)
```

69.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- matrix(parm[Data$pos.beta], Data$K, Data$J)
    gamma <- matrix(parm[Data$pos.gamma], Data$N, Data$K)
    U <- as.parm.matrix(U, Data$K, parm, Data, chol=TRUE)
    diag(U) <- exp(diag(U))
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    gamma.prior <- sum(dmvnpc(gamma, 0, U, log=TRUE))
    U.prior <- dwishartc(U, Data$K+1, Data$S, log=TRUE)
    ### Log-Likelihood
    lambda <- exp(tcrossprod(Data$X, beta) + gamma)
    LL <- sum(dpois(Data$Y, lambda, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + gamma.prior + U.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rpois(prod(dim(lambda)), lambda), parm=parm)
    return(Modelout)
    }
```

69.4. Initial Values

```
Initial.Values <- c(rep(0,K*J), rep(0,N*K), rep(0,K*(K+1)/2))
```

70. Multivariate Regression

70.1. Form

$$\mathbf{Y}_{i,k} \sim \mathcal{N}_K(\mu_{i,k}, \Sigma), \quad i = 1, \ldots, N; \quad k = 1, \ldots, K$$

$$\mu_{i,k} = \mathbf{X}_{1:N,k}\beta_{k,1:J}$$

$$\Sigma \sim \mathcal{HW}_2(\gamma, 1e6)$$

$$\beta_{k,j} \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J, \quad k = 1, \ldots, K$$

70.2. Data

```
data(mtcars)
Y <- as.matrix(mtcars[,c(1,7)])
X <- cbind(1, as.matrix(mtcars[,c(3,4,6)]))
N <- nrow(Y) #Number of records
J <- ncol(X) #Number of columns in design matrix
K <- ncol(Y) #Number of DVs
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=matrix(0,K,J), gamma=rep(0,K),
    U=diag(K)), uppertri=c(0,0,1))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$K*Data$J)
    gamma <- runif(Data$K)
    U <- rhuangwandc(2, gamma, rep(1,Data$K))
    return(c(beta, gamma, U[upper.tri(U, diag=TRUE)]))
    }
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, Y=Y, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma)
```

70.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- matrix(parm[Data$pos.beta], Data$K, Data$J)
    gamma <- interval(parm[Data$pos.gamma], 1e-100, Inf)
    parm[Data$pos.gamma] <- gamma
    U <- as.parm.matrix(U, Data$K, parm, Data, chol=TRUE)
    diag(U) <- exp(diag(U))
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    HW.prior <- dhuangwandc(U, 2, gamma, rep(1e6,Data$K), log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, beta)
    LL <- sum(dmvnc(Data$Y, mu, U, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + U.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rmvnc(nrow(mu), mu, U), parm=parm)
    return(Modelout)
    }
```

70.4. Initial Values

```
Initial.Values <- c(rep(0,J*K), rep(1,K), rep(0,K*(K+1)/2))
```

71. Negative Binomial Regression

This example was contributed by Jim Robison-Cox.

71.1. Form

$$\mathbf{y} \sim \mathcal{NB}(\mu, \kappa)$$
$$p = \frac{\kappa}{\kappa + \mu}$$
$$\mu = \exp(\mathbf{X}\beta)$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\kappa \sim \mathcal{HC}(25) \in (0, \infty]$$

71.2. Data

```
N <- 100
J <- 5 #Number of predictors, including the intercept
kappa.orig <- 2
beta.orig <- runif(J,-2,2)
X <- matrix(runif(J*N,-2, 2), N, J); X[,1] <- 1
mu <- exp(tcrossprod(X, t(beta.orig)) + rnorm(N))
p <- kappa.orig / (kappa.orig + mu)
y <- rnbinom(N, size=kappa.orig, mu=mu)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), kappa=0))
pos.beta <- grep("beta", parm.names)
pos.kappa <- grep("kappa", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    kappa <- runif(1)
    return(c(beta, kappa))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.kappa=pos.kappa, y=y)
```

71.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
```

```
beta <- parm[Data$pos.beta]
parm[Data$J + 1] <- kappa <- interval(parm[Data$pos.kappa],
     .Machine$double.xmin, Inf)
### Log-Prior
beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
kappa.prior <- dhalfcauchy(kappa, 25, log=TRUE)
### Log-Likelihood
mu <- as.vector(exp(tcrossprod(Data$X, t(beta))))
#p <- kappa / (kappa + mu)
LL <- sum(dnbinom(Data$y, size=kappa, mu=mu, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior + kappa.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
     yhat=rnbinom(length(mu), size=kappa, mu=mu), parm=parm)
return(Modelout)
}
```

71.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

72. Normal, Multilevel

This is Gelman's school example (Gelman, Carlin, Stern, and Rubin 2004). Note that **LaplacesDemon** is slower to converge than WinBUGS through the **R2WinBUGS** package (Gelman 2013), an R package on CRAN. This example is very sensitive to the prior distributions. The recommended, default, half-Cauchy priors with scale 25 on scale parameters overwhelms the likelihood, so uniform priors are used.

72.1. Form

$$\begin{aligned}
\mathbf{y}_j &\sim \mathcal{N}(\theta_j, \sigma_j^2), \quad j = 1, \ldots, J \\
\theta_j &\sim \mathcal{N}(\theta_\mu, \theta_\sigma^2) \\
\theta_\mu &\sim \mathcal{N}(0, 1000000) \\
\theta_{\sigma[j]} &\sim \mathcal{N}(0, 1000) \\
\sigma &\sim \mathcal{U}(0, 1000)
\end{aligned}$$

72.2. Data

```
J <- 8
y <- c(28.4, 7.9, -2.8, 6.8, -0.6, 0.6, 18.0, 12.2)
sd <- c(14.9, 10.2, 16.3, 11.0, 9.4, 11.4, 10.4, 17.6)
```

```
mon.names <- "LP"
parm.names <- as.parm.names(list(theta=rep(0,J), theta.mu=0,
    theta.sigma=0))
pos.theta <- 1:J
pos.theta.mu <- J+1
pos.theta.sigma <- J+2
PGF <- function(Data) {
    theta.mu <- rnorm(1)
    theta.sigma <- runif(1)
    theta <- rnorm(Data$J, theta.mu, theta.sigma)
    return(c(theta, theta.mu, theta.sigma))
    }
MyData <- list(J=J, PGF=PGF, mon.names=mon.names, parm.names=parm.names,
    pos.theta=pos.theta, pos.theta.mu=pos.theta.mu,
    pos.theta.sigma=pos.theta.sigma, sd=sd, y=y)
```

72.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperparameters
    theta.mu <- parm[Data$pos.theta.mu]
    theta.sigma <- interval(parm[Data$pos.theta.sigma], 1e-100, Inf)
    parm[Data$pos.theta.sigma] <- theta.sigma
    ### Parameters
    theta <- parm[Data$pos.theta]
    ### Log-Hyperprior
    theta.mu.prior <- dnormp(theta.mu, 0, 1.0E-6, log=TRUE)
    theta.sigma.prior <- dunif(theta.sigma, 0, 1000, log=TRUE)
    ### Log-Prior
    theta.prior <- sum(dnorm(theta, theta.mu, theta.sigma, log=TRUE))
    sigma.prior <- sum(dunif(Data$sd, 0, 1000, log=TRUE))
    ### Log-Likelihood
    LL <- sum(dnorm(Data$y, theta, Data$sd, log=TRUE))
    ### Log-Posterior
    LP <- LL + theta.prior + theta.mu.prior + theta.sigma.prior +
        sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(theta), theta, Data$sd), parm=parm)
    return(Modelout)
    }
```

72.4. Initial Values

```
Initial.Values <- c(rep(mean(y),J), mean(y), 1)
```

73. Ordinal Logit

73.1. Form

$$\mathbf{y}_i \sim \mathcal{CAT}(P_{i,1:J})$$
$$P_{,J} = 1 - Q_{,(J-1)}$$
$$P_{,j} = |Q_{,j} - Q_{,(j-1)}|, \quad j = 2, \ldots, (J-1)$$
$$P_{,1} = Q_{,1}$$
$$Q = \frac{1}{1 + \exp(\mu)}$$
$$\mu_{,j} = \delta_j - \mathbf{X}\beta, \quad \in [-5, 5]$$
$$\beta_k \sim \mathcal{N}(0, 1000), \quad k = 1, \ldots, K$$
$$\delta_j \sim \mathcal{N}(0, 1) \in [(j-1), j] \in [-5, 5], \quad j = 1, \ldots, (J-1)$$

73.2. Data

```
data(demonsnacks)
N <- nrow(demonsnacks)
J <- 3 #Number of categories in y
X <- as.matrix(demonsnacks[,c(1,3:10)])
K <- ncol(demonsnacks) #Number of columns in design matrix X
y <- log(demonsnacks$Calories)
y <- ifelse(y < 4.5669, 1, ifelse(y > 5.5268, 3, 2)) #Discretize
for (k in 1:K) X[,k] <- CenterScale(X[,k])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,K), delta=rep(0,J-1)))
pos.beta <- grep("beta", parm.names)
pos.delta <- grep("delta", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$K)
     delta <- sort(rnorm(Data$J-1))
     return(c(beta, delta))
     }
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.delta=pos.delta, y=y)
```

73.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
```

```
beta <- parm[Data$pos.beta]
delta <- interval(parm[Data$pos.delta], -5, 5)
delta <- sort(delta)
parm[Data$pos.delta] <- delta
### Log-Prior
beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
delta.prior <- sum(dtrunc(delta, "norm", a=-5, b=5, log=TRUE,
    mean=0, sd=1)
### Log-Likelihood
mu <- matrix(delta, Data$N, Data$J-1, byrow=TRUE) -
    matrix(tcrossprod(Data$X, t(beta)), Data$N, Data$J-1)
P <- Q <- invlogit(mu)
P[,-1] <- abs(Q[,-1] - Q[,-(Data$J-1)])
P <- cbind(P, 1 - Q[,(Data$J-1)])
LL <- sum(dcat(Data$y, P, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior + delta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=rcat(nrow(P), P)
    parm=parm)
return(Modelout)
}
```

73.4. Initial Values

```
Initial.Values <- c(rep(0,K), seq(from=-1, to=1, len=(J-1)))
```

74. Ordinal Probit

74.1. Form

$$\mathbf{y}_i \sim \mathcal{CAT}(P_{i,1:J})$$

$$P_{,J} = 1 - Q_{,(J-1)}$$

$$P_{,j} = |Q_{,j} - Q_{,(j-1)}|, \quad j = 2, \ldots, (J-1)$$

$$P_{,1} = Q_{,1}$$

$$Q = \phi(\mu)$$

$$\mu_{,j} = \delta_j - \mathbf{X}\beta, \quad \in [-5, 5]$$

$$\beta_k \sim \mathcal{N}(0, 1000), \quad k = 1, \ldots, K$$

$$\delta_j \sim \mathcal{N}(0,1) \in [(j-1), j] \in [-5, 5], \quad j = 1, \ldots, (J-1)$$

74.2. Data

```
data(demonsnacks)
N <- nrow(demonsnacks)
J <- 3 #Number of categories in y
X <- as.matrix(demonsnacks[,c(1,3:10)])
K <- ncol(demonsnacks) #Number of columns in design matrix X
y <- log(demonsnacks$Calories)
y <- ifelse(y < 4.5669, 1, ifelse(y > 5.5268, 3, 2)) #Discretize
for (k in 1:K) X[,k] <- CenterScale(X[,k])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,K), delta=rep(0,J-1)))
pos.beta <- grep("beta", parm.names)
pos.delta <- grep("delta", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$K)
    delta <- sort(rnorm(Data$J-1))
    return(c(beta, delta))
    }
MyData <- list(J=J, K=K, N=N, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.delta=pos.delta, y=y)
```

74.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    delta <- interval(parm[Data$pos.delta], -5, 5)
    delta <- sort(delta)
    parm[Data$pos.delta] <- delta
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    delta.prior <- sum(dtrunc(delta, "norm", a=-5, b=5, log=TRUE,
        mean=0, sd=1)
    ### Log-Likelihood
    mu <- matrix(delta, Data$N, Data$J-1, byrow=TRUE) -
        matrix(tcrossprod(Data$X, t(beta)), Data$N, Data$J-1)
    P <- Q <- pnorm(mu)
    P[,-1] <- abs(Q[,-1] - Q[,-(Data$J-1)])
    P <- cbind(P, 1 - Q[,(Data$J-1)])
    LL <- sum(dcat(Data$y, P, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + delta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=rcat(nrow(P), P)
        parm=parm)
```

```
    return(Modelout)
    }
```

74.4. Initial Values

```
Initial.Values <- c(rep(0,K), seq(from=-1, to=1, len=(J-1)))
```

75. Panel, Autoregressive Poisson

75.1. Form

$$\mathbf{Y} \sim \mathcal{P}(\Lambda)$$
$$\Lambda_{1:N,1} = \exp(\alpha + \beta\mathbf{x})$$
$$\Lambda_{1:N,t} = \exp(\alpha + \beta\mathbf{x} + \rho \log(\mathbf{Y}_{1:N,t-1})), \quad t = 2, \ldots, T$$
$$\alpha_i \sim \mathcal{N}(\alpha_\mu, \alpha_\sigma^2), \quad i = 1, \ldots, N$$
$$\alpha_\mu \sim \mathcal{N}(0, 1000)$$
$$\alpha_\sigma \sim \mathcal{HC}(25)$$
$$\beta \sim \mathcal{N}(0, 1000)$$
$$\rho \sim \mathcal{N}(0, 1000)$$

75.2. Data

```
N <- 10
T <- 10
alpha <- rnorm(N,2,0.5)
rho <- 0.5
beta <- 0.5
x <- runif(N,0,1)
Y <- matrix(NA,N,T)
Y[,1] <- exp(alpha + beta*x)
for (t in 2:T) {Y[,t] <- exp(alpha + beta*x + rho*log(Y[,t-1]))}
Y <- round(Y)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,N), alpha.mu=0,
    alpha.sigma=0, beta=0, rho=0))
pos.alpha <- 1:N
pos.alpha.mu <- grep("alpha.mu", parm.names)
pos.alpha.sigma <- grep("alpha.sigma", parm.names)
pos.beta <- grep("beta", parm.names)
pos.rho <- grep("rho", parm.names)
```

```
PGF <- function(Data) {
     alpha.mu <- rnorm(1)
     alpha.sigma <- runif(1)
     alpha <- rnorm(Data$N, alpha.mu, alpha.sigma)
     beta <- rnorm(1)
     rho <- rnorm(1)
     return(c(alpha, alpha.mu, alpha.sigma, beta, rho))
     }
MyData <- list(N=N, PGF=PGF, T=T, Y=Y, mon.names=mon.names,
     parm.names=parm.names, pos.alpha=pos.alpha, pos.alpha.mu=pos.alpha.mu,
     pos.alpha.sigma=pos.alpha.sigma, pos.beta=pos.beta, pos.rho=pos.rho,
     x=x)
```

75.3. Model

```
Model <- function(parm, Data)
     {
     ### Hyperparameters
     alpha.mu <- parm[Data$pos.alpha.mu]
     alpha.sigma <- interval(parm[Data$pos.alpha.sigma], 1e-100, Inf)
     parm[Data$pos.alpha.sigma] <- alpha.sigma
     ### Parameters
     alpha <- parm[Data$pos.alpha]
     beta <- parm[Data$pos.beta]
     rho <- parm[Data$pos.rho]
     ### Log-Hyperprior
     alpha.mu.prior <- dnormv(alpha.mu, 0, 1000, log=TRUE)
     alpha.sigma.prior <- dhalfcauchy(alpha.sigma, 25, log=TRUE)
     ### Log-Prior
     alpha.prior <- sum(dnorm(alpha, alpha.mu, alpha.sigma, log=TRUE))
     beta.prior <- dnormv(beta, 0, 1000, log=TRUE)
     rho.prior <- dnormv(rho, 0, 1000, log=TRUE)
     ### Log-Likelihood
     Lambda <- Data$Y
     Lambda[,1] <- exp(alpha + beta*x)
     Lambda[,2:Data$T] <- exp(alpha + beta*Data$x +
          rho*log(Data$Y[,1:(Data$T-1)]))
     LL <- sum(dpois(Data$Y, Lambda, log=TRUE))
     ### Log-Posterior
     LP <- LL + alpha.prior + alpha.mu.prior + alpha.sigma.prior +
          beta.prior + rho.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rpois(prod(dim(Lambda)), parm=parm)
     return(Modelout)
     }
```

75.4. Initial Values

