SDS 383D, Exercises 4: Hierarchical Models: Data-analysis Problems

Jan-Michael Cabrera

April 26, 2019

Price elasticity of demand

The data in "cheese.csv" are about sales volume, price, and advertisting display activity for packages of Borden sliced "cheese." The data are taken from Rossi, Allenby, and McCulloch's textbook on *Bayesian Statistics and Marketing.* For each of 88 stores (store) in different US cities, we have repeated observations of the weekly sales volume (vol, in terms of packages sold), unit price (price), and whether the product was advertised with an in-store display during that week (disp = 1 for display).

Your goal is to estimate, on a store-by-store basis, the effect of display ads on the demand curve for cheese. A standard form of a demand curve in economics is of the form $Q = \alpha P^{\beta}$, where Q is quantity demanded (i.e. sales volume), P is price, and α and β are parameters to be estimated. You'll notice that this is linear on a log-log scale,

$$\log P = \log \alpha + \beta \log Q$$

which you should feel free to assume here. Economists would refer to β as the price elasticity of demand (PED). Notice that on a log-log scale, the errors enter multiplicatively.

There are several things for you to consider in analyzing this data set.

1. The demand curve might shift (different α) and also change shape (different β) depending on whether there is a display ad or not in the store.
2. Different stores will have very different typical volumes, and your model should account for this.
3. Do different stores have different PEDs? If so, do you really want to estimate a separate, unrelated β for each store?
4. If there is an effect on the demand curve due to showing a display ad, does this effect differ store by store, or does it look relatively stable across stores?
5. Once you build the best model you can using the log-log specification, do see you any evidence of major model mis-fit?

Propose an appropriate hierarchical model that allows you to address these issues, and use Gibbs sampling to fit your model.

We begin by specifying the form of the hierarchical linear model:

$$\bar{y}_i = X_i^T \beta_i + e_i; \quad e_i \sim N(0, \sigma_i^2 I_{n_i})$$

where the index i corresponds to a particular store. The covariates for a particular store X_i^T is a matrix with n_i rows corresponding to the number of samples taken for store i and p columns:

$$X_i^T = \begin{bmatrix} 1 & \log P_{ij} & \delta_{ij} & \delta_{ij}\log P_{ij} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \log P_{in_i} & \delta_{in_i} & \delta_{in_i}\log P_{in_i}. \end{bmatrix}$$

Here δ_{ij} corresponds to whether sample j for store i had a display or not.

Each β_i has a prior of the form,

$$\beta_i \sim MVN(\theta, V).$$

For store i the likelihood is multivariate normal with

$$p(\bar{y}_i|\beta_i, \sigma_i^2) \propto \exp\left[-\frac{1}{2}(\bar{y}_i - X_i^T\beta_i)^T \sigma_i^2 I_{n_i}(\bar{y}_i - X_i^T\beta_i)\right]$$

Here we show the prior for β_i explicitly

$$p(\beta_i|\theta, V) \propto \exp\left[-\frac{1}{2}(\beta_i - \theta)^T V^{-1}(\beta_i - \theta)\right].$$

We would like to find the conditional distribution for β_i given the data given by

$$p(\beta_i|\bar{y}_i, \sigma_i^2, \theta, V) \propto p(\beta_i|\theta, V)p(\bar{y}_i|\beta_i, \sigma_i^2).$$

Combining the likelihood and prior and completing the square we find that the posterior for β_i is also multivariate normal:

$$\beta_i|\bar{y}_i \cdots \sim MVN\left(\frac{V^{-1}\theta + \frac{1}{\sigma_i^2}X_i\bar{y}_i}{V^{-1} + X_i\frac{1}{\sigma_i^2 I_{n_i}}X_i^T}, \left[V^{-1} + X_i\frac{1}{\sigma_i^2 I_{n_i}}X_i^T\right]^{-1}\right)$$

