Gibbs Sampler in Python
Project Interim Report

Tyler Olson Alex Zajichek

Due November 14, 2016

This project implements the Gibbs sampling algorithm for two Bayesian models: Gamma-Poisson hierarchical model, and the multi-parameter Normal model with conjugate priors. Both models will also be implemented in `R` for runtime comparison purposes, and OpenBUGS to validate and compare the results of our implementation. Shown below is the derivation of full-conditional distributions, the Python and R implementations, and the OpenBUGS code that will be run.

To do

In Model 1, the full conditional distribution of the α parameter is of a form that is not recognizable by any frequently used parametric class. With that said, we still need to figure out how we can generate random draws from this distribution. One such approach would be to use the *inverse transform sampling* technique, which requires the inverse CDF of the random variable to be evaluated at random uniform values in [0,1].

Model 1: Gamma-Poisson hierarchical model

$$\begin{aligned}
\alpha &\sim Exponential(\lambda_1) \\
\beta &\sim Exponential(\lambda_2) \\
\theta_i|\alpha,\beta &\sim Gamma(\alpha,\beta) \\
x_i|\theta_i,t_i,\alpha,\beta &\sim Poisson(\theta_i t_i)
\end{aligned}$$

Deriving full-conditional distributions

Step 1: Find joint posterior

If $\boldsymbol{\theta} = (\theta_1, \theta_2, ..., \theta_n)$ and $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, then

$$\begin{aligned}
p(\alpha,\beta,\boldsymbol{\theta}|\boldsymbol{x}) &\propto p(\alpha,\beta,\boldsymbol{\theta},\boldsymbol{x}) \\
&= p(\boldsymbol{x}|\alpha,\beta,\boldsymbol{\theta})p(\alpha,\beta,\boldsymbol{\theta}) \\
&= p(\boldsymbol{x}|\alpha,\beta,\boldsymbol{\theta})p(\boldsymbol{\theta}|\alpha,\beta)p(\alpha,\beta) \\
&= p(\boldsymbol{x}|\alpha,\beta,\boldsymbol{\theta})p(\boldsymbol{\theta}|\alpha,\beta)p(\alpha)p(\beta) \\
&= \prod_{i=1}^{n} \frac{e^{-\theta_i t_i}(\theta_i t_i)^{x_i}}{x_i} \times \prod_{i=1}^{n} \frac{\beta^\alpha}{\Gamma(\alpha)} \theta_i^{\alpha-1} e^{-\beta\theta_i} \times \lambda_1 e^{-\lambda_1 \alpha} \lambda_2 e^{-\lambda_2 \beta} \\
&= \frac{\beta^{n\alpha}}{\Gamma(\alpha)^n} \lambda_1 e^{-\lambda_1\alpha} \lambda_2 e^{-\lambda_2\beta} \times \prod_{i=1}^{n} \frac{e^{-\theta_i(t_i+\beta)} \theta^{x_i+\alpha-1} t_i^{x_i}}{x_i}
\end{aligned}$$

Step 2-3: Pull out terms containing quantity of interest, find the product

$$p(\alpha|\beta,\boldsymbol{\theta},\boldsymbol{x}) \propto \frac{\beta^{n\alpha}}{\Gamma(\alpha)^n} e^{-\lambda_1\alpha} \prod_{i=1}^{n} \theta_i^{x_i+\alpha-1}$$
$$\propto \frac{\beta^{n\alpha}}{\Gamma(\alpha)^n} e^{-\lambda_1\alpha} \prod_{i=1}^{n} \theta_i^{\alpha}$$

$$p(\beta|\alpha,\boldsymbol{\theta},\boldsymbol{x}) \propto \beta^{n\alpha} e^{-\lambda_2\beta} \prod_{i=1}^{n} e^{-\theta_i t_i - \theta_i\beta}$$
$$\propto \beta^{n\alpha} e^{-\lambda_2\beta} \prod_{i=1}^{n} e^{-\theta_i\beta}$$
$$= \beta^{n\alpha} e^{-\lambda_2\beta} e^{-\sum_{i=1}^{n}\theta_i\beta}$$
$$= \beta^{n\alpha} e^{-\beta(\lambda_2+\sum_{i=1}^{n}\theta_i)}$$
$$\beta|\alpha,\boldsymbol{\theta},\boldsymbol{x} \sim Gamma(n\alpha+1, \lambda_2+\sum_{i=1}^{n}\theta_i)$$

$$p(\theta_i|\boldsymbol{\theta}_{-i},\alpha,\beta,\boldsymbol{x}) \propto e^{-\theta_i(t_i+\beta)}\theta_i^{x_i+\alpha-1}$$
$$\theta_i|\boldsymbol{\theta}_{-i},\alpha,\beta,\boldsymbol{x} \sim Gamma(x_i+\alpha, t_i+\beta)$$