```
Initial.Values <- c(rep(0,N), 0, 1, 0, 0)
```

76. Penalized Spline Regression

This example applies penalized splines to one predictor in a linear regression. The user selects the degree of the polynomial, D, and the number of knots, K.

76.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma_1^2)$$
$$\mu = \mathbf{X}\beta + \mathbf{S}$$
$$\mathbf{S} = \mathbf{Z}\gamma$$
$$\mathbf{Z}_{i,k} = \begin{cases} (\mathbf{x}_i - k)^D & \text{if } \mathbf{Z}_{i,k} > 0 \\ 0 \end{cases}$$
$$\beta_d \sim \mathcal{N}(0, 1000), \quad d = 1, \ldots, (D+1)$$
$$\gamma_k \sim \mathcal{N}(0, \sigma_2^2), \quad k = 1, \ldots, K$$
$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, 2$$

76.2. Data

```
N <- 100
x <- 1:N
y <- sin(2*pi*x/N) + runif(N,-1,1)
K <- 10 #Number of knots
D <- 2 #Degree of polynomial
x <- CenterScale(x)
k <- as.vector(quantile(x, probs=(1:K / (K+1))))
X <- cbind(1, matrix(x, N, D))
for (d in 1:D) {X[,d+1] <- X[,d+1]^d}
Z <- matrix(x, N, K) - matrix(k, N, K, byrow=TRUE)
Z <- ifelse(Z > 0, Z, 0); Z <- Z^D
mon.names <- c("LP", paste("S[", 1:nrow(X) ,"]", sep=""))
parm.names <- as.parm.names(list(beta=rep(0,1+D), gamma=rep(0,K),
    log.sigma=rep(0,2)))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(1+Data$D)
    gamma <- rnorm(Data$K)
    sigma <- runif(2)
```

```
    return(c(beta, gamma, sigma))
    }
MyData <- list(D=D, K=K, N=N, PGF=PGF, Z=Z, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma,
    pos.sigma=pos.sigma, y=y)
```

76.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    gamma <- parm[Data$pos.gamma]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    gamma.prior <- sum(dnorm(gamma, 0, sigma[2], log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    S <- as.vector(tcrossprod(Data$Z, t(gamma)))
    mu <- as.vector(tcrossprod(Data$X, t(beta))) + S
    LL <- sum(dnorm(Data$y, mu, sigma[1], log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + gamma.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,S),
        yhat=rnorm(length(mu), mu, sigma[1]), parm=parm)
    return(Modelout)
    }
```

76.4. Initial Values

```
Initial.Values <- c(rep(0,1+D), rep(0,K), c(1,1))
```

77. Poisson Regression

77.1. Form

$$\mathbf{y} \sim \mathcal{P}(\lambda)$$

$$\lambda = \exp(\mathbf{X}\beta)$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

77.2. Data

```
N <- 10000
J <- 5
X <- matrix(runif(N*J,-2,2),N,J); X[,1] <- 1
beta <- runif(J,-2,2)
y <- round(exp(tcrossprod(X, t(beta))))
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J)))
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    return(beta)
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, y=y)
```

77.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    ### Log-Likelihood
    lambda <- exp(tcrossprod(Data$X, t(beta)))
    LL <- sum(dpois(Data$y, lambda, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rpois(length(lambda), lambda), parm=parm)
    return(Modelout)
    }
```

77.4. Initial Values

```
Initial.Values <- rep(0,J)
```

78. Polynomial Regression

In this univariate example, the degree of the polynomial is specified as D. For a more robust extension to estimating nonlinear relationships between $\mathbf{y}$ and $\mathbf{x}$, see penalized spline regression in section **??**.

78.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \mathbf{X}\beta$$
$$\mathbf{X}_{i,d} = \mathbf{x}_i^{d-1}, \quad d = 1, \ldots, (D+1)$$
$$\mathbf{X}_{i,1} = 1$$
$$\beta_d \sim \mathcal{N}(0, 1000), \quad d = 1, \ldots, (D+1)$$
$$\sigma \sim \mathcal{HC}(25)$$

78.2. Data

```
data(demonsnacks)
N <- nrow(demonsnacks)
D <- 2 #Degree of polynomial
y <- log(demonsnacks$Calories)
x <- log(demonsnacks[,10]+1)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,D+1), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(1+Data$D)
    sigma <- runif(1)
    return(c(beta, sigma))
    }
MyData <- list(D=D, N=N, PGF=PGF, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, x=x,
    y=y)
```

78.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    X <- matrix(Data$x, Data$N, Data$D)
    for (d in 2:Data$D) {X[,d] <- X[,d]^d}
```

```
    X <- cbind(1,X)
    mu <- tcrossprod(X, t(beta))
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
         yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

78.4. Initial Values

```
Initial.Values <- c(rep(0,D+1), 1)
```

79. Proportional Hazards Regression, Weibull

Although the dependent variable is usually denoted as $\mathbf{t}$ in survival analysis, it is denoted here as $\mathbf{y}$ so Laplace's Demon recognizes it as a dependent variable for posterior predictive checks. This example does not support censoring, but it will be included soon.

79.1. Form

$$\mathbf{y}_i \sim \mathcal{WEIB}(\gamma, \mu_i), \quad i = 1, \ldots, N$$
$$\mu = \exp(\mathbf{X}\beta)$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\gamma \sim \mathcal{G}(1, 0.001)$$

79.2. Data

```
N <- 50
J <- 5
X <- matrix(runif(N*J,-2,2),N,J); X[,1] <- 1
beta <- c(1,runif(J-1,-1,1))
y <- round(exp(tcrossprod(X, t(beta)))) + 1 # Undefined at zero
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), gamma=0))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    gamma <- rgamma(1,1e-3)
    return(c(beta, gamma))
    }
```

```
MyData <- list(J=J, N=N, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma, y=y)
```

79.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    gamma <- interval(parm[Data$pos.gamma], 1e-100, Inf)
    parm[Data$pos.gamma] <- gamma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    gamma.prior <- dgamma(gamma, 1, 1.0E-3, log=TRUE)
    ### Log-Likelihood
    mu <- exp(tcrossprod(Data$X, t(beta)))
    LL <- sum(dweibull(Data$y, gamma, mu, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + gamma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rweibull(length(mu), gamma, mu), parm=parm)
    return(Modelout)
    }
```

79.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

80. PVAR(p)

This is a Poisson vector autoregression, with autoregressive order p, for multivariate time-series of counts. It allows for dynamic processes and accounts for overdispersion.

80.1. Form

$$\mathbf{Y}_{t,j} \sim \mathcal{P}(\lambda_{t,j}), \quad t = 1, \ldots, T \quad j = 1, \ldots, J$$

$$\lambda_{t,j} = \sum_{p=1}^{P} \Phi_{1:J,j,p} \mathbf{Y}_{t-p,j} + \exp(\alpha_j + \gamma_{t,j})$$

$$\alpha_j \sim \mathcal{N}(0, 1000)$$

$$\Phi_{i,k,p} \sim \mathcal{N}(\Phi^{\mu}_{i,k,p}, \Sigma_{i,k,p}), \quad i = 1, \ldots, J, \quad k = 1, \ldots, J, \quad p = 1, \ldots, P$$

$$\gamma_{t,1:J} \sim \mathcal{N}_J(0, \Omega^{-1}), \quad t = 1, \ldots, T$$

$$\Omega \sim \mathcal{W}_{J+1}(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_J$$

where Φ^{μ} and Σ are set according to the Minnesota prior.

80.2. Data

```
data(demonsessions)
Y.orig <- as.matrix(demonsessions)
Y <- Y.orig[1:24,1:5]
T <- nrow(Y)
J <- ncol(Y)
L <- c(1,2,3) #Autoregressive lags
P <- length(L) #Autoregressive order
Phi.mu <- array(0, dim=c(J,J,P))
Phi.mu[, , 1] <- diag(J)
S <- diag(J)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,J),
     Phi=array(0, dim=c(J,J,P)), gamma=matrix(0,T-L[P],J), U=S),
     uppertri=c(0,0,0,1))
pos.alpha <- grep("alpha", parm.names)
pos.Phi <- grep("Phi", parm.names)
pos.gamma <- grep("gamma", parm.names)
PGF <- function(Data) {
     alpha <- rnorm(Data$J)
     Phi <- runif(Data$J*Data$J*Data$P, -1e-10, 1e-10)
     gamma <- rnorm((Data$T-Data$L[Data$P])*Data$J)
     U <- rwishartc(Data$J+1, diag(Data$J))
     return(c(alpha, Phi, gamma, U[upper.tri(U, diag=TRUE)]))
     }
MyData <- list(J=J, L=L, P=P, PGF=PGF, Phi.mu=Phi.mu, S=S, T=T, Y=Y,
     mon.names=mon.names, parm.names=parm.names, pos.alpha=pos.alpha,
     pos.Phi=pos.Phi, pos.gamma=pos.gamma)
```

80.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     alpha <- parm[Data$pos.alpha]
     Phi <- array(parm[Data$pos.Phi], dim=c(Data$J, Data$J, Data$P))
     gamma <- matrix(parm[Data$pos.gamma], Data$T-Data$L[Data$P], Data$J)
     U <- as.parm.matrix(U, Data$J, parm, Data, chol=TRUE)
     diag(U) <- exp(diag(U))
     Omega <- t(U) %*% U
     ### Log-Prior
     alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
```

```
    Sigma <- MinnesotaPrior(Data$J, lags=Data$L, lambda=1, theta=0.5,
        diag(as.inverse(Omega)))
    Phi.prior <- sum(dnormv(Phi, Data$Phi.mu, Sigma, log=TRUE))
    gamma.prior <- sum(dmvnp(gamma, 0, Omega, log=TRUE))
    U.prior <- dwishart(Omega, Data$J+1, Data$S, log=TRUE)
    ### Log-Likelihood
    lambda <- exp(matrix(alpha, Data$T, Data$J, byrow=TRUE) +
        rbind(matrix(0, Data$L[Data$P], Data$J), gamma))
    for (p in 1:Data$P)
        lambda[(1+Data$L[p]):Data$T,] <- lambda[(1+Data$L[p]):Data$T,] +
            Data$Y[1:(Data$T-Data$L[p]),]     LL <- sum(dpois(Data$Y[(1+Data$L[Data$P]):D
        lambda[(1+Data$L[Data$P]):Data$T,], log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + Phi.prior + gamma.prior + U.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rpois(prod(dim(lambda)), lambda), parm=parm)
    return(Modelout)
    }
```

80.4. Initial Values

```
Initial.Values <- c(rep(0,J), rep(0,J*J*P), rep(0,(T-L[P])*J),
    rep(0,J*(J+1)/2))
```

81. Quantile Regression

81.1. Form

$$\mathbf{y} \sim \mathcal{N}(\phi, \sigma^2)$$
$$\phi = \frac{(1-2P)}{P(1-P)}\zeta + \mu$$
$$\mu = \mathbf{X}\beta$$
$$\sigma = \frac{P(1-P)\tau}{2\zeta}$$
$$\beta \sim \mathcal{N}(0, 1000)$$
$$\tau \sim \mathcal{HC}(25)$$
$$\zeta \sim \mathcal{EXP}(\tau)$$

where P is the user-specified quantile in $(0, 1)$.

81.2. Data

```
data(demonsnacks)
y <- log(demonsnacks$Calories)
```

```
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
N <- nrow(X)
J <- ncol(X)
P <- 0.5 #Quantile in (0,1)
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), tau=0, zeta=rep(0,N)))
pos.beta <- grep("beta", parm.names)
pos.tau <- grep("tau", parm.names)
pos.zeta <- grep("zeta", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     tau <- runif(1)
     zeta <- rexp(Data$N, tau)
     return(c(beta, tau, zeta))
     }
MyData <- list(J=J, N=N, P=P, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.tau=pos.tau,
     pos.zeta=pos.zeta, y=y)
```

81.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
     zeta <- interval(parm[Data$pos.zeta], 1e-100, Inf)
     parm[Data$pos.zeta] <- zeta
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     tau.prior <- dhalfcauchy(tau, 25, log=TRUE)
     zeta.prior <- sum(dexp(zeta, tau, log=TRUE))
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     phi <- (1 - 2*Data$P) / (Data$P*(1 - Data$P))*zeta + mu
     sigma <- (Data$P*(1 - Data$P)*tau) / (2*zeta)
     LL <- sum(dnorm(Data$y, phi, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + tau.prior + zeta.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(phi), phi, sigma), parm=parm)
     return(Modelout)
     }
```

81.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1, rep(1,N))
```

82. Revision, Normal

This example provides both an analytic solution and numerical approximation of the revision of a normal distribution. Given a normal prior distribution (α) and data distribution (β), the posterior (γ) is the revised normal distribution. This is an introductory example of Bayesian inference, and allows the user to experiment with numerical approximation, such as with MCMC in `LaplacesDemon`. Note that, regardless of the data sample size N in this example, Laplace Approximation is inappropriate due to asymptotics since the data (β) is perceived by the algorithm as a single datum rather than a collection of data. MCMC, on the other hand, is biased only by the effective number of samples taken of the posterior.

```
### Analytic Solution
prior.mu <- 0
prior.sigma <- 10
N <- 10
data.mu <- 1
data.sigma <- 2
posterior.mu <- (prior.sigma^-2 * prior.mu + N * data.sigma^-2 * data.mu) /
     (prior.sigma^-2 + N * data.sigma^-2)
posterior.sigma <- sqrt(1/(prior.sigma^-2 + data.sigma^-2))
posterior.mu
posterior.sigma
```

82.1. Form

$$\alpha \sim \mathcal{N}(0, 10)$$

$$\beta \sim \mathcal{N}(1, 2)$$

$$\gamma = \frac{\alpha_\sigma^{-2}\alpha + N\beta_\sigma^{-2}\beta}{\alpha_\sigma^{-2} + N\beta_\sigma^{-2}}$$

82.2. Data

```
N <- 10
mon.names <- c("LP","gamma")
parm.names <- c("alpha","beta")
PGF <- function(Data) {
     alpha <- rnorm(1,0,10)
     beta <- rnorm(1,1,2)
     return(c(alpha, beta))
     }
```

```
MyData <- list(N=N, PGF=PGF, mon.names=mon.names, parm.names=parm.names)
```

82.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperparameters
    alpha.mu <- 0
    alpha.sigma <- 10
    beta.mu <- 1
    beta.sigma <- 2
    ### Parameters
    alpha <- parm[1]
    beta <- parm[2]
    ### Log-Prior
    alpha.prior <- dnorm(alpha, alpha.mu, alpha.sigma, log=TRUE)
    ### Log-Likelihood
    LL <- dnorm(beta, beta.mu, beta.sigma, log=TRUE)
    ### Posterior
    gamma <- (alpha.sigma^-2 * alpha + N * beta.sigma^-2 * beta) /
        (alpha.sigma^-2 + N * beta.sigma^-2)
    ### Log-Posterior
    LP <- LL + alpha.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,gamma),
        yhat=rnorm(1, beta.mu, beta.sigma), parm=parm)
    return(Modelout)
    }
```

82.4. Initial Values

```
Initial.Values <- c(0,0)
```

83. Ridge Regression

83.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma_1^2)$$
$$\mu = \mathbf{X}\beta$$
$$\beta_1 \sim \mathcal{N}(0, 1000)$$
$$\beta_j \sim \mathcal{N}(0, \sigma_2^2), \quad j = 2, \ldots, J$$
$$\sigma_k \sim \mathcal{HC}(25), \quad k = 1, \ldots, 2$$

83.2. Data

```
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,-2]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=rep(0,2)))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    sigma <- runif(2)
    return(c(beta, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)
```

83.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnorm(beta, 0, c(1000, rep(sigma[2], Data$J-1)),
        log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- sum(dnorm(Data$y, mu, sigma[1], log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma[1]), parm=parm)
    return(Modelout)
    }
```

83.4. Initial Values

```
Initial.Values <- c(rep(1,J), rep(1,2))
```

84. Robust Regression

By replacing the normal distribution with the Student t distribution, linear regression is often called robust regression. As an alternative approach to robust regression, consider Laplace regression (see section 46).

84.1. Form

$$\mathbf{y} \sim \mathrm{t}(\mu, \sigma^2, \nu)$$
$$\mu = \mathbf{X}\beta$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\sigma \sim \mathcal{HC}(25)$$
$$\nu \sim \mathcal{HC}(25)$$

84.2. Data