The prior for the scaling parameter is assumed to be an improper non-informative Jeffrey's prior:

$$p(\sigma_i^2) \propto \frac{1}{\sigma_i^2}.$$

The conditional distribution is then,

$$\begin{aligned} p(\sigma_i^2|\bar{y}_i) &\propto p(\sigma_i^2)p(\bar{y}_i|\beta_i, \sigma_i^2) \\ &\propto \frac{1}{\sigma_i^2}\left(\frac{1}{\sigma_i^2}\right)^{n_i/2} \exp\left[-\frac{1}{2}(\bar{y}_i - X_i^T\beta_i)^T\sigma_i^2 I_{n_i}(\bar{y}_i - X_i^T\beta_i)\right]. \end{aligned}$$

The conditional distribution has the form of an inverse-gamma distribution with the following parameters:

$$\sigma_i^2|\bar{y}_i \cdots \sim IG\left(\frac{n_i}{2}, \frac{1}{2}(\bar{y}_i - X_i^T\beta_i)^T(\bar{y}_i - X_i^T\beta_i)\right).$$

The prior for θ is assumed to be flat and the conditional takes the form:

$$p(\theta|\beta) \propto \exp\left[-\frac{1}{2}\sum_{i=1}^{s}(\beta_i-\theta)^T V^{-1}(\beta_i-\theta)\right].$$

This reduces to a multivariate normal distribution of the from:

$$\theta|\beta\cdots \sim MVN\left(\frac{1}{s}\sum_{i=1}^{s}\beta_i, \frac{1}{s}V\right)$$

The prior for the variance for the coefficients, β_i is assumed to have the form of an inverse-Wishart prior.

$$p(V) \propto |V|^{(d+p+1)/2}\exp\left[-\frac{1}{2}\mathrm{tr}(CV^{-1})\right]$$

The 'likelihood' for this prior comes from the coefficients and has the form,

$$p(\beta_i|\theta,V) \propto |\mathrm{det}V|^{1/2}\exp\left[-\frac{1}{2}(\beta_i-\theta)^T V^{-1}(\beta_i-\theta)\right],$$

noting that the term in the determinant is necessary for finding the conditional distribution for the variance. The full 'likelihood' is then,

$$p(\beta|\theta,V) \propto |\mathrm{det}V|^{s/2}\exp\left[-\frac{1}{2}\sum_{i=1}^{s}(\beta_i-\theta)^T V^{-1}(\beta_i-\theta)\right]$$

Combining this with the inverse-Wishart prior, we can show that the condition distribution is also inverse-Wishart:

$$\begin{aligned} p(V|\beta) &\propto |V|^{(d+p+1)/2}\exp\left[-\frac{1}{2}\mathrm{tr}(CV^{-1})\right]|\mathrm{det}V|^{s/2}\exp\left[-\frac{1}{2}\sum_{i=1}^{s}(\beta_i-\theta)^T V^{-1}(\beta_i-\theta)\right] \\ &\propto |V|^{(d+p+s+1)/2}\exp\left[-\frac{1}{2}\mathrm{tr}\left(V^{-1}\left[C+\sum_{i=1}^{s}(\beta_i-\theta)^T(\beta_i-\theta)\right]\right)\right] \end{aligned}$$

$$V|\beta\cdots \sim IW\left(d+s, C+\sum_{i=1}^{s}(\beta_i-\theta)^T(\beta_i-\theta)\right)$$