Python code

```
import numpy
from scipy.stats import gamma
from array import *
def pump(x, t, alpha_init, beta_init, iteration, l_1, l_2):
    n = len(x)
    alpha = numpy.zeros(iteration)
    cur_alpha = alpha_init
    beta = numpy.zeros(iteration)
    cur_beta = beta_init
    theta = numpy.zeros((n,iteration))
    cur_theta = numpy.zeros(n)
    for j in range(n):
        cur_theta[j] = gamma.rvs(a=x[j]+cur_alpha, scale = 1/(t[j]+cur_beta))
    for i in range(iteration):
        cur_beta = gamma.rvs(a=n*cur_alpha+1,scale=1/(l_2 + sum(cur_theta)))
        beta[i] = cur_beta
        cur_alpha = ??????????
        for j in range(n):
            cur_theta[j] = gamma.rvs(a=x[j]+cur_alpha, scale = 1/(t[j]+cur_beta))
            theta[j,i] = cur_theta[j]
```

R code

```
x = c(5, 1, 5, 14, 3, 19, 1, 1, 4, 22)
```

```
t = c(94.3, 15.7, 62.9, 126, 5.24, 31.4,
      1.05, 1.05, 2.1, 10.5)
alpha_init = 10
beta_init = 10
l_1 = 1
l_2 = 1
pump = function(x,t,alpha_init,beta_init,iteration,l_1,l_2) {
  n = length(x)
  alpha = array(NA,iteration)
  cur_alpha = alpha_init
  beta = array(NA,iteration)
  cur_beta = beta_init
  theta = matrix(NA,ncol=n,nrow=iteration)
  cur_theta = rgamma(n,shape = x+cur_alpha,scale=1/(t+cur_beta))
  for(i in 1:iteration) {
    cur_beta = rgamma(1,shape = n*cur_alpha + 1, scale = 1/(l_2 + sum(cur_theta)))
    beta[i] = cur_beta
    cur_alpha = ??????????
    cur_theta = rgamma(n,shape = x+cur_alpha,scale=1/(t+cur_beta))
    theta[i,] = cur_theta
    }
}
```

OpenBUGS code

```
model {
for (i in 1:N) {
theta[i] ~ dgamma(alpha, beta)
lambda[i] <- theta[i] * t[i]
x[i] ~ dpois(lambda[i])
}
alpha ~ dexp(1)
beta ~ dexp(1)
}

list(t = c(94.3, 15.7, 62.9, 126, 5.24, 31.4, 1.05, 1.05, 2.1, 10.5),
x = c(5, 1, 5, 14, 3, 19, 1, 1, 4, 22), N = 10)

list(alpha = 10, beta = 10)
```

Model 2: Normal multi-parameter model with conjugate prior

$$(\mu, \sigma^2) \sim IG(\alpha, \beta) \times N(\mu_0, \frac{\sigma^2}{\kappa})$$
$$\boldsymbol{y}|\mu, \sigma^2 \sim N(\mu, \sigma^2)$$

Deriving full-conditional distributions

Step 1: Find joint posterior

$$\begin{aligned} p(\mu, \sigma^2|\boldsymbol{y}) &\propto p(\mu, \sigma^2, \boldsymbol{y}) \\ &= p(\bar{y}|\mu, \sigma^2)p(\mu|\sigma^2)p(\sigma^2) \\ &= \frac{\sqrt{n}}{\sqrt{2\pi}\sigma} e^{\frac{-n}{2\sigma^2}(\bar{y}-\mu)^2} \frac{\sqrt{\kappa}}{\sqrt{2\pi}\sigma} e^{\frac{-\kappa}{2\sigma^2}(\mu-\mu_0)^2} \frac{\beta^\alpha}{\Gamma(\alpha)} \frac{1}{(\sigma^2)^{\alpha+1}} e^{\frac{-\beta}{\sigma^2}} \end{aligned}$$

Step 2-3: Pull out terms containing quantity of interest, find the product

$$\begin{aligned} p(\mu|\sigma^2, \boldsymbol{y}) &\propto e^{\frac{-n}{2\sigma^2}(\bar{y}-\mu)^2 + \frac{-\kappa}{2\sigma^2}(\mu-\mu_0)^2} \\ &= e^{\frac{-1}{2\sigma^2}(n(\bar{y}-\mu)^2 + \kappa(\mu-\mu_0)^2)} \\ &= e^{\frac{-1}{2\sigma^2}(n(\bar{y}^2 - 2\bar{y}\mu + \mu^2) + \kappa(\mu^2 - 2\mu\mu_0 + \mu_0^2))} \\ &\propto e^{\frac{-(n+\kappa)}{2\sigma^2}(\mu^2 - 2\mu\frac{n\bar{y}+\kappa\mu_0}{n+\kappa})} \\ &\propto e^{\frac{-(n+\kappa)}{2\sigma^2}(\mu^2 - 2\mu\frac{n\bar{y}+\kappa\mu_0}{n+\kappa} + (\frac{n\bar{y}+\kappa\mu_0}{n+\kappa})^2)} \\ \mu|\sigma^2, \boldsymbol{y} &\sim N\left(\frac{n\bar{y}+\kappa\mu_0}{n+\kappa}, \frac{\sigma^2}{n+\kappa}\right) \end{aligned}$$