```
N <- 100
J <- 5
X <- matrix(1,N,J)
for (j in 2:J) {X[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))}
beta <- runif(J,-3,3)
e <- rst(N,0,1,5)
y <- tcrossprod(X, t(beta)) + e
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0, nu=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.nu <- grep("nu", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     nu <- runif(1)
     return(c(beta, sigma, nu))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma,
     pos.nu=pos.nu, y=y)
```

84.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
```

```
beta <- parm[1:Data$J]
sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
parm[Data$pos.sigma] <- sigma
parm[Data$pos.nu] <- nu <- interval(parm[Data$pos.nu], 1e-100, Inf)
### Log-Prior
beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
nu.prior <- dhalfcauchy(nu, 25, log=TRUE)
### Log-Likelihood
mu <- tcrossprod(Data$X, t(beta))
LL <- sum(dst(Data$y, mu, sigma, nu, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior + sigma.prior + nu.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rst(length(mu), mu, sigma, nu), parm=parm)
return(Modelout)
}
```

84.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1, 5)
```

85. Seemingly Unrelated Regression (SUR)

The following data was used by Zellner (1962) when introducing the Seemingly Unrelated Regression methodology. This model uses the Yang-Berger prior distribution for the precision matrix of a multivariate normal distribution.

85.1. Form

$$\begin{aligned}
\mathbf{Y}_{t,k} &\sim \mathcal{N}_K(\mu_{t,k}, \Omega^{-1}), \quad t = 1, \ldots, T, \quad k = 1, \ldots, K \\
\mu_{1,t} &= \alpha_1 + \alpha_2 \mathbf{X}_{t-1,1} + \alpha_3 \mathbf{X}_{t-1,2}, \quad t = 2, \ldots, T \\
\mu_{2,t} &= \beta_1 + \beta_2 \mathbf{X}_{t-1,3} + \beta_3 \mathbf{X}_{t-1,4}, \quad t = 2, \ldots, T \\
\alpha_j &\sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J \\
\beta_j &\sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J
\end{aligned}$$

where $J = 3$, $K = 2$, and $T = 20$.

85.2. Data

```
T <- 20 #Time-periods
year <- c(1935,1936,1937,1938,1939,1940,1941,1942,1943,1944,1945,1946,
    1947,1948,1949,1950,1951,1952,1953,1954)
IG <- c(33.1,45.0,77.2,44.6,48.1,74.4,113.0,91.9,61.3,56.8,93.6,159.9,
```

```
    147.2,146.3,98.3,93.5,135.2,157.3,179.5,189.6)
VG <- c(1170.6,2015.8,2803.3,2039.7,2256.2,2132.2,1834.1,1588.0,1749.4,
    1687.2,2007.7,2208.3,1656.7,1604.4,1431.8,1610.5,1819.4,2079.7,
    2371.6,2759.9)
CG <- c(97.8,104.4,118.0,156.2,172.6,186.6,220.9,287.8,319.9,321.3,319.6,
    346.0,456.4,543.4,618.3,647.4,671.3,726.1,800.3,888.9)
IW <- c(12.93,25.90,35.05,22.89,18.84,28.57,48.51,43.34,37.02,37.81,
    39.27,53.46,55.56,49.56,32.04,32.24,54.38,71.78,90.08,68.60)
VW <- c(191.5,516.0,729.0,560.4,519.9,628.5,537.1,561.2,617.2,626.7,
    737.2,760.5,581.4,662.3,583.8,635.2,723.8,864.1,1193.5,1188.9)
CW <- c(1.8,0.8,7.4,18.1,23.5,26.5,36.2,60.8,84.4,91.2,92.4,86.0,111.1,
    130.6,141.8,136.7,129.7,145.5,174.8,213.5)
J <- 2 #Number of dependent variables
Y <- matrix(c(IG,IW), T, J)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,3), beta=rep(0,3),
    U=diag(J)), uppertri=c(0,0,1))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(3)
    beta <- rnorm(3)
    U <- runif(Data$J*(Data$J+1)/2)
    return(c(alpha, beta, U))
    }
MyData <- list(J=J, PGF=PGF, T=T, Y=Y, CG=CG, CW=CW, IG=IG, IW=IW,
    VG=VG, VW=VW, mon.names=mon.names, parm.names=parm.names,
    pos.alpha=pos.alpha, pos.beta=pos.beta)
```

85.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    beta <- parm[Data$pos.beta]
    U <- as.parm.matrix(U, Data$J, parm, Data, chol=TRUE)
    diag(U) <- exp(diag(U))
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    U.prior <- dyangbergerc(U, log=TRUE)
    ### Log-Likelihood
    mu <- Data$Y
    mu[-1,1] <- alpha[1] + alpha[2]*Data$CG[-Data$T] +
        alpha[3]*Data$VG[-Data$T]
```

```
mu[-1,2] <- beta[1] + beta[2]*Data$CW[-Data$T] +
     beta[3]*Data$VW[-Data$T]
LL <- sum(dmvnpc(Data$Y[-1,], mu[-1,], U, log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + beta.prior + U.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
     yhat=rmvnpc(nrow(mu), mu, U), parm=parm)
return(Modelout)
}
```

85.4. Initial Values

```
Initial.Values <- c(rep(0,3), rep(0,3), rep(0,J*(J+1)/2))
```

86. Simultaneous Equations

This example of simultaneous equations uses Klein's Model I (Kleine 1950) regarding economic fluctations in the United States in 1920-1941 (**N**=22). Usually, this example is modeled with 3-stage least sqaures (3SLS), excluding the uncertainty from multiple stages. By constraining each element in the instrumental variables matrix $\nu \in [-10, 10]$, this example estimates the model without resorting to stages. The dependent variable is matrix $\mathbf{Y}$, in which $\mathbf{Y}_{1,1:N}$ is **C** or Consumption, $\mathbf{Y}_{2,1:N}$ is **I** or Investment, and $\mathbf{Y}_{3,1:N}$ is **Wp** or Private Wages. Here is a data dictionary:

```
A = Time Trend measured as years from 1931
C = Consumption
G = Government Nonwage Spending
I = Investment
K = Capital Stock
P = Private (Corporate) Profits
T = Indirect Business Taxes Plus Neg Exports
Wg = Government Wage Bill
Wp = Private Wages
X = Equilibrium Demand (GNP)
```

See Kleine (1950) for more information.

86.1. Form

$$\mathbf{Y} \sim \mathcal{N}_3(\mu, \Omega^{-1})$$

$$\mu_{1,1} = \alpha_1 + \alpha_2\nu_{1,1} + \alpha_4\nu_{2,1}$$

$$\mu_{1,i} = \alpha_1 + \alpha_2\nu_{1,i} + \alpha_3\mathbf{P}_{i-1} + \alpha_4\nu_{2,i}, \quad i = 2, \ldots, N$$

$$\mu_{2,1} = \beta_1 + \beta_2\nu_{1,1} + \beta_4\mathbf{K}_1$$

$$\mu_{2,i} = \beta_1 + \beta_2\nu_{1,i} + \beta_3\mathbf{P}_{i-1} + \beta_4\mathbf{K}_i, \quad i = 2, \ldots, N$$

$$\mu_{3,1} = \gamma_1 + \gamma_2\nu_{3,1} + \gamma_4\mathbf{A}_1$$

$$\mu_{3,i} = \gamma_1 + \gamma_2\nu_{3,i} + \gamma_3\mathbf{X}_{i-1} + \gamma_4\mathbf{A}_i, \quad i = 2, \ldots, N$$

$$\mathbf{Z}_{j,i} \sim \mathcal{N}(\nu_{j,i}, \sigma_j^2), \quad j = 1, \ldots, 3$$

$$\nu_{j,1} = \pi_{j,1} + \pi_{j,3}\mathbf{K}_1 + \pi_{j,5}\mathbf{A}_1 + \pi_{j,6}\mathbf{T}_1 + \pi_{j,7}\mathbf{G}_1, \quad j = 1, \ldots, 3$$

$$\nu_{j,i} = \pi_{j,1} + \pi_{j,2}\mathbf{P}_{i-1} + \pi_{j,3}\mathbf{K}_i + \pi_{j,4}\mathbf{X}_{i-1} + \pi_{j,5}\mathbf{A}_i + \pi_{j,6}\mathbf{T}_i + \pi\mathbf{G}_i, \quad i = 1, \ldots, N, \quad j = 1, \ldots, 3$$

$$\alpha_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, 4$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, 4$$

$$\gamma_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, 4$$

$$\pi_{j,i} \sim \mathcal{N}(0, 1000) \in [-10, 10], \quad j = 1, \ldots, 3, \quad i = 1, \ldots, N$$

$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, 3$$

$$\Omega \sim \mathcal{W}_4(\mathbf{S}), \quad \mathbf{S} = \mathbf{I}_3$$

86.2. Data

```
N <- 22
A <- c(-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10)
C <- c(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,45.6,46.5,
    48.7,51.3,57.7,58.7,57.5,61.6,65,69.7)
G <- c(2.4,3.9,3.2,2.8,3.5,3.3,3.3,4,4.2,4.1,5.2,5.9,4.9,3.7,4,4.4,2.9,4.3,
    5.3,6.6,7.4,13.8)
I <- c(2.7,-0.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,2.1,2,
    -1.9,1.3,3.3,4.9)
K <- c(180.1,182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
    216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,201.2,204.5)
P <- c(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,7,11.2,
    12.3,14,17.6,17.3,15.3,19,21.1,23.5)
T <- c(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,8.3,6.7,
    7.4,8.9,9.6,11.6)
Wg <- c(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,7.4,
    6.7,7.7,7.8,8,8.5)
Wp <- c(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,29,28.5,
    30.6,33.2,36.8,41,38.2,41.6,45,53.3)
X <- c(44.9,45.6,50.1,57.2,57.1,61,64,64.4,64.5,67,61.2,53.4,44.3,45.1,
    49.7,54.4,62.7,65,60.9,69.5,75.7,88.4)
year <- c(1920,1921,1922,1923,1924,1925,1926,1927,1928,1929,1930,1931,1932,
    1933,1934,1935,1936,1937,1938,1939,1940,1941)
Y <- matrix(c(C,I,Wp),3,N, byrow=TRUE)
Z <- matrix(c(P, Wp+Wg, X), 3, N, byrow=TRUE)
S <- diag(nrow(Y))
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,4), beta=rep(0,4),
    gamma=rep(0,4), pi=matrix(0,3,7), sigma=rep(0,3),
    U=diag(3)), uppertri=c(0,0,0,0,0,1))
```

```
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.pi <- grep("pi", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     alpha <- rnorm(4)
     beta <- rnorm(4)
     gamma <- rnorm(4)
     pi <- rnorm(3*7)
     sigma <- runif(3)
     U <- rwishartc(ncol(Data$Y)+1, Data$S)
     return(c(alpha, beta, gamma, pi, sigma, U[upper.tri(U, diag=TRUE)]))
     }
MyData <- list(A=A, C=C, G=G, I=I, K=K, N=N, P=P, PGF=PGF, S=S, T=T, Wg=Wg,
     Wp=Wp, X=X, Y=Y, Z=Z, mon.names=mon.names, parm.names=parm.names,
     pos.alpha=pos.alpha, pos.beta=pos.beta, pos.gamma=pos.gamma,
     pos.pi=pos.pi, pos.sigma=pos.sigma)
```

86.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     alpha <- parm[Data$pos.alpha]
     beta <- parm[Data$pos.beta]
     gamma <- parm[Data$pos.gamma]
     parm[Data$pos.pi] <- pi <- interval(parm[Data$pos.pi], -10, 10)
     pi <- matrix(pi, 3, 7)      sigma <- interval(parm[Data$pos.sigma], 1e-100,
Inf)
     parm[Data$pos.sigma] <- sigma      U <- as.parm.matrix(U, nrow(Data$S), parm,
Data, chol=TRUE)
     parm[grep("Omega", Data$parm.names)] <- upper.triangle(Omega,
          diag=TRUE)
     diag(U) <- exp(diag(U))
     ### Log-Prior
     alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     gamma.prior <- sum(dnormv(gamma, 0, 1000, log=TRUE))
     pi.prior <- sum(dnormv(pi, 0, 1000, log=TRUE))
     sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
     U.prior <- dwishartc(U, nrow(Data$S)+1, Data$S, log=TRUE)
     ### Log-Likelihood
     mu <- nu <- matrix(0,3,Data$N)
     for (i in 1:3) {
          nu[i,1] <- pi[i,1] + pi[i,3]*Data$K[1] + pi[i,5]*Data$A[1] +
```

```
            pi[i,6]*Data$T[1] + pi[i,7]*Data$G[1]
        nu[i,-1] <- pi[i,1] + pi[i,2]*Data$P[-Data$N] +
            pi[i,3]*Data$K[-1] + pi[i,4]*Data$X[-Data$N] +
            pi[i,5]*Data$A[-1] + pi[i,6]*Data$T[-1] +
            pi[i,7]*Data$G[-1]}
    LL <- sum(dnorm(Data$Z, nu, matrix(sigma, 3, Data$N), log=TRUE))
    mu[1,1] <- alpha[1] + alpha[2]*nu[1,1] + alpha[4]*nu[2,1]
    mu[1,-1] <- alpha[1] + alpha[2]*nu[1,-1] +
        alpha[3]*Data$P[-Data$N] + alpha[4]*nu[2,-1]
    mu[2,1] <- beta[1] + beta[2]*nu[1,1] + beta[4]*Data$K[1]
    mu[2,-1] <- beta[1] + beta[2]*nu[1,-1] +
        beta[3]*Data$P[-Data$N] + beta[4]*Data$K[-1]
    mu[3,1] <- gamma[1] + gamma[2]*nu[3,1] + gamma[4]*Data$A[1]
    mu[3,-1] <- gamma[1] + gamma[2]*nu[3,-1] +
        gamma[3]*Data$X[-Data$N] + gamma[4]*Data$A[-1]
    LL <- LL + sum(dmvnpc(t(Data$Y), t(mu), U, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + beta.prior + gamma.prior + pi.prior +
        sigma.prior + U.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=t(rmvnp(ncol(mu), t(mu), U)), parm=parm)
    return(Modelout)
    }
```

86.4. Initial Values

```
Initial.Values <- c(rep(0,4), rep(0,4), rep(0,4), rep(0,3*7), rep(1,3),
    upper.triangle(S, diag=TRUE))
```

87. Space-Time, Dynamic

This approach to space-time or spatiotemporal modeling applies kriging to a stationary spatial component for points in space $s = 1, \ldots, S$ first at time $t = 1$, where space is continuous and time is discrete. Vector ζ contains these spatial effects. Next, SSM (State Space Model) or DLM (Dynamic Linear Model) components are applied to the spatial parameters (ϕ, κ, and λ) and regression effects (β). These parameters are allowed to vary dynamically with time $t = 2, \ldots, T$, and the resulting spatial process is estimated for each of these time-periods. When time is discrete, a dynamic space-time process can be applied. The matrix Θ contains the dynamically varying stationary spatial effects, or space-time effects. Spatial coordinates are given in longitude and latitude for $s = 1, \ldots, S$ points in space and measurements are taken across discrete time-periods $t = 1, \ldots, T$ for $\mathbf{Y}_{s,t}$. The dependent variable is also a function of design matrix $\mathbf{X}$ (which may also be dynamic, but is static in this example) and dynamic regression effects matrix $\beta_{1:K,1:T}$. For more information on kriging, see section 44. For more information on SSMs or DLMs, see section 92. To extend this to a large spatial data set, consider incorporating the predictive process kriging example in section 45.

87.1. Form

$$\mathbf{Y}_{s,t} \sim \mathcal{N}(\mu_{s,t}, \sigma_1^2), \quad s = 1, \ldots, S, \quad t = 1, \ldots, T$$

$$\mu_{s,t} = \mathbf{X}_{s,1:K}\beta_{1:K,t} + \Theta_{s,t}$$

$$\Theta_{s,t} = \frac{\Sigma_{s,s,t}}{\sum_{r=1}^{S}\Sigma_{r,s,t}}\Theta_{s,t-1}, \quad s = 1, \ldots, S, \quad t = 2, \ldots, T$$

$$\Theta_{s,1} = \zeta_s$$

$$\zeta \sim \mathcal{N}_S(0, \Sigma_{1:S,1:S,1})$$

$$\Sigma_{1:S,1:S,t} = \lambda_t^2 \exp(-\phi_t \mathbf{D})^{\kappa[t]}$$

$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, 4$$

$$\beta_{k,1} \sim \mathcal{N}(0, 1000), \quad k = 1, \ldots, K$$

$$\beta_{k,t} \sim \mathcal{N}(\beta_{k,t-1}, \tau_k^2), \quad k = 1, \ldots, K, \quad t = 2, \ldots, T$$

$$\phi_1 \sim \mathcal{HN}(1000)$$

$$\phi_t \sim \mathcal{N}(\phi_{t-1}, \sigma_2^2) \in [0, \infty], \quad t = 2, \ldots, T$$

$$\kappa_1 \sim \mathcal{HN}(1000)$$

$$\kappa_t \sim \mathcal{N}(\kappa_{t-1}, \sigma_3^2) \in [0, \infty], \quad t = 2, \ldots, T$$

$$\lambda_1 \sim \mathcal{HN}(1000)$$

$$\lambda_t \sim \mathcal{N}(\lambda_{t-1}, \sigma_4^2) \in [0, \infty], \quad t = 2, \ldots, T$$

87.2. Data