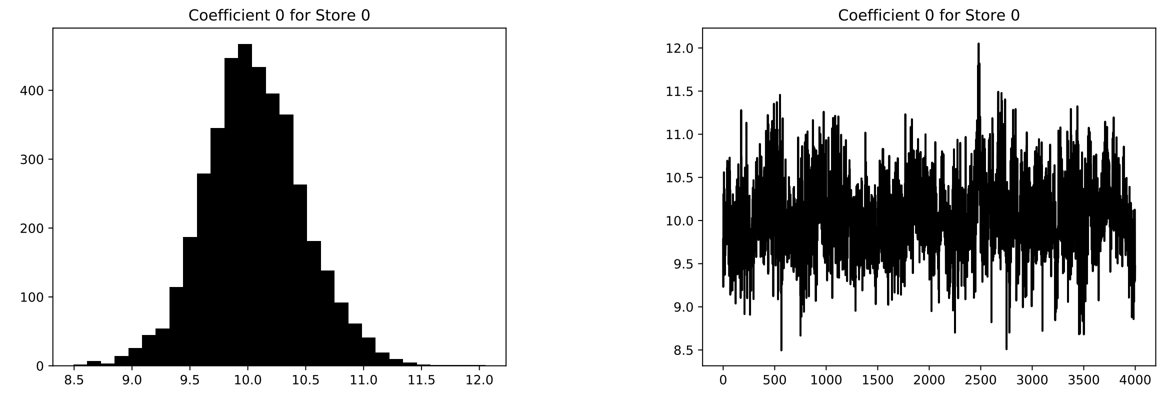

Figure 1: Histogram and trace for coefficient zero for store zero

Conditionals for Gibbs Sampling

$$\beta_i|\bar{y}_i \cdots \sim MVN\left(\frac{V^{-1}\theta + \frac{1}{\sigma_i^2}X_i\bar{y}_i}{V^{-1} + X_i\frac{1}{\sigma_i^2 I_{n_i}}X_i^T}, \left[V^{-1} + X_i\frac{1}{\sigma_i^2 I_{n_i}}X_i^T\right]^{-1}\right)$$

$$\sigma_i^2|\bar{y}_i \cdots \sim IG\left(\frac{n_i}{2}, \frac{1}{2}(\bar{y}_i - X_i^T\beta_i)^T(\bar{y}_i - X_i^T\beta_i)\right)$$

$$\theta|\beta \cdots \sim MVN\left(\frac{1}{s}\sum_{i=1}^{s}\beta_i, \frac{1}{s}V\right)$$

$$V|\beta \cdots \sim IW\left(d+s, C + \sum_{i=1}^{s}(\beta_i - \theta)^T(\beta_i - \theta)\right)$$

A Gibbs sampler was written using Python 3.7 to sample from the derived conditional distributions (see Appendix and repo). The linear fits for all the stores is shown in figure 2. Here the black dots represent data from stores where a display was not present and the red dots represent data from stores where the display was present. The black and red lines correspond the fits for display not present and present respectively.

A noteworthy trend is that for many of the stores, having a display increased sales (i.e. volume) but also tended to increase price sensitivity (shown by the steeper slopes). The model also appears to do supprisingly well for stores where little or no data is available. Figures 3, 4, 5 show examples of these cases.