$$\begin{aligned} p(\sigma^2|\mu, \boldsymbol{y}) &\propto \frac{\sqrt{n}}{\sqrt{2\pi}\sigma} \frac{\sqrt{\kappa}}{\sqrt{2\pi}\sigma} \frac{\beta^\alpha}{\Gamma(\alpha)} \frac{1}{(\sigma^2)^{\alpha+1}} e^{\frac{-n}{2\sigma^2}(\bar{y}-\mu)^2 + \frac{-\kappa}{2\sigma^2}(\mu-\mu_0)^2 + \frac{-\beta}{\sigma^2}} \\ &\propto \frac{1}{(\sigma^2)^{\alpha+2}} e^{\frac{-1}{\sigma^2}(\frac{n}{2}(\bar{y}-\mu)^2 + \frac{\kappa}{2}(\mu-\mu_0)^2 + \beta)} \\ \sigma^2|\mu, \boldsymbol{y} &\sim IG\left(\alpha+1, \frac{n}{2}(\bar{y}-\mu)^2 + \frac{\kappa}{2}(\mu-\mu_0)^2 + \beta\right) \end{aligned}$$

Python code

```
import numpy as np
from random import gauss
from scipy.stats import invgamma
import math


def candy(data, iteration, alpha, beta, mu0, kappa, mu_init):
    n = len(data)
    ybar = np.mean(data)
    s2 = np.var(data)
    mu = mu_init
    # sigma2 = sigma2_init
    samples = []
    np.random.seed(17)
    for i in range(iteration):
        alphaShape = alpha + (n / 2) + 0.5
        betaScale = beta + ((n - 1) * s2 / 2) + (kappa * n * (ybar - mu0)**2 / (2 * (kappa + n))) + \
                    ((mu - (kappa * mu0 + n * ybar))**2 / (2 / (kappa + n)))
        sigma2 = invgamma.rvs(a=alphaShape, scale=1/betaScale, size=1)
```

```
        mean = (kappa * mu0 + n * ybar) / (kappa + n)
        sd = math.sqrt(sigma2 / (kappa + n))
        mu = gauss(mu=mean, sigma=sd)
        samples.append([sigma2, mu])
    return samples


def main():
    data = [46, 58, 40, 47, 47, 53, 43, 48, 50, 55, 49, 50, 52, 56, 49, 54, 51, 50, 52, 50]
    results = candy(data, 10000, 38, 444, 51, 10, 100)
    print(np.mean([item[0] for item in results]))
    print(np.mean([item[1] for item in results]))
main()
```

R code

```
candy <- function(data, iteration, alpha, beta, mu0, kappa, mu_init) {
  n <- length(data)
  ybar <- mean(data)
  s2 <- var(data)
  mu <- mu_init
  samplesMu <- array(NA, iteration)
  samplesSigma2 <- array(NA, iteration)
  set.seed(17)
  for (i in 1:iteration) {
    alphaShape <- alpha + (n / 2) + 0.5
    betaScale <- beta + ((n - 1) * s2 / 2) +
                 (kappa * n * (ybar - mu0)^2 / (2 * (kappa + n))) +
                 ((mu - (kappa * mu0 + n * ybar))^2 / (2 / (kappa + n)))
    sigma2 <- 1 / rgamma(1, alphaShape, 1/betaScale)
    mean <- (kappa * mu0 + n * ybar) / (kappa + n)
    sd <- sqrt(sigma2 / (kappa + n))
    mu <- rnorm(1, mean, sd)
    samplesMu[i] <- mu
    samplesSigma2[i] <- sigma2
  }
  results <- matrix(c(samplesMu, samplesSigma2), nrow = iteration)
  return(results)
}

data <- c(46, 58, 40, 47, 47, 53, 43, 48, 50, 55, 49, 50, 52, 56, 49, 54, 51, 50, 52, 50)
results <- candy(data, 10000, 38, 444, 51, 10, 100)
mean(results[,1])
mean(results[,2])
```

OpenBUGS code

```
model {
tausq ~ dgamma(alpha, beta)
sigma2 <- 1 / tausq
mu ~ dnorm(mu0, precision)
precision <- kappa / tausq
ybar ~ dnorm(mu, precision2)
precision2 <- n / sigma2
}
```

```
list(ybar = 50, n = 20, alpha = 38, beta=444, mu0 = 51, kappa = 10)

list(mu = 100, tausq = 0.05)
```