```
data(demontexas)
Y <- as.matrix(demontexas[1:20,c(18:30)])
X <- cbind(1,as.matrix(demontexas[1:20,c(1,4)])) #Static predictors
latitude <- demontexas[1:20,2]
longitude <- demontexas[1:20,3]
D <- as.matrix(dist(cbind(longitude,latitude), diag=TRUE, upper=TRUE))
S <- nrow(Y) #Number of sites, or points in space
T <- ncol(Y) #Number of time-periods
K <- ncol(X) #Number of columns in design matrix X including the intercept
mon.names <- "LP"
parm.names <- as.parm.names(list(zeta=rep(0,S), beta=matrix(0,K,T),
    phi=rep(0,T), kappa=rep(0,T), lambda=rep(0,T), sigma=rep(0,4),
    tau=rep(0,K)))
pos.zeta <- grep("zeta", parm.names)
pos.beta <- grep("beta", parm.names)
pos.phi <- grep("phi", parm.names)
pos.kappa <- grep("kappa", parm.names)
pos.lambda <- grep("lambda", parm.names)
pos.sigma <- grep("sigma", parm.names)
```

```
pos.tau <- grep("tau", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$K*Data$T, rbind(mean(Data$Y),
          matrix(0, Data$K-1, Data$T)), 1)
     phi <- rhalfnorm(Data$T, 1)
     kappa <- rhalfnorm(Data$T, 1)
     lambda <- rhalfnorm(Data$T, 1)
     Sigma <- lambda[1]*lambda[1]*exp(-phi[1]*Data$D)^kappa[1]
     zeta <- as.vector(rmvn(1, rep(0,Data$S), Sigma))
     sigma <- runif(4)
     tau <- runif(Data$K)
     return(c(zeta, beta, phi, kappa, lambda, sigma, tau))
     }
MyData <- list(D=D, K=K, PGF=PGF, S=S, T=T, X=X, Y=Y, latitude=latitude,
     longitude=longitude, mon.names=mon.names, parm.names=parm.names,
     pos.zeta=pos.zeta, pos.beta=pos.beta, pos.phi=pos.phi,
     pos.kappa=pos.kappa, pos.lambda=pos.lambda, pos.sigma=pos.sigma,
     pos.tau=pos.tau)
```

87.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- matrix(parm[Data$pos.beta], Data$K, Data$T)
     zeta <- parm[Data$pos.zeta]
     parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], 1e-100, Inf)
     kappa <- interval(parm[Data$pos.kappa], 1e-100, Inf)
     parm[Data$pos.kappa] <- kappa
     lambda <- interval(parm[Data$pos.lambda], 1e-100, Inf)
     parm[Data$pos.lambda] <- lambda
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
     Sigma <- array(0, dim=c(Data$S, Data$S, Data$T))
     for (t in 1:Data$T) {
          Sigma[ , ,t] <- lambda[t]^2 * exp(-phi[t] * Data$D)^kappa[t]}
     ### Log-Prior
     beta.prior <- sum(dnormv(beta[,1], 0, 1000, log=TRUE),
          dnorm(beta[,-1], beta[,-Data$T], matrix(tau, Data$K,
          Data$T-1), log=TRUE))
     zeta.prior <- dmvn(zeta, rep(0,Data$S), Sigma[ , , 1], log=TRUE)
     phi.prior <- sum(dhalfnorm(phi[1], sqrt(1000), log=TRUE),
          dtrunc(phi[-1], "norm", a=0, b=Inf, mean=phi[-Data$T],
          sd=sigma[2], log=TRUE))
     kappa.prior <- sum(dhalfnorm(kappa[1], sqrt(1000), log=TRUE),
```

```
    dtrunc(kappa[-1], "norm", a=0, b=Inf, mean=kappa[-Data$T],
    sd=sigma[3], log=TRUE))
lambda.prior <- sum(dhalfnorm(lambda[1], sqrt(1000), log=TRUE),
    dtrunc(lambda[-1], "norm", a=0, b=Inf, mean=lambda[-Data$T],
    sd=sigma[4], log=TRUE))
sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
tau.prior <- sum(dhalfcauchy(tau, 25, log=TRUE))
### Log-Likelihood
mu <- tcrossprod(Data$X, t(beta))
Theta <- matrix(zeta, Data$S, Data$T)
for (t in 2:Data$T) {
    for (s in 1:Data$S) {
        Theta[s,t] <- sum(Sigma[,s,t] / sum(Sigma[,s,t]) * Theta[,t-1])}}
mu <- mu + Theta
LL <- sum(dnorm(Data$Y, mu, sigma[1], log=TRUE))
### Log-Posterior
LP <- LL + beta.prior + zeta.prior + sum(phi.prior) +
    sum(kappa.prior) + sum(lambda.prior) + sigma.prior + tau.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnorm(prod(dim(mu)), mu, sigma[1]), parm=parm)
return(Modelout)
}
```

87.4. Initial Values

```
Initial.Values <- c(rep(0,S), rep(c(mean(Y),rep(0,K-1)),T), rep(1,T),
    rep(1,T), rep(1,T), rep(1,4), rep(1,K))
```

88. Space-Time, Nonseparable

This approach to space-time or spatiotemporal modeling applies kriging both to the stationary spatial and temporal components, where space is continuous and time is discrete. Matrix Ξ contains the space-time effects. Spatial coordinates are given in longitude and latitude for $s = 1, \ldots, S$ points in space and measurements are taken across time-periods $t = 1, \ldots, T$ for $\mathbf{Y}_{s,t}$. The dependent variable is also a function of design matrix $\mathbf{X}$ and regression effects vector β. For more information on kriging, see section 44. This example uses a nonseparable, stationary covariance function in which space and time are separable only when $\psi = 0$. To extend this to a large space-time data set, consider incorporating the predictive process kriging example in section 45.

88.1. Form

$$\mathbf{Y}_{s,t} \sim \mathcal{N}(\mu_{s,t}, \sigma_1^2), \quad s = 1, \ldots, S, \quad t = 1, \ldots, T$$

$$\mu = \mathbf{X}\beta + \Xi$$

$$\Xi \sim \mathcal{N}_{ST}(\Xi_\mu, \Sigma)$$

$$\Sigma = \sigma_2^2 \exp\left(-\frac{\mathbf{D}_S}{\phi_1}^\kappa - \frac{\mathbf{D}_T}{\phi_2}^\lambda - \psi\frac{\mathbf{D}_S}{\phi_1}^\kappa\frac{\mathbf{D}_T}{\phi_2}^\lambda\right)$$

$$\beta_k \sim \mathcal{N}(0, 1000), \quad k = 1, \ldots, K$$

$$\phi_j \sim \mathcal{U}(1, 5), \quad j = 1, \ldots, 2$$

$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, 2$$

$$\psi \sim \mathcal{HC}(25)$$

$$\Xi_\mu = 0$$

$$\kappa = 1, \quad \lambda = 1$$

88.2. Data

```
data(demontexas)
Y <- as.matrix(demontexas[1:10,c(18:30)])
X <- cbind(1,as.matrix(demontexas[1:10,c(1,4)])) #Static predictors
latitude <- demontexas[1:10,2]
longitude <- demontexas[1:10,3]
S <- nrow(Y) #Number of sites, or points in space
T <- ncol(Y) #Number of time-periods
K <- ncol(X) #Number of columns in design matrix X including the intercept
D.S <- as.matrix(dist(cbind(rep(longitude,T),rep(latitude,T)), diag=TRUE,
    upper=TRUE))
D.T <- as.matrix(dist(cbind(rep(1:T,each=S),rep(1:T,each=S)), diag=TRUE,
    upper=TRUE))
mon.names <- "LP"
parm.names <- as.parm.names(list(Xi=matrix(0,S,T), beta=rep(0,K),
    phi=rep(0,2), sigma=rep(0,2), psi=0))
pos.Xi <- grep("Xi", parm.names)
pos.beta <- grep("beta", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.psi <- grep("psi", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$K, c(mean(Data$Y),rep(0,Data$K-1)), 1)
    phi <- runif(2,1,5)
    sigma <- runif(2)
    psi <- runif(1)
    kappa <- 1
    lambda <- 1
    Sigma <- sigma[2]*sigma[2] * exp(-(Data$D.S / phi[1])^kappa -          (Data$D.T
/ phi[2])^lambda -          psi*(Data$D.S / phi[1])^kappa * (Data$D.T / phi[2])^lambda)
    Xi <- as.vector(rmvn(1, rep(0,Data$S*Data$T), Sigma))
    return(c(Xi, beta, phi, sigma, psi))
```

```
    }
MyData <- list(D.S=D.S, D.T=D.T, K=K, PGF=PGF, S=S, T=T, X=X, Y=Y,
    latitude=latitude, longitude=longitude, mon.names=mon.names,
    parm.names=parm.names, pos.Xi=pos.Xi, pos.beta=pos.beta,
    pos.phi=pos.phi, pos.sigma=pos.sigma, pos.psi=pos.psi)
```

88.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperparameters
    Xi.mu <- rep(0,Data$S*Data$T)
    ### Parameters
    beta <- parm[Data$pos.beta]
    Xi <- parm[Data$pos.Xi]
    kappa <- 1; lambda <- 1
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], 1, 5)
    parm[Data$pos.psi] <- psi <- interval(parm[Data$pos.psi], 1e-100, Inf)
    Sigma <- sigma[2]*sigma[2] * exp(-(Data$D.S / phi[1])^kappa -
        (Data$D.T / phi[2])^lambda -
        psi*(Data$D.S / phi[1])^kappa * (Data$D.T / phi[2])^lambda)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    Xi.prior <- dmvn(Xi, Xi.mu, Sigma, log=TRUE)
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    phi.prior <- sum(dunif(phi, 1, 5, log=TRUE))
    psi.prior <- dhalfcauchy(psi, 25, log=TRUE)
    ### Log-Likelihood
    Xi <- matrix(Xi, Data$S, Data$T)
    mu <- as.vector(tcrossprod(Data$X, t(beta))) + Xi
    LL <- sum(dnorm(Data$Y, mu, sigma[1], log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + Xi.prior + sigma.prior + phi.prior + psi.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(prod(dim(mu)), mu, sigma[1]), parm=parm)
    return(Modelout)
    }
```

88.4. Initial Values

```
Initial.Values <- c(rep(0,S*T), c(mean(Y),rep(0,K-1)), rep(1,2), rep(1,2),
    1)
```

89. Space-Time, Separable

This introductory approach to space-time or spatiotemporal modeling applies kriging both to the stationary spatial and temporal components, where space is continuous and time is discrete. Vector ζ contains the spatial effects and vector θ contains the temporal effects. Spatial coordinates are given in longitude and latitude for $s = 1, \dots, S$ points in space and measurements are taken across time-periods $t = 1, \dots, T$ for $\mathbf{Y}_{s,t}$. The dependent variable is also a function of design matrix $\mathbf{X}$ and regression effects vector β. For more information on kriging, see section 44. This example uses separable space-time covariances, which is more convenient but usually less appropriate than a nonseparable covariance function. To extend this to a large space-time data set, consider incorporating the predictive process kriging example in section 45.

89.1. Form

$$\mathbf{Y}_{s,t} \sim \mathcal{N}(\mu_{s,t}, \sigma_1^2), \quad s = 1, \dots, S, \quad t = 1, \dots, T$$
$$\mu_{s,t} = \mathbf{X}_{s,1:J}\beta + \zeta_s + \Theta_{s,t}$$
$$\Theta_{s,1:T} = \theta$$
$$\theta \sim \mathcal{N}_N(\theta_\mu, \Sigma_T)$$
$$\Sigma_T = \sigma_3^2 \exp(-\phi_2 \mathbf{D}_T)^\lambda$$
$$\zeta \sim \mathcal{N}_N(\zeta_\mu, \Sigma_S)$$
$$\Sigma_S = \sigma_2^2 \exp(-\phi_1 \mathbf{D}_S)^\kappa$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \dots, 2$$
$$\sigma_k \sim \mathcal{HC}(25), \quad k = 1, \dots, 3$$
$$\phi_k \sim \mathcal{U}(1, 5), \quad k = 1, \dots, 2$$
$$\zeta_\mu = 0$$
$$\theta_\mu = 0$$
$$\kappa = 1, \quad \lambda = 1$$

89.2. Data

```
data(demontexas)
Y <- as.matrix(demontexas[1:20,c(18:30)])
X <- cbind(1,as.matrix(demontexas[1:20,c(1,4)])) #Static predictors
latitude <- demontexas[1:20,2]
longitude <- demontexas[1:20,3]
S <- nrow(Y) #Number of sites, or points in space
T <- ncol(Y) #Number of time-periods
K <- ncol(X) #Number of columns in design matrix X including the intercept
D.S <- as.matrix(dist(cbind(longitude,latitude), diag=TRUE, upper=TRUE))
D.T <- as.matrix(dist(cbind(c(1:T),c(1:T)), diag=TRUE, upper=TRUE))
```

```
mon.names <- "LP"
parm.names <- as.parm.names(list(zeta=rep(0,S), theta=rep(0,T),
    beta=rep(0,K), phi=rep(0,2), sigma=rep(0,3)))
pos.zeta <- grep("zeta", parm.names)
pos.theta <- grep("theta", parm.names)
pos.beta <- grep("beta", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$K, c(mean(Data$Y),rep(0,Data$K-1)), 1)
    phi <- runif(2,1,5)
    sigma <- runif(3)
    kappa <- 1
    lambda <- 1
    Sigma.S <- sigma[2]^2 * exp(-phi[1] * Data$D.S)^kappa
    Sigma.T <- sigma[3]^2 * exp(-phi[2] * Data$D.T)^lambda
    zeta <- as.vector(rmvn(1, rep(0,Data$S), Sigma.S))
    theta <- as.vector(rmvn(1, rep(0,Data$T), Sigma.T))
    return(c(zeta, theta, beta, phi, sigma))
    }
MyData <- list(D.S=D.S, D.T=D.T, K=K, PGF=PGF, S=S, T=T, X=X, Y=Y,
    latitude=latitude, longitude=longitude, mon.names=mon.names,
    parm.names=parm.names, pos.zeta=pos.zeta, pos.theta=pos.theta,
    pos.beta=pos.beta, pos.phi=pos.phi, pos.sigma=pos.sigma)
```

89.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperparameters
    zeta.mu <- rep(0,Data$S)
    theta.mu <- rep(0,Data$T)
    ### Parameters
    beta <- parm[Data$pos.beta]
    zeta <- parm[Data$pos.zeta]
    theta <- parm[Data$pos.theta]
    kappa <- 1; lambda <- 1
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], 1, 5)
    Sigma.S <- sigma[2]^2 * exp(-phi[1] * Data$D.S)^kappa
    Sigma.T <- sigma[3]^2 * exp(-phi[2] * Data$D.T)^lambda
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    zeta.prior <- dmvn(zeta, zeta.mu, Sigma.S, log=TRUE)
    theta.prior <- dmvn(theta, theta.mu, Sigma.T, log=TRUE)
```

```
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    phi.prior <- sum(dunif(phi, 1, 5, log=TRUE))
    ### Log-Likelihood
    Theta <- matrix(theta, Data$S, Data$T, byrow=TRUE)
    mu <- as.vector(tcrossprod(Data$X, t(beta))) + zeta + Theta
    LL <- sum(dnorm(Data$Y, mu, sigma[1], log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + zeta.prior + theta.prior + sigma.prior +
        phi.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(prod(dim(mu)), mu, sigma[1]), parm=parm)
    return(Modelout)
    }
```

89.4. Initial Values

```
Initial.Values <- c(rep(0,S), rep(0,T), rep(0,2), rep(1,2), rep(1,3))
```

90. Spatial Autoregression (SAR)

The spatial autoregressive (SAR) model in this example uses areal data that consists of first-order neighbors that were specified and converted from point-based data with longitude and latitude coordinates.