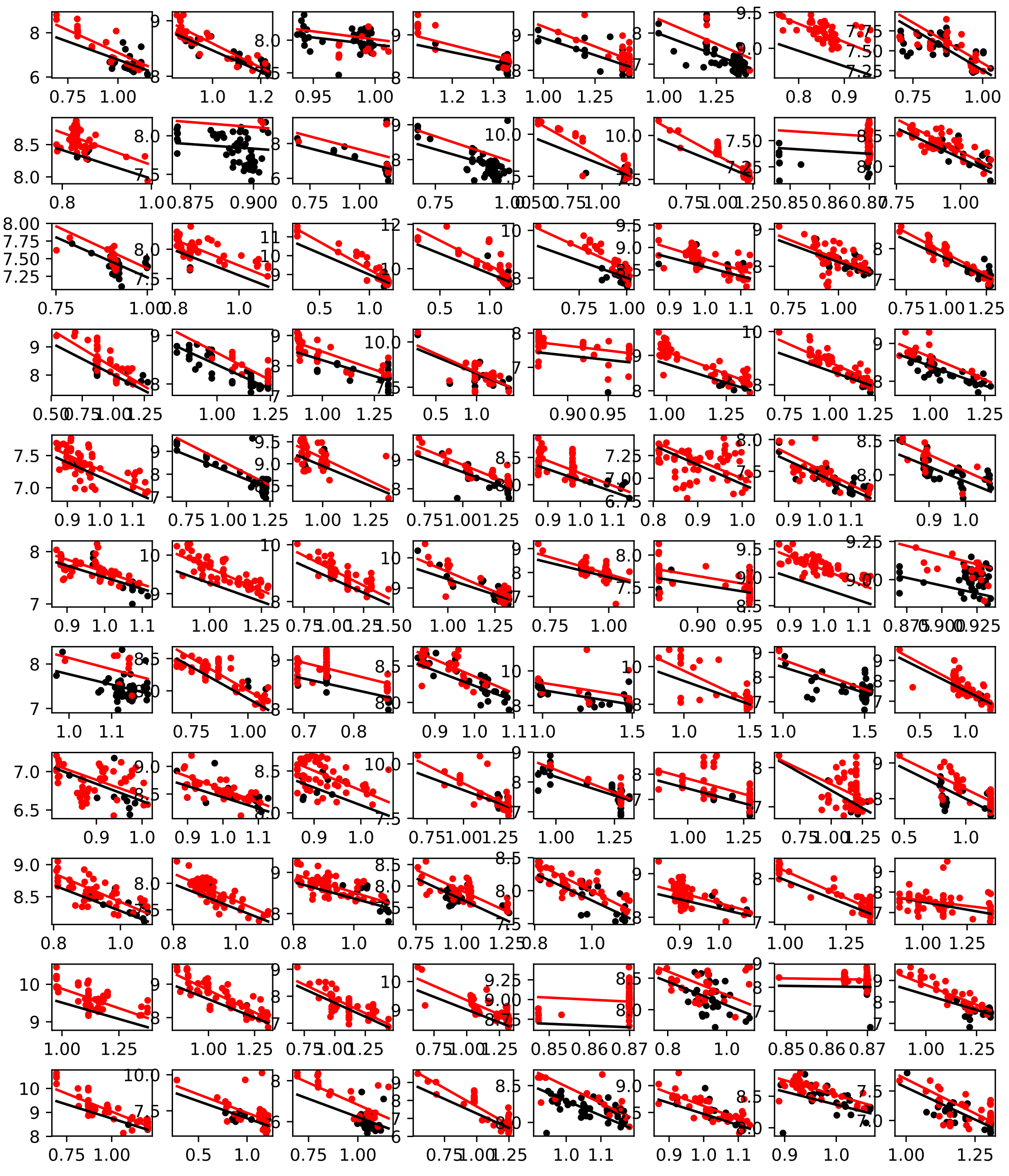

Figure 2: Data and fits of posterior means for all stores

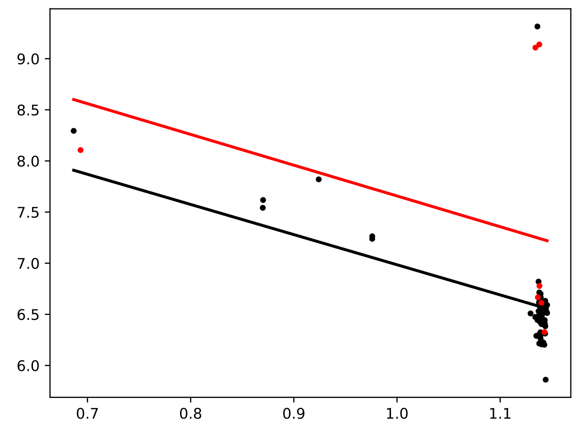

Figure 3: Data and fit for store 10

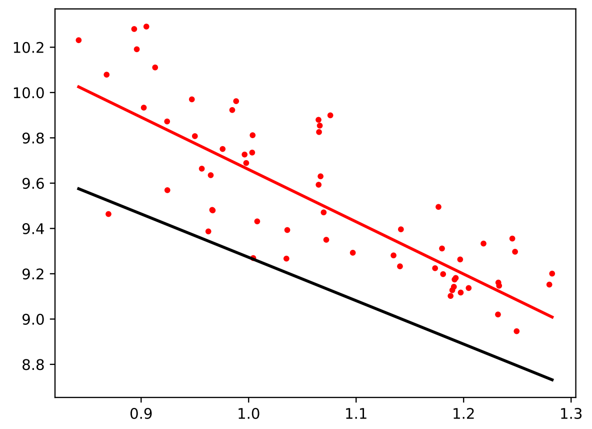

Figure 4: Data and fit for store 41

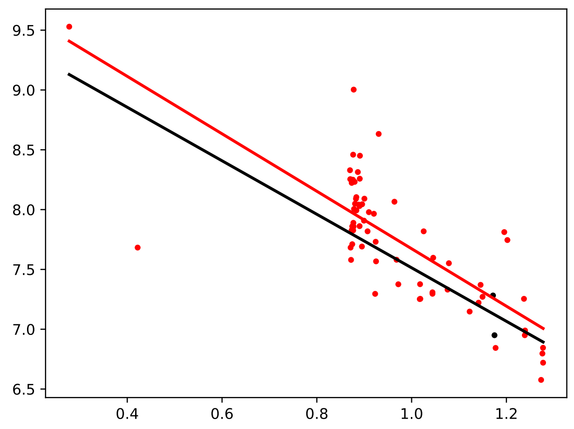

Figure 5: Data and fit for store 55

cheese.py

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sys
sys.path.append('../../scripts/')
from samplers import Trace
from samplers import Gibbs
import scipy.stats as stats
from numpy.linalg import inv

np.random.seed(3)