90.1. Form

$$\begin{aligned}
\mathbf{y} &\sim \mathcal{N}(\mu, \sigma^2) \\
\mu &= \mathbf{X}\beta + \phi\mathbf{z} \\
\beta_j &\sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J \\
\phi &\sim \mathcal{U}(-1, 1) \\
\sigma &\sim \mathcal{HC}(25)
\end{aligned}$$

90.2. Data

```
N <- 100
latitude <- runif(N,0,100); longitude <- runif(N,0,100)
J <- 3 #Number of predictors, including the intercept
X <- matrix(runif(N*J,0,3), N, J); X[,1] <- 1
beta.orig <- runif(J,0,3); phi <- runif(1,0,1)
D <- as.matrix(dist(cbind(longitude, latitude), diag=TRUE, upper=TRUE))
W <- exp(-D) #Inverse distance as weights
W <- ifelse(D == 0, 0, W)
epsilon <- rnorm(N,0,1)
```

```
y <- tcrossprod(X, t(beta.orig)) + sqrt(latitude) + sqrt(longitude) +
    epsilon
Z <- W / matrix(rowSums(W), N, N) * matrix(y, N, N, byrow=TRUE)
z <- rowSums(Z)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), phi=0, sigma=0))
pos.beta <- grep("beta", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    phi <- runif(1,-1,1)
    sigma <- runif(1)
    return(c(beta, phi, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, latitude=latitude, longitude=longitude,
    mon.names=mon.names, parm.names=parm.names, pos.beta=pos.beta,
    pos.phi=pos.phi, pos.sigma=pos.sigma, y=y, z=z)
```

90.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], -1, 1)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    phi.prior <- dunif(phi, -1, 1, log=TRUE)
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta)) + phi*Data$z
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + phi.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

90.4. Initial Values

```
Initial.Values <- c(rep(0,J), 0.5, 1)
```

91. STARMA(p,q)

The data in this example of a space-time autoregressive moving average (STARMA) are coordinate-based, and the adjacency matrix $\mathbf{A}$ is created from K nearest neighbors. Otherwise, an adjacency matrix may be specified as usual for areal data. Spatial coordinates are given in longitude and latitude for $s = 1, \ldots, S$ points in space and measurements are taken across time-periods $t = 1, \ldots, T$ for $\mathbf{Y}_{s,t}$.

91.1. Form

$$\mathbf{Y} \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mu_{s,t} = \sum_{j=1}^{J} \mathbf{X}_{s,t,j}\beta_j + \sum_{l=1}^{L}\sum_{p=1}^{P} \phi_{l,p}\mathbf{W}^1_{s,t-p,l} + \sum_{m=1}^{M}\sum_{q=1}^{Q} \theta_{m,q}\mathbf{W}^2_{s,t-q,m}, \quad j = 1, \ldots, J, \quad s = 1, \ldots, S, \quad t = p, \ldots, T$$

$$\mathbf{W}^1_{1:S,1:T,l} = \mathbf{V}_{1:S,1:S,l}\mathbf{Y}, \quad l = 1, \ldots, L$$

$$\mathbf{W}^2_{1:S,1:T,m} = \mathbf{V}_{1:S,1:S,m}\epsilon, \quad m = 1, \ldots, M$$

$$\epsilon = \mathbf{Y} - \mu$$

$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\phi_{l,p} \sim \mathcal{U}(-1, 1), \quad l = 1, \ldots, L, \quad p = 1, \ldots, P$$

$$\sigma \sim \mathcal{HC}(25)$$

$$\theta_{m,q} \sim \mathcal{N}(0, 1000), \quad m = 1, \ldots, M, \quad q = 1, \ldots, Q$$

where $\mathbf{V}$ is an adjacency array that is scaled so that each row sums to one, β is a vector of regression effects, ϕ is a matrixr of autoregressive space-time parameters, σ is the residual variance, and θ is a matrix of moving average space-time parameters.

91.2. Data

```
data(demontexas)
Y <- t(diff(t(as.matrix(demontexas[,c(18:30)])))) #Note this is not stationary
X <- array(1, dim=c(369,13-1,3))
X[, , 2] <- CenterScale(demontexas[,1])
X[, , 3] <- demontexas[,4]
latitude <- demontexas[,2]
longitude <- demontexas[,3]
S <- nrow(Y) #Number of sites, or points in space
T <- ncol(Y) #Number of time-periods
J <- dim(X)[3] #Number of columns in design matrix X including the intercept
K <- 5 #Number of nearest neighbors
L <- 2 #Spatial autoregressive order
M <- 2 #Spatial moving average order
P <- 2 #Autoregressive order
Q <- 2 #Moving average order
D <- as.matrix(dist(cbind(longitude, latitude), diag=TRUE, upper=TRUE))
```

```
A <- V <- array(0, dim=c(nrow(D),ncol(D),P))
W1 <- array(0, dim=c(S,T,max(L,M)))
for (l in 1:max(L,M)) {
    A[, , l] <- exp(-D)
    A[, , l] <- apply(A[, , l], 1, rank)
    A[, , l] <- ifelse(A[, , l] > (l-1)*K & A[, , l] <= l*K, 1, 0)
    V[, , l] <- A[, , l] / rowSums(A[, , l])
    V[, , l] <- ifelse(is.nan(V[, , l]), 1/ncol(V[, , l]), V[, , l])
    W1[, , l] <- tcrossprod(V[, , l], t(Y))}
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), phi=matrix(0,L,P), sigma=0,
    theta=matrix(0,M,Q)))
pos.beta <- grep("beta", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.theta <- grep("theta", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    phi <- runif(Data$L*Data$P,-1,1)
    sigma <- runif(1)
    theta <- rnorm(Data$M*Data$Q)
    return(c(beta, phi, sigma, theta))
    }
MyData <- list(J=J, K=K, L=L, M=M, P=P, Q=Q, PGF=PGF, S=S, T=T, V=V, W1=W1,
    X=X, Y=Y, latitude=latitude, longitude=longitude, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.phi=pos.phi,
    pos.sigma=pos.sigma, pos.theta=pos.theta)
```

91.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    phi <- matrix(interval(parm[Data$pos.phi], -1, 1), Data$L, Data$P)
    parm[Data$pos.phi] <- as.vector(phi)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    theta <- matrix(parm[Data$pos.theta], Data$M, Data$Q)
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    phi.prior <- sum(dunif(phi, -1, 1, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    theta.prior <- sum(dnormv(theta, 0, 1000, log=TRUE))
    ### Log-Likelihood
    mu <- beta[1]*Data$X[, , 1]
```

```
    for (j in 2:Data$J) mu <- mu + beta[j]*Data$X[, , j]
    for (l in 1:Data$L) {for (p in 1:Data$P) {
        mu[,-c(1:p)] <- mu[,-c(1:p)] +
            phi[l,p]*Data$W1[, 1:(Data$T - p), l]}}
    epsilon <- Data$Y - mu
    for (m in 1:Data$M) {
        W2 <- tcrossprod(Data$V[, , m], t(epsilon))
        for (q in 1:Data$Q) {
            mu[,-c(1:q)] <- mu[,-c(1:q)] +
            theta[m,q]*W2[,1:(Data$T - q)]}}
    LL <- sum(dnorm(Data$Y[,-c(1:max(Data$P,Data$Q))],
        mu[,-c(1:max(Data$P,Data$Q))], sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + phi.prior + sigma.prior + theta.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(prod(dim(mu)), mu, sigma), parm=parm)
    return(Modelout)
    }
```

91.4. Initial Values

```
Initial.Values <- c(rep(0,J), rep(0,L*P), 1, rep(0,M*Q))
```

92. State Space Model (SSM), Linear Regression

92.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma^2_{J+1}), \quad t = 1, \ldots, T$$

$$\mu = \mathbf{X}\beta$$

$$\beta_{t,j} \sim \mathcal{N}(\mu_j + \phi_j(\beta_{t-1,j} - \mu_j), \sigma^2_j), \quad t = 2, \ldots, T, \quad j = 1, \ldots, J$$

$$\beta_{1,j} \sim \mathcal{N}(\mu_j + \phi_j(b^0_j - \mu_j), \sigma^2_j), \quad j = 1, \ldots, J$$

$$b^0_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\mu_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$

$$\phi_j \sim \mathcal{BETA}(20, 1.5) \quad j = 1, \ldots, J$$

$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, (J+1)$$

92.2. Data

```
data(demonfx)
y <- demonfx[1:50,1]
```

```
X <- cbind(1, as.matrix(demonfx[1:50,2:3]))
T <- nrow(X)
J <- ncol(X)
mon.names <- "LP"
parm.names <- as.parm.names(list(b0=rep(0,J), beta=matrix(0,T,J),
    mu=rep(0,J), phi=rep(0,J), sigma=rep(0,J+1)))
pos.b0 <- grep("b0", parm.names)
pos.beta <- grep("beta", parm.names)
pos.mu <- grep("mu", parm.names)
pos.phi <- grep("phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    b0 <- rnorm(Data$J)
    beta <- c(rnorm(Data$T,mean(Data$y),1), rnorm(Data$T*(Data$J-1)))
    mu <- rnorm(Data$J)
    phi <- runif(Data$J, -1, 1)
    sigma <- runif(Data$J+1)
    return(c(beta, mu, phi, sigma))
    }
MyData <- list(J=J, PGF=PGF, T=T, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.b0=pos.b0, pos.beta=pos.beta,
    pos.mu=pos.mu, pos.phi=pos.phi, pos.sigma=pos.sigma, y=y)
```

92.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    b0 <- parm[Data$pos.b0]
    beta <- matrix(parm[Data$pos.beta], Data$T, Data$J)
    mu <- parm[Data$pos.mu]
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], -1, 1)
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    b0.prior <- sum(dnormv(b0, 0, 1000, log=TRUE))
    beta.prior <- sum(dnorm(beta, matrix(mu, Data$T, Data$J, byrow=TRUE) +
        matrix(phi, Data$T, Data$J, byrow=TRUE) *
        (rbind(b0, beta[-Data$T,]) -
        matrix(mu, Data$T, Data$J, byrow=TRUE)),
        matrix(sigma[1:Data$J], Data$T, Data$J, byrow=TRUE), log=TRUE))
    mu.prior <- sum(dnormv(mu, 0, 1000, log=TRUE))
    phi.prior <- sum(dbeta((phi+1)/2, 20, 1.5, log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    mu <- rowSums(beta*Data$X)
```

```
    LL <- sum(dnorm(Data$y, mu, sigma[Data$J+1], log=TRUE))
    yhat <- rnorm(length(mu), mu, sigma[Data$J+1]) #Fitted
    #yhat <- rnorm(length(mu), rowSums(matrix(rnorm(Data$T*Data$J,
        # rbind(b0, beta[-Data$T,]), matrix(sigma[-Data$J], Data$T, Data$J,
        # byrow=TRUE)), Data$T, Data$J) * Data$X), sigma[Data$J+1]) #One-step
ahead
    ### Log-Posterior
    LP <- LL + b0.prior + beta.prior + mu.prior + phi.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=yhat, parm=parm)
    return(Modelout)
    }
```

92.4. Initial Values

```
Initial.Values <- c(rep(0,J), rep(mean(y),T), rep(0,T*(J-1)), rep(0,J),
    rep(0,J), rep(1,J+1))
```

93. State Space Model (SSM), Local Level

The local level model is the simplest, non-trivial example of a state space model (SSM). As such, this version of a local level SSM has static variance parameters.

93.1. Form

$$\begin{aligned}
\mathbf{y}_t &\sim \mathcal{N}(\mu_t, \sigma_1^2), \quad t = 1, \ldots, T \\
\mu_t &\sim \mathcal{N}(\mu_{t-1}, \sigma_2^2), \quad t = 2, \ldots, T \\
\mu_1 &\sim \mathcal{N}(0, 1000) \\
\sigma_j &\sim \mathcal{HC}(25), \quad j = 1, \ldots, 2
\end{aligned}$$

93.2. Data

```
T <- 20
T.m <- 14
mu.orig <- rep(0,T)
for (t in 2:T) {mu.orig[t] <- mu.orig[t-1] + rnorm(1,0,1)}
y <- mu.orig + rnorm(T,0,0.1)
y[(T.m+2):T] <- NA
mon.names <- rep(NA, (T-T.m))
for (i in 1:(T-T.m)) mon.names[i] <- paste("yhat[",(T.m+i),"]", sep="")
parm.names <- as.parm.names(list(mu=rep(0,T), sigma=rep(0,2)))
pos.mu <- grep("mu", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
```

```
    mu <- rnorm(Data$T)
    sigma <- runif(2)
    return(c(mu, sigma))
    }
MyData <- list(PGF=PGF, T=T, T.m=T.m, mon.names=mon.names,
    parm.names=parm.names, pos.mu=pos.mu, pos.sigma=pos.sigma, y=y)
Dyn <- matrix(paste("mu[",1:T,"]",sep=""), T, 1)
```

93.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    mu <- parm[Data$pos.mu]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    mu.prior <- sum(dnormv(mu[1], 0, 1000, log=TRUE),
        dnorm(mu[-1], mu[-Data$T], sigma[2], log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    LL <- sum(dnorm(Data$y[1:Data$T.m], mu[1:Data$T.m], sigma[1],
        log=TRUE))
    yhat <- rnorm(length(mu), c(mu[1], rnorm(Data$T-1, mu[-Data$T],
        sigma[2])), sigma[1]) #One-step ahead     ### Log-Posterior
    LP <- LL + mu.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=mu[(Data$T.m+1):Data$T],
        yhat=yhat, parm=parm)
    return(Modelout)
    }
```

93.4. Initial Values

```
Initial.Values <- c(rep(0,T), rep(1,2))
```

94. State Space Model (SSM), Local Linear Trend

The local linear trend model is a state space model (SSM) that extends the local level model to include a dynamic slope parameter. For more information on the local level model, see section 93. This example has static variance parameters.

94.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma_1^2), \quad t = 1, \ldots, T$$

$$\mu_t \sim \mathcal{N}(\mu_{t-1} + \delta_{t-1}, \sigma_2^2), \quad t = 2, \ldots, T$$
$$\mu_1 \sim \mathcal{N}(0, 1000)$$
$$\delta_t \sim \mathcal{N}(\delta_{t-1}, \sigma_3^2), \quad t = 2, \ldots, T$$
$$\delta_1 \sim \mathcal{N}(0, 1000)$$
$$\sigma_j \sim \mathcal{HC}(25), \quad j = 1, \ldots, 3$$

94.2. Data

```
T <- 20
T.m <- 14
mu.orig <- delta.orig <- rep(0,T)
for (t in 2:T) {
    delta.orig[t] <- delta.orig[t-1] + rnorm(1,0,0.1)
    mu.orig[t] <- mu.orig[t-1] + delta.orig[t-1] + rnorm(1,0,1)}
y <- mu.orig + rnorm(T,0,0.1)
y[(T.m+2):T] <- NA
mon.names <- rep(NA, (T-T.m))
for (i in 1:(T-T.m)) mon.names[i] <- paste("yhat[",(T.m+i),"]", sep="")
parm.names <- as.parm.names(list(mu=rep(0,T), delta=rep(0,T),
    sigma=rep(0,3)))
pos.mu <- grep("mu", parm.names)
pos.delta <- grep("delta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    mu <- rnorm(Data$T)
    delta <- rnorm(Data$T)
    sigma <- runif(3)
    return(c(mu, delta, sigma))
    }
MyData <- list(PGF=PGF, T=T, T.m=T.m, mon.names=mon.names,
    parm.names=parm.names, pos.mu=pos.mu, pos.delta=pos.delta,
    pos.sigma=pos.sigma, y=y)
```

94.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    mu <- parm[Data$pos.mu]
    delta <- parm[Data$pos.delta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    mu.prior <- sum(dnormv(mu[1], 0, 1000, log=TRUE),
```

```
    dnorm(mu[-1], mu[-Data$T]+delta[-Data$T], sigma[2],
    log=TRUE))
delta.prior <- sum(dnormv(delta[1], 0, 1000, log=TRUE),
    dnorm(delta[-1], delta[-Data$T], sigma[3], log=TRUE))
sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
### Log-Likelihood
LL <- sum(dnorm(Data$y[1:Data$T.m], mu[1:Data$T.m], sigma[1],
    log=TRUE))
yhat <- rnorm(length(mu), c(mu[1], rnorm(Data$T-1, mu[-Data$T],
    sigma[2])), sigma[1]) #One-step ahead
### Log-Posterior
LP <- LL + mu.prior + delta.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=mu[(Data$T.m+1):Data$T],
    yhat=yhat, parm=parm)
return(Modelout)
}
```