# Import data
df = pd.read_csv('../../data/cheese.csv', delimiter=',')

# Create an array of unique store names
stores = np.unique(df.store)

# Number of data rows
n = 5555
p = 4
d = 10

# Instantiate lists for storing sorted data
data = []
y = []
X = []

# Sort data by store
for s, store in enumerate(stores):
    data.append(df[df.store==store])

    # \bar{y}_i
    y.append(np.log(data[s].vol))
    data[s].vol = np.log(data[s].vol)
    data[s].price = np.log(data[s].price)

    # X_i^T
    X.append(np.array([np.ones(data[s].shape[0]), data[s].price, data[s].disp, data[s].price*data[s↩
        ].disp]).T)

iterations = 5000

# Wrapper for running the model
def run_model(X, y, iterations):
    model = Gibbs(X, y, samples=iterations)
    model.cheese()

# run_model(X, y, iterations)

# Load trace
burn=1000
beta_trace = np.load('traces/cheese/beta_trace.npy')
beta_trace_mean = np.mean(beta_trace[burn:], axis=0)

# Example Trace and Histogram Plot showing convergence
plt.figure()
plt.title('Coefficient 0 for Store 0')
plt.plot(beta_trace[burn:,0,0], color='k')
# plt.show()
plt.savefig('figures/coeff_0_store_0_trace.pdf')

plt.figure()
plt.title('Coefficient 0 for Store 0')
plt.hist(beta_trace[burn:,0,0],bins = 30, color='k')
# plt.show()
plt.savefig('figures/coeff_0_store_0_hist.pdf')

# Plot all stores together into one plot
fig, ax = plt.subplots(11, 8, figsize=(10,12))
fig.subplots_adjust(hspace=0.6)
n = 0
for i in range(11):
    for j in range(8):
        x_hat = X[n][X[n][:,1].argsort()]
        ax[i,j].plot(data[n][data[n].disp==0].price, data[n][data[n].disp==0].vol, '.k', linewidth↩
            =0.1)
        ax[i,j].plot(data[n][data[n].disp==1].price, data[n][data[n].disp==1].vol, '.r', linewidth↩
            =0.1)
        ax[i,j].plot(x_hat[:,1], beta_trace_mean[n][0]+beta_trace_mean[n][1]*x_hat[:,1], '-k')
        ax[i,j].plot(x_hat[:,1], (beta_trace_mean[n][0]+beta_trace_mean[n][2])+(beta_trace_mean[n↩
            ][1]+beta_trace_mean[n][3])*x_hat[:,1], '-r')
        n+=1

# plt.show()
plt.savefig('figures/cheese_all_plots.pdf')

# Plot selected stores
i = 10, 41, 55
for p, plots in enumerate(i):
    plt.figure()
    x_hat = X[plots][X[plots][:,1].argsort()]
    plt.plot(data[plots][data[plots].disp==0].price, data[plots][data[plots].disp==0].vol, '.k', ↩
        linewidth=2)
    plt.plot(data[plots][data[plots].disp==1].price, data[plots][data[plots].disp==1].vol, '.r', ↩
        linewidth=2)
    plt.plot(x_hat[:,1], beta_trace_mean[plots][0]+beta_trace_mean[plots][1]*x_hat[:,1], '-k', ↩
        linewidth=2)
    plt.plot(x_hat[:,1], (beta_trace_mean[plots][0]+beta_trace_mean[plots][2])+(beta_trace_mean[↩
        plots][1]+beta_trace_mean[plots][3])*x_hat[:,1], '-r', linewidth=2)
    # plt.show()
    plt.savefig('figures/cheese_plots_'+str(plots)+'.pdf')
```

samplers.py