94.4. Initial Values

```
Initial.Values <- c(rep(0,T), rep(0,T), rep(1,3))
```

95. State Space Model (SSM), Stochastic Volatility (SV)

95.1. Form

$$\mathbf{y} \sim \mathcal{N}(0, \sigma^2)$$
$$\sigma^2 = \frac{1}{\exp(\theta)}$$
$$\beta = \exp(\mu/2)$$
$$\theta_1 \sim \mathcal{N}(\mu + \phi(\alpha - \mu), \tau)$$
$$\theta_t \sim \mathcal{N}(\mu + \phi(\theta_{t-1} - \mu), \tau), \quad t = 2, \ldots, T$$
$$\alpha \sim \mathcal{N}(\mu, \tau)$$
$$\phi \sim \mathcal{U}(-1, 1)$$
$$\mu \sim \mathcal{N}(0, 10)$$
$$\tau \sim \mathcal{HC}(25)$$

95.2. Data

```
T <- 20
y <- rep(10,T); epsilon <- rnorm(T,0,1)
```

```
for (t in 2:T) {y[t] <- 0.8*y[t-1] + epsilon[t-1]}
mon.names <- c("LP",paste("sigma2[",1:T,"]",sep=""))
parm.names <- as.parm.names(list(theta=rep(0,T), alpha=0, phi=0, mu=0,
    tau=0))
pos.theta <- grep("theta", parm.names)
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.mu <- grep("mu", parm.names)
pos.tau <- grep("tau", parm.names)
PGF <- function(Data) {
    phi <- runif(1,-1,1)
    mu <- rnorm(1)
    tau <- runif(1)
    alpha <- rnorm(1, mu, tau)
    theta <- rnorm(Data$T, mu + phi*(alpha - mu), tau)
    return(c(theta, alpha, phi, mu, tau))
    }
MyData <- list(PGF=PGF, T=T, mon.names=mon.names, parm.names=parm.names,     pos.theta=pos
pos.alpha=pos.alpha, pos.phi=pos.phi,
    pos.mu=pos.mu, pos.tau=pos.tau y=y)
Dyn <- matrix(paste("theta[",1:T,"]",sep=""), T, 1)
```

95.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    theta <- parm[Data$pos.theta]
    alpha <- parm[Data$pos.alpha]
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], -1, 1)
    mu <- parm[Data$pos.mu]
    parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
    ### Log-Prior
    alpha.prior <- dnormv(alpha, mu, tau, log=TRUE)
    theta.prior <- sum(dnormv(theta[1], mu + phi*(alpha-mu), tau,
        log=TRUE), dnormv(theta[-1], mu + phi*(theta[-Data$T]-mu), tau,
        log=TRUE))
    phi.prior <- dunif(phi, -1, 1, log=TRUE)
    mu.prior <- dnormv(mu, 0, 10, log=TRUE)
    tau.prior <- dhalfcauchy(tau, 25, log=TRUE)
    ### Log-Likelihood
    beta <- exp(mu / 2)
    sigma2 <- 1 / exp(theta)
    LL <- sum(dnormv(Data$y, 0, sigma2, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + theta.prior + phi.prior + mu.prior +
```

```
    tau.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP, sigma2),
    yhat=rnormv(length(Data$y), 0, sigma2), parm=parm)
return(Modelout)
}
```

95.4. Initial Values

```
Initial.Values <- c(rep(0,T), 0, 0, 0, 1)
```

96. Threshold Autoregression (TAR)

96.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\nu_t, \sigma^2), \quad t = 1, \dots, T$$

$$\nu_t = \begin{cases} \alpha_1 + \phi_1 \mathbf{y}_{t-1}, & t = 1, \dots, T \quad \text{if } t \geq \theta \\ \alpha_2 + \phi_2 \mathbf{y}_{t-1}, & t = 1, \dots, T \quad \text{if } t < \theta \end{cases}$$

$$\alpha_j \sim \mathcal{N}(0, 1000) \in [-1, 1], \quad j = 1, \dots, 2$$

$$\phi_j \sim \mathcal{N}(0, 1000), \in [-1, 1], \quad j = 1, \dots, 2$$

$$\theta \sim \mathcal{U}(2, T-1)$$

$$\sigma \sim \mathcal{HC}(25)$$

96.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
T <- length(y)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,2), phi=rep(0,2), theta=0,
    sigma=0))
pos.alpha <- grep("alpha", parm.names)
pos.phi <- grep("phi", parm.names)
pos.theta <- grep("theta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rtrunc(2, "norm", a=-1, b=1, mean=0, sd=1)
    phi <- rtrunc(2, "norm", a=-1, b=1, mean=0, sd=1)
    theta <- runif(1,2,Data$T-1)
    sigma <- runif(1)
    return(c(alpha, phi, theta, sigma))
    }
```

```
MyData <- list(PGF=PGF, T=T, mon.names=mon.names, parm.names=parm.names,
    pos.alpha=pos.alpha, pos.phi=pos.phi, pos.theta=pos.theta,
    pos.sigma=pos.sigma, y=y)
```

96.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    parm[Data$pos.alpha] <- alpha <- interval(parm[Data$pos.alpha], -1, 1)
    parm[Data$pos.phi] <- phi <- interval(parm[Data$pos.phi], -1, 1)
    theta <- interval(parm[Data$pos.theta], 2, Data$T-1)
    parm[Data$pos.theta] <- theta
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- sum(dtrunc(alpha, "norm", a=-1, b=1, mean=0,
        sd=sqrt(1000), log=TRUE))
    phi.prior <- sum(dtrunc(phi, "norm", a=-1, b=1, mean=0,
        sd=sqrt(1000), log=TRUE))
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    phi.prior <- sum(dnormv(phi, 0, 1000, log=TRUE))
    theta.prior <- dunif(theta, 2, Data$T-1, log=TRUE)
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- matrix(0, Data$T, 2)
    mu[,1] <- c(alpha[1], alpha[1] + phi[1]*Data$y[-Data$T])
    mu[,2] <- c(alpha[2], alpha[2] + phi[2]*Data$y[-Data$T])
    nu <- mu[,2]; temp <- which(1:Data$T < theta)
    nu[temp] <- mu[temp,1]
    LL <- sum(dnorm(Data$y[-1], nu[-1], sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + phi.prior + theta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(nu), nu, sigma), parm=parm)
    return(Modelout)
    }
```

96.4. Initial Values

```
Initial.Values <- c(rep(0,4), T/2, 1)
```

97. Time Varying AR(1) with Chebyshev Series

This example consists of a first-order autoregressive model, AR(1), with a time-varying pa-

rameter (TVP) ϕ, that is a Chebyshev series constructed from a linear combination of orthonormal Chebyshev time polynomials (CTPs) and parameter vector β. The user creates basis matrix $\mathbf{P}$, specifying polynomial degree D and time T. Each column is a CTP of a different degree, and the first column is restricted to 1, the linear basis. CTPs are very flexible for TVPs, and estimate quickly because each is orthogonal, unlike simple polynomials and splines.

97.1. Form

$$\mathbf{y}_t \sim \mathcal{N}(\mu_t, \sigma^2), \quad t = 1, \ldots, T$$
$$\mu_t = \alpha + \phi_{t-1}\mathbf{y}_{t-1}$$
$$\phi_t = \mathbf{P}\beta$$
$$\alpha \sim \mathcal{N}(0, 1000)$$
$$\beta_d \sim \mathcal{N}(0, 1000), \quad d = 1, \ldots, (D+1)$$
$$\sigma \sim \mathcal{HC}(25)$$

97.2. Data

```
data(demonfx)
y <- as.vector(diff(log(as.matrix(demonfx[1:261,1]))))
D <- 6 #Maximum degree of Chebyshev time polynomials
T <- length(y)
P <- matrix(1, T, D+1)
for (d in 1:D) {P[,d+1] <- sqrt(2)*cos(d*pi*(c(1:T)-0.5)/T)}
mon.names <- c("LP", "ynew", as.parm.names(list(phi=rep(0,T-1))))
parm.names <- as.parm.names(list(alpha=0, beta=rep(0,D+1), sigma=0))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(1)
    beta <- rnorm(Data$D+1)
    sigma <- runif(1)
    return(c(alpha, beta, sigma))
    }
MyData <- list(D=D, P=P, PGF=PGF, T=T, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.beta=pos.beta,
    pos.sigma=pos.sigma, y=y)
```

97.3. Model

```
Model <- function(parm, Data)
    {
```

```
### Parameters
alpha <- parm[Data$pos.alpha]
beta <- parm[Data$pos.beta]
sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
parm[Data$pos.sigma] <- sigma
### Log-Prior
alpha.prior <- dnormv(alpha, 0, 1000, log=TRUE)
beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
### Log-Likelihood
phi <- tcrossprod(Data$P[-Data$T,], t(beta))
mu <- c(alpha, alpha + phi*Data$y[-Data$T])
ynew <- rnorm(1, alpha + tcrossprod(Data$P[Data$T,], t(beta))*
    Data$y[Data$T], sigma)
LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + beta.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,ynew,phi),
    yhat=rnorm(length(mu), mu, sigma), parm=parm)
return(Modelout)
}
```

97.4. Initial Values

```
Initial.Values <- c(rep(0,D+2), 1)
```

98. Variable Selection, BAL

This approach to variable selection is one of several forms of the Bayesian Adaptive Lasso (BAL). The lasso applies shrinkage to exchangeable scale parameters, γ, for the regression effects, β.

98.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \mathbf{X}\beta$$
$$\beta_1 \sim \mathcal{L}(0, 1000)$$
$$\beta_j \sim \mathcal{L}(0, \gamma_j), \quad j = 2, \ldots, J$$
$$\gamma_j \sim \mathcal{G}^{-1}(\delta, \tau), \quad \in [0, \infty]$$
$$\delta \sim \mathcal{HC}(25)$$
$$\tau \sim \mathcal{HC}(25)$$
$$\sigma \sim \mathcal{HC}(25)$$

98.2. Data

```
data(demonsnacks)
J <- ncol(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(demonsnacks[,c(1,3:10)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), gamma=rep(0,J-1), delta=0,
    tau=0, sigma=0))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.delta <- grep("delta", parm.names)
pos.tau <- grep("tau", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    delta <- runif(1)
    tau <- runif(1)
    sigma <- runif(1)
    gamma <- rinvgamma(Data$J-1, delta, tau)
    beta <- rlaplace(Data$J, 0, c(1,gamma))
    return(c(beta, gamma, delta, tau, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma,
    pos.delta=pos.delta, pos.tau=pos.tau, pos.sigma=pos.sigma, y=y)
```

98.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperhyperparameters
    delta <- interval(parm[Data$pos.delta], 1e-100, Inf)
    parm[Data$pos.delta] <- delta
    parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
    ### Hyperparameters
    gamma <- interval(parm[Data$pos.gamma], 1e-100, Inf)
    parm[Data$pos.gamma] <- gamma
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Hyperhyperprior
    delta.prior <- dhalfcauchy(delta, 25, log=TRUE)
    tau.prior <- dhalfcauchy(tau, 25, log=TRUE)
    ### Log-Hyperprior
```

```
    gamma.prior <- sum(dinvgamma(gamma, delta, tau, log=TRUE))
    ### Log-Prior
    beta.prior <- sum(dlaplace(beta, 0, c(1000, gamma), log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + gamma.prior + delta.prior + tau.prior +
         sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
         yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

98.4. Initial Values

```
Initial.Values <- c(rep(0,J), rep(0,J-1), rep(1,3))
```

99. Variable Selection, Horseshoe

99.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \mathbf{X}\beta$$
$$\beta_1 \sim \mathcal{N}(0, 1000)$$
$$\beta_j \sim \mathcal{HS}(\lambda_j, \tau), \quad j = 2, \ldots, J$$
$$\lambda_j \sim \mathcal{HC}(1), \quad j = 2, \ldots, J$$
$$\tau \sim \mathcal{HC}(1)$$
$$\sigma \sim \mathcal{HC}(25)$$

99.2. Data

```
data(demonsnacks)
J <- ncol(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(demonsnacks[,c(1,3:10)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), lambda=rep(0,J-1),
    tau=0, sigma=0))
```

```
pos.beta <- grep("beta", parm.names)
pos.lambda <- grep("lambda", parm.names)
pos.tau <- grep("tau", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     lambda <- runif(Data$J-1)
     tau <- runif(1)
     sigma <- runif(1)
     return(c(beta, lambda, tau, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.lambda=pos.lambda,
     pos.tau=pos.tau, pos.sigma=pos.sigma, y=y)
```

99.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     lambda <- interval(parm[Data$pos.lambda], 1e-100, Inf)
     parm[Data$pos.lambda] <- lambda
     parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     beta.prior <- sum(dnormv(beta[1], 0, 1000, log=TRUE),
          dhs(beta[-1], lambda, tau, log=TRUE))
     lambda.prior <- sum(dhalfcauchy(lambda, 1, log=TRUE))
     tau.prior <- dhalfcauchy(tau, 1, log=TRUE)
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }
```

99.4. Initial Values

```
Initial.Values <- c(rep(0,J), rep(1,J-1), rep(1,2))
```

100. Variable Selection, LASSO

100.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \mathbf{X}\beta$$
$$\beta_1 \sim \mathcal{N}(0, 1000)$$
$$\beta_j \sim \mathcal{LASSO}(0, \sigma, \tau, \lambda_j), \quad j = 2, \ldots, J$$

100.2. Data

```
data(demonsnacks)
J <- ncol(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(demonsnacks[,c(1,3:10)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0, tau=rep(0,J-1),
    lambda=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
pos.tau <- grep("tau", parm.names)
pos.lambda <- grep("lambda", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    sigma <- runif(1)
    tau <- runif(Data$J-1)
    lambda <- runif(1)
    return(c(beta, sigma, tau, lambda))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma,
    pos.tau=pos.tau, pos.lambda=pos.lambda, y=y)
```

100.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    parm[Data$pos.tau] <- tau <- interval(parm[Data$pos.tau], 1e-100, Inf)
```

```
lambda <- interval(parm[Data$pos.lambda], 1e-100, Inf)
parm[Data$pos.lambda] <- lambda
### Log-Prior
beta.prior <- sum(dnormv(beta[1], 0, 1000, log=TRUE),
     dlasso(beta[-1], sigma, tau, lambda, log=TRUE))
### Log-Likelihood
mu <- tcrossprod(Data$X, t(beta))
LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
### Log-Posterior
LP <- LL + beta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
     yhat=rnorm(length(mu), mu, sigma), parm=parm)
return(Modelout)
}
```

100.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1, rep(1,J-1), 1)
```

101. Variable Selection, RJ

This example uses the RJ (Reversible-Jump) algorithm of the `LaplacesDemon` function for variable selection and Bayesian Model Averaging (BMA). Other MCMC algorithms will not perform variable selection with this example, as presented. This is an example of variable selection in a linear regression. The only difference between the following example, and the example of linear regression (48), is that RJ specifications are also included for the RJ algorithm, and that the RJ algorithm must be used.

101.1. Form

$$\begin{aligned}
\mathbf{y} &\sim \mathcal{N}(\mu, \sigma^2) \\
\mu &= \mathbf{X}\beta \\
\beta_j &\sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J \\
\sigma &\sim \mathcal{HC}(25)
\end{aligned}$$

101.2. Data

```
N <- 1000
J <- 100 #Number of predictors, including the intercept
X <- matrix(1,N,J)
for (j in 2:J) {X[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))}
beta.orig <- runif(J,-3,3)
zero <- sample(2:J, round(J*0.9)) #Assign most parameters to be zero
```

```
beta.orig[zero] <- 0
e <- rnorm(N,0,0.1)
y <- as.vector(tcrossprod(beta.orig, X) + e)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    sigma <- runif(1)
    return(c(beta, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)
### Reversible-Jump Specifications bin.n <- J-1 #Maximum allowable model size
bin.p <- 0.4 #Most probable size:  bin.p x bin.n is binomial mean and median
parm.p <- rep(1/J,J+1)
selectable=c(0, rep(1,J-1), 0)
```

101.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

101.4. Initial Values

```
Initial.Values <- GIV(Model, MyData, PGF=TRUE)
```

102. Variable Selection, SSVS

This example uses a modified form of the random-effects (or global adaptation) Stochastic Search Variable Selection (SSVS) algorithm presented in O'Hara and Sillanpaa (2009), which selects variables according to practical significance rather than statistical significance. Here, SSVS is applied to linear regression, though this method is widely applicable. For J variables, each regression effect β_j is conditional on γ_j, a binary inclusion variable. Each β_j is a discrete mixture distribution with respect to $\gamma_j = 0$ or $\gamma_j = 1$, with precision 100 or $\beta_\sigma = 0.1$, respectively. As with other representations of SSVS, these precisions may require tuning.

The binary inclusion variables are discrete parameters, and discrete parameters are not supported in all algorithms. The example below is updated with the Slice sampler.

When the goal is to select the best model, each $\mathbf{X}_{1:N,j}$ is retained for a future run when the posterior mean of $\gamma_j \geq 0.5$. When the goal is model-averaging, the results of this model may be used directly, which would please L. J. Savage, who said that "models should be as big as an elephant" (Draper 1995).

102.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \mathbf{X}\beta$$
$$\beta_1 \sim \mathcal{N}(0, 1000)$$
$$(\beta_j|\gamma_j) \sim (1-\gamma_j)\mathcal{N}(0, 0.01) + \gamma_j\mathcal{N}(0, \beta_\sigma^2) \quad j = 2, \ldots, J$$
$$\beta_\sigma \sim \mathcal{HC}(25)$$
$$\gamma_j \sim \mathcal{BERN}(1/(J-1)), \quad j = 1, \ldots, (J-1)$$
$$\sigma \sim \mathcal{HC}(25)$$

102.2. Data

```
data(demonsnacks)
J <- ncol(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(demonsnacks[,c(1,3:10)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
mon.names <- c("LP", "min.beta.sigma")
parm.names <- as.parm.names(list(beta=rep(0,J), gamma=rep(0,J-1),
    b.sd=0, sigma=0))
pos.beta <- grep("beta", parm.names)
pos.gamma <- grep("gamma", parm.names)
pos.b.sd <- grep("b.sd", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
    gamma <- rep(1,Data$J-1)
```

```
    b.sd <- rnorm(1)
    sigma <- runif(1)
    return(c(beta, gamma, b.sd, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.gamma=pos.gamma,
    pos.b.sd=pos.b.sd, pos.sigma=pos.sigma, y=y)
```

102.3. Model

```
Model <- function(parm, Data)
    {
    ### Hyperparameters
    beta.sigma <- interval(parm[Data$pos.b.sd], 1e-100, Inf)
    parm[Data$pos.b.sd] <- beta.sigma
    ### Parameters
    beta <- parm[Data$pos.beta]
    gamma <- parm[Data$pos.gamma]
    beta.sigma <- rep(beta.sigma, Data$J-1)
    beta.sigma[gamma == 0] <- 0.1
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    ### Log-Hyperprior
    beta.sigma.prior <- sum(dhalfcauchy(beta.sigma, 25, log=TRUE))
    ### Log-Prior
    beta.prior <- sum(dnorm(beta, 0, c(sqrt(1000), beta.sigma, log=TRUE)))
    gamma.prior <- sum(dbern(gamma, 1/(Data$J-1), log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta*c(1,gamma)))
    LL <- sum(dnorm(y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + beta.sigma.prior + gamma.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP, min(beta.sigma)),
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

102.4. Initial Values

```
Initial.Values <- c(rep(0,J), rep(1,J-1), rep(1,2))
```

103. VARMA(p,q) - SSVS

Stochastic search variable selection (SSVS) is applied to VARMA parameters. Note that the constants for the mixture variances are typically multiplied by the posterior standard

deviations from an unrestricted VARMA that was updated previously, and these are not included in this example. Since an unrestricted VARMA model may be difficult to identify, this should be performed only on the AR parameters.

103.1. Form

$$\mathbf{Y}_{t,j} \sim \mathcal{N}(\mu_{t,j}, \sigma_j^2), \quad t = 1, \ldots, T, \quad j = 1, \ldots, J$$

$$\mu_{t,j} = \alpha_j + \sum_{p=1}^{P} \Gamma^{\Phi}_{1:J,j,p} \Phi_{1:J,j,p} \mathbf{Y}_{t-p,j} + \sum_{q=1}^{Q} \Gamma^{\Theta}_{1:J,j,q} \Theta_{1:J,j,q} \epsilon_{t-q,j}$$

$$\alpha_j \sim \mathcal{N}(0, 1000)$$

$$\Gamma^{\Phi}_{i,k,p} \sim \mathcal{BERN}(0.5), \quad i = 1, \ldots, J, \quad k = 1, \ldots, J, \quad p = 1, \ldots, P$$

$$(\Phi_{i,k,p} | \Gamma^{\Phi}_{i,k,p}) \sim (1 - \Gamma^{\Phi}_{i,k,p}) \mathcal{N}(0, 0.01) + \Gamma^{\Phi}_{i,k,p} \mathcal{N}(0, 10), \quad i = 1, \ldots, J, \quad k = 1, \ldots, J, \quad p = 1, \ldots, P$$

$$\Gamma^{\Theta}_{i,k,q} \sim \mathcal{BERN}(0.5), \quad i = 1, \ldots, J, \quad k = 1, \ldots, J, \quad q = 1, \ldots, Q$$

$$(\Theta_{i,k,q} | \Gamma^{\Theta}_{i,k,q}) \sim (1 - \Gamma^{\Theta}_{i,k,q}) \mathcal{N}(0, 0.01) + \Gamma^{\Theta}_{i,k,q} \mathcal{N}(0, 10), \quad i = 1, \ldots, J, \quad k = 1, \ldots, J, \quad q = 1, \ldots, Q$$

$$\sigma_j \sim \mathcal{HC}(25)$$

103.2. Data

```
data(demonfx)
Y.orig <- as.matrix(demonfx[,1:3])
Y <- diff(log(Y.orig[1:100,]))
Y.scales <- sqrt(.colVars(Y))
Y <- Y / matrix(Y.scales, nrow(Y), ncol(Y), byrow=TRUE)
T <- nrow(Y)
J <- ncol(Y)
L.P <- c(1,5,20) #Autoregressive lags
L.Q <- c(1,2) #Moving average lags
P <- length(L.P) #Autoregressive order
Q <- length(L.Q) #Moving average order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,J),
    Gamma.phi=array(0, dim=c(J,J,P)), Phi=array(0, dim=c(J,J,P)),
    Gamma.theta=array(0, dim=c(J,J,Q)), Theta=array(0, dim=c(J,J,Q)),
    sigma=rep(0,J)))
pos.alpha <- grep("alpha", parm.names)
pos.Gamma.phi <- grep("Gamma.phi", parm.names)
pos.Phi <- grep("Phi", parm.names)
pos.Gamma.theta <- grep("Gamma.theta", parm.names)
pos.Theta <- grep("Theta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
```

```
    alpha <- rnorm(Data$J)
    Gamma.phi <- rep(1, Data$J*Data$J*Data$P)
    Phi <- runif(Data$J*Data$J*Data$P, -1, 1)
    Gamma.theta <- rep(1, Data$J*Data$J*Data$Q)
    Theta <- rnorm(Data$J*Data$J*Data$Q)
    sigma <- runif(Data$J)
    return(c(alpha, Gamma.phi, Phi, Gamma.theta, Theta, sigma))     }
MyData <- list(J=J, L.P=L.P, L.Q=L.Q, P=P, Q=Q, PGF=PGF, T=T, Y=Y,
    mon.names=mon.names, parm.names=parm.names, pos.alpha=pos.alpha,
    pos.Gamma.phi=pos.Gamma.phi, pos.Phi=pos.Phi,
    pos.Gamma.theta=pos.Gamma.theta, pos.Theta=pos.Theta,
    pos.sigma=pos.sigma)
```

103.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    Gamma.phi <- array(parm[Data$pos.Gamma.phi],
        dim=c(Data$J, Data$J, Data$P))
    Phi.Sigma <- Gamma.phi * 10
    Phi.Sigma[Gamma.phi == 0] <- 0.1
    Phi <- array(parm[Data$pos.Phi], dim=c(Data$J, Data$J, Data$P))
    Gamma.theta <- array(parm[Data$pos.Gamma.theta],
        dim=c(Data$J, Data$J, Data$Q))
    Theta.Sigma <- Gamma.theta * 10
    Theta.Sigma[Gamma.theta == 0] <- 0.1
    Theta <- array(parm[Data$pos.Theta], dim=c(Data$J, Data$J, Data$Q))
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    Gamma.phi.prior <- sum(dbern(Gamma.phi, 0.5, log=TRUE))
    Phi.prior <- sum(dnorm(Phi, 0, Phi.Sigma, log=TRUE))
    Gamma.theta.prior <- sum(dbern(Gamma.theta, 0.5, log=TRUE))
    Theta.prior <- sum(dnorm(Theta, 0, Theta.Sigma, log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    mu <- matrix(alpha, Data$T, Data$J, byrow=TRUE)
    for (p in 1:Data$P)
        mu[(1+Data$L.P[p]):Data$T,] <- mu[(1+Data$L.P[p]):Data$T,] +
            Data$Y[1:(Data$T-Data$L.P[p]),] %*%
            (Gamma.phi[, , p] * Phi[, , p])
    epsilon <- Data$Y - mu
    for (q in 1:Data$Q)
```

```
    mu[(1+Data$L.Q[q]):Data$T,] <- mu[(1+Data$L.Q[q]):Data$T,] +
        epsilon[1:(Data$T-Data$L.Q[q]),] %*%
        (Gamma.theta[, , q] * Theta[, , q])
Sigma <- matrix(sigma, Data$T, Data$J, byrow=TRUE)
LL <- sum(dnorm(Data$Y[(1+Data$L.P[Data$P]):Data$T,],
    mu[(1+Data$L.P[Data$P]):Data$T,],
    Sigma[(1+Data$L.P[Data$P]):Data$T,], log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + Gamma.phi.prior + Phi.prior +
Gamma.theta.prior + Theta.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rnorm(prod(dim(mu)), mu, Sigma), parm=parm)
return(Modelout)
}
```

103.4. Initial Values

```
Initial.Values <- c(colMeans(Y), rep(1,J*J*P), runif(J*J*P,-1,1),
    rep(1,J*J*Q), rep(0,J*J*Q), rep(1,J))
```

104. VAR(p)-GARCH(1,1)-M

The Minnesota prior is applied to the VAR parameters, and the multivariate GARCH component is estimated with asymmetric BEKK. Compared to VAR(p) or VARMA(p,q), this is computationally intensive. However, it also tends to result in a substantial improvement when time for computation is feasible. This model also performs well when SSVS is applied to all parameters except $\mathbf{C}$, though it is even more computationally intensive, and is not shown here.

104.1. Form

$$\mathbf{Y}_{t,1:J} \sim \mathcal{N}_J(\mu_{t,1:J}, H_{1:J,1:J,t})$$

$$\mu_{t,j} = \alpha_j + \sum_{p=1}^{P} \Phi_{1:J,j,p}\mathbf{Y}_{t-p,j} + \sum \mathbf{H}_{1:J,j,t-1}\delta_{1:J,j}$$

$$\mathbf{H}_{,,t} = \Omega + \mathbf{A}^T\epsilon_{t-1,}\epsilon_{t-1}^T\mathbf{A} + \mathbf{B}^T\mathbf{H}_{,,t-1}\mathbf{B} + \mathbf{D}^T\zeta_{t-1,}\zeta_{t-1,}^T\mathbf{D}, \quad t = 2,\ldots,T$$

$$\Omega = \mathbf{C}\mathbf{C}^T$$

$$\alpha_j \sim \mathcal{N}(0, 1000)$$

$$\delta_{i,k} \sim \mathcal{N}(0, 1000), \quad i = 1,\ldots,J, \quad k = 1,\ldots,J$$

$$\Phi_{i,k,p} \sim \mathcal{N}(\Phi_{i,k,p}^{\mu}, \Sigma_{i,k,p}), \quad i = 1,\ldots,J, \quad k = 1,\ldots,J, \quad p = 1,\ldots,P$$

$$\mathbf{C}_{i,j} \sim \mathcal{N}(0, 100)$$

$$\mathbf{A}_{i,j} \sim \mathcal{N}(0, 100)$$

$$\mathbf{B}_{i,j} \sim \mathcal{N}(0, 100)$$

$$\mathbf{D}_{i,j} \sim \mathcal{N}(0, 100)$$

where Φ has a Minnesota prior, $\mathbf{C}$ is lower-triangular with positive-only diagonal elements, and $\mathbf{A}_{1,1}$, $\mathbf{B}_{1,1}$, and $\mathbf{D}_{1,1}$ must be positive.

104.2. Data

```
data(demonfx)
Y.orig <- as.matrix(demonfx[,1:3])
Y <- diff(log(Y.orig[1:100,]))
Y.scales <- sqrt(.colVars(Y))
Y <- Y / matrix(Y.scales, nrow(Y), ncol(Y), byrow=TRUE)
T <- nrow(Y)
J <- ncol(Y)
L <- c(1,5,20) #Autoregressive lags
P <- length(L) #Autoregressive order
Phi.mu <- array(0, dim=c(J,J,P))
Phi.mu[, , 1] <- diag(J)
C <- matrix(NA, J, J)
C[lower.tri(C, diag=TRUE)] <- 0
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,J), delta=matrix(0,J,J),
    Phi=array(0, dim=c(J,J,P)), C=C, A=matrix(0,J,J), B=matrix(0,J,J),
    D=matrix(0,J,J)))
pos.alpha <- grep("alpha", parm.names)
pos.delta <- grep("delta", parm.names)
pos.Phi <- grep("Phi", parm.names)
pos.C <- grep("C", parm.names)
pos.A <- grep("A", parm.names)
pos.B <- grep("B", parm.names)
pos.D <- grep("D", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(Data$J)
    delta <- rnorm(Data$J*Data$J)
    Phi <- runif(Data$J*Data$J*Data$P, -1, 1)
    C <- runif(Data$J*(Data$J+1)/2)
    A <- as.vector(diag(Data$J)) + runif(Data$J*Data$J, -0.1, 0.1)
    B <- as.vector(diag(Data$J)) + runif(Data$J*Data$J, -0.1, 0.1)
    D <- as.vector(diag(Data$J)) + runif(Data$J*Data$J, -0.1, 0.1)
    return(c(alpha, delta, Phi, C, A, B, D))
    }
MyData <- list(J=J, L=L, P=P, PGF=PGF, Phi.mu=Phi.mu, T=T, Y=Y,
    mon.names=mon.names, parm.names=parm.names, pos.alpha=pos.alpha,
    pos.delta=pos.delta, pos.Phi=pos.Phi, pos.C=pos.C, pos.A=pos.A,
```

```
    pos.B=pos.B, pos.D=pos.D)
```

104.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    delta <- matrix(parm[Data$pos.delta], Data$J, Data$J)
    Phi <- array(parm[Data$pos.Phi], dim=c(Data$J, Data$J, Data$P))
    C <- matrix(0, Data$J, Data$J)
    C[lower.tri(C, diag=TRUE)] <- parm[Data$pos.C]
    diag(C) <- abs(diag(C))
    parm[Data$pos.C] <- C[lower.tri(C, diag=TRUE)]
    Omega <- C %*% t(C)
    A <- matrix(parm[Data$pos.A], Data$J, Data$J)
    A[1,1] <- abs(A[1,1])
    parm[Data$pos.A] <- as.vector(A)
    B <- matrix(parm[Data$pos.B], Data$J, Data$J)
    B[1,1] <- abs(B[1,1])
    parm[Data$pos.B] <- as.vector(B)
    D <- matrix(parm[Data$pos.D], Data$J, Data$J)
    D[1,1] <- abs(D[1,1])
    parm[Data$pos.D] <- as.vector(D)
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    delta.prior <- sum(dnormv(delta, 0, 1000, log=TRUE))
    Sigma <- MinnesotaPrior(Data$J, lags=Data$L, lambda=1,
        theta=0.5, sqrt(diag(Omega)))
    Phi.prior <- sum(dnormv(Phi, Data$Phi.mu, Sigma, log=TRUE))
    C.prior <- sum(dnormv(C[lower.tri(C, diag=TRUE)], 0, 100, log=TRUE))
    A.prior <- sum(dnormv(A, 0, 100, log=TRUE))
    B.prior <- sum(dnormv(B, 0, 100, log=TRUE))
    D.prior <- sum(dnormv(D, 0, 100, log=TRUE))
    ### Log-Likelihood
    mu <- matrix(alpha, Data$T, Data$J, byrow=TRUE)
    for (p in 1:Data$P)
        mu[(1+Data$L[p]):Data$T,] <- mu[(1+Data$L[p]):Data$T,] +
            Data$Y[1:(Data$T-Data$L[p]),] %*% Phi[, , p]
    LL <- 0
    Yhat <- Data$Y
    H <- array(Omega, dim=c(Data$J, Data$J, Data$T))
    for (t in 2:Data$T) {
        eps <- Data$Y - mu
        zeta <- matrix(interval(eps, -Inf, 0, reflect=FALSE), Data$T,
            Data$J)
```