```
from __future__ import division
import numpy as np
ifrom __future__ import division
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
from numpy.linalg import inv

class Trace:
    """
    This class instantiates traces
    """
    def __init__(self, name):
        """
        Parameters
        ----------
        name: str
            name of the trace for file i/o purposes

        iterations: int
            number of iterations to be ran for a given trace
        shape: int
            sets size of trace, default=1

        """
        self.name = name
        self.trace = []

    def update_trace(self, value):
        """
        Parameters
        ----------
            index: int
                point in trace to update
            value: float
                the value passed to trace at index

        """
        self.trace.append(value)

    def mean(self):
        """
        Returns
        ----------
            mean: float
                calculates the mean for each column in a given trace

        """
        mean = np.mean(np.asarray(self.trace), axis=0)
        return mean

    def save_trace(self, out_directory=''):
        np.save(out_directory+self.name+'_trace', np.asarray(self.trace))

class Gibbs:
    """
    This class will use a Gibbs sampling methods to draw full conditional distributions
    """
    def __init__(self, X, Y, samples=100):
        """
        Parameters
        ----------
            X:  float
                feature matrix

            Y:  float
                response vector

            samples: int
                number of samples to be drawn from posteriors, default is 100 for debugging ↩
                    purposes

        """
        self.X = X
        self.Y = Y
        self.samples = samples

    def cheese(self):
        """
        Attributes
        ----------
            beta_trace: array
                samples from conditional distribution for beta

            mu_trace: array
                samples from conditional distribution for beta
        """
        out_directory = 'traces/cheese/'
        iterations = self.samples

        # Instantiate traces
        beta_trace = Trace('beta')
        sigma_trace = Trace('sigma')
        V_trace = Trace('V')

        # Number of data rows
        n = 5555
        p = 4
        d = 10

        # Number of stores
        s = 88

        ### Instantiate priors
        theta = np.zeros(p)
        V = np.eye(p)*10**6

        beta = np.zeros((s,p))
        sigma_sq = np.ones(s)

        #### Iterations
        # iterations = 5000
        # burn = 500
        for j in range(iterations):
            # Store variable for inverse-Wishart
            B = 0
            # Iterate over stores to calculate \beta_i's
            for store in range(s):

                n = len(self.Y[store])
                # beta_cov = [V^{-1} + X_i \frac{1}{\sigma_i^2} I_{n_i} X_i^T]^{-1}
                beta_cov = inv(inv(V) + (1/sigma_sq[store]) * self.X[store].T @ self.X[store])

                # beta_mean = beta_cov [V^{-1} \theta + \frac{1}{\sigma_i^2}X_i \bar{y}_i]
                beta_mean = beta_cov @ (inv(V) @ theta + (1/sigma_sq[store]) * (self.X[store] @ ↩
                    self.Y[store]))

                # Sample from multivariate normal for each store
                beta[store] = stats.multivariate_normal.rvs(mean=beta_mean, cov=beta_cov)

                # Shape and scale parameters for sigma
                a = n/2
                b = (self.Y[store] - self.X[store] @ beta[store]).T @ (self.Y[store] - self.X[store↩
                    ] @ beta[store])/2

                # Sample from inverse-gamma distribution
                sigma_sq[store] = stats.invgamma.rvs(a, 1/b)

                # Sum variable for inverse-Wishart
                B += np.tensordot((beta[store] - theta),(beta[store] - theta).T, axes=0)

            # Sample from multivarate normal, theta_mean = 1/s \sum_{i=1}^s \beta_i, theta_cov = V/↩
                s
            theta = stats.multivariate_normal.rvs(mean=(1/s)*beta.sum(axis=0), cov=V/s)

            # Sample from inverse-Wishart distribution
            V = stats.invwishart.rvs(d+s, np.eye(p) + B)

            # Append samples to trace
            beta_trace.update_trace(beta.copy())
            sigma_trace.update_trace(sigma_sq.copy())
            V_trace.update_trace(V.copy())

        # Save traces for later use
        beta_trace.save_trace(out_directory=out_directory)
        sigma_trace.save_trace(out_directory=out_directory)
        V_trace.save_trace(out_directory=out_directory)

        return 0
```