```
        part1 <- t(A) %*% eps[t-1,] %*% t(eps[t-1,]) %*% A
        part2 <- t(B) %*% H[, , t-1] %*% B
        part3 <- t(D) %*% zeta[t-1,] %*% t(zeta[t-1,]) %*% D
        H0 <- Omega + part1 + part2 + part3
        H0[upper.tri(H0, diag=TRUE)] <- t(H0)[upper.tri(H0, diag=TRUE)]
        H[, , t] <- H0
        mu[t-1,] <- mu[t-1,] + colMeans(H[, , t-1]*delta)
        Sigma <- MinnesotaPrior(Data$J, lags=Data$L, lambda=1,
            theta=0.5, sqrt(diag(H[, , t])))
        Phi.prior <- Phi.prior + sum(dnormv(Phi, Data$Phi.mu, Sigma,
            log=TRUE))
        LL <- LL + dmvn(Y[t,], mu[t,], H[, , t], log=TRUE)
        Yhat[t,] <- rmvn(1, mu[t,], H[, , t])
        }
    Phi.prior <- Phi.prior / Data$T
    ### Log-Posterior
    LP <- LL + alpha.prior + delta.prior + Phi.prior + C.prior +
        A.prior + B.prior + D.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP, yhat=Yhat, parm=parm)
    return(Modelout)
    }
```

104.4. Initial Values

```
Initial.Values <- c(colMeans(Y), rnorm(J*J), runif(J*J*P,-1,1),
    runif(J*(J+1)/2), as.vector(diag(J)), as.vector(diag(J)),
    as.vector(diag(J)))
```

105. VAR(p) - Minnesota Prior

This is an example of a vector autoregression or VAR with P lags that uses the Minnesota prior to estimate Σ.

105.1. Form

$$\mathbf{Y}_{t,j} \sim \mathcal{N}(\mu_{t,j}, \sigma_j^2), \quad t = 1, \ldots, T, \quad j = 1, \ldots, J$$

$$\mu_{t,j} = \alpha_j + \sum_{p=1}^{P} \Phi_{1:J,j,p} \mathbf{Y}_{t-p,j}$$

$$\mathbf{y}_j^{new} = \alpha_j + \Phi_{1:J,j} \mathbf{Y}_{T,j}$$

$$\alpha_j \sim \mathcal{N}(0, 1000)$$

$$\Phi_{i,k,p} \sim \mathcal{N}(\Phi_{i,k,p}^{\mu}, \Sigma_{i,k,p}), \quad i = 1, \ldots, J, \quad k = 1, \ldots, J, \quad p = 1, \ldots, P$$

$$\sigma_j \sim \mathcal{HC}(25)$$

where Φ^{μ} and Σ are set according to the Minnesota prior.

105.2. Data

```
data(demonfx)
Y.orig <- as.matrix(demonfx[,1:3])
Y <- diff(log(Y.orig[1:100,]))
Y.scales <- sqrt(.colVars(Y))
Y <- Y / matrix(Y.scales, nrow(Y), ncol(Y), byrow=TRUE)
T <- nrow(Y)
J <- ncol(Y)
L <- c(1,5,20) #Autoregressive lags
P <- length(L) #Autoregressive order
Phi.mu <- array(0, dim=c(J,J,P))
Phi.mu[, , 1] <- diag(J)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,J),
    Phi=array(0, dim=c(J,J,P)), sigma=rep(0,J)))
pos.alpha <- grep("alpha", parm.names)
pos.Phi <- grep("Phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(Data$J)
    Phi <- runif(Data$J*Data$J*Data$P, -1, 1)
    sigma <- runif(Data$J)
    return(c(alpha, Phi, sigma))
    }
MyData <- list(J=J, L=L, P=P, PGF=PGF, Phi.mu=Phi.mu, T=T, Y=Y,
    mon.names=mon.names, parm.names=parm.names, pos.alpha=pos.alpha,
    pos.Phi=pos.Phi, pos.sigma=pos.sigma)
```

105.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    Phi <- array(parm[Data$pos.Phi], dim=c(Data$J, Data$J, Data$P))
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    Sigma <- MinnesotaPrior(Data$J, lags=Data$L, lambda=1, theta=0.5,
        sigma)
    Phi.prior <- sum(dnormv(Phi, Data$Phi.mu, Sigma, log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
```

```
### Log-Likelihood
mu <- matrix(alpha, Data$T, Data$J, byrow=TRUE)
for (p in 1:Data$P) {
     mu[(1+Data$L[p]):Data$T,] <- mu[(1+Data$L[p]):Data$T,] +
     Data$Y[1:(Data$T-Data$L[p]),] %*% Phi[ , , p]}
Sigma <- matrix(sigma, Data$T, Data$J, byrow=TRUE)
LL <- sum(dnorm(Data$Y[(1+Data$L[Data$P]):Data$T,],
     mu[(1+Data$L[Data$P]):Data$T,],
     Sigma[(1+Data$L[Data$P]):Data$T,], log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + Phi.prior + sigma.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
     yhat=rnorm(prod(dim(mu)), mu, Sigma), parm=parm)
return(Modelout)
}
```

105.4. Initial Values

```
Initial.Values <- c(as.vector(colMeans(Y)), rep(0,J*J*P), rep(1,J))
```

106. VAR(p) - SSVS

Stochastic search variable selection (SSVS) is applied to VAR autoregressive parameters. Note that the constants for the mixture variances are typically multiplied by the posterior standard deviations from an unrestricted VAR that was updated previously, and these are not included in this example.

106.1. Form

$$\mathbf{Y}_{t,j} \sim \mathcal{N}(\mu_{t,j}, \sigma_j^2), \quad t = 1, \ldots, T, \quad j = 1, \ldots, J$$

$$\mu_{t,j} = \alpha_j + \sum_{p=1}^{P} \Gamma_{1:J,j,p} \Phi_{1:J,j,p} \mathbf{Y}_{t-p,j}$$

$$\alpha_j \sim \mathcal{N}(0, 1000)$$

$$\Gamma_{i,k,p} \sim \mathcal{BERN}(0.5), \quad i = 1, \ldots, J, \quad k = 1, \ldots, J, \quad p = 1, \ldots, P$$

$$(\Phi_{i,k,p} | \Gamma_{i,k,p}) \sim (1 - \Gamma_{i,k,p}) \mathcal{N}(0, 0.01) + \Gamma_{i,k,p} \mathcal{N}(0, 10), \quad i = 1, \ldots, J, \quad k = 1, \ldots, J, \quad p = 1, \ldots, P$$

$$\sigma_j \sim \mathcal{HC}(25)$$

106.2. Data

```
data(demonfx)
Y.orig <- as.matrix(demonfx[,1:3])
Y <- diff(log(Y.orig[1:100,]))
```

```
Y.scales <- sqrt(.colVars(Y))
Y <- Y / matrix(Y.scales, nrow(Y), ncol(Y), byrow=TRUE)
T <- nrow(Y)
J <- ncol(Y)
L <- c(1,5,20) #Autoregressive lags
P <- length(L) #Autoregressive order
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,J),
Gamma=array(0, dim=c(J,J,P)), Phi=array(0, dim=c(J,J,P)),
sigma=rep(0,J)))
pos.alpha <- grep("alpha", parm.names)
pos.Gamma <- grep("Gamma", parm.names)
pos.Phi <- grep("Phi", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    alpha <- rnorm(Data$J)
    Gamma <- rep(1, Data$J*Data$J*Data$P)
    Phi <- runif(Data$J*Data$J*Data$P, -1, 1)
    sigma <- runif(Data$J)
    return(c(alpha, Gamma, Phi, sigma))
    } MyData <- list(J=J, L=L, P=P, PGF=PGF, T=T, Y=Y, mon.names=mon.names,
    parm.names=parm.names, pos.alpha=pos.alpha, pos.Gamma=pos.Gamma,
    pos.Phi=pos.Phi, pos.sigma=pos.sigma)
```

106.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    alpha <- parm[Data$pos.alpha]
    Gamma <- array(parm[Data$pos.Gamma], dim=c(Data$J, Data$J, Data$P))
    Phi.Sigma <- Gamma * 10
    Phi.Sigma[Gamma == 0] <- 0.1
    Phi <- array(parm[Data$pos.Phi], dim=c(Data$J, Data$J, Data$P))
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    alpha.prior <- sum(dnormv(alpha, 0, 1000, log=TRUE))
    Gamma.prior <- sum(dbern(Gamma, 0.5, log=TRUE))
    Phi.prior <- sum(dnorm(Phi, 0, Phi.Sigma, log=TRUE))
    sigma.prior <- sum(dhalfcauchy(sigma, 25, log=TRUE))
    ### Log-Likelihood
    mu <- matrix(alpha, Data$T, Data$J, byrow=TRUE)
    for (p in 1:Data$P)
        mu[(1+Data$L[p]):Data$T,] <- mu[(1+Data$L[p]):Data$T,] +
        Data$Y[1:(Data$T-Data$L[p]),] %*% (Gamma[, , p]*Phi[, , p])
```

```
    Sigma <- matrix(sigma, Data$T, Data$J, byrow=TRUE)
    LL <- sum(dnorm(Data$Y[(1+Data$L[Data$P]):Data$T,],
         mu[(1+Data$L[Data$P]):Data$T,],
         Sigma[(1+Data$L[Data$P]):Data$T,], log=TRUE))
    ### Log-Posterior
    LP <- LL + alpha.prior + Gamma.prior + Phi.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
         yhat=rnorm(prod(dim(mu)), mu, Sigma), parm=parm)
    return(Modelout)
    }
```

106.4. Initial Values

```
Initial.Values <- c(colMeans(Y), rep(1,J*J*P), runif(J*J*P,-1,1), rep(1,J))
```

107. Weighted Regression

It is easy enough to apply record-level weights to the likelihood. Here, weights are applied to the linear regression example in section 48.

107.1. Form

$$\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$$
$$\mu = \mathbf{X}\beta$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J$$
$$\sigma \sim \mathcal{HC}(25)$$

107.2. Data

```
data(demonsnacks)
N <- nrow(demonsnacks)
J <- ncol(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(demonsnacks[,c(1,3:10)]))
for (j in 2:J) X[,j] <- CenterScale(X[,j])
w <- c(rep(1,5), 0.2, 1, 0.01, rep(1,31))
w <- w * (sum(w) / N)
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
    beta <- rnorm(Data$J)
```

```
    sigma <- runif(1)
    return(c(beta, sigma))
    }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
    parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, w=w,
    y=y)
```

107.3. Model

```
Model <- function(parm, Data)
    {
    ### Parameters
    beta <- parm[Data$pos.beta]
    sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
    parm[Data$pos.sigma] <- sigma
    ### Log-Prior
    beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
    sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
    ### Log-Likelihood
    mu <- tcrossprod(Data$X, t(beta))
    LL <- sum(w * dnorm(Data$y, mu, sigma, log=TRUE))
    ### Log-Posterior
    LP <- LL + beta.prior + sigma.prior
    Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
        yhat=rnorm(length(mu), mu, sigma), parm=parm)
    return(Modelout)
    }
```

107.4. Initial Values

```
Initial.Values <- c(rep(0,J), 1)
```

108. Zero-Inflated Poisson (ZIP)

108.1. Form

$$\mathbf{y} \sim \mathcal{P}(\Lambda_{1:N,2})$$

$$\mathbf{z} \sim \mathcal{BERN}(\Lambda_{1:N,1})$$

$$\mathbf{z}_i = \begin{cases} 1 & \text{if } \mathbf{y}_i = 0 \\ 0 & \end{cases}$$

$$\Lambda_{i,2} = \begin{cases} 0 & \text{if } \Lambda_{i,1} \geq 0.5 \\ \Lambda_{i,2} & \end{cases}$$

$$\Lambda_{1:N,1} = \frac{1}{1+\exp(-\mathbf{X}_1\alpha)}$$
$$\Lambda_{1:N,2} = \exp(\mathbf{X}_2\beta)$$
$$\alpha_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J_1$$
$$\beta_j \sim \mathcal{N}(0, 1000), \quad j = 1, \ldots, J_2$$

108.2. Data

```
N <- 1000
J1 <- 4
J2 <- 3
X1 <- matrix(runif(N*J1,-2,2),N,J1); X1[,1] <- 1
X2 <- matrix(runif(N*J2,-2,2),N,J2); X2[,1] <- 1
alpha <- runif(J1,-1,1)
beta <- runif(J2,-1,1)
p <- invlogit(tcrossprod(X1, t(alpha)) + rnorm(N,0,0.1))
mu <- round(exp(tcrossprod(X2, t(beta)) + rnorm(N,0,0.1)))
y <- ifelse(p > 0.5, 0, mu)
z <- ifelse(y == 0, 1, 0)
mon.names <- "LP"
parm.names <- as.parm.names(list(alpha=rep(0,J1), beta=rep(0,J2)))
pos.alpha <- grep("alpha", parm.names)
pos.beta <- grep("beta", parm.names)
PGF <- function(Data) {
     alpha <- rnorm(Data$J1)
     beta <- rnorm(Data$J2)
     return(c(alpha, beta))
     }
MyData <- list(J1=J1, J2=J2, N=N, PGF=PGF, X1=X1, X2=X2,
     mon.names=mon.names, parm.names=parm.names, pos.alpha=pos.alpha,
     pos.beta=pos.beta, y=y, z=z)
```

108.3. Model

```
Model <- function(parm, Data)
     {
     ### Parameters
     parm[Data$pos.alpha] <- alpha <- interval(parm[Data$pos.alpha], -5, 5)
     parm[Data$pos.beta] <- beta <- interval(parm[Data$pos.beta], -5, 5)
     ### Log-Prior
     alpha.prior <- sum(dnormv(alpha, 0, 5, log=TRUE))
     beta.prior <- sum(dnormv(beta, 0, 5, log=TRUE))
     ### Log-Likelihood
     Lambda <- matrix(NA, Data$N, 2)
     Lambda[,1] <- invlogit(tcrossprod(Data$X1, t(alpha)))
```

```
Lambda[,2] <- exp(tcrossprod(Data$X2, t(beta))) + 1e-100
Lambda[which(Lambda[,1] >= 0.5),2] <- 0
LL <- sum(dbern(Data$z, Lambda[,1], log=TRUE),
    dpois(Data$y, Lambda[,2], log=TRUE))
### Log-Posterior
LP <- LL + alpha.prior + beta.prior
Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
    yhat=rpois(nrow(Lambda), Lambda[,2]), parm=parm)
return(Modelout)
}
```

108.4. Initial Values

```
Initial.Values <- GIV(Model, MyData, n=10000)
```

References

Congdon P (2003). *Applied Bayesian Modelling*. John Wiley & Sons, West Sussex, England.

Draper D (1995). "Assessment and Propagation of Model Uncertainty." *Journal of the Royal Statistical Society*, **B 57**(1), 45–97.

Gelman A (2013). ***R2WinBUGS**: Running WinBUGS and OpenBUGS from R / S-PLUS*. R package version 2.1-19, URL http://cran.r-project.org/web/packages/R2WinBUGS/index.html.

Gelman A, Carlin J, Stern H, Rubin D (2004). *Bayesian Data Analysis*. 2nd edition. Chapman & Hall, Boca Raton, FL.

Ibrahim J, Chen M (2000). "Power Prior Distributions for Regression Models." *Statistical Science*, **15**, 46–60.

Kim J, Allenby G, Rossi P (2002). "Modeling Consumer Demand for Variety." *Marketing Science*, **21**(3), 229–250.

Kleine L (1950). *Economic Fluctuations in the United States 1921-1940*. John Wiley & Sons, New York, New York.

Kotz S, Kozubowski T, Podgorski K (2001). *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*. Birkauser, Boston.

O'Hara R, Sillanpaa M (2009). "A Review of Bayesian Variable Selection Methods: What, How and Which." *Journal of Bayesian Analysis*, **4**(1), 85–118.

Spiegelhalter D, Thomas A, Best N, Lunn D (2003). *WinBUGS User Manual, Version 1.4*. MRC Biostatistics Unit, Institute of Public Health and Department of Epidemiology and Public Health, Imperial College School of Medicine, UK.

Statisticat LLC (2015). ***LaplacesDemon**: Complete Environment for Bayesian Inference*. R package version 15.03.19, URL http://www.bayesian-inference.com/software.

Zellner A (1962). "An Efficient Method of Estimating Seemingly Unrelated Regression Equations and Tests for Aggregation Bias." *Journal of the American Statistical Association*, **57**, 348–368.

Affiliation:

Statisticat, LLC
Farmington, CT
E-mail: defunct
URL: https://web.archive.org/web/20141224051720/http://www.bayesian-inference.com/index